Doctoral theses at NTNU, 2023:381

Morten Rotvold Solberg

Security for Electronic Voting Systems

Doctoral thesis

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of
Philosophiae Doctor
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Morten Rotvold Solberg

Security for Electronic Voting Systems

Thesis for the Degree of Philosophiae Doctor

Trondheim, November 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

ISBN 978-82-326-7462-6 (printed ver.)
ISBN 978-82-326-7461-9 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

Doctoral theses at NTNU, 2023:381

Printed by NTNU Grafisk senter

Acknowledgments

This thesis is submitted in partial fulfillment of the requirements for the degree of *Philosophiae Doctor* (Ph.D.) at the Norwegian University of Science and Technology (NTNU). The thesis consists of four papers written in the period from the fall of 2019 to the spring of 2023. It turns out that doing a Ph.D. is a lot of work, and this would never have been possible without the help and support of quite a lot of people.

First and foremost, I wish to thank my supervisor Kristian Gjøsteen for his patience in answering all my questions, for guiding me towards interesting and relevant research problems, for always keeping an open door and for valuable advice regarding matters both in- and outside of academia.

I also wish to thank my co-supervisor Colin Boyd for his support and pleasant conversations regarding both research and life in general.

Thanks to my co-authors Kristian Gjøsteen, Thomas Haines, François Dupressoir, Peter Rønne, Cătălin Drăgan, Peter Y.A. Ryan and Ehsan Estaji for interesting discussions and for valuable contributions to the papers in this thesis. A particular thanks goes to François for his patience in answering all my silly (and hopefully a few not so silly) questions about EasyCrypt, to Thomas for taking the time to answer all my questions about everything and to Peter Rønne for stepping in and presenting our paper at CSF in August 2022 on short notice, when I unfortunately had to travel home early due to personal reasons.

A good working environment is of crucial importance for motivating one self to go to the office and do an honest day's work. I wish to thank my office mates Endre, Magnus and Ole Martin for the laughs, the serious and not-so-serious conversations, the coffee breaks and for making the office a safe and enjoyable space. I am happy to say that I have made good friends in you. A special thanks to Endre for taking the time to proofread my thesis.

Thanks to Tjerand for taking on the role as my mentor during my early days as a Ph.D. candidate, for always having the answer to any cryptographic question, no matter how deep or complex, and for proofreading my thesis.

Thanks to the rest of the algebra group at the department and the NaCl research group for pleasant conversations, interesting talks and seminars, the hikes and the cakes. There are too many names to mention everyone, but you know who you are and I feel very lucky to have gotten to know you.

Thanks to Morten Nome for always having an open door and a free spot in the couch, giving me a place to vent, listen to crazy stories, eat rakfisk and for including me in

developing and teaching in the department's "oppfriskningskurs". Thanks to Aslak and Gereon for including me as a lecturer in the TMA4110/4115 team and giving me valuable experiences as a university teacher.

A huge thanks goes to my family and friends for always showing interest and supporting my way in life, for always listening, for giving me a place to disconnect and for teaching me how to deal with any matter in the real world, away from theoretical cryptography and formal verification. Thanks for the game nights, movie nights, runs, hikes, laughs, D&D sessions, fishing trips, dinners and everything else. Thanks to Siri for taking the time and effort to organize "Sverresborg Utegym" during the pandemic and thus helping to keep my mental and physical health somewhat from deteriorating. A special thanks to Kjetil for, in addition to everything else, taking the time to proofread my thesis.

Last but by no means least, thanks to Solveig for proofreading my thesis, for your eternal patience and unconditional love and support, for always listening and making me laugh when the skies are gray, for waiting for me when skiing down hills, for hiking long distances with me, for keeping my head above water and keeping me sane, and for always being there.

Morten Rotvold Solberg
Trondheim, June 2023

Introduction

An election is a process where a group of individuals (the voters) convene and together make a decision. Each voter somehow casts a ballot and the ballots are somehow counted to arrive at the decision (or the result of the election). Traditional voting systems, such as pure paper-based voting, where voters go to a polling station and fill in their choices on a paper ballot, have certain limitations. First of all, they rely heavily on trust in the election officials, and voters have limited means of verifying that their ballots were counted. Additionally, there are issues regarding accessibility, counting errors and timeliness.

These issues can be addressed by making use of voting systems benefiting from electronic systems support and techniques from cryptography. Broadly speaking, there are two classes of electronic voting systems. In the first class, voters still go to a polling station but fill in their ballots on a machine rather than on paper (or they use a combination of paper ballots and voting machines to facilitate the tally). In the second class, we find fully electronic voting systems, or remote electronic voting systems, where voters typically fill in their ballots on a website or in a dedicated smartphone application and submit the ballot through the Internet to a voting server maintained by the election officials. In this thesis we focus on the latter class. As such, any reference to "electronic voting" should be taken to mean "remote (internet) voting".

Although electronic voting systems have many attractive properties, there are certain challenges. The addition of cryptographic mechanisms to the voting systems might make the systems more complicated, and voting systems must be designed so that they are easy to use correctly by the voters. Furthermore, they must be easy to implement by programmers to avoid unintentional programming errors that might leak information about the votes or make it possible to tamper with the result. Finally, electronic voting systems are susceptible to a large range of *attacks*, and care must be taken to ensure that the systems we use are *secure.*

Voting systems must satisfy a large range of security properties. First and foremost, the outcome of the election must correctly reflect the opinions of the voters. This is usually called *integrity.* A voting system must provide *ballot privacy* [2, 11], so that voters are able to freely express their opinions. It is often desirable that the privacy of the votes is not only ensured during the election, but for the foreseeable future. This property is often referred to as *everlasting privacy* [17]. To avoid coercion and vote-buying, it is desirable that voting systems provide voters with some strategy for *coercion resistance* [30]. Another desirable property of electronic voting systems, that is difficult to achieve in traditional

voting systems, is *verifiability* [8, 31]. Verifiable voting systems allow voters to verify that their ballot was included in the tally, and that the outcome of the election correctly reflects the submitted ballots. An interesting question arises if a voter files a complaint, for example claiming that their ballot was not included in the tally: how should one proceed then? Often, it is not possible to restart the election and have all the voters submit their ballots again. It might also be the case that the complaining voter is dishonest and has no reason to complain, but does so only to raise questions about the integrity of the election. Furthermore, the complaining voter might be honest, but made a mistake in the verification process, making her believe that her vote was dropped even though that was not the case. This challenge can at least partially be solved by voting systems providing *accountability* [24]. Then, if anything goes wrong, voters are able to produce evidence that something indeed went wrong, which in turn will cause the misbehaving party to be blamed for their misbehaviour.

Ensuring that the voting systems we use meet the required security properties is a non-trivial task. In this thesis, the overall goal is to formally analyse existing electronic voting systems and the level of security they provide. We identify three important topics for doing so:

Topic A: *Security definitions.* Understanding the level of security provided by a voting system is not possible without a formal definition of what security means, what capabilities an attacker is assumed to have, and what an attacker would like to achieve. We aim to provide new security definitions for voting-related security properties and improve upon existing definitions.

Topic B: *Security proofs.* To increase the assurance that the voting systems we use meet the required security properties, it is necessary to provide security proofs. We aim to develop new security proofs for existing voting systems (and certain primitives that are used in voting systems) that lack such proofs.

Topic C: *Machine-checked proofs.* The assurance that a voting system is secure is no greater than the confidence that the security proofs are correct. We aim to increase the confidence in our security proofs, by using interactive theorem provers to verify that our proofs are correct.

The three topics are in some sense different, but they are closely related. To properly analyse the security provided by a protocol and write a security proof, it is necessary to first have a proper security definition, and it is impossible to machine-check a proof without first having a proof. Defining security is an important contribution on its own, but a security definition on its own still does not mean much without using it to analyse the security of some construction. Security proofs are important contributions and may well exist without a machine-checked counterpart, but machine-checking a proof is important to increase the assurance that the proof is correct.

This thesis consists of four individual research papers that all relate to two or three of the topics listed above. An illustration of how each paper relates to the three topics can be found in Figure 1.

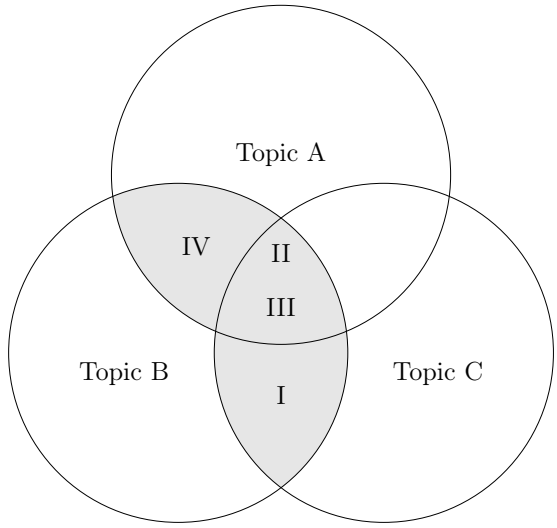


Figure 1: Relating the individual papers to the various topics in the thesis. Topic A: Security definitions. Topic B: Security proofs. Topic C: Machine-checked proofs. Roman numbers refer to the individual papers in the thesis.

Paper I does not introduce any new security definitions, but provides machine-checked security proofs for existing constructions. Papers II–IV all present new security definitions for various properties, along with security proofs for existing voting systems. Papers II and III additionally include machine-checked proofs.

In the remainder of the introduction, we will introduce techniques and tools for building secure electronic voting systems and for increasing the assurance that the systems are indeed secure, before discussing a few important security properties. We introduce and explain the contributions of each paper as we go.

Cryptographic Building Blocks

Separately, the aforementioned security properties might be easy to achieve. For example, verifiability can be trivially achieved by publishing signed ballots for each voter. However, this would completely destroy privacy. Privacy, on the other hand, can be ensured by traditional paper-based voting, but as mentioned, this gives voters no means of verifying that their ballot was counted. Achieving all security properties at once is a much greater challenge, but it is possible by using techniques from cryptography. In this section, we describe some cryptographic primitives that are often used in electronic voting systems, and briefly explain their roles in building secure electronic voting systems.

Digital Signatures. Digital signatures are (among other things) used to authenticate voters to ensure that only eligible voters vote, and that eligible voters only vote once, to avoid issues like ballot stuffing. A digital signature scheme is a triple of algorithms $\mathsf{S} = (\mathsf{Kgen_s}, \mathsf{Sign}, \mathsf{Verify})$. The key generation algorithm $\mathsf{Kgen_s}$ takes no input and outputs a pair (sk, vk) consisting of a secret signing key sk and a public verification key vk. The signing algorithm Sign takes as input a signing key sk and a message m and outputs a signature σ. The verification algorithm Verify takes as input a verification key vk, a message m and a signature σ and outputs either 0 or 1. We require that a signature scheme is *correct*, i.e. we require that for all key pairs (sk, vk) output by $\mathsf{Kgen_s}$ and for any message m, we have $\mathsf{Verify}(vk, m, \mathsf{Sign}(sk, m)) = 1$. Informally, a digital signature scheme is *secure* if it is difficult to create forgeries, even after seeing valid signatures on several messages.

Public Key Encryption. To help protect privacy, ballots are usually encrypted using a public key encryption system (PKE). A PKE is a triple of algorithms $\mathsf{E} = (\mathsf{Kgen_e}, \mathsf{Enc}, \mathsf{Dec})$. The key generation algorithm $\mathsf{Kgen_e}$ takes no input and outputs a pair (ek, dk) of encryption and decryption keys. The encryption algorithm Enc takes as input an encryption key ek and a plaintext m and outputs a ciphertext c. The decryption algorithm Dec takes as input a decryption key dk and a ciphertext c and outputs a plaintext m. As for digital signatures, we require that a PKE is correct, i.e. we require that for any pair of keys (ek, dk) output by $\mathsf{Kgen_e}$ and any plaintext m, we have $\mathsf{Dec}(dk, \mathsf{Enc}(ek, m)) = m$. Informally, a PKE is *secure* if it is difficult to learn anything about the underlying plaintext from the ciphertext, sometimes even when seeing decryptions of several other ciphertexts.

Some public key encryption systems (such as ElGamal [14]) enjoy a homomorphic property, so that two ciphertexts can be combined in such a way that the resulting ciphertext decrypts to either the sum or product of the original plaintexts. This property can be exploited to *re-encrypt* ciphertexts. In other words, from one ciphertext, one can compute a new ciphertext that decrypts to the same message as the original ciphertext. This is useful for constructing verifiable shuffles, which we discuss further down.

It is often desirable to *distribute* the decryption key among several parties, so that some subset of the election officials is needed to decrypt the ballots. This way, privacy is protected against curious election officials (unless sufficiently many officials collude). This also helps towards increasing the *robustness* of the election, as it would still be possible to compute a result even if some election officials were to become unavailable (except for cases where all officials are needed to decrypt the ballots).

While distributed decryption is much used in electronic voting systems, there are interesting systems that do not make use of this technique. A notable example is sElect [23], which we discuss in Paper III. Indeed, sElect only uses very basic cryptographic primitives: namely plain public key encryption and digital signatures.

Commitment Schemes. A commitment scheme allows a sender to *commit* to some value, and reveal the value at a later point in such a way that no one learns the committed value before the sender chooses to reveal it (this is called *hiding*), and the sender is unable

to reveal a different value than the one he committed to (this is called *binding*).

Formally, a commitment scheme is a triple of algorithms $\mathsf{C} = (\mathsf{Kgen_c}, \mathsf{Commit}, \mathsf{Open})$. The key generation algorithm $\mathsf{Kgen_c}$ takes no input and returns a commitment key ck. The commitment algorithm Commit takes as input a commitment key ck and a message m and returns a commitment ct and an opening op. The opening algorithm Open takes as input a commitment key ck, a message m, a commitment ct and an opening op and returns either 0 or 1. As for public key encryption and digital signature schemes we put a correctness requirement on the commitment scheme, so that for any commitment key ck output by $\mathsf{Kgen_c}$ and for any message m, we have $\mathsf{Open}(ck, m, \mathsf{Commit}(ck, m)) = 1$.

Commitment schemes have various applications in electronic voting. Examples include election officials committing to their share of a shared secret key or voters committing to an encryption of their submitted ballot. Commitment schemes can also be used as building blocks for more complex systems. For example, Cuvelier *et al.* [12] use commitments to construct a cryptographic primitive they call *commitment-consistent encryption* where commitments are derived from encrypted ballots to create a *perfectly private audit trail*, which in turn is used to achieve *everlasting privacy*. We discuss this construction further in Paper I.

Zero-Knowledge Proofs. Zero-knowledge proofs [15] are often used in electronic voting systems to ensure that certain computations are performed correctly. Examples include voters proving that they have encrypted a valid ballot and election officials proving that they have correctly decrypted the submitted ballots (e.g. without removing or tampering with any ballots).

A zero-knowledge proof is a protocol between two parties called the *prover* and the *verifier*. The prover is tasked with proving statements about some secret data, without leaking any information about the data. *Zero-knowledge* refers to the fact that the verifier only learns that the statement is true, but learns nothing about the secret data. A zero-knowledge proof must satisfy two additional properties: completeness and soundness. We say that a zero-knowledge proof is *complete* if the verifier always accepts a proof for a true statement. A zero-knowledge proof is *sound* if the prover is unable to convince the verifier that a false statement is true.

Verifiable Shuffles. One particular kind of zero-knowledge proof is a *verifiable shuffle*. In many electronic voting protocols, encrypted ballots are *shuffled* to destroy the link between the ballots and the voters who cast them. This is done to achieve privacy. The shuffle is performed several times, by a series of *mix servers*. The first mix server receives a list of encrypted ballots, re-encrypts each ciphertext and permutes the order of the ciphertexts in the list. The output of the mix server is a new list of ciphertexts that decrypt to the same ballots as the ciphertexts in the input list. The resulting list is sent to the next mix server, which then shuffles the list it receives, sends the resulting list to the next mix server, and so on.

To ensure that this is done honestly, each mix server must provide a zero-knowledge proof of correct computation, proving that he did not tamper with the ballots in any way

(e.g. added new ballots or removed any ballots). The addition of the zero-knowledge proofs is what makes the shuffles *verifiable*.

Various verifiable shuffle protocols exist in the literature. Cuvelier *et al.* [12] propose using the Terelius-Wikström shuffle [33, 32] to shuffle so-called PPATC ciphertexts used in a commitment-consistent encryption scheme to provide a perfectly private audit trail (and hence everlasting privacy). Haenni *et al.* [16] present a shuffle protocol for shuffling single ElGamal ciphertexts using an optimised variant of the Terelius-Wikström shuffle. However, it has not previously been shown that the Terelius-Wikström mixnet is suitable for shuffling PPATC ciphertexts, leading us to our first paper.

Paper I *Efficient Mixing of Arbitrary Ballots with Everlasting Privacy: How to Verifiably Mix the PPATC Scheme.* We build upon the work by Haenni *et al.* [16] and prove that PPATC ciphertexts can be shuffled using the optimised Terelius-Wikström shuffle. We prove that the shuffle protocol is complete, sound and zero-knowledge and verify our proofs in the Coq theorem prover [3].

Provable Security and Proof Assistants

Traditionally, cryptographic systems have been analysed by simply trying to break them. If a certain amount of time passed without an attack being found, the system in question was considered secure. This approach is clearly not satisfactory. Even if an attack is not found, it does not mean that an attack does not exist. One could also imagine that someone finds an attack, but does not tell anyone.

An approach for increasing assurance that the cryptographic protocols we use are indeed secure, is the (now standard) approach of *provable security*. The security of a cryptographic protocol is often related to some underlying mathematical problem through a *reduction*. The idea is to prove that if we are able to efficiently break the security of the cryptographic system, then we are also able to efficiently solve the underlying problem. As the underlying problem is assumed to be difficult to solve (efficiently), this leads to a contradiction. The increased assurance that the protocol is secure comes from the fact that the underlying mathematical problem typically has been subjected to a large amount of analysis, and that the hardness of solving it is thus well understood.

However, before writing a security proof, we need a precise definition of what security should mean: what the capabilities of the adversary should be and what he would like to achieve.

Security Games

One way of defining security is through a *security game*, where a benign entity called a *challenger* plays against an *adversary*. A security game specifies what we assume the adversary is able to do by specifying a set of moves as well as conditions on when and how many times the adversary is allowed to perform each move. Secondly, the game specifies the adversary's *goal*, i.e. what the adversary would like to achieve. The adversary's goal is typically defined as an event E in some probability space. We say that an adversary

wins the game if he is able to achieve his goal. A system is considered secure if we can prove that no (efficient) adversary can win the specified game with more than negligible probability.

To ease the writing and reading of game-based security proofs, one often employs the technique of *game-hopping* [28]. The idea is to construct a sequence of games $G_0, \ldots, G_n$, where G_0 is the original attack game. If we let E_i be the event that the adversary wins the game G_i, the idea is to construct the games such that $\Pr[E_i]$ is negligibly close to $\Pr[E_{i+1}]$ for all $i = 0, \ldots, n-1$ and that $\Pr[E_n]$ is negligibly close to some "target probability", e.g. 0, 1 or 1/2. It then follows through a standard hybrid argument that $\Pr[E_0]$ is also negligibly close to the target probability. The changes from G_i to G_{i+1} should be as small as possible, to make the analysis as easy as possible.

We note that other approaches exist for defining security, such as simulation-based security [25] and Dolev-Yao's symbolic model [13]. In the papers included in this thesis, however, we focus on the game-based approach when developing new security definitions.

Interactive Theorem Provers

Even with techniques such as game-hopping, cryptographic security proofs tend to be complicated and error-prone. They are difficult to write, and they are difficult to audit, and the assurance that a protocol meets the claimed security properties is no higher than our confidence that the security proof is correct. Over the past decades, several *interactive theorem provers* (or *proof assistants*) have been developed to increase the assurance that a security proof is correct.

An interactive theorem prover is a piece of software designed to aid in developing formal proofs. A wide variety of proof assistants exist. Some are designed to help develop and verify mathematical proofs in general, while others are more specialized towards one single area, such as cryptographic security proofs.

One particular proof assistant is EASYCRYPT, designed for verifying cryptographic security proofs using games [1]. EASYCRYPT is largely based on Hoare logic [18], developed by Tony Hoare to reason about the properties of computer programs. A core component of the Hoare logic is the so-called *Hoare triple* $P\{Q\}R$, where P is a *precondition*, R is a *postcondition* and Q is some computer program (for example a security game). The pre- and postconditions are assertions about the variables of the program, and the Hoare triple can informally be interpreted as "if the precondition is true before execution of the program, then the postcondition will be true after the program is completed". In EASYCRYPT, the Hoare triple is expressed as $[Q : P \implies R]$.

Another component of EASYCRYPT is a *probabilistic Hoare logic*, called pHL, used to reason about the probability of some event happening in a security game. In other words, the pHL logic is used to reason about judgments of the form $[Q : P \implies R] \circ p$, where $\circ$ is either $<, >, \leq, \geq$ or $=$ and p is a probability expression.

A central aspect of game-based security proofs is to *relate* two different games. EASYCRYPT supports a *relational* Hoare logic for probabilistic games, called pRHL. This logic reasons about judgments of the form $[Q_1 \sim Q_2 : P \implies R]$, where Q_1 and Q_2 are probabilistic programs (games) and the pre- and postconditions P and R are logical statements

relating the variables of the two games.

The pRHL logic does not reason about probabilities directly, but it is possible to derive certain probability claims from valid pRHL judgments. For example, from the judgment $[Q_1 \sim Q_2 : P \implies E_1 \to E_2]$, where E_1 and E_2 are events occurring in the programs Q_1 and Q_2, respectively, one can derive that $\Pr[E_1] \leq \Pr[E_2]$. From the judgment $[Q_1 \sim Q_2 : P \implies E_1 \leftrightarrow E_2]$ one can derive that $\Pr[E_1] = \Pr[E_2]$.[1]

EasyCrypt also implements a higher-order classical logic for proving mathematical statements. The classical logic also enables the use of specialised software (called *SMT solvers*) that can be used to automatically prove simple mathematical facts.

EasyCrypt offers a module system which is used to model cryptographic games and cryptographic primitives. Modules consist of procedures written in a simple imperative language and may be parameterised by other (abstract) modules. The module system allows for writing modular proofs and is highly useful for structuring large proofs consisting of several games and involving multiple primitives or sub-protocols.

Since its birth around a decade ago, EasyCrypt has seen extensive use, and has been used to verify the security of both basic cryptographic primitives and more complex protocols. Lately, EasyCrypt has also been used to verify the security of cryptographic voting protocols (see e.g. [6, 7]). We adopt this approach in our work and machine-check our proofs in three of the papers included in this thesis (Papers I–III). Most of the proofs we machine-check in our work are checked with EasyCrypt. The exceptions are the proofs for the shuffle protocols we present in Paper I, which are checked using the Coq theorem prover [3].

Security Properties for Voting Systems

In this section, we elaborate on a few central security properties for electronic voting protocols. We briefly describe past efforts at defining security and explain the contributions of our thesis work as we go. The security properties we discuss are ballot privacy, accountability, coercion resistance and coercion mitigation.

Ballot Privacy

Numerous security definitions for ballot privacy exist in the literature. Bernhard *et al.* [2] give a thorough analysis of existing game-based definitions up to 2015 and conclude that none of them are satisfactory. Some definitions are too weak (meaning that there are real attacks not captured by the definitions), some are too strong (so that no voting system with any kind of verifiability can be proven secure under the definition) and some are too limited (meaning that they capture only a narrow class of voting systems).

Bernhard *et al.* [2] address the issue by presenting a new definition they call BPRIV. The BPRIV definition captures the idea that no information about the votes should be leaked, besides what can be inferred from the election result. In BPRIV, the adversary is tasked with distinguishing between two worlds, called the real world and the fake world.

[1] Here, we use $\to$ and $\leftrightarrow$ to denote logical implication and equivalence, respectively.

In the real world, the adversary gets to see a ballot box containing real ballots, a real result and a proof of correct tally. In the fake world, the adversary gets to see a fake ballot box, but he still sees the real result and a simulated proof of correct tally. However, as pointed out by Cortier *et al.* [11], the BPRIV definition makes the fairly strong assumption that the voting server (and hence the ballot box) is honest. As such, it is assumed that once the ballots are placed in the ballot box, they are not dropped or tampered with in any way.

Cortier *et al.* [11] get rid of this trust assumption by introducing a new ballot privacy definition they call mb-BPRIV, in which the adversary is allowed to tamper with the ballots in the ballot box. The main idea is similar to BPRIV: the adversary is tasked with distinguishing between two worlds. As the adversary is allowed to tamper with the ballot box and gets to see the real result also when he is in the fake world, care is needed to ensure that distinguishing between the two worlds is not trivial. In particular, it is necessary to decide which of the real ballots to include in the tally, when we are in the fake world. Cortier *et al.* address this by introducing what they call a *recovery algorithm*, which detects how the adversary has tampered with the ballots in the fake ballot box. To avoid making distinguishing trivial, the same "tampering" is performed to the real ballot box, before tallying the ballots.

The mb-BPRIV definition has one drawback, namely that the adversary gets to see the tally only if voters who verify that their ballot is included, are happy. In other words, in the mb-BPRIV definition, tallying must occur after the voters have verified, meaning that this definition is only applicable to voting systems where verification happens before the tally is computed. However, there are interesting voting systems such as Selene [27], where verification must happen after the outcome of the election is made public as a countermeasure against coercion. Systems like Selene are not accommodated by the mb-BPRIV definition, indicating that there is still a need for new ballot privacy definitions.

Paper II *Machine-Checked Proofs of Privacy Against Malicious Boards for Selene & Co.*
We build upon the mb-BPRIV definition and present du-mb-BPRIV, a ballot privacy definition where the adversary is allowed to tamper with the ballot box, which is also applicable to voting systems where verification either can or must happen after the tally has been computed. We model our security definition in EasyCrypt and prove that Selene is ballot-private. Our definition is also applicable to voting systems where verification can happen both before and after the tally has been computed, such as Belenios [9]. We demonstrate the applicability by modelling Labelled-MiniVoting [6] and Belenios in our framework and prove that they are ballot-private. We verify all our security proofs using EasyCrypt. Furthermore, we model the mb-BPRIV definition in EasyCrypt and prove that Labelled-MiniVoting and Belenios satisfy this definition (also with proofs checked with EasyCrypt). Finally, we prove that the du-mb-BPRIV definition implies individual verifiability, and we present a few lessons we learned when modelling our definitions and proofs in EasyCrypt, which we believe to be useful for future efforts on modelling electronic voting systems and related security properties in EasyCrypt.

Accountability

Accountability captures the idea that if anything goes wrong during an election (e.g. if any ballots are dropped), voters should be able to produce evidence pinpointing which party misbehaved. Accountability is a property that is not only desirable for electronic voting protocols, but also for more general security protocols. Several accountability definitions in the symbolic model exist in the literature (see e.g. [4, 21, 22, 26]). Furthermore, Küsters *et al.* [23] put forward quantitative measures of accountability. To the best of our knowledge, no game-based definition of accountability has been proposed earlier. Furthermore, no definition of accountability seems to have been widely accepted by the cryptographic community and the analysis performed by Küsters *et al.* [23] contains a few errors that we point out in Paper III. This indicates that existing definitions are difficult to work with and that there is still a need for workable accountability definitions.

Paper III *Machine-Checked Proofs of Accountability: How to sElect who is to Blame.* We propose the first game-based definition of accountability for electronic voting protocols. As a case study, we apply our definition to the sElect voting system [23] and prove that sElect provides accountability. We model our definition in EasyCrypt, and verify that our proofs are correct. Furthermore, we uncover and address two shortcomings in the original analysis of sElect by Küsters *et al.* [23]. Finally, we prove (on paper only) that our definition implies the definition by Küsters *et al.* in the sense that any voting system that can be proven accountable under our definition is also accountable under their definition, and we prove that accountability implies individual verifiability.

Our definition adopts the idea from previous definitions that there are two aspects of accountability: fairness and completeness. Fairness means that a party following the protocol should not be blamed for anything and completeness means that if the goal of the protocol is not met, then someone will be held accountable (i.e. someone will be blamed). It follows from fairness that the blamed party actually misbehaved.

Coercion Resistance and Mitigation

Coercion resistant voting systems provide voters with a strategy for casting their vote freely even under observation by a coercer. The first formal definition and the first coercion resistant electronic voting system (JCJ) was proposed in 2002 by Juels *et al.* [20]. Since then, several coercion resistant voting systems have been proposed, e.g. Civitas [5], CHide [10] and Athena [29].

The typical strategy for achieving coercion resistance is to let voters *re-vote* at a time when the coercer is not present. This can be achieved for example by equipping the voters with two credentials, one true credential and one fake credential. If a voter is under coercion, she submits her ballot using her fake credential, and she can use her true credential to cast her intended ballot. In the tally phase, only ballots submitted with true credentials are counted.

However, this strategy has certain drawbacks. Re-voting often comes with complicated procedures for the voters or computationally expensive tallying. Furthermore, there are countries where re-voting is simply prohibited. Thus, different strategies are necessary to allow voters to vote freely when they are under coercion.

One interesting voting system that *mitigates* the coercion threat while also being verifiable and easy to use is Selene [27]. In Selene, voters are equipped with personal *tracking numbers*, which are published in the clear along with the votes. This gives the voters a simple way of verifying that their ballot was included: they simply look up their ballot on the bulletin board and check that it appears next to their tracker.

Intuitively, this approach increases the danger of coercion as a coercer can demand that a voter hands over their tracker, allowing the coercer to verify that the voter fulfilled his demands. However, in Selene, the voters do not gain knowledge of which tracker is theirs until after all the votes and trackers have been published. This gives the voters an opportunity to look up a tracker pointing to the coercer's desired ballot and send over that tracker.

Although Selene gives voters a way of convincingly lying about which ballot is theirs *after* the tally has been computed, they are not protected against a coercer who is present during the time of *ballot submission* (in Selene, re-voting is not possible). This is in contrast to e.g. JCJ, where voters can cast a ballot satisfying the coercer's demand while the coercer is present, and then cast a new ballot at a later time when the coercer is not present. Thus, Selene does not provide full coercion *resistance*, against a coercer who is present during ballot submission. However, as voters are able to convincingly lie about their trackers, Selene intuitively offers good protection against coercers present during the verification phase or coercers who demand that voters send over any data necessary to perform verification. We will refer to this property as *coercion mitigation*. Although the term coercion mitigation has been used in the literature when describing the security provided by Selene (see e.g. [19, 27, 34]), no formal definition of coercion mitigation seems to exist in the literature, and the coercion mitigation property of Selene has (to the best of our knowledge) not previously been formally analysed.

Paper IV *Coercion Mitigation for Voting Systems with Trackers: A Selene Case Study.* We present a framework for modelling verifiable voting systems based on tracking numbers and propose the first game-based definition of coercion mitigation. Our security experiment simultaneously captures coercion mitigation, ballot privacy and verifiability. We present a complete model of Selene in our framework and prove that Selene satisfies the aforementioned security properties. We end the paper with a discussion of how different variants of Selene fit into our framework.

Although the ballot privacy property of Selene has been formally verified earlier, we include this property in Paper IV as well for a more complete analysis of Selene. However, we define ballot privacy in a slightly different manner than in Paper II. The privacy definition in Paper II (and other definitions in the BPRIV style) defines privacy based on how much information is leaked from the casting and tallying processes. In Paper IV, we define privacy based on the question of which voter cast which particular ballot.

Privacy is captured through left-or-right challenge queries and the adversary is tasked with determining if the left or right ballots were tallied (as opposed to the BPRIV style, where one always computes the result from the left side ballots). Of course, distinguishing becomes trivial if the left and right ballots produce different outcomes, so we must require that the challenge queries taken together produce the same outcome on both sides.

References

[1] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. *EasyCrypt: A Tutorial*, pages 146–166. Springer International Publishing, Cham, 2014.

[2] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. SoK: A comprehensive analysis of game-based ballot privacy definitions. In *2015 IEEE Symposium on Security and Privacy*, pages 499–516. IEEE Computer Society Press, May 2015.

[3] Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

[4] Alessandro Bruni, Rosario Giustolisi, and Carsten Schürmann. Automated analysis of accountability. In Phong Q. Nguyen and Jianying Zhou, editors, *ISC 2017*, volume 10599 of *LNCS*, pages 417–434. Springer, Heidelberg, November 2017.

[5] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. In *2008 IEEE Symposium on Security and Privacy*, pages 354–368. IEEE Computer Society Press, May 2008.

[6] Véronique Cortier, Constantin Catalin Dragan, François Dupressoir, Benedikt Schmidt, Pierre-Yves Strub, and Bogdan Warinschi. Machine-checked proofs of privacy for electronic voting protocols. In *2017 IEEE Symposium on Security and Privacy*, pages 993–1008. IEEE Computer Society Press, May 2017.

[7] Véronique Cortier, Constantin Catalin Dragan, François Dupressoir, and Bogdan Warinschi. Machine-checked proofs for electronic voting: Privacy and verifiability for belenios. In Steve Chong and Stephanie Delaune, editors, *CSF 2018 Computer Security Foundations Symposium*, pages 298–312. IEEE Computer Society Press, 2018.

[8] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Mueller, and Tomasz Truderung. SoK: Verifiability notions for E-voting protocols. In *2016 IEEE Symposium on Security and Privacy*, pages 779–798. IEEE Computer Society Press, May 2016.

[9] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. *Belenios: A Simple Private and Verifiable Electronic Voting System*, pages 214–238. Springer International Publishing, Cham, 2019.

[10] Véronique Cortier, Pierrick Gaudry, and Quentin Yang. Is the JCJ voting system really coercion-resistant? Cryptology ePrint Archive, Report 2022/430, 2022. https://eprint.iacr.org/2022/430.

[11] Véronique Cortier, Joseph Lallemand, and Bogdan Warinschi. Fifty shades of ballot privacy: Privacy against a malicious board. In Limin Jia and Ralf Küsters, editors, *CSF 2020 Computer Security Foundations Symposium*, pages 17–32. IEEE Computer Society Press, 2020.

[12] Edouard Cuvelier, Olivier Pereira, and Thomas Peters. Election verifiability or ballot privacy: Do we need to choose? In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 481–498. Springer, Heidelberg, September 2013.

[13] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols (extended abstract). In *22nd FOCS*, pages 350–357. IEEE Computer Society Press, October 1981.

[14] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[15] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[16] Rolf Haenni, Philipp Locher, Reto E. Koenig, and Eric Dubuis. Pseudo-code algorithms for verifiable re-encryption mix-nets. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y. A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *FC 2017 Workshops*, volume 10323 of *LNCS*, pages 370–384. Springer, Heidelberg, April 2017.

[17] Thomas Haines, Rafieh Mosaheb, Johannes Müller, and Ivan Pryvalov. SoK: Secure E-voting with everlasting privacy. *PoPETs*, 2023(1):279–293, January 2023.

[18] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, oct 1969.

[19] Vincenzo Iovino, Alfredo Rial, Peter B. Rønne, and Peter Y. A. Ryan. Using selene to verify your vote in JCJ. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y. A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *FC 2017 Workshops*, volume 10323 of *LNCS*, pages 385–403. Springer, Heidelberg, April 2017.

[20] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. Cryptology ePrint Archive, Report 2002/165, 2002. https://eprint.iacr.org/2002/165.

[21] Robert Künnemann, Ilkan Esiyok, and Michael Backes. Automated verification of accountability in security protocols. In Stephanie Delaune and Limin Jia, editors, *CSF 2019 Computer Security Foundations Symposium*, pages 397–413. IEEE Computer Society Press, 2019.

[22] Robert Künnemann, Deepak Garg, and Michael Backes. Accountability in the decentralised-adversary setting. In Ralf Küsters and Dave Naumann, editors, *CSF 2021 Computer Security Foundations Symposium*, pages 1–16. IEEE Computer Society Press, 2021.

[23] Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. sElect: A lightweight verifiable remote voting system. In Michael Hicks and Boris Köpf, editors, *CSF 2016 Computer Security Foundations Symposium*, pages 341–354. IEEE Computer Society Press, 2016.

[24] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: definition and relationship to verifiability. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010*, pages 526–535. ACM Press, October 2010.

[25] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046, 2016. https://eprint.iacr.org/2016/046.

[26] Kevin Morio and Robert Künnemann. Verifying accountability for unbounded sets of participants. In Ralf Küsters and Dave Naumann, editors, *CSF 2021 Computer Security Foundations Symposium*, pages 1–16. IEEE Computer Society Press, 2021.

[27] Peter Y. A. Ryan, Peter B. Rønne, and Vincenzo Iovino. Selene: Voting with transparent verifiability and coercion-mitigation. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *LNCS*, pages 176–192. Springer, Heidelberg, February 2016.

[28] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. https://eprint.iacr.org/2004/332.

[29] Ben Smyth. Athena: A verifiable, coercion-resistant voting system with linear complexity. Cryptology ePrint Archive, Report 2019/761, 2019. https://eprint.iacr.org/2019/761.

[30] Ben Smyth. Surveying definitions of coercion resistance. Cryptology ePrint Archive, Report 2019/822, 2019. https://eprint.iacr.org/2019/822.

[31] Ben Smyth and Michael R. Clarkson. Surveying definitions of election verifiability. Cryptology ePrint Archive, Report 2022/305, 2022. https://eprint.iacr.org/2022/305.

[32] Björn Terelius and Douglas Wikström. Proofs of restricted shuffles. In Daniel J. Bernstein and Tanja Lange, editors, *AFRICACRYPT 10*, volume 6055 of *LNCS*, pages 100–113. Springer, Heidelberg, May 2010.

[33] Douglas Wikström. A commitment-consistent proof of a shuffle. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP 09*, volume 5594 of *LNCS*, pages 407–421. Springer, Heidelberg, July 2009.

[34] Marie-Laure Zollinger, Verena Distler, Peter Rønne, Peter Ryan, Carine Lallemand, and Vincent Koenig. User experience design for e-voting: How mental models align with security mechanisms. 10 2019.

Paper I

Efficient Mixing of Arbitrary Ballots with Everlasting Privacy: How to Verifiably Mix the PPATC Scheme

Kristian Gjøsteen, Thomas Haines and Morten Rotvold Solberg

Full version of a paper published at the Nordic Conference on Secure IT Systems, NordSec 2020. The paper is available at: eprint.iacr.org/2020/1331.

Efficient Mixing of Arbitrary Ballots with Everlasting Privacy: How to Verifiably Mix the PPATC Scheme

Kristian Gjøsteen[1], Thomas Haines[1], and Morten Rotvold Solberg[1]

Norwegian University of Science and Technology, Trondheim, Norway
{kristian.gjosteen,thomas.haines,mosolb}@ntnu.no

Abstract. The long term privacy of voting systems is of increasing concern as quantum computers come closer to reality. Everlasting privacy schemes offer the best way to manage these risks at present. While homomorphic tallying schemes with everlasting privacy are well developed, most national elections, using electronic voting, use mixnets. Currently the best candidate encryption scheme for making these kinds of elections everlastingly private is PPATC, but it has not been shown to work with any mixnet of comparable efficiency to the current ElGamal mixnets. In this work we give a paper proof, and a machine checked proof, that the variant of Wikström's mixnet commonly in use is safe for use with the PPATC encryption scheme.

Keywords: Everlasting Privacy · E-Voting · Verifiable Shuffles · Coq.

1 Introduction

Traditional paper-based and electronic voting has many good properties, but also limitations. A voter is not able to verify that her vote was counted as she cast it, and confidentiality of the votes relies heavily on trust in the election officials and procedures. In addition there are problems regarding for example counting errors and accessibility. Verifiable electronic voting systems can solve some of these issues. In particular, cryptographic techniques can be used to provide public verifiability of election results and raise each individual voter's confidence in the privacy and integrity of her vote.

Electronic voting has been plagued by mistakes, both in implementations but also in the cryptographic protocols. This means that security proofs are essential, and in particular it is desirable with automatically verifiable security proofs.

To achieve verifiable elections, encrypted votes are often published on a public bulletin board, along with sophisticated cryptographic proofs that allow an individual voter to verify that their ballot was not only listed on the bulletin board, but also included correctly in the tally.

Mix nets were first introduced by Chaum [2] as a solution to the traffic analysis problem in which an adversary is able to extract useful information from patterns of communication, even when that communication is encrypted. The traffic analysis problem can be thought of, more generally, as the set of problems that arise by the ability to link the messages between sets of senders and receivers. Mix nets therefore consist of a finite sequence of authorities (mixers), each of which permutes (shuffles) and hides the relationship between its inputs and its outputs. In the context of elections, mix nets are used to

transform the set of submitted encrypted ballots (which are linked to the voters) to the set of decrypted votes in the tally.

The votes are encrypted to provide confidentiality, which is usually considered essential for a fair vote. Confidentiality requires votes to remain private not only during the time of the election, but for all foreseeable future. However, due to computers and algorithms getting faster and the potential introduction of quantum computers, there is no way to safely predict how long it may take before a ciphertext encrypted today is broken. Thus, the property of *everlasting privacy* has been introduced.

Everlasting privacy is a property of electronic voting schemes where the information released to the public perfectly (or information-theoretically) hides how each voter voted, up to the outcome of the election. This means that regardless of developments in practical computing power and algorithm design, individual votes cannot be recovered from the public record.

Everlasting privacy is a subtle concept. In all systems that are practical for large-scale voting, functional requirements mean that the voter will have to encrypt their ballot and transmit this encryption to some infrastructure. The subtlety is that this ciphertext is not part of the public record. This essentially assumes that the potential powerful *future* attacker did not record the network traffic, and is only working with the public record of the election. This is in many cases a reasonable assumption. We emphasize that it is only privacy against these potential future attackers that relies on this assumption. Computationally secure cryptography still protects against adversaries with greater network access. So schemes that provide everlasting privacy are no less secure than conventional cryptographic voting schemes, but they have greater security against future adversaries that work only from the public record.

There are various candidate constructions which achieve everlasting privacy while maintaining verifiability. Most of the schemes are inspired by Cramer *et al.*'s "Multi-Authority Secret-Ballot Elections with Linear Work" [3] and Moran and Naor's "Split-ballot voting: Everlasting privacy with distributed trust" [11]. In both cases perfectly hiding commitments are combined with zero knowledge proofs to provide verifiability without leaking any information. In this work we will focus on schemes in the style of [11] which are able to handle arbitrary ballots rather than the homomorphic tally supported by [3]. This style of schemes are less developed than the homomorphic schemes, but have greater practical implications since mixnet style schemes have been used in many of countries who have voted electronically (Australia, Estonia, Norway, and Switzerland), and where homomorphic counting is often hard to do.

The general idea in these schemes is to have a publicly verifiable part dealing only with commitments to ballots. We achieve everlasting privacy by using perfectly hiding commitments. However, somehow the ballots must be recovered by the infrastructure, and this is done in a private part, typically working on encrypted openings for the commitments. In this way, we get everlasting privacy. Note that we only get computational integrity.

There are two encryption schemes which are commonly suggested for use in this context, both involve first committing to the message and then encrypting the opening to the commitment. The schemes fit into a wider everlasting privacy scheme with the perfectly

hiding commitments being publicly shuffled and then opened providing both verifiability and everlasting privacy; the encrypted openings are shuffled by the authorities and then publicly posted. The first is the MN encryption scheme from Moran and Naor [11] which is built on Paillier encryption [12] and Pedersen commitments [13]. The second is the PPATC encryption from Cuvelier *et al.* [4] which uses ElGamal and Abe *et al.*'s [1] commitment scheme. Since the latter encryption scheme can be instantiated on prime order elliptic curves, rather than the semi-prime RSA groups of the former, it is significantly faster.

Simple and efficient zero-knowledge proofs for correct encryption and decryption of both encryption schemes are known. An efficient mixnet for the MN encryption scheme was proven by Haines and Gritti [10], but at present the most efficient known mixnet for PPATC uses the general version of Terelius-Wikström proof of shuffle [14] which proves statements over the integers using Fujisaki-Okamoto commitments [6], based on an RSA modulus, which hampers the efficiency of the mixnet. The reason is that every operation must happen modulo the RSA modulus, which means that basic arithmetic is very slow. We will use pairing groups, but we arrange it so that most of the group arithmetic happens in a group where arithmetic is much faster, which means that Fujisaki-Okamoto commitments will be slow compared to most of our arithmetic. In practice everyone using the Terelius-Wikström proof of shuffle uses an optimised variant which avoids the use of Fujisaki-Okamoto commitments. It is folklore that the optimised variant of Terelius-Wikström works for wide class of encryption schemes but the precise variant for each encryption scheme should be proven.

1.1 Contribution

We prove a variation of the optimised Terelius-Wikström shuffle [14] for the PPATC encryption scheme [4]. This is essentially the optimised variation which is widely used, and which avoids the use of Fujisaki-Okamoto commitments. In addition we show how the Fiat-Shamir transform can be applied so that the public proofs of correct shuffling can be trivially derived from the private proofs of correct shuffling, nearly doubling the speed of mixing.

We provide a machine-checked proof using the interactive theorem prover Coq. The machine-checked proof relies on recent work which shows that any encryption scheme with certain properties works with the optimised Terelius-Wikström shuffle. For completeness and human understanding, we also give a straight-forward traditional paper proof.

2 Notation and Tools

We denote by $\mathbb{G}_1$ and $\mathbb{G}_2$ cyclic groups of large prime order q, and by $\mathbb{Z}_q$ the field of integers modulo q. Let A^n be the set of vectors of length n, with elements from the set A. We denote vectors in bold, e.g. $\mathbf{a}$. We denote by a_i the ith element of the vector $\mathbf{a}$. Sometimes, we will work with vectors that have tuples as elements. In such cases, we also denote by a_i the ith element of $\mathbf{a}$, and by $a_{i,j}$ the jth element of the tuple a_i. Multiplication

of tuples is elementwise multiplication, that is, $\mathbf{ab}$ is the tuple where the ith element is $a_i b_i$. We denote by $A^{n\times n}$ the set of $n \times n$-matrices with elements from the set A. Matrices will be denoted using capital letters, e.g. M. We denote by M_i the ith column of M, by $M_{i,*}$ the ith row of M, and by $M_{i,j}$ the element in row i and column j. A binary relation for a set S of statements and a set W of witnesses is a subset of $S \times W$ and is denoted by $\mathscr{R}$.

Matrix Commitments. We now describe how to commit to a matrix using a variation of Pedersen commitments [14]. We denote by $\mathsf{Com}_{\gamma,\gamma_1}(m,t)$ the Pedersen commitment of $m \in \mathbb{Z}_q$ with randomness $t \in \mathbb{Z}_q$, i.e. $\mathsf{Com}_{\gamma,\gamma_1}(m,t) = \gamma^t\gamma_1^m$ for group generators γ and γ_1. To commit to a vector $\mathbf{v} \in \mathbb{Z}_q^n$, we compute $u = \mathsf{Com}_{\gamma,\gamma_1,\cdots,\gamma_n}(\mathbf{v},t) = \gamma^t \prod_{i=1}^n \gamma_i^{v_i}$, where t is chosen at random from $\mathbb{Z}_q$, and the γs are random group generators. If the commitment parameters are omitted, it is implicit that they are $\gamma, \gamma_1, \cdots, \gamma_n$. An $n \times n$ matrix M is committed to column-wise. For a matrix $M \in \mathbb{Z}_q^{n\times n}$ and a vector $\mathbf{t}$ chosen at random from $\mathbb{Z}_q^n$, we compute the commitment $\mathbf{u}$ of M as

$$\mathbf{u} = \mathsf{Com}(M,\mathbf{t}) = \big(\mathsf{Com}(M_1,t_1),\ldots,\mathsf{Com}(M_n,t_n)\big) = \big(\gamma^{t_1}\Pi_{i=1}^n\gamma_i^{M_{i,1}},\ldots,\gamma^{t_n}\Pi_{i=1}^n\gamma_i^{M_{i,n}}\big).$$

Abe Commitments. We now describe a perfectly hiding commitment scheme due to Abe *et al.* [1], that is used in a a construction of the PPATC encryption scheme that we describe further down. Let $\Lambda_{sxdh} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$ be a description of bilinear groups, where g is a generator of $\mathbb{G}_1$, h is a generator of $\mathbb{G}_2$ and e is an efficient and non-degenerate bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We assume that the DDH problem is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$. In our notation, an Abe commitment to a message $m \in \mathbb{G}_1$ is the tuple $(h^{r_1}h_1^{r_2}, mg_1^{r_2})$, where r_1 and r_2 are random elements in $\mathbb{Z}_q$ and g_1 and h_1 are random elements of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. An Abe commitment to m can be thought of as an ElGamal encryption of m where the first coordinate is hidden in a Pedersen commitment. An opening is of the form $(g_1^{r_1}, m)$ which is valid if $e(g, h^{r_1}h_1^{r_2}) = e(g_1^{r_1}, h)e(mg_1^{r_2}/m, h_1)$.

Polynomial Identity Testing. We will make use of the Schwartz-Zippel lemma to analyze the soundness of our protocol. The lemma gives an efficient method for testing whether a polynomial is equal to zero.

Lemma 1 (Schwartz-Zippel). *Let $f \in \mathbb{Z}_q[X_1, ..., X_n]$ be a non-zero polynomial of total degree $d \geq 0$ over $\mathbb{Z}_q$. Let $S \subseteq \mathbb{Z}_q$ and let $x_1, ..., x_n$ be chosen uniformly at random from S. Then* $\Pr[f(x_1, ..., x_n) = 0] \leq d/|S|$.

3 Commitment Consistent Encryption

We now describe *commitment consistent encryption* (CCE), as defined by Cuvelier *et al.* [4]. The key idea is that for any ciphertext, one can derive a commitment to that ciphertext, and the secret key can be used to obtain an opening to that commitment. Furthermore, applied in a voting protocol, the idea is that the voters compute a CC encryption of their ballot, and the authorities derive a commitment to the ciphertext and

post this commitment on a public bulletin board PB. If the commitments are perfectly hiding, they can be used to provide a perfectly private audit trail, which allows anyone to verify the correctness of the count, but does not contain any information about who submitted which ballots.

Definition 1 (CC Encryption [4]). *A commitment consistent encryption scheme Π is a tuple of six efficient algorithms*

$$\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{DeriveCom}, \mathsf{Open}, \mathsf{Verify}),$$

defined as follows:

- $\mathsf{Gen}(1^\lambda)$*: on input a security parameter λ, output a triple (pp, pk, sk) of public parameters, public key and secret key. The public parameter pp is implicitly given as input to the rest of the algorithms.*
- $\mathsf{Enc}_{pk}(m)$*: output a ciphertext c, which is an encryption of a message m in the plaintext space $\mathcal{M}$ (defined by pp) using public key pk.*
- $\mathsf{Dec}_{sk}(c)$*: for a ciphertext c in the ciphertext space $\mathcal{C}$ (defined by pp), output a message m using secret key sk.*
- $\mathsf{DeriveCom}_{pk}(c)$*: From a ciphertext c, output a commitment d using pk.*
- $\mathsf{Open}_{sk}(c)$*: from a ciphertext c, output an auxiliary value a, that can be considered as part of an opening for a commitment d.*
- $\mathsf{Verify}_{pk}(d, m, a)$*: On input a message m and a commitment d wrt. public key pk, and auxiliary value a, output a bit. The algorithm checks that the opening (m, a) is valid wrt. d and pk.*

Correctness. We expect that any commitment consistent encryption scheme satisfies the following correctness property: For any triple (pp, pk, sk) output by Gen, any message $m \in \mathcal{M}$ and any $c = \mathsf{Enc}_{pk}(m)$, it holds with overwhelming probability in the security parameter that $\mathsf{Dec}_{sk}(c) = m$ and $\mathsf{Verify}_{pk}\big(\mathsf{DeriveCom}_{pk}(c), \mathsf{Dec}_{sk}(c), \mathsf{Open}_{sk}(c)\big) = 1$.

The above definition does not guarantee that it is infeasible to create honest-looking CCE ciphertexts that are in fact not consistent. To address this issue, Cuvelier *et al.* [4] define the concept of *validity augmentation* (VA) for CCE schemes. A validity augmentation adds three new algorithms $\mathsf{Expand}, \mathsf{Valid}$ and Strip to the scheme.

The Expand algorithm augments the public key for use in the other algorithms. The Valid algorithm takes as input an augmented ciphertext c^{va} along with some proofs of validity. It then checks whether it is possible to derive a commitment and an encryption of an opening to that commitment. The Strip algorithm removes the proofs of validity.

Definition 2 (Validity Augmentation [4]). *A scheme*

$$\Pi^{va} = (\mathsf{VA.Gen}, \mathsf{VA.Enc}, \mathsf{VA.Dec}, \mathsf{VA.DeriveCom}, \mathsf{VA.Open}, \mathsf{VA.Verify}, \mathsf{Expand}, \mathsf{Strip}, \mathsf{Valid})$$

is a validity augmentation of the CCE scheme

$$\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{DeriveCom}, \mathsf{Open}, \mathsf{Verify})$$

if the following conditions are satisfied:

- Augmentation: $\mathsf{VA.Gen}$ *runs* Gen *to obtain* (pp, pk, sk) *and outputs an updated triple* $(pp^{\mathsf{va}}, pk^{\mathsf{va}}, sk^{\mathsf{va}}) = (pp, \mathsf{Expand}(pk), sk)$.
- Validity: $\mathsf{Valid}_{pk^{\mathsf{va}}}(c^{\mathsf{va}}) = 1$ *for all honestly generated public keys and ciphertexts. In addition, for any PPT adversary* $\mathcal{A}$*, the following probability is negligible in* λ*:*

$$\Pr[\mathsf{Valid}_{pk^{\mathsf{va}}}(c^{\mathsf{va}}) = 1 \wedge \neg \mathsf{Verify}_{pk}(\mathsf{Strip}_{pk^{\mathsf{va}}}(c^{\mathsf{va}})) = 1 \mid c \leftarrow \mathcal{A}(pp^{\mathsf{va}}, pk^{\mathsf{va}}); (pp^{\mathsf{va}}, pk^{\mathsf{va}}, sk^{\mathsf{va}}) \leftarrow \mathsf{VA.Gen}]$$

- Consistency: *The values* $\mathsf{Strip}_{pk^{\mathsf{va}}}(\mathsf{VA.enc}_{pk^{\mathsf{va}}}(m))$ *and* $\mathsf{Enc}_{pk}(m)$ *are equally distributed for all* $m \in \mathcal{M}$*, i.e. it is possible to strip a validity augmented ciphertext into a "normal" one. In addition, it holds, for all ciphertexts and keys, that* $\mathsf{VA.Dec}_{sk^{\mathsf{va}}}(c^{\mathsf{va}}) = \mathsf{Dec}_{sk}(\mathsf{Strip}_{pk^{\mathsf{va}}}(c^{\mathsf{va}}))$*, that* $\mathsf{VA.Open}_{sk^{\mathsf{va}}}(c^{\mathsf{va}}) = \mathsf{Open}_{sk}(\mathsf{Strip}_{pk^{\mathsf{va}}}(c^{\mathsf{va}}))$ *and that* $\mathsf{VA.Verify}_{pk^{\mathsf{va}}}(c^{\mathsf{va}}) = \mathsf{Verify}_{pk}(\mathsf{Strip}_{pk^{\mathsf{va}}}(c^{\mathsf{va}}))$*. In other words, the decryption, opening and verification for* Π^{va} *is consistent with those of* Π*.*

3.1 The PPATC Encryption System

We now describe an augmented CCE system called PPATC (Perfectly Private Audit Trail with Complex ballots). The different algorithms are defined as follows [4]:

- $\mathsf{VA.Gen}(1^\lambda)$: Generate $\Lambda_{sxdh} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$ and random generators $g_1 = g^{x_1}, g_2 = g^{x_2} \in \mathbb{G}_1$ and $h_1 \in \mathbb{G}_2$. Now, $(pp, pk, sk) = ((\Lambda_{sxdh}, h_1), (g_1, g_2), (x_1, x_2))$. The augmented key $pk^{\mathsf{va}} = \mathsf{Expand}(pk)$ is computed by adding a description of a hash function $\mathcal{H}$ with range $\mathbb{Z}_q$ to the public key, resulting in the triple $(pp^{\mathsf{va}} = pp, pk^{\mathsf{va}}, sk^{\mathsf{va}} = sk)$.
- $\mathsf{VA.Enc}_{pk^{\mathsf{va}}}(m; r)$: Compute the CCE ciphertext $c = \mathsf{Enc}_{pk}(m; r)$ where $c = (c_1, c_2, c_3, d_1, d_2) = (g^{r_2}, g^{r_3}, g_1^{r_1} g_2^{r_3}, h^{r_1} h_1^{r_2}, m g_1^{r_2})$ and $r = (r_1, r_2, r_3) \in \mathbb{Z}_q^3$. Then compute the following validity proof. Select $s_1, s_2, s_3 \xleftarrow{r} \mathbb{Z}_q$ and compute $c' = (c'_1, c'_2, c'_3, d'_1) = (g^{s_2}, g^{s_3}, g_1^{s_1} g_2^{s_3}, h^{s_1} h_1^{s_2})$. Compute $\nu_{cc} = \mathcal{H}(pp^{\mathsf{va}}, pk^{\mathsf{va}}, c, c')$, $f_1 = s_1 + \nu_{cc} r_1$, $f_2 = s_2 + \nu_{cc} r_2$ and $f_3 = s_3 + \nu_{cc} r_3$. Let $\sigma_{cc} = (\nu_{cc}, f_1, f_2, f_3)$. The ciphertext is $c^{\mathsf{va}} = (c, \sigma_{cc})$.
- $\mathsf{VA.Dec}_{sk^{\mathsf{va}}}(c^{\mathsf{va}})$: Parse c^{va} as $(c_1, c_2, c_3, d_1, d_2, \sigma_{cc})$ and return $d_2 / c_1^{x_1}$.
- $\mathsf{VA.DeriveCom}_{pk^{\mathsf{va}}}(c^{\mathsf{va}})$: Parse c^{va} as $(c_1, c_2, c_3, d_1, d_2, \sigma_{cc})$ and return (d_1, d_2).
- $\mathsf{VA.Open}_{sk^{\mathsf{va}}}(c^{\mathsf{va}})$: Parse c^{va} as $(c_1, c_2, c_3, d_1, d_2, \sigma_{cc})$ and return $a = c_3 / c_2^{x_2}$.
- $\mathsf{VA.Verify}_{pk^{\mathsf{va}}}(d_1, d_2, m, a)$: Return 1 if $e(g, d_1) = e(a, h) e(d_2 / m, h_1)$ and 0 otherwise.
- $\mathsf{Valid}_{pk^{\mathsf{va}}}(c^{\mathsf{va}})$: Parse c^{va} as $(c_1, c_2, c_3, d_1, d_2, \nu_{cc}, f_1, f_2, f_3)$ and check if all elements of c^{va} are properly encoded. Compute $c'_1 = g^{f_2} / c_1^{\nu_{cc}}$, $c'_2 = g^{f_3} / c_2^{\nu_{cc}}$, $c'_3 = g_1^{f_1} g_2^{f_3} / c_3^{\nu_{cc}}$ and $d'_1 = h^{f_1} h_1^{f_2} / d_1^{\nu_{cc}}$. Return 1 only if

$$\nu_{cc} = \mathcal{H}(pp^{\mathsf{va}}, pk^{\mathsf{va}}, c_1, c_2, c_3, d_1, d_2, c'_1, c'_2, c'_3, d'_1, d'_2).$$

– $\mathsf{Strip}_{pk^{\mathsf{va}}}(c^{\mathsf{va}})$: Parse c^{va} as $(c_1, c_2, c_3, d_1, d_2, \sigma_{cc})$ and return the CCE ciphertext $c = (c_1, c_2, c_3, d_1, d_2)$ and the commitment $d = (d_1, d_2)$.

A CCE ciphertext $c = \mathsf{Enc}_{pk}(m; r) = (g^{r_2}, g^{r_3}, g_1^{r_1} g_2^{r_3}, h^{r_1} h_1^{r_1}, m g_1^{r_2})$ can be *re-encrypted*, by multiplying c with the encryption of 1 using randomness $r' = (r'_1, r'_2, r'_3) \in \mathbb{Z}_q^3$. Thus, a ciphertext c', where

$$\begin{aligned} c' &= c \cdot \mathsf{Enc}_{pk}(1; r') = (g^{r_2}, g^{r_3}, g_1^{r_1} g_2^{r_3}, h^{r_1} h_1^{r_2}, m g_1^{r_2}) \cdot (g^{r'_2}, g^{r'_3}, g_1^{r'_1} g_2^{r'_3}, h^{r'_1} h_1^{r'_2}, g_1^{r'_2}) \\ &= (g^{r_2+r'_2}, g^{r_3+r'_3}, g_1^{r_1+r'_1} g_2^{r_3+r'_3}, h^{r_1+r'_1} h_1^{r_2+r'_2}, m g_1^{r_2+r'_2}), \end{aligned}$$

can be thought of as an encryption of m using randomness $r + r'$.

4 Shuffling Commitment Consistent Ciphertexts

In this section, we first describe how the PPATC can be used as a building block in a voting system. We then concrete shuffle algorithms for shuffling PPATC ciphertexts and their derived commitments, before describing how to apply the Fiat-Shamir heuristic to make the shuffles non-interactive.

4.1 Using the PPATC scheme in a Voting System

A validity augmented CCE scheme can be applied in an election as follows [4]. First, a setup phase takes place, where the election authorities generate encryption and decryption keys, as well as two bulletin boards PB and SB. The public board PB will contain the public audit trail, while SB will contain encrypted votes, be kept secret by the authorities and will be used to compute the tally. To produce a ballot, each voter encrypts her vote using the PPATC scheme, and sends the resulting ciphertext to the authorities. The ciphertext is stored on SB and the derived commitment is stored on PB.

To preserve privacy, the link between voter and vote must be destroyed, the list of ciphertexts on SB is *shuffled*. A shuffle of a list $\mathbf{v}$ of ciphertexts is a new list $\mathbf{v}'$, such that for all $i = 1, \ldots, n$, $v'_i = v_{\pi(i)} \cdot \mathsf{Enc}_{pk}(1; r_{\pi(i)})$, where $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ is a randomly chosen permutation. Thus, the two lists $\mathbf{v}$ and $\mathbf{v}'$ contain encryptions of the same plaintexts in permuted order. To also provide verifiability, we keep track of the concordance between the ciphertexts on SB and the corresponding commitments on PB. To achieve this, the list of commitments on PB is also shuffled, using the same permutation as for SB.

The lists are shuffled several times, by a series of *mix servers*. It is necessary that each mix server provides a *proof of shuffle*, to prove that he follows the protocol, and that the lists of ciphertexts in fact decrypt to the same plaintexts. For our shuffle algorithms we will use the optimised version of the Terelius-Wikström shuffle presented by Haenni *et al.* [7], where a proof of shuffle consists of proving knowledge of the permutation π and the random vector $\mathbf{r}$ used to re-encrypt the ciphertexts.

Thus, the tally procedure will proceed as follows:

1. *Stripping*: Algorithms Valid and Strip are run on the ciphertexts stored on SB to obtain a vector $\mathbf{v}$ of n CCE ciphertexts and a vector $\mathbf{d}$ of the corresponding commitments.
2. *Performing the shuffles*: Each mix server selects a random permutation $\pi : \{1,\ldots,n\} \to \{1,\ldots,n\}$, also defining a permutation matrix M, and computes a commitment $\mathbf{u}$ on that permutation matrix, along with a proof of knowledge of the permutation. The mix server then selects a random vector $\mathbf{r} = ((r_{1,1}, r_{1,2}, r_{1,3}), \cdots, (r_{n,1}, r_{n,2}, r_{n,3}))$ and computes a new vector $\mathbf{v}'$ where $v'_i = v_{\pi(i)} \cdot \mathsf{Enc}_{pk}(1; r_{\pi(i)})$, and $r_{\pi(i)} = (r_{\pi(i),1}, r_{\pi(i),2}, r_{\pi(i),3})$. Let the last two components of each ciphertext v'_i form a vector $\mathbf{d}'$. This vector is posted on PB. Finally, the mix server computes two commitment consistent proofs of shuffle, showing that $\mathbf{v}'$ is a shuffle of $\mathbf{v}$ and $\mathbf{d}'$ is a shuffle of $\mathbf{d}$, with respect to the permutation π.
3. *Decryption of openings:* The authorities verify the proofs and perform a threshold decryption of the ciphertexts in $\mathbf{v}'$. In addition, they run the algorithm Open on these ciphertexts to obtain the auxiliary values for the commitments. The plaintexts and the auxiliary values are posted on PB.

4.2 Proof of Shuffle on the Private Board

We start with the shuffle on the private board, i.e. the shuffle of the CCE ciphertexts. In the following, let $\mathscr{R}_{com}$ be a relation between the commitment parameters $\gamma, \gamma_1, ..., \gamma_n \in \mathbb{G}_1$, $\mathbf{m}, \mathbf{m}' \in \mathbb{Z}_q^n$ and $t, t' \in \mathbb{Z}_q$ which holds if and only if $\mathsf{Com}_{\gamma,\gamma_1,...,\gamma_n}(\mathbf{m}, t) = \mathsf{Com}_{\gamma,\gamma_1,...,\gamma_n}(\mathbf{m}', t')$ and $\mathbf{m} \neq \mathbf{m}'$. Let $\mathscr{R}_\pi$ be the relation between the commitment parameters $\gamma, \gamma_1, ..., \gamma_n$, a commitment $\mathbf{u} \in \mathbb{G}_1^n$, a permutation matrix $M \in \mathbb{Z}_q^{n\times n}$ and a randomness vector $\mathbf{t} \in \mathbb{Z}_q^n$ which holds only if $\mathbf{u} = \mathsf{Com}_{\gamma,\gamma_1,...\gamma_n}(M, \mathbf{t})$. Let $\mathscr{R}^{shuf}_{ReEnc}(pk, (v_1, ..., v_n), (v'_1, ..., v'_n))(\pi, (r_1, ..., r_n))$, where π is a permutation of the set $\{1, ..., n\}$, be the relation which holds if and only if $v'_i = v_{\pi(i)} \cdot \mathsf{Enc}_{pk}(1; r_{\pi(i)})$ for all $i \in \{1, ..., n\}$.

Theorem 1. *Protocol 1 is a perfectly complete, 4-round special soundness, special honest-verifier zero-knowledge proof of knowledge of the relation* $\mathscr{R}_{com} \vee (\mathscr{R}_\pi \wedge \mathscr{R}^{shuf}_{ReEnc})$.

It is infeasible under the discrete log assumption to find a witness for $\mathscr{R}_{com}$, so Theorem 1 implies a proof of knowledge for $(\mathscr{R}_\pi \wedge \mathscr{R}^{shuf}_{ReEnc})$. To prove the theorem, we now demonstrate the completeness of the protocol, as well as the special soundness extractor and the special honest-verifier zero-knowledge simulator.

Completeness. We now show that Protocol 1 is complete, i.e. that in an honest run, the verifier accepts the proof. The proof consists of algebraic manipulations.

$$a_1 = \gamma^{z_1} = (\gamma^{\bar{t}})^{-\beta}\gamma^{z_1+\beta\bar{t}} = (\gamma^{\bar{t}} \Pi_{i=1}^n \gamma_i / \Pi_{i=1}^n \gamma_i)^{-\beta}\gamma^{b_1} = (\Pi_{i=1}^n u_i / \Pi_{i=1}^n \gamma_i)^{-\beta}\gamma^{b_1}.$$

$$a_2 = \gamma^{z_2} = (\gamma^{\hat{t}})^{-\beta}\gamma^{z_2+\beta\hat{t}} = (\gamma^{\hat{t}}\gamma_1^{\Pi_{i=1}^n w'_i} / \gamma_1^{\Pi_{i=1}^n w_i})^{-\beta}\gamma^{b_2} = (\hat{u}_n / \gamma_1^{\Pi_{i=1}^n w_i})^{-\beta}\gamma^{b_2}.$$

Protocol 1 Interactive ZK-Proof of Shuffle on Private Board

Common Input: A public key pk, a matrix commitment $\mathbf{u}$, commitment parameters $\gamma, \gamma_1, ..., \gamma_n$ and ciphertext vectors $\mathbf{v}, \mathbf{v}' \in (\mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1)^n$.

Private Input: Permutation matrix $M \in \mathbb{Z}_q^{n \times n}$ and randomness $\mathbf{t} \in \mathbb{Z}_q^n$ such that $\mathbf{u} = \mathsf{Com}(M, \mathbf{t})$. Randomness $\mathbf{r} \in (\mathbb{Z}_q \times \mathbb{Z}_q \times \mathbb{Z}_q)^n$ such that $v'_i = v_{\pi(i)} \cdot \mathsf{Enc}_{pk}(1; r_{\pi(i)})$ for $i = 1, ..., n$.

1: $\mathcal{V}$ chooses a random $\mathbf{w} \in \mathbb{Z}_q^n$ and sends $\mathbf{w}$ to $\mathcal{P}$.

2: $\mathcal{P}$ computes $\mathbf{w}' = (w'_1, ..., w'_n) = M\mathbf{w}$, and randomly chooses $\hat{\mathbf{t}} = (\hat{t}_1, \ldots, \hat{t}_n)$, $\hat{\mathbf{z}} = (\hat{z}_1, ..., \hat{z}_n)$, $\mathbf{z}' = (z'_1, ..., z'_n) \in \mathbb{Z}_q^n, z_1, z_2, z_3 \in \mathbb{Z}_q$ and $\tilde{\mathbf{z}} = (\tilde{z}_1, \tilde{z}_2, \tilde{z}_3) \in \mathbb{Z}_q^3$. $\mathcal{P}$ defines

$$\bar{t} = \langle \mathbf{1}, \mathbf{t} \rangle,\ \tilde{t} = \langle \mathbf{t}, \mathbf{w} \rangle,\ \hat{t} = \hat{t}_n + \sum_{i=1}^{n-1} \left(\hat{t}_i \prod_{j=i+1}^{n} w'_j \right) \text{ and}$$

$$\mathbf{r}' = \left(\sum_{i=1}^{n} r_{i,1} w_i, \sum_{i=1}^{n} r_{i,2} w_i, \sum_{i=1}^{n} r_{i,3} w_i \right),$$

and sends the following elements to $\mathcal{V}$ (for $i = 1, \ldots, n$):

$$\hat{u}_0 = \gamma_1 \qquad \hat{u}_i = \gamma^{\hat{t}_i} (\hat{u}_{i-1})^{w'_i} \qquad a_1 = \gamma^{\tilde{z}_1} \qquad a_2 = \gamma^{\tilde{z}_2}$$
$$a_3 = \gamma^{\tilde{z}_3} \Pi_{i=1}^n \gamma_i^{z'_i} \qquad a_4 = \mathsf{Enc}_{pk}(1; \tilde{\mathbf{z}}) \Pi_{i=1}^n (v'_i)^{z'_i} \qquad \hat{a}_i = \gamma^{\hat{z}_i} (\hat{u}_{i-1})^{z'_i}.$$

3: $\mathcal{V}$ chooses a random challenge $\beta \in \mathbb{Z}_q$ and sends β to $\mathcal{P}$.

4: For $i \in \{1, \ldots, n\}$, $\mathcal{P}$ responds with

$$b_1 = z_1 + \beta \cdot \bar{t} \qquad b_2 = z_2 + \beta \cdot \hat{t} \qquad b_3 = z_3 + \beta \cdot \tilde{t}$$
$$\tilde{\mathbf{b}} = \tilde{\mathbf{z}} - \beta \cdot \mathbf{r}' \qquad \hat{b}_i = \hat{z}_i + \beta \cdot \hat{t}_i \qquad b'_i = z'_i + \beta \cdot w'_i.$$

5: $\mathcal{V}$ accepts if and only if, for $i \in \{1, \ldots, n\}$

$$a_1 = (\Pi_{i=1}^n u_i / \Pi_{i=1}^n \gamma_i)^{-\beta} \cdot \gamma^{b_1} \quad a_2 = (\hat{u}_n / \gamma_1^{\Pi_{i=1}^n w_i})^{-\beta} \cdot \gamma^{b_2}$$
$$a_3 = (\Pi_{i=1}^n u_i^{w_i})^{-\beta} \cdot \gamma^{b_3} \cdot \Pi_{i=1}^n \gamma_i^{b'_i} \qquad a_4 = (\Pi_{i=1}^n v_i^{w_i})^{-\beta} \cdot \mathsf{Enc}_{pk}(1; \tilde{\mathbf{b}}) \cdot \Pi_{i=1}^n (v'_i)^{b'_i}$$
$$\hat{a}_i = (\hat{u}_i)^{-\beta} \cdot \gamma^{\hat{b}_i} \cdot (\hat{u}_{i-1})^{b'_i}$$

$$\begin{aligned} a_3 &= \gamma^{z_3} \Pi_{i=1}^n \gamma_i^{z'_i} = (\gamma^{\tilde{t}} \Pi_{i=1}^n \gamma_i^{w'_i})^{-\beta} \gamma^{z_3+\beta\tilde{t}} \Pi_{i=1}^n \gamma_i^{z'_i+\beta w'_i} \\ &= (\Pi_{i=1}^n u_i^{w_i})^{-\beta} \gamma^{b_3} \Pi_{i=1}^n \gamma_i^{b'_i}. \end{aligned}$$

$$\begin{aligned} a_4 &= \mathsf{Enc}_{pk}(1;\tilde{\mathbf{z}}) \cdot \Pi_{i=1}^n (v'_i)^{z'_i} = \mathsf{Enc}_{pk}(1;\tilde{\mathbf{b}}) \cdot \mathsf{Enc}_{pk}(1;\beta \cdot \mathbf{r}') \cdot \Pi_{i=1}^n (v'_i)^{z'_i} \\ &= \mathsf{Enc}_{pk}(1;\tilde{\mathbf{b}}) \cdot \Pi_{i=1}^n (v'_i)^{b'_i} \cdot \mathsf{Enc}_{pk}(1;\beta \cdot \mathbf{r}') \cdot \Pi_{i=1}^n (v'_i)^{-\beta w'_i} \\ &= (\Pi_{i=1}^n v_i^{w_i})^{-\beta} \cdot \mathsf{Enc}_{pk}(1;\tilde{\mathbf{b}}) \cdot \Pi_{i=1}^n (v'_i)^{b'_i}. \end{aligned}$$

$$\hat{a}_i = \gamma^{\hat{b}_i} \gamma^{-\beta\hat{t}_i} (\hat{u}_{i-1})^{z'_i} = \gamma^{\hat{b}_i} (\hat{u}_{i-1})^{b'_i} \gamma^{-\beta\hat{t}_i} (\hat{u}_{i-1})^{-\beta w'_i} = (\hat{u}_i)^{-\beta} \gamma^{\hat{b}_i} (\hat{u}_{i-1})^{b'_i}.$$

Thus, all verification equations are satisfied.

Special Soundness. We will follow the structure of Terelius & Wikström [14] and split the extractor in two parts. In the first part, the basic extractor, we show that for two accepting transcripts with the same $\mathbf{w}$ but different β, we can extract witnesses for certain sub-statements. In the second part, the extended extractor, we show that we can extract a witness to the main statement, given witnesses which hold for these sub-statements, for n different $\mathbf{w}$.

Basic extractor. Given two accepting transcripts

$$\begin{aligned} &(\mathbf{w}, \hat{\mathbf{u}}, a_1, a_2, a_3, a_4, \hat{\mathbf{a}}, \beta, b_1, b_2, b_3, \tilde{\mathbf{b}}, \hat{\mathbf{b}}, \mathbf{b}') \\ &(\mathbf{w}, \hat{\mathbf{u}}, a_1, a_2, a_3, a_4, \hat{\mathbf{a}}, \beta^*, b_1^*, b_2^*, b_3^*, \tilde{\mathbf{b}^*}, \hat{\mathbf{b}}^*, \mathbf{b}'^*) \end{aligned}$$

where $\beta \neq \beta^*$, the basic extractor computes

$$\begin{aligned} &\bar{t} = (b_1 - b_1^*)/(\beta - \beta^*) \qquad \hat{t} = (b_2 - b_2^*)/(\beta - \beta^*) \qquad \tilde{t} = (b_3 - b_3^*)/(\beta - \beta^*) \\ &\hat{\mathbf{t}}' = (\hat{\mathbf{b}} - \hat{\mathbf{b}}^*)/(\beta - \beta^*) \qquad \mathbf{w}' = (\mathbf{b}' - \mathbf{b}'^*)/(\beta - \beta^*) \qquad \mathbf{r}' = (\tilde{\mathbf{b}} - \tilde{\mathbf{b}^*})/(\beta - \beta^*) \end{aligned}$$

We will prove that

$$\begin{gathered} \Pi_{i=1}^n u_i = \mathsf{Com}(\mathbf{1}, \bar{t}), \quad \Pi_{i=1}^n u_i^{w_i} = \mathsf{Com}(\mathbf{w}', \tilde{t}), \quad \Pi_{i=1}^n v_i^{w_i} = \Pi_{i=1}^n (v'_i)^{w'_i} \cdot \mathsf{Enc}_{pk}(1; -\mathbf{r}'), \\ \hat{u}_i = \mathsf{Com}_{\gamma, \hat{u}_{i-1}}(w'_i, \hat{t}'_i) \quad \text{and} \quad \hat{u}_n = \mathsf{Com}_{\gamma,\gamma_1}\left(\Pi_{i=1}^n w_i, \hat{t}\right). \end{gathered}$$

The proof consists of algebraic manipulations:

$$\Pi_{i=1}^n u_i = \left(\frac{(\Pi_{i=1}^n u_i)^\beta \cdot a_1}{(\Pi_{i=1}^n u_i)^{\beta^*} \cdot a_1} \right)^{\frac{1}{\beta - \beta^*}} = \gamma^{\frac{b_1 - b_1^*}{\beta - \beta^*}} \cdot \Pi_{i=1}^n \gamma_i = \mathsf{Com}(\mathbf{1}, \bar{t}).$$

$$\Pi_{i=1}^n u_i^{w_i} = \left(\frac{(\Pi_{i=1}^n u_i^{w_i})^\beta \cdot a_3}{(\Pi_{i=1}^n u_i^{w_i})^{\beta^*} \cdot a_3} \right)^{\frac{1}{\beta - \beta^*}} = \gamma^{\frac{b_3 - b_3^*}{\beta - \beta^*}} \cdot \Pi_{i=1}^n \gamma_i^{\frac{b'_i - b'^*_i}{\beta - \beta^*}} = \mathsf{Com}(\mathbf{w}', \tilde{t}).$$

$$\Pi_{i=1}^{n} v_i^{w_i} = \left(\frac{(\Pi_{i=1}^{n} v_i^{w_i})^{\beta} \cdot a_4}{(\Pi_{i=1}^{n} v_i^{w_i})^{\beta^*} \cdot a_4} \right)^{\frac{1}{\beta - \beta^*}} = \Pi_{i=1}^{n} (v'_i)^{\frac{b'_i - b'^*_i}{\beta - \beta^*}} \cdot \mathsf{Enc}_{pk}\left(1; \frac{\tilde{\mathbf{b}} - \tilde{\mathbf{b}^*}}{\beta - \beta^*}\right)$$
$$= \Pi_{i=1}^{n} (v'_i)^{w'_i} \cdot \mathsf{Enc}_{pk}(1; -\mathbf{r}').$$

$$\hat{u}_i = \gamma^{\frac{\hat{b}_i - \hat{b}^*_i}{\beta - \beta^*}} \cdot (\hat{u}_{i-1})^{\frac{b'_i - b'^*_i}{\beta - \beta^*}} = \gamma^{\hat{t}'_i} \cdot (\hat{u}_{i-1})^{w'_i} = \mathsf{Com}_{\gamma, \hat{u}_{i-1}}(w'_i, \hat{t}'_i).$$

$$\hat{u}_n = \gamma^{\frac{b_2 - b^*_2}{\beta - \beta^*}} \cdot \gamma_1^{\Pi_{i=1}^{n} w_i} = \gamma^{\hat{t}} \cdot \gamma_1^{\Pi_{i=1}^{n} w_i} = \mathsf{Com}_{\gamma, \gamma_1}(\Pi_{i=1}^{n} w_i; \hat{t}).$$

Thus, all the equations are satisfied.

Extended Extractor. The extended extractor takes, for one statement, n different witnesses extracted by the basic extractor, and produces a witness for the main statement. Let $\bar{\mathbf{t}}, \hat{\mathbf{t}}, \tilde{\mathbf{t}} \in \mathbb{Z}_q^n$, $\mathbf{r}' \in (\mathbb{Z}_q \times \mathbb{Z}_q \times \mathbb{Z}_q)^n$ and $\hat{T}', W' \in \mathbb{Z}_q^{n \times n}$ be the collective output from the n runs of the basic extractor, extracted from challenges $W \in \mathbb{Z}_q^{n \times n}$. Let W_j be the jth column of W, i.e. the challenge vector from the jth run of the basic extractor. The challenge vectors are sampled from a uniform distribution, but since the cheating prover may not succeed with uniform probability for all challenge vectors, the final distribution of challenge vectors is non-uniform. However, since the adversary has a significant probability of success, any set of challenge vectors with a significant success probability must be much larger than the set of non-invertible matrices. It follows that the columns of W will be linearly independent with overwhelming probability.

Thus, W will, with overwhelming probability, have an inverse. We call this inverse A. For such matrix A, we have that WA_k is the kth standard unit vector in $\mathbb{Z}_q^n$, where A_k is the kth column of A. We see that

$$u_k = \Pi_{i=1}^{n} u_i^{WA_k} = \Pi_{i=1}^{n} \left(\Pi_{j=1}^{n} u_i^{W_{i,j} A_{j,k}} \right) = \Pi_{j=1}^{n} \mathsf{Com}(W'_j, \tilde{t}_j)^{A_{j,k}}$$
$$= \Pi_{j=1}^{n} \mathsf{Com}(W'_j A_{j,k}, \tilde{t}_j A_{j,k}) = \mathsf{Com}(W' A_k, \langle \tilde{\mathbf{t}}, A_k \rangle).$$

Thus, we can open $\mathbf{u}$ to a matrix M, where $M_k = W' A_k$ has been committed to using randomness $\langle \tilde{\mathbf{t}}, A_k \rangle$.

We expect M to be a permutation matrix. If it is not, we can find a witness breaking the binding property of the commitment scheme. We extract this witness in two different ways, depending on whether $M\mathbf{1} = \mathbf{1}$ or not.

If $M\mathbf{1} \neq \mathbf{1}$, let $\mathbf{w}'' = M\mathbf{1}$. We note that $\mathbf{w}'' \neq \mathbf{1}$ and that $\mathsf{Com}(\mathbf{1}, \bar{t}_j) = \Pi_{i=1}^{n} u_i = \mathsf{Com}(\mathbf{w}'', \tilde{\mathbf{t}} A)$, meaning that we have found a witness violating the binding property of the commitment scheme.

Now, assume that $M\mathbf{1} = \mathbf{1}$. Terelius & Wikström [14] prove that M is a permutation matrix if and only if $M\mathbf{1} = \mathbf{1}$ and $\Pi_{i=1}^{n} \langle M_i, \mathbf{x} \rangle = \Pi_{i=1}^{n} x_i$ for a vector $\mathbf{x} \in \mathbb{Z}_q^n$ of independent elements. This fact, along with the Schwartz-Zippel lemma and the assumptions that $M\mathbf{1} = \mathbf{1}$ and that M is not a permutation matrix, implies that there exists, with overwhelming probability, some $j \in \{1, ..., n\}$ such that $\Pi_{i=1}^{n} \langle M_{i,*}, W_j \rangle - \Pi_{i=1}^{n} W_{i,j} \neq 0$

(recall that $M_{i,*}$ is the ith row of M). Since this is true with overwhelming probability, we assume that it is true and rewind if it is not.

Now, let $\mathbf{w}'' = MW_j$. Note that $\Pi_{i=1}^n W'_{i,j} = \Pi_{i=1}^n W_{i,j}$ and that $\Pi_{i=1}^n W_{i,j} \neq \Pi_{i=1}^n w''_i$. The equality follows from the base statements, and the inequality follows from the Schwartz-Zippel lemma and the definition of $\mathbf{w}''$. Together, these facts imply that $\mathbf{w}'' \neq W'_j$.

We also see that $\mathsf{Com}(W'_j, \tilde{t}_j) = \Pi_{i=1}^n u_i^{W_{i,j}} = \mathsf{Com}(\mathbf{w}'', \langle \tilde{\mathbf{t}} A, W_j \rangle)$. Since $\mathbf{w}'' \neq W'_j$, this means that we have found a witness violating the binding property of the commitment scheme. We conclude that either M is a permutation matrix, or the binding property of the commitment scheme does not hold. We conclude further that we either violate the binding property of the commitment scheme, or we have that $\mathbf{w}'' = W_j$, meaning that $W'_j = MW_j$, for all $j \in \{0, ..., n\}$.

Extracting the randomness. We now show that we can extract $\mathbf{r} \in (\mathbb{Z}_q \times \mathbb{Z}_q \times \mathbb{Z}_q)^n$ such that $\mathbf{v}'$ is a re-encryption of $\mathbf{v}$, i.e. $v'_i = v_{\pi(i)} \cdot \mathsf{Enc}_{pk}(1; r_{\pi(i)})$. In the following, recall that $\mathbf{w}' = M\mathbf{w}$, $M_k = W'A_k$, and that WA_k is the kth standard unit vector in $\mathbb{Z}_q^n$. Thus, we get

$$\begin{aligned} v_k &= \Pi_{i=1}^n v_i^{WA_k} = \Pi_{j=1}^n (\Pi_{i=1}^n (v'_i)^{W'_{i,j}} \cdot \mathsf{Enc}_{pk}(1; -\mathbf{r}'))^{A_{j,k}} \\ &= \Pi_{i=1}^n (v'_i)^{\Sigma_{j=1}^n W'_{i,j} A_{j,k}} \cdot \mathsf{Enc}_{pk}(1; -\langle \mathbf{r}', A_k \rangle) \\ &= \Pi_{i=1}^n (v'_i)^{M_k} \cdot \mathsf{Enc}_{pk}(1; -\langle \mathbf{r}', A_k \rangle) = v'_{\pi^{-1}(k)} \cdot \mathsf{Enc}_{pk}(1; -\langle \mathbf{r}', A_k \rangle). \end{aligned}$$

This shows that $v'_{\pi^{-1}(k)} = v_k \cdot \mathsf{Enc}_{pk}(1; \langle \mathbf{r}', A_k \rangle)$, so $r_k = \langle \mathbf{r}', A_k \rangle$.

Special Honest-Verifier Zero-Knowledge The zero-knowledge simulator chooses the following values at random: $\hat{u}_i \in \mathbb{G}_1$ for $i = 1, ..., n$, $\mathbf{w}, \mathbf{b}', \hat{\mathbf{b}} \in \mathbb{Z}_q^n$, $b_1, b_2, b_3, \beta \in \mathbb{Z}_q$ and $\tilde{\mathbf{b}}, \mathbf{r}' \in \mathbb{Z}_q \times \mathbb{Z}_q \times \mathbb{Z}_q$. The simulator then computes a_1, a_2, a_3, a_4 and $\hat{a}_i$ for $i = 1, ..., n$ using the verification equations in step 5. This is a perfect simulation. To see that, consider the statistical distribution of the values in the real run and the simulated run:

- $\mathbf{w}$ is chosen at random from $\mathbb{Z}_q^n$ in both the simulated and the real run.
- The $\hat{u}_i$ are randomly distributed in $\mathbb{G}_1$ in both the real and the simulated run. It is obvious in the simulated run since the simulator samples the elements at random from $\mathbb{G}_1$. In the real run, we have $\hat{u}_i = \gamma^{\hat{t}_i} (\hat{u}_{i-1})^{w'_i}$, where the $\hat{t}_i$ are chosen at random from $\mathbb{Z}_q$. Thus, the $\hat{u}_i$ will be uniformly distributed in $\mathbb{G}_1$.
- β is chosen uniformly at random from $\mathbb{Z}_q$ in both runs.
- In the simulated run, $b_1, b_2, b_3, \tilde{\mathbf{b}}, \hat{\mathbf{b}}$ and $\mathbf{b}'$ are chosen uniformly at random from their respective domains. In the real run, the challenge β defines a bijection between $b_1, b_2, b_3, \tilde{\mathbf{b}}, \hat{\mathbf{b}}, \mathbf{b}'$ and $z_1, z_2, z_3, \tilde{\mathbf{z}}, \hat{\mathbf{z}}, \mathbf{z}'$ (given by the equations in Step 4 of Protocol 1). Since the latter values are chosen uniformly at random, the former values will be uniformly distributed as well.
- The above values determine the values of a_1, a_2, a_3, a_4 and $\hat{a}_i$ for $i = 1, ..., n$ by the verification equations in Step 5, in both runs.

4.3 Proof of Shuffle on the Public Board

A verifiable shuffle for the public board is given in Protocol 2. Note that it is very similar to the shuffle in Protocol 1. The difference is that on the public board, the shuffle is performed on the two last components of each ciphertext, rather than on the full ciphertext.

Let $\mathscr{R}^{shuf}_{ReRand}(pk, \mathbf{d}, \mathbf{d}')(\pi, \mathbf{r}')$, where π is a permutation on $\{1, \ldots, n\}$, be the relation which holds if $d'_i = \mathsf{ReRand}(d_{\pi(i)}; r'_{\pi(i)})$ for all $i \in \{1, \ldots, n\}$, where $\mathsf{ReRand}(d_i; r'_i) = (h^{r_{i,1}+r'_{i,1}} h_1^{r_{i,2}+r'_{i,2}}, m g_1^{r_{i,2}+r'_{i,2}})$ for $d_i = (h^{r_{i,1}} h_1^{r_{i,2}}, m g_1^{r_{i,2}})$ and random $\mathbf{r}, \mathbf{r}' \in (\mathbb{Z}_q \times \mathbb{Z}_q)^n$. Let $\mathscr{R}_\pi$ and $\mathscr{R}_{com}$ be as in Section 4.2.

Theorem 2. *Protocol 2 is a perfectly complete, 4-round special soundness, special honest-verifier zero-knowledge proof of knowledge of the relation* $\mathscr{R}_{com} \vee (\mathscr{R}_\pi \wedge \mathscr{R}^{shuf}_{ReRand})$.

The proof is very similar to the proof of Theorem 1 and will be omitted.

4.4 Applying the Fiat-Shamir Heuristic

We now describe how we can make the shuffle non-interactive, by applying the Fiat-Shamir heuristic [5]. The main idea is to replace the challenges sent by the verifier (in step 1 and 3) by a call to some hash function, making the challenges look random. This is straight-forward, but we do not want to run the argument twice, once for the public board and once for the private board. We want to have only one computation. It is easy to see that the interactive public board argument can be extracted from the interactive private board argument, but applying Fiat-Shamir is not straight-forward now, since different knowledge is available in the two cases.

The idea is to use a nested hash function for the private board argument, and then provide the inner hash value as part of the public board argument. This allows us to extract the public board argument from the private board argument by replacing the knowledge that is not present on the public board by their hash value. In order to ensure that no knowledge leaks, we actually commit to the hash of the private values, so that we can prove that the hash value does not contain any information about the private values. This is safe, since commitments are binding.

To obtain $\mathbf{w}$, we first hash the parts of the common input on the private board that is not part of the common input on the public board, i.e. the first three components of the CCE ciphertexts. We then commit to this hash, and hash the commitment along with the part of the common input that is also present on the public board. The challenge $\mathbf{w}$ is set to be this second hash value. The commitment is posted on the public board and opened on the private board.

The challenge β is obtained in a similar manner. We first hash the information on the private board that is not present on the public board, commit to this hash, post the commitment on the public board and then open the commitment on the private board. Further, the commitment is hashed along with the information on the private board that is also present on the public board. This hash is set to be the challenge value β.

Protocol 2 Interactive ZK-Proof of Shuffle on Public Board

Common Input: A public key pk, a matrix commitment $\mathbf{u}$, commitment parameters $\gamma, \gamma_1, ..., \gamma_n$ and vectors $\mathbf{d}, \mathbf{d}' \in (\mathbb{G}_2 \times \mathbb{G}_1)^n$.

Private Input: Permutation matrix $M \in \mathbb{Z}_q^{n \times n}$ and randomness $\mathbf{t} \in \mathbb{Z}_q^n$ such that $\mathbf{u} = \mathsf{Com}(M, \mathbf{t})$. Randomness $\mathbf{r} \in (\mathbb{Z}_q \times \mathbb{Z}_q)^n$ such that $d'_i = \mathsf{ReRand}(d_{\pi(i)}, r_{\pi(i)})$ for $i = 1, ..., n$.

1: $\mathcal{V}$ chooses a random $\mathbf{w} \in \mathbb{Z}_q^n$ and sends $\mathbf{w}$ to $\mathcal{P}$.

2: $\mathcal{P}$ computes $\mathbf{w}' = (w'_1, ..., w'_n) = M\mathbf{w}$, and randomly chooses $\hat{\mathbf{t}} = (\hat{t}_1, \ldots, \hat{t}_n)$, $\hat{\mathbf{z}} = (\hat{z}_1, ..., \hat{z}_n)$, $\mathbf{z}' = (z'_1, ..., z'_n) \in \mathbb{Z}_q^n, z_1, z_2, z_3 \in \mathbb{Z}_q$ and $\tilde{\mathbf{z}} = (\tilde{z}_1, \tilde{z}_2) \in \mathbb{Z}_q^2$. $\mathcal{P}$ defines

$$\bar{t} = \langle \mathbf{1}, \mathbf{t} \rangle,\ \tilde{t} = \langle \mathbf{t}, \mathbf{w} \rangle,\ \hat{t} = \hat{t}_n + \sum_{i=1}^{n-1} \left(\hat{t}_i \prod_{j=i+1}^{n} w'_j \right) \text{ and}$$

$$\mathbf{r}' = \left(\sum_{i=1}^{n} r_{i,1} w_i, \sum_{i=1}^{n} r_{i,2} w_i \right),$$

and sends the following elements to $\mathcal{V}$ (for $i = 1, \ldots, n$):

$$\hat{u}_0 = \gamma_1 \qquad \hat{u}_i = \gamma^{\hat{t}_i} (\hat{u}_{i-1})^{w'_i} \qquad a_1 = \gamma^{z_1} \qquad a_2 = \gamma^{z_2}$$
$$a_3 = \gamma^{z_3} \Pi_{i=1}^n \gamma_i^{z'_i} \qquad a_4 = (h^{\tilde{z}_1} h_1^{\tilde{z}_2}, g_1^{\tilde{z}_2}) \Pi_{i=1}^n (d'_i)^{z'_i}$$
$$\hat{a}_i = \gamma^{\hat{z}_i} (\hat{u}_{i-1})^{z'_i}.$$

3: $\mathcal{V}$ chooses a random challenge $\beta \in \mathbb{Z}_q$ and sends β to $\mathcal{P}$.

4: For $i \in \{1, \ldots, n\}$, $\mathcal{P}$ responds with

$$b_1 = z_1 + \beta \cdot \bar{t} \qquad b_2 = z_2 + \beta \cdot \hat{t} \qquad b_3 = z_3 + \beta \cdot \tilde{t}$$
$$\tilde{\mathbf{b}} = \tilde{\mathbf{z}} - \beta \cdot \mathbf{r}' \qquad \hat{b}_i = \hat{z}_i + \beta \cdot \hat{t}_i \qquad b'_i = z'_i + \beta \cdot w'_i.$$

5: $\mathcal{V}$ accepts if and only if, for $i \in \{1, \ldots, n\}$

$$a_1 = (\Pi_{i=1}^n u_i / \Pi_{i=1}^n \gamma_i)^{-\beta} \cdot \gamma^{b_1} \quad a_2 = (\hat{u}_n / \gamma_1^{\Pi_{i=1}^n w_i})^{-\beta} \cdot \gamma^{b_2}$$
$$a_3 = (\Pi_{i=1}^n u_i^{w_i})^{-\beta} \cdot \gamma^{b_3} \cdot \Pi_{i=1}^n \gamma_i^{b'_i}$$
$$a_4 = (\Pi_{i=1}^n d_i^{w_i})^{-\beta} \cdot (h^{\tilde{b}_1} h_1^{\tilde{b}_2}, g_1^{\tilde{b}_2}) \cdot \Pi_{i=1}^n (d'_i)^{b'_i}$$
$$\hat{a}_i = (\hat{u}_i)^{-\beta} \cdot \gamma^{\hat{b}_i} \cdot (\hat{u}_{i-1})^{b'_i}$$

5 Machine Checked Proof

Having given a paper proof of the mixnet we now turn our attention to the machine checked proof. The obvious approach would be to codify the above paper proof in an interactive theorem prover. However, codifying such proofs is a complex process, so instead we reuse previous work. The idea is that our variant of the mixnet has a machine checked proof. The gap is that the mixnet is not proved for our particular encryption scheme. But the existing proof applies to a large class of encryption schemes. We need only prove that our scheme is in this class, after which we know that the general results also apply to our concrete mixnet.

For the machine checked proof we will make use of the interactive theorem prover Coq. Our work expands upon Haines *et al.* [9]; who demonstrated how interactive theorem provers and code extraction can be used to gain much higher confidence in the outcome of elections; they achieved this by using the interactive theorem prover Coq and its code extraction facility to produce verifiers, for verifiable voting schemes, with the verifiers proven to be cryptographically correct. They also showed that it was possible to verify the correctness (completeness, soundness and zero-knowledge) of a proof of correct shuffle. Their work was subsequently expanded upon by [8] who removed a number of limitations in the original work and expanded the result. Specifically they proved that for any encryption scheme that falls within a class, which they formally defined, it can be securely mixed in the optimised variant of Wikström's mixnet. We exploit this result by proving that PPATC falls within this class and hence can be verifiably mixed by Wikström's mixnet. Note that the mixnet generated in the Coq code is equivalent to Protocol 1.

In the rest of this section we will present our work in standard notation. Interested readers can find the Coq code at `https://github.com/gerlion/secure-e-voting-with-coq`. We begin by proving that the ciphertext space is a group. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ represent the elements of the two groups of the bilinear pairing both of which are of prime order p. We let the set S of the ciphertext space equal $\mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1$. All operations are performed pairwise and the group axioms are satisfied trivially.

We then show that the ciphertext group is isomorphic to a vector space over the field of integers modulo p. This follows directly from the fact that two groups of the same order are themselves isomorphic to vector spaces over the field of integers modulo p. We are now ready to define the encryption scheme. Beyond the groups already mentioned we denote the field of integers modulo p as $\mathbb{F}$.

Let PPATC denote the encryption scheme.

Key generation space := $\mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_2 \times \mathbb{F} \times \mathbb{F}$.
Public key space := $\mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_2 \times \mathbb{G}_1 \times \mathbb{G}_1$.
Secret key space := $\mathbb{F} \times \mathbb{F}$.
Message space := $\mathbb{G}_1$.
Randomness space := $\mathbb{F} \times \mathbb{F} \times \mathbb{F}$.
Key generation := On input (g, h, h_1, x_1, x_2) from key generation space output public key $(g, h, h_1, g^{x_1}, g^{x_2})$ and secret key (x_1, x_2)
Encryption := On input public key (g, h, h_1, y_1, y_2), message m, and randomness (r_1, r_2, r_3) and output ciphertext $(g^{r_1}, g^{r_2}, y_2^{r_2} g^{r_3}, h^{r_3} h_1^{r_1}, y_1^{r_1} m)$.

Decryption := Given secret key (x_1, x_2) and ciphertext $(c_1, c_2, c_3, c_4, c_5)$ and return $c_5/c_1^{x_1}$

To show that the encryption scheme can be correctly mixed we need to prove three theorems which are stated below. We will also require the vector space properties for the spaces defined above, see the Coq code for a formal definition of these properties.

```
Lemma correct : forall (kgr : KGR)(m : M)(r : Ring.F),
let (pk,sk) := keygen kgr in
dec sk (enc pk m r) = m.
```

Theorem 3. *Correctness:* $\forall kgr \in$ *Key generation space*, $m \in$ *Message space*, $r \in$ *Randomness space*, $(pk, sk) =$ *Key generation*(kgr) *Decryption*$(sk$ *Encryption*$(pk\ m\ r)) = m$.

The correctness of PPATC follows directly from the correctness of ElGamal.

```
Lemma homomorphism : forall (pk : PK)(m m' : M)(r r' : Ring.F),
C.Gdot (enc pk m' r')(enc pk m r) =
enc pk (Mop m m') (Ring.Fadd r r').
```

Theorem 4. *Homomorphism:* $\forall pk \in$ *Public key space*, $m\ m' \in$ *Message space*, $r\ r' \in$ *Randomness space*, *Encryption*$(pk\ m\ r) \times$ *Encryption*$(pk\ m'\ r') =$ *Encryption*$(pk\ (m \cdot m')\ (r * r'))$

The homomorphic property of PPATC follows from the homomorphic properties of ElGamal and Abe *et al.*'s commitments.

```
Lemma encOfOnePrec : forall (pk : PK)(a : Ring.F)(b: F),
(VS.op (enc pk Mzero a) b) = enc pk Mzero (MVS.op3 a b).
```

Theorem 5. *Encryption of one preserved:* $\forall pk \in$ *Public key space*, $r\ r' \in$ *Randomness space*, *Encryption*$(pk\ 1\ a)^b =$ *Encryption*$(pk\ 1\ (a * b))$

To see that this property holds, first consider a PPATC ciphertext encrypting zero: $(g^{r_1}, g^{r_2}, y_2^{r_2} g^{r_3}, h^{r_3} h_1^{r_1}, y_1^{r_1})$. Now observe that raising it to any power a is an encryption of one with randomness $(r_1 a, r_2 a, r_3 a)$, $(g^{r_1}, g^{r_2}, y_2^{r_2} g^{r_3}, h^{r_3} h_1^{r_1}, y_1^{r_1})^a = (g^{r_1 a}, g^{r_2 a}, y_2^{r_2 a} g^{r_3 a}, h^{r_3 a} h_1^{r_1 a}, y_1^{r_1 a})$.

Conclusion This suffices for a proof that the PPATC scheme can be safely mixed by the optimised variant of the Wikström's mixnet.

Readers will have noted that we proved the scheme for any pair of groups with the same prime order. Technically, we didn't even require that there exists a billinear pairing between them, though this would be required to get the verifiable component of the Abe *et al.* commitments to work. The current work could be extracted into OCaml code and appropriate groups provided to check election transcripts. However, further work is ongoing in Coq to allow these groups to be instantiated within Coq.

6 Conclusion

We have given a paper proof for a variant of the optimised Wikström's mixnet for the PPATC encryption scheme. This is a useful result for anyone wanting to build an efficient e-voting scheme with everlasting privacy which can handle arbitrary ballots. In addition we provide a machine checked proof of the mixnet.

Acknowledgments. This work was supported by the Luxembourg National Research Fund (FNR) and the Research Council of Norway for the joint project SURCVS.

References

1. Masayuki Abe, Kristiyan Haralambiev, and Miyako Ohkubo. Group to group commitments do not shrink. In *EUROCRYPT*, volume 7237 of *LNCS*, pages 301–317. Springer, 2012.
2. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.
3. Ronald Cramer, Matthew K. Franklin, Berry Schoenmakers, and Moti Yung. Multi-autority secret-ballot elections with linear work. In *EUROCRYPT*, volume 1070 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 1996.
4. Edouard Cuvelier, Olivier Pereira, and Thomas Peters. Election verifiability or ballot privacy: Do we need to choose? In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 481–498. Springer, Heidelberg, September 2013.
5. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.
6. Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 1997.
7. Rolf Haenni, Philipp Locher, Reto E. Koenig, and Eric Dubuis. Pseudo-code algorithms for verifiable re-encryption mix-nets. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y. A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *FC 2017 Workshops*, volume 10323 of *LNCS*, pages 370–384. Springer, Heidelberg, April 2017.
8. Thomas Haines, Rajeev Goré, and Bhavesh Sharma. Did you mix me? Formally verifying verifiable mix nets in voting. In *2021 IEEE Symposium on Security and Privacy, SP 2021, San Jose, CA, USA, May 23-27, 2021*. IEEE, 2021.
9. Thomas Haines, Rajeev Goré, and Mukesh Tiwari. Verified verifiers for verifying elections. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 685–702. ACM, 2019.
10. Thomas Haines and Clémentine Gritti. Improvements in everlasting privacy: Efficient and secure zero knowledge proofs. In *E-VOTE-ID*, volume 11759 of *Lecture Notes in Computer Science*, pages 116–133. Springer, 2019.
11. Tal Moran and Moni Naor. Split-ballot voting: Everlasting privacy with distributed trust. *ACM Trans. Inf. Syst. Secur.*, 13(2):16:1–16:43, 2010.
12. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238. Springer, 1999.
13. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, volume 576 of *LNCS*, pages 129–140. Springer, 1991.
14. Björn Terelius and Douglas Wikström. Proofs of restricted shuffles. In Daniel J. Bernstein and Tanja Lange, editors, *AFRICACRYPT 10*, volume 6055 of *LNCS*, pages 100–113. Springer, Heidelberg, May 2010.

Paper II

Machine-Checked Proofs of Privacy Against Malicious Boards for Selene & Co

Constantin Cătălin Drăgan, François Dupressoir, Ehsan Estaji, Kristian Gjøsteen, Thomas Haines, Peter Y. A. Ryan, Peter B. Rønne and Morten Rotvold Solberg

Extended version of a paper published at the 35th IEEE Computer Security Foundations Symposium, CSF 2022. The extended version will appear in The Journal of Computer Security. The conference version is available at: eprint.iacr.org/2022/1182.

Machine-Checked Proofs of Privacy Against Malicious Boards for Selene & Co

Constantin Cătălin Drăgan[1], François Dupressoir[2], Ehsan Estaji[3], Kristian Gjøsteen[4], Thomas Haines[5], Peter Y.A. Ryan[3], Peter B. Rønne[6], and Morten Rotvold Solberg[4]

[1] University of Surrey, Guildford, United Kingdom c.dragan@surrey.ac.uk
[2] University of Bristol, Bristol, United Kingdom f.dupressoir@bristol.ac.uk
[3] University of Luxembourg, Esch-sur-Alzette, Luxembourg {ehsan.estaji,peter.ryan}@uni.lu
[4] Norwegian University of Science and Technology, Trondheim, Norway {kristian.gjosteen,mosolb}@ntnu.no
[5] Australian National University, Canberra, Australia thomas.haines@anu.edu.au
[6] CNRS, LORIA, Université de Lorraine, Nancy, France peter.roenne@loria.fr

Abstract. Privacy is a notoriously difficult property to achieve in complicated systems and especially in electronic voting schemes. Moreover, electronic voting schemes is a class of systems that require very high assurance. The literature contains a number of ballot privacy definitions along with security proofs for common systems. Some machine-checked security proofs have also appeared. We define a new ballot privacy notion that captures a larger class of voting schemes. This notion improves on the state of the art by taking into account that verification in many schemes will happen or must happen after the tally has been published, not before as in previous definitions.
As a case study we give a machine-checked proof of privacy for Selene, which is a remote electronic voting scheme which offers an attractive mix of security properties and usability. Prior to our work, the computational privacy of Selene has never been formally verified. Finally, we also prove that MiniVoting and Belenios satisfies our definition.

1 Introduction

Confidence in the validity of the outcome and privacy of the votes is supremely important for elections. We build confidence in elections by using carefully selected methods, routines, and election officers. In particular, extensive use of various forms of auditing helps build confidence.

For the analysis of a voting mechanism, we need to know what security means and why the mechanism is secure. The former requires a so-called *security notion*, while the latter is best achieved with a *security proof*, a mathematical argument for why the mechanism satisfies the security notion.

It is easy to have an intuitive notion of what security should mean, but defining privacy for voting mechanisms is non-trivial, as shown by the many attempts to do so in the literature. (Bernhard *et al.* [5] has a good overview of privacy definitions.)

One problem is that many security notions are highly specialised for a particular class of voting mechanisms. But there are a large number of cryptographic voting mechanisms and they exhibit great variety in their form and shape. Defining security notions that usefully and uniformly capture a larger class of voting mechanisms is a good thing in principle, but it is also an essential task if existing security notions do not cover the

voting mechanism of interest. ***We need security notions that capture a larger class of voting mechanisms.***

Once we have a security notion, we return to the problem of creating and auditing a security proof. *Machine-checked* proofs is a good way to increase assurance for security proofs. One system designed for handling security proofs is EasyCrypt. A large set of security notions, cryptographic constructions and corresponding security proofs have been written in EasyCrypt, and the system has seen extensive use. ***We need machine-checked security proofs for voting mechanisms.***

Selene [21] is a voting mechanism designed to provide a simple method for verification, while at the same time mitigating the threat of coercion. The key idea in Selene is that every voter is assigned a personal tracking number, and when the election period is over and everyone has cast their vote, the tracking numbers and the votes are published in plaintext on a web bulletin board. This gives the voters a direct and easy to understand way of verifying that their vote was correctly included in the tally. The key innovation in Selene is how to give the voter a tracking number so that no single party know what tracking number was given to the voter (other than the voter themselves) and the voter can plausibly lie about which tracker they received. The first property is achieved by mixing the trackers as part of the setup.

There is a danger of coercion here, namely that a coercer requires a voter to hand over her tracking number, so that the coercer himself can verify that the voter fulfilled his demands. However, the coercer has a limited window of opportunity, because he needs the coerced voter to hand over her tracker before the trackers and the votes are published. Otherwise, the coerced voter could simply find a vote corresponding to the coercer's demand, and give the coercer the tracker next to this vote. This observation is used in Selene to counter the threat of coercion: the voters first learn their tracking numbers after the trackers and votes are published on the web bulletin board.

Abstractly, Selene has a different order of operations than many existing voting mechanisms. Schemes like MiniVoting (which in some sense models a large class of voting mechanisms including variants of Helios) do voter verification before tallying. For Selene voter verification must happen after tallying, since the personal tracking numbers do not appear until after tallying. This means that Selene does not fit very well into existing security notions for voting mechanisms. This is also true for a number of other systems where voters or their delegates first can (or choose to) verify after tally. The Selene verification mechanism has also been trialled with a commercial partner [22]. ***We need a high-assurance security proof for Selene.***

1.1 Our Contribution

This paper extends upon the work by Drăgan *et al.* [14], where we define the new security notion *delay-use malicious-ballotbox ballot privacy* (du–mb–BPRIV) to capture the security of schemes that delay the use of verification information to a post-tallying verification step. This is necessary for tracker based schemes (like Selene [21], Electryo [20] and sElect [19]), for in-person voting schemes where the verification is first done later at home, but further it also applies to e-voting schemes where the verification step is not made mandatory before

tallying, or often happens after tally, e.g. when verification is delegatable. To construct our definition, we build upon a recent ballot privacy definition called mb–BPRIV [13].

We model our new security notion in the proof assistant EasyCrypt [1] (`https://github.com/easycrypt/easycrypt`), and to validate our security notion, we also model the Labelled-MiniVoting scheme [10] and Belenios [12] and verify that these schemes satisfy ballot privacy both under our new definition and under the original mb–BPRIV. Furthermore, we model the Selene voting system, and prove that this scheme satisfies ballot privacy under our new security definition. The EasyCrypt code is available at `https://github.com/mortensol/du-mb-bpriv`.

Here, we extend the original work by investigating the relation between our ballot privacy definition and the notion of individual verifiability, as well as the corresponding relation for the mb–BPRIV definition (Sec. 6). Furthermore, we highlight a few lessons that we learned during our effort to formalise the security definitions and proofs in EasyCrypt (Sec. 7), and which we believe will be useful for future efforts on formalising security notions and security proofs in EasyCrypt. In particular, we discuss the following:

- Lesson 1: how to properly restrict adversarial access to an oracle's internal state, to avoid vacuously true results due to false axioms;
- Lesson 2: dealing with random oracles in EasyCrypt;
- Lesson 3: how small changes in definitions can lead to large changes in proofs; and
- Lesson 4: why it is necessary to split the voters' internal states into two parts when formalising voting systems (and related proofs) where voter verification happens after the tally has been computed, for security definitions using recovery algorithms.

1.2 Related Work

Many authors have tried to capture the notion of ballot privacy using standard cryptographic games. Bernhard *et al.* [5] gives a good general overview of such notions. We give an overview of the history leading up to the recent definition of mb–BPRIV in Section 2.7, directly preceding our new security notion.

The need for assurance with respect to voting systems makes cryptographic voting schemes a natural target for formalized security proofs, either through symbolic models and automatic verification or via proof assistants. While symbolic models have historically yielded good insights into the analysis of cryptographic protocols, see e.g. [8, 25] for symbolic analysis of Selene, we prefer a cryptographic analysis.

EasyCrypt [1] is a proof assistant focused on formalizing computational security proofs in the style of Shoup's *Sequences of Games* [24]. EasyCrypt supports constructive proofs of concrete security—leaving the complexity analysis of the constructed reduction to be done by hand. For simplicity in the rest of this paper, we discuss asymptotic notions. The formalized proof is concrete.

Cortier *et al.* use EasyCrypt to prove that Helios is BPRIV-secure [10], and that Belenios is BPRIV-secure and verifiable [11]. Our proof builds upon their framework—we in fact prove that Labelled MiniVoting and Belenios meet our new privacy definition, and further formalize their security in mb–BPRIV.

2 Background

In this section, we first introduce some basic cryptographic models, primitives and algorithms that make up a voting system, before we move on to describe earlier definitions of ballot privacy.

2.1 Random Oracle Model

In our analysis, we model hash functions as random oracles [2]. That is, to compute the value of a hash function at a point x, any party can make a call to an oracle $\mathcal{O}$, implementing a random function from some domain D to some range R. The oracle $\mathcal{O}$ maintains an initially empty table T, and whenever someone calls $\mathcal{O}(x)$ for some $x \in D$, the oracle $\mathcal{O}$ checks if there is an entry (x, y) in T for some $y \in R$. If so, it returns y; if not, $\mathcal{O}$ randomly generates a $y' \in R$, adds (x, y') to T and outputs y'. We will use the Random oracle model implicitly below when modelling the non-interactive zero-knowledge proofs that help ensure privacy in e-voting.

2.2 Public Key Encryption

Public key encryption systems are often used in voting protocols, to help protect the privacy of the votes, and possibly other things. In Selene, for instance, both the votes and the voter's personal tracking numbers are encrypted using some form of public key encryption. Formally, a *public key encryption system* (PKE) is defined as follows:

Definition 1. *A public key encryption scheme (PKE) is a triple of algorithms* $\mathsf{E} = (\mathsf{kgen}, \mathsf{enc}, \mathsf{dec})$*; where*

- kgen *is a probabilistic algorithm that takes as input a security parameter* λ *and outputs a key pair* $(\mathsf{pk}, \mathsf{sk})$*,*
- enc *is a probabilistic algorithm that on input a public key* pk *and a plaintext* m *outputs a ciphertext* c*,*
- dec *is a deterministic algorithm that on input a secret key* sk *and a ciphertext* c *outputs either a plaintext* m *or a special error symbol* $\perp$ *indicating that something went wrong.*

We require that decryption "undoes" encryption, i.e. for any key pair $(\mathsf{pk}, \mathsf{sk})$ *output by* kgen*, and any plaintext* m*, we have that* $\mathsf{dec}(\mathsf{sk}, \mathsf{enc}(\mathsf{pk}, m)) = m$*.*

A *labelled public key encryption scheme (LPKE)* extends the notion of a "regular" PKE by adding some additional, non-malleable data called a *label* [23]. One important property for a labelled PKE is that decrypting a ciphertext using a different label than the one used for encryption, should not reveal anything about the original plaintext. Formally, a labelled PKE is defined similarly to how we define a PKE in Definition 1, but a label ℓ is given as additional input in the encryption and decryption algorithms.

The security of the schemes we analyze rely on a security notion for labelled public key encryption called *indistinguishability under chosen ciphertext attack with one parallel*

decryption query (IND–1–CCA) [3]. Informally, this notion captures that any efficient adversary is unable to distinguish between encryptions of two messages of the same length, when given access to a batch decryption oracle that can be called once.

A similar security notion, namely poly–IND–1–CCA, allows the adversary to make up to n challenge queries, for some polynomially bounded integer n. This notion is formalized in Figure 1, where an adversary $\mathcal{B}$ is given access to an encryption oracle $\mathcal{O}_{\mathsf{enc}}$ and a decryption oracle $\mathcal{O}_{\mathsf{dec}}$. The adversary can make n challenge queries to $\mathcal{O}_{\mathsf{enc}}$, who encrypts one of two plaintexts, depending on the bit β. The adversary can query $\mathcal{O}_{\mathsf{dec}}$ at most once, and the oracle then decrypts a list of ciphertexts. For any ciphertexts created by the encryption oracle, the decryption oracle returns $\perp$.

The advantage of a poly–IND–1–CCA adversary $\mathcal{B}$ against a labelled public key encryption scheme E is defined as

$$\mathsf{Adv}^{\mathsf{poly-ind1cca}}_{\mathcal{B},\mathsf{E},n}(\lambda) = \left|\Pr\left[\mathsf{Exp}^{\mathsf{poly-ind1cca},0}_{\mathcal{B},\mathsf{E},n}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{poly-ind1cca},1}_{\mathcal{B},\mathsf{E},n}(\lambda) = 1\right]\right|,$$

and we say that the labelled PKE E is n-challenge poly–IND–1–CCA-secure if the advantage defined above is negligible in λ for all efficient adversaries $\mathcal{B}$.

As noted in [10], if $n = 1$, poly–IND–1–CCA security is essentially reduced to IND–1–CCA security. Indeed, it is possible to prove, through a hybrid argument, that a labelled PKE is IND–1–CCA secure if and only if it is poly–IND–1–CCA secure. This fact was also verified in EASYCRYPT [10], and we were able to reuse this framework in our formalization of the ballot privacy of Selene.

$\mathsf{Exp}^{\mathsf{poly-ind1cca},\beta}_{\mathcal{B},\mathsf{E},n}(\lambda)$

1 : $\mathsf{encL} \leftarrow [\,]$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{kgen}(\lambda)$
3 : $\beta' \leftarrow \mathcal{B}^{\mathcal{O}_{\mathsf{enc}},\mathcal{O}_{\mathsf{dec}}}(\mathsf{pk})$
4 : **return** β'

$\mathcal{O}_{\mathsf{enc}}(\ell, m_0, m_1)$

1 : $c \leftarrow \perp$
2 : **if** $|\mathsf{encL}| < n$ **then**
3 : $\quad c \leftarrow \mathsf{enc}(\mathsf{pk}, \ell, m_\beta)$
4 : $\quad \mathsf{encL} \leftarrow \mathsf{encL} + [(c, \ell)]$
5 : **return** c

$\mathcal{O}_{\mathsf{dec}}(cL)$

1 : $mL \leftarrow [\,]$
2 : **for** $(c, \ell) \in cL$ **do**
3 : $\quad$ **if** $(c, \ell) \notin \mathsf{encL}$ **then**
4 : $\quad\quad mL \leftarrow mL + [\mathsf{dec}(\mathsf{sk}, \ell, c)]$
5 : $\quad$ **else** $mL \leftarrow mL + [\perp]$
6 : **return** mL

Fig. 1. Security experiment for poly–IND–1–CCA [10]

2.3 Commitment Protocols

A *commitment protocol* allows a prover P to commit to some value b, and send the commitment to a verifier V. The verifier can ask the prover to open the commitment at some later point and verify the output value. Two important properties of a commitment protocol is that it should be *binding* and *hiding*. The first property informally means that once P has committed to a value, he should not be able to open the commitment to another value. The second property informally means that before the commitment is opened, V should not be able to determine what was committed to.

More formally, a commitment protocol is defined as follows:

Definition 2 (Commitment protocol). *A* commitment protocol *is a triple of algorithms* $\mathsf{CP} = (\mathsf{gen}, \mathsf{commit}, \mathsf{open})$*, where*

- gen *takes as input a security parameter* λ *and returns a pair* (upk, usk) *of user public and secret keys,*
- commit *takes as input a user public key* upk *and a value we want to commit to, and returns a commitment* ct *and an opening key* d*,*
- open *takes as input a commitment and an opening key and returns the committed value.*

Commitments are an important part of the coercion mitigation strategy in Selene, where the election officials make commitments to personal tracking numbers for each voter. These commitments are opened by the voters at the end of the election, allowing the voters to verify that their vote was included in the tally, without being able to hand over their tracker to a coercer before all votes and trackers are published. For coercion-resistance, Selene actually employs trapdoor commitments and the voters have secret trapdoor keys that allow them to open the commitment to a tracker satisfying the coercer.

2.4 Proof Systems

We now describe proof systems which are used in Selene to ensure that various operations are performed correctly. We say that a binary relation $\mathcal{R}$ is a subset $\mathcal{R} \subseteq X \times W$, where X is a set of *statements* and W is a set of *witnesses*. A *proof system* for the relation $\mathcal{R}$ is a pair of efficient algorithms (P, V), where P is called the *prover* and V is called the *verifier*. The prover and verifier work on a common input $x \in X$, and the prover has some additional input $w \in W$. In a *non-interactive proof system*, P uses his input to compute a proof Π. He sends the proof to V, who, on input (x, Π) produces a verification output in $\{0, 1\}$.

A proof system is said to be *complete* if the prover can produce a valid proof whenever the statement is true. More formally, for any $(x, w) \in \mathcal{R}$, if Π is a proof output by $\mathsf{P}(x, w)$, then $\mathsf{V}(x, \Pi)$ outputs 1 with probability 1.

A proof system is *sound* if a prover is unable to convince a verifier that a false statement is true.

A proof system is *zero-knowledge* if the proof leaks no information beyond the fact that the relation holds. More formally, we demand the existence of an efficient algorithm Sim, called the *simulator*, that produces valid-looking proofs for a statement $x \in X$ without access to the witness w. Formally, we consider a zero-knowledge adversary $\mathcal{B}$ in the following experiments:

$\underline{\mathsf{Exp}^{zk,0}_{\mathcal{B},\mathsf{P},\mathcal{R}}(\lambda)}$

1 : $(x, w, \mathsf{state}) \leftarrow \mathcal{B}(\lambda)$
2 : $\Pi \leftarrow \bot$
3 : **if** $(\mathcal{R}(x, w))$ **then**
4 : $\quad \Pi \leftarrow \mathsf{P}(x, w)$
5 : $\beta' \leftarrow \mathcal{B}(\mathsf{state}, \Pi)$
6 : **return** β'

$\underline{\mathsf{Exp}^{zk,1}_{\mathcal{B},\mathsf{Sim},\mathcal{R}}(\lambda)}$

1 : $(x, w, \mathsf{state}) \leftarrow \mathcal{B}(\lambda)$
2 : $\Pi \leftarrow \bot$
3 : **if** $(\mathcal{R}(x, w))$ **then**
4 : $\quad \Pi \leftarrow \mathsf{Sim}(x)$
5 : $\beta' \leftarrow \mathcal{B}(\mathsf{state}, \Pi)$
6 : **return** β'

The advantage of the zero-knowledge adversary $\mathcal{B}$ over the proof system (P, V) and simulator Sim is defined as

$$\mathsf{Adv}^{zk}_{\mathcal{B},\mathsf{P},\mathsf{Sim},\mathcal{R}}(\lambda) = \left|\Pr\left[\mathsf{Exp}^{zk,0}_{\mathcal{B},\mathsf{P},\mathcal{R}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{zk,1}_{\mathcal{B},\mathsf{Sim},\mathcal{R}}(\lambda) = 1\right]\right|,$$

and we say that a proof system is zero-knowledge if, for any adversary $\mathcal{B}$, there exists a simulator Sim such that the advantage defined above is negligible.

2.5 Voting Systems

We define a *voting system* as being built upon a tuple of algorithms

$$\mathcal{V} = (\mathsf{Setup}, \mathsf{Register}, \mathsf{Vote}, \mathsf{ValidBoard}, \mathsf{Tally}, \mathsf{VerifyVote}, \mathsf{VerifyTally}, \mathsf{Publish}),$$

where the different algorithms informally work as follows:

- $\mathsf{Setup}(1^\lambda)$: Returns a pair $(\mathsf{pd}, \mathsf{sd})$ of public and secret data, typically including a public encryption key and a secret decryption key, respectively, but this data might also contain other things.
- $\mathsf{Register}(id, \mathsf{pd}, \mathsf{sd})$: Takes as input a user identity and some public and secret data and returns a public credential pc and a secret credential sc for that user.
- $\mathsf{Vote}(\mathsf{pd}, pc, sc, v)$: Takes as input some public data, a user's public and secret credentials, and a vote, and returns the user's public credential, a ciphertext encrypting the vote and a state that the voter later can use for verification.
- $\mathsf{ValidBoard}(\mathsf{BB}, \mathsf{pd})$: Checks the validity of the ballot box BB.
- $\mathsf{Tally}(\mathsf{BB}, \mathsf{pd}, \mathsf{sd})$: Computes the result r of the election, along with a proof Π of correct tallying.
- $\mathsf{VerifyVote}(id, \mathsf{state}, \mathsf{BB}, pc, sc)$: Run by a voter to check whether or not her ballot was included in the tally.
- $\mathsf{VerifyTally}((\mathsf{pd}, pbb, r), \Pi)$: Checks that Π is a valid proof of correct tally, with respect to the result r and the public part pbb of the ballot box BB.
- $\mathsf{Publish}(\mathsf{BB})$: Returns a public part pbb of the ballot box BB.

2.6 Voting Friendly Relations

In the voting systems we analyze in this paper, proof systems are used to compute and validate proofs of correct tally. In our analysis and EASYCRYPT formalization, we keep

the relation $\mathcal{R}$ abstract, and thus, we need to ensure that the relation is compatible with the result of the election. For this, we adopt the notion of *voting friendly relations*, as defined by Cortier *et al.* [10] and generalize it a bit, so that it also accommodates schemes like Selene. Very informally, a relationship is voting friendly if for any adversarially chosen bulletin board it is possible to find a corresponding tally such that the pair (bulletin board and tally) are in the language.

The relation being compatible with the result of the election means that if $\mathcal{V}$ is a voting system, $(\mathsf{pd}, \mathsf{sd})$ is the public and secret data generated by the Setup algorithm, the result r of the election corresponds to the tally of the votes obtained by decrypting the ciphertexts in the ballot box BB, and if pbb is the public part of BB, then the relation $\mathcal{R}((\mathsf{pk}, pbb, r), (\mathsf{sk}, \mathsf{BB}))$ holds. In other words, it is possible to prove that r is the correct result. More formally, a voting friendly relation is defined as follows:

Definition 3 (Voting friendly relation [10]). *Let $\mathcal{V}$ be a voting system and let $\Sigma_{\mathcal{R}}$ be a proof system for some relation $\mathcal{R}$. We say that $\mathcal{R}$ is a voting friendly relation if, for any efficient adversary $\mathcal{B}$, the following experiment returns 1 with negligible probability:*

$\mathsf{Exp}^{\mathsf{vfr}}_{\mathcal{B},\mathcal{V},\Sigma_{\mathcal{R}}}(\lambda)$

1: $(\mathsf{pd}, \mathsf{sd}) \leftarrow \mathcal{V}.\mathsf{Setup}$
2: $\mathsf{BB} \leftarrow \mathcal{B}(\mathsf{pd})$
3: $dbb \leftarrow \mathsf{dec}^*(\mathsf{sd}, \mathsf{BB})$
4: $r \leftarrow \rho(dbb)$
5: $pbb \leftarrow \mathcal{V}.\mathsf{Publish}(\mathsf{BB})$
6: **return** $\neg\mathcal{R}((\mathsf{pd}, pbb, r), (\mathsf{sd}, \mathsf{BB}))$

In the above experiment, the algorithm dec^* decrypts the ballot box BB line by line using the secret data sd, and returns a list $[(a_1, v_1), \ldots, (a_n, v_n)]$ of votes v_i and some additional information a_i, e.g. voter identities as in MiniVoting or tracking numbers as in Selene. We say that ρ is a counting function that takes in a list of the form described above, and returns a result.

2.7 Early Definitions of Ballot Privacy

In 2015, Bernhard *et al.* [5] conducted a survey of existing game-based ballot privacy definitions and found that they were all unsatisfactory. Some of the definitions were too weak, declaring protocols that intuitively did not have ballot privacy to be secure. Some definitions were too strong, making any voting protocol with even minimal verifiability impossible to prove private. Finally, some definitions were too limited, restricting the class of captured voting protocols and privacy breaches too much.

Based on this survey, Bernhard *et al.* [5] proposed a new definition of ballot privacy which was named BPRIV. The BPRIV definition captures the idea that a voting system should not leak any information about the votes that are cast, beyond what can be derived from the result of the election. This is formalized by having an adversary attempting to distinguish between two worlds. In one world, the adversary gets to see a ballot box

containing real ballots submitted by honest voters, as well as any ballots the adversary has submitted on behalf of dishonest voters. The adversary then gets to see the result corresponding to these ballots and a proof of correct tally. In the other world, the adversary gets to see a fake ballot box, but he still gets to see the result as tallied on the real ballot box. It is also assumed that there exists a simulator that can simulate a proof of correct tally corresponding to the real result, but with respect to the fake ballot box. The adversary also gets to see this simulated proof.

As the name suggests, ballot privacy BPRIV only captures the privacy of the ballots and not of the tally. To account for this Bernhard *et al.* say that any scheme satisfying BPRIV should also satisfy a property called strong consistency which captures the idea that the tally produced by the scheme should not leak more information than an idealised tally function. The exact idealised tally function is a parameter of the definition. We have proved the strong consistency of Selene as part of our work.

To avoid having to trust the voting server, the strategy in many voting systems is to encrypt the votes under a key for which the corresponding decryption key is split into several parts and distributed among several authorities. However, as is pointed out in [13], this trust assumption is not properly captured in the BPRIV definition. In BPRIV, the adversary plays a game where he can control the votes cast by honest parties, but he cannot control the resulting ballots once they are put in the ballot box. This means that BPRIV assumes that every ballot that is put in the ballot box stays in the ballot box and is not tampered with in any way.

To address this, Cortier *et al.* introduced a new definition, which they called mb–BPRIV [13]. The main idea in mb–BPRIV is similar to BPRIV: the adversary has to try and distinguish between two worlds: one where he sees real ballots and the real result, and one where he sees fake ballots, but still sees the real result. The difficulty is that an adversary who is in control of the ballot box is able to remove or tamper with any ballots submitted by honest voters. Since we perform the tally on the real ballots in both situations, we need to somehow determine which of the real ballots to perform the tally on. A bad choice would make distinguishing trivial for the adversary.

Cortier *et al.*'s solution is to parameterize their security definition by a *recovery algorithm*, an approach we also adopt in our definition. Informally, the idea is to use the recovery algorithm on the adversary's board in the fake world, to determine how the adversary has tampered with the ballots on the fake board. We then perform the same transformation on the real board, and tally the resulting board.

Formally, Cortier *et al.* define the aforementioned transformation as a *selection function*, and recovery algorithm as the process of finding the transformation.

Definition 4 (Selection function [13]). *For integers $m, n \geq 1$, a selection function for m and n is any mapping*

$$\pi : \{1, \ldots, n\} \to \{1, \ldots, m\} \cup (\{0,1\}^* \times \{0,1\}^*) .$$

The selection function π represents how the adversary constructs a bulletin board BB *with n ballots, given a bulletin board* BB_1 *with m ballots. For $i \in \{1, \ldots, n\}$,*

– $\pi(i) = j$, *with* $j \in \{1, \ldots, m\}$ *means that this is the* j*th element in* BB_1,
– $\pi(i) = (pc, c)$ *means that this element is* (pc, c).

The function $\overline{\pi}$ *associated to* π *maps a bulletin board* BB_0 *of length* m *to a board* $\overline{\pi}(\mathsf{BB}_0)$ *of length* n *such that*

$$\overline{\pi}(\mathsf{BB}_0)[j] = \begin{cases} (pc, c) \text{ if } \pi(j) = i \text{ and } \mathsf{BB}_0[i] = (id, (pc, c)) \\ (pc, c) \text{ if } \pi(j) = (pc, c) \end{cases}$$

for any $j \in \{1, \ldots, n\}$.

Definition 5 (Recovery algorithm [13]). *A recovery algorithm is any algorithm* Recover *that takes as input two bulletin boards* BB *and* BB_1 *and returns a selection function* π *for* $|\mathsf{BB}_1|$ *and* n *for some integer* n.

We will sometimes abuse notation and write $\mathsf{BB}' \leftarrow \mathsf{Recover}(\mathsf{BB}, \mathsf{BB}_0, \mathsf{BB}_1)$ to denote the process of determining the transformation from BB_1 to BB, and applying this transformation to BB_0, to get the board BB'.

In mb–BPRIV, the tally occurs only if the adversary's board is valid, and if none of the voters are unhappy after they perform some kind of verification. This means that the verification process needs to occur before the tally, so mb–BPRIV does not accommodate voting systems like Selene. Indeed, in Selene, the tally occurs before the verification as a way of mitigating the threat of coercion. Therefore, there is still need for a new privacy definition, that both allows for the voting server (and thus the ballot box) to be dishonest, and that accommodates voting protocols where the verification phase happens only after the tally has been computed.

3 New Security Notion

In this section, we present a new definition of ballot privacy against a malicious ballot box, which we call *delay-use malicious ballotbox ballot privacy* (du–mb–BPRIV). This definition essentially extends the range of applicable voting schemes to include those where the verification can be *delayed*, i.e. happening after tallying, and also includes schemes where a secret key is needed in the verification step. Our new definition is similar to mb–BPRIV, the most notable difference being that in our definition, the adversary gets to see the tally after we check if his board is valid, and then he gets to see the result of the verification phase. A formal description of the security game for du–mb–BPRIV is found in Figure 2. The relation between du–mb–BPRIV and mb–BPRIV is studied in more detail in Section 3.2.

In the following, let $\mathcal{I} = \mathsf{H} \cup \mathsf{D}$ be a set of voter identities, partitioned into a set H of honest voters and a set D of dishonest voters. Furthermore, let H be partitioned into the set $\mathsf{H}_{\mathsf{check}}$ of voters who we assume will perform some verification check and the set $\mathsf{H}_{\overline{\mathsf{check}}}$ of voters who we assume will not verify.

The security experiment begins with the generation of some public and secret data pd and sd (which typically includes the public and secret keys used to encrypt and decrypt

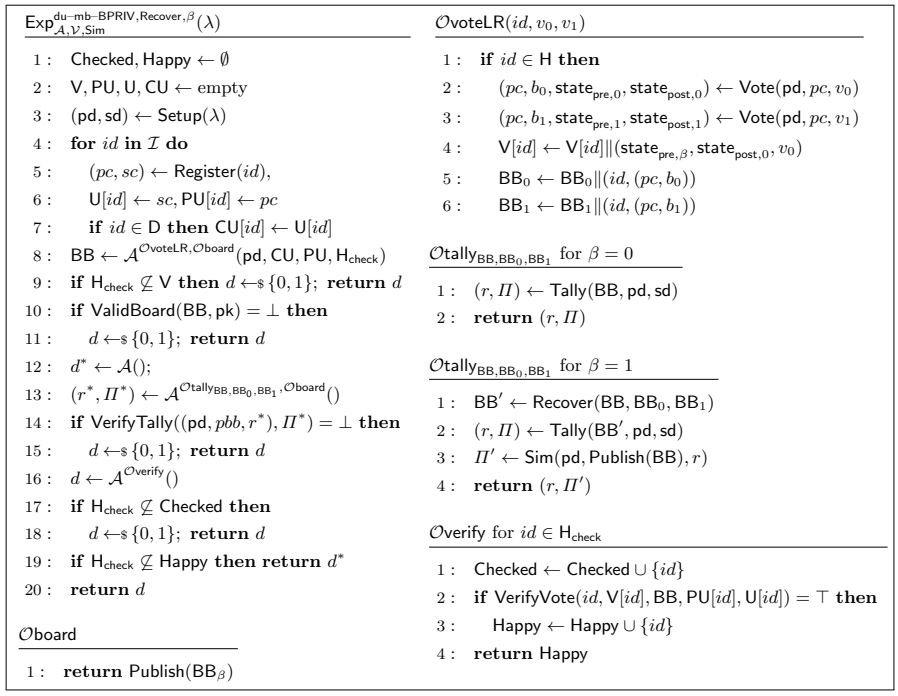


Fig. 2. The new security notion for ballot privacy against a dishonest ballot box.

the votes). Then, a number of voters in a set $\mathcal{I}$ are registered. In the registration phase, public and secret credentials pc and sc are generated for each voter, and stored in finite maps PU and U, respectively. Furthermore, we store the secret credentials of a set D of dishonest voters in a finite map CU.

The adversary is now given pd, PU and CU as input. In addition, he gets access to a vote oracle, that on input (id, v_0, v_1) computes two ballots for this user. The first ballot is stored in a list BB_0 and the second ballot is stored in a list BB_1. The vote oracle also records a state, containing any information the voter later needs to verify that her vote was correctly cast and counted; we split the state into two components. The first component ($\mathsf{state}_{\mathsf{pre}}$) covers information checked which is generated before tallying and the second component ($\mathsf{state}_{\mathsf{post}}$) covers information generated after tallying; this is necessary due to recovery which ensures that information after tallying (including the tally) always acts as if $\beta = 0$. The adversary may also call on a publish oracle, allowing him to see, essentially, BB_β for a secret bit β.

Using this information, the adversary creates a public bulletin board BB. If the board is invalid, we output a random bit. If the board is valid, we allow the adversary to make

an initial guess at the secret bit β, based on the information he has seen so far. This guess is stored in a variable d^*, possibly to be returned by the experiment at a later point.

The adversary now gets access to the tally, and is allowed to add some extra information to the bulletin board, namely a result r^* and a proof Π^* that this result corresponds to the votes. If this result and proof fails to pass verification, we output a random bit.

Otherwise, the adversary gets to make a guess d, given access to a verification oracle $\mathcal{O}\mathsf{verify}$. The verification oracle records the users who have verified in a set called $\mathsf{Checked}$, and the users who are happy with the verification are recorded in the set Happy. If anyone who we expect should verify actually does not verify, we output a random bit. If a voter is unhappy with the verification process, we output the initial guess d^* the adversary made before seeing the tally. Otherwise, the experiment outputs the guess d that the adversary made after calling the verify oracle.

When given access to the tally oracle, the adversary can call this oracle only once, and the behavior of the tally oracle depends on whether we are in the left world ($\beta = 0$) or in the right world ($\beta = 1$). If we are in the left world, the tally is performed directly on the board BB created by the adversary. The adversary then gets to see a real result and a proof of correct tally. In the right world, however, we first run the recovery algorithm to detect how the adversary has tampered with the ballots in BB_1, to create BB. We then change the ballots on BB_0 accordingly, yielding a new board BB', which we tally. The adversary then gets to see the result r corresponding to BB' and a simulated proof Π' of correct tally, with respect to r and the adversarial board BB.

Definition 6. *Let $\mathcal{V}$ be a voting system, and let* Recover *be a recovery algorithm. We say that $\mathcal{V}$ satisfies* du–mb–BPRIV *with respect to* Recover *if there exists an efficient simulator* Sim*, such that for any efficient adversary $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathsf{du\text{–}mb\text{–}bpriv}}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}(\lambda) = \Big| \Pr\left[\mathsf{Exp}^{\mathsf{du\text{–}mb\text{–}BPRIV},\mathsf{Recover},0}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{du\text{–}mb\text{–}BPRIV},\mathsf{Recover},1}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}(\lambda) = 1\right]\Big|,$$

is negligible in the security parameter λ, where $\mathsf{Exp}^{\mathsf{du\text{–}mb\text{–}BPRIV},\mathsf{Recover},\beta}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}$ is the game defined in Fig. 2.

mb–BPRIV with the recovery functions originally suggested implies the satisfying scheme is equivalent to some ideal functionalities. Future versions of du–mb–BPRIV may wish to prove similar results in which a different recovery function is likely necessary.

3.1 Recovery function

The mb–BPRIV definition is well defined for many recovery functions, but three are suggested in the body of [13]. All three recovery functions when used with mb–BPRIV are only satisfied by schemes which fulfil strong constraints on their verification procedure, and two of them presumes that voters are authenticated, see also discussion in Sec. 6.

We introduce a simple new recovery function (Fig. 3) which we call $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del,reorder'}}$ which is essentially identical to $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del,reorder}}$ and $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del,reorder,change}}$ in [13], but does not require ballot authentication and without explicit constraints on the verification procedure. mb–BPRIV with the recovery functions originally suggested implies the satisfying scheme is equivalent to some ideal functionalities. Future versions of du–mb–BPRIV may wish to prove similar results in which a different recovery function is likely necessary. In Sec. 6 below, we will discuss the precise relation between du–mb–BPRIV and verifiability, depending on the chosen recovery function.

$\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del,reorder'}}(\mathsf{BB}_1, \mathsf{BB})$

1 : $L \leftarrow [\,]$
2 : **for** $(pc, c) \in \mathsf{BB}$:
3 : **if** $\exists j, id : \mathsf{BB}_1[j] = (id, (pc, c))$:
4 : $L \leftarrow L \parallel j$ (if several such j exist, pick the first one)
5 : **else** :
6 : $L \leftarrow L \parallel (pc, c)$
7 : **return** $(\lambda i.\ L[i])$

Fig. 3. The $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del,reorder'}}$ algorithm.

3.2 Comparison of du–mb–BPRIV to mb–BPRIV

In this section we briefly analyse the differences between the mb–BPRIV definition and our new du–mb–BPRIV definition. As mentioned above, the main difference is that the verification oracle is first available after the tally oracle has been called. This accommodates schemes where the verification first happens after tally and allows a secret key to be used for the verification process, however, it naturally also applies to schemes with early verification. We have changed some parts of the definition to adapt to the delayed use of the verification, but also to make it optimised and precise enough for EasyCrypt.

The main differences are:

- We only have one voter map V but the state stored depends on secret bit β, see line 4 in the definition of $\mathcal{O}\mathsf{voteLR}$ in Figure 2. However, the state is split into a part relevant before tally and a post-tally part (only relating to $\beta = 0$ which is the board used for tallying). This is necessary for the state handling in EasyCrypt oracles, and was not necessary in mb–BPRIV since the stateful verification happened before tallying.
- If the adversary does not output a valid board, the experiment outputs a random guess bit, whereas mb–BPRIV allows the adversary to output a guess but without tally access. In both cases this corresponds to real life, where a board will not be tallied if it is not valid. Outputting a random bit makes our proofs in EasyCrypt slightly easier, and it is actually equivalent to mb–BPRIV unless the $\mathsf{ValidBoard}$ algorithm always outputs $\perp$.

- In our definition the experiment also outputs a random guess bit if the verification of the tally fails via the algorithm VerifyTally. Again, this corresponds to not allowing any adversarial advantage when the tally fails publicly. This case was not explicitly considered in mb–BPRIV, but is natural in our case where the verification step will not proceed on an invalid tally output.
- In du–mb–BPRIV the verification status of the honest voters contained in Happy is directly output to the adversary. In mb–BPRIV this is not defined precisely. In both definitions the appearance of failed verifications inside the set of checking honest voters $\mathsf{H}_{\mathsf{check}}$ will in any case imply that the guess bit has to be determined without knowing the tally result. This is to punish the adversary for creating a board with verifications failing which would cause complaints in real life protocols.

Consider a voting scheme where the verification step does not depend on a secret key and can be done before the tally. For such schemes both privacy definitions can be applied to the scheme. We claim that under reasonable conditions our definition is stronger. Further, for voting schemes where the outcome of the individual verifications can be computed by the adversary using the data from the bulletin board, as e.g. happens in Helios, we have equivalence of the two definitions. We now sketch why this is the case.

We show that du–mb–BPRIV privacy implies mb–BPRIV assuming that ValidBoard does not constantly output $\perp$, and that VerifyTally never fails on an honestly computed tally. Finally, we also assume that the verification status Happy is not output to the adversary in mb–BPRIV or that the verification status can be computed using public data, as e.g. happens in Helios. To prove the implication we assume that we have an attack algorithm for mb–BPRIV. We use the vote choices from the mb–BPRIV algorithm. If the attack algorithm outputs an invalid board, we change the board that is being output to a valid board which we have assumed exists. Since the tally also does not fail we are allowed to output a guess and we use the one from the attack algorithm and will win with the same advantage since in this case no verification will be done in mb–BPRIV. If the attack algorithm outputs a valid board, we use the same board in the du–mb–BPRIV experiment. In du–mb–BPRIV line 12 the adversary has to output a guess bit d^* which will be used if a verification fails for the honest verifier set $\mathsf{H}_{\mathsf{check}}$. Here we use what will be output from the mb–BPRIV attack algorithm in case of failed verifications, which by assumption either does not depend on Happy, which we do not yet have access to at this stage in the du–mb–BPRIV experiment, or it can be computed from the public data on the board. In the experiment du–mb–BPRIV line 13 we now get the tally before verification (and the tally verifies by assumption) but we ignore this at first and choose the same verifying voters as in the attack algorithm. At this point the two experiments will be equivalent. If the verification fails we are forced to go back to d^* but this was from the attack algorithm and will be equivalently successful. If no verification fails in $\mathsf{H}_{\mathsf{check}}$ then we output the guess from the attack algorithm using the tally result we got earlier as input. The advantage will be the same as for the mb–BPRIV attack algorithm. This was the important implication direction since it demonstrates that our definition is not too weak, i.e. if an early verification scheme is declared du–mb–BPRIV then it is also

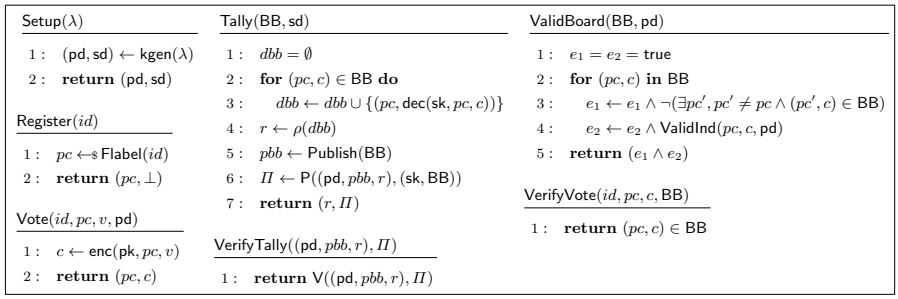


Fig. 4. Algorithms defining the Labelled-MiniVoting scheme for the labelled PKE $\mathsf{E} = (\mathsf{kgen}, \mathsf{enc}, \mathsf{dec})$, and the proof system $\Sigma = (\mathsf{P}, \mathsf{V})$.

mb–BPRIV private which in turn has ideal functionality implications under assumptions such as strong consistency [13].

Considering whether mb–BPRIV privacy implies du–mb–BPRIV, the main problem is that the choice of verifying voters in du–mb–BPRIV could depend on the tally output. However, for voting schemes where the outcome of the honest voters' individual verification can be computed by the adversary, we can prove the implication. We thus assume we have an attack algorithm for du–mb–BPRIV with non-negligible advantage. In the experiment mb–BPRIV we use the votes and output the board from this attack algorithm. If the board is not valid, we simply output a random bit in the experiment mb–BPRIV, which is equivalent to what happens in du–mb–BPRIV. If the board is valid, the next step in mb–BPRIV is to use the verification oracle. Here we simply let all the required honest voters $\mathsf{H}_{\mathsf{check}}$ perform verifications. These will also all have to verify in the attack against du–mb–BPRIV to get an advantage since the experiment will otherwise output a random bit. If a verification fails we will use the output d^* in the attack algorithm which was computed without tally access. If none of the verifications fail, we will get tally access in the experiment mb–BPRIV. If more voters were chosen to verify in the attack algorithm we can compute the outcomes by assumption. If we encounter a failure in the verification here, we again output d^*, otherwise the output from the attack algorithm. The experiments will be equivalent at this point.

4 Labelled-MiniVoting and Belenios

The MiniVoting scheme was first introduced by Bernhard *et al.* [6] as an abstraction meant to capture several existing constructions in the literature. It is based on two building blocks: public key encryption and a zero-knowledge proof system, and assumes the ballot has the form (id, c), for a voter's identity id and for a ciphertext c - that simply encrypts the voter's choice (or vote) v.

This scheme was later refined by Cortier *et al.* [10], resulting in the *Labelled-MiniVoting* scheme. Here, the class of captured voting schemes was broadened by introducing some public information associated to the users, called *labels*, and creating a strong link between

a voter identity in a particular election and its ciphertext - now parameterized by this label via the use of labelled public key encryption. The labels can be used to represent generic information about the election (as in the case of Helios) and/or information pertaining to a voter's public persona: pseudonym or public verification key (as in Belenios). The ballot in Labelled-MiniVoting takes the form (id, ℓ, c) with id the voter's identity and (ℓ, c) the label-ciphertext pair created by the labelled public key encryption scheme.

A predominant feature of the MiniVoting class of schemes is the enforcement of unique label-ciphertext pairs, via a procedure called weeding [12]. This step, done by the ValidBoard algorithm, prevents trivial attacks on privacy, where an adversary can cast copied ciphertexts (and their label) and observe the changes in the election result.

In Figure 4 we refine Labelled-MiniVoting to align with the voting notations from previous sections, such that we treat the public credential pc as the label and consider the following ballot format (pc, c). The removal of id doesn't have a significant impact, as we use the operation Flabel to model the link between identity and public credential:

$$pc \leftarrow \mathsf{Flabel}(id).$$

Typical instantiations for this operator depend on the assumptions over the voter's identity and the degree of separation we want to capture in the public credential. Here, we consider the voter's identity a pseudonym or a public encryption key and as such we implement Flabel as the identity function $\mathsf{Flabel}(x) = x$; an approach also taken by Selene. Other options may consider the real voter's identity (e.g. email, name) as input and create a pseudonym or a public credential (especially if signatures are considered). In all cases, there has to be an injectivity assumption over Flabel, which is trivially satisfied by the identity function.

Labelled-MiniVoting is further parameterized by three other classic operators:

$\mathsf{ValidInd}(pc, c, \mathsf{pd}) \colon \{0,1\}$**.** Checks if the label-ciphertext pair (pc, c) is well-formed.

$\rho((pc_i, v_i)_i)$**.** This function returns the election result in a predefined format, e.g. lexicographic order for mixnet tally or a value for homomorphic tally. This is done by first deciding which votes to keep using Policy, and then computing the result over the votes kept by Policy using Count.

- Policy is fixed to "last vote counts" for a particular voter, in the modelling of both Labelled-MiniVoting and Selene.
- Count is left abstract for Labelled-MiniVoting, and made concrete for Selene.

$\mathsf{Publish}(\mathsf{BB}) \colon \{0,1\}^*$**.** This is an abstraction of the public bulletin board, and most of the times it is identical to the ballot box.

Definition 7. *Let* E *be a* poly–IND–1–CCA *secure labelled PKE, and* $\Sigma = (\mathsf{P}, \mathsf{V})$ *be a zero-knowledge proof system. Given the operators* Flabel, ValidInd, ρ, Publish *defined as above, we define the Labelled-MiniVoting scheme*

$$\mathsf{MiniVoting}(\mathsf{E}, \Sigma, \mathsf{Flabel}, \mathsf{ValidInd}, \rho, \mathsf{Publish}) =$$
$$(\mathsf{Setup}, \mathsf{Register}, \mathsf{Vote}, \mathsf{ValidBoard}, \ \mathsf{Tally}, \mathsf{VerifyVote}, \mathsf{VerifyTally}, \mathsf{Publish})$$

as the single-pass voting scheme whose algorithms are presented in Figure 4, and which we informally present below:

$\mathsf{Setup}(\lambda) : (\mathsf{pd}, \mathsf{sd})$. Given the security parameter λ it returns the output $(\mathsf{pd}, \mathsf{sd})$ of the key generation algorithm for the encryption scheme.

$\mathsf{Register}(id) : (pc, sc)$. It applies Flabel over id to build the public credential pc, and does not consider a secret credential, i.e. $sc \leftarrow \perp$.

$\mathsf{Vote}(id, pc, v, \mathsf{pd}) : (pc, c)$. Encrypts a vote v using the public credential pc as the label, and outputs the ciphertext together with the voter's public credential. The secret credential sc is omitted from the input just for simplicity as it is unused.

$\mathsf{ValidBoard}(\mathsf{BB}, \mathsf{pd}) : \{0, 1\}$. Returns true if all ballots (pc, c) are well-formed, according to $\mathsf{ValidInd}$, and that each ciphertext is always used with the same public credential (weeding property is respected); and false otherwise.

$\mathsf{Tally}(\mathsf{BB}, \mathsf{sd}) : (r, \Pi)$. Computes the result r of the election by applying the counting function ρ over the list of $(pc, v_\perp)$, where $v_\perp$ is the decryption of (pc, c) from the ballot box. It also provides a proof of correct decryption Π using the P algorithm of the proof system Σ.

$\mathsf{VerifyVote}(id, pc, c, \mathsf{BB}) : \{0, 1\}$. It checks if its last ballot (pc, c) is in the ballot box BB. The voter may have a state with all cast ballots, but the verification is done with respect to the last vote that was cast.

$\mathsf{Verifytally}((\mathsf{pd}, pbb, r), \Pi) : \{0, 1\}\}$. Run the V algorithm of the proof-system to check if the tally proof is valid w.r.t the given statement.

$\mathsf{Publish}(\mathsf{BB})$: Calls the $\mathsf{Publish}$ operator over the ballot box BB.

We prove in EASYCRYPT that $\mathsf{MiniVoting}$ satisfies du–mb–BPRIV under standard cryptographic assumptions for the encryption scheme E and proof system Σ. We also consider the identity function for Flabel, and "last vote counts" for Policy. The following theorem corresponds to lemma `du_mb_bpriv` in the `MiniVotingSecurity_mb.ec` file.

Theorem 1. *Let* $\mathcal{V} = \mathsf{MiniVoting}(\mathsf{E}, \Sigma, \mathsf{Flabel}, \mathsf{ValidInd}, \rho, \mathsf{Publish})$ *be defined as in Definition 7. Then, for any* du–mb–BPRIV *adversary* $\mathcal{A}$*, there exists a simulator* Sim *and three adversaries* $\mathcal{B}, \mathcal{C}$ *and* $\mathcal{D}$*, such that*

$$\mathsf{Adv}^{\mathsf{du-mb-bpriv}}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}(\lambda) \leq 2 \cdot \Pr\left[\mathsf{Exp}^{\mathsf{vfr}}_{\mathcal{D},\mathcal{V},\Sigma}(\lambda) = 1\right] + \mathsf{Adv}^{zk}_{\mathcal{B},\mathsf{P},\mathsf{Sim},\mathcal{R}}(\lambda) + \mathsf{Adv}^{\mathsf{poly-ind1cca}}_{\mathcal{C},\mathsf{E},n}(\lambda).$$

4.1 du–mb–BPRIV for Belenios

We have used Labelled-MiniVoting to validate our privacy definition, and to infer proof strategies and assumptions that then can be applied to other e-voting systems, e.g. Belenios [12] and Selene. Belenios can be viewed as an instance of Labelled-MiniVoting with some concrete decisions for operators and algorithms. This translates into directly applying the EASYCRYPT proof developed for MiniVoting to Belenios without the need to re-do it - as highlighted in `Belenios.ec`.

We take the labelled encryption scheme E and realize it by combining the ElGamal encryption scheme E_B with a zero-knowledge proof system Σ_B. As the public credential pc is included in the statement for the proof system Σ_B we use $\pi[pc]$ to express this fact. Encrypting the vote v under some public credential pc by E becomes $(pc, c) = (pc, (\mathsf{c}_\mathsf{B}, \pi[pc]))$,

and decrypting by E returns the decryption of $\mathsf{c_B}$ by $\mathsf{E_B}$ only if the proof $\pi[pc]$ verifies, and $\perp$ otherwise.

This form of the ciphertext also has an impact on the ValidInd algorithm that now uses $\pi[pc]$ to decide if a ciphertext is well-formed.

For the result of the election, we only need to instantiate the Count operator, as we already consider the last vote policy. We can model any type of ideal counting function that can be performed over votes, and instantiate it to lexicographic order `lex-order` to model an ideal verifiable shuffle.

In Belenios, the public bulletin board computed by Publish shows only the last ballot cast by a voter (policy applied over the public credential) together with a hash of that ballot. With verification done against the hash compared to the entire ballot. However, nothing prevents voters from checking their full ballot against the ballot box (as we have modelled in Labelled-MiniVoting). Moreover Cortier *et al.* [11] have modelled in EASYCRYPT different options of Publish for Belenios and as one would expect the two approaches are equivalent modulo hash collisions.

Definition 8. *Let* $\mathsf{E} = (\mathsf{E_B}, \Sigma_\mathsf{B})$ *be a* poly–IND–1–CCA *secure labelled PKE, and* Σ *be a zero-knowledge proof system. Given the operators* $\mathsf{Flabel}, \mathsf{ValidInd}, \rho, \mathsf{Publish}$ *defined as above, we define* $\mathsf{Belenios}(\mathsf{E_B}, \Sigma_\mathsf{B}, \Sigma)$ *as*

$$\mathsf{MiniVoting}(\mathsf{E}, \Sigma, \mathsf{Flabel}, \mathsf{ValidInd}, \rho, \mathsf{Publish}).$$

The privacy result for Belenios follows directly by simply applying Theorem 1 with the concrete values highlighted here.

Theorem 2. *Let* $\mathcal{V} = \mathsf{Belenios}(\mathsf{E_B}, \Sigma_\mathsf{B}, \Sigma)$ *be defined as in Definition 8. Then, for any* du–mb–BPRIV *adversary* $\mathcal{A}$*, there exists a simulator* Sim *and three adversaries* $\mathcal{B}, \mathcal{C}$ *and* $\mathcal{D}$*, such that*

$$\mathsf{Adv}^{\mathsf{du-mb-bpriv}}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}(\lambda) \leq 2 \cdot \Pr\left[\mathsf{Exp}^{\mathsf{vfr}}_{\mathcal{D},\mathcal{V},\Sigma}(\lambda) = 1\right] + \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{B},\mathsf{P},\mathsf{Sim},\mathcal{R}}(\lambda) + \mathsf{Adv}^{\mathsf{poly-ind1cca}}_{\mathcal{C},\mathsf{E},n}(\lambda).$$

Both Theorem 1 and 2 are proven with respect to the recovery algorithm described in Section 3.1.

4.2 mb–BPRIV for MiniVoting and Belenios

We also stated the original mb–BPRIV definition [13] in EASYCRYPT and proved that Theorem 1 and 2 also hold with respect to this privacy definition. The proof strategy was essentially the same, but the proofs had to be re-worked due to the differences between the definitions, especially when the verification happens. This constitutes the first machine-checked proof of mb–BPRIV. The corresponding EASYCRYPT lemma `mb_bpriv` is found in the `MiniVotingSecurity_omb.ec` and `Belenios_omb.ec` files.

5 Selene

Although Selene offers properties such as verification and coercion mitigation, we focus—in this paper—on formalizing its ballot privacy properties. Like Cortier *et al.* [10, 11], we abstract away the verifiable shuffles (assuming they are non-interactive) and the ElGamal + PoK construction (assuming instead an abstract IND–1–CCA-secure labelled PKE).

More precisely, we replace the verifiable shuffle—which is used in the tally phase to mix the encrypted votes and trackers while erasing connections between the votes and the voters—with a parametric Multiset distribution, which takes as parameters a list of vote/-tracker pairs $(v_i, \mathsf{tr}_i)_i$, calculates all possible permutations of the original list, and defines the uniform distribution over the result. Sampling in $\mathsf{Multiset}(\ell)$ captures the semantics of a perfect shuffle on ℓ, with a proof of correct shuffle computed separately.[1] Formalizing proofs for interactive protocols, such as verifiable shuffles, in EasyCrypt remains a complex task, and is somewhat orthogonal to our contributions here. In particular, in a setting—like ours—in which the tallier is honest, the shuffle is indeed indistinguishable from our idealization. Prior work [16, 17] has proved that the interactive variants of the verifiable shuffles suggested for use with Selene are zero-knowledge proofs which leak no information; this would suffice to prove equivalence with our idealisation. However, this has not been machine checked for the interactive variant due to issues the currently available tools have with handling random oracles.

Before being cast, the votes in Selene are encrypted using the ElGamal public key encryption system [15], and a non-interactive proof of knowledge of the underlying plaintext is appended to the ciphertext. In our EasyCrypt formalization, we abstract away details of the underlying cryptosystem, and encrypt the votes using an abstract labelled PKE that we assume is IND–1–CCA secure. The ElGamal with proof of knowledge construction used in Selene has in fact been proven to be IND–1–CCA secure [4, 7]. This proof remains out of reach of machine-checking due to its use of the rewinding lemma.

In addition, since we focus on privacy in this paper, we also remove the signatures Selene[2] includes on cast ballots to prevent ballot stuffing. The signatures cannot compromise privacy: The signing keys are independent of the encryption of the ballot, the ciphertexts that are signed are public, and the signatures are anyway stripped before shuffling, so they cannot be correlated to any plaintext ballot.

These simplifications, taken together, yield the following model for Selene.

Definition 9. *Let* $\mathsf{E_v}$ *be a* poly–IND–1–CCA *secure labelled PKE, let* $\mathsf{E_t}$ *be an* IND–CPA *secure PKE, let* $\Sigma_{\mathcal{R}} = (\mathsf{P}, \mathsf{V})$ *be a proof system for a relation* $\mathcal{R}$ *and let* $\mathsf{CP} = (\mathsf{gen}, \mathsf{commit}, \mathsf{open})$ *be a commitment protocol. We define*

$$\mathsf{Selene}(\mathsf{E_v}, \mathsf{E_t}, \Sigma_{\mathcal{R}}, \mathsf{CP}, \mathsf{ValidInd})$$

as the voting system built upon the algorithms given in Figure 5, which we informally describe below.

[1] We note that Multiset itself is not probabilistic polynomial time. We treat it as an ideal functionality for verifiable shuffles, whose complexity would normally be probabilistic polynomial time.

[2] We also have an EasyCrypt proof for Selene with signatures. This is available with the other proofs at `https://github.com/mortensol/du-mb-bpriv`.

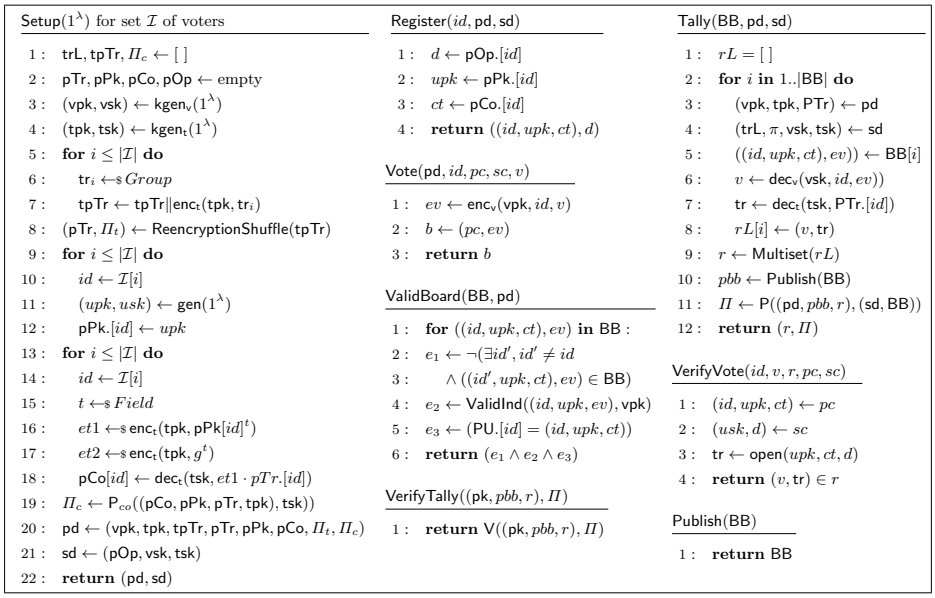


Fig. 5. Algorithms defining the Selene[$\mathsf{E_v}, \mathsf{E_t}, \mathsf{C}, \Sigma,$ ValidInd] voting scheme, given an IND–1–CCA secure labelled PKE scheme $\mathsf{E_v} = (\mathsf{kgen_v}, \mathsf{enc_v}, \mathsf{dec_v})$, an IND–CPA secure homomorphic encryption scheme $\mathsf{E_t} = (\mathsf{kgen_t}, \mathsf{enc_t}, \mathsf{dec_t})$, zero-knowledge proof systems $\Sigma_{ta} = (\mathsf{P}_{ta}, \mathsf{V}_{ta})$, $\Sigma_{tsh} = (\mathsf{P}_{tsh}, \mathsf{V}_{tsh})$ and $\Sigma_{co} = (\mathsf{P}_{co}, \mathsf{V}_{co})$ for the tally proof, the tracker shuffle proof and the proof that commitments are correctly formed, respectively, a commitment scheme $\mathsf{CP} = (\mathsf{gen}, \mathsf{commit}, \mathsf{open})$, and the abstract operator ValidInd.

Setup: Takes as input a security parameter λ and returns a key pair (vpk, vsk) used to encrypt and decrypt votes, a key pair (tpk, tsk) used to encrypt and decrypt trackers, a list tpTr of encrypted trackers, finite maps pTr, pPk, pCo and pOp from voter identities to encrypted trackers, public commitment keys, tracker commitments and openings, respectively, as well as a proof Π_t of correct shuffle of the trackers and a proof Π_c that the tracker commitments are correctly formed. The public data is

$$\mathsf{pd} = (\mathsf{vpk}, \mathsf{tpk}, \mathsf{tpTr}, \mathsf{pTr}, \mathsf{pPk}, \mathsf{pCo}, \Pi_t, \Pi_c)$$

and the secret data is

$$\mathsf{sd} = (\mathsf{pOp}, \mathsf{vsk}, \mathsf{tsk}).$$

Register: Takes as input a voter identity and a pair (pd, sd) of public and secret data, and returns the voter's public commitment key, the commitment to her tracker and an opening to the commitment.

Publish: Outputs the public part of the ballot box.

Vote: Encrypts a vote v and outputs the ciphertext together with the voter's public credential.

VerifyTally: Run the V algorithm of the proof-system to check if the tally proof is valid.

```
lemma du_mb_bpriv &m : BP.setidents{m} = BP.setH{m} `|` BP.setD{m} =>
(* The du-mb-BPRIV advantage of some adversary A upper bounded by the sum of: *)
`|Pr[DU_MB_BPRIV_L(Selene(Et,Ev,P,Ve,C,CP),A,HRO.ERO,G).main() @ &m: res]
- Pr[DU_MB_BPRIV_R(Selene(Et,Ev,P,Ve,C,CP),A,HRO.ERO,G,S,Recover').main() @ &m: res]|
(* - the advantages of BVFR(G) and BVFR(S) in breaking the voting friendly relation *)
<= Pr[VFRS(Et,Ev,BVFR(Selene(Et,Ev,P,Ve,C,CP),A,CP),R,HRO.ERO,G).main() @ &m: res]
 + Pr[VFRS(Et,Ev,BVFR(Selene(Et,Ev,P,Ve,C,CP),A,CP),R,HRO.ERO,S).main() @ &m: res]
(* - the ZK advantage of adversary BZK against the underlying NIZK, and *)
+ `|Pr[ZK_L(R(Et,Ev,HRO.ERO),P,BZK(Et,Ev,P,C,Ve,A,CP,HRO.ERO),G).main() @ &m: res]
 - Pr[ZK_R(R(Et,Ev,HRO.ERO),S,BZK(Et,Ev,P,C,Ve,A,CP,HRO.ERO)).main() @ &m: res]|
(* - the IND1-CCA advantage of adversary BCCA against the ballot-encryption scheme *)
+ `|Pr[Ind1CCA(Ev,BCCA(Selene(Et,Ev,P,Ve,C,CP),Et,CP,A,S),HRO.ERO,Left).main() @ &m: res]
 - Pr[Ind1CCA(Ev,BCCA(Selene(Et,Ev,P,Ve,C,CP),Et,CP,A,S),HRO.ERO,Right).main() @ &m: res]|
```

Fig. 6. EASYCRYPT lemma establishing that Selene satisfies du–mb–BPRIV. HRO.ERO and G are independent random oracles. S is the ZK simulator.

ValidBoard: For each element in the ballotbox BB, we perform three checks: every ballot contains a unique voter identity, every ballot is well-formed, and every public credential corresponds to the correct identity.

Tally: Decrypts every encrypted vote in the ballot box BB and the tracker for each voter. Returns the multiset of all vote/tracker pairs (v, tr).

VerifyVote: To verify, a voter opens her tracker commitment and checks if her vote v and tracker tr is in the list of vote/tracker pairs returned by Tally.

The following theorem establishes that Selene satisfies du–mb–BPRIV.

Theorem 3. *Let* $\mathcal{V} = \mathsf{Selene}(\mathsf{E_v}, \mathsf{E_t}, \Sigma_{\mathcal{R}}, \mathsf{CP}, \mathsf{ValidInd})$*, where* $\mathsf{ValidInd}(\mathsf{pc}, \mathsf{vpk}) = \top$ *for* $c \leftarrow \mathsf{enc_v}(\mathsf{vpk}, id, v)$ *and any public encryption key* vpk*, identity* id*, public credential pc and vote* v*. For any* du–mb–BPRIV *adversary* $\mathcal{A}$*, there exists a simulator* Sim *and four adversaries* $\mathcal{B}, \mathcal{C}, \mathcal{D}_S$ *and* $\mathcal{D}_G$*, such that*

$$\mathsf{Adv}^{\mathsf{du-mb-bpriv}}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}(\lambda) \leq \Pr\left[\mathsf{Exp}^{\mathsf{vfr}}_{\mathcal{D}_G,\mathcal{V},\Sigma_{\mathcal{R}}}(\lambda) = 1\right] + \Pr\left[\mathsf{Exp}^{\mathsf{vfr}}_{\mathcal{D}_S,\mathcal{V},\Sigma_{\mathcal{R}}}(\lambda) = 1\right] + \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{B},\mathsf{P},\mathsf{Sim},\mathcal{R}}(\lambda) + \mathsf{Adv}^{\mathsf{poly-ind1cca}}_{\mathcal{C},\mathsf{E_v},n}(\lambda).$$

Theorem 3 corresponds to lemma `du_mb_bpriv` in `SeleneBpriv.ec`. Figure 6 displays the EASYCRYPT formulation of Theorem 3. The lemma itself is inside a section which quantifies over all core components: `Et` and `Ev` denote the encryption schemes used for the trackers and the votes, respectively, `P` and `Ve` denote the NIZK's prover and verifier, `C` denotes the ValidInd algorithm, `CP` denotes the commitment protocol. `A` is the adversary. The zero-knowledge simulator `S` and the random oracles for tracker encryption (`HRO.ERO`) and for the NIZK proof system (`G`) are defined concretely in the code, but not fully displayed in this paper. The VFR, zero knowledge and poly–IND–1–CCA security experiments are denoted by `VFRS`, `ZK_L`, `ZK_R` and `Ind1CCA`, respectively, while the respective reductions are denoted by `BVFR`, `BZK` and `BCCA`. These are also given concrete definitions. Also note that the `Ind1CCA` module is parameterized by a left-or-right module, representing the case where $\beta = 0$ and

the case where $\beta = 1$, respectively. The du–mb–BPRIV security experiment is parameterized by a recovery algorithm, and we use the concrete recovery algorithm described in Section 3.1.

We now sketch the proof of Theorem 3. The EasyCrypt formalization of the full proof is found in the file `SeleneBpriv.ec`. Unless explicitly stated, all the modules and lemmas we refer to in the following are also found in this file.

In our EasyCrypt formalization, we split the experiment $\mathsf{Exp}^{\mathsf{du\text{–}mb\text{–}BPRIV},\mathsf{Recover},\beta}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}$ into two different games, one for $\beta = 0$ and one for $\beta = 1$. The difference between the two games is, as described earlier, what the tally algorithm does. These games are modeled in the modules `DU_MB_BPRIV_L` and `DU_MB_BPRIV_R`, respectively, which are found in the file `VotingSecurity_mb.ec`.

Starting out from the left side security experiment, the first step is to replace the tally proof produced by the prover in the proof system, by a simulated proof produced by the simulator Sim. This change is modeled in the game `G1L`. Provided that the proof system is zero-knowledge, the adversary cannot distinguish between the original game and the game where we simulated the proof, except with negligible probability.

We then define a new game, `G2L`, where we stop decrypting honestly created ciphertexts, and instead use the ciphertexts stored in V (as described in Section 3). Ciphertexts submitted by the adversary are decrypted as usual. We also remove one of the bulletin boards from the vote oracle, so that only the left-side votes are stored. We prove that `G1L` and `G2L` are equivalent. The equivalence follows from the correctness property of the encryption scheme used to encrypt the votes, and the fact that the adversary only gets to see BB_0 in the left side security experiment.

Starting out from the right side security experiment (`DU_MB_BPRIV_R`), we first stop decrypting honest ciphertexts, and prove that the resulting game (`G1R`) is equivalent to `DU_MB_BPRIV_R`. The intuition is the same as for the equivalence between `G1L` and `G2L`.

We then define a game `G2R` where we stop performing recovery on the adversarially created ballot box, and simply perform the tally on the ballot box the adversary outputs. We prove that `G1R` and `G2R` are equivalent. Intuitively, this holds because the honest votes no longer come from the adversary's board, but from V, and the ballots submitted by the adversary are present both on the adversary's board BB and on the recovered board, by definition of our recovery algorithm.

In the final game, `G3R`, we remove one of the bulletin boards in the vote oracle. This is similar to what we did in `G2L`, but now only the right side votes are stored. We prove that `G2R` and `G3R` are equivalent. This also shows that the final game on the right side, `G3R`, is completely equivalent to `DU_MB_BPRIV_R`.

Finally, we show that the probability in distinguishing between the games `G2L` and `G3R` is equivalent to the probability of winning the IND–1–CCA game.

6 Verifiability

In this section we will consider the relation between du–mb–BPRIV and individual verifiability. Indeed, in [13] it was proven that a scheme satisfying mb–BPRIV and *strong*

consistency will satisfy a special form of individual verifiability which depends on the chosen recovery function in mb–BPRIV.

We will here shed more light on this relation both for mb–BPRIV and du–mb–BPRIV, in particular, the relation is hardcoded into the recovery function and it is instructive to understand this in more detail.

Actually, the recovery function not only defines which attacks are allowed by the adversary, it also determines which properties the individual verification has to verify, by directly allowing attacks if the verification fails to check for this. Consider the two recovery functions $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del,reorder}}$ in Fig. 7 and $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del,reorder,change}}$ in Fig. 8. In $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del,reorder}}$ the attacker is allowed to delete and reorder votes of honest non-checking voters, and in $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del,reorder,change}}$ to delete, reorder and replace ballots of honest non-checking voters. For both it is hardcoded that if an honest checking voter's ballot is deleted, replaced or changed[3] in any way there will be different tally results in the two worlds. Reordering would not cause different results. Thus if the verification procedure does not detect these changes, the adversary will be able to tally and win the game. In turn, this hardcoding of what the verification should achieve will also give the relation to verifiability, as we will prove below. However, what we define for the verifying voters, may not be related to what is defined to be legal actions for the attacker for the honest non-verifying voters, which makes the definition somewhat opaque.

Following the argument of [13] e.g. deleting votes does have implications for privacy, basically reducing the remaining anonymity set – in the extreme case an attacker could delete all votes except one to reveal that vote with the tally. However, above we did not demand that verification detects vote re-ordering which could potentially have just as large consequences for privacy, e.g. in the case where all voters cast two ballots and the attacker knows all the first ballots. Swapping ballots in a last-vote-counts policy then decreases privacy just as deleting votes would do.

In our recovery function $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del,reorder'}}$ we do not put a constraint on what verification should do, but treat all honest voters equally. This means that replacement of voter ballots is allowed for verifying voters, contrary to $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del,reorder,change}}$ where it would lead to an attack. This, in principle, allows us to model attacks where the adversary learns the outcome of a plaintext verification after submitting a ballot with a given candidate on behalf of the voter. As an example, if the adversary submits a ballot for candidate A on behalf of a voter and observes that the verification is successful, then the adversary can deduce that the voter voted for A. This does not mean that mb–BPRIV with $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del,reorder,change}}$ is too weak, because replacement is ruled out for verifying voters, but it does mean that this high precision privacy attack would be overlooked, due to concerns of less privacy-violating attacks.

On the other hand, since $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del,reorder'}}$ does not give an explicit constraint on the verification, this also means that the du–mb–BPRIV with our recovery function is not

[3] We note that all the proposed recovery functions are too strong, in the sense that if a ballot is modified in any way but without changing the plaintext, this would give an attack, but the vote result would be unchanged and hence this does not constitute an attack. E.g. Belenios-RF [9] or Selene would not detect this during verification, but would still be private in practice.

$\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del},\mathsf{reorder}}(\mathsf{BB}_1, \mathsf{BB})$

$L \leftarrow [\,]$
for $(pc, c) \in \mathsf{BB}$:
  if $\exists j, id : \mathsf{BB}_1[j] = (id, (pc, c))$:
    $L \leftarrow L \parallel j$ (if several such j exist, pick the first one)
  elseif $\mathsf{Extract}_{id}(\mathsf{U}, pc) \notin \mathsf{H}$
    $L \leftarrow L \parallel (pc, c)$
$L' \leftarrow [i | \mathsf{BB}_1[i] = (id, (pc, c)) \wedge id \in \mathsf{H}_{\mathsf{check}} \wedge (pc, c) \notin \mathsf{BB}]$
$L'' = L \| L'$
return $(\lambda i.\ L''[i])$

Fig. 7. The $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del},\mathsf{reorder}}$ algorithm from [13]. Here $\mathsf{Extract}_{id}(\mathsf{U}, pc)$ extracts the voter identity id from the public credential pc.

$\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del},\mathsf{reorder},\mathsf{change}}(\mathsf{BB}_1, \mathsf{BB})$

$L \leftarrow [\,]$
for $(pc, c) \in \mathsf{BB}$:
  if $\exists j, id : \mathsf{BB}_1[j] = (id, (pc, c))$:
    $L \leftarrow L \parallel j$ (if several such j exist, pick the first one)
  elseif $\mathsf{Extract}_{id}(\mathsf{U}, pc) \notin \mathsf{H}_{\mathsf{check}}$
    $L \leftarrow L \parallel (pc, c)$
$L' \leftarrow [i | \mathsf{BB}_1[i] = (id, (pc, c)) \wedge id \in \mathsf{H}_{\mathsf{check}} \wedge (pc, c) \notin \mathsf{BB}]$
$L'' = L \| L'$
return $(\lambda i.\ L''[i])$

Fig. 8. The $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del},\mathsf{reorder},\mathsf{change}}$ algorithm from [13]. This is identical to $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del},\mathsf{reorder}}$ from Fig. 7, except that votes from $\mathsf{H}_{\overline{\mathsf{check}}}$ are also considered as cast.

in general directly implying verifiability. On a technical level: A verifiability attack that changes ballots would in most cases be seen as cast ballots by the recovery function and hence be both on the left and right world boards. Hence the tally would not change.

To investigate the relation to verifiability, we will thus focus on the recovery functions from [13]. We note that in [13] the implication to individual verifiability was proven by first using that strong consistency and mb–BPRIV implies an ideal functionality (depending on the chosen recovery function) which in turn implies individual verifiability. It is however instructive to prove this relation directly, which also allows a comparison of the advantage.

We note that a privacy definition can only ever imply a weak kind of verifiability, since the adversary is not allowed to know the secret key. Normally, we want to ensure all types of verifiability even if there are inside attackers. There are several examples of e-voting systems not being secure when the adversary knows the election secret key. We thus label this as weak individual verifiability.

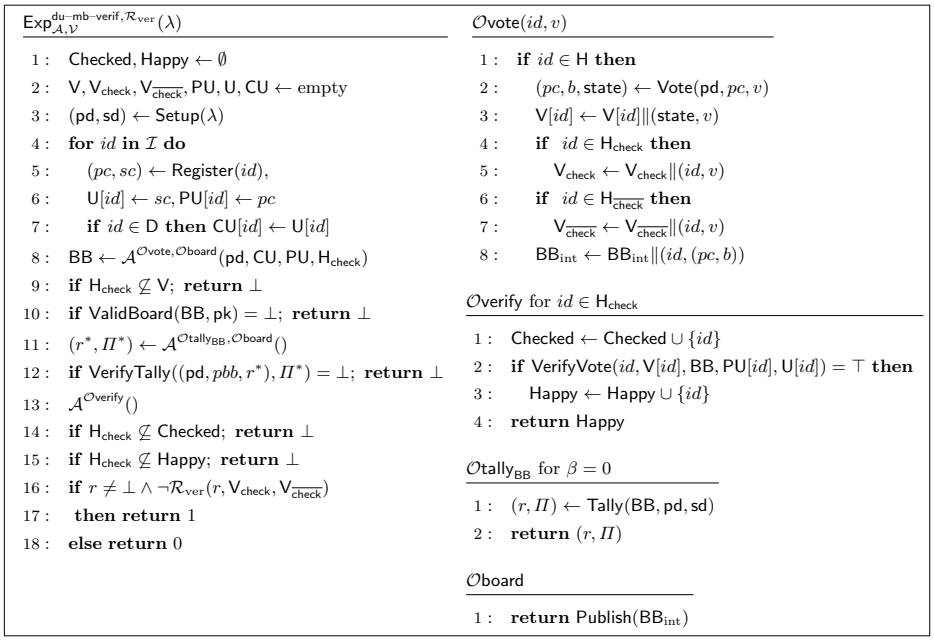


Fig. 9. Weak individual verifiability with post-tally verification for maliciously generated board and tally.

In Fig. 9 we define the experiment $\mathsf{Exp}^{\mathsf{du\text{-}mb\text{-}verif},\mathcal{R}_{\mathrm{ver}}}_{\mathcal{A},\mathcal{V}}(\lambda)$ for weak individual verifiability with post-tally verification as an updated and stronger version of the individual verifiability in [13]. We here use $\mathcal{O}\mathsf{board}$ to control the information the adversary gets from vote casting, just as in du–mb–BPRIV, whereas in [13] the adversary gets the output of the vote casting algorithm. $\mathsf{V}_{\mathsf{check}}$ and $\mathsf{V}_{\overline{\mathsf{check}}}$ are the list of identity-vote pairs (id, v) from the checking and non-checking honest voters, respectively. The relation $\mathcal{R}_{\mathrm{ver}}$ defines the property on the honestly cast ballots that the verifiability should protect.

The reason that our individual verifiability definition is stronger than [13] is that their definition assumes an honestly created tally, however, in our case the adversary can output the tally result and proof, but we will check whether the proof is valid using VerifyTally.

Definition 10 (Weak individual verifiability). *Let $\mathcal{V}$ be a voting system and $\mathcal{R}_{\mathrm{ver}}$ a relation. We say that that $\mathcal{V}$ fulfills* weak individual verifiability *w.r.t. $\mathcal{R}_{\mathrm{ver}}$ against a malicious board and tally if for any polynomial adversary $\mathcal{A}$ we have that $\Pr\left[\mathsf{Exp}^{\mathsf{du\text{-}mb\text{-}verif},\mathcal{R}_{\mathrm{ver}}}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right]$ is negligible in λ.*

Further, we need *strong consistency*. We here refer for App. A in [13] for the precise definition. However, what it says is that there exist extractors which extract an identity and a plaintext vote from a general ballot. The extractors are denoted $\mathsf{Extract}_{id}$ from the identity, which was already used in the recovery functions above, and $\mathsf{Extract}_v$ for

extracting the vote, which may use the secret election key in sd. When applied to honestly generated ballots the extractors are correct with overwhelming probability. In general, the extractors can return $\perp$ on invalid ballots. Further, with overwhelming probability, the tally of a bulletin board submitted by the adversary is the result function on the votes extracted from the ballots. That is, if the submitted bulletin board is of the form $[(pc_1, b_1), \ldots, (pc_k, b_k)]$ then if the board is valid we will with overwhelming probability have that the result is the result function computed on the extracted votes:

$$r = \rho(\mathsf{Extract}(\mathsf{sd}, \mathsf{PU}, pc_1, b_1), \ldots, \mathsf{Extract}(\mathsf{sd}, \mathsf{PU}, pc_k, b_k)),$$

where we have combined the two extractors into one.

The recovery function used in du–mb–BPRIV will decide which relation $\mathcal{R}_{\text{ver}}$ is secured in the weak individual verifiability. We will only focus on the two recovery functions above. Following [13], also for the notation, the relation corresponding to $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del,reorder,change}}$ is

$$\mathcal{R}_{\text{ver}}^{\mathsf{del,reorder,change}}(r, \mathsf{V}_{\mathsf{check}}, \mathsf{V}_{\overline{\mathsf{check}}}) = \\ \exists \mathsf{V}_a \approx \mathsf{V}_{\mathsf{check}}.\exists \mathsf{V}_c.(\forall (pc, b) \in \mathsf{V}_c.id \notin \mathsf{H}_{\mathsf{check}}) \wedge r = \rho(\mathsf{V}_a \| \mathsf{V}_c).$$

Here $\approx$ means that the list is the same up to ordering, i.e. they are the same in a set-sense. As discussed above, the recovery function does not recover the order for the checking voters. In words, each verifying voter will have one of their votes counted.

For $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del,reorder}}$ the relation is

$$\mathcal{R}_{\text{ver}}^{\mathsf{del,reorder}}(r, \mathsf{V}_{\mathsf{check}}, \mathsf{V}_{\overline{\mathsf{check}}}) = \\ \exists \mathsf{V}_a \approx \mathsf{V}_{\mathsf{check}}.\ \exists \mathsf{V}_b \sqsubseteq \mathsf{V}_{\overline{\mathsf{check}}}\ \exists \mathsf{V}_c.(\forall (pc, b) \in \mathsf{V}_c.id \in \mathsf{D}) \wedge r = \rho(\mathsf{V}_a \| \mathsf{V}_b \| \mathsf{V}_c).$$

Here $\sqsubseteq$ means inclusion in a set-sense. In words, the verifying voters will have one of their votes counted and the non-verifying voters can have their votes deleted, but not changed (in practice this would have to be ensured via authentication). In both cases, we assume that the result function is *stable* in the sense of [13], namely is independent of the ordering of votes, as long as the votes for each *id* appear in the same order.

We are now ready to state our theorem that du–mb–BPRIV and strong consistency implies weak individual verifiability with the corresponding relation function. Since we go directly from privacy to verifiability, and not via an ideal functionality as in [13], we can directly relate the adversarial advantages, and further, as discussed above, we descend to a stronger version of verifiability with non-honest tally. For the proof there is a technical obstacle in that the verification in the right world still uses the state from the left world for the post-tally verification, a fact that we will discuss in lesson 4 in the next section. This gives a degradation factor 1/2.

Theorem 4. *Let $\mathcal{V}$ be a strongly consistent voting system satisfying* du–mb–BPRIV *with respect to the recovery function* $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del,reorder}}$ *respectively* $\mathsf{Recover}_{\mathsf{U}}^{\mathsf{del,reorder,change}}$ *and let the result function be stable, then $\mathcal{V}$ satisfies weak individual verifiability w.r.t. to the relation $\mathcal{R}_{\text{ver}}^{\mathsf{del,reorder}}$ respectively $\mathcal{R}_{\text{ver}}^{\mathsf{del,reorder,change}}$. We here assume that these relations are computable in polynomial time. Given an adversary $\mathcal{A}$ with advantage* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{du\text{–}mb\text{–}verif}}(\lambda)$ *against*

weak verifiability, we can build an adversary $\mathcal{B}$ *with* $\mathsf{Adv}^{\mathsf{du\text{–}mb\text{–}bpriv}}_{\mathcal{B},\mathcal{V},\mathsf{Sim}}(\lambda) \geq \frac{1}{2}\mathsf{Adv}^{\mathsf{du\text{–}mb\text{–}verif}}_{\mathcal{A}}(\lambda)$ *against* du–mb–BPRIV.

Proof. Let $\mathcal{A}$ be an adversary against individual verifiability w.r.t. to the relation $\mathcal{R}^{\mathsf{del,reorder}}_{\mathrm{ver}}$ and advantage $\mathsf{Adv}^{\mathsf{du\text{–}mb\text{–}verif}}_{\mathcal{A}}(\lambda)$. We then take a challenge from du–mb–BPRIV and feed it to $\mathcal{A}$ against the verifiability experiment. When $\mathcal{A}$ calls $\mathcal{O}\mathsf{vote}(id, v)$ we call $\mathcal{O}\mathsf{vote}(id, v, v)$ in the privacy experiment, and we answer $\mathcal{A}$'s calls to $\mathcal{O}\mathsf{board}$ by calling it in the privacy experiment. For d^* we output a random value. In the same way we answer $\mathcal{O}\mathsf{tally}_{\mathsf{BB}}$ from the privacy experiment, and we get a result and proof from $\mathcal{A}$. If we are in the left world, VerifyTally will succeed with probability greater than $\mathsf{Adv}^{\mathsf{du\text{–}mb\text{–}verif}}_{\mathcal{A}}(\lambda)$. We can now again feed queries to $\mathcal{O}\mathsf{verify}$ from the privacy experiment. In the left world the two games are aligned and with probability $\mathsf{Adv}^{\mathsf{du\text{–}mb\text{–}verif}}_{\mathcal{A}}(\lambda)$ the individual verification will succeed, but the result will not fulfill the relation. On the other hand, if we are in the right world, the recovery function will make sure that the board always fulfils the relation: For $\mathsf{Recover}^{\mathsf{del,reorder,change}}_{\mathsf{U}}$ it will make sure that all votes from verifying voters will appear on BB' (but might be reordered). By strong consistency the tally result will thus contain the votes from the honest voters and fulfil the relation. For $\mathsf{Recover}^{\mathsf{del,reorder}}_{\mathsf{U}}$, the votes from the checking voters will likewise be added back in by the recovery function. For the ballots for the honest non-verifying voters, they will either be kept if they were untouched or thrown away if they were changed in any way. Again by strong consistency, the result will contain all votes from checking voters, and a subset from non-checking honest voters, and hence fulfil the relation. Since we assume that the relation can be computed in polynomial time, we can let the adversary $\mathcal{B}$ output 1 if the relation does not hold, and 0 if it holds. Since the recovery function changes the ballots to the left hand ballots, and we use the left state for verification, the VerifyVote algorithm will most likely work, however, this cannot be guaranteed. Hence we do not know in the right world how often the board will pass verification. However, when we reach a verified board, we can distinguish the two worlds using the relation, in all other cases the outputs are manifestly random. We thus have that

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{du\text{–}mb\text{–}bpriv}}_{\mathcal{B},\mathcal{V},\mathsf{Sim}}(\lambda) &= \Big|\Pr\left[\mathsf{Exp}^{\mathsf{du\text{–}mb\text{–}BPRIV,Recover},0}_{\mathcal{B},\mathcal{V},\mathsf{Sim}}(\lambda) = 1\right] \\
&\quad - \Pr\left[\mathsf{Exp}^{\mathsf{du\text{–}mb\text{–}BPRIV,Recover},1}_{\mathcal{B},\mathcal{V},\mathsf{Sim}}(\lambda) = 1\right]\Big| \\
&\geq \mathsf{Adv}^{\mathsf{du\text{–}mb\text{–}verif}}_{\mathcal{A}}(\lambda) + \left(1 - \mathsf{Adv}^{\mathsf{du\text{–}mb\text{–}verif}}_{\mathcal{A}}(\lambda)\right)\tfrac{1}{2} - \tfrac{1}{2} \\
&= \tfrac{1}{2}\mathsf{Adv}^{\mathsf{du\text{–}mb\text{–}verif}}_{\mathcal{A}}(\lambda)
\end{aligned}$$

up to negligible contributions.

We note that a weaker form of strong consistency would have been sufficient for this proof, but might be needed for more general recovery functions and relations.

We note that the fact that our proof of the implication from privacy to verifiability is direct means that it should be possible to formalise the proof in EASYCRYPT. This is in contrast to the proof in [13] which relies on the intermediate ideal functionality, which

would be much harder to formalise in EasyCrypt. We leave such a formalisation as future work.

7 Lessons learned

In this section we highlight and discuss a few important lessons that we learned in the process of formalising the du–mb–BPRIV security definition and the security proof of Selene in EasyCrypt. We believe that these lessons will be useful to develop and evaluate future efforts on formalising both security proofs for electronic voting protocols and other security proofs in EasyCrypt.

7.1 Lesson 1: Vacuity in adversary quantifications

EasyCrypt has proved useful in formalising—and sometimes spotting issues in—existing security proofs, related to standard security notions. This work and the series of work on the security of electronic voting that precedes it also make definitional contributions, which are developed in parallel with their formal counterparts.

In a setting where any change is costly, having definitions and proofs evolve together is a risky proposition, and can very easily lead to very complex proofs of vacuous results. Such issues can obviously arise, from stating trivially false axioms under which the results are proved to hold. But EasyCrypt proofs also offer less obvious pitfalls in the form of universal quantification over adversaries.

Our top-level results, expressed on paper, are simply of the form

$$\forall \mathcal{A}, \mathsf{Adv}(\mathcal{A}) \leq \mathsf{Adv}(\mathcal{B}(\mathcal{A})) + \varepsilon$$

In EasyCrypt, the universally-quantified adversary $\mathcal{A}$ above very often needs to be further restricted to:

1. Not share memory with other parts of the system—often the security experiment and related oracles; and
2. Terminate with probability 1 when run with access to oracles that terminate with probability 1.

The first restriction accounts for a convenience feature of EasyCrypt's memory model, which makes all global variables visible to all modules—an issue when trying to hide their value from an adversary, but a convenience when writing a reduction that peeks into an oracle's state.

The second restriction accounts for the fact that termination cannot be either assumed (because it is possible to write a non-terminating adversary) or discharged automatically. Making the assumption explicit where needed gives some flexibility in instantiating complex arguments. It is possible to write—in intermediate proof steps that hold regardless of the adversary's termination status—non-terminating adversaries (or adversaries whose termination is hard to prove, such as those that make use of rejection sampling). But in places where termination is needed, a termination proof must be provided.

As we discovered after completing our proof, it is very easy to state these restrictions too strongly—and sometimes strongly enough as to be vacuous.

More concretely, let us consider the case of some adversary $\mathcal{A}$ that expects oracle access to some oracle $\mathcal{O}$. A natural way of expressing restriction 2 above on some fixed adversary $\mathcal{A}$ would be as follows.

$$\forall \mathcal{O}. \Pr[\mathcal{O} : \top] = 1 \Rightarrow \Pr\left[\mathcal{A}^{\mathcal{O}} : \top\right] = 1$$

This natural expression is, however, too strong: it expects that $\mathcal{A}$ should terminate *even* if the oracles she can query can interfere with her memory (and even if she, in turn, can interfere with her oracles' internal memory). This would, pathologically, prevent a theorem proved under this restriction from applying to an adversary who calls its oracle inside a simple `while` loop—where the oracle could simply bump the loop index back every time it is called to prevent the adversary from terminating. The correct way of stating the restriction above in EasyCrypt is (in prettified syntax) as follows, taking care to prevent $\mathcal{O}$ from interfering with the execution of $\mathcal{A}$.

$$\forall \mathcal{O}\,\{-\mathcal{A}\}\,. \Pr[\mathcal{O} : \top] = 1 \Rightarrow \Pr\left[\mathcal{A}^{\mathcal{O}} : \top\right] = 1$$

Spotting such missing restrictions is not easy: our first proof fell prey to one of them. Such assumptions are often present in top-level statements, which are not immediately instantiated. As a consequence, the hypothesis is not discharged—which would reveal it as too strong—and simply remains as a "hidden axiom" which severely limits the applicability of the formal result.

This issue is one of a few issues with EasyCrypt's approachability to new users which may cause more than frustration, and we have fed it back to the development team. We will keep in touch with them to design and develop new ways of expressing adversary constraints that are less vacuity-prone.

7.2 Lesson 2: How to deal with random oracles in EasyCrypt

EasyCrypt already provides a library for working with Random Oracle Models (ROM) that formalizes *non-programmable eager and lazy* models, bounded eager and lazy models, and programmable lazy models. Moreover, they also provide a lemma that shows that *eager* and *lazy* ROM can be used interchangeable within a non-programmable model. This result proves to be invaluable, especially when dealing with modules that have access to a random oracle, like the non-malleable encryption scheme $E^{\mathcal{O}}$ we had to deal with in our formalisation and proofs.

As EasyCrypt does not allow one to write invariants using modules we had to introduce operators to capture the same behavior while taking care to model exactly how these modules interact with the random oracles. For example, on paper one may write the following invariant independent of the type of ROM $\mathcal{O}$ considered:

$$\forall (id, pc, c) \in \mathsf{BB}_0,\ \mathsf{V}[id].v = E^{\mathcal{O}}.\mathsf{dec}(\mathsf{sd}, id, pc, c),$$

to express that for any ballot (id, pc, c) that we have in BB_0 we store the plaintext v_0 in $\mathsf{V}[id].v$ when the adversary has called the oracle $\mathcal{O}\mathsf{voteLR}(id, v_0, v_1)$. However, to do this in EasyCrypt we had to introduce an operator `dec_cipher_vo` that is equivalent to the $E^{\mathcal{O}}.\mathsf{dec}$ call as long as the state (or map) of the random oracle $\mathcal{O}.m$ has not changed or $\mathcal{O}$ is an eager random oracle. Thus, we used the following invariant:

$$\forall (id, pc, c) \in BB. \\ \mathsf{V}[id] = \mathtt{dec\_cipher\_vo}(\mathsf{sd}, id, pc, c, \mathcal{O}.m).$$

As an additional benefit of this formalisation and using the non-programmable eager ROM, we could start using these type of invariants as soon as calls to $\mathcal{O}\mathsf{voteLR}(id, v_0, v_1)$ are made and reason on the outputs of $E^{\mathcal{O}}.\mathsf{enc}$ and maintain that invariant until we need to discharge it later in the tally phase.

Another important lesson was on understanding that e-voting protocols can use more than one ROM, and the adversary and proof strategy may rely on having different assumptions over them. This is where the lesson on ROM from [10] proved invaluable as we had to model the following ROM for our e-voting protocols Selene, MiniVoting and Belenios:

$\mathcal{H}$ - the ROM that the encryption scheme $\mathsf{E_v}$ used,
$\mathcal{G}$ - the ROM used by the zero-knowledge proof $\Sigma_{\mathcal{R}} = (\mathsf{P}, \mathsf{V})$ of correct tally, and
$\mathsf{Sim}.\mathcal{O}$ - the ROM defined by the simulator Sim.

This separation allowed to accurately model the cryptographic primitives that require specific ROM access, and to have efficient proof reduction steps, e.g., replacing the zero-knowledge proof system $\Sigma_{\mathcal{R}}$ withe the simulator Sim also implied replacing ROM $\mathcal{G}$ with the ROM of $\mathsf{Sim}.\mathcal{O}$. That meant, for example in the case of Selene going from this definition

$$\mathsf{Selene}(\mathsf{E}_\mathsf{v}^{\mathcal{H}}, \mathsf{E_t}, \Sigma_{\mathcal{R}}^{\mathcal{G}}, \mathsf{CP}, \mathsf{ValidInd}^{\mathcal{H}})$$

to this type of definition undetected by the adversary

$$\mathsf{Selene}(\mathsf{E}_\mathsf{v}^{\mathcal{H}}, \mathsf{E_t}, \mathsf{Sim}, \mathsf{CP}, \mathsf{ValidInd}^{\mathcal{H}}).$$

7.3 Lesson 3: Small changes lead to large changes

Proofs in EasyCrypt are tied to exact module and type formats, and even trivial changes can have drastic impact. One such example was when we adapted the proof of Selene to work for "Selene with signatures". More precisely, we enriched the ballot considered by Selene $b = (id, pc, c)$ to $b = (id, pc, c, s)$ by including a signature s verifiable using the public credential pc of the voter id providing the encryption c of the vote v. That meant, given the signature scheme S we changed the Selene from Definition 9 to

$$\mathrm{Selene}(\mathsf{E_v}, \mathsf{E_t}, \Sigma_{\mathcal{R}}, \mathsf{CP}, \mathsf{ValidInd}, \mathsf{S}),$$

and updating all previous steps to handle the signature. However, it still amounted to definitions and proof changes of *2500 line out of 7070 lines* for the Selene proof [4].

The main reason for this significant number of updated lines is due to the fact all EasyCrypt proofs are manually produced. EasyCrypt only provides automatic guarantees on the proof verification. For example, any modification in an invariant (independent of the scale, e.g. $b = (id, pc, c)$ to $b = (id, pc, c, s)$) leads to re-proving all statements involving that invariant.

Additionally, the experience of previous EasyCrypt development [10,11] has helped in avoiding making direct assumptions even when we were confident on the statements, e.g., non-malleability (or IND-1-CCA) over the entire ballot b compared to just (id, pc, c). As an alternation to that ballot form to $b = (id, pc, c, s)$ makes this malleable due to the presence of the signature.

7.4 Lesson 4: Split the voter state into two parts in the presence of a recovery algorithm

In Section 3 we briefly mention that we split the voter state into two different parts, $\mathsf{state}_{\mathsf{pre}}$ and $\mathsf{state}_{\mathsf{post}}$. We now discuss this further and elaborate on the reason why this is necessary.

First, recall that in the du–mb–BPRIV definition, the adversary gets access to a vote oracle $\mathcal{O}\mathsf{voteLR}$ when creating the ballot box. The vote oracle records, among other things, a voter state, which is used for verification later on. The state can for example be a ciphertext encrypting the voter's ballot (as in Labelled-MiniVoting) or a plaintext ballot (as in Selene). The reason why we need to split the state into two parts is partly due to the fact that we want to accommodate schemes where post-tally verification is not enforced (e.g. Labelled-MiniVoting) and schemes where verification can only happen after the tally has been computed (e.g. Selene), and partly due to the presence of the recovery algorithm.

For schemes where post-tally verification is not enforced, i.e. schemes where it is typically possible to do verification both before and after tally, the information needed to verify is typically a ciphertext, which the voter checks whether or not is included in the ballot box. In other words, the voter state is checked against some data that is available prior to tallying. On the other hand, in schemes where verification has to happen after tally, the verification check is typically performed on data that is produced by the tally algorithm (e.g. a list of plaintext ballots).

Now, one could imagine that it would be sufficient to only have one voter state, as this state would (for example) be a ciphertext in schemes where post-tally verification is not enforced. It would still be a ciphertext that is checked against some available data, independent of whether or not the voter actually performs the verification check before or after the tally. However, due to the recovery algorithm, it is indeed necessary to split the state into two parts.

[4] Our EasyCrypt formalization has 19340 lines of code with 1310 line for CORE (e.g., ElGamal, LPKE, proof system, hybrid argument), 7070 line for the Selene proof, 6860 lines for Selene with Signature proof, and 4100 lines for Belenios and MiniVoting proof.

The first part of the state, $\mathsf{state}_{\mathsf{pre}}$, should depend on the secret bit β. Since this part of the state is used in schemes where verification could occur before tallying, the voter should verify against the ballot box created by the adversary, both in the real world and in the fake world. For example, in Labelled-MiniVoting, if we are in the real world, the voter should check that the left side encrypted ballot b_0 is a member of the left side ballot box BB_0, and if we are in the fake world, the voter should check that the right side ballot b_1 is a member of the right side ballot box BB_1.

However, for schemes with post-tally verification, the situation is slightly different. Recall first how the recovery algorithm works, and what the goal of the adversary is. The adversary is tasked with distinguishing between two different worlds. In the real world, the adversary gets to see real ballots and the real result as well as a proof of correct tally. In the fake world, the adversary gets to see fake ballots, but he still gets to see the result as tallied on the real ballots, as well as a simulated proof of correct tally. As the adversary is allowed to tamper with the ballot box, the recovery algorithm is needed to decide which of the honest ballots to include in the tally. Thus, for schemes where post-tally verification is enforced, the voter should always verify against data as if she was in the real world. For this reason, the vote oracle should always record the second part of the state as if it was in the real world.

In summary, in the presence of a recovery algorithm, it is necessary to split the voter's state into two parts: one for where verification occurs on the bulletin board, and one for where verification occurs on data produced during the tally. As the state should depend on the secret bit β when verification happens with respect to information produced before tallying and it should be independent of β when verification happens with respect to information produced by the tally algorithm, the vote oracle records $(\mathsf{state}_{\mathsf{pre},0}, \mathsf{state}_{\mathsf{post},0})$ when $\beta = 0$, and $(\mathsf{state}_{\mathsf{pre},1}, \mathsf{state}_{\mathsf{post},0})$ when $\beta = 1$.

Note that we discuss state splitting only for electronic voting protocols. Examining the applicability of this approach to other kinds of protocols is out of scope for this paper, but an interesting area of future research.

8 Concluding Remarks and Future Work

In this work we presented a refined version of the mb–BPRIV privacy definition which we call delay-use malicious-ballotbox ballot privacy (du–mb–BPRIV). Our new definition allows the modeling of schemes (such as Selene) where verification occurs after tallying. The security claim is also more explicit. We formalised our new definition in the interactive theorem prover EasyCrypt and showed that labelled MiniVoting, Belenios and Selene all satisfy the definition. We also proved that MiniVoting and Belenios satisfy the original mb–BPRIV privacy definition. Furthermore, we have discussed the relation between du–mb–BPRIV and individual verifiability (and the corresponding relation for mb–BPRIV) and proved that a voting system satisfying du–mb–BPRIV and a property called strong consistency, also satisfies a form of individual verifiability which depends on the choice of recovery function. Finally, we have highlighted and discussed a few important lessons that we believe will be useful for future efforts on formalising electronic voting protocols and related security properties in EasyCrypt.

While we have encoded Selene correctly in what we firmly believe is the most appropriate privacy definition in literature, our work highlights certain defiances in privacy definitions which future work should address. The defiances are fairly deep and addressing them is far out of scope for this work. We will, however, briefly mention two principal defiances of mb–BPRIV and related definitions. First, the definition, while handling a malicious bulletin board, assumes honest setup. Secondly, du–mb–BPRIV and related definitions are highly calibrated to schemes where the auxiliary data produced by tally to be used in verification are zero-knowledge proofs. In particular, schemes like Selene which output trackers to use for verification are difficult to express in these definitions.

Further, the BPRIV style of definition implies certain restrictions. As an example, the adversary can only see the result and verification from the left side board which constrains the attacker model. In particular, this means that we cannot detect privacy attacks relying on inducing candidate-specific errors for an observed voter while giving the adversary access to whether the corresponding voter verification fails or not, see e.g. [18] for such a style of attack. Of course, such attacks can be ruled out by considering recovery functions preventing any changes to honestly cast votes as in [13], but it is not the case in general. An important line of future research is thus to find alternative definitions capturing both more general and more transparent attacker models, e.g. by decreasing the generality of the definition or to consider simulation based security.

Acknowledgment

We thank the anonymous reviewers at IEEE Computer Security Foundations Symposium for their helpful comments. This work was supported by the Luxembourg National Research Fund (FNR) and the Research Council of Norway (NFR) for the joint project SURCVS (FNR project ID 11747298, NFR project ID 275516), and by the FNR project STV (C18/IS/12685695/IS/STV/Ryan). Thomas Haines is the recipient of an Australian Research Council Australian Discovery Early Career Award (project number DE220100595). Constantin Cătălin Drăgan was supported by the EPSRC grant EP/W032473/1 (AP4L), and European Union's Horizon grants 101069688 (CONNECT) and 101070627 (REWIRE) - funded by UK government Horizon Europe guarantee and administered by UKRI under grants 10043743 (CONNECT) and 10043730 (REWIRE). Further, this work received funding from the France 2030 program managed by the French National Research Agency under grant agreement No. ANR-22-PECY-0006. We thank the (rest of the) EasyCrypt team for the continued development of the tool and its libraries.

References

1. Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. *EasyCrypt: A Tutorial*, pages 146–166. Springer International Publishing, Cham, 2014.
2. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

3. Mihir Bellare and Amit Sahai. Non-malleable encryption: Equivalence between two notions, and an indistinguishability-based characterization. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 519–536. Springer, Heidelberg, August 1999.
4. David Bernhard. *Zero-knowledge proofs in theory and practice*. PhD thesis, University of Bristol, 2014.
5. David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. SoK: A comprehensive analysis of game-based ballot privacy definitions. In *2015 IEEE Symposium on Security and Privacy*, pages 499–516. IEEE Computer Society Press, May 2015.
6. David Bernhard, Véronique Cortier, Olivier Pereira, Ben Smyth, and Bogdan Warinschi. Adapting helios for provable ballot privacy. In Vijay Atluri and Claudia Díaz, editors, *ESORICS 2011*, volume 6879 of *LNCS*, pages 335–354. Springer, Heidelberg, 2011.
7. David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 626–643. Springer, Heidelberg, December 2012.
8. Alessandro Bruni, Eva Drewsen, and Carsten Schürmann. Towards a mechanized proof of selene receipt-freeness and vote-privacy. In *International Joint Conference on Electronic Voting*, pages 110–126. Springer, 2017.
9. Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. BeleniosRF: A non-interactive receipt-free electronic voting scheme. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1614–1625. ACM Press, October 2016.
10. Véronique Cortier, Constantin Catalin Dragan, François Dupressoir, Benedikt Schmidt, Pierre-Yves Strub, and Bogdan Warinschi. Machine-checked proofs of privacy for electronic voting protocols. In *2017 IEEE Symposium on Security and Privacy*, pages 993–1008. IEEE Computer Society Press, May 2017.
11. Véronique Cortier, Constantin Catalin Dragan, François Dupressoir, and Bogdan Warinschi. Machine-checked proofs for electronic voting: Privacy and verifiability for belenios. In Steve Chong and Stephanie Delaune, editors, *CSF 2018 Computer Security Foundations Symposium*, pages 298–312. IEEE Computer Society Press, 2018.
12. Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election verifiability for helios under weaker trust assumptions. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS 2014, Part II*, volume 8713 of *LNCS*, pages 327–344. Springer, Heidelberg, September 2014.
13. Véronique Cortier, Joseph Lallemand, and Bogdan Warinschi. Fifty shades of ballot privacy: Privacy against a malicious board. In Limin Jia and Ralf Küsters, editors, *CSF 2020 Computer Security Foundations Symposium*, pages 17–32. IEEE Computer Society Press, 2020.
14. Constantin Cătălin Drăgan, François Dupressoir, Ehsan Estaji, Kristian Gjøsteen, Thomas Haines, Peter Y.A. Ryan, Peter B. Rønne, and Morten Rotvold Solberg. Machine-checked proofs of privacy against malicious boards for selene & co. In *2022 IEEE 35th Computer Security Foundations Symposium (CSF)*, pages 335–347, 2022.
15. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg, August 1984.
16. Thomas Haines, Rajeev Goré, and Bhavesh Sharma. Did you mix me? formally verifying verifiable mix nets in electronic voting. In *IEEE Symposium on Security and Privacy*, pages 1748–1765. IEEE, 2021.
17. Thomas Haines, Rajeev Goré, and Mukesh Tiwari. Verified verifiers for verifying elections. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 685–702. ACM Press, November 2019.
18. Steve Kremer and Peter B Rønne. To du or not to du: A security analysis of du-vote. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 473–486. IEEE, 2016.
19. Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. select: A lightweight verifiable remote voting system. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 341–354. IEEE, 2016.

20. Peter B Rønne, Peter YA Ryan, and Marie-Laure Zollinger. Electryo, in-person voting with transparent voter verifiability and eligibility verifiability. *arXiv preprint arXiv:2105.14783*, 2021.
21. Peter Y. A. Ryan, Peter B. Rønne, and Vincenzo Iovino. Selene: Voting with transparent verifiability and coercion-mitigation. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *LNCS*, pages 176–192. Springer, Heidelberg, February 2016.
22. Muntadher Sallal, Steve A. Schneider, Matthew Casey, Constantin Catalin Dragan, François Dupressoir, Luke Riley, Helen Treharne, Joe Wadsworth, and Phil Wright. VMV: augmenting an internet voting system with selene verifiability. *CoRR*, abs/1912.00288, 2019.
23. Victor Shoup. A proposal for an ISO standard for public key encryption. Cryptology ePrint Archive, Report 2001/112, 2001. `https://eprint.iacr.org/2001/112`.
24. Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `https://eprint.iacr.org/2004/332`.
25. Marie-Laure Zollinger, Peter B Rønne, and Peter YA Ryan. Short paper: Mechanized proofs of verifiability and privacy in a paper-based e-voting scheme. In *International Conference on Financial Cryptography and Data Security*, pages 310–318. Springer, 2020.

Paper III

Machine-Checked Proofs of Accountability: How to sElect who is to Blame

Constantin Cătălin Drăgan, François Dupressoir, Kristian Gjøsteen, Thomas Haines, Peter B. Rønne and Morten Rotvold Solberg

Published at the 28th European Symposium on Research in Computer Security, ESORICS 2023

Machine-Checked Proofs of Accountability: How to sElect who is to Blame

Constantin Cătălin Drăgan[1], François Dupressoir[2], Kristian Gjøsteen[3], Thomas Haines[4], Peter B. Rønne[5], and Morten Rotvold Solberg[3]

[1] University of Surrey, Guildford, United Kingdom c.dragan@surrey.ac.uk
[2] University of Bristol, Bristol, United Kingdom f.dupressoir@bristol.ac.uk
[3] Norwegian University of Science and Technology, Trondheim, Norway
{kristian.gjosteen,mosolb}@ntnu.no
[4] Australian National University, Canberra, Australia thomas.haines@anu.edu.au
[5] CNRS, LORIA, Université de Lorraine, Nancy, France peter.roenne@loria.fr

Abstract. Accountability is a critical requirement of any deployed voting system as it allows unequivocal identification of misbehaving parties, including authorities. In this paper, we propose the first game-based definition of accountability and demonstrate its usefulness by applying it to the sElect voting system (Küsters *et al.*, 2016) – a voting system that relies on no other cryptographic primitives than digital signatures and public key encryption.
We strengthen our contribution by proving accountability for sElect in the EasyCrypt proof assistant. As part of this, we identify a few errors in the proof for sElect as presented by Küsters *et al.* (2016) for their definition of accountability.
Finally, we reinforce the known relation between accountability and verifiability, and show that it is still maintained by our new game-based definition of accountability.

1 Introduction

A system is accountable if, when something goes wrong, it is possible to judge who is responsible based on evidence provided by the system participants. For a voting system, this means that if we do not accept the outcome of an election, the honest parties should be able to produce evidence that pinpoints who is to blame, in the sense that they have not followed the protocol. This is in principle trivial for some voting systems, such as the Helios voting system where each party proves their correct behaviour using zero knowledge arguments. This is, however, not trivial for every reasonable voting system, in particular voting systems with complex ballot submission procedures, such as the Swiss Post voting system [20]; in the Swiss Post case the system involves a complicated protocol between half a dozen participants to decide if a ballot was cast by a valid voter and well-formed and hence should be counted.

The sElect voting system [17] is an interesting case for accountability. Unlike Helios, the system does not use any advanced cryptography, relying entirely on secure public key encryption. The system uses nested public key encryption to allow a very simple mixnet decryption. The voter creates a nested encryption of their ballot and a random check value, each layer encrypted with a mix server public key. Each mix server decrypts one layer of encryption, sorting the result lexicographically to effect mixing. The last mix server simply outputs decrypted ballots, together with the voter-specific check value. Voters verify that

their ballot is included in the count by checking that the ballot appears together with the voter's check value.

Informally, the sElect system is accountable because voters can reveal the randomness used in the nested encryption, thereby enabling tracing of the encrypted ballot through the mixnet, which will pinpoint which mix server did not correctly decrypt.

Accountability might seem to be a fairly simple notion, but it is technically difficult to find a definition that both captures accountability and is easy to work with. This can be seen from the fact that no definition of accountability seems to have been broadly accepted in the community. Also, when Küsters *et al.* [17] apply the definition from [18] to sElect, there are a number of errors in the result they claim; we will discuss these in greater length in Sec. 1.2. These errors suggest that the existing accountability definitions are hard to work with. In other words, there is a need for a workable general definition of accountability.

The simplicity of sElect comes at a cost, which is that the system is only private for voters that accept the election outcome. This problem can be mitigated using the final cryptosystem trick from [12]; with this trick, "a sender first encrypts her message under the "final" public key and uses this encrypted message as an input to the protocol as described so far. This innermost encryption layer is jointly decrypted only if the protocol does not abort. If the protocol does abort, only the encrypted values are revealed and privacy is protected by the final layer of encryption." However, using this mitigation in sElect would require the voters' devices to check the mix before the result is decrypted which substantially complicates the protocol and delays the tally result, which would be unacceptable in most cases.

Privacy is of course essential for voting systems, but we note that we are not studying privacy in this paper, only accountability, since the privacy of sElect is well-understood.

1.1 Our Contribution

This paper contains two main contributions: The first game-based definition of accountability, and a proof of accountability for the sElect [17] voting system. A variant of the latter proof has been formalised in the EasyCrypt [3] proof assistant.

This game-based definition is significant because this style of definitions are often easier to understand and work with. For security proofs, ease of understanding and use is a significant factor in getting things right and later verifying that things are indeed correct. Further, it allows us to use existing tools for game-based proofs, specifically EasyCrypt, to formally verify security.

The accountability proof for sElect is significant, first because it demonstrates that our new definition of accountability works. Second, the sElect voting system is interesting because it is so simple, requiring no other primitives than digital signatures and public key encryption. Proving security properties for interesting voting systems is intrinsically interesting.

As we have seen, informal arguments sometimes contain errors. A proof formalised in EasyCrypt is significant, in that it ensures that we have no errors in arguments, making the overall security proof easier to verify.

In addition to the main contribution, we also make the relation between verifiability and accountability precise, in the sense that accountability implies verifiability (when suitably defined). This is a significant result, suggesting that future system designers should focus on achieving accountability.

1.2 Related Work

To the best of our knowledge, no game-based definition of accountability has been proposed earlier. However, several definitions of accountability (for general security protocols, not only electronic voting protocols) have been proposed in the symbolic model. Bruni *et al.* [5] propose a general definition amenable to automated verification. Künnemann *et al.* [16] give a definition of accountability in the decentralised-adversary setting, in which single protocol parties can choose to deviate from the protocol, while Künnemann *et al.* [15] give a definition in the single-adversary setting, where all deviating parties are controlled by a single, centralised adversary. Morio & Künnemann [19] combine the definition from [15] with the notion of case tests to extend the definition's applicability to protocols with an unbounded number of participants. Furthermore Küsters *et al.* [18] put forward quantitative measures of accountability both in the symbolic and computational model. Similar for all these definitions is that they clearly distinguish between *dishonest* parties and *misbehaving* parties. Even though a party is dishonest (controlled by an adversary), it does not necessarily deviate from the protocol and cause a violation of the security goal. In such cases, the party is not misbehaving and should not be held accountable for anything.

While no game-based definition of accountability has been proposed, game-based definitions for other voting-related security properties do exist in the literature. Some of these definitions have also been formalised in the proof assistant EasyCrypt [3], with related machine-checked proofs for a variety of voting protocols. Cortier *et al.* [6] formalise a game-based definition of ballot-privacy called BPRIV [4] in EasyCrypt and give a machine-checked proof that Labelled-MiniVoting [6] and several hundred variants of Helios [2] satisfy this notion of ballot privacy. Cortier *et al.* [7] build on work from [6] and also formalise a game-based definition of verifiability in EasyCrypt, in addition to giving a machine-checked proof that Belenios [9] is ballot-private and verifiable. Drăgan *et al.* [10] formalise the mb-BPRIV ballot privacy definition [8] in EasyCrypt and give a machine-checked proof that Labelled-MiniVoting and Belenios satisfy this definition. They also propose a new game-based ballot privacy definition called du-mb-BPRIV, which is applicable to schemes where voter verification can or must happen after the election result has been computed, and give a machine-checked proof that Labelled-MiniVoting, Belenios and Selene [21] all satisfy this definition.

Problems in the Küsters et al. [17] Accountability Proof. In carefully analysing sElect we became aware of two errors in the Accountability theorem which we detail below; to our knowledge these errors have not previously been documented in the literature. There is a significant complexity in the parameters used in Theorem 3 (Accountability) in the full

version of sElect [17], but fortunately this is largely orthogonal to the points we need to discuss.

Ballot Stuffing The goal for which accountability is proven (see Definition 1 in [17]) somewhat implicitly requires that the multiset containing the election result contains at most n elements, where n is the number of voters. However, no argument is made in the proof that the judge will hold anyone accountable if there are more than n ballots. Both the pen-and-paper description and the implementation of sElect omit any checks which would catch the addition of ballots by the mix servers, and it seems that the authentication server could also stuff ballots though this would be more involved. As significant as this vulnerability is, it is easy to fix and we have done so in the version of sElect we prove accountability for.

Honest Nonce Collision A described above, the goal the theorem aims for uses multisets and hence if multiple honest voters vote for the same choice we expect to see at least that many copies of the choice in the output; this is somewhat complicated in sElect by the augmentation of voter choices with nonces. The mechanism which sElect uses to detect ballots being removed relies on the plaintext encrypted by the honest voters being unique; however, this does not happen when the nonces and choices of the honest voters collide. The chance of such collision should appear in the security bound of accountability for sElect unless it is explicitly negligible in the security parameter. Strangely, sElect will drop these votes even with no adversarial involvement since the protocol specifies that the final mix server (like all others) should filter its output for duplicates. We note that the probability of collisions does appear in the verifiability theorem and proof.

2 Game Based Accountability

In this section we present our game based definition of accountability for electronic voting protocols, and we start by presenting the parties and their roles.

2.1 Parties

We consider the following parties and their role in the election process.

Voting Authority VA that sets up the election process, generates public parameters, defines voter eligibility, etc. The election secret keys are managed by separate parties, called decryption and mixnet authorities.

Decryption and Mixnet Authorities $\mathsf{MS}_i(\mathsf{msk}_i, \mathsf{mpk}_i)$ that manage together the decryption process, and each party has been allocated a part of the decryption key/election secret key. This is typically done by decryption or re-encryption together with shuffling of ballots/votes to break the link between recorded ballots and the votes.

Authentication Server $\mathsf{AS}(\mathsf{ask}, \mathsf{apk})$ issues confirmation tokens that ballots were recorded as cast, typically under the form of signatures.

Judge J assigns blame to misbehaving parties based on publicly available data and voter reported evidence. We model this by having an algorithm Judge.

Voters id_i that cast their vote v_i. The voting process is facilitated by a voting supporting device VSD that builds ballots for the user and then casts them.

Bulletin board BB stores publicly verifiable information relevant to an election, e.g. ballots, mixnet outcomes, and election outcome. The bulletin board may be divided into subcomponents such as a list of submitted ballots or the election outcome.

2.2 Voting System

The election process is defined by the following tuple of algorithms.

$\mathsf{Setup}()$**:** This algorithm produces the public election data pd and the secret election data sd. This is done by interaction between VA, $\mathsf{MS}_0, \ldots, \mathsf{MS}_k$, and potentially AS.

$\mathsf{Vote}(\mathsf{pd}, v)$**:** This algorithm builds the ballot b based on the vote v and public data pd. Additionally, it produces the internal state of the voter, state, to facilitate the verification process later.

$\mathsf{ASCreate}(\mathsf{ask}, b)$**:** This algorithm produces a token σ that the ballot b has been received and accepted by the authentication server $\mathsf{AS}(\mathsf{ask}, \mathsf{apk})$.

$\mathsf{ASVerify}(\mathsf{pd}, b, \sigma)$**:** This algorithm verifies if the token σ is valid for the ballot b and public data pd.

$\mathsf{Tally}(\mathsf{sd}, \mathsf{BB})$**:** This algorithm models the sequence of calls to the mixnet and decryption authorities to produce the election outcome.

$\mathsf{VSDVerify}((\mathsf{state}, b, \sigma), \mathsf{pd}, \mathsf{BB})$**:** Checks if the system has followed the required processes for this user's vote and ballot, and it outputs $\perp$ if no misbehaving party has been identified. Otherwise, it returns the misbehaving party and the corresponding evidence.

$\mathsf{Judge}(\mathsf{pd}, \mathsf{BB}, \mathsf{E})$**:** It checks that the publicly available data is valid with respect to some predefined metrics and against the list of evidence E. It returns the error symbol $\perp$ if no misbehaving party has been identified; otherwise, it outputs the misbehaving party B. As all checks can be replicated publicly, it does not need to return evidence.

The following algorithm is unbounded and is hence only part of the experiment and will not be run during an election.

$\mathsf{Bad}(\mathsf{pd}, \mathsf{BB}, \mathsf{E}, \mathsf{V})$**:** This unbounded algorithm serves to provide a ground truth of which parties misbehaved. By the requirements of our definition, it always blames a party when the election result does not reflect the votes of voters - given the public data pd, the bulletin board BB, the list of evidence E, and the internal state of honest voters V. Optionally, it may detect whether a party has deviated from the protocol in a way which does not change the election result. It should never blame an honest party.

2.3 Accountability

We consider that the adversary has full control over all parties introduced in Section 2.1, except the Judge. The adversary can also incorporate their own evidence to Judge. If a party deviates from the protocol steps, then that party becomes *misbehaving* and could

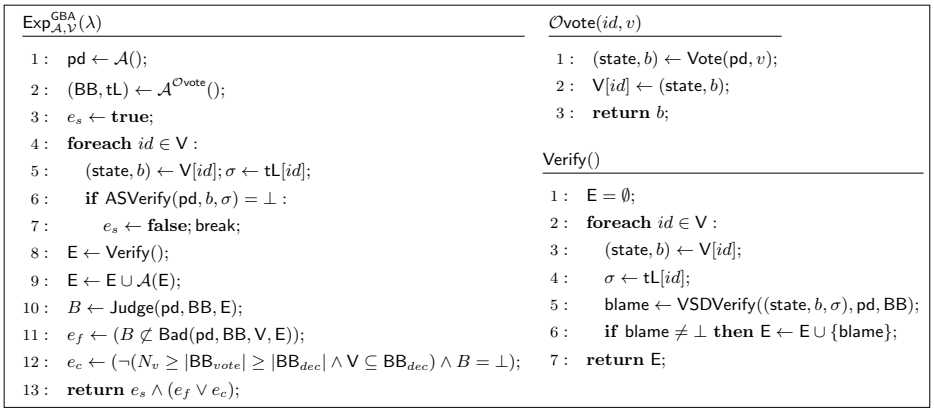


Fig. 1. The new game-based security notion for accountability. BB_{vote} and BB_{dec} denote different subcomponents of the bulletin board, respectively ballots submitted through $\mathcal{O}\mathsf{vote}$ and information produced by tallying.

be identified and blamed by either Judge or Bad. However, if the party follows exactly the protocol steps we call that party *behaving*, independent of them being honest or dishonest (corrupted by the adversary).

The formal accountability definition is found in Fig. 1. The first step for the adversary is to start the election process and provide the public data pd. Then, the adversary runs the voting and tally phase and commits to the current state of the bulletin board BB, together with a list of all authentication tokens tL. During the voting phase, the adversary can make use of the oracle $\mathcal{O}\mathsf{vote}$ to replicate the behavior of behaving voters and build their ballot b and internal state state.

To capture the natural behavior of behaving and honest voters that would check their tokens and complain before the tally is provided, we incorporate an automatic lose condition for the adversary if any of the provided tokens for those voters cannot be verified by ASVerify; this approach is similar to that taken by Küsters *et al.* [17] in their (non-game based) accountability proof of sElect.

Verification is done as a two-stage process, first by collecting evidence E from all honest voters (those that used $\mathcal{O}\mathsf{vote}$ and whose internal states are stored in V) and from the adversary $\mathcal{A}(\mathsf{E})$; and secondary by calling Judge to check the public data together with that evidence. The Judge is responsible for providing either a misbehaving party B if there is enough evidence to do so, or $\perp$ if nothing could be detected. The adversary wins if one of the following happens:

Fairness: Judge wrongly blames a party B when it did not misbehave. This is checked by running the Bad algorithm to identify all misbehaving parties in the system and check whether B has been included, or

Completeness: the result is not consistent with the honest votes but no one is blamed. This is done by Judge producing $\perp$ when an honest voter's ballot was dropped, or

there are more submitted ballots than there are voters, or more ballots in the election outcome than the number of ballots that were cast in the first place.

Definition 1 (Game-Based Accountability). *Let $\mathcal{V}$ be a voting system as defined in this section. We say that $\mathcal{V}$ satisfies* GBA *if for any efficient adversary $\mathcal{A}$ their advantage is negligible in λ:*

$$\mathsf{Adv}^{\mathsf{gba}}_{\mathcal{A},\mathcal{V}}(\lambda) = \Pr\left[\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right].$$

Our adversary winning conditions aligns our definition with the one from Küsters *et al.* [17], such that any voting system that satisfies our accountability definition will also satisfy the one by Küsters *et al.* [17] (with the goal used for sElect), with some possible caveats about the casting of schemes between the two definitions. We expand on this in Section 4.

3 sElect

In this section we introduce sElect [17] using the format of Section 2; we focus on the elements with are important for accountability and omit some orthogonal details. The formal description is in Figure 2. We denote by $\mathsf{BB}_{vote}, \mathsf{BB}_{mix}$ and BB_{dec} the different subcomponents of BB: respectively the submitted ballots, data produced by the mixnet and the election outcome, which is a list of plaintext votes.

3.1 Cryptographic primitives

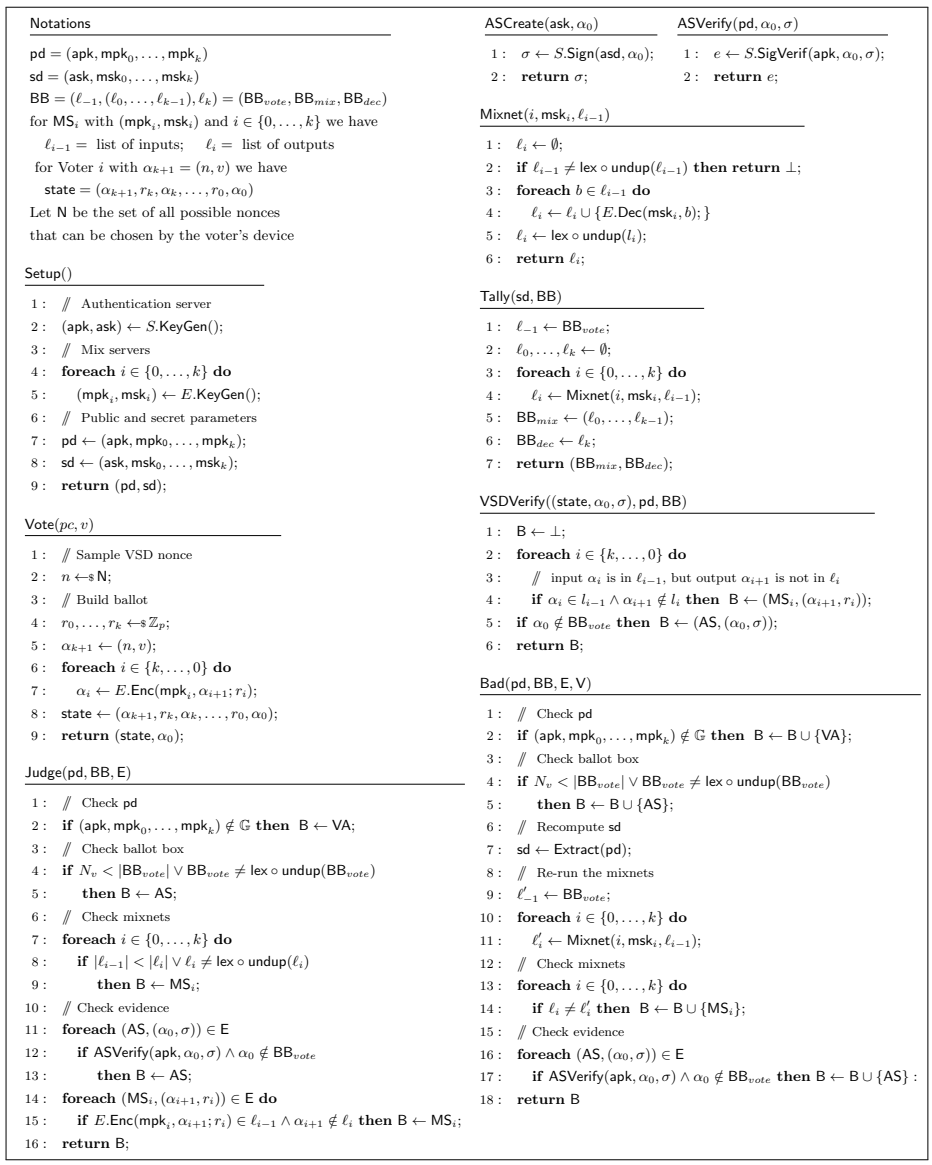
The voting system sElect relies on two basic cryptographic primitives: an IND-CCA2 encryption scheme $E = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ and an EU-CMA signature scheme $S = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{SigVerif})$. To make the encryption scheme compatible with decryption mixnets it needs to allow nested encryptions. Typically, this is done through a hybrid cryptosystem [1], by combining hybrid ElGamal and AES in a suitable mode such that each encryption contains an AES encryption of the message under a random AES key and an ElGamal encryption of the AES key.

As part of the formalisation for the shuffling done by the mixnet servers $\mathsf{MS}_0, \ldots, \mathsf{MS}_k$, we consider the operators lex for sorting a list in lexicographic order, and undup for removing duplicates. We additionally have that the authentication authority AS runs S.

3.2 sElect Algorithms

$\mathsf{Setup}()$**:** The authentication server key pair $(\mathsf{apk}, \mathsf{ask})$ is generated by $S.\mathsf{KeyGen}$, and the mixnet servers key pairs $(\mathsf{mpk}_i, \mathsf{msk}_i)$ are computed by $E.\mathsf{KeyGen}$. The algorithm returns the public data $\mathsf{pd} = (\mathsf{apk}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_k)$ and secret data $\mathsf{sd} = (\mathsf{ask}, \mathsf{msk}_0, \ldots, \mathsf{msk}_k)$.

$\mathsf{Vote}(\mathsf{pd}, v)$**:** The algorithm samples a supporting device verification code n such that it can be used later by the voter to ensure their vote was counted. sElect also considers a short voter verification code n_{voter} that has no security assumptions (for accountability);



Notations

$\mathsf{pd} = (\mathsf{apk}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_k)$
$\mathsf{sd} = (\mathsf{ask}, \mathsf{msk}_0, \ldots, \mathsf{msk}_k)$
$\mathsf{BB} = (\ell_{-1}, (\ell_0, \ldots, \ell_{k-1}), \ell_k) = (\mathsf{BB}_{vote}, \mathsf{BB}_{mix}, \mathsf{BB}_{dec})$
for MS_i with $(\mathsf{mpk}_i, \mathsf{msk}_i)$ and $i \in \{0, \ldots, k\}$ we have
$\ell_{i-1} =$ list of inputs; $\ell_i =$ list of outputs
for Voter i with $\alpha_{k+1} = (n, v)$ we have
$\mathsf{state} = (\alpha_{k+1}, r_k, \alpha_k, \ldots, r_0, \alpha_0)$
Let N be the set of all possible nonces
that can be chosen by the voter's device

Setup()

// Authentication server
$(\mathsf{apk}, \mathsf{ask}) \leftarrow S.\mathsf{KeyGen}()$;
// Mix servers
foreach $i \in \{0, \ldots, k\}$ **do**
  $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow E.\mathsf{KeyGen}()$;
// Public and secret parameters
$\mathsf{pd} \leftarrow (\mathsf{apk}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_k)$;
$\mathsf{sd} \leftarrow (\mathsf{ask}, \mathsf{msk}_0, \ldots, \mathsf{msk}_k)$;
return $(\mathsf{pd}, \mathsf{sd})$;

Vote(pc, v)

// Sample VSD nonce
$n \leftarrow\!\!\$\ \mathsf{N}$;
// Build ballot
$r_0, \ldots, r_k \leftarrow\!\!\$\ \mathbb{Z}_p$;
$\alpha_{k+1} \leftarrow (n, v)$;
foreach $i \in \{k, \ldots, 0\}$ **do**
  $\alpha_i \leftarrow E.\mathsf{Enc}(\mathsf{mpk}_i, \alpha_{i+1}; r_i)$;
$\mathsf{state} \leftarrow (\alpha_{k+1}, r_k, \alpha_k, \ldots, r_0, \alpha_0)$;
return $(\mathsf{state}, \alpha_0)$;

Judge$(\mathsf{pd}, \mathsf{BB}, \mathsf{E})$

// Check pd
if $(\mathsf{apk}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_k) \notin \mathbb{G}$ **then** $\mathsf{B} \leftarrow \mathsf{VA}$;
// Check ballot box
if $N_v < |\mathsf{BB}_{vote}| \vee \mathsf{BB}_{vote} \neq \mathsf{lex} \circ \mathsf{undup}(\mathsf{BB}_{vote})$
  then $\mathsf{B} \leftarrow \mathsf{AS}$;
// Check mixnets
foreach $i \in \{0, \ldots, k\}$ **do**
  if $|\ell_{i-1}| < |\ell_i| \vee \ell_i \neq \mathsf{lex} \circ \mathsf{undup}(\ell_i)$
    then $\mathsf{B} \leftarrow \mathsf{MS}_i$;
// Check evidence
foreach $(\mathsf{AS}, (\alpha_0, \sigma)) \in \mathsf{E}$
  if $\mathsf{ASVerify}(\mathsf{apk}, \alpha_0, \sigma) \wedge \alpha_0 \notin \mathsf{BB}_{vote}$
    then $\mathsf{B} \leftarrow \mathsf{AS}$;
foreach $(\mathsf{MS}_i, (\alpha_{i+1}, r_i)) \in \mathsf{E}$ **do**
  if $E.\mathsf{Enc}(\mathsf{mpk}_i, \alpha_{i+1}; r_i) \in \ell_{i-1} \wedge \alpha_{i+1} \notin \ell_i$ **then** $\mathsf{B} \leftarrow \mathsf{MS}_i$;
return B;

ASCreate(ask, α_0)

$\sigma \leftarrow S.\mathsf{Sign}(\mathsf{asd}, \alpha_0)$;
return σ;

ASVerify$(\mathsf{pd}, \alpha_0, \sigma)$

$e \leftarrow S.\mathsf{SigVerif}(\mathsf{apk}, \alpha_0, \sigma)$;
return e;

Mixnet$(i, \mathsf{msk}_i, \ell_{i-1})$

$\ell_i \leftarrow \emptyset$;
if $\ell_{i-1} \neq \mathsf{lex} \circ \mathsf{undup}(\ell_{i-1})$ **then return** $\perp$;
foreach $b \in \ell_{i-1}$ **do**
  $\ell_i \leftarrow \ell_i \cup \{E.\mathsf{Dec}(\mathsf{msk}_i, b);\}$
$\ell_i \leftarrow \mathsf{lex} \circ \mathsf{undup}(l_i)$;
return ℓ_i;

Tally$(\mathsf{sd}, \mathsf{BB})$

$\ell_{-1} \leftarrow \mathsf{BB}_{vote}$;
$\ell_0, \ldots, \ell_k \leftarrow \emptyset$;
foreach $i \in \{0, \ldots, k\}$ **do**
  $\ell_i \leftarrow \mathsf{Mixnet}(i, \mathsf{msk}_i, \ell_{i-1})$;
$\mathsf{BB}_{mix} \leftarrow (\ell_0, \ldots, \ell_{k-1})$;
$\mathsf{BB}_{dec} \leftarrow \ell_k$;
return $(\mathsf{BB}_{mix}, \mathsf{BB}_{dec})$;

VSDVerify$((\mathsf{state}, \alpha_0, \sigma), \mathsf{pd}, \mathsf{BB})$

$\mathsf{B} \leftarrow \perp$;
foreach $i \in \{k, \ldots, 0\}$ **do**
  // input α_i is in ℓ_{i-1}, but output α_{i+1} is not in ℓ_i
  if $\alpha_i \in l_{i-1} \wedge \alpha_{i+1} \notin l_i$ **then** $\mathsf{B} \leftarrow (\mathsf{MS}_i, (\alpha_{i+1}, r_i))$;
if $\alpha_0 \notin \mathsf{BB}_{vote}$ **then** $\mathsf{B} \leftarrow (\mathsf{AS}, (\alpha_0, \sigma))$;
return B;

Bad$(\mathsf{pd}, \mathsf{BB}, \mathsf{E}, \mathsf{V})$

// Check pd
if $(\mathsf{apk}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_k) \notin \mathbb{G}$ **then** $\mathsf{B} \leftarrow \mathsf{B} \cup \{\mathsf{VA}\}$;
// Check ballot box
if $N_v < |\mathsf{BB}_{vote}| \vee \mathsf{BB}_{vote} \neq \mathsf{lex} \circ \mathsf{undup}(\mathsf{BB}_{vote})$
  then $\mathsf{B} \leftarrow \mathsf{B} \cup \{\mathsf{AS}\}$;
// Recompute sd
$\mathsf{sd} \leftarrow \mathsf{Extract}(\mathsf{pd})$;
// Re-run the mixnets
$\ell'_{-1} \leftarrow \mathsf{BB}_{vote}$;
foreach $i \in \{0, \ldots, k\}$ **do**
  $\ell'_i \leftarrow \mathsf{Mixnet}(i, \mathsf{msk}_i, \ell_{i-1})$;
// Check mixnets
foreach $i \in \{0, \ldots, k\}$ **do**
  if $\ell_i \neq \ell'_i$ **then** $\mathsf{B} \leftarrow \mathsf{B} \cup \{\mathsf{MS}_i\}$;
// Check evidence
foreach $(\mathsf{AS}, (\alpha_0, \sigma)) \in \mathsf{E}$
  if $\mathsf{ASVerify}(\mathsf{apk}, \alpha_0, \sigma) \wedge \alpha_0 \notin \mathsf{BB}_{vote}$ **then** $\mathsf{B} \leftarrow \mathsf{B} \cup \{\mathsf{AS}\}$:
return B

Fig. 2. Algorithms defining the sElect voting scheme with an IND-CCA2 secure public key encryption system $E = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ and an EU-CMA secure signature scheme $S = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{SigVerif})$.

we have included that code together with the voter's candidate choices c as part of the vote $v = (n_{voter}, c)$. The algorithm sets $\alpha_{k+1} = (n, v)$ and uses a series of encryptions $\alpha_i \leftarrow \mathsf{Enc}(\mathsf{mpk}_i, \alpha_{i+1}, r_i)$ to build the ballot α_0 and internal state $\mathsf{state} = (\alpha_{k+1}, \alpha_k, r_k, \ldots, \alpha_0, r_0)$, given some random coins $r_0, \ldots, r_k \in \mathbb{Z}_p$.

$\mathsf{ASCreate}(\mathsf{asd}, \alpha_0)$**:** It returns a signature σ by calling $S.\mathsf{Sign}$ over the ballot α_0.

$\mathsf{ASVerify}(\mathsf{pd}, \alpha_0, \sigma)$**:** This algorithm calls $S.\mathsf{SigVerif}$ to check if the signature σ is valid for the ballot α_0.

$\mathsf{Tally}(\mathsf{sd}, \mathsf{BB})$**:** Given the ballot box $\ell_{-1} = \mathsf{BB}_{vote}$, this algorithm runs each mixnet MS_i over ℓ_{i-1} to produce ℓ_i, for $i \in \{0, \ldots, k\}$. Each mixnet MS_i, ensures first that the inputs are in lexicographic order and contain no duplicates, before decrypting all ciphertexts received as inputs, and finally outputting them in lexicographic order and without duplicates. The last mixnet server produces the election outcome $\mathsf{BB}_{dec} = \ell_k$. The algorithm returns the mixing info $\mathsf{BB}_{mix} = (\ell_0, \ldots, \ell_{k-1})$ and election outcome BB_{dec}.

$\mathsf{VSDVerify}((\mathsf{state}, \alpha_0, \sigma), \mathsf{pd}, \mathsf{BB})$**:** Voters check the output of each mixnet server by using their internal state $(\alpha_{k+1}, \alpha_k, r_k, \ldots, \alpha_0, r_0)$. The voter blames a mixnet server MS_i if they see that their ciphertext α_{i+1} is in the input list of that server, but the ciphertext α_i is not in the output list. Recall that α_i has been created by encrypting α_{i+1} under that server's public key mpk_i: $\alpha_i \leftarrow \mathsf{Enc}(\mathsf{mpk}_i, \alpha_{i+1}, r_i)$; thus, (α_{i+1}, r_i) can be used as evidence of misbehaviour of MS_i. The voter also checks that their ballot α_0 has been included in the ballot box BB_{vote}, and blames the authentication server AS if that has not happened, using the signature σ the user received during voting as evidence. This step can be done at any point in the election if one considers an ideal bulletin board, or at the end of an election under weaker trust assumptions over the bulletin board [11].

$\mathsf{Judge}(\mathsf{pd}, \mathsf{BB}, \mathsf{E})$**:** This algorithm does an initial round of checks over the public data before evaluating the collected evidence E. The verification of public data consists of

- Ensuring that the public data is valid - that is, the public keys are group elements. If this is not true, then the voting authority VA is blamed as it allowed the election to run.
- Checking that the size of the ballot box does not exceed the number of voters and that the ballot box has been ordered lexicographically and duplicates have been removed. Otherwise, the authentication server AS is blamed.
- Checking that each mixnet server output is in lexicographic order and has no duplicates, and that the size of the output list does not exceed the size of the input list. If these properties do not hold for mixnet server MS_i then the algorithm blames this mixnet server.

Once all the public data has been verified, the algorithm looks at the evidence collected by voters from their $\mathsf{VSDVerify}$ algorithm:

- Evidence (α_0, σ) against AS. If the evidence contains a valid signature σ for a ballot α_0 not in the ballot box, then the authentication server AS is blamed.
- Evidence (α_{i+1}, r_i) against MS_i. If the evidence shows that $\alpha_i \leftarrow \mathsf{Enc}(\mathsf{mpk}_i, \alpha_{i+1}, r_i)$ is in the input list of this server, but α_{i+1} is not in the output, then this mixnet server is blamed.

$\mathsf{Bad}(\mathsf{pd}, \mathsf{BB}, \mathsf{E}, \mathsf{V})$**:** This algorithm uses a computationally unbounded algorithm $\mathsf{sd} \leftarrow \mathsf{Extract}(\mathsf{pd})$ to obtain the secret keys of all authorities sd from their public data pd; similar to the vote extraction algorithm from [13] and [14]. $\mathsf{Extract}$ will never fail to return something as it will see any bitstring in the public data as a group element. However, it may not produce meaningful data or the real secret keys if these do not exist.

Bad uses the secret data from $\mathsf{Extract}$ to re-run the election tally and perform all verification steps to identify misbehaving parties. It looks at the validity of the public data pd and ballot box BB_{vote} using the same methods employed by Judge. Then, it re-creates for each mixnet server MS_i its estimated output ℓ'_i and blames that party if their estimated output ℓ'_i is different from the declared output ℓ_i. This type of check already includes the checks on the evidence submitted by voters against mixnet servers. Finally, it performs the checks on the evidence against the authentication server AS.

3.3 EasyCrypt Proof

Informally, we prove that the probability that the adversary is able to produce valid public data, a valid bulletin board and valid signatures, while at the same time violating either fairness or completeness, is negligible. We assume throughout the proof that the public key encryption scheme used to encrypt and decrypt ballots is perfectly correct, i.e. if we let $E = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be the (IND-CCA2 secure) PKE used in sElect, then we assume that for all key pairs $(\mathsf{pk}, \mathsf{sk})$ output by KeyGen and for all plaintexts m in the message space, we have $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$. As we assume that sElect is implemented with hybrid encryption of ElGamal and AES (cf. Section 3.1), this assumption holds. Under this assumption, the probability that the adversary violates the fairness aspect of accountability is in fact 0. The probability that the adversary violates the completeness aspect of accountability, is related to whether or not *nonce collisions* occur, i.e. whether or not the devices of two or more honest voters sample the same nonce.

Theorem 1. *Let* $\mathsf{sElect}(E, S)$ *be defined as in Figure 2 for an IND-CCA2 encryption scheme* E *and an EU-CMA signature scheme* S*. Then, for all PPT adversaries* $\mathcal{A}$ *against* GBA*, we have*

$$\mathsf{Adv}^{\mathsf{gba}}_{\mathcal{A},\mathsf{sElect}}(\lambda) \leq \Pr[Col]\,,$$

where Col is the event that a collision occurs in the nonces chosen by the voters' devices.

The proof sketch can be found in App. A.

Differences between our paper proof and EasyCrypt *proof.* The main difference in the above proof and the proof formalised in EasyCrypt[1] is that in EasyCrypt we let the adversary choose both the plaintext vote and the verification nonce and compress this into a single plaintext. Under the assumption that the choices made by the adversary are unique, this allows us to use sets rather than multisets in EasyCrypt which is technically

[1] The EasyCrypt code can be accessed from `https://github.com/mortensol/acc-select`

easier. In some sense this is however also a stronger result than above since it proves accountability even in the case where the nonces are adversarially chosen, but still unique. What is not proven in EasyCrypt is the probability of a collision happening, but for uniform distributions of nonces this is the well-known birthday paradox which is not interesting for the present paper to verify in EasyCrypt. Finally, keeping a general collision probability for the full plaintext consisting of device-generated nonce, voter-chosen nonce and plaintext vote is more general, and cannot be assumed to be uniformly random in practice, but can be bounded by the birthday probability on the device-generated nonces.

4 Relation to the Küsters et al. Definition

In this section we relate the above presented definition of accountability, GBA, to the one by Küsters et al. [18], which we denote Acc_{KTV}. More precisely, we sketch a proof that for the class of voting schemes expressible in our definition, if they satisfy GBA for a certain definition of Bad then they must be accountable under Acc_{KTV} with a standard goal.

Consider a voting scheme as defined in Section 2, consisting of a voting authority VA, decryption authorities DA, mixnet authorities MS, authentication server AS, voters id_i with voter supporting devices VSD_i, and bulletin board BB. We assume there are authenticated channels from the VSDs to the AS. We assume that each VSD has one authenticated and one anonymous channel to the BB. We assume that all communication is authenticated with signatures with the exception of the anonymous channel and for simplicity omit the description of this occurring from the exposition below.

4.1 Modeling

A voting scheme of this kind can be modeled in the framework of [18] in a straightforward way as a protocol $\mathcal{P}(n, m, q, \mu, p^{verif}_{voter}, p^{verif}_{abst})$. We refer to [18] for the notation used. We denote by n the number of voters and supporting devices, by m the number of mix servers, by q the number of decryption servers. By μ we denote the probability distribution on the set of candidates/choices, including abstention. We denote by p^{verif}_{voter} and p^{verif}_{abst} the probability that the voting voter will verify and an absenting voter will verify respectively.[2]

We define Φ_k as the accountability property consisting of the constraints:

$$\chi_i \rightarrow dis(id_i) \vee dis(AS), \quad \chi'_i \rightarrow dis(id_i) \vee dis(AS)$$
$$\neg\gamma_k \wedge \neg\chi \rightarrow dis(\mathsf{VA})|dis(\mathsf{AS})|dis(\mathsf{DA}_i)^q_{i=1}|dis(\mathsf{MS}_j)^m_{j=1}$$

where

- γ_k contains all runs of the protocol where at most n votes are in the result and where at most k of the honest votes are not included in the result. See [17] for a formal definition and discussion of this goal.
- χ_i contains all the runs of $\mathcal{P}$ where the voter i complains they did not get a receipt.
- χ'_i contains all the runs of $\mathcal{P}$ where the voter i complains they did not vote but a vote was cast on their behalf.
- χ contains the union of all runs in χ_i and χ'_i for all $i \in [1, ..., n]$

[2] Absenting voters verify that their identifier is not included on the list published by the AS.

4.2 Result

Let Bad be defined as follows: Bad returns all parties whose output is not in the co-domain of the honest algorithms. When parties are called multiple times on different algorithms and pass states, we take the co-domain over all possible states consistent with their early public output.

Let the Judge_{KTV} algorithm for Acc_{KTV} in [18] be constructed as follows:

(J1) first it runs Judge (from our definition) and if this outputs blame, then Judge_{KTV} blames the party returned by Judge.

(J2) If no valid complaints were made by the voters causing blame, the judge checks the complaints posted by the voters. If there is any such complaint then Judge_{KTV} blames (disjunctively) both the party accused and the voter accusing.

Definition 2 (Voter Verification Correct). *For a scheme π we say that it is voter verification correct if for all runs of the protocol the party blamed by* VSDVerify *is in the set output by* Bad *or it blames the* AS *after receiving an invalid confirmation.*

Theorem 2 (GBA implies Acc_{KTV}). *Let the judge Judge_{KTV} and algorithm* Bad *be defined as above. Then for any scheme which has* GBA *and voter verification correctness, Judge_{KTV} ensures $(\Phi_k, \delta^k(p^{verif}_{voter}, p^{verif}_{abst}))$-accountability for $\mathcal{P}(n, m, q, \mu, p^{verif}_{voter}, p^{verif}_{abst})$, where*

$$\delta^k(p^{verif}_{voter}, p^{verif}_{abst}) = (1 - \min(p^{verif}_{voter}, p^{verif}_{abst}))^{k+1}.$$

Due to space constraints, we detail this in App. B. The proof relies on analysing fairness and completeness for the two definitions.

5 Verifiability

In this section we show that our definition of accountability implies verifiability; a relation already shown in the framework of Küsters et al [18]. To prove this implication here, we introduce a new game-based definition of verifiability, that we formalize via the experiment $\mathsf{Exp}^{Ver}_{\mathcal{A}}(\lambda)$ in Fig. 3.[3] Our definition of verifiability ensures individual verifiability and no ballot stuffing during tally, and is appropriate for lightweight voting systems like sElect. Our definition is modular, and can be enhanced to model stronger notions of verifiability (e.g., universal verifiability or no ballot stuffing at submission time); however, to achieve them voting systems will require heavier cryptographic primitives, likes zero-knowledge proofs for correct tallying or shuffling.

We consider $\mathcal{I}$ the set of eligible voter IDs, and we introduce algorithm VoterVerif that enables voters to verify their vote. We keep track of voters that successfully verified using the set Checked and we raise the flag Complain when verification fails. The adversary can choose which voters verify via the oracle $\mathcal{O}\mathsf{Verify}(id)$. Additionally, the adversary uses the vote oracle $\mathcal{O}\mathsf{Vote}$ to model honest voters (re-)casting their votes; we focus on the last vote counts policy, but this can easily be generalized for any policies.

[3] In the game, we use the notation "Require" for **if** $\cdots$ **else return** $\perp$.

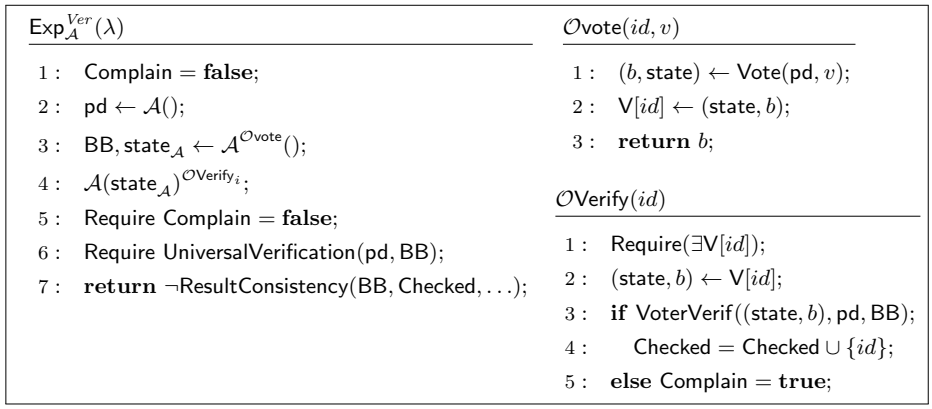


Fig. 3. Verifiability assuming uncorrupted vote-casting.

The adversary also controls the bulletin board BB, however anyone can perform UniversalVerification(pd, BB) to universally verify this state. We further use ResultConsistency(BB, Checked, ...) to model the consistency relations on the bulletin board, and can depend on the different subcomponents of BB: list of submitted ballots $\mathsf{BB}|_{submit}$, the election result $\mathsf{BB}|_{res}$ and extra info $\mathsf{BB}|_{extra}$.

Consider the election result function $\rho : \mathsf{Cand}^* \mapsto \mathsf{Res}$ as a symmetric function from the set of plaintext votes, chosen from the space of candidates Cand, to a given result set Res. Using $\mathsf{V}[S]$ the corresponding list of plaintext votes from the vote oracle, we model

- Individual Verifiability: Intuitively this should ensure that the verified votes are all included in the tally. Using the verification oracles $\mathcal{O}\mathsf{Verify}_i, i = 1, \ldots, k$ we denote the successful verifiers Checked. The constraint from ResultConsistency is $\exists v_1, \ldots, v_i \in \mathsf{Cand}, i + |\mathsf{Checked}| \leq |\mathcal{I}|$:

$$\rho(v_1, \ldots, v_i, \mathsf{V}[\mathsf{Checked}]) = \mathsf{BB}|_{res}$$

  where we have slightly abused notation for readability. We have included a constraint on the number of malicious votes since if the result function allows cancelling votes the inclusion of the honest votes would make little sense if the adversary can add malicious votes arbitrarily.
- No Ballot Stuffing at Tally Time: $|\mathcal{I}| \geq |\mathsf{BB}|_{submit}|$ and $\exists i \leq |\mathsf{BB}|_{submit}|$ $\exists v_1, \ldots, v_i \in \mathsf{Cand} : \rho(v_1, \ldots, v_i) = \mathsf{BB}|_{res}$, i.e. there is at most as many submitted ballots as eligible voters and the result is consistent with a number of votes that is less than or equal to the submitted ballots.

In the case of schemes where all the decrypted votes are displayed individually in $\mathsf{BB}|_{res}$, especially this holds for the mixnet-tally schemes, the slightly stronger statement can be made that

$$|\mathcal{I}| \geq |\mathsf{BB}|_{submit}| \geq |\mathsf{BB}|_{res}| \wedge \mathsf{V}[\mathsf{Checked}] \subseteq_{ms} \mathsf{BB}|_{res}\,, \tag{1}$$

where we use $\mathsf{V}[\mathsf{Checked}]$ and $\mathsf{BB}|_{res}$ as multisets.

We define verifiability given a chosen ResultConsistency if any efficient adversary has negligible advantage in $\mathsf{Exp}^{Ver}_{\mathcal{A}}(\lambda)$. In particular, we define verifiability for voting systems with the result being the plaintext votes as:

Definition 3. *We say that a voting system $\mathcal{V}$, with result function being the set of votes, satisfies individual verifiability and no ballot stuffing at tally time if for any efficient adversary $\mathcal{A}$ their advantage $\mathsf{Adv}^{ver}_{\mathcal{A},\mathcal{V}}(\lambda) = \mathsf{Exp}^{Ver}_{\mathcal{A},\mathcal{V}}(\lambda)$ is negligible in λ, where ResultConsistency checks Eq. 1.*

We note that there are some verifiability properties that sElect does not fulfill but could be easily captured by the ResultConsistency or separate games, namely

- Tally Uniqueness: The adversary cannot produce two boards both satisfying UniversalVerification and individual verifications but with different tally results and having the same submitted ballots $\mathsf{BB}|_{submit}$.
- Universal Verifiability: Here ResultConsistency requires that the result is the same as the result from votes extracted from the valid ballots in $\mathsf{BB}|_{submit}$ given only that the board satisfies $\mathsf{UniversalVerification}(\mathsf{pd}, \mathsf{BB})$.

5.1 Accountability implies Verifiability

We will now prove that the GBA accountability definition implies verifiability for individual verifiability and no ballot stuffing as defined in Def. 3. However, in order to do so, we need to relate the Judge and the VSDVerify algorithms used in $\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathcal{V}}(\lambda)$ with the algorithms UniversalVerification and VoterVerif used in $\mathsf{Exp}^{Ver}_{\mathcal{A},\mathcal{V}}(\lambda)$. Especially, the verifiability definition does not consider the authentication server AS and its signatures, since it is not relevant for defining verifiability. To this end we make the following definition for a voting system $\mathcal{V}$ fitting both the accountability and the verifiability framework:

Definition 4. *We call a voting system $\mathcal{V}$* accountability-verifiability-correct *if the signature part for AS is an independent part that can be removed to give a reduced system valid for the verifiability framework, or correspondingly added. Further, the Judge will never output blame if all verification checks by the verifying voters using VSDVerify does not output blame and $\mathsf{UniversalVerification} = \top$. Further, $\mathsf{VSDVerify}((\mathsf{state}, b, \sigma), \mathsf{pd}, \mathsf{BB}_{vote}, \mathsf{BB}_{mix}, \mathsf{BB}_{dec})$ will not output blame if $\mathsf{ASVerify}(\mathsf{pd}, b, \sigma) = \top$ and $\mathsf{VoterVerif}((\mathsf{state}, b), \mathsf{pd}, \mathsf{BB}) = \top$.*

Theorem 3. *Given an accountability-verifiability-correct voting system, then accountability as defined in Def. 1 implies individual verifiability and no ballot stuffing at tally time as defined in Def. 3 assuming the AS signature scheme is perfectly correct and we have a constant number of voters. More precisely for any efficient adversary $\mathcal{A}$ against $\mathsf{Exp}^{Ver}_{\mathcal{A},\mathcal{V}_r}(\lambda)$ with advantage $\mathsf{Adv}^{ver}_{\mathcal{A},\mathcal{V}_r}(\lambda)$ in the reduced system $\mathcal{V}_r$ without signatures, we can construct an adversary $\mathcal{B}$ against $\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathcal{V}}(\lambda)$ with advantage at least $\frac{1}{2^{|\mathcal{I}|}}\mathsf{Adv}^{ver}_{\mathcal{A},\mathcal{V}_r}(\lambda)$.*

Due to space constraints the full proof is in App. C.

It follows from Thm. 1 and Thm. 3 that sElect fulfills individual verifiability and no ballot stuffing at tally time as defined in Def. 3.

6 Concluding Remarks

We study notions of accountability for electronic voting, and produce the first *game-based* notion of accountability for mix-based electronic voting schemes. We relate our notion to Küsters et al's quantitative notion, arguing that they coincide at the extremes of the parameter range.

We demonstrate the value of such a game-based notion by formalising it in EasyCrypt, and produce a machine-checked proof of accountability—as we define it—for Küsters et al.'s sElect protocol, discussing issues with previous accountability results for sElect as we go. Finally, we use our new game-based definition of accountability to study the relationship between accountability, verifiability, demonstrating in particular that accountability implies verifiability.

Generalisation beyond sElect We framed our discussions, and our definitions, with sElect. However, our definitions would also somewhat trivially apply to other voting schemes. In particular, as mentioned in the introduction, any scheme making judicious use of sound zero-knowledge proofs for verifiability can be trivially argued to be accountable: an adversary who is able to break accountability with sound zero-knowledge proofs does so either by breaking soundness of the zero-knowledge proof systems, or by breaking accountability of a scheme in which verification for the zero-knowledge proofs is idealised to reject any proof that was not produced as is by the prover—relying then only on the correctness of the encryption scheme as in our sElect proof. Although this argument is easy to make on paper, formalising it in EasyCrypt on existing formal definitions for Helios (for example) would involve effort incommensurate to its scientific value as part of this specific paper.

Beyond Accountability Capturing accountability as a game-based notion is not just useful to allow a more precise analysis of accountability. By doing so, we hope to open the way to the study of privacy and security properties of voting schemes *with dispute resolution*. Formally taking into account dispute resolution requires a precise understanding of the individual and overall guarantees offered by verifiability in terms of the accuracy of the election result.

References

1. Abdalla, M., Bellare, M., Rogaway, P.: DHIES: An encryption scheme based on the Diffie-Hellman problem. Contributions to IEEE P1363a (Sep 1998)
2. Adida, B.: Helios: Web-based open-audit voting. In: van Oorschot, P.C. (ed.) USENIX Security 2008. pp. 335–348. USENIX Association (Jul / Aug 2008)
3. Barthe, G., Dupressoir, F., Grégoire, B., Kunz, C., Schmidt, B., Strub, P.Y.: EasyCrypt: A Tutorial, pp. 146–166. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-10082-1˙6, https://doi.org/10.1007/978-3-319-10082-1_6
4. Bernhard, D., Cortier, V., Galindo, D., Pereira, O., Warinschi, B.: A comprehensive analysis of game-based ballot privacy definitions. Cryptology ePrint Archive, Report 2015/255 (2015), https://eprint.iacr.org/2015/255
5. Bruni, A., Giustolisi, R., Schürmann, C.: Automated analysis of accountability. In: Nguyen, P.Q., Zhou, J. (eds.) ISC 2017. LNCS, vol. 10599, pp. 417–434. Springer, Heidelberg (Nov 2017)
6. Cortier, V., Dragan, C.C., Dupressoir, F., Schmidt, B., Strub, P.Y., Warinschi, B.: Machine-checked proofs of privacy for electronic voting protocols. In: 2017 IEEE Symposium on Security and Privacy. pp. 993–1008. IEEE Computer Society Press (May 2017). https://doi.org/10.1109/SP.2017.28

7. Cortier, V., Dragan, C.C., Dupressoir, F., Warinschi, B.: Machine-checked proofs for electronic voting: Privacy and verifiability for belenios. In: Chong, S., Delaune, S. (eds.) CSF 2018 Computer Security Foundations Symposium. pp. 298–312. IEEE Computer Society Press (2018). https://doi.org/10.1109/CSF.2018.00029
8. Cortier, V., Lallemand, J., Warinschi, B.: Fifty shades of ballot privacy: Privacy against a malicious board. In: Jia, L., Küsters, R. (eds.) CSF 2020 Computer Security Foundations Symposium. pp. 17–32. IEEE Computer Society Press (2020). https://doi.org/10.1109/CSF49147.2020.00010
9. Cortier, V., Gaudry, P., Glondu, S.: Belenios: A Simple Private and Verifiable Electronic Voting System, pp. 214–238 (04 2019). https://doi.org/10.1007/978-3-030-19052-1˙14
10. Drăgan, C.C., Dupressoir, F., Estaji, E., Gjøsteen, K., Haines, T., Ryan, P.Y., Rønne, P.B., Solberg, M.R.: Machine-checked proofs of privacy against malicious boards for selene & co. In: 2022 IEEE 35th Computer Security Foundations Symposium (CSF). pp. 335–347 (2022). https://doi.org/10.1109/CSF54842.2022.9919663
11. Hirschi, L., Schmid, L., Basin, D.A.: Fixing the achilles heel of E-voting: The bulletin board. In: Küsters, R., Naumann, D. (eds.) CSF 2021 Computer Security Foundations Symposium. pp. 1–17. IEEE Computer Society Press (2021). https://doi.org/10.1109/CSF51468.2021.00016
12. Khazaei, S., Moran, T., Wikström, D.: A mix-net from any CCA2 secure cryptosystem. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 607–625. Springer, Heidelberg (Dec 2012). https://doi.org/10.1007/978-3-642-34961-4˙37
13. Kiayias, A., Zacharias, T., Zhang, B.: End-to-end verifiable elections in the standard model. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 468–498. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46803-6˙16
14. Kiayias, A., Zacharias, T., Zhang, B.: Ceremonies for end-to-end verifiable elections. In: Fehr, S. (ed.) PKC 2017, Part II. LNCS, vol. 10175, pp. 305–334. Springer, Heidelberg (Mar 2017). https://doi.org/10.1007/978-3-662-54388-7˙11
15. Künnemann, R., Esiyok, I., Backes, M.: Automated verification of accountability in security protocols. In: Delaune, S., Jia, L. (eds.) CSF 2019 Computer Security Foundations Symposium. pp. 397–413. IEEE Computer Society Press (2019). https://doi.org/10.1109/CSF.2019.00034
16. Künnemann, R., Garg, D., Backes, M.: Accountability in the decentralised-adversary setting. In: Küsters, R., Naumann, D. (eds.) CSF 2021 Computer Security Foundations Symposium. pp. 1–16. IEEE Computer Society Press (2021). https://doi.org/10.1109/CSF51468.2021.00007
17. Küsters, R., Müller, J., Scapin, E., Truderung, T.: sElect: A lightweight verifiable remote voting system. In: Hicks, M., Köpf, B. (eds.) CSF 2016 Computer Security Foundations Symposium. pp. 341–354. IEEE Computer Society Press (2016). https://doi.org/10.1109/CSF.2016.31
18. Küsters, R., Truderung, T., Vogt, A.: Accountability: definition and relationship to verifiability. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) ACM CCS 2010. pp. 526–535. ACM Press (Oct 2010). https://doi.org/10.1145/1866307.1866366
19. Morio, K., Künnemann, R.: Verifying accountability for unbounded sets of participants. In: Küsters, R., Naumann, D. (eds.) CSF 2021 Computer Security Foundations Symposium. pp. 1–16. IEEE Computer Society Press (2021). https://doi.org/10.1109/CSF51468.2021.00032
20. Post, S.: Swiss post voting system. https://gitlab.com/swisspost-evoting (2022)
21. 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.S., Brenner, M., Rohloff, K. (eds.) FC 2016 Workshops. LNCS, vol. 9604, pp. 176–192. Springer, Heidelberg (Feb 2016). https://doi.org/10.1007/978-3-662-53357-4˙12

Appendix A Sketch of Proof of Theorem 1

We now sketch the proof of Theorem 1. We begin by defining two new games: a fairness game G_f and a completeness game G_c. These games are almost identical to the original security game, with the exception that in G_f, we remove the variable e_c from the experiment and only consider the fairness aspect of accountability, while in G_c, we remove

the variable e_f and only consider the completeness aspect of accountability. Let E_f resp. E_c be the event that the game G_f resp. G_c returns 1. It is straightforward to see that $\Pr\left[\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathsf{sElect}}(\lambda) = 1\right] \leq \Pr[E_f] + \Pr[E_c]$. Thus, the adversary has two possibilities to win. Either Judge has blamed an innocent party, or it has blamed no one, but the result is inconsistent with the honest votes. We analyze the fairness and completeness aspects separately, and argue that the adversary has zero probability of winning the fairness game and negligible probability of winning the completeness game.

We begin with fairness. We will consider each way in which the judge may blame a party and show that it will never blame a party that did not misbehave. Recall the various checks performed by the judge: The judge first checks that the public keys used by the authentication server and the mix servers are valid. If not, it will blame the voting authority as it allowed the election to run with invalid keys. Note that Bad will also blame the voting authority if the keys are invalid (but not otherwise), meaning that the voting authority will only be blamed by the judge if it indeed misbehaved. As invalid keys will result in the voting authority being blamed by both the judge and by Bad, we assume for the remainder of the proof that all the public keys are valid.

The judge then checks that the bulletin board BB_{vote} is valid, i.e. that it contains at most N_v elements and that its contents are in lexicographic order and duplicate-free. If this check fails, the judge blames the authentication server. If this is the case, the authentication server will also be blamed by Bad, ensuring that if AS is blamed for producing an invalid board, it must indeed have misbehaved. Next, the judge checks that the output of each mix server contains at most as many elements as in its received input and that the output of each mix server is duplicate free and in lexicographic order. Since an honest mixer filters out duplicates and sorts the output list, it will always pass this check.

The judge then checks, for all ciphertext and signature pairs in the evidence list whether or not there is a ballot with a valid signature that is not present on BB_{vote}. Since an honest authentication server only authenticates the first ballot from each voter, and posts all these on the bulletin board, it will always pass this check. Note that if a dishonest voter blames the authentication server with valid evidence, the Bad algorithm will also blame the authentication server, and thus the judge will not blame the authentication server unless it is also blamed by Bad. Finally, Judge checks, for any triple $(\mathsf{mpk}_i, \alpha_{i+1}, r_i)$, whether or not $\mathsf{Enc}(\mathsf{mpk}_i, \alpha_{i+1}; r_i)$ is in the input to the ith mix server, but α_{i+1} is not in its output. Since the encryption system is correct, $\mathsf{Enc}(\mathsf{mpk}_i, \alpha_{i+1}; r_i)$ will decrypt to α_{i+1} and since an honest mix server does not remove any ciphertexts other than duplicates, it will always pass this check. In summary, Judge will never blame an honestly behaving party, and thus, the adversary has zero probability of winning the fairness game.

We now move on to completeness and bound the adversarial advantage in the completeness game, i.e. that if extra ballots are added or honest voters' ballots are dropped, the judge will, with overwhelming probability, hold someone accountable. Fairness ensures that the blamed party actually misbehaved.

We begin with the first criterion for completeness, i.e. that the number of ballots on BB_{vote} is not greater than the number of eligible voters. This follows from the second check of the Judge algorithm, where it checks if the bulletin board is valid. The second criterion

(that the number of votes on BB_{dec} is not greater than the number of cast ballots) follows from the judge checking that the output of each mix server contains at most as many elements as its input.

Now consider the criterion that says that all honest votes are in the multiset of votes output by the last mix server (i.e. BB_{dec}). Every honest voter checks, using $\mathsf{VSDVerify}$, that their ballot appears in BB_{vote}. If not there, they output the token σ given to them by the authentication server. This, in turn, causes the authentication server to be blamed by the judge. If AS was not blamed, we know that all honest ballots were present in BB_{vote}. If any honest ballot is dropped by one of the mix servers, this will be detected by $\mathsf{VSDVerify}$, which will output some evidence that this mix server misbehaved, which in turn causes this mix server to be blamed by the judge.

Now, the adversary has one possibility of winning the completeness game, namely if two (or more) voters have cast the same vote, and their sampled nonces happen to be equal. In this case, the adversary may drop all but one of these ballots without it being detected. To analyze this situation, we slightly modify the completeness game. We call the new game G'_c and let E'_c be the probability that G'_c returns 1. The difference from G_c to G'_c is that in G'_c, we keep track of the nonces that are sampled when the adversary calls the vote oracle, and only sample new nonces from the set of nonces that have not been used earlier. The two games are equivalent unless there is a collision in the first game, hence $|\Pr[E_c] - \Pr[E'_c]| \leq \Pr[Col]$.

In G'_c, as there are no collisions in the nonces, any ballot that is dropped by the adversary will be detected by $\mathsf{VSDVerify}$, which in turns causes the judge to blame the misbehaving party. In other words, in G'_c, the adversary will have zero probability of winning, so $\Pr[E'_c] = 0$. Thus, the probability that the adversary wins the completeness game is bounded by $\Pr[Col]$. As the adversary has zero probability of winning the fairness game, and the probability of winning the accountability game is bounded by the sum of winning the fairness game and the completeness game, we arrive at the conclusion of Theorem 1 that the advantage is bounded by the collision probability. By the birthday paradox the collission probability is bounded by $\frac{q_v(q_v-1)}{2\cdot|\mathsf{N}|}$, where q_v is the number of queries vote oracle queries and N is the nonce space.

Appendix B Proof for Theorem 2

The proof of the theorem follows from analyzing Fairness and Completeness.

Lemma 1 (Fairness). *The judge J is computationally fair in $\mathcal{P}(n, m, \mu, p^{verif}_{voter}, p^{verif}_{abst})$.*

Proof. The proof is essentially the same as for sElect in [17] for the voting phase but relies on GBA in the mixing and decryption phases.

Consider what happens if the voter makes a complaint and the judge blames both the party accused and the voter (J2). Since the bulletin board is honest and the channel is authenticated the voter must really have made the complaint. There are two cases. If the voter is dishonest the verdict is clearly true. If the voter is honest, the correctness of the

verdict follows from the voter verification correctness of the protocol either because the person it blamed has misbehaved or because the authentication server did not send a valid confirmation. (J2) covers both the case where the voter's ballot is dropped and when it is added.

Case (J1) is covered by GBA. Since the scheme has GBA it follows that the adversary cannot make either of these conditions trigger when the party ran its honest program, otherwise GBA would not hold.

Lemma 2 (Completeness). *For every instance* π *of* $\mathcal{P}(n, m, \mu, p_{voter}^{verif}, p_{abst}^{verif})$*, we have*

$$Pr\left[\pi(1^l) \rightarrow \neg(J : \Phi_k)\right] \leq \delta^k(p_{voter}^{verif}, p_{abst}^{verif}) = \left(1 - \min\left(p_{voter}^{verif}, p_{abst}^{verif}\right)\right)^{k+1}$$

with overwhelming probability as a function of l*.*

Again, the proof is essentially the same as for sElect in the voting phase but relies on GBA in the mixing and decryption phases. We need to show that the following probabilities are bounded for every i:

a) $Pr\left[\pi(1^l) \rightarrow (\chi_i \wedge \neg dis(v_i) \wedge \neg dis(AS))\right]$,
b) $Pr\left[\pi(1^l) \rightarrow (\chi_i' \wedge \neg dis(v_i) \wedge \neg dis(AS))\right]$,
c) $Pr\left[\pi(1^l) \rightarrow (\neg\gamma_k \wedge \neg\chi \rightarrow dis(\mathsf{VA})|dis(\mathsf{AS})|dis(\mathsf{DA}_i)_{i=1}^{q}|dis(\mathsf{MS}_j)_{j=1}^{m})\right]$.

The first two probabilities are equal to zero as noted in the sElect proof [17]. The last probability is δ^k bounded by the completeness component of GBA. This is immediate when p_{voter}^{verif} is equal to one since our definition assumes all honest voters vote and verify; when p_{voter}^{verif} is lower this is more complicated and requires guessing ahead of time which voters will verify. This can be achieved using standard techniques from complexity leveraging.

Appendix C Proof for Theorem 3

Proof. Consider an adversary $\mathcal{A}$ against $\mathsf{Exp}_{\mathcal{A},\mathcal{V}_r}^{Ver}(\lambda)$. We start by running $\mathcal{A}$ getting the output pd which we use for $\mathcal{B}$ in addition to an honestly generated signing keypair for AS. We then make a random guess about which voters $\mathcal{A}$ is going to ask to verify. The probability of guessing correctly is at least $1/2^{|\mathcal{I}|}$. Now, we keep running $\mathcal{A}$ to choose honestly cast votes and creating the bulletin board BB. Every time the vote oracle is called and we guessed the voter is going to verify, we let $\mathcal{B}$ query the same and forward the output to $\mathcal{A}$. If we guessed that the voter is not going to verify, we simply honestly generate the ballot and send it to $\mathcal{A}$ without $\mathcal{B}$ querying the vote oracle. We use the board BB output by $\mathcal{A}$ in addition to honestly generated signatures for AS. Since the signature scheme is perfectly correct, the signatures will verify in lines 4-7 of $\mathsf{Exp}_{\mathcal{A},\mathcal{V}}^{\mathsf{GBA}}(\lambda)$.

We now run $\mathcal{O}$Verify for $\mathcal{B}$ which will call verification for all voters used in the oracle calls in $\mathsf{Exp}_{\mathcal{A},\mathcal{V}}^{\mathsf{GBA}}(\lambda)$. We can use the outputs to $\mathcal{A}$'s calls to the verification oracle. Here we assume that we guessed the verifiers correctly and, further, in this case the two sets of verifying voters will be the same in the two experiments. For the sake of

the proof, we will abort if they do not match, hence the degradation factor in the advantage. Now with probability $\frac{1}{2^{|\mathcal{I}|}}\mathsf{Adv}^{ver}_{\mathcal{A},\mathcal{V}_r}(\lambda)$ in $\mathsf{Exp}^{Ver}_{\mathcal{A},\mathcal{V}_r}(\lambda)$ we will have no complaints from the individual verification, the universal verification will be successful and we have $\neg(|\mathcal{I}| \geq |\mathsf{BB}|_{submit}| \geq |\mathsf{BB}|_{res}| \wedge \mathsf{V}[\mathsf{Checked}] \subseteq_{ms} \mathsf{BB}|_{res})$. Using that the scheme is accountability-verifiability-correct in $\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathcal{V}}(\lambda)$ all individual verification will also produce no blame since the signatures will verify by perfect correctness, and, finally, again by accountability-verifiability-correctness and successful universal verification, no blame will be output by Judge, i.e. $|B| = 0$. Since the votes from the verifying voters, $\mathsf{V}[\mathsf{Checked}]$, in $\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathcal{V}}(\lambda)$ exactly corresponds to the votes from the oracle vote calls in $\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathcal{V}}(\lambda)$ and $|\mathsf{BB}|_{submit}| = |\mathsf{BB}_{vote}|$ and $\mathsf{BB}|_{res} = \mathsf{BB}_{dec}$ we exactly get the winning condition $(\neg(n \geq |\mathsf{BB}_{vote}| \geq |\mathsf{BB}_{dec}| \wedge V \subseteq \mathsf{BB}_{dec}) \wedge B = \perp)$ in $\mathsf{Exp}^{\mathsf{GBA}}_{\mathcal{A},\mathcal{V}}(\lambda)$.

Paper IV

Coercion Mitigation for Voting Systems with Trackers: A Selene Case Study

Kristian Gjøsteen, Thomas Haines and Morten Rotvold Solberg

Preprint version of a paper published at the Eight International Joint Conference on Electronic Voting, E-Vote-ID 2023

Coercion Mitigation for Voting Systems with Trackers: A Selene Case Study

Kristian Gjøsteen[1], Thomas Haines[2], and Morten Rotvold Solberg[1]

[1] Norwegian University of Science and Technology, Trondheim, Norway
{kristian.gjosteen,mosolb}@ntnu.no
[2] Australian National University, Canberra, Australia thomas.haines@anu.edu.au

Abstract. An interesting approach to achieving verifiability in voting systems is to make use of tracking numbers. This gives voters a simple way of verifying that their ballot was counted: they can simply look up their ballot/tracker pair on a public bulletin board. It is crucial to understand how trackers affect other security properties, in particular privacy. However, existing privacy definitions are not designed to accommodate tracker-based voting systems. Furthermore, the addition of trackers increases the threat of coercion. There does however exist techniques to mitigate the coercion threat. While the term *coercion mitigation* has been used in the literature when describing voting systems such as Selene, no formal definition of coercion mitigation seems to exist. In this paper we define what privacy, verifiability and coercion mitigation mean for tracker-based voting systems, we model Selene in our framework and we prove that Selene satisfies the aforementioned properties.

1 Introduction

Electronic voting has seen widespread use over the past decades, ranging from smaller elections within clubs and associations, to large scale national elections as in Estonia. It is therefore necessary to understand the level of security that electronic voting systems provide. In this paper, we define precisely what verifiability, privacy and coercion mitigation means for voting systems using so-called *trackers*, and we prove that Selene provides these properties.

Verifiability is an interesting voting system property, allowing a voter to verify that their particular ballot was counted and that the election result correctly reflects the verified ballots. One example of a system with verifiability is Helios [2], which is used in the elections of the International Association for Cryptologic Research [1], among others. However, the Benaloh challenges used to achieve verifiability in Helios are hard to use for voters [25].

Schneier [33] proposed using human-readable *tracking numbers* for verifiability. Each voter gets a personal tracking number that is attached to their ballot. At the end of the election, all ballots with attached trackers are made publicly available. A voter can now trivially verify that their ballot appears next to their tracking number, which gives us verifiability as long as the trackers are unique. Multiple voting systems making use of tracking numbers have been proposed and deployed. Two notable examples are sElect [26] and Selene [31]. Tracking numbers intuitively give the voters a simple way of verifying that their ballot was recorded and counted. However, other security properties must also be

considered. In particular, it is necessary to have a good understanding of how the addition of tracking numbers affects the voters' *privacy*.

Verifiable voting may exacerbate threats such as *coercion*, in particular for remote electronic voting systems (e.g. internet voting) where a coercer might be present to "help" a coerced voter submit their ballot. *Coercion resistant* voting systems [24, 8] have been developed. Coercion resistance typically involves voters re-voting when the coercer is not present, but this often complicates voting procedures or increases the cost of the tallying phase. Furthermore, re-voting might not always be possible and may even be prohibited by law.

Like verifiability in general, tracking numbers may make coercion simpler: if a coercer gets access to a voter's tracker, the coercer may also be able to verify that the desired ballot was cast. While tracking numbers complicate coercion resistance, it may be possible to *mitigate* the threat of coercion. For instance, if the voter only learns their tracking number after the result (ballots with trackers) has been published, as in Selene, they may lie to a coercer by observing a suitable ballot-tracker pair. *Coercion mitigation* is weaker than coercion resistance, but may be appropriate for low-stakes elections or where achieving stronger properties is considered to be impractical.

1.1 Related work

Privacy Bernhard *et al.* [5] analysed then-existing privacy definitions. They concluded that previous definitions were either too weak (there are real attacks not captured by the definitions), too strong (no voting system with any form of verifiability can be proven secure under the definition), or too narrow (the definitions do not capture a wide enough range of voting systems).

The main technical difficulty compared to standard cryptographic privacy notions is that the result of the election must be revealed to the adversary. Not only could the result reveal information about individual ballots, but it also prevents straight-forward cryptographic real-or-random definitions from working. Roughly speaking, there are two approaches to defining privacy for voting systems, based on the two different questions: "Does anything leak out of the casting and tallying prosesses?" *vs.* "Which voter cast this particular ballot?" The first question tends to lead to simulation-based security notions, while the second question can lead to more traditional left-or-right cryptographic definitions.

Bernhard *et al.* [5] proposed the BPRIV definition, where the adversary plays a game against a challenger and interacts with two worlds (real and fake). The adversary first specifies ballots to be cast separately for each world. In the real world, ballots are cast and then counted as usual. In the fake world, the specified ballots are cast, but the ballots from the left world are counted and any tally proofs are simulated. The adversary then gets to see one of the worlds and must decide which world it sees. The idea is that for any secure system, the result in the fake world should be identical to what the result would have been in the real world, proving that – up to the actual result – the casting and tallying processes do not leak anything about the ballots cast, capturing privacy in this sense.

Bernhard *et al.* [5] proposed MiniVoting, an abstract scheme that models many voting systems (e.g. Helios), and proved that it satisfies the BPRIV definition. Cortier *et al.* [9] proved that Labelled-MiniVoting, an extension of MiniVoting, also satisfies BPRIV. Belenios [12] also satisfies BPRIV [10].

The original BPRIV definition does not attempt to model corruption in any part of the tally process. Cortier *et al.* [13] proposed mb-BPRIV which models adversarial control over which encrypted ballots should go through the tally process. Drăgan *et al.* [16] proposed the du-mb-BPRIV model which also covers systems where verification happens after tallying.

The other approach to privacy is a traditional left-or-right game, where the adversary interacts with the various honest components of a voting system (voters, their computers, shuffle and decryption servers, etc.), all simulated by an experiment. Privacy is captured by a left-or-right query, and the adversary must determine if the left or the right ballots were cast. The game becomes trivial if the left and the right ballots would give different tallies, so we require that the challenge queries taken together yield the same tally for left and right. In the simplest instantiation, the left and right ballots contain distinct permutations of the same ballots, so showing that they cannot be distinguished shows that the election processes do not leak who cast which ballots. Smyth [36] and Gjøsteen [20] provide examples of this definitional style. As far as we know, no definition in this style captures tracker-based voting systems.

The advantage of the traditional cryptographic left-or-right game relative to the BPRIV approach is that it is easier to model adversarial interactions with all parts of the protocol, including the different parts of the tally process. In principle, the BPRIV requirement that the tally process be simulatable is troublesome, since such simulators cannot exist in the plain model, which means that the definition itself technically exists in some unspecified idealised model (typically the random oracle model). In practice, this is not troublesome.

Verifiability Verifiability intuitively captures the notion that if a collection of voters verify the election, the result must be consistent with their cast ballots. For voters that do not verify or whose verification failed, we make no guarantees.

Several definitions of verifiability have appeared in the literature, see e.g. [11] or [37] for an overview. Furthermore, the verifiability properties of Selene have been thoroughly analysed both from a technical point of view (e.g. [31, 3]) and with respect to the user experience (e.g. [38, 15]).

Coercion Coercion resistance models a coercer that controls the voter for a period of time. We refer to Smyth [35] for an overview of definitions. A weaker notion is receipt-freeness, where the coercer does not control the voter, but asks for evidence that the voter cast the desired ballot. This was introduced by Benaloh and Tuinstra [4], while Chaidos *et al.* [6] gave a BPRIV-style security definition. Selene, as generally instantiated, is not receipt-free. *Coercion mitigation* is a different notion, where we assume that the coercer is not present during vote casting and is somehow not able to ask the voter to perform particular operations. This could allow the voter to fake information consistent with following the coercer's demands. While the term coercion mitigation has been used to describe the

security properties provided by Selene (e.g. in [31, 22, 38]), there seems to be no formal definition of coercion mitigation in the literature.

Selene Selene as a voting system has been studied previously, in particular with respect to privacy [17]. But a study of the complete protocol, including the tally phase, is missing. The coercion mitigation properties of Selene have also been extensively discussed [31, 22], but have not received a cryptographic analysis.

1.2 Our contribution

We define security for cryptographic voting systems with trackers, capturing privacy, verifiability and coercion mitigation. A unified experiment models the adversary's interaction with the honest players through various queries.

To break privacy, the adversary must decide who cast which ballot. Our definition is based on a similar definition by Gjøsteen [20, p. 492], adapted to properly accommodate voting systems using trackers. To break verifiability, the adversary must cause verifying voters to accept a result that is inconsistent with the ballots they have cast. To break coercion mitigation, the adversary is allowed to reveal the verification information of coerced voters and must decide if the coerced voter lied or not. Selene is vulnerable to collisions among such lies; e.g. multiple coerced voters claim the same ballot. We do not want this fact to clutter up the cryptographic analysis, so we require that the coercer organises the voting such that collisions do not happen. For schemes that are not vulnerable, we would remove the requirement.

Our definitions are easy to work with, which we demonstrate by presenting a complete model of Selene (expressed in our framework) and prove that Selene satisfies both privacy, verifiability and coercion mitigation. Selene has seen some use [32], so we believe these results are of independent interest.

We developed our definitions with Selene in mind, but they also accommodate other tracker based voting systems such as sElect [26] and Hyperion [30]. Furthermore, our models also capture voting systems that do not use trackers.

2 Background

2.1 Notation

We denote tuples/lists in bold, e.g. $\mathbf{v} = (v_1, \ldots, v_n)$, and the length of the tuple by $|\mathbf{v}|$. If we have multiple tuples, we denote the jth tuple by $\mathbf{v}_j$ and the ith element of the jth tuple by $v_{j,i}$. We denote the element-wise multiplication of two tuples $\mathbf{u}$ and $\mathbf{v}$ (i.e. $(u_1 v_1, \ldots, u_n v_n)$) by $\mathbf{u} \cdot \mathbf{v}$, with similar notation for element-wise addition $\mathbf{u} + \mathbf{v}$. We will also use similar notation for exponentiation, where $\mathbf{v}^a$ denotes the tuple $(v_1^a, \ldots, v_n^a)$, $a^{\mathbf{v}}$ denotes the tuple $(a^{v_1}, \ldots, a^{v_n})$ and $\mathbf{v}^{\mathbf{u}}$ denotes the tuple $(v_1^{u_1}, \ldots, v_n^{u_n})$. For a list $\mathbf{v} = (v_1, \ldots, v_n)$ and a permutation π on $\{1, \ldots, n\}$, we denote by $\mathbf{v}_\pi$ the list $(v_{\pi(1)}, \ldots, v_{\pi(n)})$.

2.2 Cryptographic Building Blocks

We briefly introduce some cryptographic primitives we need for our work. Due to space constraints we omit much of the details.

To protect voters' privacy, ballots are usually encrypted. Selene makes use of the ElGamal public key encryption system [18], which is used to encrypt both ballots and trackers. Throughout this paper, we will denote an ElGamal ciphertext by $(x, w) := (g^r, m \cdot \mathsf{pk}^r)$, where g is the generator of the cyclic group $\mathbb{G}$ (of prime order q) we are working in, m is the encrypted message, $\mathsf{pk} = g^{\mathsf{sk}}$ is the public encryption key (with corresponding decryption key sk) and r is a random element in $\mathbb{Z}_q$ (the field of integers modulo q).

Cryptographic voting systems typically make use of zero-knowledge proofs to ensure that certain computations are performed correctly. We refer to [14] for general background on zero-knowledge proofs. In particular, we use *equality of discrete logarithm* proofs and correctness proofs for *shuffles* of encrypted ballots. The former ensures correctness of computations. The latter preserves privacy by breaking the link between voters and their ballots. It is necessary that the shuffles are *verifiable* to ensure that no ballots are tampered with in any way. We refer to [21] for an overview of verifiable shuffles. In Selene it is necessary to shuffle two lists of ciphertexts (ballots and trackers) in parallel. Possible protocols are given in [29] and in Appendix A.

Furthermore, in Selene, the election authorities make use of Pedersen-style commitments [28] to commit to tracking numbers.

3 Voting Systems with Trackers

We model a voting protocol as a simple protocol built on top of a cryptographic voting scheme in such a way that the protocol's security properties can be easily inferred from the cryptographic voting scheme's properties. This allows us to separate key management (who has which keys) and plumbing (who sends which message when to whom) from the cryptographic issues, which simplifies analysis.

Due to space limitations, we model a situation with honest setup and tracker generation, as well as a single party decrypting. The former would be handled using a bespoke, verifiable multi-party computation protocol (see [31] for a suitable protocol for Selene), while the latter is handled using distributed decryption.

3.1 The Syntax of Voting Systems with Trackers

A voting system $\mathcal{S}$ with trackers consists of the following algorithms:

- Setup: takes as input a security parameter and returns a pair $(\mathsf{pk}, \mathsf{sk})$ of election public and secret keys.
- $\mathsf{UserKeyGen}$: takes as input an election public key pk and returns a pair $(\mathsf{vpk}, \mathsf{vsk})$ of voter public and secret keys.

- TrackerGen: takes as input an election public key pk and a list $(\mathsf{vpk}_1, \ldots, \mathsf{vpk}_n)$ of voter public keys and returns a list $\mathbf{t}$ of trackers, a list $\mathbf{et}$ of ciphertexts, a list $\mathbf{ct}$ of commitments, a list $\mathbf{op}$ of openings and a permutation π on the set $\{1, \ldots, n\}$.
- ExtractTracker: takes as input a voter secret key vsk, a tracker commitment ct and an opening op and returns a tracker t.
- ClaimTracker: takes as input a voter secret key vsk, a tracker commitment ct and a tracker t and returns an opening op.
- Vote: takes as input an election public key pk and a ballot v and returns a ciphertext ev, a ballot proof Π^v and a receipt ρ.
- Shuffle: takes as input a public key pk and a list $\mathbf{evt}$ of encrypted ballots and trackers, and returns a list $\mathbf{evt}'$ and a proof Π^s of correct shuffle.
- DecryptResult: takes as input a secret key sk and a list $\mathbf{evt}$ of encrypted ballots and trackers and returns a result res and a result proof Π^r.
- VoterVerify: takes as input a receipt ρ, a tracker t, a list $\mathbf{evt}$ of encrypted ballot/tracker pairs, a result res and a result proof Π^r and returns 0 or 1.
- VerifyShuffle: takes as input a public key pk, two lists $\mathbf{evt}, \mathbf{evt}'$ of encrypted ballots and trackers and a shuffle proof Π^s and returns 0 or 1.
- VerifyBallot: takes as input a public key pk, a ciphertext ev and a ballot proof Π^v and returns 0 or 1.
- VerifyResult: takes as input a public key pk, a list $\mathbf{evt}$ of encrypted ballots and trackers, a result res and a result proof Π^r and returns 0 or 1.

We say that a verifiable, tracker-based voting system is *n_s-correct* if for any $(\mathsf{pk}, \mathsf{sk})$ output by Setup, any $(\mathsf{vpk}_1, \mathsf{vsk}_1), \ldots, (\mathsf{vpk}_{n_v}, \mathsf{vsk}_{n_v})$ output by UserKeyGen, any lists $\mathbf{t}, \mathbf{et}, \mathbf{ct}, \mathbf{op}$ and permutations $\pi\colon \{1, \ldots, n_v\} \to \{1, \ldots, n_v\}$ output by $\mathsf{TrackerGen}(\mathsf{pk}, \mathsf{sk}, \mathsf{vpk}_1, \ldots, \mathsf{vpk}_{n_v})$, any ballots $v_1, \ldots, v_{n_v}$, any (ev_i, Π_i^v, ρ_i) output by $\mathsf{Vote}(\mathsf{pk}, v_i), i = 1, \ldots, n_v$, any sequence of n_s sequences of encrypted ballots and trackers $\mathbf{evt}_i$ and proofs Π_i^s output by $\mathsf{Shuffle}(\mathsf{pk}, \mathbf{evt}_{i-1})$, and any (res, Π^r) possibly output by $\mathsf{DecryptResult}(\mathsf{sk}, \mathbf{evt}_{n_s})$, the following hold:

- $\mathsf{DecryptResult}(\mathsf{sk}, \mathbf{evt}_{n_s})$ did not output $\perp$,
- $\mathsf{VoterVerify}(\rho_i, t_i, \mathbf{evt}_{n_s} \mathsf{res}, \Pi^r) = 1$ for all $i = 1, \ldots, n_v$,
- $\mathsf{VerifyShuffle}\left(\mathsf{pk}, \mathbf{evt}_{j-1}, \mathbf{evt}_j, \Pi_j^s\right) = 1$ for all $j = 1, \ldots, n_s$,
- $\mathsf{VerifyResult}(\mathsf{pk}, \mathbf{evt}_{n_s}, \mathsf{res}, \Pi^r) = 1$,
- $\mathsf{VerifyBallot}(\mathsf{pk}, ev_i, \Pi_i^v) = 1$ for all $i = 1, \ldots, n_v$, and

for any voter key pair $(\mathsf{vpk}, \mathsf{vsk})$, ct in $\mathbf{ct}$ and tracker t in $\mathbf{t}$, we have that

$$\mathsf{ExtractTracker}(\mathsf{vsk}, ct, \mathsf{ClaimTracker}(\mathsf{vsk}, ct, t)) = t.$$

We will describe later how Selene fits into our framework, but we note that this framework also captures voting systems that do not use trackers for verification. Such protocols are simply augmented with suitable dummy algorithms for TrackerGen, ExtractTracker and ClaimTracker.

3.2 Defining Security

We use a single experiment, found in Figure 1, to define privacy, integrity and coercion mitigation. Verifiability is defined in terms of integrity. The experiment models the cryptographic actions of honest parties.

The test query is used to model integrity. The challenge query is used to define privacy. The coerce and coercion verification queries are used to model coercion, again modified by freshness. The coerce query specifies two voters (actually, two indices into the list of voter public keys) and two ballots. The first voter is the coerced voter. The first ballot is the coerced voter's intended ballot, while the second ballot is the coercer's desired ballot. The second voter casts the opposite ballot of the coerced voter. In the *coercion verification query*, the coerced voter either reveals an opening to their true tracker, or an opening to the tracker corresponding to the coercer's desired ballot, cast by the second voter, thereby ensuring that the coerced voter can lie about its opening without risking a collision (as discussed in Section 1.2).

We make some restrictions on the order and number of queries (detailed in the caption of Figure 1), but the experiment allows the adversary to make combinations of queries that do not correspond to any behaviour of the voting protocol. Partially, we do so because we can, but also in order to simplify definitions of certain cryptographic properties (such as uniqueness of results).

The adversary decides which ballots should be counted. We need to recognise when the adversary has organised counting such that it results in a trivial win. We say that a sequence $\mathbf{evt}$ of encrypted ballots and trackers is *valid* if

- L_s contains a sequence of tuples $(\mathbf{evt}_{j-1}, \mathbf{evt}_j, \Pi_j^s)_{j=1}^{n_s}$, not necessarily appearing in the same order in L_s, with $\mathbf{evt}_{n_s} = \mathbf{evt}$;
- L_v contains tuples

$$(i_1, j_1, v_{0,1}, v_{1,1}, ev_1, \Pi_1^v, \rho_1), \ldots, (i_{n_c}, j_{n_c}, v_{0,n_c}, v_{1,n_c}, ev_{n_c}, \Pi_{n_c}^v, \rho_{n_c})$$

  such that $\mathbf{evt}_0 = (ev_1, \ldots, ev_{n_c})$; and
- for any $k, k' \in \{1, \ldots, n_c\}$ with $k \neq k'$, we have $i_k \neq i_{k'}$ (only one ballot per voter public key).

In this case, we also say that $\mathbf{evt}$ *originated* from $\mathbf{evt}_0$, alternatively from

$$(i_1, j_1, v_{0,1}, v_{1,1}, ev_1, \Pi_1^v, \rho_1), \ldots, (i_{n_c}, j_{n_c}, v_{0,n_c}, v_{1,n_c}, ev_{n_c}, \Pi_{n_c}^v, \rho_{n_c}).$$

Furthermore, we say that a valid sequence $\mathbf{evt}$ is *honest* if at least one of the tuples $(\mathbf{evt}_{j-1}, \mathbf{evt}_j, \Pi_j^s)$ comes from a shuffle query. A valid sequence is *balanced* if the ballot sequences $(v_{0,1}, \ldots, v_{0,n_c})$ and $(v_{1,1}, \ldots, v_{1,n_c})$ are equal up to order.

An execution is *fresh* if the following all hold:

- If a voter secret key, a receipt or a tracker is revealed, then any challenge query for that voter contains the same ballot on the left and the right side.
- For any result query $\mathbf{evt}$ that does not return $\perp$, $\mathbf{evt}$ is balanced and honest.

The experiment proceeds as follows:

- Sample $b, b'' \xleftarrow{r} \{0,1\}$. Let L_r, L_v, L_s, L_d be empty lists.
  We denote by $(\mathsf{vpk}_i, \mathsf{vsk}_i)$ the ith entry in L_r.
- Compute $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}$ and send pk to the adversary.
- On a *register query*, compute $(\mathsf{vpk}, \mathsf{vsk}) \leftarrow \mathsf{UserKeyGen}(\mathsf{pk})$, append $(\mathsf{vpk}, \mathsf{vsk})$ to L_r and send vpk to the adversary.
- On a *chosen voter key query* vpk, append $(\mathsf{vpk}, \bot)$ to L_r.
- On a *tracker generation query*, compute $(\mathbf{t}, \mathbf{et}, \mathbf{ct}, \mathbf{op}, \pi) \leftarrow \mathsf{TrackerGen}(\mathsf{pk}, \mathsf{vpk}_1, \ldots, \mathsf{vpk}_{n_r})$ where $\mathsf{vpk}_1, \ldots, \mathsf{vpk}_{n_r}$ are the public keys from L_r, and send $(\mathbf{t}, \mathbf{et}, \mathbf{ct})$ to the adversary.
  We denote by t_i, et_i, ct_i and op_i the ith entries in the corresponding lists.
- On a *chosen ciphertext query* (i, ev, Π^v), if $\mathsf{VerifyBallot}(ev, \Pi^v) = 1$, append $(i, \bot, \bot, \bot, ev, \Pi^v, \bot)$ to L_v.
- On a *challenge query* (i, v_0, v_1), compute $(ev, \Pi^v, \rho) \leftarrow \mathsf{Vote}(\mathsf{pk}, v_b)$, append $(i, \bot, v_0, v_1, ev, \Pi^v, \rho)$ to L_v and send (ev, Π^v) to $\mathcal{A}$.
- On a *coerce query* (i, j, v_0, v_1), compute $(ev_i, \Pi_i^v, \rho_i) \leftarrow \mathsf{Vote}(\mathsf{pk}, v_b)$ and $(ev_j, \Pi_j^v, \rho_j) \leftarrow \mathsf{Vote}(\mathsf{pk}, v_{1-b})$, append $(i, j, v_0, v_1, ev_i, \Pi_i^v, \rho_i)$ and $(j, i, v_1, v_0, ev_j, \Pi_j^v, \rho_j)$ to L_v, and send (ev_i, Π_i^v) and (ev_j, Π_j^v) to $\mathcal{A}$.
- On a *shuffle query* $\mathbf{evt}$, compute $(\mathbf{evt}', \Pi^s) \leftarrow \mathsf{Shuffle}(\mathsf{pk}, \mathbf{evt})$, append $(\mathbf{evt}, \mathbf{evt}', \Pi^s)$ to L_s and send $(\mathbf{evt}', \Pi^s)$ to $\mathcal{A}$.
- On a *chosen shuffle query* $(\mathbf{evt}, \mathbf{evt}', \Pi^s)$, if $\mathsf{VerifyShuffle}(\mathbf{evt}, \mathbf{evt}', \Pi^s) = 1$, append the query to L_s.
- On a *result query* $\mathbf{evt}$, compute $(\mathsf{res}, \Pi^r) \leftarrow \mathsf{DecryptResult}(\mathsf{sk}, \mathbf{evt})$, send (res, Π^r) to $\mathcal{A}$ and append $(\mathbf{evt}, \mathsf{res}, \Pi^r)$ to L_d.
- On a *voter verification query* $(k, \mathbf{evt}, \mathsf{res}, \Pi^d)$ with $(i, \bot, v_0, v_1, ev, \Pi^v, \rho)$ being the kth entry in L_v, compute $t \leftarrow \mathsf{ExtractTracker}(\mathsf{vsk}_i, op_i, ct_i)$ and $d \leftarrow \mathsf{VoterVerify}(\rho_i, t, \mathbf{evt}, \mathsf{res}, \Pi^r)$ and send d to $\mathcal{A}$.
- On a *coercion verification query* k, with $(i, j, \ldots)$ being the kth entry in the L_v list, then if $b = 0$ send op_i to $\mathcal{A}$, otherwise compute $op \leftarrow \mathsf{ClaimTracker}(\mathsf{vsk}_i, ct_i, t_{\pi(j)})$ and send op to $\mathcal{A}$.
- On a *test query* $(\mathbf{evt}, \mathsf{res}, \Pi^r)$, compute $d \leftarrow \mathsf{VerifyResult}(\mathbf{evt}, \mathsf{res}, \Pi^r)$ and send d to $\mathcal{A}$.
- On a *voter key reveal query* i, send $(\mathsf{vpk}_i, \mathsf{vsk}_i)$ to $\mathcal{A}$.
- On a *tracker reveal query* i, compute $t \leftarrow \mathsf{ExtractTracker}(\mathsf{vsk}_i, ct_i, op_i)$ and send t to $\mathcal{A}$.
- On a *receipt reveal query* k, where the kth entry of L_v is $(\cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \rho_k)$, send ρ_k to $\mathcal{A}$.
- On an *election key reveal query*, send sk to $\mathcal{A}$.

Eventually, the adversary outputs a bit b'.

Fig. 1. Security experiment for privacy, integrity and coercion mitigation. The bit b'' is not used in the experiment, but simplifies the definition of advantage. The adversary makes register and chosen voter key queries, followed by a single tracker generation query, followed by other queries. If there are coerce or coercion verification queries in an execution, every challenge query must have $v_0 = v_1$.

- For any voter verification query $(j, \mathbf{evt}, \mathsf{res}, \Pi^r)$, $\mathbf{evt}$ contains an encryption of $v_{b,j}$ and $\mathsf{VerifyResult}(\mathsf{pk}, \mathbf{evt}, \mathsf{res}, \Pi^r)$ evaluates to 1.
- For any encrypted ballot returned by a coerce query, if it is in an origin of any result query, the other encrypted ballot returned by the coerce query is also in the same origin of the same result query.
- There is no election key reveal query.

We define the joint privacy and coercion mitigation event E_p to be the event that after the experiment and an adversary has interacted, the execution is fresh and $b' = b$, or the execution is not fresh and $b' = b''$. In other words, if the adversary makes a query that results in a non-fresh execution of the experiment, we simply compare the adversary's guess to a random bit, giving the adversary no advantage over making a random guess.

In the integrity game, the adversary's goal is to achieve inconsistencies:

- The *count failure* event F_c is that a result query for a valid sequence of encrypted ballots and trackers results in $\perp$.
- The *inconsistent result* event F_r is that a test query $(\mathbf{evt}, \mathsf{res}, \Pi^r)$ evaluates to 1, $\mathbf{evt}$ originated from

$$(i_1, \cdot, v_{0,1}, v_{1,1}, ev_1, \Pi_1^v, \rho_1), \ldots, (i_{n_c}, \cdot, v_{0,n_c}, v_{1,n_c}, ev_{n_c}, \Pi_{n_c}^v, \rho_{n_c})$$

  and there is no permutation π on $\{1, \ldots, n_c\}$ such that for $i = 1, \ldots, n_c$, either $v_{b,i} = \perp$ or $\mathsf{Dec}(\mathsf{sk}, ev_{\pi(i)}) = v_{b,i}$.
- The *no unique result* event F_u is that two test queries $(\mathbf{evt}, \mathsf{res}_1, \Pi_1^r)$ and $(\mathbf{evt'}, \mathsf{res}_2, \Pi_2^r)$ both evaluate to 1, $\mathbf{evt}$ and $\mathbf{evt'}$ have a common origin, and res_1 and res_2 are not equal up to order.
- The *inconsistent verification* event F_v is that a sequence of voter verification queries $\{(k_j, \mathbf{evt}, \mathsf{res}, \Pi^r)\}_{j=1}^n$ all return 1, $\mathbf{evt}$ is valid, and with $L_v = \left((i_1, \cdot, v_{0,1}, v_{1,1}, ev_1, \Pi_1^v, \rho_1), \ldots, (i_{n_c}, \cdot, v_{0,n_c}, v_{1,n_c}, ev_{n_c}, \Pi_{n_c}^v, \rho_{n_c})\right)$ there is no permutation π on $\{1, \ldots, n_c\}$ such that $\mathsf{Dec}\left(\mathsf{sk}, ev_{\pi(k_j)}\right) = v_{b,k_j}$ for all $j = 1, \ldots, n$, i.e. that all the specified voters think their ballots are included in the tally, but at least one of the ballots is not.

We define the advantage of an adversary $\mathcal{A}$ against a voting system $\mathcal{S}$ to be

$$\mathsf{Adv}_{\mathcal{S}}^{\mathsf{vote}}(\mathcal{A}) = \max\left\{2 \cdot \left|\Pr[E_p] - 1/2\right|, \Pr[F_c \vee F_r \vee F_u \vee F_v]\right\}.$$

3.3 The Voting Protocol

The different parties in the voting protocol are the n_v voters and their devices, a trusted *election authority* (EA) who runs setup, registration, tracker generation and who tallies the cast ballots, a collection of n_s shuffle servers, one or more auditors, and a public append-only bulletin board BB. There are many simple variations of the voting protocol.

In the *setup phase*, the EA runs Setup to generate election public and secret keys pk and sk. The public key pk is posted to BB.

In the *registration phase*, the EA runs $\mathsf{UserKeyGen}(\mathsf{pk})$ to generate per-voter keys $(\mathsf{vpk}, \mathsf{vsk})$ for each voter. The public key vpk is posted to BB and the secret key vsk is sent to the voter's device.

In the *tracker generation phase*, the EA runs $\mathsf{TrackerGen}(\mathsf{pk}, \mathsf{sk}, \mathsf{vpk}_1, \ldots, \mathsf{vpk}_{n_v})$ to generate trackers, encrypted trackers, tracker commitments and openings to the commitments. To break the link between voters and their trackers, the trackers are encrypted and put through a re-encryption mixnet before they are committed to. Each encrypted tracker and commitment is assigned to a voter public key and posted to BB next to this key. Plaintext trackers are also posted to BB.

In the *voting phase*, a voter instructs her device on which ballot v to cast. The voter's device runs the Vote algorithm to produce an encrypted ballot ev and a proof of knowledge Π^v of the underlying plaintext. The encrypted ballot and the proof are added to the bulletin board next to the voter's public key, encrypted tracker and tracker commitment.

In the *tallying phase*, the auditors first verify the ballot proofs Π_i^v, subsequently ignoring any ballot whose ballot proof does not verify. The pairs (ev_i, et_i) of encrypted ballots and trackers are extracted from the bulletin board and sent to the first shuffle server. The first shuffle server uses the shuffle algorithm $\mathsf{Shuffle}$ on the input encrypted ballots and trackers, before passing the shuffled ballots on the next shuffle server, which shuffles the ballots again and sends the shuffled list to the next shuffle server, and so on. All the shuffle servers post their output ciphertexts and shuffle proofs on the bulletin board, and the auditors verify the proofs. If all the shuffles are correct, the EA runs $\mathsf{DecryptResult}$ on the output from the final shuffle server, to obtain a result res and a proof Π^r. The auditors verify this too and add their signatures to the bulletin board.

In the *verification phase*, the EA tells each voter which tracker belongs to them (the exact details of how this happens depends on the underlying voting system). The voters then run $\mathsf{VoterVerify}$ to verify that their vote was correctly cast and counted. For voting systems without trackers (such as Helios [2] and Belenios [12]), voters simply run $\mathsf{VoterVerify}$ without interacting with the EA.

Security Properties It is easy to see that we can simulate a run of the voting protocol using the experiment. It is also straight-forward for anyone to verify, from the bulletin board alone, if the list of encrypted ballots and trackers that is finally decrypted in a run of the protocol is valid.

For simplicity, we have assumed trusted setup (including tracker generation) and no distributed decryption. We may also assume that any reasonable adversary against the voting scheme has negligible advantage.

It follows, under the assumption of trusted tracker generation, that as long as the contents of the bulletin board verifies, we have verifiability in the sense that the final result is consistent with the ballots of voters that successfully verify.

If at least one of the shuffle servers is honest and the election secret key has not been revealed, and the adversary does not manage to organise the voting to get a trivial win, we also have *privacy* and *coercion mitigation*.

4 The Selene Voting System

We provide a model of Selene and analyse it under our security definition. Relative to the original Selene paper, there are three interesting differences/choices: (1) We do not model distributed setup and tracker generation, nor distributed decryption. (2) The voter proves knowledge of the ballot using an equality of discrete logarithm proof. (3) We assume the shuffle from Appendix A is used. The latter two simplify the security proof by avoiding rewinding. The first is due to lack of space (though see [31] for distributed setup protocols, and [20] for how to model distributed decryption).

4.1 The Voting System

Let $\mathbb{G}$ be a group of prime order q, with generator g. Let $\mathsf{E} = (\mathsf{Kgen}, \mathsf{Enc}, \mathsf{Dec})$ be the ElGamal public key encryption system. Let $\Sigma_{dl} = (\mathcal{P}_{dl}, \mathcal{V}_{dl})$ be a proof system for proving equality of discrete logarithms in $\mathbb{G}$ (e.g. the Chaum-Pedersen protocol [7]). We abuse notation and let $\Sigma_s = (\mathcal{P}_s, \mathcal{V}_s)$ denote both a proof system for shuffling ElGamal ciphertexts and a proof system for shuffling pairs of ElGamal ciphertexts. Our instantiation of Selene works as follows:

- Setup: sample $h_v \xleftarrow{r} \mathbb{G}$ and compute $(\mathsf{pk_v}, \mathsf{sk_v}) \leftarrow \mathsf{Kgen}(1^\lambda)$ and $(\mathsf{pk_t}, \mathsf{sk_t}) \leftarrow \mathsf{Kgen}(1^\lambda)$. The election public key is $\mathsf{pk} = (\mathsf{pk_v}, \mathsf{pk_t}, h_v)$ and the election secret key is $\mathsf{sk} = (\mathsf{sk_v}, \mathsf{sk_t})$.
- $\mathsf{UserKeyGen}(\mathsf{pk})$: compute $(\mathsf{vpk}, \mathsf{vsk}) \leftarrow \mathsf{Kgen}(1^\lambda)$.
- $\mathsf{TrackerGen}(\mathsf{pk}, \mathsf{vpk}_1, \ldots, \mathsf{vpk}_n)$: set $\mathbf{t} \leftarrow (1, \ldots, n)$. Choose a random permutation π on the set $\{1, \ldots, n\}$. For each i, choose random elements $r_i, s_i \xleftarrow{r} \{0, \ldots, q-1\}$, compute ElGamal encryptions $et_i \leftarrow (g^{r_{\pi(i)}}, \mathsf{pk_t}^{r_{\pi(i)}} g^{t_{\pi(i)}})$ and commitments $ct_i \leftarrow \mathsf{vpk}_i^{s_i} \cdot g^{t_{\pi(i)}}$. Set $op_i = g^{s_i}$. The public output is the list of trackers $\mathbf{t}$, the list of encrypted trackers $\mathbf{et}$ and the list of tracker commitments $\mathbf{ct}$. The private output is the list of openings $\mathbf{op}$ to the commitments and the permutation π.
- $\mathsf{ExtractTracker}(\mathsf{vsk}, ct, op)$: compute $g^t \leftarrow ct \cdot op^{-\mathsf{vsk}}$.
- $\mathsf{ClaimTracker}(\mathsf{vsk}, ct, g^t)$: compute $op \leftarrow (ct/g^t)^{1/\mathsf{vsk}}$.
- $\mathsf{Vote}(\mathsf{pk}, v)$: sample $r \xleftarrow{r} \{0, \ldots, q-1\}$ and compute $x \leftarrow g^r$, $\hat{x} \leftarrow h_v^r$ and $w \leftarrow \mathsf{pk}_v^r v$. Compute a proof $\Pi^{dl} \leftarrow \mathcal{P}_{dl}((g, h_v, x, \hat{x}), r)$ showing that $\log_g x = \log_{h_v} \hat{x} = r$. Output $c = (x, w)$, $\Pi^v = (\hat{x}, \Pi^{dl})$ and $\rho = v$.
- $\mathsf{Shuffle}(\mathsf{pk}, \mathbf{evt})$: sample two lists $\mathbf{r}_v, \mathbf{r}_t \xleftarrow{r} \{0, \ldots, q-1\}^n$ and a random permutation on the set $\{1, \ldots, n\}$. For each $((x_{v,i}, w_{v,i}), (x_{t,i}, w_{t,i})) \in \mathbf{evt}$, compute $x'_{v,i} \leftarrow g^{r_{v,\pi(i)}} x_{v,\pi(i)}$, $w'_{v,i} \leftarrow \mathsf{pk}_v^{r_{v,\pi(i)}} w_{v,\pi(i)}$, $x'_{t,i} \leftarrow g^{r_{t,\pi(i)}} x_{t,\pi(i)}$ and $w'_{t,i} \leftarrow \mathsf{pk}_t^{r_{t,\pi(i)}} w_{t,\pi(i)}$. Compute a proof $\Pi^s \leftarrow \mathcal{P}_s((\mathbf{evt}, \mathbf{evt}'), (\mathbf{r}_v, \mathbf{r}_t, \pi))$ of correct shuffle and output $(\mathbf{evt}', \Pi^s)$.
- $\mathsf{DecryptResult}(\mathsf{sk}, \mathbf{evt})$: for each $((x_{v,i}, w_{v,i}), (x_{t,i}, w_{t,i})) \in \mathbf{evt}$, compute $v_i \leftarrow \mathsf{Dec}(\mathsf{sk_v}, (x_{v,i}, w_{v,i}))$, $t_i \leftarrow \mathsf{Dec}(\mathsf{sk_t}, (x_{t,i}, w_{t,i}))$ and proofs $\Pi^{dl}_{v,i} \leftarrow \mathcal{P}_{dl}((g, x_{v,i}, \mathsf{pk_v}, w_{v,i}/v_i), \mathsf{sk_v})$ and $\Pi^{dl}_{t,i} \leftarrow \mathcal{P}_{dl}((g, x_{t,i}, \mathsf{pk_t}, w_{t,i}/t_i), \mathsf{sk_t})$, proving that $\log_g \mathsf{pk_v} = \log_{x_{v,i}}(w_{v,i}/v_i) = \mathsf{sk_v}$ and $\log_g \mathsf{pk_t} = \log_{x_{t,i}}(w_{t,i}/t_i) = \mathsf{sk_t}$. Set $\mathsf{res} \leftarrow \mathbf{v}$ and $\Pi^r \leftarrow (\{\Pi^{dl}_{v,i}\}, \{\Pi^{dl}_{t,i}\}, \mathbf{t})$ and output (res, Π^r).

- $\mathsf{VoterVerify}(\rho, t, \mathbf{evt}, \mathbf{v}, \Pi^r)$: parse Π^r as $(\{\Pi^{dl}_{v,i}\}, \{\Pi^{dl}_{t,i}\}, \mathbf{t})$ and check if $\rho \in \mathbf{v}$, and $t \in \mathbf{t}$, and that if $t = t_i$ then $\rho = v_i$, i.e. the ballot appears next to the correct tracker.
- $\mathsf{VerifyShuffle}(\mathsf{pk}, \mathbf{evt}, \mathbf{evt}', \Pi^s)$: compute $d \leftarrow \mathcal{V}_s(\mathsf{pk}, \mathbf{evt}, \mathbf{evt}', \Pi^s)$.
- $\mathsf{VerifyBallot}(\mathsf{pk}, ev, \Pi^v)$: parse Π^v as $(\hat{x}, \Pi^{dl})$ and compute $d \leftarrow \mathcal{V}_{dl}((g, h, x, \hat{x}), \Pi^{dl})$.
- $\mathsf{VerifyResult}(\mathsf{pk}, \mathbf{evt}, \mathsf{res}, \Pi^r)$: parse Π^r as $(\{\Pi^{dl}_{v,i}\}, \{\Pi^{dl}_{t,i}\}, \mathbf{t})$ and compute $d_{v,i} \leftarrow \mathcal{V}_{dl}((g, x_{v,i}, \mathsf{pk_v}, w_{v,i}/v_i), \Pi^{dl}_{v,i})$ and $d_{t,i} \leftarrow \mathcal{V}_{dl}((g, x_{t,i}, \mathsf{pk_t}, w_{t,i}/t_i), \Pi^{dl}_{t,i})$ for all $i = 1, \ldots, n$, where $((x_{v,i}, w_{v,i}), (x_{t,i}, w_{t,i})) \in \mathbf{evt}, v_i \in \mathsf{res}, t_i \in \mathbf{t}$.

The correctness of Selene follows from the correctness of ElGamal, the completeness of the verifiable shuffles and the straight-forward computation

$$\mathsf{ExtractTracker}(\mathsf{vsk}, ct, \mathsf{ClaimTracker}(\mathsf{vsk}, ct, g^t)) = ct \cdot \left(\left(ct/g^t\right)^{1/\mathsf{vsk}}\right)^{-\mathsf{vsk}} = g^t.$$

Note that in the original description of Selene [31], the exact manner of which the voters prove knowledge of their plaintext in the voting phase is left abstract. However, several different approaches are possible. One may, for example, produce a Schnorr proof of knowledge [34] of the randomness used by the encryption algorithm. We choose a different approach, and include a check value $\hat{x}$ and give a Chaum-Pedersen proof that $\log_{h_v} \hat{x} = \log_g x$. Both are valid approaches, however our approach simplifies the security proof by avoiding rewinding.

4.2 Security Result

We say that an adversary against a voting scheme is *non-adaptive* if every voter key reveal query is made before the tracker generation query.

Theorem 1. *Let $\mathcal{A}$ be a non-adaptive $(\tau, n_v, n_c, n_d, n_s)$-adversary against Selene, making at most n_v registration and chosen voter key queries, n_c challenge and coerce queries, n_d chosen ciphertext queries, and n_s shuffle/chosen shuffle queries, and where the runtime of the adversary is at most τ. Then there exist a τ'_1-distinguisher $\mathcal{B}_1$, a $\tau'_{2,1}$-distinguisher $\mathcal{B}_{2,1}$, a $\tau'_{2,2}$-distinguisher $\mathcal{B}_{2,2}$ and a τ'_3-distinguisher $\mathcal{B}_3$, all for DDH, $\tau'_1, \tau'_{2,1}, \tau'_{2,2}, \tau'_3$ all essentially equal to τ, such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{vote}}_{\mathsf{Selene}}(\mathcal{A}) \leq\ & \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},g}(\mathcal{B}_1) + 2n_s(\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},g}(\mathcal{B}_{2,1}) + \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},g}(\mathcal{B}_{2,2})) \\ & + \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},g}(\mathcal{B}_3) + \textit{negligible terms.}\end{aligned}$$

4.3 Proof of Theorem 1

We begin by analysing the integrity events. *Count failures* cannot happen. If we get an inconsistent result, then either the equality of discrete logarithm proofs used by the decryption algorithm or the shuffle proofs are wrong. The soundness errors of the particular proofs we use are negligible (and unconditional), so an *inconsistent result* happens with negligible probability. The same analysis applies to *non-unique results* as well as *inconsistent verification*.

We now move on to analysing the privacy event. The proof is structured as a sequence of games. We begin by simulating the honestly generated non-interactive proofs during ballot casting. This allows us to randomize the check values $\hat{x}_v$ in honestly generated ballot proofs, so that we afterwards can embed a trapdoor in h_v. The trapdoors allow us to extract ballots from adversarially generated ciphertexts. The shuffle we use also allows us to extract permutations from adversarially generated shuffles by tampering with a random oracle. This allows us to use the ballots from chosen ciphertext queries to simulate the decryption, so we no longer use the decryption key. The next step is to also simulate the honest shuffles, before randomising the honestly generated ciphertexts (including encrypted trackers) and the re-randomisations of these ciphertexts. Finally, we sample tracker commitments at random and compute the openings from tracker generation using the $\mathsf{ClaimTracker}$ algorithm. This change is not observable, and makes the computation of tracker commitments and openings independent of the challenge bit. This makes the entire game independent of the challenge bit, proving that the adversary has no advantage.

We now give a more detailed description of each game in the security proof. In the following, let $E_{p,i}$ be the event that $b = b'$ (i.e. that the adversary's guess b' is equal to the challenge bit b) in Game i, given that the execution is fresh.

Game 0. The initial game is the adversary $\mathcal{A}$ interacting with the original security experiment. Thus,

$$\mathsf{Adv}^{\mathsf{vote}}_{\mathcal{S}}(\mathcal{A}) = \max\left\{2 \cdot \left|\Pr[E_{p,0}] - 1/2\right|, \Pr[F_c \vee F_r \vee F_u \vee F_v]\right\}.$$

We have already bounded the integrity event terms, so all that remains is to bound the privacy term.

Game 1. In this game we simulate the proof of equal discrete logarithms when responding to challenge or coerce queries. The simulation is perfect, but the change is still detectable if the adversary already has queried the random oracle on one of the commitments made by the prover in the proof of discrete logarithm equality. This happens with probability $n_h(n_r + n_c)/q$. Thus, we get

$$\left|\Pr[E_{p,1}] - \Pr[E_{p,0}]\right| \leq \frac{n_h(n_r + n_c)}{q}. \tag{1}$$

Game 2. In this game, we first change the response to challenge or coerce queries so that the experiment encrypts ballots using the secret key, i.e. instead of computing $x \leftarrow g^r$ and $w \leftarrow \mathsf{pk}_v^r \cdot v$, we compute x as before and then $w \leftarrow x^{\mathsf{sk}_v} \cdot v$. This change is not noticeable by the adversary. Furthermore, when responding to challenge or coerce queries, we sample $\hat{x}_v \xleftarrow{\mathrm{r}} \mathbb{G}$ instead of computing it.

Lemma 1. *There is a τ_1'-distinguisher $\mathcal{B}_1$ for Decision Diffie-Hellman, τ_1' essentially equal to τ, such that*

$$\left|\Pr[E_{p,2}] - \Pr[E_{p,1}]\right| \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},g}(\mathcal{B}_1) + 1/q.$$

Proof. In Game 1, when making a challenge or coerce query, the adversary gets to see a tuple $(x, h_v, \hat{x}_v) = (g^r, g^s, g^{rs})$ where r is the randomness used to encrypt and s is some random number in $\{0, \ldots, q-1\}$. This is a DDH-tuple. In Game 2, the adversary gets to see the random tuple (g^r, g^s, g^t), where r, s, t are all chosen uniformly at random.

Using random self-reducibility, it is trivial to build a DDH distinguisher $\mathcal{B}_1$ that perfectly simulates Game 1 when given a DDH tuple and perfectly simulates Game 2 when given a non-DDH tuple. The term $1/q$ comes because not all random tuples are non-DDH tuples. □

Game 3. First, challenge and coerce queries again encrypt ballots as usual (though $\hat{x}_v$ is still just a random element). Instead of sampling $h_v \xleftarrow{r} \mathbb{G}$, the experiment samples $b_0 \xleftarrow{r} \{0, \ldots, q-1\}$ and computes $h_v \leftarrow \mathsf{pk}_v^{b_0}$. For a hash query for the non-interactive shuffle of ciphertexts to get (ζ, β), we sample $\omega \xleftarrow{r} \{0, \ldots, q-1\}$ and compute $\zeta \leftarrow g^\omega$ and record ω together with ζ.

For a chosen ciphertext query $(i, (x, w), (\hat{x}_v, \Pi_v^{dl}))$ that is accepted, we compute $v \leftarrow w\hat{x}_v^{-1/b_0}$ and append $(i, v, v, (x, w), (\hat{x}_v, \Pi_v^{dl}), v)$ to L_v.

When $\mathcal{A}$ makes a shuffle query $\mathbf{evt}$, we append $(\mathbf{evt}, \mathbf{evt}', \Pi^s, \pi)$ to L_s, where π is the permutation used by the non-interactive shuffle algorithm.

When $\mathcal{A}$ makes a chosen shuffle query $(\mathbf{evt}, \mathbf{evt}', \Pi^s)$ that is accepted, we use the value ω that we recorded for the relevant hash query to extract the permutation π that is used. The extraction proceeds as follows. By computing ζ as described, we get $\hat{v}_i = \zeta^{\lambda_{\pi(i)}} = g^{\omega\lambda_{\pi(i)}}$ (cf. Protocol 2). Since the $\hat{v}_i's$ are public, we can compute $\hat{v}_i^{1/\omega} = g^{\lambda_{\pi(i)}} = u_{\pi(i)}$. Since the list $\mathbf{u}$ is also public, we can compare the $\mathbf{u}$ with the list $\mathbf{u}_\pi$ to recover the permutation. If the extraction fails for any reason (e.g. if the elements of $\mathbf{u}$ are not unique), we let π be the identity permutation. Finally, we append $(\mathbf{evt}, \mathbf{evt}', \Pi^s, \pi)$ to L_s.

These changes are not observable to the adversary, so

$$\Pr[E_{p,3}] = \Pr[E_{p,2}]. \tag{2}$$

Game 4. In this game, we simulate the decryption proofs in the tally. The get the correct decryption, we first find an origin for the list $\mathbf{evt}$ of encrypted ballots and trackers. We can then recover a list of ballots $\mathbf{v}$ from L_c. We then compose the permutations recorded in L_s into a single permutation π and give the $v_{\pi(i)}$ as inputs to the simulator. This list of ballots may be incorrect if any of the accepted ballot proofs or chosen shuffle proofs are incorrect, which happens only with negligible probability. The simulator is perfect, but this might still be detectable if the random oracle has been queried on any of the commitments computed by the prover in the non-interactive proofs of discrete logarithm equality. This happens only with probability $n_h(n_r + n_v)/q$. Thus, we get

$$\Pr[E_{p,4}] - \Pr[E_{p,3}]| \leq \text{negligible terms}. \tag{3}$$

Note that at this point, we no longer use the secret key.

Game 5. In this game, we simulate the proofs of the honest shuffles. The following claim is immediate.

Lemma 2. *There exists τ_2'-distinguishers $\mathcal{B}_{2,1}$ and $\mathcal{B}_{2,2}$ for DDH, τ_2' essentially equal to τ, such that*

$$|\Pr[E_{p,5}] - \Pr[E_{p,4}]| \leq 2n_s(\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},g}(\mathcal{B}_{2,1}) + \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},g}(\mathcal{B}_{2,2})). \tag{4}$$

Game 6. In this game, we encrypt a random message when responding to challenge and coercion queries, and we let the output ciphertexts of shuffle queries be encryptions of random messages instead of re-randomizations of the input ciphertexts. We also encrypt random group elements instead of encrypting the actual tracker when responding to a tracker generation query, with the exception that we need to use the proper trackers for voters who have had their keys revealed to avoid making distinguishing trivial for the adversary. Again, the following claim is immediate by random self-reduction.

Lemma 3. *There exists a τ_3'-distinguisher $\mathcal{B}_3$ for DDH, τ_3' essentially equal to τ, such that*

$$|\Pr[E_{p,6}] - \Pr[E_{p,5}]| \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},g}(\mathcal{B}_3). \tag{5}$$

Game 7. In this game, we respond to coercion verification events when $b = 0$, as well as tracker reveal queries, by computing the tracker opening as $\mathsf{ClaimTracker}(\mathsf{vsk}, ct_i, g^{t_{\pi(i)}})$ instead of fetching the opening from **op** or using $\mathsf{ExtractTracker}$. This change is not observable, so

$$\Pr[E_{p,7}] = \Pr[E_{p,6}]. \tag{6}$$

Note that the openings are never used after this.

Game 8. In this game, we sample $ct_i \xleftarrow{\mathrm{r}} \mathbb{G}$ instead of computing ct_i during the tracker generation query. Since the openings are never used or revealed, this is unobservable and we get

$$\Pr[E_{p,8}] = \Pr[E_{p,7}]. \tag{7}$$

Conclusion. In Game 8, everything the adversary sees is independent of the challenge bit b, so we get that

$$\Pr[E_{p,6}] = 1/2. \tag{8}$$

5 Other Variants of Selene

There are [30, 31] some challenges tied to the use of trackers in Selene. First, if the coercer is also a voter, there is a possibility that a coerced voter points to the coercer's own tracker when employing the coercion evasion strategy. Second, publishing the trackers in the clear next to the ballots might affect the voters' *perceived* privacy, and some might find this troublesome.

To address the first challenge, the authors of Selene have proposed a variant they call Selene II. Informally, the idea is to provide each voter with a set of alternative (or dummy)

trackers, one for each possible candidate, in a way that the set of alternative trackers is unique to each voter. This way, it is not possible for a coerced voter to accidentally point to the coercer's tracker. However, trackers are still published in the clear.

Both challenges are also addressed by Ryan *et al.* [30], who have proposed a voting system they call Hyperion. The idea is to only publish commitments next to the plaintext ballots, rather than plaintext trackers. Furthermore, to avoid the issue that voters might accidentally point to the coercer's own tracker, each voter is given their unique view of the bulletin board.

For both Selene II [31] and Hyperion [30], we refer to the original papers for the full details of the constructions, but we briefly describe here how these systems fit into our framework. We first remark that in Selene II, it is necessary that the encryption system used to encrypt the ballots supports *plaintext equivalence tests* (PETs). As in the original description of Selene, we use ElGamal encryption to encrypt the ballots, so PETs are indeed supported (see e.g. [23]).

For Selene II, we need to change the $\mathsf{TrackerGen}$ algorithm so that it outputs $c + 1$ trackers for each voter, where c is the number of candidates, and c "dummy" ciphertexts, one ciphertext for each candidate. We let the last tracker be the one that is sent to the voter to be used for verification. By construction, for all voters there will be an extra encrypted ballot for each candidate. Thus, the $\mathsf{DecryptResult}$ algorithm works similarly as for Selene, except that it needs to subtract n_v votes for each candidate, where n_v is the number of voters. The *voting protocol* must also be changed. Before notifying the voters of their tracking numbers, the EA must now perform a PET between each voter's submitted ciphertext, and each of the "dummy" ciphertexts belonging to the voter, before removing the ciphertext (and the corresponding tracker) containing the same candidate as the voter voted for. This way, all voters receive a set of trackers, each pointing to a different candidate, which is unique to them. The opening to their real trackers is transmitted as usual, and thus the $\mathsf{ExtractTracker}$ algorithm works as in Selene. The $\mathsf{ClaimTracker}$ algorithm also works exactly as in Selene, except that voters now can choose a tracker from their personal set of dummy trackers, thus avoiding the risk of accidentally choosing the coercer's tracker.

For Hyperion, the modification of the $\mathsf{TrackerGen}$ algorithm is straight forward: we simply let it compute tracker commitments as described in [30], namely by (for each voter) sampling a random number r_i and computing the commitment as $\mathsf{vpk}_i^{r_i}$. At the same time, an opening is computed as $op_i \leftarrow g^{r_i}$. The $\mathsf{Shuffle}$ algorithm still shuffles the list of encrypted ballots and tracker commitments in parallel, in the sense that they are subjected to the same permutation. However, the encrypted ballots are put through the same re-encryption shuffle as before, but the tracker commitments are put through an *exponentiation mix*, raising all commitments to a common secret power s. The $\mathsf{DecryptResult}$ algorithm now performs additional exponentiation mixes to the commitments, one mix for each voter (by raising the commitment to a secret power s_i, unique to each voter), giving the voters their own unique view of the result. For each voter, it also computes the final opening to their commitments, as $op_i \leftarrow g^{r_i \cdot s \cdot s_i}$. Again, we need to change the voting protocol, this time so that each voter actually receives their own view of the bulletin

board. The ExtractTracker algorithm raises the opening op_i to the voter's secret key and and loops through the bulletin board to find a matching commitment. The ClaimTracker algorithm uses the voter's secret key to compute an opening to a commitment pointing to the coercer's desired ballot.

6 Concluding Remarks

In this work, we have presented security definitions for cryptographic voting systems making use of tracking numbers, that simultaneously capture ballot privacy, integrity and coercion mitigation. To the best of our knowledge, this is the first game-based definition of coercion mitigation in the literature. Our definitions closely resemble standard cryptographic definitions, making them fairly easy and intuitive to work with. Furthermore, we have presented a complete model of Selene in our framework, and proved that Selene satisfies all the aforementioned security properties.

6.1 Future work

In this paper, we only present pen-and-paper security definitions and proofs. As such, an idea for future work is to model our definitions and proofs in a proof assistant, such as EasyCrypt. The main value of modeling security definitions and proofs in EasyCrypt is perhaps the increased assurance that the definitions and proofs are sound. However, it also has a scientific value on its own. Indeed, EasyCrypt is still a product under development, and modeling complex security definitions and proofs might help uncover shortcomings in the tool, as well as aid in developing the EasyCrypt standard library.

While ballot privacy at first glance seems to be a strictly weaker property than coercion resistance (and receipt-freeness), Küsters *et al.* [27] demonstrate that the relationship between privacy and coercion resistance is more subtle than first thought. Similarly, coercion mitigation is intuitively a weaker property than receipt-freeness, but no certain claims can be made without a thorough analysis of the relationship between the two. However, this is out of scope for this paper and is, as such, left for future research.

References

1. Final report of IACR electronic voting committee. `https://www.iacr.org/elections/eVoting/finalReportHelios_2010-09-27.html`. Accessed: 2023-05-05.
2. Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security 2008*, pages 335–348. USENIX Association, July / August 2008.
3. S. Baloglu, S. Bursuc, S. Mauw, and J. Pang. Election verifiability in receipt-free voting protocols. In *2023 2023 IEEE 36th Computer Security Foundations Symposium (CSF) (CSF)*, pages 63–78, Los Alamitos, CA, USA, jul 2023. IEEE Computer Society.
4. Josh Cohen Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *26th ACM STOC*, pages 544–553. ACM Press, May 1994.
5. David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. A comprehensive analysis of game-based ballot privacy definitions. Cryptology ePrint Archive, Report 2015/255, 2015. `https://eprint.iacr.org/2015/255`.

6. Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. BeleniosRF: A non-interactive receipt-free electronic voting scheme. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1614–1625. ACM Press, October 2016.
7. David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, Heidelberg, August 1993.
8. Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. In *2008 IEEE Symposium on Security and Privacy*, pages 354–368. IEEE Computer Society Press, May 2008.
9. Véronique Cortier, Constantin Catalin Dragan, François Dupressoir, Benedikt Schmidt, Pierre-Yves Strub, and Bogdan Warinschi. Machine-checked proofs of privacy for electronic voting protocols. In *2017 IEEE Symposium on Security and Privacy*, pages 993–1008. IEEE Computer Society Press, May 2017.
10. Véronique Cortier, Constantin Catalin Dragan, François Dupressoir, and Bogdan Warinschi. Machine-checked proofs for electronic voting: Privacy and verifiability for belenios. In Steve Chong and Stephanie Delaune, editors, *CSF 2018 Computer Security Foundations Symposium*, pages 298–312. IEEE Computer Society Press, 2018.
11. Véronique Cortier, David Galindo, Ralf Küsters, Johannes Mueller, and Tomasz Truderung. SoK: Verifiability notions for E-voting protocols. In *2016 IEEE Symposium on Security and Privacy*, pages 779–798. IEEE Computer Society Press, May 2016.
12. Véronique Cortier, Pierrick Gaudry, and Stéphane 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 *Lecture Notes in Computer Science*, pages 214–238. Springer, 2019.
13. Véronique Cortier, Joseph Lallemand, and Bogdan Warinschi. Fifty shades of ballot privacy: Privacy against a malicious board. In Limin Jia and Ralf Küsters, editors, *CSF 2020 Computer Security Foundations Symposium*, pages 17–32. IEEE Computer Society Press, 2020.
14. Ivan Damgård. *Commitment Schemes and Zero-Knowledge Protocols*, pages 63–86. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.
15. Verena Distler, Marie-Laure Zollinger, Carine Lallemand, Peter B. Roenne, Peter Y. A. Ryan, and Vincent Koenig. Security - visible, yet unseen? CHI '19, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery.
16. Constantin Cătălin Drăgan, François Dupressoir, Ehsan Estaji, Kristian Gjøsteen, Thomas Haines, Peter Y. A. Ryan, Peter B. Rønne, and Morten Rotvold Solberg. Machine-checked proofs of privacy against malicious boards for selene & co. Cryptology ePrint Archive, Report 2022/1182, 2022. `https://eprint.iacr.org/2022/1182`.
17. Constantin Catalin Dragan, François Dupressoir, Ehsan Estaji, Kristian Gjøsteen, Thomas Haines, Peter Y. A. Ryan, Peter B. Rønne, and Morten Rotvold Solberg. Machine-checked proofs of privacy against malicious boards for selene & co. In *CSF 2022 Computer Security Foundations Symposium*, pages 335–347. IEEE Computer Society Press, August 2022.
18. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg, August 1984.
19. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.
20. Kristian Gjøsteen. *Practical Mathematical Cryptography*. Chapman and Hall/CRC, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742, 2023.
21. Thomas Haines and Johannes Müller. SoK: Techniques for verifiable mix nets. In Limin Jia and Ralf Küsters, editors, *CSF 2020 Computer Security Foundations Symposium*, pages 49–64. IEEE Computer Society Press, 2020.
22. Vincenzo Iovino, Alfredo Rial, Peter B. Rønne, and Peter Y. A. Ryan. Using selene to verify your vote in JCJ. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y. A. Ryan, Vanessa

Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *FC 2017 Workshops*, volume 10323 of *LNCS*, pages 385–403. Springer, Heidelberg, April 2017.
23. Markus Jakobsson and Ari Juels. Mix and match: Secure function evaluation via ciphertexts. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 162–177. Springer, Heidelberg, December 2000.
24. Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. Cryptology ePrint Archive, Report 2002/165, 2002. `https://eprint.iacr.org/2002/165`.
25. Fatih Karayumak, Maina M Olembo, Michaela Kauer, and Melanie Volkamer. Usability analysis of helios-an open source verifiable remote electronic voting system. *EVT/WOTE*, 11(5), 2011.
26. Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. sElect: A lightweight verifiable remote voting system. In Michael Hicks and Boris Köpf, editors, *CSF 2016 Computer Security Foundations Symposium*, pages 341–354. IEEE Computer Society Press, 2016.
27. Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Verifiability, privacy, and coercion-resistance: New insights from a case study. In *2011 IEEE Symposium on Security and Privacy*, pages 538–553. IEEE Computer Society Press, May 2011.
28. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.
29. Kim Ramchen. Parallel shuffling and its application to prêt à voter. In *EVT/WOTE*, 2010.
30. Peter Y. A. Ryan, Simon Rastikian, and Peter B Rønne. Hyperion: An enhanced version of the selene end-to-end verifiable voting scheme. *E-Vote-ID 2021*, page 285, 2021.
31. Peter Y. A. Ryan, Peter B. Rønne, and Vincenzo Iovino. Selene: Voting with transparent verifiability and coercion-mitigation. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *LNCS*, pages 176–192. Springer, Heidelberg, February 2016.
32. Muntadher Sallal, Steve Schneider, Matthew Casey, Catalin Dragan, François Dupressoir, Luke Riley, Helen Treharne, Joe Wadsworth, and Phil Wright. Vmv: Augmenting an internet voting system with selene verifiability. 11 2019.
33. Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., USA, 2nd edition, 1995.
34. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.
35. Ben Smyth. Surveying definitions of coercion resistance. Cryptology ePrint Archive, Report 2019/822, 2019. `https://eprint.iacr.org/2019/822`.
36. Ben Smyth. Ballot secrecy: Security definition, sufficient conditions, and analysis of helios. *J. Comput. Secur.*, 29(6):551–611, 2021.
37. Ben Smyth and Michael R. Clarkson. Surveying definitions of election verifiability. Cryptology ePrint Archive, Report 2022/305, 2022. `https://eprint.iacr.org/2022/305`.
38. Marie-Laure Zollinger, Verena Distler, Peter Rønne, Peter Ryan, Carine Lallemand, and Vincent Koenig. User experience design for e-voting: How mental models align with security mechanisms. 10 2019.

A Examples of Verifiable Shuffles

We here give two examples of verifiable shuffle protocols: one for shuffling two lists of ElGamal ciphertexts in parallel (Protocol 2) and one for shuffling random numbers (Protocol 1). Protocol 1 is due to Gjøsteen [20] and Protocol 2 is an extension of a protocol by Gjøsteen [20], adapted to shuffling two lists of ciphertexts in parallel. Note that in this paper, Protocol 1 is only used as a sub-protocol in Protocol 2. Protocol 2 can easily be modified to a protocol for shuffling single sequences of ElGamal ciphertexts instead of two sequences in parallel. In the single sequence setting, the common input is only one ElGamal public key (i.e. y_v) and two sequences of ciphertexts (i.e. $(\mathbf{x}_v, \mathbf{w}_v)$ and $(\tilde{\mathbf{x}}_v, \tilde{\mathbf{w}}_v)$), while the private input consists of only one random tuple $\mathbf{r}_v$, in addition to a permutation π. Furthermore, we remove every computation involving $\mathbf{x}_t, \tilde{\mathbf{x}}_t, \mathbf{w}_t, \tilde{\mathbf{w}}_t$ and $\mathbf{r}_t$. In other words, we remove everything with subscript t.

Protocol 1 Interactive argument for shuffle of secret random values, based on a group $\mathbb{G}$ of prime order q with generator g.

Common Input: A generator $\xi \in \mathbb{G}$ and commitments $\mathbf{u}$ and $\mathbf{v}$.

Private Input: Messages $\mathbf{m}$, an integer a and a permutation π such that $\xi = g^a$, $\mathbf{u} = g^{\mathbf{m}}$ and $\mathbf{v} = \xi^{\boldsymbol{\pi}\mathbf{m}}$.

1: $\mathcal{V}$ samples $\beta_0 \xleftarrow{\mathrm{r}} \mathbb{Z}_q \setminus \{m_1, \ldots, m_l\}$ and sends β_0 to $\mathcal{P}$.

2: $\mathcal{P}$ samples $\rho_1, \ldots, \rho_{2l-1} \xleftarrow{\mathrm{r}} \mathbb{Z}_q$ and computes $\alpha_1 \leftarrow g^{\rho_1 a(\beta_0 - m_{\pi(1)})}$, $\alpha_i \leftarrow g^{\rho_{i-1}(\beta_0 - m_i) + \rho_i a(\beta_0 - m_{\pi(i)})}$ for $i = 2, \ldots, l$, $\alpha_i \leftarrow g^{\rho_{i-1} a + \rho_i}$ for $i = l+1, \ldots, 2l-1$ and $\alpha_{2l} \leftarrow g^{\rho_{2l-1} a}$, and sends $(\alpha_1, \ldots, \alpha_{2l})$ to $\mathcal{V}$.

3: $\mathcal{V}$ samples $\beta_1 \xleftarrow{\mathrm{r}} \mathbb{Z}_q$ and sends β_1 to $\mathcal{P}$.

4: $\mathcal{P}$ computes $\rho'_1 \leftarrow -(\beta_0 - m_1)/(a(\beta_0 - m_{\pi(1)}))$, $\rho'_i \leftarrow -\rho'_{i-1}(\beta_0 - m_i)/(a(\beta_0 - m_{\pi(i)}))$ for $i = 2, \ldots, l$, $\rho'_i \leftarrow -\rho'_{i-1} a$ for $i = l+1, \ldots, 2l-1$ and $\boldsymbol{\gamma} \leftarrow \boldsymbol{\rho} + \beta_1 \boldsymbol{\rho}'$ and sends $\boldsymbol{\gamma}$ to $\mathcal{V}$.

5: $\mathcal{V}$ accepts if and only if $\alpha_i = (g^{\beta_0} u_i^{-1})^{\gamma_{i-1}} (\xi^{\beta_0} v_i^{-1})^{\gamma_i}$ for $i = 1, 2, \ldots, l$ and $\alpha_i = \xi^{\gamma_{i-1}} g^{\gamma_i}$ for $i = l+1, l+2, \ldots, 2l$, where $\gamma_0 = \gamma_{2l} = \beta_1$.

The shuffle arguments in Protocols 1 and 2 can be made non-interactive by applying the Fiat-Shamir heuristic [19]. The idea is to replace the challenges sent by the verifier in step 1 and 3 in Protocol 1 and step 2 and 4 in Protocol 2 by a call to some hash function, making the challenges look random. This is fairly straight-forward, but in step 1 of Protocol 1, we need to make sure that the hash value β_0 does not land in the set $\{m_1, \ldots, m_l\}$. As Protocol 1 is only used as a sub-protocol of Protocol 2, if the unlikely event occurs that β_0 lands in the set $\{m_1, \ldots, m_l\}$, we simply abort and start Protocol 2 from the beginning, resulting in new values for the public information to be hashed (and hence a new value for β_0), as well as new values for $m_1, \ldots, m_l$. Note that the verifier can verify in the final step that β_0 is indeed in the correct set, by computing g^{β_0} and checking that $g^{\beta_0} \notin \mathbf{u}$.

Protocol 2 Interactive argument for a parallel shuffle of ElGamal ciphertexts, based on ElGamal over a group $\mathbb{G}$ of prime order q with generator g.

Common Input: Two ElGamal public keys y_v and y_t, and four sequences of ciphertexts $(\mathbf{x}_v, \mathbf{w}_v)$, $(\tilde{\mathbf{x}}_v, \tilde{\mathbf{w}}_v)$, $(\mathbf{x}_t, \mathbf{w}_t)$ and $(\tilde{\mathbf{x}}_t, \tilde{\mathbf{w}}_t)$.

Private Input: A permutation π on $\{1, 2, \ldots, l\}$ and random tuples $\mathbf{r}_v$ and $\mathbf{r}_t$ such that $\tilde{\mathbf{x}}_v = g^{\pi\mathbf{r}_v}\mathbf{x}_v^{\pi}$, $\tilde{\mathbf{w}}_v = y_v^{\pi\mathbf{r}_v}\mathbf{w}_v^{\pi}$, $\tilde{\mathbf{x}}_t = g^{\pi\mathbf{r}_t}\mathbf{x}_t^{\pi}$ and $\tilde{\mathbf{w}}_t = y_t^{\pi\mathbf{r}_t}\mathbf{w}_t^{\pi}$.

1: $\mathcal{P}$ samples $a \xleftarrow{r} \{0, 1, \ldots, q-1\}$ and $\mathbf{r}'_v, \mathbf{r}''_v, \mathbf{r}'_t, \mathbf{r}''_t, \boldsymbol{\lambda}_0, \boldsymbol{\lambda}_2 \xleftarrow{r} \{0, 1, \ldots, q-1\}^l$, such that $\lambda_{0,1}, \ldots, \lambda_{0,l}$ are all distinct, and computes re-randomisations $\bar{\mathbf{x}}_v \leftarrow g^{\mathbf{r}'_v}\mathbf{x}_v$, $\bar{\mathbf{w}}_v \leftarrow y_v^{\mathbf{r}'_v}\mathbf{w}_v$, $\dot{\mathbf{x}}_v \leftarrow g^{\mathbf{r}''_v}\tilde{\mathbf{x}}_v$, $\dot{\mathbf{w}}_v \leftarrow y_v^{\mathbf{r}''_v}\tilde{\mathbf{w}}_v$, $\bar{\mathbf{x}}_t \leftarrow g^{\mathbf{r}'_t}\mathbf{x}_t$, $\bar{\mathbf{w}}_t \leftarrow y_t^{\mathbf{r}'_t}\mathbf{w}_t$, $\dot{\mathbf{x}}_t \leftarrow g^{\mathbf{r}''_t}\tilde{\mathbf{x}}_t$, $\dot{\mathbf{w}}_t \leftarrow y_t^{\mathbf{r}''_t}\tilde{\mathbf{w}}_t$, $\xi \leftarrow g^a$ and permutation commitments $\mathbf{u}_0 \leftarrow g^{\boldsymbol{\lambda}_0}$, $\mathbf{u}_2 \leftarrow g^{\boldsymbol{\lambda}_2}$ and $\mathbf{v}_0 \leftarrow \xi^{\pi\boldsymbol{\lambda}_0}$. The prover sends $\xi, (\bar{\mathbf{x}}_v, \bar{\mathbf{w}}_v), (\dot{\mathbf{x}}_v, \dot{\mathbf{w}}_v), (\bar{\mathbf{x}}_t, \bar{\mathbf{w}}_t), (\dot{\mathbf{x}}_t, \dot{\mathbf{w}}_t), \mathbf{u}_0, \mathbf{u}_2$ and $\mathbf{v}_0$ to $\mathcal{V}$.

2: $\mathcal{V}$ checks that $u_{0,1}, \ldots, u_{0,l}$ are all distinct, samples $\boldsymbol{\lambda}_3 \leftarrow \{0, 1, \ldots, q-1\}^l$ and sends $\boldsymbol{\lambda}_3$ to the prover. Both parties compute $\mathbf{u}_1 \leftarrow g^{\boldsymbol{\lambda}_3}\mathbf{u}_2$.

3: $\mathcal{P}$ computes $\boldsymbol{\lambda}_1 \leftarrow \boldsymbol{\lambda}_2 + \boldsymbol{\lambda}_3$ and $\mathbf{v}_1 \leftarrow \xi^{\pi\boldsymbol{\lambda}_1}$ and sends $\mathbf{v}_1$ to $\mathcal{V}$.

4: $\mathcal{V}$ samples $\zeta \xleftarrow{r} \mathbb{G}$ and $\beta \xleftarrow{r} \mathbb{Z}_q$ and sends (ζ, β) to $\mathcal{P}$.

5: $\mathcal{P}$ computes $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda}_0 + \beta\boldsymbol{\lambda}_1$, and both parties compute $\mathbf{u} \leftarrow \mathbf{u}_0\mathbf{u}_1^{\beta}$ and $\mathbf{v} \leftarrow \mathbf{v}_0\mathbf{v}_1^{\beta}$.

6: $\mathcal{P}$ computes $\hat{\mathbf{v}} \leftarrow \zeta^{\pi\boldsymbol{\lambda}}$, $\check{\mathbf{x}}_v \leftarrow \bar{\mathbf{x}}_v^{\boldsymbol{\lambda}}, \check{\mathbf{w}}_v \leftarrow \bar{\mathbf{w}}_v^{\boldsymbol{\lambda}}, \hat{\mathbf{x}}_v \leftarrow \dot{\mathbf{x}}_v^{\pi\boldsymbol{\lambda}}, \hat{\mathbf{w}}_v \leftarrow \dot{\mathbf{w}}_v^{\pi\boldsymbol{\lambda}}, \check{\mathbf{x}}_t \leftarrow \bar{\mathbf{x}}_t^{\boldsymbol{\lambda}}, \check{\mathbf{w}}_t \leftarrow \bar{\mathbf{w}}_t^{\boldsymbol{\lambda}}, \hat{\mathbf{x}}_t \leftarrow \dot{\mathbf{x}}_t^{\pi\boldsymbol{\lambda}}, \hat{\mathbf{w}}_t \leftarrow \dot{\mathbf{w}}_t^{\pi\boldsymbol{\lambda}}$, and sends $\hat{\mathbf{v}}, (\check{\mathbf{x}}_v, \check{\mathbf{w}}_v), (\hat{\mathbf{x}}_v, \hat{\mathbf{w}}_v), (\check{\mathbf{x}}_t, \check{\mathbf{w}}_t)$ and $(\hat{\mathbf{x}}_t, \hat{\mathbf{w}}_t)$ to $\mathcal{V}$.

7: $\mathcal{P}$ and $\mathcal{V}$ run the random value shuffle from Protocol 1 with public input $(\xi, \mathbf{u}, \mathbf{v})$ and private input $(\boldsymbol{\lambda}, a, \pi)$.

8: For $i = 1, 2, \ldots, l$, $\mathcal{P}$ and $\mathcal{V}$ run the Chaum-Pedersen argument for equality of discrete logarithms with input as in the following table:

public input	private input
$(g, \bar{x}_{v,i}, u_i, \check{x}_{v,i})$	λ_i
$(g, \bar{w}_{v,i}, u_i, \check{w}_{v,i})$	λ_i
$(\xi, \dot{x}_{v,i}, v_i, \hat{x}_{v,i})$	$\lambda_{\pi(i)}$
$(\xi, \dot{w}_{v,i}, v_i, \hat{w}_{v,i})$	$\lambda_{\pi(i)}$
$(g, \bar{x}_{t,i}, u_i, \check{x}_{t,i})$	λ_i
$(g, \bar{w}_{t,i}, u_i, \check{w}_{t,i})$	λ_i
$(\xi, \dot{x}_{t,i}, v_i, \hat{x}_{t,i})$	$\lambda_{\pi(i)}$
$(\xi, \dot{w}_{t,i}, v_i, \hat{w}_{t,i})$	$\lambda_{\pi(i)}$
$(\xi, \zeta, v_i, \hat{v}_i)$	$\lambda_{\pi(i)}$
$(g, y_v, \bar{x}_{v,i}/x_{v,i}, \bar{w}_{v,i}/w_{v,i})$	$r'_{v,i}$
$(g, y_v, \dot{x}_{v,i}/\tilde{x}_{v,i}, \dot{w}_{v,i}/\tilde{w}_{v,i})$	$r''_{v,i}$
$(g, y_t, \bar{x}_{t,i}/x_{t,i}, \bar{w}_{t,i}/w_{t,i})$	$r'_{t,i}$
$(g, y_t, \dot{x}_{t,i}/\tilde{x}_{t,i}, \dot{w}_{t,i}/\tilde{w}_{t,i})$	$r''_{t,i}$
$(g, y_v, \prod_i \check{x}_{v,i}/\hat{x}_{v,i}, \prod_i \check{w}_{v,i}/\hat{w}_{v,i})$	$\left(\sum_i \lambda_i r'_{v,i}\right) - \left(\sum_i \lambda_{\pi(i)}(r_{v,\pi(i)} + r''_{v,i})\right) \mod q$
$(g, y_t, \prod_i \check{x}_{t,i}/\hat{x}_{t,i}, \prod_i \check{w}_{t,i}/\hat{w}_{t,i})$	$\left(\sum_i \lambda_i r'_{t,i}\right) - \left(\sum_i \lambda_{\pi(i)}(r_{t,\pi(i)} + r''_{t,i})\right) \mod q$

ISBN 978-82-326-7462-6 (printed ver.)
ISBN 978-82-326-7461-9 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

NTNU

Norwegian University of
Science and Technology