Hyperion: Transparent End-to-End Verifiable Voting with Coercion Mitigation

Aditya Damodaran
SnT, University of Luxembourg, Luxembourg
aditya.damodaran@uni.lu

Simon Rastikian
IBM Research Europe, Zurich
ETH, Zurich
sra@zurich.ibm.com

Peter B. Rønne
Université de Lorraine, LORIA, CNRS, Nancy
SnT, University of Luxembourg, Luxembourg
peter.roenne@gmail.com

Peter Y. A. Ryan
FSTM, University of Luxembourg, Luxembourg
peter.ryan@uni.lu

ABSTRACT

We present *Hyperion*, an end-to-end verifiable e-voting scheme that allows the voters to identify their votes in cleartext in the final tally. In contrast to schemes like *Selene* or *sElect*, identification is not via (private) tracker numbers but via cryptographic commitment terms. After publishing the tally, the *Election Authority* provides each voter with an individual dual key. Voters identify their votes by raising their dual key to their secret trapdoor key and finding the matching commitment term in the tally.

The dual keys are self-certifying in that, without the voter's trapdoor key, it is intractable to forge a dual key that, when raised to the trapdoor key, will match an alternative commitment. On the other hand, a voter can use their own trapdoor key to forge a dual key to fool any would-be coercer.

Additionally, we propose a variant of *Hyperion* that counters the tracker collision threat present in *Selene*. We introduce individual verifiable views: each voter gets their own independently shuffled view of the master *Bulletin Board*.

We provide new improved definitions of privacy and verifiability for e-voting schemes and prove the scheme secure against these, as well as proving security with respect to earlier definitions in the literature.

Finally, we provide a prototype implementation and provide measurements which demonstrate that our scheme is practical for large scale elections.

KEYWORDS

End-to-end verifiable voting, game-based proofs, benchmarks

1 INTRODUCTION

Many democracies are moving towards voting over the internet, and some, e.g. Estonia, have already adopted it. While internet voting has many attractions it introduces new, poorly understood threats. The internet is inherently insecure and remote voting introduces coercion threats not present in in-person voting. To counter these threats, cryptographic mechanisms and protocols have been proposed. However, designing and analysing such protocols is very challenging, and we have not reached consensus on rigorous definitions of security properties such as *vote secrecy*, *verifiability*, *receipt-freeness*, *coercion-resistance* and *dispute resolution*.

A good voting system should not only deliver the correct result w.r.t. the legitimately cast votes, but also provide sufficient evidence to convince all observers of the announced result. Ensuring both vote secrecy and verifiability is complex, and indeed, many technologies sacrifice the latter, forcing the stakeholders to place total, blind trust in the correct behaviour of the machines. An example is the direct-recording electronic (DRE) machines which fail to provide any evidence as to how votes are recorded or counted. Such observations motivated the development of end-to-end verifiable (E2E V) schemes [30] and the notion of software independence [37].

E2E V schemes usually involve the creation of an encryption or encoding of the vote at the time of casting, a copy of which is retained by the voter. Later the voter can check that her "protected ballot" – to use Rivest's terminology – appears correctly on an append-only public ledger called the *Bulletin Board* (BB). After this, a universally verifiable, anonymising tally is performed on the posted, encrypted ballots to reveal the result. Voters can also perform some form of ballot auditing before casting to gain assurance that their vote is correctly represented in their ballot. Putting these steps together ensures that the corruption of any vote during recording and tallying is detectable. Along with mechanisms to prevent ballot stuffing and clash attacks (ballot collisions) etc. we can detect any inaccuracy in the announced outcome.

Such schemes, while technically appealing, have at least two drawbacks. First, the fact that errors can be detected does not guarantee that they will be: it is essential that sufficient numbers of voters and observers actually perform the checks diligently and report anomalies. Second, a voting scheme must be easily understandable and usable by voters and voting officials. The assurance argument outlined above is rather subtle, and not easy for many voters or stakeholders to digest. Many find the idea of voters having to perform checks on encrypted ballots unreasonable.

These observations prompted the exploration of more direct and transparent forms of verification, in particular based on the idea of private tracker numbers to identify votes in cleartext in the tally. Examples of such schemes include the CNRS scheme [4], *Selene* [39] and sElect [34]. Of these, *Selene* is of particular interest as it provides mitigation of the coercion threats that tracker based schemes otherwise exhibit: the coercer demands the voter to reveal her tracker.

1.1 Contribution

We present a novel, end-to-end verifiable scheme, inspired by the *Selene* scheme [39], that not only provides a highly transparent verification, but also affords voters a greater sense of privacy than with *Selene*. The main *Hyperion* variant is significantly more efficient as it greatly simplifies the setup (section 3.1) of *Selene*, eliminating all the encryption, mixing, decryption and ZK proofs computations implied by the use of tracker numbers. Furthermore, this construction enables us to neatly sidestep the tracker collision attack alluded to above, where a coerced voter equivocates to the tracker of the coercer.

Hyperion, in contrast to *Selene*, does not publicly reveal trackers, indeed, we can do away entirely with trackers. Instead, the voter identifies her vote in the tally by identifying the row in the tally containing the commitment that opens, with her trapdoor secret key and dual key, to a constant, e.g. 1. This is rather like identifying your house by finding the door that opens to your key. This is still deniable, but the mechanism is now different: a coerced voter identifies a commitment paired with the coercer's required vote and, if necessary computes, using her trapdoor key, the fake dual key that opens the chosen commitment to 1.

Doing away with the trackers also improves the situation, especially for coerced voters, since they do not need to equivocate and lie about trackers that they have seen and which could have easy-to-remember or characteristic features. As an example, a voter might accidentally reveal having a tracker with all consecutive numbers.

At first glance *Hyperion* seems to counter the tracker collision threat[1], but a problem still exists: the coercer claiming instead ownership of the commitment term that the voter proffers. To address this we propose a further innovation: each voter gets an individual view of the BB. Each view is verifiably derived from the BB with **its own**, **independent** re-randomisation and shuffling. Thus the rows, and hence vote/commitment pairs, appear in a different form and order for each voter. So even a shoulder-surfing coercer cannot identify his commitment in the voter's view, and if he claims to, the voter can be sure that this is a bluff. This construction might be used[2] in smaller elections as this higher coercion-resistance advantage comes with an efficiency loss (more work is required to re-randomise BB, produce and verify ZK proofs).

The *Hyperion* construction has further advantages over *Selene*. In particular, we demonstrate variants that exploit the fact that the cryptographic commitments are perfectly randomly distributed to an observer only seeing the bulletin board, and we use this to sketch versions which are quantum-safe against future attackers or even satisfying everlasting privacy.

Further, we demonstrate that the construction can be generalised: the commitments identifying the vote can be viewed as a secret key derived from a key exchange, and this gives a transformation from implicitly authenticated key exchange protocols to *Hyperion*-like constructions.

[1]Collision threats do not break receipt-freeness but violates coercion-resistance: none provide a way for the voter to prove how she voted, however the voter might be detected evading the coercer's instructions. Or, more subtly, the voter may be put in a situation in which she fears she has been detected.

[2]The idea of verifiable individual views of a public ledger may well have applications beyond voting.

Our construction gives rise naturally to a number of variants, notably one in which trackers are retained but each tracker is only revealed to the associated voter by opening a commitment to the voter. This has the feature that a coerced voter does not need to choose an alternative tracker but can just compute a dual key that will open an alternative commitment to the voter's own tracker. This has the surprising consequence that voters can learn their tracker in advance, indeed such knowledge could even be made public. The existence of such variants demonstrates the richness of the construction.

We also contribute with definitional work, firstly by providing a ballot privacy definition allowing maliciously generated public keys by corrupted voters, and considering stronger adversaries with access to information about whether voters verify successfully or not. Secondly, we give a definition of verifiability against a malicious voting board and consider malware on the user side, and we prove verifiability when either the vote-casting or the vote-verification device is uncorrupted. We also provide proofs of security against established definitions in the literature, especially we prove privacy against a malicious board as in [17, 23].

Finally we provide a prototype implementation along with performance data.

1.2 Structure of the Paper

We first discuss related work, and then introduce the notation that then helps exhibit the new scheme in its main version. We describe the voter experience: casting and verifying a vote and, where necessary, evading the coercer. The basic version of the scheme does not include the voter individual views and so is still vulnerable to the tracker/commitment collision threat. We then sketch the construction with individual views, and further describe the alternative constructions mentioned above, including the possibility of introducing voting codes to make it more robust and dispute resistant.

The remainder of the paper presents the security definitions followed by game-based proofs. These definitions are also novel and represent a contribution to the state of the art in the field. For the ballot privacy definition, we consider adversaries who get information on whether the verification of voters failed or not. Such information can lead to privacy attacks, as demonstrated in other protocols, and are important to counter in *Hyperion*. For verifiability, we craft a definition considering that each voter has two devices – one for vote casting and one for verification, and we demonstrate that both need to be corrupted for successful verifiability attacks against *Hyperion*.

Finally, we include some performance statistics for a prototype implementation that demonstrate that the scheme is practical for large scale elections, e.g of the order of a million voters.

1.3 Related Work

Most of the end-to-end verifiable schemes proposed to date involve a rather indirect verification by the voter: checking that an encryption of their vote appears on the BB in the input to a (universally verifiable) tally process. This check is only really meaningful if voters are convinced that their vote is correctly encrypted and subsequently correctly processed and decrypted. Unsurprisingly, many

people find this rather unconvincing and uncompelling. In the light of this, some recent schemes seek a more direct and more compelling voter verification process: identifying the vote in plaintext in the final tally. Here we focus on this class of scheme.

Schneier [43], proposes the idea of voters attaching a password to their vote which is then posted alongside the vote on the BB. Later, [4] elaborate on this in a boardroom context. sElect [34], is also tracker-based but with the additional feature of having an accountable tally process. All of these systems are vulnerable to the obvious threat of the coercer demanding the voter reveal her tracker. *Selene* [39] introduced the idea of delayed notification of the trackers to mitigate the coercion threat, along with constructions to guarantee uniqueness of the trackers.

Some adaptations of *Selene* have been presented in an in-person variant [38, 49] and in a JCJ-like variant [31] which offers greater coercion-resistance. *Selene* has been analysed symbolically in [8] and implemented (using a distributed ledger) in [42].

Hyperion, like *Selene*, provides a direct and intuitive way for voters to verify their votes, however, it does away with the need for trackers. This modification greatly simplifies the setup and, more importantly, voters should feel much more comfortable about the privacy of their vote. Studies, [2, 22, 48] suggest that some voters are troubled by having their vote appear publicly beside their tracker. We hypothesise that voters will be more comfortable with the *Hyperion* verification, but this needs to be investigated by a complementary user study.

Selene has the problem that the coercer might claim ownership of a faked tracker offered by a coerced voter, or that it coincides with one offered by another victim. Several enhancements to *Selene* to counter this have been suggested including adding extra dummy trackers [39] and shrouding parts of the trackers or votes [32]. The *Hyperion* construction presented here, combined with a further innovation: individual bulletin boards, enables a more elegant solution.

Regarding the definitions, our verifiability definition builds on [15], which following [16] is the best choice for our case. Our definition benefits from a detailed model that allows corruption of vote casting and usage of verification devices.

For the ballot privacy definition, the state of the art was summarised in the SoK paper [6] which also presented a game-based definition BPRIV that implies an ideal functionality under certain conditions and which covers all types of tally functions. This definition was further extended to considering more general attacks during vote casting and malicious boards in [17]. Unfortunately, that definition does not capture schemes where the verification happens after the tally as in *Hyperion*. Recently, a new definition was presented in [23] allowing late verification and which also included a machine-checked proof of ballot privacy for Selene. Whereas the last definition would be applicable to *Hyperion*, it does not capture attacks where the attacker has access to whether the voter's verification is successful or not. Since Hyperion allows a direct check of the tallied plaintext vote, this would immediately cause privacy problems if the adversary manages to cast a vote on behalf of the voter. However, the BPRIV type of definitions, especially [23], does not capture these types of attack, and are not well-suited to do this due to being based on a simulated view. Instead, we here go back to a very early definition by Benaloh [5], but update this with inspiration from [23], also taking into account that the adversary can register maliciously generated keys.

2 PRELIMINARIES

In this section, we introduce the notation that will be used throughout the paper, as well as the parties involved in the protocol.

2.1 Notation

This paper includes writing program code. In such code, the assignment operator ' $\leftarrow$ ' assigns to the left-hand side elements the value on the right-hand side. The symbol ' $\twoheadleftarrow$ ' assigns to its left-hand side, a value sampled randomly from a finite set on the right-hand side. All the variables are represented in binary strings and could be appended to each others by using the ' $\|$ ' operator. If m is a variable, then ' $|m|$ ' denotes its length. Variables written in capital letters denote either arrays or sets depending on the context, and elements in arrays are designated by an index number between square brackets. We use the notation $\mathrm{B} \leftarrow [\,]_{\perp}$ to initialise all the elements in the array B with $\perp$. If X and Y are two sets, then we denote $\mathrm{X} \xleftarrow{\cup} \mathrm{Y}$ shorthand for $\mathrm{X} \leftarrow \mathrm{X} \cup \mathrm{Y}$. Similarly, we write $m \xleftarrow{\|} n$ shorthand for $m \leftarrow m \| n$ when m and n are two variables.

We formalise security properties with games written in pseudocode in which, for simplicity, we omit the security parameter. These games invoke an efficient adversary $\mathcal{A}$ with access to some oracles. The games terminate when executing **Stop with** $\cdot$ command. Each game is associated to a certain winning probability. We write $\Pr[G(\mathcal{A})]$ for the probability that game G invoked with adversary $\mathcal{A}$ stops with $\top$. Game codes will be compacted by introducing the instructions **Require** $\cdot$ which stands for ' if not $\cdot$ then **Stop with** $\perp$ ' and **Promise** $\cdot$ which stands for ' if not $\cdot$ then **Stop with** $\top$ '.

2.2 Parties Involved

Election Authority (EA**).** Performs the general election setup, i.e. defines the election parameters, the ballot styles etc. and sets up the initial Bulletin Board.
Bulletin Board (BB**).** We consider an append-only board with a consistent view for all participants.
Voters. Each voter i is identified uniquely with an id_i and holds two secret keys: A signing key used to authenticate the ballot and a verification trapdoor key used to verify the plaintext vote. These keys can be stored on two different devices/apps that assist the voters in casting and verifying their votes.
Registration Authority. Identifies the eligible voters and posts their public keys on BB.
Tally Tellers (TT**).** Are responsible for setting up a shared (threshold) public election key pk_{EA} which will be used for encryption. They also perform a verifiable decryption during the tally phase.
Mix-Net. Is a set of mix tellers that perform a verifiable parallel mix in order to anonymise the votes.

3 DETAILS OF THE SCHEME

In this section, we describe the main variant of *Hyperion*. We note that the *Hyperion* verification mechanism is versatile and could

be incorporated in an existing voting scheme. For concreteness, we present it as a self contained scheme. We depict the protocol in Fig. 7 of appendix C.

3.1 The Setup

The election authority publishes the relevant details of the election including a cryptographic setup of a secure group on the BB. A set of tally tellers create a threshold public key pair for the election $(sk_{\mathsf{EA}}, pk_{\mathsf{EA}})$ and publishes pk_{EA}. We assume here that each eligible voter holds a valid private signing key with corresponding certification key pk_i published along with unique voter identifiers id_i on BB. The unique identifiers enable *universal eligibility verifiability*. We trust the registration authority to set this up correctly[3]. Note that the setup here is much simpler than that of *Selene* which requires additional verified generation, encryption and mixing of tracking numbers.

3.2 Voting

Each voter generates an ephemeral trapdoor key x_i using her device. The public component $h_i \coloneqq g^{x_i}$ will be registered during vote casting, along with a Zero Knowledge Proof of Knowledge (ZKPoK) of x_i. For all proofs that follow, we assume that these proofs are non-malleable and include binding to a unique election identifier and the public election key pk_{EA}. The proofs here should also be bound to the identity id_i of the voters to prevent the public keys from being copied. In our case a simple Schnorr proof [44] is sufficient, made non-interactive via the (strong) Fiat-Shamir transformation [7, 25] and including all the necessary information in the hash for non-malleability.

Voting proceeds as follows: voter i sends her trapdoor key h_i along with a ZKPoK of x_i, an encryption $\{v_i\}_{pk_{\mathsf{EA}}}$ of her vote v_i (e.g. ElGamal [24]) and the well-formedness ZK proofs of encryption, i.e. a proof of the vote be in the correct space and a proof of plaintext-knowledge[4]. Recall that these proofs are non-malleable and bound to the voter *id* to prevent vote copy attacks[5] [18]. The encryption scheme should support verifiable mixing and together with the ZKPs be IND-1-CCA (see appendix A). We denote the concatenation of the ZK proofs by Π_i. Registering the (ephemeral) trapdoor keys at the same time as casting the vote avoids the need for an extra registration phase. All of this is signed, sent to the EA and appended next to the appropriate pk_i on the BB:

$$id_i,\ pk_i,\ \mathsf{sign}_i(\{v_i\}_{pk_{\mathsf{EA}}},\ h_i,\ \Pi_i)$$

3.3 Tallying

Once the voting phase has closed, ballots posted to the BB with valid signatures and proofs are identified. For these, the Tally Tellers now take each trapdoor key h_i and privately raise this to a fresh, random, secret r_i, encrypt it and post the output on BB together with Π_i^{TT}, a ZKPoK of honest construction with knowledge of r_i and the encryption random coins.[6] For ElGamal this proof can be efficiently implemented, see e.g. [9]. The Tellers keep the corresponding g^{r_i} terms secret, they will be needed in the verification phase. The BB now contains, for the rows with valid ballots, the following:

$$id_i,\ pk_i,\ \mathsf{sign}_i(\{v_i\}_{pk_{\mathsf{EA}}},\ h_i,\ \Pi_i),\ \{h_i^{r_i}\}_{pk_{\mathsf{EA}}},\ \Pi_i^{\mathsf{TT}}$$

The pairs $(\{v_i\}_{pk_{\mathsf{EA}}},\ \{h_i^{r_i}\}_{pk_{\mathsf{EA}}})$ are shuffled in parallel by a verifiable mix-net and verifiably decrypted to obtain the final Tally Board

$$v_i,\ h_i^{r_i}$$

together with the ZKP of correct parallel mixing and decryption, e.g. using Verificatum [46]. If an element $h_i^{r_i} = 1$ an error is output which only happens with negligible probability if at least one Tally Teller is honest.

3.4 Notification and Verification

After a suitable delay we can move to the notification phase: g^{r_i} is sent[7] to voter i over a private channel at a randomly chosen time during the notification period. The voter raises this to her secret trapdoor key x_i and finds the match among the $h_j^{r_j}$ terms, so identifying her vote in the tally column.

3.5 Coercion Mitigation

Suppose a coercer instructed voter i to submit the vote v^*.[8] Voter i identifies a row in the tally that contains the pair $(v_k,\ h_k^{r_k})$ s.t. $v_k = v^*$. Using her trapdoor key x_i, she computes the *fake* dual key that will match this row $(h_k^{r_k})^{-x_i}$.

As with *Selene*, care has to be taken in designing the notification channel to avoid a coercer being able to observe the notification of the *real* dual key. In contexts in which we anticipate extreme coercion, where for example the coercer demands access to the channel, we suggest that coerced voters can notify a suitable authority, before the notification phase, to request a particular fake dual key be sent over the channel. In this case it is unwise to also notify the voter of her real dual key as this might be detected by the coercer. This gives rise to a modified flavour of coercion resistance since the coerced voter benefits from coercion resistance at the cost of losing verifiability.

We note that the vote casting method presented here is not fully coercion-resistant, but is software-dependent receipt-free, i.e. like Helios [1] would rely on the vote-casting device or app not leaking the randomness used in the vote encryption. However, *Hyperion* can be combined with different forms of vote-casting to achieve better receipt-freeness e.g. using the BeleniosRF construction, [10]. Also better coercion-resistance can be achieved providing protection against a coercer even trying to vote on behalf of the coerced voter, e.g. using JCJ style credentials [33], see [31] but at the cost of an interactive vote verification.

---

[3] In Estonia, each voter has her keys integrated in her identity card.

[4] A simple choice is Chaum-Pedersen proofs of discrete log equality using OR Sigma protocols for the different vote choices.

[5] Vote copy attacks would undermine coercion resistance with plaintext verification.

[6] This can easily be distributed over the Tally Tellers for ElGamal. For instance, each TT_j posts $\{h_i^{r_{i,j}}\}_{pk_{\mathsf{EA}}}$ together with the appropriate ZKPoK, then these ciphertexts are multiplied together to obtain $\{h_i^{r_i}\}_{pk_{\mathsf{EA}}}$ where $\sum_j r_{i,j} = r_i$. Each Teller then keeps $g^{r_{i,j}}$.

[7] With multiple Tally Tellers, TT_j can send $g^{r_{i,j}}$ to the voter or they can be collected and sent to the voter under encryption of h_i.

[8] This presumes that some votes v^* are cast by other voters otherwise it will, in any case, be evident that voter i did not cast v^*. For techniques to deal with the situation of unpopular candidates, see [32, 40].

3.6 Dispute Resolution

It is possible when verifying that a voter either fails to find the matching term or finds it but the associated vote does not match the vote they cast. The voter should notify this to the appropriate authority for the matter to be investigated.

Possible causes:

(1) The voter's ballot was not correctly posted to the BB.
(2) The voter's device did not encrypt the correct vote.
(3) The voter's ballot was not correctly processed during the mixing and tallying.
(4) The g^{r_i} term was corrupted.

Regarding the first, we should remark that voters should be encouraged to check the presence of their ballot on the BB before tallying starts, as with other E2E V schemes. Early detection of such problems makes them easier to resolve, but *Hyperion* (and indeed *Selene*) is less reliant than conventional E2E verifiable schemes on such checks being performed diligently.

It is of course possible that a voter claims falsely to have found a problem in which case we hit dispute resolution problems: it is not clear whether the problem is with the system, the voter's device or the voter, either lying or mis-remembering. We will discuss mechanisms to resolve disputes in Section 7.4.

4 HYPERION INSTANTIATION

We will here present the algorithms, EASetup, Setup, ValidCred, Vote, ValidBallot, Tally, GetSecret, Publish, Verify, VerifyVote and VerifyBallot, which we will use in the games for privacy and verifiability, and how they are instantiated for Hyperion.

EASetup sets up the secure cyclic DH group (of prime order) (G, g) and creates the threshold public and secret keys $(pk_{\mathsf{EA}}, sk_{\mathsf{EA}})$. Setup uses the EA keys to generate for each *id* a "unique" signing key pair (sk, pk) along with the proof of well-formedness; it also picks a random exponent x_i and computes $h_i \coloneqq g^{x_i}$ along with Π_{x_i} the proof of knowledge of x_i bound non-malleably to the voter *id*. The previous algorithms should be randomized when generating the keys. ValidCred outputs $\top$ if Π_{x_i} is valid, and $\bot$ otherwise. Vote extracts h_i from pk and Π_{x_i}, encrypts the vote v with ElGamal encryption scheme using pk_{EA}, generates the proof of well-formedness and plaintext knowledge Π_v which is bound to the voter *id*, and signs these elements using sk. It finally outputs the signed elements along with the signature as a ballot *blt* and an empty state *st*. ValidBallot verifies the correctness of the signature using pk and the validity of Π_v and Π_x: if both verifications pass, then the function outputs $\top$, otherwise, it outputs $\bot$.

The Tally function has two main jobs, first computing the mix-net inputs while updating the BB, and second inserting some computed values to the decryption mix-nets and outputting the result along with the vote count. In its first functionality, Tally extracts h_i from BB, picks a random exponent r_i for each row i, computes $h_i^{r_i}$, g^{r_i}, internally stores g^{r_i}, then, it computes the encryption $\{h_i^{r_i}\}_{pk_{\mathsf{EA}}}$ with the proof of knowledge of r_i and correct encryption Π_i^{TT} and sends $(\{h_i^{r_i}\}_{pk_{\mathsf{EA}}}, \Pi_i^{\mathsf{TT}})$ to BB. In the second functionality, the pair $(\{h_i^{r_i}\}_{pk_{\mathsf{EA}}}, \{v_i\}_{pk_{\mathsf{EA}}})$ are put through the mix-net and decryption to output $(h_{\sigma_i}^{r_{\sigma_i}}, v_{\sigma_i}, \Pi_{\mathrm{mix}}, \Pi_{\mathrm{dec}})$ as the final tally.

Further, GetSecret outputs g^{r_i}. VerifyVote extracts $h_i^{r_i}$ from the bulletin board, raises the input g^{r_i} to the secret key input x_i and outputs the equality check. Publish outputs the verifiable mix and the decryption of $(v_i, h_i^{r_i})$ along with the BB.

Finally, Verify will verify all public evidence on BB, VerifyBallot will generally verify that a ballot appears correctly on BB for a given voter, however in Hyperion this can often be relaxed to check that some valid ballot has appeared for the given voter *id* which we denote VerifyVoted. In the privacy game we ignore this and it will always output $\top$. By ρ we denote the election result function.

In our privacy games we choose ρ to compute the array of votes created by extracting, from each element in the input array, the last submitted vote in the concatenated sequence.

5 BALLOT PRIVACY

In this section, we introduce the game-based definition of ballot privacy Ballot-Priv. In this definition, we take into account voters having secret credentials *sk* and capture privacy leaks from verification, especially plaintext verification (as in *Hyperion*, *Selene* and the Estonian e-voting system).

Even though ballot privacy is a fundamental property in secure voting, it has been perplexingly hard to come up with a generic definition which, at the same time, supports standard proof techniques and encompasses large classes of voting systems and tally functions. A good overview of game-based definitions can be found in [6], which also concludes with a privacy definition (BPRIV) for general tally functions. BPRIV is however not directly applicable to the current context of post-tally verification. Instead we take advantage of *Hyperion* having a simple tally function, namely revealing all plaintext votes. This means we can use a much earlier definition as starting point, namely Benaloh's definition [5], which was rewritten in modern game-based notation in [6].

Figure 1 is a rework of Benaloh's definition, with inspiration from [17] and especially [23], allowing voters to hold secret key material and adding a verification phase. This verification phase has the potential to introduce privacy attacks, if the adversary has access to whether the verification was successful or not. This is a realistic real-world scenario even without compromised parties since voters might share a failed verification with others, perhaps even on social media.

For transparency and to detect wide-spread attacks such behaviour should even be endorsed, and hence better not invalidate privacy.

We also want to model robust voting systems in which the voting process proceeds even if individual verification fails (as would probably happen in larger elections). In the case of covert attackers against privacy, the definition can easily be updated to punish an adversary for failed verification attempts. In the Ballot-Priv definition, Figure 1, we assume a trusted BB and secure channels between the voters and BB, meaning that an honest ballot will arrive unchanged to BB. We also assume an initial setup giving each voter a unique identity *id*.

First, in line 02, the EA prepares the master keys that are used to generate the voters credentials (lines 06-09), to verify a voter's credentials (lines 07, 11), to allow the voting process (line 16), to

Game Ballot-Priv$^b(\mathcal{A})$
00 $\mathrm{SK}, \mathrm{PK}, \mathrm{ST}, \mathrm{V}^0, \mathrm{V}^1 \leftarrow [\,]_\perp$
01 $\mathrm{HV}, \mathrm{DV} \leftarrow \{\}$
02 $(pk_{\mathsf{EA}}, sk_{\mathsf{EA}}) \leftarrow \mathsf{EASetup}()$
03 $st_{\mathcal{A}} \leftarrow \mathcal{A}_1(pk_{\mathsf{EA}}, \mathrm{PK})$
04 $b' \leftarrow \mathcal{A}_2(st_{\mathcal{A}})$
05 **Stop with** $b = b'$

Oracle$_1$ HonestSetup(id)
06 $(sk, pk) \leftarrow \mathsf{Setup}(id, sk_{\mathsf{EA}}, pk_{\mathsf{EA}})$
07 **Promise** ValidCred(id, pk, pk_{EA})
08 $\mathrm{SK}[id] \leftarrow sk;\ \mathrm{PK}[id] \leftarrow pk$
09 $\mathrm{HV} \xleftarrow{\cup} \{id\}; \mathrm{DV} \xleftarrow{\setminus} \{id\}$

Oracle$_1$ DishonestSetup(id, pk)
10 **Require** $id \notin \mathrm{HV}$
11 **Require** ValidCred(id, pk, pk_{EA})
12 $\mathrm{PK}[id] \leftarrow pk$
13 $\mathrm{DV} \xleftarrow{\cup} \{id\}$

Oracle$_2$ LoR(id, v^0, v^1)
14 **Require** $id \in \mathrm{HV}$
15 $sk \leftarrow \mathrm{SK}[id];\ pk \leftarrow \mathrm{PK}[id]$
16 $(blt, st) \leftarrow \mathrm{Vote}(pk_{\mathsf{EA}}, id, sk, pk, v^b)$
17 **Promise** ValidBallot(BB, blt)
18 $\mathrm{V}^0[id] \xleftarrow{\shortparallel} v^0;\ \mathrm{V}^1[id] \xleftarrow{\shortparallel} v^1$
19 $\mathrm{ST}[id] \xleftarrow{\shortparallel} st;\ \mathrm{BB}[id] \xleftarrow{\shortparallel} blt$

Oracle$_2$ Tally()
20 **Require** $\rho(\mathrm{V}^0) = \rho(\mathrm{V}^1)$
21 Return Tally(BB, $sk_{\mathsf{EA}}, pk_{\mathsf{EA}}$, PK)

Oracle$_2$ VerifyVote(id)
22 **Require** $id \in \mathrm{HV} \cup \mathrm{DV}$
23 $pk \leftarrow \mathrm{PK}[id]$
24 $s \leftarrow \mathsf{GetSecret}(id, pk, sk_{\mathsf{EA}}, pk_{\mathsf{EA}}, \mathrm{BB})$
25 if $id \in \mathrm{DV}$ then Return s
26 $(r, \pi) \leftarrow \mathrm{Tally}()$ // scheme dependent
27 $sk \leftarrow \mathrm{SK}[id];\ st \leftarrow \mathrm{ST}[id]$
28 Return VerifyVote($id, sk, st, s, \mathrm{BB}, r, \pi$)

Oracle$_2$ Board()
29 $\mathrm{BB}' \leftarrow \mathrm{Publish}(\mathrm{BB})$
30 Return BB'

Oracle$_2$ Cast(id, blt)
31 **Require** $id \in \mathrm{HV} \cup \mathrm{DV}$
32 **Require** ValidBallot(BB, blt)
33 $\mathrm{BB}[id] \xleftarrow{\shortparallel} blt$

Oracle$_2$ VerifyBlt(id)
34 **Require** $id \in \mathrm{HV}$
35 $sk \leftarrow \mathrm{SK}[id];\ pk \leftarrow \mathrm{PK}[id]$
36 $st \leftarrow \mathrm{ST}[id]$
37 Return VerifyBallot(id, st, sk, pk, BB)

Figure 1: The game-based security definition of Ballot Privacy. The adversary wins if it distinguishes the left world from the right one, i.e. if it guesses the bit b. In line 16, the Left-or-Right (LoR) oracle either inputs the left vote v^0 or the right one v^1 based on the bit b. To simplify the notation, we divide our adversary into $\mathcal{A}_1$ and $\mathcal{A}_2$ in 03, 04 and assume that they respectively have access to the oracles sub-indexed by 1 and 2.

tally the BB (line 21) and to allow the generation of the voters' verification secrets (line 24).

Lines 10-13 give the adversary the possibility to dynamically register dishonest credentials for some voters: lines 09 and 10 prevent the adversary from registering a set of credentials as both honest and dishonest at the the same time e.g. by calling HonestSetup on a specific id and then DishonestSetup on the same id, causing the voter to be honest and dishonest at the same time. Notice that, for a voter $id \in \mathrm{ID}$, checks of honesty occur in lines 14, 25, 34.

Line 17 ensures that the ballots created in the left or right voting oracle are well-formed. Notice that in lines 18-19, the elements are concatenated to the history: this provides more generality then just overwriting the previous value using the $\leftarrow$ operator to accommodate elections that take into consideration the whole history of vote submissions.

In line 20, the ρ function guarantees that both V^0 and V^1 have the same count: this prevents $\mathcal{A}$ from trivially winning by querying $\mathrm{LoR}(v^0, v^1)$, $\mathrm{LoR}(v^0, v^0)$ and then querying Tally(). Additionally, $\mathcal{A}$ is capable of querying the ballot casting oracle (31-33), the board publishing oracle (29-30), the ballot verification oracle (34-37) and the vote verification oracle (22-28). The ballot verification oracle and the vote verification oracle, both, can provide the adversary with extra information about the honest and dishonest voters (secret s and/or verification output).
We define the advantage of the adversary

$$\mathsf{Adv}^{\text{Ballot-Priv}}_{\mathcal{A}} \coloneqq \left|\Pr[\text{Ballot-Priv}^0(\mathcal{A})] - \Pr[\text{Ballot-Priv}^1(\mathcal{A})]\right|$$

The generality of this type of Benaloh definition is limited to certain types of result functions, see [16], which however is fulfilled for Hyperion where we output all votes after mixing. Especially, we notice that, a necessary condition on ρ is that it should fulfill the following relation $\rho(\mathrm{V}_0) = \rho(\mathrm{V}_1) \implies \rho(\mathrm{V}_0 \shortparallel \mathrm{V}') = \rho(\mathrm{V}_1 \shortparallel \mathrm{V}')$. Probably the definition could be extended to general cases, with an assumption of extraction properties of the ballots.

Theorem 5.1. *For all $\mathcal{A}$ playing* Ballot-Priv *(Fig. 1 instantiated with Hyperion, there exists adversaries $\mathcal{B}, C, \mathcal{D}, \mathcal{E}, \mathcal{F}$ such that the following relation holds:*

$$\mathsf{Adv}^{\text{Ballot-Priv}}_{\mathcal{A}} \leq \mathsf{Adv}^{\mathrm{ZK}}_{\mathcal{B}} + \mathsf{Adv}^{\mathrm{EUF-CMA}}_{C} + \mathsf{Adv}^{\mathrm{Mix}}_{\mathcal{D}} + \mathsf{Adv}^{\mathrm{ZK}'}_{\mathcal{E}} + \mathsf{Adv}^{\text{poly-IND-1-CCA}}_{\mathcal{F}}$$

Further, in App. B we will also prove that our scheme satisfies du-mb-BPRIV against a malicious board from [17, 23]

Theorem 5.2. *For all $\mathcal{A}$ playing* du-mb-BPRIV *(Fig. 6 instantiated with Hyperion, there exists adversaries $\mathcal{B}, \mathcal{D}, \mathcal{D}', \mathcal{E}, \mathcal{F}$ such that the following relation holds:*

$$\mathsf{Adv}^{\text{du-mb-BPRIV}}_{\mathcal{A}} \leq \mathsf{Adv}^{\mathrm{ZK}}_{\mathcal{B}} + \mathsf{Adv}^{\mathrm{Mix}}_{\mathcal{D}} + \mathsf{Adv}^{\mathrm{Mix}}_{\mathcal{D}'} + \mathsf{Adv}^{\mathrm{ZK}'}_{\mathcal{E}} + \mathsf{Adv}^{\text{poly-IND-1-CCA}}_{\mathcal{F}}$$

The main difference for the bounding is that du-mb-BPRIV does not capture attacks for the verification success seen as a side-channel to the adversary, and hence the ballot signatures are not necessary.

5.1 Proof of Ballot Privacy

We now prove that our scheme meets Ballot-Priv property. In order to do so, we instantiate the algorithms as described in section 4. We use game hopping technique to bound the adversary advantage. Since oracles can be called multiple times by the adversary, we will suppress pre-factors in the advantage bounds. We note by G_0 the instantiated Ballot-Priv game: $\mathsf{Adv}^{\text{Ballot-Priv}}_{\mathcal{A}} = \mathsf{Adv}^{G_0}_{\mathcal{A}}$.

In the first game hop, we remove line 07 in G_1. In fact under the assumption stated in 3.1 we have that line 07 will always pass, and

thus $\mathsf{Adv}^{G_0}_{\mathcal{A}} = \mathsf{Adv}^{G_1}_{\mathcal{A}}$.
In G_2, by the zero knowledge property, the proofs Π_x and Π_{r_i} are simulated for honest voters. This is possible since these proofs are created by the challenger. Now, the adversary cannot extract any information from the simulated proofs and $\mathsf{Adv}^{G_1}_{\mathcal{A}} \leq \mathsf{Adv}^{G_2}_{\mathcal{A}} + \mathsf{Adv}^{\mathrm{ZK}}_{\mathcal{B}}$ ($\mathsf{Adv}^{\mathrm{ZK}}_{\mathcal{B}}$ is the advantage of $\mathcal{B}$ to distinguish simulation from real proofs).
In game G_3, we modify line 31 to require $id \in \mathrm{DV}$ only. In this case, because of the requirement in 32, when casting a ballot for an honest voter, adversary has to be capable of forging a valid signature for the honest voter. If the signature scheme is existentially unforgeable, we have $\mathsf{Adv}^{G_2}_{\mathcal{A}} \leq \mathsf{Adv}^{G_3}_{\mathcal{A}} + \mathsf{Adv}^{\mathrm{EUF-CMA}}_{C}$.
In the fourth game G_4, we replace the verification step on line 28 to always output success. Because our scheme is correct, and since the adversary is not capable of submitting ballots on behalf of honest voters, then we have $\mathsf{Adv}^{G_3}_{\mathcal{A}} = \mathsf{Adv}^{G_4}_{\mathcal{A}}$.
In game G_5, we modify line 21 in the tally oracle: rather than picking r_i at random for each voter and computing $h_i^{r_i}$ then encrypting the computed value, we sample a uniformly random group element g_i and then encrypt it. Since we are working in a cyclic group of prime order, then the distributions of g_i and $h_i^{r_i}$ are exactly the same for all registered voters i, thus $\mathsf{Adv}^{G_4}_{\mathcal{A}} = \mathsf{Adv}^{G_5}_{\mathcal{A}}$. [9]
In the next game G_6, we modify again line 21, by replace the secure mix-net by its ideal functionality. We thus have $\mathsf{Adv}^{G_5}_{\mathcal{A}} \leq \mathsf{Adv}^{G_6}_{\mathcal{A}} + \mathsf{Adv}^{\mathrm{Mix}}_{\mathcal{D}}$. [10]
In G_7, analogously to G_2, we simulate the decryption proofs Π_{dec} for all the ciphertexts output by the mix-net. Further, for the honest voters, the decryption values for the plaintext votes are taken from the calls to the LoR oracle. Due to the correctness of the encryption scheme, the adversary's advantage is $\mathsf{Adv}^{G_6}_{\mathcal{A}} \leq \mathsf{Adv}^{G_7}_{\mathcal{A}} + \mathsf{Adv}^{\mathrm{ZK}'}_{\mathcal{E}}$.
In the final game hop, we require that the mix-nets in G_8 output the honest votes (taken from the LoR oracle) concatenated with the decryption of the dishonest votes. The views in the left world and the right world should be the same, thus we require the decryption mix to output $\rho(\mathrm{V}^b)$ concatenated with the dishonest votes. We have that $\mathsf{Adv}^{G_8}_{\mathcal{F}} = \mathsf{Adv}^{G_7}_{\mathcal{A}}$.
Finally, we argue that the advantage of the final game is exactly $\mathsf{Adv}^{\text{poly-IND-1-CCA}}_{\mathcal{F}}$ in Fig. 4 since we can remove all simulated proofs. The label is the *id* of the voters. We assume that the the encryption scheme with the non-malleable proofs of plaintext knowledge including the *id* satisfy poly-IND-1-CCA security.
We conclude the following:

$$\mathsf{Adv}^{\text{Ballot-Priv}}_{\mathcal{A}} \leq \mathsf{Adv}^{\mathrm{ZK}}_{\mathcal{B}} + \mathsf{Adv}^{\mathrm{EUF-CMA}}_{C} + \mathsf{Adv}^{\mathrm{Mix}}_{\mathcal{D}} + \mathsf{Adv}^{\mathrm{ZK}'}_{\mathcal{E}} + \mathsf{Adv}^{\text{poly-IND-1-CCA}}_{\mathcal{F}}$$

6 INTEGRITY

6.1 Correctness

We first note that the scheme satisfies correctness in the sense that if the voting protocol is run honestly, the tally will give the correct result on the intended votes and all voters will verify correctly. This assumes correctness of the underlying zero-knowledge proofs, signatures, mix-net and correctness of the encryption scheme. This could be relaxed to non-perfect correctness if needed.

6.2 Verifiability

For an overview of verifiability definitions see [16], especially we will use the definition in [15]. This is because the specific voting-casting construction that we instantiate *Hyperion* with here is close to Helios-C (i.e. Helios with signatures) presented and proven verifiable in [15].

The election schemes in [15] are defined via algorithms Setup = EASetup, Credential = Setup, Vote, $\mathsf{VerifyVote}_{\mathrm{CGGI}}$, Validate = ValidBallot, Tally, Verify, where we have indicated by which algorithms they correspond to in our scheme, see Sec. 4. Verify will simply check all proofs on BB. The main difference is in VerifyVote: In Helios-like constructions this corresponds to checking that your actual ballot *blt* has appeared on BB. Here, it corresponds to performing the *Hyperion* verification and will involve getting the dual key from the EA. Since we will define security against a malicious BB, we further need that the voters check that a valid vote was registered under their *id*, but without having to check which specific cryptographic ballot is recorded (for improved usability). We denote this VerifyVoted. As in [15] we consider schemes without vote updates for simplicity.

In [15] combined individual and universal verifiability is defined against a malicious BB. This means the board is completely malicious up until the Tally, where it will be output by the adversary, and there will be a unified view of BB. This also models that vote casting channels might not be secure. The definition is via a game $\mathrm{Exp}^{verb}_{\mathcal{A}}$ for which the adversary has negligible chance in creating a valid BB and tally where it is not true that 1) the vote count will contain the honest verifying voters' votes, 2) for the non-verifying honest voters their votes can maximally be deleted, and 3) there is maximally one vote per corrupted voter in the tally. To count this it is assumed that the result function ρ allows partial tally. The main assumption is that the Registration Authority is honest i.e. signing keys are setup honestly and not leaked and are existentially unforgeable, EUF-CMA. This will also hold for *Hyperion*.

Theorem 6.1. *Hyperion will satisfy Verifiability against a dishonest bulletin board [15] if the signing keys are not leaked, the signature scheme is EUF-CMA and the ballot verification is via* VerifyVoted, *i.e. the voter only checks if a valid vote was cast.*

The proof follows as for Helios-C, however, we use mix-nets instead of homomorphic tally, which still ensures one-vote per voter due to the soundness of the mixes. Also, we can replace $\mathsf{VerifyVote}_{\mathrm{CGGI}}$ with VerifyVoted since the adversary cannot forge a signature. [11]

However, this did not take into account the actual *Hyperion* verification check which also allows a voter to verify if the vote intent was captured directly in the tally. We now extend the verifiability definition to fully incorporate this. The main point will be that an honest checking voter can rely on her vote being counted correctly

[9] We can allow the adversary to have the dual key g^{r_i} for *all voters* in line 28 and still prove security of the scheme under the DDH assumptions: $h_i := g^{x_i}$ adversary cannot distinguish $(g^{x_i}, g^{r_i}, g^{x_i \cdot r_i})$ and (g^{x_i}, g^{r_i}, g_i).
[10] Alternatively, we could model the mix-net as a re-encryption mix with a NIZKP, and use IND-CPA of the encryptions of g_i to ignore these ciphertexts.
[11] Since we are in a single pass setting this is particularly simple. With vote updates more care needs to be taken. Either we need to assume an append only board or that a vote update number is included in the signature and remembered by the voter.

if either her signing key or *Hyperion* secret key is not compromised. In particular, we get a resistance against malware if we have separate devices for vote casting (containing the signing key) and vote verification (containing the *Hyperion* key), and not both devices are corrupted.

We now introduce a verifiability definition against a malicious voting bulletin board in the presence of malware with separate vote casting and vote verification devices. We stress that in the definition it is only the vote casting part of the bulletin board which is determined by the adversary, the registered public keys cannot be altered for honest devices.[12]

The security is defined via the experiment Verif-MBM in Fig. 2, where the advantage of the adversary is

$$\mathsf{Adv}^{\text{Verif-MBM}}_{\mathcal{A}} = \Pr[\text{Verif-MBM}(\mathcal{A}) = 1]$$

The malicious bulletin board and corrupted authorities are modeled by the adversary outputting the bulletin board as well as the tally result and proofs in line 05. $\mathcal{H}_c$ and $\mathcal{D}_c$ respectively denote the voter IDs with honest and corrupted vote casting devices which have registered public verification keys and hence constitute the eligible voters. Correspondingly, $\mathcal{H}_v$ and $\mathcal{D}_v$ are the voter IDs with honest and corrupted vote verification devices. We split the algorithm Setup into a part for the signing key and for the verification key denoted respectively Setup_c and Setup_v, and we do the same split for the ValidCred algorithm.

$\mathcal{V}_i$ denotes the set of voters intending to vote and V_i captures their intended vote, with $\mathbb{V}$ the allowed vote space. $\mathcal{V}_{\text{Chkd}}$ denotes the voters who make successful verification checks. Failing checks would lead to complaints and the adversary loses the game. As in [15], the set of voters who are going to check can be input to the game to capture that not all voters verify. For simplicity we assume only voters with a vote intention will try to verify. Those who check will try to do both VerifyVote and VerifyBallot which we here simplify to VerifyVoted.[13]

We assume VerifyVoted is unaffected by malware since it just requires access to BB (and could be delegated). For VerifyVote, if the voter's verification device is corrupted, or the voter is not registered for verification, the check will be assumed successful.

For uncorrupted devices, since EA is corrupted, the adversary can choose which dual key the voter receives. We do not need a corrupted category since if both devices are corrupted and the voter does not perform verifications then the voter is completely controlled by the adversary. A stronger version can let the adversary choose the election setup, but here it is honestly created.

Note that this definition does not capture the probability of detecting the presence of malware, but the guarantee given to a successfully verifying voter and what votes the adversary can choose for the rest. In particular, the adversary will win if he can output a valid BB and tally and manages to either 1) change the vote of a voter who has at least one honest device and who verified (line 14), or 2) for voters with honest vote casting devices, he manages to stuff votes or change a cast vote in another way than simply deleting it (line 20), or 3) for the remaining eligible voters can cast more than one vote per voter (line 18). The lower bound on votes in the last category comes from the voters with both devices being corrupted, and who are successfully verifying, will know that some vote arrived on their behalf, but not which plaintext vote it contains. Finally, in line 21 the $\star$ denotes the combination of partial tallies in the result function ρ.

The verifiability of *Hyperion* relies on the computational 1-Diffie-Hellman Inversion Problem (1-DHI) [36] for a cyclic prime order group of order q and generator (G, g).

Definition 6.2 (Computational 1-DHI). Given $g^x \in G$ with $x \twoheadleftarrow \mathbb{Z}_q$ compute $g^{1/x}$. Under the *1-DHI assumption* the advantage $\mathsf{Adv}^{1-DHI}_{\mathcal{A}} = \Pr[x \twoheadleftarrow \mathbb{Z}_q : g^{1/x} = \mathcal{A}(g^x)]$ is negligible for all PPT algorithms.

If we use ballot verification VerifyBallot via VerifyVoted, i.e. the voter only checks if a valid vote was cast, then we have the following theorem

Theorem 6.3. *With EUF-CMA signatures, sound mix-nets, encryption correctness, simulation-sound extractability [7, 27] for the proofs of knowledge and under the 1-DHI assumption, the advantage in verifiability against a malicious* BB *and malware,* $\mathsf{Adv}^{\text{Verif-MBM}}_{\mathcal{A}}$ *is negligible.*

6.3 Proof of Verifiability, Theorem 6.3

We here give a short proof, the finite advantage bound can easily be inferred. The proof is done without reference to whether a CRS or RO setup is used. By line 06 and 13 in Fig. 2 we can assume that the adversary outputs a valid BB with a result and valid proof. Since the mix-net proofs validate, by the soundness of the mix-net proofs, decryption proofs and correctness of the encryption scheme, we have that the resulting multiset of votes are equal to the plaintext inputs. For honestly cast ballots we have correctness and they will validate if added to BB. Also, honestly generated key will validate. All votes will be in the correct vote space, this can either be directly checked after decryption or derived from the soundness of the ballot proof of well-formedness.

For all voters in $\mathcal{H}_c$ we have valid signatures if they cast votes and by EUF-CMA the adversary cannot forge any signature. Hence for $\mathcal{H}_c$ no votes can be stuffed and cast votes can never be altered, only deleted. Thus for successfully checking voters with honest vote casting device, $\mathcal{V}_{\text{Chkd}} \cap \mathcal{H}_c$, all votes has to appear unaltered (remember $\mathcal{V}_{\text{Chkd}} \subseteq \mathcal{V}_i$ i.e. the checking voters are part of the voters intending to vote), this proves the $\mathcal{V}_{\text{Chkd}} \cap \mathcal{H}_c$ part of line 14. For the remaining cast votes from voters in $(\mathcal{V}_i \setminus \mathcal{V}_{\text{Chkd}}) \cap \mathcal{H}_c$ the adversary can choose which to delete, ensuring line 20.

We can now consider the voters with a malicious vote-casting device $\mathcal{D}_c$. If these voters are not checking, we have no guarantees. If they check and have a corrupted verification device, then there has to be a ballot for their *id* due to VerifyVoted, however there is no guarantee which vote it contains. This explains the lower bound on the number of maliciously created ballots in line 18.

Finally, we need to consider voters successfully verifying with an uncorrupted verification device. We want to show that they will be able to verify their plaintext vote, hence proving the $\mathcal{V}_{\text{Chkd}} \cap \mathcal{H}_v$ part of line 14 and the upper bound in line 18. We first simulate the ZKPoKs of x_i for the honestly registered Hyperion keys $h_i = g^{x_i}$.

[12] In practice this can be secured in a full malicious board setting by forwarding the public keys to proxies at registration time, who will check later that these appear correctly.

[13] The definition can use VerifyBallot by defining which part of the voter state the adversary can control.

Game Verif-MBM($\mathcal{A}$)
00 $\text{SK}_c, \text{PK}_c \leftarrow [\,]_\perp$
01 $\text{SK}_v, \text{PK}_v, \text{ST}, \text{V} \leftarrow [\,]_\perp$
02 $\mathcal{H}_c, \mathcal{D}_c, \mathcal{H}_v, \mathcal{D}_v, \mathcal{V}_i \leftarrow \{\}$
03 $(pk_{\text{EA}}, sk_{\text{EA}}) \leftarrow \text{EASetup}()$
04 $st_{\mathcal{A}} \leftarrow \mathcal{A}_1(pk_{\text{EA}}, \text{PK})$
05 $(\text{BB}, r, \pi, st_{\mathcal{A}}) \leftarrow \mathcal{A}_2(st_{\mathcal{A}})$
06 **Require** $\text{Verify}(\text{BB}, r, \pi)$
07 for $id \in \mathcal{V}_{\text{Chkd}} \cap \mathcal{V}_i$
08 **Require** $\text{VerifyVoted}(id, \text{BB})$
09 if $id \in \mathcal{H}_v$ then:
10 $sk \leftarrow \text{SK}_v[id], s \leftarrow \mathcal{A}_3(st_{\mathcal{A}})$
11 $v \leftarrow \text{V}[id]$
12 **Require** $\text{VerifyVote}(id, sk, v, s, \text{BB}, r, \pi)$
13 **Require** $r = \top$
14 $\mathcal{M} \leftarrow \mathcal{V}_{\text{Chkd}} \cap (\mathcal{H}_c \cup \mathcal{H}_v)$
15 $a \leftarrow |\mathcal{V}_{\text{Chkd}} \cap \mathcal{D}_v|$
16 $b \leftarrow |\mathcal{D}_c \setminus (\mathcal{V}_{\text{Chkd}} \cap \mathcal{H}_v)|$
17 **Require**
18 $\exists n \in \{a, \ldots, b\}$
19 $\exists v_1, \ldots, v_n \in \mathbb{V}$ // vote set
20 $\exists \mathcal{S} \subseteq (\mathcal{V}_i \setminus \mathcal{V}_{\text{Chkd}}) \cap \mathcal{H}_c$
21 s.t. $\text{r}= \rho([\text{V}[j]]_{j\in\mathcal{S}}) \star \rho([v_j]_{j=1}^{n}) \star \rho([\text{V}[j]]_{j\in\mathcal{M}})$
22 **Stop with** $\top$

Oracle$_1$ $\text{HonestCastSetup}(id)$
23 **Require** $id \notin \mathcal{D}_c$
24 $(sk, pk) \leftarrow \text{Setup}_c(id, pk_{\text{EA}})$
25 **Promise** $\text{ValidCred}_c(id, pk, pk_{\text{EA}})$
26 $\text{SK}_c[id] \leftarrow sk$; $\text{PK}_c[id] \leftarrow pk$
27 $\mathcal{H}_c \xleftarrow{\cup} \{id\}$

Oracle$_1$ $\text{DishonestCastSetup}(id, pk)$
28 **Require** $id \notin \mathcal{H}_c$
29 **Require** $\text{ValidCred}_c(id, pk, pk_{\text{EA}})$
30 $\text{PK}_c[id] \leftarrow pk$
31 $\mathcal{D}_c \xleftarrow{\cup} \{id\}$

Oracle$_1$ $\text{HonestVerSetup}(id)$
32 **Require** $id \notin \mathcal{D}_v$
33 $(sk, pk) \leftarrow \text{Setup}_v(id, pk_{\text{EA}})$
34 **Promise** $\text{ValidCred}_v(id, pk, pk_{\text{EA}})$
35 $\text{SK}_v[id] \leftarrow sk$; $\text{PK}_v[id] \leftarrow pk$
36 $\mathcal{H}_v \xleftarrow{\cup} \{id\}$

Oracle$_1$ $\text{DishonestVerSetup}(id, pk)$
37 **Require** $id \notin \mathcal{H}_v$
38 **Require** $\text{ValidCred}_v(id, pk, pk_{\text{EA}})$
39 $\text{PK}_v[id] \leftarrow pk$
40 $\mathcal{D}_v \xleftarrow{\cup} \{id\}$

Oracle$_2$ $\text{Vote}(id, v)$
41 **Require** $id \in \mathcal{H}_c \cup \mathcal{D}_c$
42 $\mathcal{V}_i \xleftarrow{\cup} \{id\}$; $\text{V}[id] \leftarrow v$
43 if $id \in \mathcal{H}_c$ then
44 $sk \leftarrow \text{SK}[id]$; $pk \leftarrow \text{PK}[id]$
45 $(blt, st) \leftarrow \text{Vote}(pk_{\text{EA}}, id, sk, pk, v)$
46 $\text{ST}[id] \leftarrow st$
47 Return blt

Figure 2: The game-based definition of verifiability against a malicious voting board and malware. The indices on the oracles denote which adversary can use them. We use sub-index v to denote verify, c for cast and i for intended.

We will give a proof by contradiction, i.e. we assume that a voter in $\mathcal{V}_{\text{Chkd}} \cap \mathcal{H}_v$ will get pointed to another vote than her intended vote by the Hyperion verification with some non-negligible advantage $\text{Adv}_{\mathcal{A}}$. We will use this to create an adversary against computations 1-DHI. To this end, we take a 1-DHI challenge g^x and use this as the key for a random voter in $\mathcal{V}_{\text{Chkd}} \cap \mathcal{H}_v$ with a simulated proof. Since the key g^x is indistinguishable from random this voter will be targeted by the attack with probability at least $1/|\mathcal{V}_{\text{Chkd}} \cap \mathcal{H}_v|$.

For all the voters with corrupted casting devices, we now extract their secret keys x_i from the ZKPoKs using the simulation sound extractability (for the honest verification devices, we know their secret keys). Let α denote the dual key term sent by the adversary to the voter. By the soundness of the ZKPoK for the encryption of the elements $h_i^{r_i}$, the soundness of the mix-net and the correctness of the encryption, the output commitments are all of the form $h_i^{r_i}$.

We further extract all r_is from the ZKPoKs. If the voter gets pointed to another vote we have that $\alpha^x = h_i^{r_i} = g^{x_i r_i}$ for some i with $x_i \neq x$. We don't know which i this is, but we guess at random between the k choices. Hence we can compute $\alpha^{1/(x_i r_i)}$ which will be equal $g^{1/x}$ with a non-negligible probability $\text{Adv}_{\mathcal{A}}/(|\mathcal{V}_{\text{Chkd}} \cap \mathcal{H}_v| \cdot k)$ breaking the computational 1-DHI assumption and concluding the proof.

7 VARIATIONS ON AN ORIGINAL THEME

In this section we sketch variants and extensions of the *Hyperion* scheme. Some of these variants may be better suited to certain contexts and threat environments. Full details of and analysis of these variants will appear elsewhere, but we include the outlines here to illustrate the potential of the *Hyperion* construction.

7.1 Lightweight Variant, PQ Privacy and Everlasting Privacy

The main version of *Hyperion* is based on encrypting the voter's trapdoor key h_i, then blinding it with an extractable exponent r_i under the encryption to reach $\{h_i^{r_i}\}_{pk_{\text{EA}}}$. These terms are mixed in parallel with $\{v_i\}_{pk_{\text{EA}}}$ to reveal the mixed pairs v_i, $h_i^{r_i}$. The reason for using this construction is two-fold: first, we can use standard mix-net constructions already implemented and readily available, e.g. Verificatum [46], second, the collection and transmission of the g^{r_i} terms is straight-forward, also in the distributed case with several parties involved in the tally, since they are created for known voters before mixing.

In principle, it would be sufficient to have a verifiable mix-net that takes the pairs $(\{v_i\}_{pk_{\text{EA}}}, h_i)$ as input and outputs the mixed pairs $(v_i, h_i^{r_i})$ together with a proof of correct mixing including a proof of knowledge of r_i. Furthermore, the mix-net protocol needs to be able to output g^{r_i}, however, it is sufficient that this is done privately and without proof.

There are two main advantages of this latter construction. First, the group elements h_i and $h_i^{r_i}$ do not have to be protected by encryption. Second, the terms $h_i^{r_i}$ are information-theoretically indistinguishable from random when no side-channel information is leaked, i.e. the adversary solely knows the information published on the BB (e.g. no information about the voter's privately received value g^{r_i}). This variant constitutes a lightweight version of *Hyperion* since we do not need to encrypt the ElGamal terms and the mixing proof can be made more efficient.

The efficiency gain stays decent when using ElGamal encryption, nevertheless, considering a future quantum adversary (harvest

now, decrypt later), the lightweight version is expected to be significantly more efficient than the original one. In this case, one must replace the classical encryption of the (possibly single bit) vote, with a quantum-resilient encryption scheme. In this version, one does not have to accommodate large message spaces holding ElGamal terms. Additionally, by using a quantum-resilient encryption scheme, the lightweight variant ensures privacy against a future quantum attacker, who only has access to the data on the BB. In fact, it is believed that such attacker can neither break the encryption, nor the zero-knowledge nature of the proofs (which is the case for standard simulatable NIZK proofs).

Going further, we predict that we can achieve *Everlasting Privacy*, i.e. information theoretical protection against a computationally unbounded future attacker seeing BB using the PPAT construction [19], see also [26] for everlasting mixes), since the exponentially randomised trapdoor keys are information-theoretically-hiding the voter's keys.

Let us now return to the point of how to have several nodes for exponential mix-nets which, at each mixnode, takes group elements g_i as inputs and outputs $g^{r_i}_{\pi(i)}$, for a permutation π. The problem is that the later nodes do not know which voter their g^{r_i} belongs to. However, after the mixing has been done in public on BB, the mixnodes can run a mix-net in the reverse order (using the inverse permutation) *internally* starting from encryptions of g to obtain $\{g^{r_1 \cdots r_k}\}_{pk_{\mathsf{EA}}}$ for each voter, where k is the number of mix-nodes. These terms can be internally decrypted and sent privately to the voters. It is important that the internal mix-net cannot give rise to privacy attacks. This can be insured if the internal mixers prove knowledge of the exponents and that these are non-zero, and that each mix node checks the proofs of the previous mix nodes.

A final consideration for less critical elections to achieve more lightweight constructions is to replace the parallel verifiable mix-net with the marked mix-net construction [35], see also [29] which also discusses a post-quantum version, and [3] for a recent lattice-based mix-net. This will not be universally verifiable, but still preserves privacy. However, the individual verification via the *Hyperion* mechanism is still possible, and if the mix-net outputs a proof of knowledge of the r_is, then individual verifiability still holds, as in the proof of Sec. 6.3, in the sense that the checks of honest voters are injective, i.e. two honest voters cannot be pointed to the same vote. To be more precise, mix server j outputs $y_i^{(j)} = g^{r_j r_{j-1} \cdots r_1 x_{\pi_j \cdots \pi_1(i)}}$ and a proof of knowledge of r_j being the exponent of one of the $y_i^{(j-1)}$s output by mix server $j-1$ for $i = 1, \ldots, n$, with n the number of voters.

7.2 Individual Views

As mentioned in the introduction, *Hyperion* does not immediately solve the tracker collision threats present in *Selene*, but combined with a further innovation, that of *individual voter views*, it does.

For each voter the final column of the *Hyperion* tally is subjected to a further, independent, verifiable shuffle. These will actually be exponentiation mixes: for voter i, the $h_j^{r_j}$ terms are all raised to a single secret exponent s_i and shuffled. This should of course come with a ZKPoK of s_i proving that this exponent is the same for all terms in a single view as is standard for exponential mixes, e.g. via proofs of equal discrete logs. For each voter (view) a different s_i exponent will be used. Only one dual key per individual board is released, namely $g^{r_i \cdot s_i}$, which is sent to voter i for which the individual board was created. On receipt of the $g^{r_i \cdot s_i}$ term, voter i can again identify her row in the tally by raising this to x_i. However, the coercer's $g^{r_c \cdot s_c}$ will be completely independent of the randomisation of the voter's view, and so will not match any terms in this view.

A coerced voter V_i can identify a row in the master *BB* with the coercer's required vote and compute the corresponding fake dual key and pass this to the tracker notification authority over a private channel. The authority will then raise this fake dual key to s_i and send this to V_i in due course. In the event that the coercer shoulder surfs when the voter performs the verification, this will point to the coercer's vote in the individual view.

7.3 Version Retaining Trackers

We here observe that we could in fact retain trackers in our construction. This may inspire a greater sense of assurance in the verification, but maybe at the cost of undermining the voter's sense of privacy. Such a trade-off may be appropriate in some contexts. In this case, we assign trackers, tr_i for voter i, in the setup phase, as in *Selene*. Under encryption they can be combined with the public key term $h_i^{r_i}$ to obtain a trapdoor commitment to the tracker:

$$id_i,\ pk_i,\ \mathsf{sign}_i(\{v_i\}_{pk_{\mathsf{EA}}},\ h_i,\ \Pi_i),\ \{h_i^{r_i} \cdot tr_i\}_{pk_{\mathsf{EA}}},\ \Pi_i^{\mathsf{TT}}$$

As above, after a parallel mix we reveal the pairs $(v_i,\ h_i^{r_i} \cdot tr_i)$ as the Tally board. Now, when voter i receives g^{r_i}, she raises this to her trapdoor key x_i and divides this into all the trapdoor commitments. For exactly one, a valid tracker is revealed, her tracker; and all others will yield random elements of the group.

An appealing feature of this is that a coerced voter chooses a row containing the coercer's vote and computes a fake dual key that opens the trapdoor commitment to **her own** tracker. Thus, there is no need for a coerced voter to identify a fake tracker, and thus no possibility that it will collide with the coercer's. This means that voters can actually be notified of their tracker value ahead of time, but of course the dual key still needs to be withheld until after posting of the tally. Indeed, even more surprisingly, the association of trackers with voters could be made public! This is not necessary and would probably be psychologically a bit disconcerting for the voters, but it does have the advantage of demonstrating that all the trackers are distinct. Note that in this construction each voter sees only their own tracker in their own view, and it will not appear in any other voter's view. Nowhere are all the trackers displayed alongside the votes, as was the case with *Selene*, so the privacy concerns of *Selene* should be lessened.

Finally we note that we could incorporate both trackers and individual views into the *Hyperion* construction. Which of these variants would be appropriate in what context will depend on voters' sense of security and assumed threat model and will require further investigation. For example, in contexts in which coercion threats are deemed low, it should be sufficient to use *Hyperion* with just the single bulletin board, without the individual views. In large elections the likelihood of a coerced voter hitting the coercer's commitment is lower. Of course this does not stop the coercer from

maliciously **claiming** that the voter has identified his commitment, but with individual views the voter can be sure that this is a bluff.

7.4 Hyperion with Codes for Dispute Resolution

Hyperion, and tracker based schemes in general, is weak in terms of the resolution of disputes arising from the cast-as-intended verification mechanism. A voter may falsely or mistakenly claim that the vote she finds against her tracker is not the vote she cast. As the scheme stands, it is difficult the establish whether such a challenge is true or false, i.e. whether it is the system or the voter at fault. A barrage of false challenges could serve to undermine credibility of the system and election even if the announced result is in fact correct. Thus, although it is a second order property in some sense, dispute resolvability is nonetheless an important one.

An approach to addressing this is to incorporate codes into *Hyperion*. Code Voting was first proposed by Chaum [11] and involves generating and sending over an assumed secure channel, such as the post, individual code sheets with random codes against each candidate/option to each voter. The voter now enters the appropriate code into her device and this is sent to the vote server. This helps protect both the integrity and privacy of votes over the internet channels.

In addition to voting codes, some systems incorporate *return codes*: once the system has registered the vote it should look up the corresponding return code and send this back to the voter, who can then check that it matches the code shown on their code sheet. In the Swiss system the voter should then, assuming that the return codes match, send in a confirmation code, read off their code sheet, to finalise the casting of the vote. Note that it suffices to have a single confirmation code per code sheet. In Remotegrity [47] the finalisation code is under a scratch field to ensure dispute resolution by being able to capture a collusion in the election system using the codes to vote on behalf of the voter. Such a scratch code could also be used with Hyperion.

Pretty Good Democracy (PGD) [41], proposes spreading the trust in the server by threshold secret sharing the codes amongst a set of trustees. Here the return code will only be revealed if a threshold set of the tellers cooperate to register the vote on the BB. Thus, receipt of the correct return code should assure the voter that the vote is correctly posted to the BB and so correctly entered into the tally. PGD suggests, in order to lessen coercion threats, the possibility of using a single return code per code sheet, but for dispute resolution it is better to use different return codes for each choice.

Here we propose to use a construction similar to the Swiss Post, using return codes and a single confirmation code but using techniques from PGD [41]. Furthermore, rather than the server simply sending the voter a return code that matches the code on the code sheet, it will additionally send a dispute resolution code (DRC) that does not appear on the code sheet and which is unique per candidate. The DRC will act much like the codes revealed to voters in Scantegrity II, [12], and knowledge of this code will further support the voter's challenge in the event of a dispute.

As with PGD and Scantegrity, the consistency of the codes committed to the BB and those printed on the code sheets will have to be enforced by independent entities performing random audits of the setup phase.

In the event of a dispute, the voter is asked to reveal the DRC and, *in camera* the code for the voter's claimed vote committed to the BB is revealed. If these match this provides strong support for the voter's claim.

For dispute resolution we have proposed above the use of return codes, *DRC* codes and a confirmation code, but avoiding voting codes. The latter seem less effective against disputes and they pose usability problems. However, voting codes are still useful in protecting the privacy and integrity against corrupted voter devices, so there may be merit in retaining them.

A further argument for incorporating voting codes in *Hyperion* is to provide an additional layer of defence against ballot copying and injection attacks. Countermeasures against such attacks are already incorporated in Hyperion, including ensuring that the voter's id is cryptographically bound to the Zero-Knowledge plaintext awareness proof. However, if we consider other vote casting methods, e.g. that of BeleniosRF [10], the vote codes can also help to make a more efficient solution by already preventing ballot copy attacks without extra zero-knowledge proofs.

7.5 Deniable Verifiable Tracking from Implicitly Authenticated Key Exchange

We note that the term $h_i^{r_i}$, allowing to track the vote, essentially is a Diffie-Hellman key term. We now use this to show that Hyperion-type systems for deniable tracking of votes, or messages in general, can be obtained from implicitly authenticated key exchange protocols fulfilling a certain condition.

Consider a two-move Key Exchange between two parties Alice (A) and Bob (B). This consists of the algorithms $\mathsf{gen}_A, \mathsf{gen}_B$ to derive the messages sent between the parties and two algorithms $\mathsf{derive}_A, \mathsf{derive}_B$ to derive the secret key.

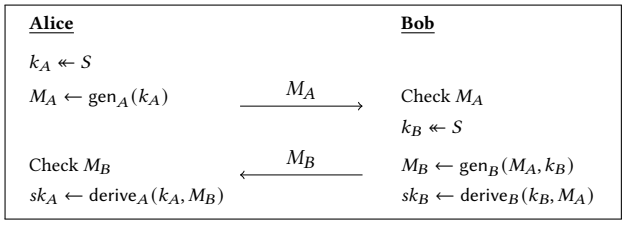

Figure 3: Two-move key exchange

We can now use this to create a Hyperion-type voting protocol. In order to cast a vote, each voter generates $k_i \twoheadleftarrow S$ then computes and publishes $M_i \leftarrow \mathsf{gen}_A(k_i)$ together with their encrypted vote and proofs (of knowledge) for the correctness of M_i and v_i

$$id_i,\ M_i,\ \{v_i\}_{pk_{\mathsf{EA}}}, \Pi_i$$

The voting authority generates for each voter $k_B^i \twoheadleftarrow S$ and computes $\{\mathsf{derive}_B(k_B, M_i)\}_{pk_{\mathsf{EA}}}$ from M_i together with a zero-knowledge proof of knowledge, Π_i^{TT}, that this was done correctly. We then obtain

$$id_i,\ \{\mathsf{derive}_B(k_B^i, M_i)\}_{pk_{\mathsf{EA}}},\ \{v_i\}_{pk_{\mathsf{EA}}}, \Pi_i^{\mathsf{TT}}$$

We then do a verifiable parallel mix of the last two ciphertexts to achieve

$$sk_B^i,\ v_i$$

We could also arrive at this step using the method outlined in Sec. 7.1 without encryption of the secret key. In the real-or-random model the key derived from the key exchange will be indistinguishable from random and hence ballot privacy is preserved.

The authorities then internally computes $M_B^i \leftarrow \text{gen}_B(M_i, k_B^i)$ and sends this privately to the voter i. The voter can now compute $sk_i \leftarrow \text{derive}_A(k_i, M_B^i)$. By the key exchange functionality this should be equal to sk_B^i and the voter can find her vote.

We need two further non-trivial properties from the Key Exchange scheme: (1) For Coercion-mitigation: $\text{derive}_A(k_i, \cdot)$ needs to be invertible, allowing the voter to point to another vote. (2) For Individual Verifiability: It should not be possible to get the same key in two different fresh sessions (even if one party is the same in both sessions), i.e. that two different honest voters will not be pointed to the same vote. This corresponds to demanding that the key exchange satisfies implicit authentication, see [21]. Note that the this assumption only has to hold for honest generation of M_A and sk_B with extraction of both random coins used in the protocol since this is ensured by the public zero-knowledge proofs.

A three-move key exchange protocols could also be used without more interaction on the voter side. Here, we let the election authorities take the role of the party sending two messages, and the election authority posts the first message under encryption and with proper zero-knowledge proofs on the BB before the voter registration step. For key-exchange protocols with more than three moves, the election authority could be online during the registration phase to ensure that the voter can register without having to connect several times.

In order to distribute the creation of the secret key k_B on the election side one could run several key exchanges in parallel and use the XOR of these keys as the final key (e.g. added under homomorphic encryption.) However, in many cases the key exchange will contain homomorphic properties that allows a natural distributed form as in the DH case.

Note that if we base the protocol on a PQ key exchange together with quantum safe encryption schemes, we will be able to achieve verifiability and privacy against present quantum attackers, and not just future adversaries as in section 7.1 above.

8 IMPLEMENTATION

We implement and instantiate *Hyperion* [14] in Python, using the GNU Multiple Precision Arithmetic library, and evaluate its performance on a server equipped with a 32 core AMD EPYC 7302P CPU clocked at 3 GHz and 256 gigabytes of RAM. The implementation is parameterized by the P-256 curve. An implementation of a Terelius-Wikström mixnet [28, 45] was employed for parallel shuffling in the tallying phase. Analogously, we also implement and instantiate *Selene*[15] in order to compare the performance of both schemes; these measurements are provided in appendix E. Table 1 presents measurements collected during the course of 3 trial runs of the *Hyperion* scheme for 1000 voters, 10000 voters, and 100000 voters. We comment that though this is a prototype implementation, the mixnet code has been parallelised to run faster on multi-core systems. The last row of Table 2 lists the computation costs for generating an individual view for a single voter as described in section 7.2. Though this cost scales with N, individual views are generated on demand and the protocol may benefit from the pre-computation of these views. We also provide descriptions of the zero knowledge proofs employed in our implementation, in appendix D.

Phase	$N = 1000$	$N = 10000$	$N = 100000$
Setup	0.0004s	0.0005s	0.0005s
Voting	0.0085s	0.0090s	0.0160s
Tallying (Mix)	42.205s	1541.62s	6886.90s
Tallying (Decrypt)	5.4640s	33.889s	1092.32s
Coercion-Mitigation	0.0008s	0.0008s	0.0007s
Individual Views	14.091s	256.72s	3498.16s

Table 1: Execution times of each phase of the Hyperion scheme in seconds.

9 CONCLUSION

We present a new end-to-end verifiable scheme, inspired by the *Selene* tracker based scheme, that provides a similar, highly transparent, intuitive way for voters to verify their vote: by identifying their vote in cleartext in the tally. Our new construction however allows us to achieve this without the need for trackers and allows us to neatly avoid the tracker collision problem that undermined the *Selene* scheme. The collision threat however could re-emerge as collision of commitments rather than trackers. This prompts a further innovation: the idea of individual voter views, that avoids the collisions threat of *Selene* and should afford voters a greater sense of privacy. Voters should feel more comfortable with *Hyperion* as it does not involve the public posting of all the tracker numbers paired with the votes.

While we do not advocate the use of *Hyperion* for high-stakes elections, we do believe that it is well suited to many less critical contexts. The transparency of the verification and the underlying simplicity of the constructions should be appealing to many stakeholders: the voters, the election officials, the candidates etc. The individual views version introduces some additional computation and complexity, but is efficient for small elections, and in any case could be done on demand when a voters seeks to verify their vote.

We have proven that the system satisfies ballot privacy and verifiability, the latter even under partial malware corruption of the voters' vote casting and verification devices. Finally, we have sketched how the *Hyperion* scheme can be made everlasting private or post-quantum secure. We also outline some possible variants of the core scheme, including the re-introduction of trackers and the use of return or confirmation codes to address dispute resolution.

Future work includes detailing the variants and formally proving them. We will also perform focus groups and user trials to gauge user response and preferences amongst the variants and w.r.t. to *Selene*.

[14] https://github.com/hyperion-voting/hyperion
[15] https://github.com/hyperion-voting/selene

10 ACKNOWLEDGEMENTS

This paper acknowledges the Luxembourg National Research Fund (FNR) CORE project EquiVox (C19/IS/13643617/EquiVox/Ryan) and the CORE project (C21/IS/16221219/ ImPAKT).

REFERENCES

[1] Ben Adida, Olivier De Marneffe, Olivier Pereira, Jean-Jacques Quisquater, et al. Electing a university president using open-audit voting: Analysis of real-world use of helios. *EVT/WOTE*, 9(10), 2009.

[2] Mohammed Alsadi and Steve Schneider. Verify my vote: voter experience. *E-Vote-ID 2020*, page 280, 2020.

[3] Diego F Aranha, Carsten Baum, Kristian Gjøsteen, and Tjerand Silde. Verifiable mix-nets and distributed decryption for voting from lattice-based assumptions. *Cryptology ePrint Archive*, 2022.

[4] Mathilde Arnaud, Véronique Cortier, and Cyrille Wiedling. Analysis of an electronic boardroom voting system. In *International Conference on E-Voting and Identity*, pages 109–126. Springer, 2013.

[5] Josh Daniel Cohen Benaloh. *Verifiable secret-ballot elections*. PhD thesis, Yale University, 1987.

[6] 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, 2015.

[7] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 626–643. Springer, 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] Jan Camenisch. *Group signature schemes and payment systems based on the discrete logarithm problem*. PhD thesis, ETH Zurich, 1998.

[10] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. Beleniosrf: A non-interactive receipt-free electronic voting scheme. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1614–1625, 2016.

[11] David Chaum. Surevote: technical overview. In *Proceedings of the Workshop on Trustworthy Elections, WOTE*, 2001.

[12] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L Rivest, Peter YA Ryan, Emily Shen, Alan T Sherman, et al. Scantegrity ii: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes. *EVT*, 8(1):13, 2008.

[13] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 1992.

[14] Véronique Cortier, Constantin Cătălin Drăgan, 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 (SP)*, pages 993–1008. IEEE, 2017.

[15] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachene. Election verifiability for helios under weaker trust assumptions. In *European Symposium on Research in Computer Security*, pages 327–344. Springer, 2014.

[16] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz Truderung. Sok: Verifiability notions for e-voting protocols. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 779–798. IEEE, 2016.

[17] Véronique Cortier, Joseph Lallemand, and Bogdan Warinschi. Fifty shades of ballot privacy: Privacy against a malicious board. In *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*, pages 17–32. IEEE, 2020.

[18] Véronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. *Journal of Computer Security*, 21(1):89–148, 2013.

[19] Edouard Cuvelier, Olivier Pereira, and Thomas Peters. Election verifiability or ballot privacy: Do we need to choose? In *European Symposium on Research in Computer Security*, pages 481–498. Springer, 2013.

[20] Ivan Damgård. On σ-protocols. *Lecture Notes, University of Aarhus, Department for Computer Science*, page 84, 2002.

[21] Cyprien Delpech de Saint Guilhem, Marc Fischlin, and Bogdan Warinschi. Authentication in key-exchange: Definitions, relations and composition. In *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*, pages 288–303. IEEE, 2020.

[22] Verena Distler, Marie-Laure Zollinger, Carine Lallemand, Peter B Roenne, Peter YA Ryan, and Vincent Koenig. Security-visible, yet unseen? In *Proceedings of the 2019 CHI conference on human factors in computing systems*, pages 1–13, 2019.

[23] 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 *35th IEEE Computer Security Foundations Symposium, CSF 2022, Haifa, Israel, August 7-10, 2022*, pages 335–347. IEEE, 2022.

[24] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[25] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

[26] Kristian Gjøsteen, Thomas Haines, and Morten Rotvold Solberg. Efficient mixing of arbitrary ballots with everlasting privacy: How to verifiably mix the ppatc scheme. In *Nordic Conference on Secure IT Systems*, pages 92–107. Springer, 2020.

[27] Jens Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 444–459. Springer, 2006.

[28] Rolf Haenni, Philipp Locher, Reto Koenig, and Eric Dubuis. Pseudo-code algorithms for verifiable re-encryption mix-nets. In *Financial Cryptography and Data Security*, pages 370–384. Springer, 2017.

[29] Thomas Haines, Olivier Pereira, and Peter B Rønne. Short paper: An update on marked mix-nets: An attack, a fix and pq possibilities. In *International Conference on Financial Cryptography and Data Security*, pages 360–368. Springer, 2020.

[30] Feng Hao and Peter YA Ryan. *Real-world electronic voting: Design, analysis and deployment*. CRC Press, 2016.

[31] Vincenzo Iovino, Alfredo Rial, Peter B Rønne, and Peter YA Ryan. Using selene to verify your vote in jcj. In *International Conference on Financial Cryptography and Data Security*, pages 385–403. Springer, 2017.

[32] Wojciech Jamroga, Peter B Roenne, Peter YA Ryan, and Philip B Stark. Risk-limiting tallies. In *International Joint Conference on Electronic Voting*, pages 183–199. Springer, 2019.

[33] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Towards Trustworthy Elections*, pages 37–63. Springer, 2010.

[34] Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. select: a lightweight verifiable remote voting system. In *Computer Security Foundations Symposium (CSF), 2016 IEEE 29th*, pages 341–354. IEEE, 2016.

[35] Olivier Pereira and Ronald L Rivest. Marked mix-nets. In *International Conference on Financial Cryptography and Data Security*, pages 353–369. Springer, 2017.

[36] Birgit P Pfitzmann and Ahmad-Reza Sadeghi. Anonymous fingerprinting with direct non-repudiation. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 401–414. Springer, 2000.

[37] Ronald L Rivest. On the notion of 'software independence'in voting systems. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3759–3767, 2008.

[38] 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.

[39] Peter Y A Ryan, Peter B Rønne, and Vincenzo Iovino. Selene: Voting with transparent verifiability and coercion-mitigation. In *International Conference on Financial Cryptography and Data Security*, pages 176–192. Springer, 2016.

[40] Peter YA Ryan, Peter B Roenne, Dimiter Ostrev, Fatima-Ezzahra El Orche, Najmeh Soroush, and Philip B Stark. Who was that masked voter? the tally won't tell! In *International Joint Conference on Electronic Voting*, pages 106–123. Springer, 2021.

[41] Peter YA Ryan and Vanessa Teague. Pretty good democracy. In *International Workshop on Security Protocols*, pages 111–130. Springer, 2009.

[42] Muntadher Sallal, Steve Schneider, Matthew Casey, François Dupressoir, Helen Treharne, Catalin Dragan, Luke Riley, and Phil Wright. Augmenting an internet voting system with selene verifiability using permissioned distributed ledger. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 1167–1168. IEEE, 2020.

[43] Bruce Schneier. *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons, 2007.

[44] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[45] Björn Terelius and Douglas Wikström. Proofs of restricted shuffles. In *Progress in Cryptology – AFRICACRYPT 2010*, pages 100–113. Springer, 2010.

[46] Douglas Wikström. User manual for the verificatum mix-net version 1.4. 0. *Verificatum AB, Stockholm, Sweden*, 2013.

[47] Filip Zagórski, Richard T Carback, David Chaum, Jeremy Clark, Aleksander Essex, and Poorvi L Vora. Remotegrity: Design and use of an end-to-end verifiable remote voting system. In *International Conference on Applied Cryptography and Network Security*, pages 441–457. Springer, 2013.

[48] Marie-Laure Zollinger, Verena Distler, Peter B Roenne, Peter YA Ryan, Carine Lallemand, and Vincent Koenig. User experience design for e-voting: How mental models align with security mechanisms. *arXiv preprint arXiv:2105.14901*, 2021.

[49] 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.

A IND-1-CCA

Figure 4 introduces the definition of poly-IND-1-CCA game. This notion is necessary for the ballot privacy proof. We define the adversarial advantage as

$$\mathsf{Adv}^{\text{poly-IND-1-CCA}}_{\mathcal{A}} \coloneqq \left|\Pr[\text{poly-IND-1-CCA}^0(\mathcal{A})] - \Pr[\text{poly-IND-1-CCA}^1(\mathcal{A})]\right|$$

Game poly-IND-1-CCA$^b(\mathcal{A})$
00 $L \leftarrow [\,]_\perp$
01 $(pk, sk) \leftarrow \mathsf{gen}()$
02 $b' \leftarrow \mathcal{A}(pk)$
03 **Stop with** $b = b'$

Oracle $\mathsf{Enc}(m^0, m^1, l)$
04 **Require** $|m^0| = |m^1|$
05 $c \leftarrow \mathsf{enc}(pk, m^b, l)$
06 $L \xleftarrow{+} [(c, l)]$
07 Return c

Oracle $\mathsf{Dec}(c, l)$
08 **Require** $(c, l) \notin L$
09 $m \leftarrow \mathsf{dec}(sk, c, l)$
10 Return m

Figure 4: Labelled poly-IND-1-CCA **game-based definition. The label here could be thought of as a voter's identity.**

B PRIVACY AGAINST A MALICIOUS BOARD

In section 5.1, we present the proof and background for Thm. 5.2. In order to be self-contained, we need to recall the definition of du-mb-BPRIV ballot-privacy against a malicious board in the case where verification happens after the tally has been published, as defined in [23], generalising the definition of [17]. In this definition, the adversary controls the bulletin board BB completely, i.e. what to put on it, but BB has a consistent static view to all voters from the tally time allowing for verification. This leads to obvious privacy attacks e.g. deleting all ballots except for one voter and tally this. Thus, the definition is parameterised with a recovery algorithm which defines the behaviour, we explicitly allow the adversary to do, but ensures privacy besides these allowed manipulations. The manipulations, and corresponding privacy-loss, which we explicitly accept are deletions and reordering of votes, as in the privacy proof of Selene in [23], but du-mb-BPRIV will only be fulfilled if no further attacks are possible, e.g. ballot copying attacks. The definition from [23] is as follows

Definition B.1. 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 $\mathcal{A}$, the following advantage:

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

is negligible, where du-mb-BPRIV is defined in Fig. 6 and instantiated with $\mathcal{V}$.

We have here written the experiment in Fig. 6 in terms of our notation and algorithms already used for privacy in Sec. 5. Comparing Fig. 1 with Fig. 6, the latter always outputs the recovered result which will be related to the left-hand side (LHS), but uses a simulator for the proofs of the result on the right-hand side (RHS). What leverage the possibility to simulate the decryption proofs after mixing. These are standard discrete logarithm equality proofs, which can be simulated also outside the relation.

The recovery function $\mathsf{Recover}(\mathrm{BB}, \mathrm{BB}^0, \mathrm{BB}^1)$ in Line 32, will in our case replace unaltered ballots from the RHS vote oracle with the LHS oracle outputs, and keep the rest of the ballots unaltered. Hence deletion and ballot re-ordering will not lead to attacks. More formally, we define a selection function as in [17]:

Definition B.2 (Selection function [17]). For integers $m, n \geq 1$, a selection function for m and n is a mapping

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

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 jth element in BB^1,
- $\pi(i) = (pk, c)$ means that this element is (pk, c).

The function $\overline{\pi}$ associated to π, maps a bulletin board BB^0 of length m to a board $\overline{\pi}(\mathrm{BB}^0)$ of length n such that

$$\overline{\pi}(\mathrm{BB}^0)[j] = \begin{cases} (pk, c) \text{ if } \pi(j) = i \text{ and } \mathrm{BB}^0[i] = (id, (pk, c)) \\ (pk, c) \text{ if } \pi(j) = (pk, c) \end{cases}$$

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

We can now define the recovery algorithm as

Definition B.3 (Recovery algorithm [17]). 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 $|\mathrm{BB}^1|$ and n for some integer n.

Proc. $\mathsf{Recover}^{\text{del,reorder'}}_{\mathsf{U}}(\mathrm{BB}^1, \mathrm{BB})$
00 $\mathrm{L} \leftarrow [\,]_\perp$
01 for $(pk, c) \in \mathrm{BB}$:
02 if $\exists j, id$ s.t. $\mathrm{BB}^1[j] = (id, (pk, c))$:
03 $\mathrm{L} \xleftarrow{\shortparallel} j$ // Pick the first j found
04 else: $\mathrm{L} \xleftarrow{\shortparallel} (pk, c)$
05 Return $(\lambda i.\mathrm{L}[i])$

Figure 5: The $\mathsf{Recover}^{\text{del,reorder'}}_{\mathsf{U}}$ **algorithm.**

As in [23] we will write $\mathrm{BB}' \leftarrow \mathsf{Recover}(\mathrm{BB}, \mathrm{BB}^0, \mathrm{BB}^1)$ to denote the process of determining the transformation from BB^1 to BB, and applying this transformation to BB^0, in order to get the board BB'. The recovery function that we will use is defined in Fig. 5.[16]

[16]The reader might ask why we do not consider the other recovery functions from [17] here. The reason is that it implies attacks when verifying voters do not detect changes to the ballots even if the plaintext remains the same. However, for the Hyperion verification mechanism only the plaintext is verified.

In the security experiment, the private data needed for verification has been included into the secret key given to the voters before election by an abuse of the notation for the setup algorithm in line 12, which is possible since it does not depend on the actual cast ballot.

B.1 Proof-Sketch of du-mb-BPRIV Privacy

First, denote the experiments du-mb-BPRIV$^0_{\mathsf{Sim}}(\mathcal{A})$ by G_{0L} and du-mb-BPRIV$^1_{\mathsf{Sim}}(\mathcal{A})$ by G_{0R}. Starting from the G_{0L} we simulate the zero-knowledge proofs Π_x, Π_{r_i} and Π_{dec}. This is possible since these proofs are created by the challenger. We thus reach game G_{1L}. The difference is bounded by $\mathsf{Adv}^{\text{ZK}}_{\mathcal{B}}$, where $\mathcal{B}$ is a reduction to an attacker distinguishing real and simulated proofs .

Without any advantage difference, we can move to a game G_{2L} where the plaintexts corresponding to honestly cast votes from BB^0 are taken from $\mathbf{Oracle}_2$ LoR(id, v^0, v^1). Since the ElGamal encryption scheme is perfectly correct this will always with probability 1 give the same result. The ciphertexts which the adversary has added himself to BB are decrypted as usual.

We then move to game G_{3L} where we replace the mixnet with its ideal functionality including simulating its zero-knowledge proof. We bound the difference by $\mathsf{Adv}^{\text{Mix}}_{\mathcal{D}}$.

We note that G_{3L} essentially corresponds to the left-hand side of the poly-IND-1-CCA game in Fig. 4. This is because the Hyperion terms $h_i^{r_i} = g^{x_i r_i}$ are perfectly random for the honest voters after the simulation of the proof, and hence provide no further information. Also, the mix provides no information about its input after being replaced by its ideal functionality. For the dishonest voters, the adversary can find his corresponding plaintexts via the Hyperion verification, which corresponds to the decryption oracle in the poly-IND-1-CCA game. The adversary can also submit ciphertext on behalf of honest voters which will get decrypted (the only difference compared to poly-IND-1-CCA is that those decryptions are mixed). Further, the adversary gets to know if these decryptions are the same as the intended vote, but this provides no further information, since the adversary has chosen the intended vote and could have checked this himself against the decryption. The verification checks for the honest voters which have their honestly generated ballots will always verify successfully. We thus replace the ballots submitted by honest voters that appear in BB^0 with the corresponding ones from BB^1. The difference is bounded by $\mathsf{Adv}^{\text{poly-IND-1-CCA}}_{\mathcal{F}}$.

This brings us to game G_{3R}. The main idea is that the verification checks for the honest voters do not change even though we change the ciphertexts. The reason is that the voter always verifies according to the LHS plaintext vote, see the state in line 26 in Fig. 6, and the recovery function changes the ballots from BB^1 to those from BB^0. Put differently, the side-channel from verifications do not help the adversary in du-mb-BPRIV contrary to our definition in Sec. 5, where the ballot signatures were necessary for privacy. The advantage difference is bounded by $\mathsf{Adv}^{\text{poly-IND-1-CCA}}_{\mathcal{F}}$

We now deal with the real mix and its non-simulated zero-knowledge proof from G_{2R}. Again this is bounded by $\mathsf{Adv}^{\text{Mix}}_{\mathcal{D}'}$.

Again, we let the ciphertexts come from the decryption rather than the vote oracle. The corresponding game G_{1R} is indistinguishable from G_{2R} due to the perfect correctness of ElGamal scheme.

Finally, we stop simulating the zero-knowledge proofs for Π_x, Π_{r_i} but not Π_{dec}, which has to be simulated by definition, in order to reach G_{0R}. This is bounded by $\mathsf{Adv}^{\text{ZK}'}_{\mathcal{E}}$.

C PROTOCOL DIAGRAM

In this section, we depict *Hyperion* in Fig. 7. In this figure, we omit from representing the election authority in this figure and assume that the cryptographic schemes and parameters are already known by all the four parties in the figure. The algorithms should be instantiated following the specifications in sect. 4; A good candidate of such instantiation can be found in our *Hyperion* implementation. We denote with ZK the computation of the zero-knowledge proofs based on the protocol specification. More details about how to construct the zero-knowledge proofs can be found in sect. D. In this representation, we assume that a public warning will be broadcast when a certain ZK proof, test or a certain signature does not verify properly, or data is ill-formed.

D ZERO-KNOWLEDGE PROOFS USED IN IMPLEMENTATION

We here specify the the non-interactive zero-knowledge proofs that we use in our implementation, see also the protocol diagram in Fig. 7. The proofs are all in the form of Sigma protocol proofs that have been made non-interactive using the Fiat-Shamir transformation [25]. We use the strong Fiat-Shamir transformation as is generally necessary in voting [7]. This means that the three-move Sigma protocol (a, e, z) for a statement x, where a is the commitment from the prover, e the challenge from the verifier and z the final response from the verifier is made non-interactive using a hash function H by letting $e = H(x, a)$. We further strengthen the non-malleability by also including the group generator and the public election key. Even further, when the proofs are for specific voters we also include the voters public Hyperion key to bind the proof to the voter.

The proofs used in the implementation are

- Π_{x_i}: For this proof we use the standard Schnorr proof [44]. Since we use the strong Fiat-Shamir this implies that the proofs are simulation-sound extractable (using Theorem 1 of [7] since the challenge space is exponential in our security parameter). This is needed for verifiability in Theorem 6.3.
- Π_v is a OR proof that the vote is zero or one (yes or no). This proof is based on a Chaum-Pedersen proof for discrete log equivalence [13]. The OR part is done using the standard Sigma protocol method [20]. This constitutes a proof of knowledge of the vote due to the special soundness and hence following Theorem 2 of [7] when we combine it with ElGamal which has IND-CPA security under the DDH assumption, we get an encryption scheme which is IND-1-CCA (NM-CPA in the notation of [7]). The encryption is labelled by the identity of the voter since we include the voter's public key in the hash function. Finally Lemma 1 in [14] gives us that the combined encryption scheme is poly-IND-1-CCA, which we use in the theorems of ballot privacy Theorem 5.1 and 5.2.

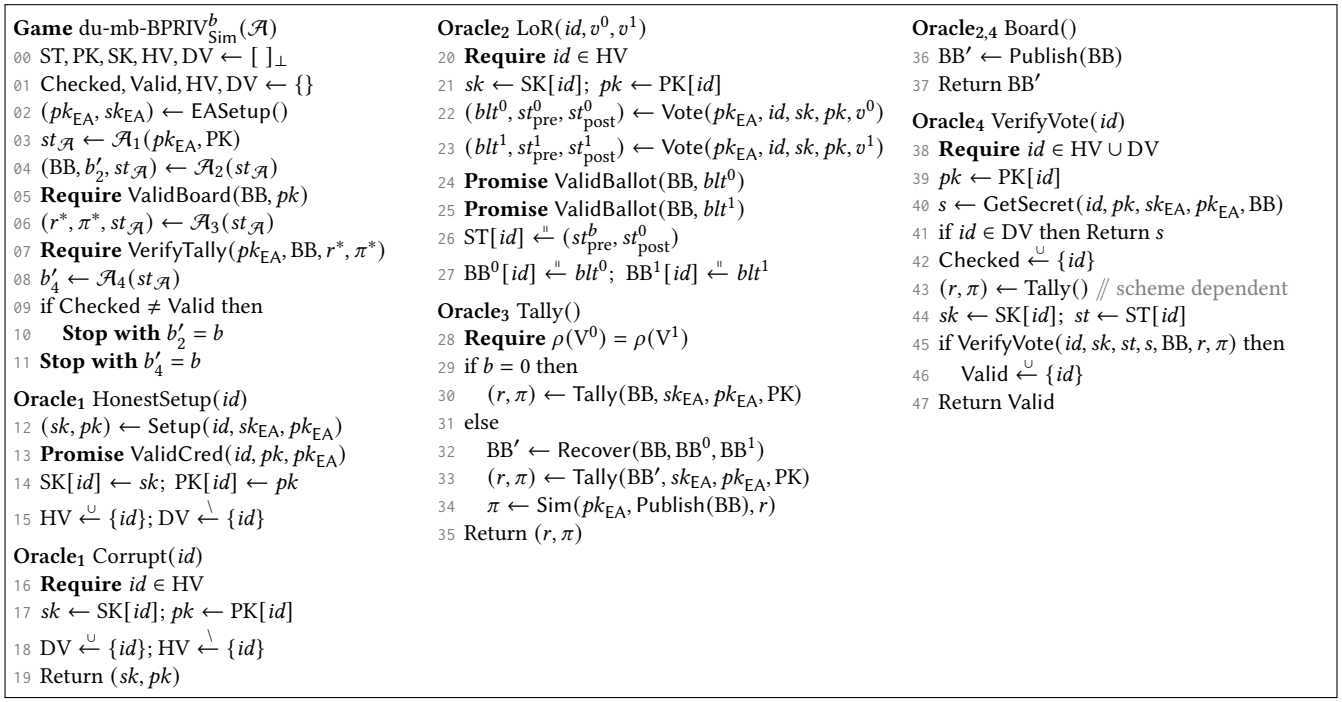

Figure 6: The du-mb-BPRIV **notion for ballot privacy against a dishonest ballot box.**

- Π_i^{TT}: For this proof we use a proof from [9]. Since we use the strong Fiat-Shamir transformation this is again simulation-sound extractable as we need for verifiability.
- Π_{mix}: This is the most advanced proof and the only one which does not follow the Sigma protocol and Fiat-Shamir heurestic. It is an implementation of the proof in the Terelius-Wikström mixnet, see [28, 45].
- Π_{dec} is again a Chaum-Pedersen proof of discete log equivalence.

E ADDITIONAL BENCHMARKS

Table 2 lists the mean execution times (obtained from 100 trial runs) of each phase of the *Hyperion* scheme in seconds, and compares them to the corresponding phases of *Selene*, for 3 tellers and voter counts (N) of 50, 100, and 150. The Setup phase in *Selene* constitutes more than half of the execution time of the entire protocol, and this cost scales linearly with the number of voters, whilst Hyperion benefits from the removal of trackers and exhibits a significantly more efficient Setup phase. This is also evident from Figure 8, which depicts the mean execution times for both protocols for varying voter counts, from the Setup phase through to the Notification phase. The Voting phase produces similar execution times in both schemes. Tallying phase execution times scale linearly in both schemes as the number of voters increase.

F OVERVIEW TRUST ASSUMPTIONS

This Section summarizes the trust assumptions for the privacy and verifiability properties. In Table 9, we list the minimal trust assumptions on the different participants and the allowed leaks. Each voter is assumed to hold a Vote Casting Device (VCD) holding the individual voter signing key sk_i and a Vote Verification Device (VVD) holding the secret *Hyperion* key x_i and receiving the term g^{x_i} used for verification. The Tally Tellers (TT) hold the secret election key sk_{EA}. Trusted parties are marked with 'T' whereas untrusted parties are marked with 'U'. Allowed leaks are marked '✓' and not allowed with '✕'.

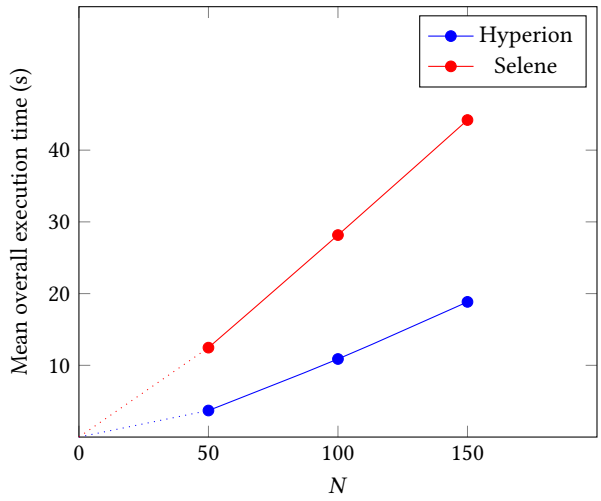

Figure 8: Voter counts plotted against mean overall execution times of the Hyperion and the Selene schemes in seconds.

Security Property	Trust Assumption VCD	VVD	TT	Allowed Leaks sk_i	x_i	g^{x_i}	sk_{EA}
Verif-MBM	U	T	U	✓	✗	✓	✓
	T	U	U	✗	✓	✓	✓
Ballot-Priv	T	T	T	✗	✗	✓	✗
	T	T	T	✗	✓	✗	✗
du-mb-BPRIV	T	T	T	✓	✗	✓	✗
	T	T	T	✓	✓	✗	✗

Figure 9: Minimal trust and leakage assumptions for the individual verifiability Verif-MBM **and the two privacy properties** Ballot-Priv **and** du-mb-BPRIV**.**

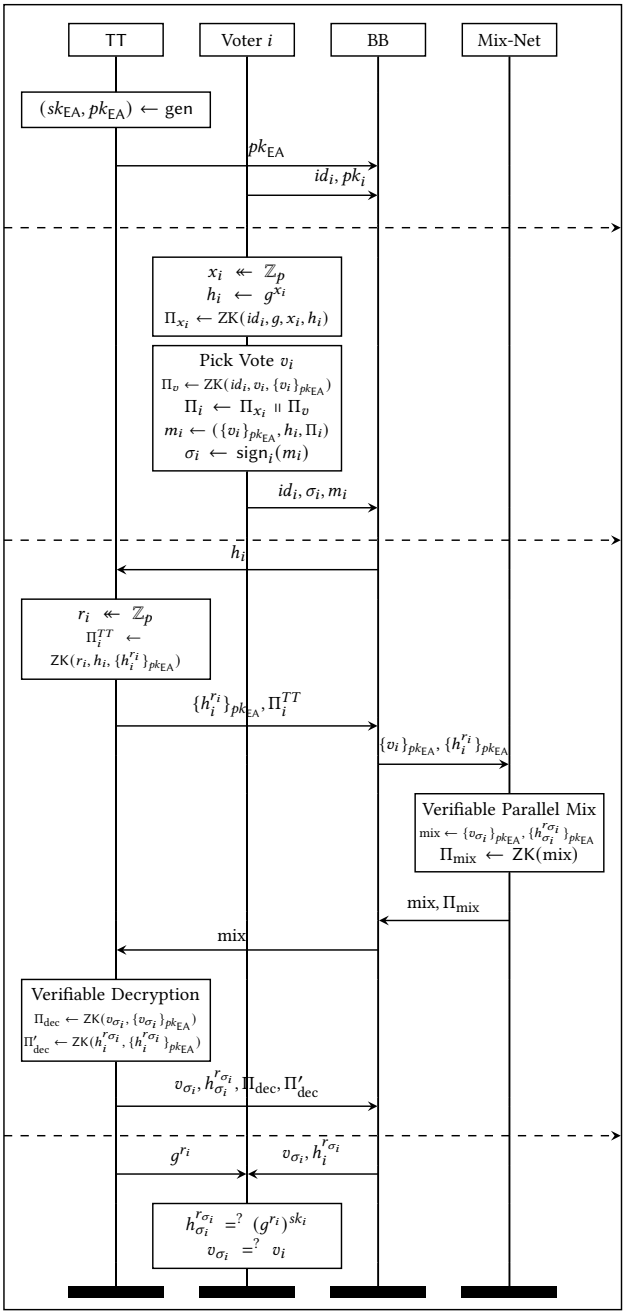

Figure 7: Flow diagram of *Hyperion* protocol. In this representation, we assume the EA **already defined and broadcast the cryptographic schemes to be used along with the corresponding parameters. The dashed lines separate the protocol into four phases: setup, voting, tallying, verification. We assume that the order of the group generated by g is a prime p, thus the exponents x_i and r_i are picked uniformly at random from $\mathbb{Z}_p$.**

Phase	**Selene**						**Hyperion**					
	$N = 50$	σ	$N = 100$	σ	$N = 150$	σ	$N = 50$	σ	$N = 100$	σ	$N = 150$	σ
Setup	7.7970s	1.3	15.171s	1.6	21.83s	4.0	0.0004s	7.7×10^{-5}	0.0005s	0.0002	0.0005s	8.3×10^{-5}
Voting	0.0089s	0.002	0.0089s	0.001	0.0084s	0.001	0.009s	0.001	0.0093s	0.0008	0.009s	0.0002
Tallying (Mix)	1.4822s	0.3	3.085s	0.5	4.693s	0.5	1.362s	0.2	2.9689s	0.4	4.2675s	0.1
Tallying (Decrypt)	0.6712s	0.2	1.2209s	0.3	1.5269s	0.3	0.5552s	0.03	1.1356s	0.2	1.4728	0.07
Coercion-Mitigation	0.003s	0.001	0.003s	0.0006	0.003s	0.0006	0.0008s	0.0001	0.0009s	0.0002	0.0008s	7×10^{-5}
Individual Views	-	-	-	-	-	-	0.4194s	0.01	0.8881s	0.08	1.3154s	0.1

Table 2: Execution times of each phase of the Hyperion and the Selene schemes in seconds.