Can we cast a ballot as intended and be receipt free?

Henri Devillez, Olivier Pereira, Thomas Peters, Quentin Yang

Can we cast a ballot as intended and be receipt free?

Henri Devillez*, Olivier Pereira*†, Thomas Peters* and Quentin Yang‡
*UCLouvain – ICTEAM – Crypto Group, B-1348 Louvain-la-Neuve – Belgium
† Microsoft Research, Redmond, WA, USA
‡ Université de Lorraine, Inria, CNRS, LORIA – France
Email: henri.devillez@uclouvain.be

***Abstract*—This paper explores the interaction between receipt-freeness and cast-as-intended verifiability, a property that has been overlooked until now or assumed to be granted through procedural means in the context of receipt-free voting protocols.**

We first demonstrate that it is impossible to obtain a receipt-free voting protocol with cast-as-intended verifiability if the voting process is non-interactive, unless a trusted authority is available. We also demonstrate that, if a trusted voter registration authority is available, then cast-as-intended verifiability and receipt-freeness can be obtained.

Furthermore, after extending standard receipt-freeness security definitions to an interactive voting (and corruption) setting, we demonstrate that the same security properties can be obtained using an interactive voting process.

Finally, we discuss the performance of our protocols based on a prototype implementation.

1. Introduction

From a mostly academic security property, verifiability has become a central requirement for voting protocols used in government elections: it is part of the Council of Europe recommendation on e-voting [1] and systems that offer at least some flavor of verifiability are now deployed in various countries including Switzerland [2], Estonia [3] and France [4].

Verifiability increases the challenges related to the privacy of the votes: the audit data that is published increases the attack surface that is available to break the privacy of the votes. Reconciling verifiability and privacy requirements has been a flourishing endeavor for more than 30 years. These properties are also supported by rigorous definitions and proof techniques, surveyed in [5], [6] for instance.

Reconciling verifiability and receipt freeness is even more challenging: verifiability must be achieved in a context in which a protocol must not only guarantee that voters can keep their vote private, but must go as far as preventing voters from being able to demonstrate to a buyer or coercer how they voted.

An increasing body of literature focuses on offering verifiable receipt-free voting. However, cast-as-intended verifiability, which guarantees that the ballot cast by a voter reflects her intent, is only guaranteed under the assumption that the voting client is trusted or that physical objects like paper are available [7], [8], [9], [10], [11], [12].

1.1. Our contributions

This paper explores the possibility to offer receipt-free voting protocols that offer cast-as-intended verifiability on top of the traditional properties of recorded-as-cast verifiability and universal verifiability.

1.1.1. Definitions and Impossibility Results. We start by exploring protocol requirements that are needed in order to obtain verifiable receipt-free protocols.

Our starting point is the game-based definition of receipt-freeness proposed by Chaidos et al. [10], which also serves as basis for the variations used in more recent works [11], [12]. These definitions model receipt freeness by offering the adversary the possibility to cast pairs of ballots on behalf of a single voter: one ballot represents the instruction of the coercer while the second ballot is the one cast by a voter who deviates from the coercer's instruction. A protocol is receipt free if the adversary cannot decide which of the two ballots is actually cast, except if this can be determined from the election tally. Interestingly, these definitions all assume a non-interactive voting process [5], [6], [13], also called single-pass voting, in which a ballot is cast by sending a single message.

As a first contribution, we demonstrate that, without the help of a trusted voter registration authority, it is impossible to achieve cast-as-intended verifiability and receipt-freeness with a non-interactive voting process.

As a second contribution, we extend the traditional definition of receipt-freeness in order to be able to support an interactive voting process, in which the adversary's instructions are provided as an interactive voting machine (ITM). The adversary then requires the voter to run that machine and simply forward messages back and forth between this ITM and the vote casting server. As before, the voter has also the possibility to follow a deviating strategy in order to cast a ballot that reflects his intent. As before, the adversary wins if he can decide which of the two voting processes is actually followed by the voter. These two contributions are detailed in Sections 2 and 3.

1.1.2. Verifiable Receipt-free Voting. In Section 5, we demonstrate that receipt-free voting with cast-as-intended verifiability can be achieved in the other natural settings. Our first two protocols rely on an interactive voting phase: the first protocol makes strong computational requirements on the voter but runs in a constant number of rounds, while the second protocol is more user friendly and requires a number of rounds that is linear in the security parameter. Our last protocol is non-interactive but relies on the availability of a trusted registration authority.

Our main goal here is to demonstrate the feasibility of reconciling cast-as-intended verifiability and receipt-freeness in relevant settings, and we do not make strong practicality claims regarding our solutions. Nevertheless, in Section 7, we propose a prototype implementation of our solutions, which demonstrates that our protocols can actually be executed within a very reasonable time frame: our most computationally demanding protocol, which is the non-interactive one, requires around 0.14 second of computation per candidate. Deciding which of our protocols is the most efficient overall will actually depend on the bandwidth and latency of the network connection: our interactive protocols may become slower under poor networking conditions.

1.2. Related Works

First impossibility results for voting were proposed by Chevallier-Mames et al. [14]. In particular, they demonstrate that universal verifiability and receipt-freeness cannot be jointly obtained unless private channels are available between the voters and the voting authorities. Our result extends that one by further demonstrating incompatibilities between cast-as-intended verifiability and non-interactive voting, in the absence of trusted authorities. More recently, Cortier and Lallemand showed that private elections cannot be obtained unless there is individual verifiability [15]. Receipt-freeness, on which we focus here, is itself implied by privacy (if my ballot is not private, it is itself a receipt), and our work sheds some light on the structure of the ballot submission process that is needed to achieve a strong form of privacy. But we see the main contribution of our work as the focus on cast-as-intended verifiability, while Cortier and Lallemand essentially express individual verifiability as a recorded-as-cast property, without separation between the voter and the voting device.

In terms of protocols, we believe that our protocols are the first to offer cast-as-intended verifiability and receipt-freeness without relying on further assumptions. The cast-as-intended verifiability can be obtained with overwhelming probability, contrary to previous protocols like Helios [16] that rely on a Benaloh challenge. To this purpose, we borrow and extend techniques from the recent Themis protocol by Bougon et al [17], where ballots contain two ciphertexts that contain secret shares of the vote, and one of the ciphertexts is opened. Our non-interactive protocol also takes advantage of traceable receipt-free encryption [12], a recently proposed public key encryption primitive

2. Definitions for Voting

2.1. Interactive algorithm

We use the following notation for interactive protocols between two parties. An interactive protocol $\mathcal{T}$ is a pair of two algorithms $(\mathcal{T}_1, \mathcal{T}_2)$. We write an execution of this protocol as $y_1, y_2 \leftarrow [\mathcal{T}_1(x_1) \leftrightarrow \mathcal{T}_2(x_2)]$ where x_i and y_i are respectively the input and output of $\mathcal{T}_i$.

Initially, the state τ_i of each party is set as their input. Then at each round of the protocol, the first party computes $m_1, \tau_1 \leftarrow \mathcal{T}_1(\tau_1)$ and sends m_1 to the other party. Then the second party appends this message to its state and computes $m_2, \tau_2 \leftarrow \mathcal{T}_2(\tau_2)$. It sends m_2 to the first party, which also updates its state. The two parties repeat this procedure until one message is equal to 0. In that case, no more messages are outputted and the parties return their output.

2.2. Voting system & Security

Let $\mathcal{C}$ be a set of voting options (*e.g.* candidates or ordered lists of candidates), $\mathcal{V}$ the set of voters, $\mathcal{R}$ the set of results (*e.g.* the name of the winner(s)) and $\rho : (\mathcal{V} \times \mathcal{C})^* \to \mathcal{R}$ be some counting function. The goal of a voting system is to evaluate ρ on the private choices of the voters, in a verifiable manner.

We define a *voting system* Π as a tuple of protocols $(\mathsf{Setup}, \mathsf{Valid}, \mathsf{Register}, \mathsf{Vote}, \mathsf{Tally}, \mathsf{VerifyVote}, \mathsf{Verify})$ which involve the following parties:

- The *election administrator* EA organizes the election and coordinates the phases of the protocol.
- The *voters* V cast an encrypted ballot to the casting server. If the voting device is corrupted, we treat as independent parties the voting device VD used to vote and the human being H interacting with the voting device. We also define an honest auditing device AD used to verify that their vote is correctly counted.
- The *registrar* R distributes private credentials to voters and registers the corresponding public credentials.
- The *casting server* CS receives the encrypted ballots submitted by the voters and update a public board PB with them.
- The *talliers* T compute the result of the election from the (encrypted) valid ballots.

We call *official parties* all the parties except the voters. We do not place any specific restriction on the capabilities of human voters. On the one hand, this makes our impossibility result stronger: our impossibility does not depend on limitations that we would place on what humans can do. Still, it is obvious that humans cannot perform sophisticated cryptographic operations in their head, and we will seek to make the task of humans as simple as possible. Some of our protocols are unrealistic from that point of view, but serve as a base for our next protocols that are much less demanding. In any case, we are not making any strong usability claim here.

Those protocols have the following signatures and functionalities. The public board PB is implicitly an input of all the protocols.

- Setup is an interactive protocol run by EA and the talliers: $\mathsf{pk}, \mathsf{sk} \leftarrow [\mathsf{Setup}_{\mathsf{EA}}(1^\lambda, \mathcal{C}, \mathcal{V}, \mathcal{R}, \rho) \leftrightarrow \mathsf{Setup}_{\mathsf{T}}(1^\lambda)]$. We assume that the public key pk, which might include the cryptographic group used and the public encryption keys of the election, is published on PB. T also receives some secret information sk used for the tallying. To simplify the notations, we write an execution of the setup protocol as $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$.
- Register is a protocol run by the registrar R and a voter id: $(\mathsf{upk}, \mathsf{usk})^2 \leftarrow [\mathsf{Register}_{\mathsf{R}}(1^\lambda, \mathsf{id}, \mathcal{V}) \leftrightarrow \mathsf{Register}_{\mathsf{V}}(1^\lambda)]$. Both parties receive as output the pair of credentials $(\mathsf{upk}, \mathsf{usk})$. usk is the private credentials and upk is the public credential, which is published on PB. We call the voting system registration-free if usk is set to $\perp$ and the registration system non-interactive if the protocol consists in only one interaction from R to the voter.
  To simplify the notations, we write an execution of the registration protocol as $\mathsf{upk}, \mathsf{usk} \leftarrow \mathsf{Register}(\mathsf{id})$.
- Vote is an interactive protocol run by a voter id and CS: $\mathsf{rcpt}, \mathsf{b} \leftarrow [\mathsf{Vote}_{\mathsf{V}}(1^\lambda, \mathsf{usk}, \mathsf{v}) \leftrightarrow \mathsf{Vote}_{\mathsf{CS}}(1^\lambda, \mathsf{id})]$. The voter has as input her choice $\mathsf{v} \in \mathcal{C}$ and her credentials usk.
  Without loss of generality, we assume that the output of CS is a value b that we call ballot and that is appended to the bulletin board at the end of the protocol. Also, we call the output of the voter *receipt*. Even if our model prevents CS to update PB during the execution of the protocol, we can imagine that it sends future updates to the voter during the protocol, which can appear in the receipt. If CS does not publish these updates, it will be caught by the voter in the VerifyVote procedure by comparing the receipt with the bulletin board.
  To simplify the notations, we write write an execution of the voting protocol as $\mathsf{rcpt}, \mathsf{b} \leftarrow \mathsf{Vote}(\mathsf{id}, \mathsf{v})$.
  Also, we model each round of the $\mathsf{Vote}_{\mathsf{V}}$ algorithm as a human-machine interaction to be able to model the corruption of the voting device. Hence, $\mathsf{Vote}_{\mathsf{V}}(1^\lambda, \mathsf{usk}, \mathsf{v})$ is an interactive protocol $[\mathsf{Vote}_{\mathsf{H}}(1^\lambda, \mathsf{v}, \mathsf{usk}) \leftrightarrow \mathsf{Vote}_{\mathsf{VD}}(1^\lambda)]$ between the actual voter and her voting device VD. The input of the voter is the voting intention v but also the credentials usk, which we consider the voter has acquired in some trusted way before. Then, the messages sent by $\mathsf{Vote}_{\mathsf{H}}$ are real human inputs (eg. a choice on a screen) and the messages sent by $\mathsf{Vote}_{\mathsf{VD}}$ are data that the human can access (eg. information on a screen). Finally, the output of $\mathsf{Vote}_{\mathsf{VD}}$ is the message sent to CS in one round of $\mathsf{Vote}_{\mathsf{V}}$ and the output of $\mathsf{Vote}_{\mathsf{H}}$ in each round of $\mathsf{Vote}_{\mathsf{V}}$ forms the receipt. Both the voter and the voting device keep their state across iterations of $\mathsf{Vote}_{\mathsf{V}}$.
  Finally, we say that he voting protocol is non-interactive if the only message computed by $\mathsf{Vote}_{\mathsf{CS}}$ is 0.
- Valid is an algorithm that takes as input a ballot and outputs 1 if the ballot is valid with respect to the public board. It outputs 0 otherwise. We write an execution of this algorithm as $\mathsf{Valid}(\mathsf{b})$.
- VerifyVote is an interactive protocol run by the voter and her auditing device: $[\mathsf{VerifyVote}_{\mathsf{H}}(1^\lambda, \mathsf{rcpt}) \leftrightarrow \mathsf{VerifyVote}_{\mathsf{AD}}(1^\lambda)]$. The voter takes as input the receipt of their last execution of the Vote protocol. It outputs 1 if the voter is convinced that after their last execution of Vote, PB contains their vote and outputs 0 otherwise. To simplify the notations, we write an execution of this protocol as $\mathsf{VerifyVote}(\mathsf{rcpt})$.
- Tally is a protocol during which T compute the result r of the election from the valid ballots in PB and sk. The algorithm also returns as a transcript Π proving the correctness of their computation. We write an execution of this protocol as $r, \Pi \leftarrow \mathsf{Tally}(\mathsf{PB}, \mathsf{sk})$.
- Verify is an interactive protocol run by any voter and her auditing device: $[\mathsf{Verify}_{\mathsf{H}}(1^\lambda, r, \Pi) \leftrightarrow \mathsf{Verify}_{\mathsf{AD}}(1^\lambda)]$. It takes as input the result r and the transcript Π of the talliers and outputs 1 if the data are consistent, 0 otherwise. To simplify the notations, we write an execution of the protocol as $\mathsf{Verify}(r, \Pi)$.

We require the voting system to have the usual correctness property. Especially, we want Tally to always return the same value as ρ and VerifyVote to always return 1 if all the parties are honest.

2.3. Receipt-freeness

Receipt-freeness is a property guaranteeing that a voter cannot prove to another party how she voted, even if she is required to follow instructions given by the coercing party. It is worth mentioning that this property does not cover forced abstention, that is a scenario in which the adversarial party coerces the voter into not voting.

To formalize this notion, we start from the usual definition [10], [12] which is reminiscent of the BPRIV game-based definition [5].

This definition is suitable if the vote protocol is non-interactive and the adversary instruction to the voter is then only to send a message to CS.

We thus propose an alternative definition that supports interactive voting processes. The main difference is that the adversary's instruction now takes the form of an interactive Turing machine that can give instructions to the voter at each step of the voting process execution. The Turing machine, which is possibly obfuscated, is expected to produce the messages sent to the server instead of the voter and to produce an arbitrary string (the receipt) at the end of the voting protocol, as a replacement of the $\mathsf{Vote}_{\mathsf{V}}$ algorithm.

Definition 1 (Receipt-freeness)**.** *A voting system* Π *has receipt-freeness if there exists two PPT algorithms* SimSetup*,* SimProof *and an interactive PPT deceiving algorithm* $\mathcal{D}$ *such that the two following conditions are met:*

- *no efficient adversary can distinguish between games* $\mathsf{Exp}^{\mathrm{rf},0}_{\mathcal{A},\Pi,\mathcal{D}}(\lambda)$ *and* $\mathsf{Exp}^{\mathrm{rf},1}_{\mathcal{A},\Pi,\mathcal{D}}(\lambda)$ *defined in Figure 1. That is, for any PPT algorithm* $\mathcal{A}$*, the following advantage* $\mathsf{Adv}^{\mathrm{rf}}_{\mathcal{A},\Pi,\mathcal{D}}(\lambda)$ *is negligible in* λ*:*

$$\left|\Pr\left(\mathsf{Exp}^{\mathrm{rf},0}_{\mathcal{A},\Pi,\mathcal{D}}(\lambda) = 1\right) - \Pr\left(\mathsf{Exp}^{\mathrm{rf},1}_{\mathcal{A},\Pi,\mathcal{D}}(\lambda) = 1\right)\right|;$$

$\mathsf{Exp}^{\mathrm{rf},\beta}_{\mathcal{A},\Pi,\mathcal{D}}(\lambda)$

$\mathsf{pk} \leftarrow \mathcal{O}\mathsf{init}^{\mathsf{rf}}(1^\lambda)$;
$\mathsf{st} \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{register}^{\mathsf{rf}},\mathcal{O}\mathsf{corrupt}^{\mathsf{rf}},\mathcal{O}\mathsf{cast}^{\mathsf{rf}},\mathcal{O}\mathsf{board}^{\mathsf{rf}},\mathcal{O}\mathsf{voteLR}^{\mathsf{rf}},\mathcal{O}\mathsf{receiptLR}^{\mathsf{rf}}}(\mathsf{pk})$;
$(\mathsf{r},\Pi) \leftarrow \mathcal{O}\mathsf{tally}^{\mathsf{rf}}(\mathsf{sk})$;
return $\mathcal{A}(\mathsf{st},\mathsf{r},\Pi)$;

$\mathcal{O}\mathsf{init}^{\mathsf{rf}}(1^\lambda)$

if $\beta = 0$
then $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
else $(\mathsf{pk},\mathsf{sk},\tau) \leftarrow \mathsf{SimSetup}(1^\lambda)$
$\mathsf{PB}_0 \leftarrow \varnothing$; $\mathsf{PB}_1 \leftarrow \varnothing$
return pk

$\mathcal{O}\mathsf{register}^{\mathsf{rf}}(\mathsf{id})$

if id has not been queried yet:
then $\mathsf{upk},\mathsf{usk} \leftarrow \mathsf{Register}(\mathsf{id})$
$\mathcal{U} \leftarrow \mathcal{U} \cup \{\mathsf{id}\}$
return upk

$\mathcal{O}\mathsf{corrupt}^{\mathsf{rf}}(\mathsf{id})$

if $\mathsf{id} \in \mathcal{U}$:
then $\mathcal{CU} \leftarrow \mathcal{CU} \cup \{\mathsf{id}\}$
and **return** usk

$\mathcal{O}\mathsf{board}^{\mathsf{rf}}()$

return PB_β

$\mathcal{O}\mathsf{receiptLR}^{\mathsf{rf}}(\mathsf{id},\mathcal{I},\mathsf{v})$

if $\mathsf{v} \notin \mathcal{C}$ **then return** 0
$\mathsf{rcpt}_0,\mathsf{b}_0 \leftarrow [\mathcal{D}(\mathsf{id},\mathcal{I},\mathsf{v}) \leftrightarrow \mathsf{Vote}_{\mathsf{CS}}(\mathsf{id})]$
$\mathsf{rcpt}_1,\mathsf{b}_1 \leftarrow [\mathcal{I}() \leftrightarrow \mathsf{Vote}_{\mathsf{CS}}(\mathsf{id})]$
if $\mathsf{Valid}(\mathsf{PB}_1,\mathsf{b}_1) = 0$ **then return** 0
$\mathsf{PB}_0 \leftarrow \mathsf{PB}_0||\mathsf{b}_0$; $\mathsf{PB}_1 \leftarrow \mathsf{PB}_1||\mathsf{b}_1$
return rcpt_β

$\mathcal{O}\mathsf{voteLR}^{\mathsf{rf}}(\mathsf{id},\mathsf{v}_0,\mathsf{v}_1)$

if $\mathsf{v}_0 \notin \mathcal{C}$ **or** $\mathsf{v}_1 \notin \mathcal{C}$ **then return** 0
$\mathsf{rcpt}_0,\mathsf{b}_0 \leftarrow \mathsf{Vote}(\mathsf{id},\mathsf{v}_0)$
$\mathsf{rcpt}_1,\mathsf{b}_1 \leftarrow \mathsf{Vote}(\mathsf{id},\mathsf{v}_1)$
$\mathsf{PB}_0 \leftarrow \mathsf{PB}_0||\mathsf{b}_0$; $\mathsf{PB}_1 \leftarrow \mathsf{PB}_1||\mathsf{b}_1$
return rcpt_β

$\mathcal{O}\mathsf{cast}^{\mathsf{rf}}(\mathsf{id},\mathcal{I})$

$\mathsf{rcpt},\mathsf{b} \leftarrow [\mathcal{I}() \leftrightarrow \mathsf{Vote}_{\mathsf{CS}}(\mathsf{id})]$
if $\mathsf{Valid}(\mathsf{PB}_\beta,\mathsf{b}) = 0$ **then return** 0
$\mathsf{PB}_0 \leftarrow \mathsf{PB}_0||\mathsf{b}$; $\mathsf{PB}_1 \leftarrow \mathsf{PB}_1||\mathsf{b}$
return rcpt

$\mathcal{O}\mathsf{tally}^{\mathsf{rf}}()$

$(\mathsf{r},\Pi) \leftarrow \mathsf{Tally}(\mathsf{PB}_0,\mathsf{sk})$
if $\beta = 1$ **then** $\Pi \leftarrow \mathsf{simproof}(\mathsf{PB}_1,\mathsf{r},\tau)$
return (r,Π)

Figure 1: The receipt-free experiment $\mathsf{Exp}^{\mathrm{rf},\beta}_{\mathcal{A},\Pi,\mathcal{D}}(\lambda)$. In the $\mathcal{O}\mathsf{receiptLR}^{\mathsf{rf}}$ oracle, the deceiving algorithm takes as input the instructions of the adversary, the true voting intention of the voter and potential information attached to the voter, such as her id and her credentials if there is a registration. It interacts with CS and output a rcpt that can be different from the messages exchanged in the interactive protocol. Also, the bulletin boards are not updated during any execution of the voting protocols. Instead, the ballots are appended to each of the bulletin board if they are both valid.

- *no efficient adversary can win the vote deviation correctness experiment* $\mathsf{Exp}^{\mathrm{dev}}_{\mathcal{A},\Pi,\mathcal{D}}(\lambda)$ *defined in Figure 2. That is, for any PPT algorithm* $\mathcal{A}$*, the following advantage is negligible in* λ*:*

$$\mathsf{Adv}^{\mathrm{dev}}_{\mathcal{A},\Pi,\mathcal{D}}(\lambda) = \Pr\left(\mathsf{Exp}^{\mathrm{dev}}_{\mathcal{A},\Pi,\mathcal{D}}(\lambda) = 1\right).$$

Just as in the BeleniosRF definition [10], we define receipt-freeness using an experiment $\mathsf{Exp}^{\beta}_{\mathcal{A},\Pi,\mathcal{D}}(\lambda)$, where the adversary must guess if the bit β is equal to 0 or 1. The intuition is that when $\beta = 0$, the protocol is ran honestly and the voters deviate from the instructions $\mathcal{I}$ given by the adversary. $\mathcal{I}$ is the description of a Turing machine that the voter is asked to execute and which gives at each round of the voting protocol the message that the voter should send to CS. In the deviation process, the voters use the algorithm $\mathcal{D}$ instead of $\mathsf{Vote}_{\mathsf{V}}$, which allows the voter to not only produce a ballot which looks like the one cast when following the adversary's instruction, but also to generate a deceiving receipt. Most importantly, the voter has the possibility to run $\mathcal{I}$ as a subroutine and to rewind it. Indeed, if $\mathcal{I}$ can send messages to CS directly, there is no hope of security. This

$\mathsf{Exp}^{\mathrm{dev}}_{\mathcal{A},\Pi,\mathcal{D}}(\lambda)$

$\mathsf{pk} \leftarrow \mathcal{O}\mathsf{init}^{\mathsf{dev}}(1^\lambda)$; $\mathcal{A}^{\mathcal{O}\mathsf{register}^{\mathsf{dev}},\mathcal{O}\mathsf{voteLR}^{\mathsf{dev}}}(\mathsf{pk})$;
$(\mathsf{r}_0,\Pi_0) \leftarrow \mathsf{Tally}(\mathsf{PB}_0,\mathsf{sk})$;
$(\mathsf{r}_1,\Pi_1) \leftarrow \mathsf{Tally}(\mathsf{PB}_1,\mathsf{sk})$;
if $\mathsf{r}_0 \neq \mathsf{r}_1$ **return** 1 **else return** 0;

$\mathcal{O}\mathsf{init}^{\mathsf{dev}}(1^\lambda)$

$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathsf{PB}_0 \leftarrow \varnothing$; $\mathsf{PB}_1 \leftarrow \varnothing$
return pk

$\mathcal{O}\mathsf{register}^{\mathsf{dev}}(\mathsf{id})$

if id has not been queried yet:
then $\mathsf{upk},\mathsf{usk} \leftarrow \mathsf{Register}(\mathsf{id})$
$\mathcal{U} \leftarrow \mathcal{U} \cup \{\mathsf{id}\}$
return upk, usk

$\mathcal{O}\mathsf{voteLR}^{\mathsf{dev}}(\mathsf{id},\mathcal{I},\mathsf{v})$

if $\mathsf{v} \notin \mathcal{C}$ **or** $(\mathsf{id},\star,\star) \notin \mathcal{U}$ **then return** 0
$\mathsf{rcpt}_0,\mathsf{b}_0 \leftarrow \mathsf{Vote}(\mathsf{id},\mathsf{v})$
$\mathsf{rcpt}_1,\mathsf{b}_1 \leftarrow [\mathcal{D}(\mathsf{id},\mathcal{I},\mathsf{v}) \leftrightarrow \mathsf{Vote}_{\mathsf{CS}}(\mathsf{id})]$
$\mathsf{rcpt}',\mathsf{b}' \leftarrow [\mathcal{I}() \leftrightarrow \mathsf{Vote}_{\mathsf{CS}}(\mathsf{id})]$
if $\mathsf{Valid}(\mathsf{PB}_1,b') = 0$ **return** 0
$\mathsf{PB}_0 \leftarrow \mathsf{PB}_0||\mathsf{b}_0$; $\mathsf{PB}_1 \leftarrow \mathsf{PB}_1||\mathsf{b}_1$
return $\mathsf{rcpt}_0,\mathsf{rcpt}_1,\mathsf{b}_0,\mathsf{b}_1$

Figure 2: Vote deviation correctness experiment.

is thus similar to organizational measures to ensure that the voter can interact with the instructions in this way.

When $\beta = 1$, however, the voters follow the instructions given by the adversary. Hence, if the adversary is unable to guess β with a non-negligible advantage, it means that the voter cannot convince the adversary that they followed the instruction instead of applying the deceiving strategy.

A key element to understand this definition is that, when $\beta = 1$, the tally is simulated so that the result of the election is the same whatever the value of β. Indeed, if PB_β is tallied, then an adversary can immediately win the game with a $\mathcal{O}\mathsf{voteLR}^{\mathsf{rf}}(\mathsf{id},0,1)$ request. Consequently, during the experiment, the adversary cannot use any information from the tally to infer whether the voters obeyed or not. In practice, if all the voters are instructed to vote for A, the adversary can guess that a majority disobeyed if B wins. Therefore, our definition actually states that the adversary does not have any other information than the result of the tally.

Also, the second part of the definition in Figure 2 alleviates a shortcoming of the [10] definition, which is the potential inability for a voter to cast her true voting intention regardless of the adversary's instruction. Indeed, this experiment ensures that the ballots produced by the deviation algorithm for a vote for v are counted in the same way as ballots produced by an honest $\mathsf{Vote}(\mathsf{v})$ algorithm.

This additional requirement is crucial. For example, a $\mathsf{Vote}_{\mathsf{V}}(\mathsf{id},\mathsf{v})$ algorithm which would send a random tag t and v to CS and a $\mathsf{Vote}_{\mathsf{CS}}(\mathsf{id})$ algorithm which would append the hash of the tag and an NM-CPA encryption of v to the bulletin board can be similarly proven to be secure under this definition. However, if the Valid algorithm returns false if the first value bit of the vote is different from the first bit of the tag, then the candidates for which the voter can vote is limited and she does not fully escape receipt-freeness. This example is admittedly convoluted and we are not aware of any reasonable voting scheme with such shortcoming, but this is still not captured by the definition.

Recall, however, that we do not consider forced-abstention attacks, so that an adversary can still instruct a voter not to vote, even if the voting protocol is receipt

free. A more powerful attacker could even ask the voter not to vote and to give away any voting material, in order to vote by itself. To protect against such scenarios, we need a stronger notion which is coercion-resistance [9]. However, it involves more engaged deceiving strategy and setting and makes stronger assumptions about the voting channels, hence we do not consider such scenarios in this work.

We also stress that this definition does not always imply privacy. For example, a $\mathsf{Vote}_{\mathsf{V}}(\mathsf{id}, \mathsf{v})$ algorithm which would send v to CS and a $\mathsf{Vote}_{\mathsf{CS}}(\mathsf{id})$ algorithm which would append an NM-CPA encryption of v to the bulletin board can be proven to satisfy this definition in an Helios-like tallying procedure. However, this totally breaks the voter's privacy for a semi-honest casting server, which means that this definition only implies privacy for external parties.

2.4. Verifiability

Intuitively, a protocol is verifiable if, by performing a determined set of verifications, the participants have the guarantee that the result is correct.

For a voting protocol, the verifiability is split into four parts: the *eligibility* verification checks that only eligible voters can vote, and do so at most once; the *cast-as-intended* verification allows the voters to check that their encrypted ballot actually contains a vote for the chosen voting option, even if the voting device is compromised; the *recorded-as-cast* verification assures that their ballot has been added to the list of ballots to be tallied; and the *tallied-as-recorded* verification guarantees that the result of the election is computed honestly from the said list. This decomposition allows a modular approach, and the literature offers many generic ways to address each property separately.

Namely, the eligibility is obtained using authentication protocols and the tallied-as-recorded property can be checked using Zero Knowledge Proofs that the result is correctly computed from the bulletin board.

To obtain the cast-as-intended verification one popular option is to use a so-called Benaloh's challenge [18], which uses an "audit-or-cast" strategy. However, the audited ballots can be used as receipts and this notion has often been overlooked in the several protocols trying to achieve receipt-freeness. Then, the recorded-as-cast verification is usually enforced thanks to the public bulletin board. Usually, this consists in the voter checking that their ballot is on the bulletin board, but more elaborate protocols achieving receipt-freeness might need more elaborate verification procedures. Hence, we focus in this work on these two notions of verifiability.

2.5. Verifiability against a corrupted voting device

We start with the *cast-as-intended* property, which can be seen as verifiability against a corrupted voting device. We are aware of only one formalization of this property as a game-based definition in the computational model of cryptography, which is proposed in [19]. However, their definition does not model voting credentials and does not extend easily to a setting in which more parties are corrupted.

We thus take inspiration of the verifiability definition of [20], which can be easily extended. In our definition, we assume first that all the official parties are honest and focus on the voting device. We will remove this assumption later.

More formally, we consider for the recorded-as-cast a variant of the verifiability definition of [20] to take into account the corruption of a voting device. We focus on result functions ρ that are not constant[1] and that admit partial counting. That is, for any integer τ and any family V of τ voters, there exists two family of τ votes S_1 and S_2 such that $\rho(\{V_i, S_{1,i}\}) \neq \rho(\{V_i, S_{2,i}\})$. Moreover, there exists a commutative operation $\star_{\mathcal{R}} : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$ such that $\rho(\{V_{1,i}, S_{1,i}\} \cup \{V_{2,i}, S_{1,i}\}) = \rho(\{V_{1,i}, S_{1,i}\}) \star_{\mathcal{R}} \rho(\{V_{2,i}, S_{2,i}\})$ for any families of voters V_1, V_2 and any family of votes S_1, S_2.

Intuitively, this definition guarantees that the result output by Tally counts the actual votes cast by honest voters, even if the adversary controls a subset of eligible voters, the voting device of all the other voters and knows which voters do not check their ballot.

Definition 2 (Verifiability against a corrupted voting device)**.** *A voting system Π is verifiable against a corrupted voting device if for all PPT $\mathcal{A}$, the following is negligible in λ:*

$$\mathsf{Adv}^{\mathsf{vd}-\mathrm{verif}}_{\mathcal{A},\mathcal{V}}(\lambda) = \Pr\left(\mathsf{Exp}^{\mathsf{vd}-\mathrm{verif}}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right)$$

where $\mathsf{Exp}^{\mathsf{vd}-\mathrm{verif}}_{\mathcal{A},\Pi}(\lambda)$ is defined in Figure 3.

We stress that in this scenario, the corrupted voting device does not leak the voter's secret credentials to the adversary. We assume that the voter receives these credentials from another trusted device (or from a postal channel). However, some protocols might require the voter to give their credentials to the voting device. We thus propose a variant $\mathsf{Exp}^{\mathsf{cred},\mathsf{vd}-\mathrm{verif}}_{\mathcal{A},\mathcal{V}}(\lambda)$ in which the $\mathcal{O}\mathsf{register}^{\mathsf{vd}}$ oracle also returns usk to the adversary.

2.6. Verifiability against corrupted voting device and all the official parties

We extend the previous definition to more corrupted parties. Especially, it is desirable to have verifiability even if all the official parties are corrupted. In such a low-trust setting, we only assume that the voter has a trusted access to the bulletin board, via the auditing device for example.

Definition 3 (Verifiability against all parties)**.** *A voting system Π is verifiable against all parties if for all PPT $\mathcal{A}$, the following is negligible in λ:*

$$\mathsf{Adv}^{\mathrm{all}-\mathrm{verif}}_{\mathcal{A},\mathcal{V}}(\lambda) = \Pr\left(\mathsf{Exp}^{\mathrm{all}-\mathrm{verif}}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right) \leq \mu(\lambda)$$

where $\mathsf{Exp}^{\mathrm{all}-\mathrm{verif}}_{\mathcal{A},\Pi}(\lambda)$ is defined in Figure 4.

1. A result function which does not depend on the votes would be questionable anyway.

$\mathsf{Exp}^{\mathsf{vd-verif}}_{\mathcal{A},\Pi}(\lambda)$

$\mathsf{pk} \leftarrow \mathcal{O}\mathsf{init}^{\mathsf{vd}}(1^\lambda); \mathcal{A}^{\mathcal{O}\mathsf{register}^{\mathsf{vd}},\mathcal{O}\mathsf{corrupt}^{\mathsf{vd}},\mathcal{O}\mathsf{vote}^{\mathsf{vd}},\mathcal{O}\mathsf{cast}^{\mathsf{vd}},\mathcal{O}\mathsf{check}^{\mathsf{vd}}}(\mathsf{pk})$;
$(\mathsf{r}, \Pi) \leftarrow \mathsf{Tally}(\mathsf{PB}, \mathsf{sk})$
if $\mathsf{Verify}(\mathsf{r}, \mathsf{PB}, \Pi) == 0$ **then return** 0
for each $\mathsf{id} \in \mathsf{Checked}$:
  if $\mathsf{VerifyVote}(\mathsf{PB}, \mathsf{rcpt}) == 0$ where $(\mathsf{id}, \star, \mathsf{rcpt}) \in \mathsf{Hvote}$
  then return 0
if $\exists \mathsf{v}_1^A, \ldots, \mathsf{v}_{n_A}^A \in \mathcal{C}$ with $0 \leq n_A \leq |\mathsf{Hvote} \setminus \mathsf{Checked}|$
if $\exists \mathsf{v}_1^B, \ldots, \mathsf{v}_{n_B}^B \in \mathcal{C}$ with $0 \leq n_B \leq |\mathcal{CU}|$
such that $r = \rho(\{\mathsf{id}_i^E, \mathsf{v}_i^E\}_{i=1}^{n_E}) \star_{\mathcal{R}} \rho(\{\mathsf{id}_i^A, \mathsf{v}_i^A\}_{i=1}^{n_A}) \star_{\mathcal{R}} \rho(\{\mathsf{id}_i^B, \mathsf{v}_i^B\}_{i=1}^{n_B})$
where $(\mathsf{id}_i^E, \mathsf{v}_i^E, \star) \in \mathsf{Hvote}$ and $\mathsf{Checked} = \{\mathsf{id}_1^E, \ldots, \mathsf{id}_{n_E}^E\}$
then return 0 **else return** 1

$\mathcal{O}\mathsf{init}^{\mathsf{vd}}(1^\lambda)$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathsf{PB} \leftarrow \varnothing; \mathsf{Hvote} \leftarrow \varnothing;$
return pk

$\mathcal{O}\mathsf{register}^{\mathsf{vd}}(\mathsf{id})$

$\mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}} \leftarrow \mathsf{Register}(\mathsf{id}, 1^\lambda)$
$\mathcal{U} \leftarrow \mathcal{U} \cup \{(\mathsf{id}, \mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}})\}$
return $\mathsf{upk}_{\mathsf{id}}$

$\mathcal{O}\mathsf{corrupt}^{\mathsf{vd}}(\mathsf{id})$

if $(\mathsf{id}, \star, \star) \in \mathcal{U}$
then $\mathcal{CU} \leftarrow \mathcal{CU} \cup \{\mathsf{id}, \mathsf{upk}_{\mathsf{id}}\}$
remove any $(\mathsf{id}, \star, \star)$ from Hvote
return $(\mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}})$

$\mathcal{O}\mathsf{vote}^{\mathsf{vd}}(\mathsf{id}, \mathsf{v})$

if $(\mathsf{id}, \star, \star) \notin \mathcal{U}$ **or** $(\mathsf{id}, \star) \in \mathcal{CU}$
or $\mathsf{v} \notin \mathcal{C}$: **return** 0
$\mathsf{rcpt}, \mathsf{b} \leftarrow [(\mathsf{Vote}_{\mathsf{H}}(\mathsf{v}) \leftrightarrow \mathcal{A}) \leftrightarrow \mathsf{Vote}_{\mathsf{CS}}(\mathsf{id})]$
$\mathsf{Hvote} \leftarrow (\mathsf{id}, \mathsf{v}, \mathsf{rcpt});$
$\mathsf{PB} \leftarrow \mathsf{PB}||\mathsf{b};$
return b

$\mathcal{O}\mathsf{cast}^{\mathsf{vd}}(\mathsf{id})$

if $(\mathsf{id}, \star) \notin \mathcal{CU}$: **return** 0
$\mathsf{rcpt}, \mathsf{b} \leftarrow [\mathcal{A} \leftrightarrow \mathsf{Vote}_{\mathsf{CS}}(\mathsf{id})]$
$\mathsf{PB} \leftarrow \mathsf{PB}||\mathsf{b};$

$\mathcal{O}\mathsf{check}^{\mathsf{vd}}(\mathsf{id})$

if $(\mathsf{id}, \star, \star) \notin \mathcal{U}$: **return** 0
else $\mathsf{Checked} \leftarrow \mathsf{Checked} \cup \{\mathsf{id}\};$

Figure 3: The verifiability against a corrupted voting device experiment and its oracles. In the $\mathcal{O}\mathsf{vote}^{\mathsf{vd}}$ oracle, the adversary can freely chooses the messages sent to the voter in the interactive voting protocol.

$\mathsf{Exp}^{\mathsf{all-verif}}_{\mathcal{A},\Pi}(\lambda)$

$\mathsf{pk} \leftarrow \mathcal{O}\mathsf{init}^{\mathsf{all}}(1^\lambda); \mathsf{PB}, \Pi, r \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{register}^{\mathsf{all}},\mathcal{O}\mathsf{corrupt}^{\mathsf{all}},\mathcal{O}\mathsf{vote}^{\mathsf{all}},\mathcal{O}\mathsf{check}^{\mathsf{all}}}(\mathsf{pk})$;
if $\mathsf{Verify}(r, \mathsf{PB}, \Pi) == 0$ or $\rho = 0$ **then return** 0
for each $\mathsf{id} \in \mathsf{Checked}$:
  if $\mathsf{VerifyVote}(\mathsf{PB}, \mathsf{rcpt}) == 0$ where $(\mathsf{id}, \star, \mathsf{rcpt}) \in \mathsf{Hvote}$
  then return 0
if $\exists \mathsf{v}_1^A, \ldots, \mathsf{v}_{n_A}^A \in \mathcal{C}$ with $0 \leq n_A \leq |\mathsf{Hvote} \setminus \mathsf{Checked}|$
if $\exists \mathsf{v}_1^B, \ldots, \mathsf{v}_{n_B}^B \in \mathcal{C}$ with $0 \leq n_B \leq |\mathcal{CU}|$
such that $r = \rho(\{\mathsf{id}_i^E, \mathsf{v}_i^E\}_{i=1}^{n_E}) \star_{\mathcal{R}} \rho(\{\mathsf{id}_i^A, \mathsf{v}_i^A\}_{i=1}^{n_A}) \star_{\mathcal{R}} \rho(\{\mathsf{id}_i^B, \mathsf{v}_i^B\}_{i=1}^{n_B})$
where $(\mathsf{id}_i^E, \mathsf{v}_i^E, \star) \in \mathsf{Hvote}$ and $\mathsf{Checked} = \{\mathsf{id}_1^E, \ldots, \mathsf{id}_{n_E}^E\}$
then return 0 **else return** 1

$\mathcal{O}\mathsf{init}^{\mathsf{all}}(1^\lambda)$

$\mathsf{pk} \leftarrow \mathcal{A}()$
$\mathsf{PB} \leftarrow \varnothing; \mathsf{Hvote} \leftarrow \varnothing;$
return pk

$\mathcal{O}\mathsf{register}^{\mathsf{all}}(\mathsf{id})$

$\mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}} \leftarrow \mathcal{A}(\mathsf{id})$
$\mathcal{U} \leftarrow \mathcal{U} \cup \{(\mathsf{id}, \mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}})\}$
return $\mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}}$

$\mathcal{O}\mathsf{corrupt}^{\mathsf{all}}(\mathsf{id})$

if $(\mathsf{id}, \star, \star) \in \mathcal{U}$
then $\mathcal{CU} \leftarrow \mathcal{CU} \cup \{\mathsf{id}, \mathsf{upk}_{\mathsf{id}}\}$
remove any $(\mathsf{id}, \star, \star)$ from Hvote
return $(\mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}})$

$\mathcal{O}\mathsf{vote}^{\mathsf{all}}(\mathsf{id}, \mathsf{v})$

if $(\mathsf{id}, \star, \star) \notin \mathcal{U}$ **or** $(\mathsf{id}, \star) \in \mathcal{CU}$
or $\mathsf{v} \notin \mathcal{C}$: **return** 0
$\mathsf{rcpt}, \mathsf{b} \leftarrow [(\mathsf{Vote}_{\mathsf{H}}(\mathsf{id}, \mathsf{v}) \leftrightarrow \mathcal{A}) \leftrightarrow \mathcal{A}]$
$\mathsf{Hvote} \leftarrow (\mathsf{id}, \mathsf{v}, \mathsf{rcpt});$
return b

$\mathcal{O}\mathsf{check}^{\mathsf{all}}(\mathsf{id})$

if $(\mathsf{id}, \star, \star) \notin \mathcal{U}$: **return** 0
else $\mathsf{Checked} \leftarrow \mathsf{Checked} \cup \{\mathsf{id}\};$

Figure 4: The verifiability experiment against all parties.

For the sake of readability, we will omit the subscript of the experiment and advantage notations when there is no ambiguity.

2.7. Vote deviation verifiability

These definitions do not give any verifiability guarantee in case the voter is following the deceiving strategy instead of the normal voting procedure. In practice, we would want to preserve verifiability when deviating from a receipt-free adversary. As we did with the Vote algorithm, we can split $\mathcal{D}$ as an interactive protocol $[\mathcal{D}_{\mathsf{H}} \leftrightarrow \mathcal{D}_{\mathsf{VD}}]$ between the human voter and the voting device and adapt the verifiability definitions accordingly. More precisely, we can add an $\mathcal{O}\mathsf{dev}^{\mathsf{vd}}(\mathsf{id}, \mathcal{I}, \mathsf{v})$ oracle to the original definition that executes $\mathcal{D}_{\mathsf{H}}(\mathsf{id}, \mathcal{I}, \mathsf{v})$ instead of $\mathsf{Vote}_{\mathsf{H}}(\mathsf{id}, \mathsf{v})$.

Given a verifiable protocol, one sufficient condition to obtain verifiability when following the deceiving strategy would be to have a $\mathcal{D}_{\mathsf{H}}$ algorithm indistinguishable from the $\mathsf{Vote}_{\mathsf{H}}$ algorithm from the view of the voting device, for any $\mathcal{I}$:

$$\mathcal{D}_{\mathsf{H}}(\mathsf{id}, \mathcal{I}, \mathsf{v}) \approx \mathsf{Vote}_{\mathsf{H}}(\mathsf{id}, \mathsf{v}) \quad \forall \mathsf{id}, \mathcal{I}, \mathsf{v}$$

Indeed, we can easily reduce to the original verifiability definition by a sequence of transitions that replaces executions of $\mathcal{D}_{\mathsf{H}}$ by $\mathsf{Vote}_{\mathsf{H}}$.

3. Impossibility results

Our definitions make an important departure from most of the traditional ones, that focus on a non-interactive voting protocol, also often called single-pass voting [5], [6], [13]. Here, we consider an interactive voting process, during which the voter and the casting server may have multiple rounds of interaction.

Since many voting protocols (e.g., Helios, Belenios, etc.) support a single pass voting process, we may question the motivation for this additional complexity. The reason appears in the following impossibility result, which demonstrates that there is no registration-free non-interactive voting protocol that has receipt-freeness and verifiability against a corrupted voting device. Helios [16], for instance, is registration-free, has a non-interactive voting protocol and is verifiable against a corrupted voting device, but is not receipt-free. BeleniosRF [10] brings receipt-freeness but requires a trusted voter registration process and does not support verifiability against a corrupted voting device. A consequence of our impossibility result is that it is not possible to make Helios receipt-free without adding either a trusted registration process or making the voting protocol interactive. In the next sections, we will show that both these options are possible.

The impossibility result proceeds by showing that any successful deviating strategy that a voter could use to protect herself from coercion could actually also be used by a corrupted voting device in order to break the verifiability property. Intuively, the corrupted voting device can submit

a ballot while using the deviating strategy and treating the voter as a coercer.

Theorem 1. *There is no registration-free voting system with a non-interactive voting protocol that has receipt-freeness and verifiability against a corrupted voting device.*

More precisely, for any registration-free voting system Π *with a non-interactive voting protocol and any deviating strategy* $\mathcal{D}$ *there is an adversary* $\mathcal{A}_V$ *for the verifiability experiment* $\mathsf{Exp}^{\mathsf{vd-verif}}_{\mathcal{A}_V,\Pi}(\lambda)$ *of Definition 2, an adversary* $\mathcal{A}_{RF}$ *for the receipt-freeness experiment* $\mathsf{Exp}^{\mathrm{rf}}_{\mathcal{A}_{RF},\Pi}(\lambda)$ *and an adversary* $\mathcal{A}_{\mathcal{D}}$ *for the vote deviation correctness experiment* $\mathsf{Exp}^{\mathrm{dev}}_{\mathcal{A}_{\mathcal{D}},\Pi}(\lambda)$*, both of Definition 1, such that:*

$$\mathsf{Adv}^{\mathsf{vd-verif}}_{\mathcal{A}_V,\Pi}(\lambda) + \mathsf{Adv}^{\mathrm{rf}}_{\mathcal{A}_{RF},\Pi}(\lambda) + \mathsf{Adv}^{\mathrm{dev}}_{\mathcal{A}_{\mathcal{D}},\Pi}(\lambda) \geq 1$$

Proof. Let us consider a voting system Π with n voters $\mathcal{V}$ and a deviating strategy $\mathcal{D}$. We will build three adversaries $\mathcal{A}_V$, $\mathcal{A}_{RF}$ and $\mathcal{A}_{\mathcal{D}}$ such that at least one of them wins. From the assumption on the result function, there are two lists of votes S_0 and S_1 such that $\rho(\{\mathcal{V}_i, \mathsf{v}_{0,i}\}) \neq \rho(\{\mathcal{V}_i, \mathsf{v}_{1,i}\})$. We respectively note $\mathsf{v}_{0,i}$ and $\mathsf{v}_{1,i}$ the i-th element of S_0 and S_1. Since the voting system is registration free, the private credential of every voter is equal to $\perp$.

First, we note $\mathcal{I}^{\mathsf{b}}$ an instruction that outputs b as the message to send to CS and $\perp$ the receipt. As the voting system is non-interactive, the instruction stops there. We also define $\mathcal{X}$ as the the distribution of $(\mathsf{PB}, \mathsf{RCPT})$ produced as follows, and parametrized by the security parameter:

$$\begin{aligned}
&\mathsf{msg}, \mathsf{rcpt} \leftarrow \mathsf{Vote}_{\mathsf{V}}(\mathsf{id}_i, \mathsf{v}_{1,i})\\
&\mathsf{rcpt}', \mathsf{b} \leftarrow\!\$\ [\mathcal{D}(\mathsf{id}_i, \mathcal{I}^{\mathsf{msg}}, \mathsf{v}_{0,i}) \leftrightarrow \mathsf{Vote}_{\mathsf{CS}}(\mathsf{id}_i)]\\
&\mathsf{PB} \leftarrow \mathsf{PB}||\mathsf{b}\\
&\mathsf{RCPT} \leftarrow \mathsf{RCPT}||\mathsf{rcpt}
\end{aligned}$$

for i in range(n).

We then proceed to build the adversaries for each experiment. All our three adversaries create a distribution in their respective experiment that we call $\mathcal{X}_V, \mathcal{X}_{RF}$ and $\mathcal{X}_{\mathcal{D}}$ that we want to be equal to $\mathcal{X}$.

First, we build $\mathcal{A}_V$ in the following way. For the i-th voter id, $\mathcal{A}_V$ runs the $\mathcal{O}\mathsf{register}^{\mathsf{vd}}(\mathsf{id})$, the $\mathcal{O}\mathsf{check}^{\mathsf{vd}}(\mathsf{id})$ and the $\mathcal{O}\mathsf{vote}^{\mathsf{vd}}(\mathsf{id}, \mathsf{v}_{1,i})$ oracles. During this call of the $\mathcal{O}\mathsf{vote}^{\mathsf{vd}}$ oracle, $\mathcal{A}_V$ interacts with an honest execution of the $\mathsf{Vote}_{\mathsf{H}}$ algorithm and simulates an execution of $\mathsf{Vote}_{\mathsf{VD}}$ up to the point of sending a ballot b to CS. Instead, it runs the algorithm $\mathcal{D}(\mathsf{id}, \mathcal{I}^{\mathsf{b}}, \mathsf{v}_{0,i})$ and sends to CS the resulting ballot. Crucially, $\mathcal{I}^{\mathsf{b}}$ has the same distribution as $\mathsf{Vote}_{\mathsf{V}}(\mathsf{id}, \mathsf{v}_{1,i})$, and thus so from $\mathcal{D}$'s view. After all the oracle calls, the tally is computed on PB and every voter runs the $\mathsf{VerifyVote}$ algorithm. Finally, we define $\mathcal{X}_V$ as the pair of the bulletin board PB of the experiment and the list of receipts RCPT outputed by $\mathsf{Vote}_{\mathsf{H}}$ in the $\mathcal{O}\mathsf{vote}^{\mathsf{vd}}$ oracle call.

Second, we build $\mathcal{A}_{RF}$ in the following way. For the i-th voter id, $\mathcal{A}_{RF}$ runs the $\mathcal{O}\mathsf{register}^{\mathsf{rf}}(\mathsf{id})$ and the $\mathcal{O}\mathsf{corrupt}^{\mathsf{rf}}(\mathsf{id})$ oracles. Then, it runs the $\mathcal{O}\mathsf{receiptLR}^{\mathsf{rf}}(\mathsf{id}, \mathcal{I}^{\mathsf{b}}, \mathsf{v}_{0,i})$ oracle, where $\mathsf{b}, \mathsf{rcpt} \leftarrow\!\$\ \mathsf{Vote}_{\mathsf{V}}(\mathsf{id}, \mathsf{v}_{1,i})$. Finally, $\mathcal{A}_{RF}$ calls the $\mathcal{O}\mathsf{tally}^{\mathsf{rf}}$ oracle and receive the result and a proof of validity of the tally. For each of the voters, it runs the $\mathsf{VerifyVote}$ algorithm on its view of the bulletin board and the receipt received from the $\mathcal{O}\mathsf{receiptLR}^{\mathsf{rf}}$ oracle. If any of these checks returns 0, $\mathcal{A}_{RF}$ outputs 1. Otherwise, it outputs 0. Finally, we define $\mathcal{X}_{RF}$ as the pair of the bulletin board PB_0 of the experiment and the list of receipts RCPT outputted by the $\mathsf{Vote}_{\mathsf{V}}$ algorithm used to create the instruction $\mathcal{I}^{\mathsf{b}}$ of each $\mathcal{O}\mathsf{receiptLR}^{\mathsf{rf}}$ oracle call.

Third, $\mathcal{A}_{\mathcal{D}}$ runs as follows. For the i-th voter id, $\mathcal{A}_{\mathcal{D}}$ runs the $\mathcal{O}\mathsf{register}^{\mathsf{dev}}(\mathsf{id})$ oracle and the $\mathcal{O}\mathsf{vote}^{\mathsf{dev}}(\mathsf{id}, \mathcal{I}^{\mathsf{b}}, \mathsf{v})$ oracle with $\mathsf{b}, \mathsf{rcpt} \leftarrow\!\$\ \mathsf{Vote}_{\mathsf{V}}(\mathsf{id}, \mathsf{v}_{1,i})$. Finally, we define $\mathcal{X}_{\mathcal{D}}$ as the pair of the bulletin board PB_1 of the experiment and the list of receipts RCPT outputed by the $\mathsf{Vote}_{\mathsf{V}}$ algorithm used to create the instruction $\mathcal{I}^{\mathsf{b}}$ of each $\mathcal{O}\mathsf{vote}^{\mathsf{dev}}$ oracle call.

By construction, for all these three adversary, the ballots appended to the bulletin board and the receipts are computed exactly in the same way as $\mathcal{X}$ and these distributions are then equal: we have $\mathcal{X}_V = \mathcal{X}_{RF} = \mathcal{X}_{\mathcal{D}} = \mathcal{X}$. Let $\mathsf{Supp}(\mathcal{X}) = \{\mathsf{x} \,|\, \mathsf{Pr}_{\mathsf{y}\in\mathcal{X}}[\mathsf{y} = \mathsf{x}] \neq 0\}$ be the support of the distribution $\mathcal{X}$.

For each $(\mathsf{PB}, \mathsf{RCPT}) \in \mathsf{Supp}(\mathcal{X})$, we now look at the following cases:

- $\mathsf{VerifyVote}(\mathsf{rcpt}) = 1$ for all $\mathsf{rcpt} \in \mathsf{RCPT}$: We write $r = \mathsf{Tally}(\mathsf{PB}, \mathsf{sk})$, where the secret key sk of the election is equally distributed in all the experiments.
  - If $r = \rho(\{\mathcal{V}_i, \mathsf{v}_{0,i}\})$: In that case, $\mathcal{A}_V$ wins its experiment as $\rho(\{\mathcal{V}_i, \mathsf{v}_{0,i}\}) \neq \rho(\{\mathcal{V}_i, \mathsf{v}_{1,i}\})$. We write $\mathcal{W}_V = \{\mathsf{x} \in \mathsf{Supp}(\mathcal{X}) \,|\, \mathsf{x} \text{ satisfies this case}\}$.
  - If $r \neq \rho(\{\mathcal{V}_i, \mathsf{v}_{0,i}\})$: In that case, $\mathcal{A}_{\mathcal{D}}$ wins its experiment as the results computed in the experiments are such that $r_0 = \rho(\{\mathcal{V}_i, \mathsf{v}_{0,i}\})$ and $r_1 = r$. We write $\mathcal{W}_{\mathcal{D}} = \{\mathsf{x} \in \mathsf{Supp}(\mathcal{X}) \,|\, \mathsf{x} \text{ satisfies this case}\}$.
- $\mathsf{VerifyVote}(\mathsf{rcpt}) = 0$ for some $\mathsf{rcpt} \in \mathsf{RCPT}$: In that case, $\mathcal{A}_{RF}$ wins its experiment. Indeed, $\mathcal{A}_{RF}$ returns 1 if $\beta = 1$ because of the correctness of the voting protocol and returns 0 otherwise. We write $\mathcal{W}_{RF} = \{\mathsf{x} \in \mathsf{Supp}(\mathcal{X}) \,|\, \mathsf{x} \text{ satisfies this case}\}$.

Since $\forall \mathsf{x} \in \mathsf{Supp}(\mathcal{X})$ at least one of those three cases occurs, we have $\bigcup_{i\in\{V,RF,\mathcal{D}\}} \mathcal{W}_i = \mathsf{Supp}(\mathcal{X})$. Eventually,

$$\mathsf{Pr}_{\mathsf{y}\leftarrow\!\$\mathcal{X}_i}[\mathcal{A}_i \text{ wins } | \mathsf{x} = \mathsf{y}] \geq \mathsf{Pr}_{\mathsf{y}\leftarrow\!\$\mathcal{X}_i}[\mathsf{y} \in \mathcal{X}_i \,|\, x = y],$$

and, by the law of total probability, we conclude that

$$\begin{aligned}
&\sum_{i\in\{V,RF,\mathcal{D}\}} \mathsf{Adv}^{\mathsf{i}}_{\mathcal{A}_i}(\lambda)\\
&\quad= \sum_{\mathsf{x}\in\mathsf{Supp}(\mathcal{X})} \sum_i \mathsf{Pr}_{\mathsf{y}\leftarrow\!\$\mathcal{X}_i}[\mathcal{A}_i \text{ wins} \wedge \mathsf{x} = \mathsf{y}]\\
&\quad= \sum_{\mathsf{x}} \sum_i \mathsf{Pr}_{\mathsf{y}\leftarrow\!\$\mathcal{X}_i}[\mathcal{A}_i \text{ wins} \,|\, \mathsf{x} = \mathsf{y}]\mathsf{Pr}_{\mathsf{y}\leftarrow\!\$\mathcal{X}_i}[\mathsf{x} = \mathsf{y}]\\
&\quad\geq \sum_{\mathsf{x}} \sum_i \mathsf{Pr}_{\mathsf{y}\leftarrow\!\$\mathcal{X}_i}[\mathsf{y} \in \mathcal{X}_i \,|\, \mathsf{x} = \mathsf{y}]\mathsf{Pr}_{\mathsf{y}\leftarrow\!\$\mathcal{X}}[\mathsf{x} = \mathsf{y}]\\
&\quad= \sum_{\mathsf{x}} \mathsf{Pr}_{\mathsf{y}\leftarrow\!\$\mathcal{X}}[\mathsf{x} = \mathsf{y}] \sum_i \mathsf{Pr}_{\mathsf{y}\leftarrow\!\$\mathcal{X}_i}[\mathsf{y} \in \mathcal{W}_i \,|\, \mathsf{x} = \mathsf{y}]\\
&\quad\geq \sum_{\mathsf{x}\in\mathsf{Supp}(\mathcal{X})} \mathsf{Pr}_{\mathsf{y}\leftarrow\!\$\mathcal{X}}[\mathsf{x} = \mathsf{y}] \;=\; 1
\end{aligned}$$

Hence, even if we do not know which adversary wins with a non-negligible advantage, we get $\mathsf{Adv}^{\mathrm{vd-verif}}_{\mathcal{A}_V}(\lambda) + \mathsf{Adv}^{\mathrm{rf}}_{\mathcal{A}_{RF}}(\lambda) + \mathsf{Adv}^{\mathrm{dev}}_{\mathcal{A}_{\mathcal{D}}}(\lambda) \geq 1$. □

We stress that this impossibility result holds even if we considerably weaken the verifiability experiment by forbidding the adversary to corrupt voters and by forcing the adversary to make all the voters check their vote. Indeed, the adversary for the verifiability experiment that we build in the proof behaves in this way.

Now if the protocol has a registration or an interactive voting protocol, then we cannot build the adversaries of our proof. Indeed, if there is a registration, then the deceiving strategy take as input the private credentials of the voter to which the adversary for the verifiability experiment does not have access. Then, if the voting protocol is interactive, the adversary for the verifiability experiment cannot just send the output of the deceiving algorithm to CS. The instructions given as input to the deceiving algorithms are now interactions between the voter and CS that we cannot rewind if needed.

It is tempting to only use a registration (like BeleniosRF). Indeed, if the voting device does not have access to the secret credentials, we will give a protocol that has verifiability against a corrupted voting device and receipt-freeness. However, the secrecy of the private credentials is essential, as it is suggested by the following corollary.

Corollary 1. *There is no voting system with a non-interactive voting protocol that satisfies receipt-freeness and verifiability against a corrupted voting device with leaked credentials.*

Proof. We proceed similarly as in the previous proof but we add the private credential as input of the $\mathcal{I}$ and $\mathcal{D}$ algorithms. □

4. Building blocks

The protocols that we present in the next sections are based on a set existing building blocks that we introduce here. We will work in cyclic groups in which the DDH assumption is assumed to be hard. Given a security parameter λ, we use a PPT algorithm $\mathsf{GrpGen}(1^\lambda)$ to generate such a group $\mathbb{G}$ of order p such that $|p| = \mathsf{poly}(\lambda)$, together with a generator g. We also define a canonical encoding of $\mathcal{C}$ in $\mathbb{G}$ such that the first candidate is mapped to g, the second to g^2, and so on.

4.1. Partially homomorphic randomizable ciphertexts

A CPA-secure encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is said to be partially homomorphic if for some operations on the messages (for which we use the multiplicative notation), on the random coins (for which we use the additive notation) and on the ciphertexts (for which we use again the multiplicative notation) we have that $\mathsf{Enc}(\mathsf{pk}, m_0; r_0)\mathsf{Enc}(\mathsf{pk}, m_1; r_1) = \mathsf{Enc}(\mathsf{pk}, m_0 m_1; r_0 + r_1)$ for any $\mathsf{pk}, \mathsf{sk} \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$, any pair of messages m_0, m_1 and any pair of randomness r_0, r_1. This induces a natural ciphertext re-randomization algorithm Rand such that $\mathsf{Rand}(\mathsf{pk}, c) = c\mathsf{Enc}(\mathsf{pk}, 1; r)$ where 1 is the unit element and r is a fresh random coin.

The encryption scheme is re-randomizable if for any $\mathsf{pk}, \mathsf{sk} \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$, any plaintext m in the domain of Enc and any $c \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pk}, m)$, the distributions of $c_0 \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pk}, m)$ and $c_1 \leftarrow\!\!\$\ \mathsf{Rand}(\mathsf{pk}, c)$ are identical.

We also require Rand to have some additional properties focusing on maliciously generated ciphertexts, namely that for any $\mathsf{pk}, \mathsf{sk} \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$ and any ciphertext c, $\mathsf{Dec}(\mathsf{sk}, c) = \mathsf{Dec}(\mathsf{sk}, \mathsf{Rand}(\mathsf{pk}, c))$ and that for any adversary $\mathcal{A}$, the following is negligible:

$$Pr\left[\begin{array}{l|l} c \notin \mathsf{Enc}(\mathsf{pk}, m) \wedge & (\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda) \\ \mathsf{Dec}(\mathsf{sk}, c) = m \neq \perp & c \leftarrow\!\!\$\ \mathcal{A}(\mathsf{pk}) \end{array}\right]$$

This essentially ensures that even for a malicious ciphertext (which decrypts correctly), the distribution of the rerandomization of this ciphertext and the distribution of an honest encryption of the same message are statistically close. This property is particularly useful for building a receipt-free protocol, as an adversary cannot distinguish a re-randomized ciphertext posted on the board from a fresh encryption of the same message, which is indistinguishable from a fresh encryption of another message.

This can be instantiated by an ElGamal scheme [21] which is trivially re-randomizable under DDH assumption. We write $\mathsf{Enc}(\mathsf{pk}, m; r) = (g^r, \mathsf{pk}^r m)$ to encrypt m with randomness r and $\mathsf{Rand}(\mathsf{pk}, (c_0, c_1); s) = (c_0 g^s, c_1 \mathsf{pk}^s)$ to rerandomize (c_0, c_1) with randomness s.

4.2. Divertible proof

A divertible proof is an interactive protocol between three parties, which we call the prover, the intermediate and the verifier. In this protocol, the intermediate proves to the verifier a statement for which it does not know a witness by interacting with the prover. In our setting, we want an hybrid protocol that is interactive between the prover and the intermediate and non-interactive between the intermediate and the verifier. Indeed it is run between the voter (the prover) and CS (the intermediate) to prove that a ciphertext contains a valid vote. Regarding the verifier, we want the proof to be verifiable by anyone reading the bulletin board for the universal verifiability of the voting scheme. Concretely, the protocols between the intermediate and the verifier are made non-interactive via a Fiat-Shamir heuristic.

More precisely, we are interested in a divertible proof that an ElGamal ciphertext contains a plaintext in a given set of values $\mathcal{X}$. This is to ensure that the vote contained in a ciphertext is indeed valid, e.g., that the vote is 0 or 1 in the case of approval voting. While the voter can prove the knowledge of a witness for the ciphertext c sent to CS, that is, prove that $c \in \mathcal{L} = \{(g^\theta, h^\theta v) : v \in \mathcal{X}, \theta \in \mathbb{Z}_p\}$, we want to ultimately prove that a rerandomization c_r of c is in $\mathcal{L}$, for which none of the voter or CS has an opening.

We write such protocol as $\mathsf{view}_P, \pi \leftarrow [\mathsf{DivProve}_\mathsf{P}(c_r, w_P) \leftrightarrow \mathsf{DivProve}_\mathsf{I}(c_r, w_I)]$, where c_r is an Elgamal ciphertext and w_P, w_I are the witnesses

of the prover and the intermediate respectively. More precisely, for a ciphertext $c = (g^r, h^r v)$ randomized into $c_r = (g^{r+s}, h^{r+s} v)$, $w_P = (v, r)$ and $w_I = (c, s)$. view_P is the view of the prover during the protocol and π is the resulting non-interactive proof that can be verified with a Verify algorithm.

In addition of the typical completeness, soundness and zero-knowledge properties of a zero-knowledge proof, we also require that an adversarial prover cannot prove any relation between the witness w_P that she used and the final proof. That is, for any PPT prover P', there exists a PPT simulator $\mathcal{S}$ such that $\forall\, c_r \in \mathcal{L}, \mathsf{View}_{\mathsf{P}'}[\mathsf{P}'(\mathsf{c_r}, \mathsf{w}'_{\mathsf{P}}) \leftrightarrow \mathsf{DivProve}_{\mathsf{I}}(\mathsf{c_r}, \mathsf{w_I})] = \mathcal{S}(\mathsf{c_r})$.

This additional property, that we refer as prover obliviousness, is already used in prior works aiming to achieve receipt-freeness and is instantiated by efficient Σ-protocols [22] that uses three interactions.

4.3. Traceable Receipt-Free Encryption

TREncs are a recently proposed public key encryption primitive supporting the receipt-free submission of secret ballots [12]. A TREnc ciphertext has two special features (in addition to being a regular ciphertext).

First, each ciphertext comes with a public trace given by a Trace algorithm that is independent of the plaintext (think of a verification key for a one-time signature scheme for instance). This trace is what makes it possible for a voter to track his ballot through the voting system. More precisely, TREncs introduce a link key (generated by an algorithm $\mathsf{LGen}(\mathsf{pk})$) that enables the creation of multiple ciphertexts with an $\mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, m)$ algorithm such that all these ciphertexts have the same trace (that is, Trace always return the same value when given these ciphertexts as input). Crucially, the trace of a ciphertext does not depend on the message that is encrypted.

Then, each ciphertext, even if it was encrypted using adversarially chosen randomness (which is crucial for receipt-freeness), can be re-randomized into a fresh encryption of the same plaintext with the same trace. This guarantees that, if a voter computes and submits an encrypted vote, and if this ciphertext is re-randomized before being posted on a bulletin board, then the voter becomes unable to demonstrate the content of that ciphertext to any third party.

More formally, TREncs capture this inability to create a receipt through a new security game called TCCA security, presented in Definition 4. This security notion essentially guarantees that, if a voter submits a ciphertext that is randomized before it is posted on a public bulletin board, then the resulting ciphertext becomes indistinguishable from any other ciphertext that would have the same trace.

Definition 4 (TCCA)**.** *A TREnc is secure against traceable chosen-ciphertext attacks if for every efficient adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ the experiment $\mathsf{Exp}^{\mathrm{tcca}}_{\mathcal{A}}(\lambda)$ defined in Figure 5 (left) returns 1 with a probability negligibly close in λ to $\frac{1}{2}$.*

It is worth mentioning that initially, TREncs have an additional property ensuring that as long as a voter encrypts only one ciphertext with a link key (and keep this link key secret), it will be infeasible for anyone to produce a ciphertext encrypting a different message while having the same trace. This property was essentially used to guarantee verifiability and to ensure that a corrupted CS could not replace the vote contained in a ciphertext by another vote while keeping the same trace. This form of non-malleability gives insurance to the voter that her vote was rightly recorded after the rerandomization. However, we will not use this property and even exploit the same link key more than once as we will use other mechanisms for the verifiability.

$\mathsf{Exp}^{\mathrm{tcca}}_{\mathcal{A}}(\lambda)$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\$\ \mathsf{Gen}(1^\lambda)$
$(c_0, c_1, \mathsf{st}) \leftarrow\!\$\ \mathcal{A}_1^{\mathsf{Dec}(\cdot)}(\mathsf{pk})$
$b \leftarrow\!\$\ \{0,1\}$
if $\mathsf{Trace}(\mathsf{pk}, c_0) \neq \mathsf{Trace}(\mathsf{pk}, c_1)$ or $\mathsf{Ver}(\mathsf{pk}, c_0) = 0$ or $\mathsf{Ver}(\mathsf{pk}, c_1) = 0$ **then return** b
$c^\star \leftarrow\!\$\ \mathsf{Rand}(\mathsf{pk}, c_b)$
$b' \leftarrow\!\$\ \mathcal{A}_2^{\mathsf{Dec}^\star(\cdot)}(c^\star, \mathsf{st})$
return $b' = b$

Figure 5: TCCA experiments. In the TCCA experiment, $\mathcal{A}_2$ has access to a decryption oracle $\mathsf{Dec}^\star(\cdot)$ which, on input c, returns $\mathsf{Dec}(c)$ if $\mathsf{Trace}(\mathsf{pk}, c) \neq \mathsf{Trace}(\mathsf{pk}, c^\star)$ and test otherwise. Here, $\mathsf{Ver}(\mathsf{pk}, c)$ is an algorithm that returns 1 if and only if c is in the range of $\mathsf{Enc}(\mathsf{pk}, \cdot)$.

There exists instantiation of TREncs in pairing friendly groups under the SXDH assumption [12]. In particular, this instantiation contains an ElGamal homomorphic CPA secure part that can be stripped from this ciphertext. We will exploit this structure in one of our protocol.

4.4. Malleable non-interactive proofs

Similarly to re-randomizable encryption, a malleable proof system is a proof system with an additional PAdapt algorithm. In our case, we use a malleable proof to prove that a rerandomization of an encryption is still valid. Given a valid proof π that a randomizable ciphertext c verifies a statement, this PAdapt algorithm produces a new proof from π that $\mathsf{Rand}(\mathsf{pk}, c)$ verifies the same statement, using the old proof and the random coins of the rerandomization of c. For the receipt-freeness of our schemes, we require this new proof to be distributed as a fresh proof. That is for $c = \mathsf{Enc}(\mathsf{pk}, m; r)$ and $\tilde{c} = \mathsf{Rand}(\mathsf{pk}, c; s)$, the distributions of $\pi_0 \leftarrow\!\$\ \mathsf{Prove}(\tilde{c}, (m, r+s))$ and $\pi_1 \leftarrow\!\$\ \mathsf{PAdapt}(\pi, \tilde{c}, s)$ are identical.

For example, the Groth-Sahai proof system [23] is non-interactive and is known to have perfectly rerandomizable proofs, even for statements involving quadratic relations of committed values (in order to prove that a value is 0 or 1 for example) [24], [25]. This can be instantiated as well in a pairing friendly group under SXDH.

4.5. Plaintext Equality Proof

Given an encryption scheme with keys pk, sk, a Plaintext Equality Proof is a pair of algorithms $(\mathsf{Prove}, \mathsf{Verify})$

such that for any pair of ciphertexts c_0, c_1, $\mathsf{Prove}(c_0, c_1, \mathsf{sk})$ produces a bit indicating if $\mathsf{Dec}(\mathsf{sk}, c_0) = \mathsf{Dec}(\mathsf{sk}, c_1)$ or not and a zero-knowledge proof that this is the case. The Verify algorithm can be used by anyone knowing the public key to check the proof [26].

These proofs do not give any additional information about $\mathsf{Dec}(\mathsf{sk}, c_0)$ and $\mathsf{Dec}(\mathsf{sk}, c_1)$ to the verifiers, and if the key of the encryption scheme is shared by several parties, they also do not have any information about $\mathsf{Dec}(\mathsf{sk}, c_0)$ and $\mathsf{Dec}(\mathsf{sk}, c_1)$ if they do not collude. This can be instantiated with an ElGamal scheme under the DDH assumption [27].

5. Protocols

We now propose three protocols that have receipt-freeness and verifiability against all parties. The first two protocols demonstrates how we can achieve these security properties without voter registration and by using an interactive voting process. The first protocol has a constant number of rounds but is quite demanding for the voter, while the second protocol is much simpler for the voter but comes with a number of rounds that is linear in the security parameter. Our third protocol achieves the same security properties with a non-interactive voting process but depends on a voter registration phase. We will prove the security of these protocols in Section 6.

For the sake of simplicity, we first focus on approval voting elections of one candidate. That is, $\mathcal{C} = \{0, 1\}$ and $\rho(\{\mathsf{id}_i, \mathsf{v}_i\}) = \sum \mathsf{v}_i$. We then extend the protocol to deal with the encryption of a larger number of bits.

5.1. A Constant-Round Interactive protocol

The first protocol relies on a pair of secret values α_0, α_1 that the voter generates outside of her voting device. Instead of directly casting a vote v, the voter encrypts instead two multiplicative shares of her vote in two ciphertexts c_0 and c_1 which are randomized and published on PB. The voter will then ask the voting device and the casting server to publish an an opening of the ciphertext $c_0^{\alpha_0} c_1^{\alpha_1}$, which does not reveal anything about the vote. Since neither the voting device nor the casting server knows α_0 and α_1 in advance, they cannot predict the value of this opening when publishing the randomized ballot on the board. Hence, if they modify the value of c_0 or c_1, they will not be able to provide an opening consistent with the voter's ciphertexts. Crucially, the voter has to commit on these secret α_0, α_1 before giving them to the voting device and casting server, which is the form of interaction that is needed in this protocol. Otherwise, a voter would be unable to get around the instructions of a receipt-freeness adversary for the same reasons the voting device cannot modify the voter's ciphertexts. Here is a more precise description of our first interactive protocol Π_1.

1) $\mathsf{Setup}(1^\lambda)$: First, EA generates a group $\mathbb{G} \leftarrow\!\!\$\ \mathsf{GrpGen}(1^\lambda)$ and two additional generators h_0 and h_1 of $\mathbb{G}$. Then, the talliers compute a distributed Elgamal key pair $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow\!\!\$\ \mathsf{KeyGen}(\mathcal{G})$ with non-interactive proofs of knowledge of their shares of the secret key. Return $\mathsf{pk} = (\mathcal{G}, h_0, h_1, \mathsf{pk}_e)$, $\mathsf{sk} = \mathsf{sk}_e$.
2) $\mathsf{Vote}(\mathsf{id}, \mathsf{v})$: The voting procedure has 3 steps which are detailed in Fig 6.
   In the first step, the voter samples two uniformly random values $\alpha_0, \alpha_1 \in \mathbb{Z}_p$ such that $\alpha_0 \neq \alpha_1$. Then, it computes a multi-Pedersen commitment $\mathsf{Com} = g^t h_0^{\alpha_0} h_1^{\alpha_1}$ and sends this commitment to CS.
   In the second step, the voter encodes her v to the voting device. VD chooses two random shares of v in $\mathbb{G}$ such that $\mathsf{v} = \mathsf{v}_0 \mathsf{v}_1$. Then, it computes encryptions of these shares $c_0 = \mathsf{Enc}(\mathsf{pk}, \mathsf{v}_0; r_0)$ and $c_1 = \mathsf{Enc}(\mathsf{pk}, \mathsf{v}_1; r_1)$ and sends these values to CS. CS rerandomizes them into $\tilde{c}_0 = \mathsf{Rand}(\mathsf{pk}, c_0; s_0), \tilde{c}_1 = \mathsf{Rand}(\mathsf{pk}, c_1; s_1)$, and generates with the voter a divertible proof π that $\tilde{c}_0 \tilde{c}_1$ decrypts into a plaintext $\in \mathcal{C}$.
   In the last step, the voter sends to the voting device her opening of (t, α_0, α_1) of Com. VD computes $x = \alpha_0 r_0 + \alpha_1 r_1$ and $y = \mathsf{v}_0^{\alpha_0} \mathsf{v}_1^{\alpha_1}$ and sends $(t, \alpha_0, \alpha_1, x, y)$ to CS. Then, CS updates x into $\tilde{x} = x + \alpha_0 s_0 + \alpha_1 s_1$ and returns it to the voter. It publishes on PB the ballot $\mathsf{b} = (\tilde{c}_0, \tilde{c}_1, \pi, \mathsf{Com}, t, \alpha_0, \alpha_1, \tilde{x}, y)$.
   The receipt of the voter is $\mathsf{rcpt} = (\mathsf{v}, \mathsf{v}_0, \mathsf{v}_1, \tilde{c}_0, \tilde{c}_1, \alpha_0, \alpha_1)$.
3) $\mathsf{Valid}(\mathsf{b}, \mathsf{PB})$: Return 1 if b has the shape of $(\tilde{c}_0, \tilde{c}_1, \pi, \mathsf{Com}, t, \alpha_0, \alpha_1, \tilde{x}, y)$, if the proof π is valid, if $\alpha_0 \neq \alpha_1 \neq 0$ if t, α_0, α_1 is an opening of Com and if $\tilde{c}_0^{\alpha_0} \tilde{c}_1^{\alpha_1} = \mathsf{Enc}(\mathsf{pk}, y; \tilde{x})$.
4) $\mathsf{Tally}(\mathsf{PB}, \mathsf{sk})$: At the tally phase, the talliers first check for each ballot that it is valid. If this is not the case, the ballot is dropped. Finally, the tallier compute homomorphically the tally from the valid $c_0 c_1$ and publish a proof of correct decryption as well.
5) $\mathsf{VerifyVote}(\mathsf{PB}, \mathsf{rcpt})$: returns 1 if rcpt has the shape of $(\mathsf{v}, \mathsf{v}_0, \mathsf{v}_1, \tilde{c}_0, \tilde{c}_1, \alpha_0, \alpha_1)$, if there is a valid ballot b on PB with the same $\tilde{c}_0, \tilde{c}_1, \alpha_0, \alpha_1$ as in the rcpt, if $\alpha_0 \neq \alpha_1$ and if $\mathsf{v} = \mathsf{v}_0 \mathsf{v}_1$. Returns 0 otherwise.

We also propose the following deviating strategy $\mathcal{D}$ to cast a vote for v:

- Given some instruction $\mathcal{I}$, $\mathcal{D}$ first runs $\mathcal{I}$ once to get the commitment Com and sends it to CS.
- Then, $\mathcal{D}$ simulates an execution of $\mathsf{Vote}_{\mathsf{CS}}$ with $\mathcal{I}$. In the last step of the protocol, it receives an opening of t, α_0, α_1 of Com. If this is an invalid opening, $\mathcal{D}$ aborts.
- Using this opening, $\mathcal{D}$ finds the solution d_0, d_1 of this system (we use here the additive notation to have a clearer presentation):

$$\begin{pmatrix} 1 & 1 \\ \alpha_0 & \alpha_1 \end{pmatrix} \begin{pmatrix} d_0 \\ d_1 \end{pmatrix} = \begin{pmatrix} \mathsf{Enc}(\mathsf{pk}, \mathsf{v}; r) \\ \alpha_0 c_0 + \alpha_1 c_1 \end{pmatrix}$$

  and sends d_0, d_1 to CS. It receives a randomized $\tilde{d}_0, \tilde{d}_1$ and runs the divertible proof with CS.
- $\mathcal{D}$ finally rewinds $\mathcal{I}$ to the beginning of the divertible proof and runs it with the randomized $\tilde{d}_0, \tilde{d}_1$ received from CS. Then if $\mathcal{I}$ outputs a valid opening of Com in the last round, $\mathcal{D}$ sends t, α_0, α_1 to CS. It aborts

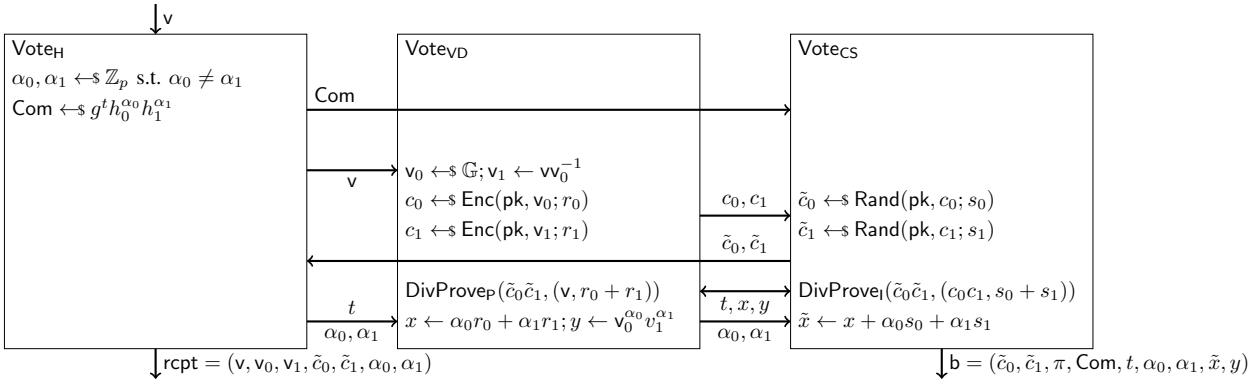

Figure 6: Description of the Vote algorithm of Π_1.

otherwise. Finally, $\mathcal{D}$ returns the output of $\mathcal{I}$ as the receipt.

On the number of rounds: With some optimizations, this protocol only needs three interactions between the voting device and the server. In the first interaction, VD sends Com, c_0 and c_1 and the commitments of the Σ-protocol for the divertible proof. It then receives the randomized ciphertext and the second message of the Σ-protocol in the second interaction. In the last interaction, VD sends the remaining values and the responses of the Σ-protocol.

On the vote deviation verifiability: This protocol is also verifiable when using the deviating strategy, at the cost of more work for the voter. We write $\mathcal{D}$ as an interactive protocol $\mathcal{D}_{\mathsf{H}} \leftrightarrow \mathcal{D}_{\mathsf{VD}}$, where $\mathcal{D}_{\mathsf{H}}$ runs most of the steps described above. It only needs to send $\mathsf{Com}, d_0, d_1, t, \alpha_0, \alpha_1$ to VD which forwards them to CS and run the divertible proof with VD forwarding the messages. Similarly as we did, we can show that if a corrupted VD forwards a pair of ciphertexts d'_0, d'_1 that contains another vote, then the voter will notice it with overwhelming probability.

Scalability for more than one bit: To encrypt more than one bit in a ballot, one can run the second and third round of the protocol in parallel for each additional bit, using the same α_0 and α_1. The protocol has thus the same round complexity and the number of operations grows linearly with the number of bits.

Instantiation: We can instantiate this protocol using the ElGamal encryption scheme and the divertible proof described in the building blocks and a multi-Pedersen commitment scheme. This can thus be build in a (plain) DDH group.

While this protocol is technically secure, it is impractical to consider that a voter can compute the commitment of the α_0, α_1 values without the voting device. They could use another trusted device, or multiple devices in a MPC-way, but this may not be very practical from a usability perspective and this approach still relies on some trust (if all the devices are corrupted, the protocol is not verifiable anymore).

We thus present another protocol which, despite the use of more interaction, is more tractable for the voter.

5.2. A human-compatible interactive protocol

The second protocol relies on a different approach. Instead of asking the voting device and casting server to provide an opening of $c_0^{\alpha_0} c_1^{\alpha_1}$, we ask them to provide an opening of either c_0 or c_1, depending on the choice of the voter. If a corrupted party modifies c_0 or c_1, it will have a probability $\frac{1}{2}$ to get caught. By repeating this process $\mathcal{O}(\lambda)$ times, we can amplify this probability as much as we want. Here is a more precise description of our first interactive protocol Π_2:

1) $\mathsf{Setup}(1^\lambda)$: First, EA generates a group $\mathbb{G} \leftarrow\$ \mathsf{GrpGen}(1^\lambda)$. Then, the talliers compute a distributed Elgamal key pair $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow\$ \mathsf{KeyGen}(\mathbb{G})$ with non-interactive proofs knowledge of their share of the key. Return $\mathsf{pk} = (\mathbb{G}, \mathsf{pk}_e)$, $\mathsf{sk} = \mathsf{sk}_e$.
2) $\mathsf{Vote}(\mathsf{id}, \mathsf{v}, \mathsf{PB})$: The voting procedure has $k = \mathcal{O}(\lambda)$ rounds. In the first round, the voter sends her vote v to the voting device, which computes $c = \mathsf{Enc}(\mathsf{pk}, \mathsf{v}; r)$ and sends the ciphertext to CS. Finally, CS rerandomizes it into $\tilde{c} = \mathsf{Rand}(\mathsf{pk}, c; s)$, and generates with the voter a divertible proof π that $\tilde{c}$ decrypts into a plaintext $\in \mathcal{C}$. Then, in each of the following round, the voting device chooses again random shares of v in $\mathbb{G}$ such that $\mathsf{v} = \mathsf{v}_0\mathsf{v}_1$. It computes $c_0 = \mathsf{Enc}(\mathsf{pk}, \mathsf{v}_0; r_0)$ and $c_1 = \mathsf{Enc}(\mathsf{pk}, \mathsf{v}_1; r_1)$ such that $r_0 + r_1 = r$. It then sends the ciphertext to CS, which rerandomize them into $\tilde{c}_0 = \mathsf{Rand}(\mathsf{pk}, c_0; s_0), \tilde{c}_1 = \mathsf{Rand}(\mathsf{pk}, c_1; s_1)$ such that $s_0 + s_1 = s$ and sends them to the voter.
The voter can either choose to drop the ballot or to cast it. If the voter chooses to drop it, the voting device starts again the round from the beginning, chooses new shares of the vote and proceeds as before. If the voter chooses to cast it, it also chooses a bit b and sends v_b, r_b and b to CS. In that case, CS publishes on PB the randomized ciphertexts, the bit b, v_b and $z = r_b + s_b$. Anyone can check that $c_b = (g^z, \mathsf{v}_b\mathsf{pk}^z)$. If the voter is not trying to deviate from the instructions of a receipt-free adversary, she can always choose to cast the ballot.
The ballot published on the board has thus the shape of $(\tilde{c}, \pi, \{\tilde{c}_{0,i}, \tilde{c}_{1,i}, b_i, \mathsf{v}_i, z_i\}_{i=1}^k)$ and the receipt of the voter is $\mathsf{rcpt} = (\mathsf{v}, \tilde{c}_0, \tilde{c}_1, \{b_i, \mathsf{v}_{b_i}, \tilde{c}_{0,i}, \tilde{c}_{1,i}\}_{i=1}^k)$.

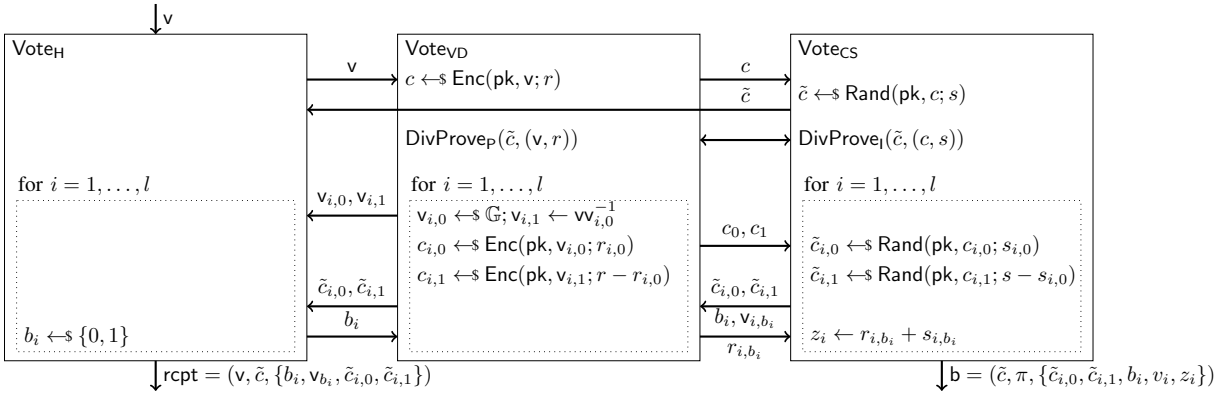

Figure 7: Description of the Vote algorithm of Π_2.

3) Valid(b, PB): Return 1 if b has the shape of $(\tilde{c}, \pi, \{\tilde{c}_{0,i}, \tilde{c}_{1,i}, b_i, \mathsf{v}_i, z_i\}_{i=1}^k)$, if π is a valid proof, if $\tilde{c} = \tilde{c}_{0,i}\tilde{c}_{1,i}$ and if $\tilde{c}_{b_i,i} = (g_i^z, \mathsf{v}_i \mathsf{pk}^{z_i})$ for all i.
4) Tally(PB, sk): At the tally phase, the talliers first check for each ballot that it is valid. If this is not the case, the ballot is dropped. Finally, the tallier compute homomorphically the tally from the valid $\tilde{c}$ and publish a proof of correct decryption as well.
5) VerifyVote(PB, rcpt): In the verification phase, each voter returns 1 if for the receipt $\mathsf{rcpt} = (\mathsf{v}, \tilde{c}, \{b_i, \mathsf{v}_{b_i}, \tilde{c}_{i,0}, \tilde{c}_{i,1}\}_{i=1}^k)$, if there is a ballot b on PB with the same ciphertext $\tilde{c}$ and the same tuples $(\tilde{c}_{i,0}, \tilde{c}_{i,1}, b_i, \mathsf{v}_{b_i})$ for each round as in the rcpt.

We also propose the following deviating strategy $\mathcal{D}$:

- Given some instruction $\mathcal{I}$, $\mathcal{D}$ first run $\mathcal{I}$ once to get a ciphertext c' that will be ignored. Then, it computes its own $d = \mathsf{Enc}(\mathsf{pk}, \mathsf{v}; r)$ and sends it to CS. It gets the rerandomization $\tilde{d}$ and computes the divertible proof protocol π that the rerandomization of d is decrypted into a plaintext $\in \mathcal{C}$. Also, it simulates an execution of this protocol with $\mathcal{I}$ as in the deviation strategy of Π_1.
- Then in the i-th round, $\mathcal{D}$ runs $\mathcal{I}$ with the current state. We call the current state as the initial $(\tilde{d}, \pi)$ augmented by each of the part of the ballot that is cast. $\mathcal{I}$ outputs a tuple $(c_{i,0}, c_{i,1})$ and $\mathcal{D}$ chooses a bit b_i at random. It computes an encryption $d_i = \frac{d}{c_{1-b_i}}$ and it sends to CS either (d_i, c_1) if $b_i = 0$ or (c_0, d_i) if $b_i = 1$. $\mathcal{D}$ then receives a rerandomization of these ciphertexts from CS and it forwards these values to $\mathcal{I}$ which output a bit $b_{i,\mathcal{I}}$. If $b_{i,\mathcal{I}} = b_i$, then $\mathcal{D}$ sends a drop message to CS and starts again the previous step. It also rewinds $\mathcal{I}$ to its state before the start of this previous step.
  If $b_{i,\mathcal{I}} \neq b_i$, it receives from $\mathcal{I}$ the values $v_{b_{i,\mathcal{I}}}$ and $r_{b_{i,\mathcal{I}}}$. $\mathcal{D}$ then casts the ballot with these values and repeats this step until k values have been posted on the board.
- $\mathcal{D}$'s receipt is the same as $\mathcal{I}$'s receipt.

On the number of rounds: It is possible to pack several rounds together. Instead of sending only one tuple (c_0, c_1), the voter can send l tuples in one round, and ask one opening for each of those tuple in the next round. However, the deviating strategy has now a probability $\frac{1}{2^l}$ to guess the value which will not reveal the deviating v to the adversary. To keep an expected polynomial time deviating strategy, we can thus batch at most $\mathcal{O}(\log^c \lambda)$ rounds together.

On the vote deviation verifiability: Unlike the previous protocol, this protocol does not have verifiability against a corrupted voting device if the voter use the $\mathcal{D}$ algorithm. Indeed, for an instruction that always ask to reveal the bit $b = 0$, the voting device can cheat on the d_1 ciphertext without being detected. It is however possible to slightly change the protocol to attain this notion of verifiability. Instead of choosing the random bit on her own, the voter, runs a verifiable coin flipping algorithm with CS. This can be done by VD in the honest algorithm but must be done by the voter herself in the vote deviation algorithm.

Scalability for more than one bit: To encrypt more than one bit in a ballot, one can encrypt each bit independently in the setup round. Then in each of the following round, we compute two encryptions of the shares of each bits and ask CS to open the same encryption for all the bits. The deviation strategy proceeds similarly by always replacing the same share of the adversary by the desired value. The protocol has thus also the same round complexity and the number of operations also grows linearly with the number of bits.

Instantiation: We can instantiate this protocol using the ElGamal encryption scheme and the divertible proof described in the building blocks. Again, we can thus instantiate the protocol in a (plain) DDH group.

5.3. Protocol with registration

We now show how to modify the first protocol Π_1 to replace the interactions with a registration process.

5.3.1. Getting rid of the commit-then-open mechanism.

We make three changes in Π_1:

1) Instead of choosing α_0, α_1, the voter receives them from the registration authority as there is no interaction to prove the knowledge of them.
2) As the protocol is single pass, the voter cannot give the opening of y, x to CS in a later round. However the voter can send $y = v_0^{\alpha_0} v_1^{\alpha_1}$ to the voting device and still have the individual verifiability. Indeed, even if the

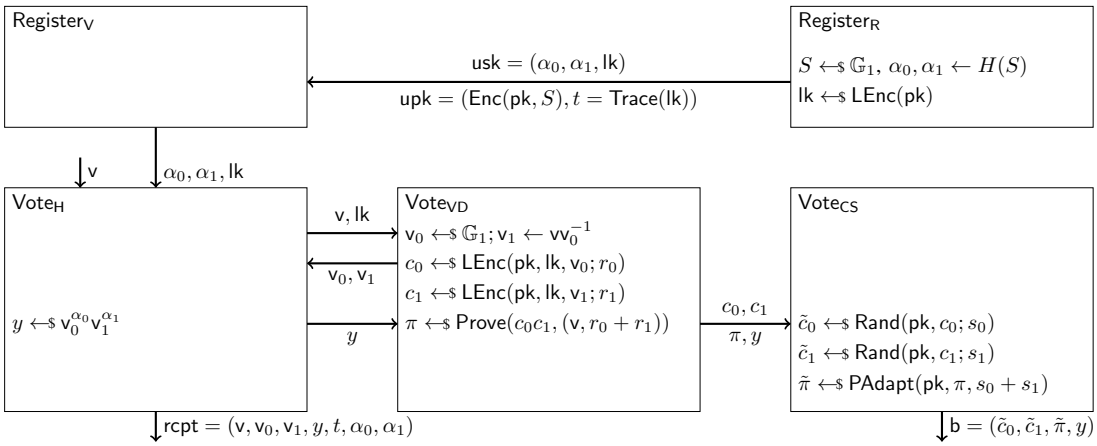

Figure 8: Description of the Vote algorithm of Π_3

voting device knows the target, it does not help to pass the test.

3) As there is no opening of the $\tilde{c}_0^{\alpha_0}\tilde{c}_1^{\alpha_1}$ value on the bulletin board to test the authenticity of the ciphertext, we proceed differently. Instead, the talliers publish the result of a plaintext equivalence test (PET) that $\tilde{c}_0^{\alpha_0}\tilde{c}_1^{\alpha_1}$ and $\mathsf{Enc}(\mathsf{pk}, y; 0)$ contain the same value.

5.3.2. Non-interactive proof of valid vote. We still need to make a non-interactive proof that $\tilde{c}_0\tilde{c}_1$ contains a valid vote. Instead of the divertible proof protocol, we could use a simulation sound randomizable proof. However, such a ballot would be replayable: an adversary could copy a ballot on the bulletin board and cast a rerandomization of it in order to break the receipt-freeness or even the ballot privacy [13]. In order to address this problem, we turn to a traceable receipt-free encryption scheme (TREnc), which offers the TCCA security notion that we need [12].

5.3.3. Non-interactive Protocol. We propose a non-interactive protocol Π_3 that makes use of the registration system:

1) $\mathsf{Setup}(1^\lambda)$: First, EA generates a bilinear group $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T) \leftarrow\$ \mathsf{GrpGen}(1^\lambda)$ and the CRS crs of a malleable non-interactive proof system as described in section 4 in a public coin manner. Then, the talliers compute a distributed TREnc key pair $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow\$ \mathsf{KeyGen}(\mathbb{G}_1)$, with non-interactive proofs of knowledge of their share of the key. Return $\mathsf{pk} = (\mathcal{G}, \mathsf{crs}, \mathsf{pk}_e)$, $\mathsf{sk} = \mathsf{sk}_e$.
2) $\mathsf{Register}(\mathsf{id}, 1^\lambda)$: We first generate random $S \leftarrow\$ \mathbb{G}_1$ and set α_0, α_1 as the hash $H(S)$. Then, we generate the link key of a TREnc with the $\mathsf{lk} \leftarrow\$ \mathsf{LGen}(\mathsf{pk})$ algorithm. Finally, we set usk_id to $(\alpha_0, \alpha_1, \mathsf{lk})$ (which is sent to the voter) and upk_id to $(\mathsf{Enc}(\mathsf{pk}, S), t)$ (which is sent to the bulletin board) where t is the trace of the TREnc linked to lk .
3) $\mathsf{Vote}(\mathsf{id}, \mathsf{v})$: The voter first encodes v and lk in the voting device. Then the voting device chooses two shares $(\mathsf{v}_0, \mathsf{v}_1)$ of v in $\mathbb{G}_1$ such that $\mathsf{v} = \mathsf{v}_0\mathsf{v}_1$. From these shares, the voter computes the target value $y = \mathsf{v}_0^{\alpha_0}\mathsf{v}_1^{\alpha_1}$ and encodes it in the voting device.
Then, the device computes $c_0 = \mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, \mathsf{v}_0)$ and $c_1 = \mathsf{Enc}(\mathsf{pk}, \mathsf{lk}, \mathsf{v}_1)$ and a non-interactive divertible proof π that c_0c_1 decrypts into a plaintext $\in \mathcal{C}$. (c_0, c_1, π, y) is then sent to CS. CS then randomizes the ciphertext into $(\tilde{c}_0, \tilde{c}_1, \tilde{\pi}, y)$ and publishes it on the bulletin board next to the id of the voter. In case of revote, the last ballot will be counted.
We note $\mathsf{b} = (\tilde{c}_0, \tilde{c}_1, \tilde{\pi}, y)$ and $\mathsf{rcpt} = (\mathsf{v}, \mathsf{v}_0, \mathsf{v}_1, y, t, \alpha_0, \alpha_1)$.
4) $\mathsf{Valid}(\mathsf{b}, \mathsf{PB})$: Return 1 if b has the shape of $(\mathsf{id}, \tilde{c}_0, \tilde{c}_1, \tilde{\pi})$, if $\mathsf{Trace}(\tilde{c}_0) = \mathsf{Trace}(\tilde{c}_1) = t$ where t is the trace of upk_id and if the proof is valid.
5) $\mathsf{Tally}(\mathsf{PB}, \mathsf{sk})$: First the talliers decrypt together each of the upk_id to get the α_0 and α_1 of the valid ballots. Then, they run the plaintext equality proof algorithm to check that each of the $c_0^{\alpha_0}c_1^{\alpha_1}$ is an encryption of the y value of the voter and publish the output π_{PEP} of the algorithm on the bulletin board. If the test is passed, they compute homomorphically the tally from the valid $\tilde{c}_0\tilde{c}_1$ and publish a non-interactive zero-knowledge proof (using a standard Σ protocol) that the result is indeed a correct decryption.
6) $\mathsf{VerifyVote}(\mathsf{PB}, \mathsf{rcpt})$: returns 1 if rcpt has the shape of $(\mathsf{v}, \mathsf{v}_0, \mathsf{v}_1, y, t, \alpha_0, \alpha_1)$, if there is a valid ballot b on PB with trace t and the same y value as in the rcpt, if the decrypted α_0, α_1 associated to this ballot are the same as the values in the receipt, if the associated π_{PEP} is verified and if $\mathsf{v} = \mathsf{v}_0\mathsf{v}_1$. Returns 0 otherwise.

Since there is no interaction, the deviating strategy is simpler:

- Given some instruction $\mathcal{I}$, the voter first run $\mathcal{I}$ once to get the ballot c_0, c_1, π, t to send to the server.
- To cast a vote v, the voter finds the solution d_0, d_1 of this system:

$$\begin{pmatrix} 1 & 1 \\ \alpha_0 & \alpha_1 \end{pmatrix} \begin{pmatrix} d_0 \\ d_1 \end{pmatrix} = \begin{pmatrix} \mathsf{Enc}(\mathsf{pk}, \mathsf{v}; r) \\ \alpha_0 c_0 + \alpha_1 c_1 \end{pmatrix} \quad (1)$$

and compute the associated proof.

- The voter use d_0 and d_1 for the CPA parts of two TREnc encryptions with lk, compute her own proof and sends the resulting ballot to CS.

On the vote deviation verifiability: This protocol has vote deviation verifiability for the same reason as Π_1.

Scalability for more than one bit: To encrypt more than one bit in a ballot, one can run the vote protocol for each bit independently, using the same link key lk. Thus the credentials keep a constant size and the number of operations also grows linearly with the number of bits.

Instantiation: We can instantiate this protocol using the TREnc, the malleable proof system and the plaintext equality proof described in the building blocks. Those blocks are compatible, since the TREnc of [12] contains an ElGamal CPA-secure part that can be used for the homomorphic operations and the plaintext equality test. Hence, we can thus instantiate the protocol in a pairing-friendly SXDH group.

Even though this theorem does not give any guarantee about verifiability in presence of malicious parties besides the voting device, it can be shown that the protocol is secure as long as neither the voting device nor CS has access to the private credentials of the voter.

However, Π_3 cannot be secure against a corrupted voting device if the credentials are leaked, as the voting device could follow the same deviating strategy. As hinted by Corollary 1, we need interactions in the voting procedure to achieve this.

6. Security of the protocols

Theorem 2. *Protocol Π_1 has receipt-freeness and is verifiable against all parties. Especially*

- *Π_1 has receipt-freeness if the encryption scheme is CPA-secure, partially homomorphic and rerandomizable and if the divertible proof has the zero-knowledge, special soundness and prover obliviousness properties.*
- *Π_1 is verifiable against all parties if the encryption scheme is partially homomorphic and if the proofs computed by the talliers are sound.*

Proof. **Receipt-freeness**: For a simplified Vote protocol à la Helios that outputs a ballot (c, π) where c is an Elgamal encryption of the vote and π is a proof of knowledge of the plaintext and the randomness of c, it has been shown that ballot privacy in the sense of BPRIV holds [30][2]. Hence we use the same usual $\mathsf{SimSetup}$ and $\mathsf{SimProof}$ algorithms as those used to prove the BPRIV security [5] of Helios-like systems.

We will aim to reduce the receipt-freeness game to a BPRIV game of this simplified protocol. But first, we define an intermediate game as follows: in each of the $\mathcal{O}\mathsf{receiptLR}^{\mathsf{rf}}$, $\mathcal{O}\mathsf{voteLR}^{\mathsf{rf}}$ and $\mathcal{O}\mathsf{cast}^{\mathsf{rf}}$ request, we simulate the divertible proof that the ballot has a valid vote. From the prover obliviousness property of this proof, even the proving party participating in this proof cannot distinguish between a real proof and a simulated proof.

From this game, we can now reduce to the BPRIV game. To do so, we will transform every $\mathcal{O}\mathsf{receiptLR}^{\mathsf{rf}}$ request into a $\mathcal{O}\mathsf{voteLR}^{\mathsf{rf}}$ request of the BPRIV game.

First, we have to recover the votes to use in the $\mathcal{O}\mathsf{voteLR}^{\mathsf{rf}}$ request. For the ballot posted in PB_0, we already have the vote v as a parameter of $\mathcal{O}\mathsf{receiptLR}^{\mathsf{rf}}$. For the other ballot, we run $\mathcal{I}$ up to the divertible proof. Using the special soundness of the proof, we can rewind and extract from $\mathcal{I}$ an opening of the ciphertext it would send to CS, from which we can recover the vote. We call the $\mathcal{O}\mathsf{voteLR}^{\mathsf{rf}}$ oracle with these two votes.

Then, given the ballot (c, π) of the simplified protocol, we transform it in the following way. We run $\mathcal{I}$ to get the commitment Com and simulates the interactions with CS until we receive the opening t, α_0, α_1 of Com and the value y in the last round of the protocol. Then, we draw a random $\tilde{x}$ and compute $\tilde{c}_0, \tilde{c}_1$ such that:

$$\begin{pmatrix} 1 & 1 \\ \alpha_0 & \alpha_1 \end{pmatrix} \begin{pmatrix} \tilde{c}_0 \\ \tilde{c}_1 \end{pmatrix} = \begin{pmatrix} c \\ \mathsf{Enc}(\mathsf{pk}, y; \tilde{x}) \end{pmatrix}$$

We then rewind $\mathcal{I}$ once again and rerun the protocol one last time with the $(\tilde{c}_0, \tilde{c}_1)$ previously computed. We append the resulting ballot $\mathsf{b} = (\tilde{c}_0, \tilde{c}_1, \pi, \mathsf{Com}, t, \alpha_0, \alpha_1, \tilde{x}, y)$ on our bulletin board and return the receipt outputted by $\mathcal{I}$ to the adversary.

This procedure is equivalent to the $\mathcal{D}$ strategy, only that we do not know the opening of c and thus simulate the proof instead. From the rerandomizability of the encryption scheme, $\mathcal{I}$ cannot distinguish between $\tilde{c}_0, \tilde{c}_1$ and a real rerandomization of the ciphertext it has outputted.

Vote deviation correctness: In each of the $\mathcal{O}\mathsf{voteLR}(\mathsf{id}, \mathcal{I}, \mathsf{v})$ call, either $\mathcal{I}$ does not produce a valid opening of Com such that $\alpha_0 \neq \alpha_1$ (and in that case $\mathcal{I}$ forces the ballot to be dropped), or from this opening the linear system used in $\mathcal{D}$ has a solution. If so, the crafted ballot has the same distribution as an output of $\mathsf{Vote}_{\mathsf{H}}$.

Verifiability against all parties: We proceed as a sequence of hops, starting with the initial verifiability experiment.

Game 1: In the first game, we create a trapdoor for the multi-Pedersen commitments. Namely, instead of picking h_0 and h_1 as random group elements, elements we generate $h_0 = g^{\gamma_0}$ and $h_1 = g^{\gamma_1}$.

This is statistically indistinguishable for the adversary.

Game 2: In the second game, instead of computing a commitment for α_0 and α_1 as the first step of $\mathsf{Vote}_{\mathsf{H}}$, we compute a commitment for random α'_0 and α'_1. Using the trapdoor, we compute a consistent opening t, α_0, α_1.

This is statistically indistinguishable for the adversary as well, this is essentially the perfectly hiding property of the commitment scheme.

Game 3: In the third game, we abort if for some honest voter, we have a $\mathsf{rcpt} = (\mathsf{v}, \mathsf{v}_0, \mathsf{v}_1, \tilde{c}_0, \tilde{c}_1, \alpha_0, \alpha_1)$ such that $\mathsf{Dec}(\mathsf{sk}, \tilde{c}_0\tilde{c}_1) \neq \mathsf{v}$ but $\mathsf{VerifyVote}(\mathsf{PB}, \mathsf{rcpt}) = 1$. Then for

2. The BPRIV experiment can essentially be seen as our receipt-freeness experiment without the $\mathcal{O}\mathsf{receiptLR}^{\mathsf{rf}}$ oracle

$\mathsf{v}_{0,\mathcal{A}} = \mathsf{Dec}(\mathsf{sk}, \tilde{c}_0)$ and $\mathsf{v}_{1,\mathcal{A}} = \mathsf{Dec}(\mathsf{sk}, \tilde{c}_1)$, we have that either $\mathsf{v}_{0,\mathcal{A}} \neq \mathsf{v}_0$ or $\mathsf{v}_{1,\mathcal{A}} \neq \mathsf{v}_1$.

This event happens with negligible probability. Since $\tilde{c}_0, \tilde{c}_1, \tilde{\pi}$ are computed before $\mathcal{A}$ has any information about (α_0, α_1), $\mathcal{A}$ has to guess the value of $\mathsf{v}_0^{\alpha_0}\mathsf{v}_1^{\alpha_1}$ at random, and the the probability that $\mathsf{v}_{0,\mathcal{A}}^{\alpha_0}\mathsf{v}_{1,\mathcal{A}}^{\alpha_1}$ is equal to $\mathsf{v}_0^{\alpha_0}\mathsf{v}_1^{\alpha_1}$ is bounded by $\frac{1}{p}$.

From this game, it is straightforward to prove the result using standard universal verifiability techniques à la Helios. □

Theorem 3. *Protocol Π_2 has receipt-freeness and is verifiable against all parties. Especially*

- *Π_2 has receipt-freeness if the encryption scheme is CPA-secure, partially homomorphic and rerandomizable and if the divertible proof has the zero-knowledge, special soundness and prover obliviousness properties.*
- *Π_2 is verifiable against all parties if the encryption scheme is partially homomorphic, if the commitment scheme is perfectly hiding and if the proofs computed by the talliers are sound.*

Proof. **Receipt-freeness**: Like we did in Π_1, we aim to reduce the receipt-freeness game into a BPRIV game of a simplified Helios-like protocol whose ballots are a pair consisting of an El-Gamal ciphertext and a proof of knowledge of the content of the ciphertext.

Again, we define an intermediate game as follows: in each of the $\mathcal{O}\mathsf{receiptLR}^{\mathsf{rf}}$, $\mathcal{O}\mathsf{voteLR}^{\mathsf{rf}}$ and $\mathcal{O}\mathsf{cast}^{\mathsf{rf}}$ request, we simulate the divertible proof that the ballot has a valid vote. We can also extract $\mathcal{I}'s$ vote similarly and call $\mathcal{O}\mathsf{voteLR}^{\mathsf{rf}}$ to obtain a ballot c, π.

Then, given the ballot (c, π) of the simplified protocol, we transform it in the following way. For each round of the voting protocol, we run $\mathcal{I}$ to get a pair of ciphertext $(c_{i,0}, c_{i,1})$ that we randomize and then get (b, v_b, r_b). We rewind $\mathcal{I}$ to the last interaction and compute $z = r_b + s_b$ where s_b is a fresh randomness, $d_b = \mathsf{Enc}(\mathsf{pk}, \mathsf{v}_b; z)$ and $d_{1-b} = \frac{c}{d_b}$. We feed $\mathcal{I}$ with this new (d_0, d_1) and get again a tuple $(b', \mathsf{v}'_b, r'_b)$. If those values are different from the previous tuple, we rewind and try again with these new values. From the randomizability of the ciphertext and the CPA-security of the scheme, this happens with probability at most $\frac{1}{2}$, so we should ultimately get the same tuple. When this happens, we append $d_0, d_1, b, \mathsf{v}_b, z$ to the ballot and start the next round.

This procedure perfectly simulates a ballot and a receipt of the receipt-freeness game, independently of its origin (from the adversary's instructions or from the deviating strategy).

Vote deviation correctness: In the first step of the protocol, $\mathcal{D}$ encrypts a vote and compute the divertible proof in the same way as the $\mathsf{Vote}_{\mathsf{V}}$ algorithm. Hence if the resulting ballot is valid, it will be counted similarly.

Then in each round, the deviating algorithm has one chance over two to choose the bit that will not be chosen by $\mathcal{I}$. Indeed, from the DDH assumption and the witness indistinguishability of the proof, $\mathcal{I}$ cannot distinguish between a randomization of their ciphertext c and a randomization of the ciphertext of the deviating algorithm d. Since the chosen bit b is independent of $b_{\mathcal{I}}$, $\mathcal{D}$ will send a cast message with probability one over two. Hence for each round, the probability that it takes more than λ tries for the deviating strategy to send a "cast" message is bounded by $2^{-\lambda}$ (which is negligible) as the number of tries follows a geometric distribution of parameter $\frac{1}{2}$. Thus, the probability that for all the rounds combined the deviating strategy needs more than $k\lambda$ tries is also negligible.

The expected number of rounds before the termination of the protocol is thus linear and the ballot posted on PB correctly encodes the voter's intention.

Verifiability against a corrupted voting device: Let us modify the verifiability experiment in the following way: if for some honest voter, $\mathsf{Dec}(\mathsf{sk}, \tilde{c}) \neq \mathsf{v}$ but $\mathsf{VerifyVote}(\mathsf{PB}, \mathsf{rcpt}) = 1$ where rcpt is associated to this voter, then we abort. This is similar to the last game of the proof for protocol Π_1.

This event happens with negligible probability. Indeed, since the the adversary has no information about which ciphertext will be opened in each round when the randomized ciphertext $\tilde{c}_{i,0}, \tilde{c}_{i,1}$ is computed, it has at least one chance over two to modify a plaintext that will be seen by the voter. Also, it has to modify k of them. Since every product of ciphertexts is equal to the same value, it ensures that the adversary has to modify one plaintext in each round. Hence, it has at most a probability $\frac{1}{2^k}$ to modify the vote of a voter without being detected.

Again from this game, we can prove the result using standard universal verifiability techniques à la Helios. □

Theorem 4. *Protocol Π_3 has receipt-freeness and is verifiable against all parties. Especially*

- *Π_3 has receipt-freeness if the encryption scheme is TCCA-secure and partially homomorphic, if the malleable proofs are sound, rerandomizable and zero-knowledge and if the proof computed by the tallier are zero-knowledge.*
- *Π_3 is verifiable against all parties if the encryption scheme is CPA-secure and if the plaintext equality proof is sound.*

Proof. **Receipt-freeness**: We prove the result for the simple case of one honest tallier. This can be extended to threshold tallying using standard techniques. We proceed as a sequence of hops, starting with the initial receipt-freeness game.

Game 1: In the first game, we simulate all the proofs of the tallier using the zero-knowledge simulator. Regarding the plaintext equality proof, we proceed as follows. When we append a ballot (c_0, c_1, π, t) to the PB, we check if $\mathsf{Dec}(\mathsf{sk}, c_0^{\alpha_0} c_1^{\alpha_1})$ is equal to t. If this is the case, we prove the equality result. Otherwise, we prove the inequality result. From the zero-knowledge property of the game, this is indistinguishable from the initial game.

Game 2: In this game, we replace all the proofs of validity of the ballots by simulated proofs instead of rerandomizing them. From the zero-knowledge property of the proofs, this is again indistinguishable from the previous game.

Game 3: In the third game, we abort if there is a ballot $b = (c_0, c_1, \pi, t)$ that is appended on PB such that the proof π is valid but $\mathsf{Dec}(\mathsf{sk}, c_0c_1)$ is not a valid vote. From the simulation soundness of the proof, this event happens with negligible probability.

Game 4: In the fourth game, instead of computing the tally homomorphically from the ballots in PB_0, we decrypt each of the c_0c_1 associated to a valid ballot and compute the results from them. The result is computed in a different way but is equal in both case.

Game 5: In the next game, instead of rerandomizing and publishing the c_0 part of the ballot b_0 provided by the deviation strategy, we rerandomize and publish the c_0 part of the ballot b_1 provided by $\mathcal{I}$. Similarly, in the last game, we rerandomize the c_1 part of b_1 instead of the c_1 part of b_0 for the ballot published on PB_0. These games are indistinguishable thanks to the TCCA security of the TREnc scheme.

Vote deviation correctness: With overwhelming probability, $\alpha_0 \neq \alpha_1$ and the linear system used in $\mathcal{D}$ has a solution. If so, the crafted ballot has the same distribution as an output of $\mathsf{Vote}_{\mathsf{H}}$.

Verifiability against a corrupted voting device: We use an hybrid in which we replace the encryption of the private S of the voters on PB with trivial encryptions of 0. In the tally phase, the talliers directly use the $\mathsf{upk}_{\mathsf{id}}$ of each voter without decrypting the value of the bulletin board. From the CPA-security of the encryption scheme, this change is indistinguishable by the adversary.

Again, the ballot published on the board is computed before $\mathcal{A}$ has any information about (α_0, α_1). Using the same argument as protocol Π_1, $\mathsf{VerifyVote}$ will return 0 with overwhelming probability if the content of the ballot has been changed. We can then use standard universal verifiability techniques à la Helios to prove the result. □

7. Performances and comparisons

We provide a python implementation [28] to evaluate the time to encrypt and craft a ballot of various size using our protocols. We conducted our experiments on a laptop with a 1.8Ghz Intel i7 processor and 16GB of RAM running Ubuntu 20.04 LTS and Python 3.7.9. We used the Charm framework [29] with pbc-0.5.14 and OpenSSL-1.0.1. We used a secp256k1 elliptic curve for Π_1 and Π_2 and a BN254 pairing friendly curve for Π_3. For the protocol Π_2, we set the number of round to 16, which means that an adversary can change the voter of a voter without being detected with probability at most 2^{16}. All our results are averaged over 100 runs.

	lb = 1	2	4	8	32
Π_1	0.005	0.009	0.018	0.035	0.137
Π_2	0.026	0.047	0.097	0.195	0.763
Π_3	0.139	0.288	0.584	1.231	4.819

TABLE 1: Time to encrypt the vote of a ballot containing $1, 2, 4, 32$ bits. The times reported are in seconds.

We see in Table 1 that even for a 32-choices race, the encrypting time of the protocol with registration is only a couple of seconds. It is not surprising that this is protocol is considerably slower than the other two protocols, as it uses a pairing friendly group. The first two protocols are faster, but only the computation time on the voter's device is computed. This means that the communication time with the server is ignored, which would not be negligible. We stress that our implementation prototype was designed to offer an upper-bound on the running time of our protocols (in the absence of deviation strategy) and to verify that this running time is not prohibitive for natural use cases. Faster implementations could be obtained easily if needed.

We also feel the computational overhead of our protocols, compared to traditional techniques, remains within reasonable bounds. For instance, for submitting an encrypted bit, our first protocol computes 14 exponentiations (using the suggested building blocks): 3 for the computation of the commitment, 4 for the computation of the encryptions, 2 for the computation of the value y and 5 for the computation of the divertible proof. Compared to a standard ElGamal + zero-knowledge proof of validity, as implemented in ElectionGuard 2.0 for instance, this is 8 additional exponentiations. This computational overhead of course comes with considerably stronger security properties in terms of receipt-freeness, which is not a goal of the ElectionGuard SDK.

The main properties and performances of our protocol are summarized in Figure 9.

Acknowledgments. Henri Devillez is a research fellow of the Belgian FRIA. Thomas Peters is research associate of the Belgian Fund for Scientific Research (F.R.S.-FNRS). This work has been supported in part by a Microsoft Research Award.

References

[1] Council of Europe, "Recommendation cm/rec(2017)5 of the committee of ministers to member stateson standards for e-voting," 2017.

[2] Swiss Federal Chancellery, "Federal chancellery ordinance on electronic voting of 25 may 2022," Available from https://www.fedlex.admin.ch/eli/cc/2022/336/en.

[3] D. Clarke and T. Martens, *Real-World Electronic Voting: Design, Analysis and Deployment*. CRC Press, 2016, ch. E-voting in Estonia.

[4] A. Debant and L. Hirschi, "Reversing, breaking, and fixing the french legislative election e-voting protocol," Cryptology ePrint Archive, Paper 2022/1653, 2022.

[5] D. Bernhard, V. Cortier, D. Galindo, O. Pereira, and B. Warinschi, "SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions," in *S&P'15*. IEEE Computer Society, 2015.

[6] V. Cortier, D. Galindo, R. Küsters, J. Müller, and T. Truderung, "Sok: Verifiability notions for e-voting protocols," in *IEEE Symposium on Security and Privacy, SP 2016*. IEEE Computer Society, 2016.

	Π_1	Π_2	Π_3
# rounds	3	$\mathcal{O}(\log^c \lambda)$	1
Registration	no		yes
Receipt-Freeness	yes		
Verifiability	against all parties		against the voting device
Complexity of Vote_H	complex; need to compute a Pedersen commitment	easy; only need to sample random bits	complex; need to compute $y = \mathsf{v}_0^{\alpha_0}\mathsf{v}_1^{\alpha_1}$
Security assumption	plain DDH	plain DDH	pairing-friendly SXDH
Time to create a ballot (for one bit, in seconds)	0.005	0.026 (k = 16)	0.139
Ballot size (for one bit)	6 elements in $\mathbb{G}$ 8 elements in $\mathbb{Z}_p$ ≈ 448 bytes	$5k+2$ elements in $\mathbb{G}$ $k+4$ elements in $\mathbb{Z}_p$ ≈ 3264 bytes (k = 16)	30 elements in $\mathbb{G}_1$ 14 elements in $\mathbb{G}_2$ ≈ 1856 bytes
Scalability	Linear in the number of bits (for time and ballot size)		

Figure 9: Summary of the properties and performances of the protocols.

[7] J. C. Benaloh and D. Tuinstra, "Receipt-free secret-ballot elections (extended abstract)," in *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*. ACM, 1994.

[8] M. Hirt and K. Sako, "Efficient receipt-free voting based on homomorphic encryption," in *Advances in Cryptology - EUROCRYPT 2000*, ser. Lecture Notes in Computer Science, vol. 1807. Springer, 2000.

[9] A. Juels, D. Catalano, and M. Jakobsson, "Coercion-resistant electronic elections," in *ACM Workshop on Privacy in the Electronic Society, WPES'05*. ACM, 2005.

[10] P. Chaidos, V. Cortier, G. Fuchsbauer, and D. Galindo, "BeleniosRF: A Non-interactive Receipt-Free Electronic Voting Scheme," in *CCS'16*. ACM, 2016.

[11] W. Lueks, I. Querejeta-Azurmendi, and C. Troncoso, "Voteagain: A scalable coercion-resistant voting system," in *29th USENIX Security Symposium, USENIX Security 2020*. USENIX Association, 2020.

[12] H. Devillez, O. Pereira, and T. Peters, "Traceable receipt-free encryption," in *Advances in Cryptology - ASIACRYPT 2022*, ser. LNCS, vol. 13793. Springer, 2022.

[13] D. Bernhard, V. Cortier, O. Pereira, B. Smyth, and B. Warinschi, "Adapting helios for provable ballot privacy," in *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security Proceedings*, ser. Lecture Notes in Computer Science, vol. 6879. Springer, 2011.

[14] B. Chevallier-Mames, P. Fouque, D. Pointcheval, J. Stern, and J. Traoré, "On some incompatible properties of voting schemes," in *Towards Trustworthy Elections, New Directions in Electronic Voting*, ser. LNCS, vol. 6000. Springer, 2010.

[15] V. Cortier and J. Lallemand, "Voting: You can't have privacy without individual verifiability," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*. ACM, 2018, pp. 53–66.

[16] B. Adida, O. de Marneffe, O. Pereira, and J. Quisquater, "Electing a university president using open-audit voting: Analysis of real-world use of helios," in *2009 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '09*. USENIX Association, 2009.

[17] M. Bougon, H. Chabanne, V. Cortier, A. Debant, E. Dottax, J. Dreier, P. Gaudry, and M. Turuani, "Themis: An on-site voting system with systematic cast-as-intended verification and partial accountability," in *CCS'22*. ACM, 2022.

[18] J. Benaloh, "Simple Verifiable Elections," in *USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, ser. EVT'06. USENIX Association, 2006.

[19] P. B. Rønne, P. Y. Ryan, and B. Smyth, "Cast-as-intended: A formal definition and case studies," in *Financial Cryptography and Data Security. FC 2021 International Workshops: CoDecFin, DeFi, VOTING, and WTSC 2021, Revised Selected Papers 25*. Springer, 2021.

[20] V. Cortier, D. Galindo, S. Glondu, and M. Izabachene, "Election verifiability for helios under weaker trust assumptions," in *Computer Security-ESORICS 2014: 19th European Symposium on Research in Computer Security Proceedings*. Springer, 2014.

[21] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, 1985.

[22] M. Hirt, "Multi party computation: Efficient protocols, general adversaries, and voting," PhD thesis – Diss. ETH No. 14376, 2001.

[23] J. Groth and A. Sahai, "Efficient non-interactive proof systems for bilinear groups," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2008.

[24] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn, "Malleable proof systems and applications," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012.

[25] G. Fuchsbauer, "Commuting signatures and verifiable encryption and an application to non-interactively delegatable credentials," *Cryptology ePrint Archive*, 2010.

[26] E. McMurtry, O. Pereira, and V. Teague, "When is a test not a proof?" in *Computer Security - ESORICS 2020 - 25th European Symposium on Research in Computer Security*, ser. LNCS, vol. 12309. Springer, 2020.

[27] M. Jakobsson and A. Juels, "Mix and match: Secure function evaluation via ciphertexts," in *Advances in Cryptology—ASIACRYPT 2000: 6th International Conference on the Theory and Application of Cryptology and Information Security Kyoto, Japan, December 3–7, 2000 Proceedings 6*. Springer, 2000, pp. 162–177.

[28] H. Devillez, "Receipt-free verfiable protocols benchmarking," 2023. [Online]. Available: https://github.com/VerifiableRF/verifiable_rf_benchmarks

[29] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: a framework for rapidly prototyping cryptosystems," *Journal of Cryptographic Engineering*, vol. 3, no. 2, 2013.

[30] D. Bernhard, O. Pereira, and B. Warinschi, "How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios," in *Advances in Cryptology–ASIACRYPT 2012*. Springer, 2012.

Appendix

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

1. Summary

The paper provides a formal treatment of the relation between receipt-freeness and cast-as-intended voting. The main focus is understanding receipt-freeness and its relation to verifiability, both of which are significant to voting systems (and other systems). In this vein, an impossibility result is presented, which –in approach– is interesting beyond e-voting. The paper additionally advances the state of the art by proposing three new voting systems.

2. Scientific Contributions

- A Valuable Step Forward in an Established Field
- Creates a New Tool to Enable Future Science

3. Reasons for Acceptance

1) The paper's results are sound and interesting. It considers past e-voting in terms of formal properties and reasoning about their inter-relations.
2) The paper will be of interest to the secure e-voting community, which is active and has broader impact.
3) In general, the impossibility result is interesting per se, and the protocols proposed can advance the field of e-voting with new ideas.