Threshold Receipt-Free Single-Pass eVoting

Thi Van Thao Doan[1], Olivier Pereira[1,2], and Thomas Peters[1]

[1] Université catholique de Louvain
ICTEAM – Crypto Group
B-1348 Louvain-la-Neuve - Belgium
[2] Microsoft Research, Redmond, WA, USA
{thi.doan, olivier.pereira, thomas.peters}@uclouvain.be

Abstract. In 2001, Hirt proposed a receipt-free voting scheme, which prevents malicious voters from proving to anybody how they voted, under the assumption of the availability of a helping server that is trusted for receipt-freeness, and only for that property. This appealing design led to a number of subsequent works that made this approach non-interactive and more efficient. Still, in all of these works, receipt-freeness depends on the honesty of one single server.
In order to remove this single point of failure, we design a new model in which multiple helping servers are available and propose a new security definition called *threshold receipt-freeness*. Our definition requires that receipt-freeness should be guaranteed even if some of the helping servers happen to be fully malicious and ensures that voters can express their votes even if the corrupted servers choose the content of their local view of the ballots.
Eventually, we propose a generic construction of a single-pass verifiable voting system achieving threshold receipt freenes with a mixnet-based tallying process. Our ballot submission process relies on the recently designed traceable receipt-free encryption primitive.

1 Introduction

Electronic voting (eVoting) systems allow voters to generate ballots containing their vote intent, and submit these ballots to an electronic server. When all the ballots have been recorded, the talliers compute the result of the election and, hopefully, provide information allowing to check the validity of the tallying process. This verification mechanism ensures voters that their individual ballots have been faithfully recorded and counted in the tally. However, the information supporting verifiability can often be leveraged by malicious observers to demand voters to convincingly demonstrate how they voted, e.g., by disclosing all the random coins used to produce their ballots. eVoting systems preventing this kind of attacks are called receipt-free.

1.1 Receipt-Freeness

Since the introduction of Receipt-Freeness (RF) by Benaloh and Tuinstra [1], there has been a large amount of work targeting a well-balanced combination

of receipt-freeness and verifiability. One such approach requires the use of fully trusted devices during the election [10,11,13]. Hirt [9] refined this approach by proposing a scheme in which every voter must have their encrypted vote re-randomized by a ballot processing server, which must be trusted for receipt-freeness. The re-randomized ballot is then posted on a public bulletin board. Since the randomness contained in that ballot is no longer known by the voters, the voters cannot prove how they voted. This results in receipt-freeness. The ballot processing server of Hirt [9] serves as an observer [5] establishing receipt-freeness, but cannot violate the privacy of the votes or the correctness of the tally. However, the server must interact with the voter to confirm the correctness of the randomization, and the observer and the voter must jointly generate a proof of the validity of the randomized ballot.

To avoid such interactions between parties, the more recent line of work [3,4,7,8] has designed mechanisms that make it possible to achieve non-interactive receipt-freeness. A voter generates a signed encrypted vote in a manner that allows the ciphertext and signature to be re-randomized by any individual, knowing neither the signing key nor the encrypted message, but the re-randomized signature will only be valid as long as the plaintext has not been modified. This creates a single-pass voting system [2] in which the ballot is sent from the voter, processed by a ballot processing server, and then posted on the bulletin board.

An encryption primitive designed to capture the desired form of non-interactive receipt-freeness was named traceable receipt-free encryption (TREnc) by Devillez et al. [6], who propose a mechanism realizing this primitive based on the signature of randomizable ciphertext with a one-time linearly homomorphic signature scheme, a primitive proposed by Libert et al. [12].

Yet, it is worth noting that even though the ballot processing server is not trusted for ballot privacy and election integrity, all of the aforementioned proposals rely on the assumption that the ballot processing server to which the voter submits his vote is trusted for RF, i.e., it does not collaborate with any vote-seller or coercer. This maintains a single point of failure. For instance, if the random coins utilized for re-randomizing ballots are transmitted to malicious voters, then the voters can demonstrate how they voted to any third party.

1.2 Our contributions

We design an approach that removes the single point of failure of the existing receipt-free voting solutions. We first extend the traditional voting models to account for the involvement of *multiple* ballot processing servers. More precisely, the voting algorithm creates a ballot that can be split into n ballot pieces. Each ballot piece is processed in parallel by a ballot processing server before being gathered at a single place, usually a public bulletin board. Second, based on this enlarged syntax, we propose a security notion of *threshold receipt freeness*, and define a related correctness security notion. These security definitions capture the property that the voting system will accurately record the voter intent, while guaranteeing receipt-freeness as long as the number of malicious servers remains below the chosen threshold.

More precisely, we enhance the security model by introducing two thresholds: (1) the threshold for receipt-freeness ($\mathsf{t_{rf}}$), which sets the maximum number of malicious ballot processing servers that the system can tolerate without compromising RF; and (2) the threshold for correctness ($\mathsf{t_{corr}}$), which specifies the maximum number of ballot pieces that can be missing (intentionally or not) while still allowing the ballot to be processed in the tally. To reflect the fact that voters must be able to cast their intentions vote as in the single ballot processing model when $\mathsf{t_{rf}} = 0$, our threshold RF notions is defined in two steps. The first step ensures that as long as at most $\mathsf{t_{rf}}$ ballot processing servers fix the content of any voter's incoming ballot pieces, the voter can still cast a ballot containing any vote intent. The second step is a natural RF extension of the indistinguishable notion due to Devillez et al. [6].

Eventually, as a feasibility result, we build a generic voting system from a linear secret sharing scheme and a TREnc. In a nutshell, the voter simply relies on a t-out-of-n threshold secret sharing to split the vote in n pieces, and on a TREnc to encrypt all these pieces independently. As long as an homomorphic part of the TREnc ciphertexts can be stripped from the published randomized ballot pieces, the recombination of the vote can be made on the bulletin board during the tally before executing a mixnet. We show that this construction is threshold RF up to $\mathsf{t_{rf}} = t - 1$ malicious servers and threshold correct up to $\mathsf{t_{corr}} = n - t$ missing ballot pieces.

Roadmap. The remainder of this paper is organized as follows. The background is provided in Section 2. In Section 3, we extend the traditional model of voting systems to the context of multiple ballot processing servers, and define out notions of threshold receipt-freeness and correctness. Section 4 describes our generic construction, and we demonstrate the security of this construction in Section 5. We conclude in section 6.

2 Background on Traceable Receipt-Free Encryption

We start by reminding the notion of Traceable Receipt-Free Encryption (TREnc), a public key encryption primitive supporting the receipt-free submission of secret ballots [6].

Definition 1 (TREnc). *A* TREnc *is a public key encryption* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *that is augmented with a 5-tuple of algorithms* $(\mathsf{LGen}, \mathsf{LEnc}, \mathsf{Trace}, \mathsf{Rand}, \mathsf{Ver})$*:*

- $\mathsf{LGen}(\mathsf{pk})$*: The link generation algorithm takes as input a public encryption key* pk *in the range of* Gen *and outputs a link key* lk*.*
- $\mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, m)$*: The linked encryption algorithm takes as input a pair of public/link keys* $(\mathsf{pk}, \mathsf{lk})$ *and a message* m*, outputs a ciphertext.*
- $\mathsf{Trace}(\mathsf{pk}, c)$ *: The tracing algorithm takes as input a public key* pk*, a ciphertext* c *and outputs a trace* τ*. We call* τ *the trace of* c*.*
- $\mathsf{Rand}(\mathsf{pk}, c)$*: The randomization algorithm takes as input a public key* pk *and a ciphertext* c*, outputs another ciphertext.*

$\mathsf{Ver}(\mathsf{pk}, c)$*: The verification algorithm takes as input a public key* pk*, a ciphertext* c *and outputs* 1 *if the ciphertext is valid,* 0 *otherwise.*

Definition 2 (TREnc **correctness).** *A* TREnc *scheme is required to satisfy the following correctness requirements:*
Encryption compatibility: *For every* pk *in the range of* Gen *and message* m*, the distributions of* $\mathsf{Enc}(\mathsf{pk}, m)$ *and* $\mathsf{LEnc}(\mathsf{pk}, \mathsf{LGen}(\mathsf{pk}), m)$ *are identical;*
Link traceability: *For every* pk *in the range of* Gen*, every* lk *in the range of* $\mathsf{LGen}(\mathsf{pk})$*, the encryptions of every pair of messages* (m_0, m_1) *trace to the same trace, that is,* $\mathsf{Trace}(\mathsf{pk}, \mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, m_0)) = \mathsf{Trace}(\mathsf{pk}, \mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, m_1))$*, always;*
Publicly Traceable Randomization: *For every* pk *in the range of* Gen*, every message* m *and every* c *in the range of* $\mathsf{Enc}(\mathsf{pk}, m)$*, we have that* $\mathsf{Dec}(\mathsf{sk}, c) = \mathsf{Dec}(\mathsf{sk}, \mathsf{Rand}(\mathsf{pk}, c))$ *and* $\mathsf{Trace}(\mathsf{pk}, c) = \mathsf{Trace}(\mathsf{pk}, \mathsf{Rand}(\mathsf{pk}, c))$*;*
Honest verifiability: *For every* pk *in the range of* Gen *and every messages* m*, it holds that* $\mathsf{Ver}(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m)) = 1$*.*

A voter encrypts his vote m by picking a link key lk using LGen and computing LEnc to generate a ciphertext c, which is then submitted to a ballot processing server. The server runs Rand to produce a re-randomized ciphertext c', which is included in the tally as long as $\mathsf{Ver}(\mathsf{pk}, c') = 1$. The correctness ensures that the voter's intent is accurately reflected in the resulting ciphertext. The voter can also store the trace of the ciphertext $\mathsf{Trace}(\mathsf{pk}, c)$, which is equal to $\mathsf{Trace}(\mathsf{pk}, c')$, and confirms that it has been correctly recorded on the public bulletin board.

The *traceability* notion ensures that no corrupt authority would be able to modify a ciphertext in a way that modifies the plaintext without modifying the trace at the same time. Moreover, the trace cannot serve as a receipt for an encrypted message, since the trace and the message are independent, as guaranteed by the link traceability correctness. The traceability guarantees that, if the voter only produced a single ciphertext with a given trace, then any other ciphertext with the same trace will be an encryption of the same plaintext.

Definition 3 (Traceability). *A* TREnc *is* traceable *if for every* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the experiment* $\mathsf{Exp}^{\mathrm{trace}}_{\mathcal{A}}(\lambda)$ *defined in Figure 1 (right) returns* 1 *with a probability negligible in* λ*.*

While the traceability relates to a model in which the voter is honest but the ballot processing server might be corrupted, the *traceable-CCA* notion (TCCA) focuses on protecting the privacy of the vote against a malicious voter. This is achieved through an indistinguishable experiment that guarantees that any malicious ballots computed with an identical trace cannot be recognized after randomization. This essentially guarantees the absence of a vote receipt.

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

Homomorphic part. In this paper we assume that it is easy to strip an homomorphic part of any TREnc ciphertext that carries the encrypted vote and allows decryption. The constructions of[6] satisfy this condition.

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

$(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$
$(c_0, c_1, \mathsf{st}) \leftarrow\!\!\$\ \mathcal{A}_1^{\mathsf{Dec}(\cdot)}(\mathsf{pk})$
$b \leftarrow\!\!\$\ \{0,1\}$
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 $b' = b$

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

$(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$
$(m, \mathsf{st}) \leftarrow\!\!\$\ \mathcal{A}_1(\mathsf{pk}, \mathsf{sk})$
$c \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pk}, m)$
$c^\star \leftarrow\!\!\$\ \mathcal{A}_2(c, \mathsf{st})$
if $\mathsf{Trace}(\mathsf{pk}, c) = \mathsf{Trace}(\mathsf{pk}, c^\star)$ and
$\mathsf{Ver}(\mathsf{pk}, c^\star) = 1$ and $\mathsf{Dec}(\mathsf{sk}, c^\star) \neq m$
then return 1
else return 0

Fig. 1. Security experiments. In the case of TCCA, $\mathcal{A}_2$ has access to an 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 $\perp$ otherwise.

3 Voting scheme with multiple ballot processing severs

We adapt the general e-voting definition of [6] to cope with multiple ballot processing servers, and formalize threshold receipt-freeness and correctness.

3.1 Definitions and notations

The election adminstrator (EA) is in charge of generating the keys of the system by running SetupElection. Given the public parameter of the system, voters can cast their intentions through Vote which creates a ballot $\mathsf{b} = \{\mathsf{b}_i\}_{i=1}^n$. Each ballot piece b_i is processed by the i-th ballot processing server which simply runs the algorithm ProcessBallot. The other algorithms are unchanged.

Definition 5 (Voting System). *A voting system equipped with n ballot processing servers is a tuple of PPT algorithms* (SetupElection, Vote, ProcessBallot, TraceBallot, Valid, Append, Publish, VerifyVote, Tally, VerifyResult) *associated to a result function $\rho_m : \mathbb{V}^m \cup \{\perp\} \to \mathbb{R}$ where $\mathbb{V}$ is the set of valid votes and $\mathbb{R}$ is the result space such that:*

- $\mathsf{SetupElection}(1^\lambda)$*: on input security parameter 1^λ, generate the public and secret keys* $(\mathsf{pk}, \mathsf{sk})$ *of the election. Below,* pk *is an implicit input.*
- $\mathsf{Vote}(\mathsf{id}, \mathsf{v})$*: when receiving a voter* id *and a vote* v*, output a ballot* $\mathsf{b} = \{\mathsf{b}_i\}_{i=1}^n$ *and auxiliary data* aux*. Given* aux*, running* $\mathsf{Vote}(\mathsf{id}, \mathsf{v}, \mathsf{aux})$ *allows computing ballots that can be related (in our case, ciphertexts with the same traces).* $\mathsf{Vote}(\mathsf{id}, \mathsf{v}; \rho)$ *denotes the deterministic computation of the algorithm given random coin ρ.*
- $\mathsf{ProcessBallot}(\mathsf{b}_i)$*: on input a ballot share* b_i*, output an updated ballot share* b'_i*.*
- $\mathsf{TraceBallot}(\mathsf{b})$*: on input ballot* b*, outputs a tag τ. The tag is the information that a voter can use to trace his ballot, using* VerifyVote*.*
- $\mathsf{Valid}(\mathsf{BB}, \mathsf{b})$*: on input ballot box* BB *and ballot* b*, outputs* 0 *or* 1*. The algorithm outputs* 1 *if and only if the ballot is valid.*
- $\mathsf{Append}(\mathsf{BB}, \mathsf{b})$*: on input ballot box* BB *and ballot* b*, appends* b *to* BB *if* $\mathsf{Valid}(\mathsf{BB}, \mathsf{b}) = 1$*.*

$\mathsf{Publish}(\mathsf{BB})$*: on input ballot box* BB*, outputs the public view* PBB *of* BB*, which is the one that is used to verify the election. Depending on the context, it may be used to remove some voter credentials for instance.*

$\mathsf{VerifyVote}(\mathsf{PBB}, \tau)$*: on input public ballot box* PBB *and tag* τ*, outputs* 0 *or* 1*. This algorithm is used by voters to check if their ballot has been processed and recorded properly.*

$\mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$*: on input ballot box* BB *and private key of the election* sk*, outputs the tally* r *and a proof* Π *that the tally is correct w.r.t. the result function* ρ_m*.*

$\mathsf{VerifyResult}(\mathsf{PBB}, \mathsf{r}, \Pi)$*: on input public ballot box* PBB*, result of the tally* r *and proof of the tally* Π*, outputs* 0 *or* 1*. The algorithm outputs* 1 *only if* Π *is a valid proof that* r *is the correct election result.*

When voters cast their votes, they record the tracking code computed from $\mathsf{TraceBallot}$ to trace their processed ballots on the public bulletin board PBB. Processed ballot shares that originate from the same voter are gathered and pushed to the ballot box BB, using Append. Recombined processed ballots are made publicly available on PBB by running $\mathsf{Publish}$. Voters can verify that their ballots have been correctly recorded on PBB by relying on $\mathsf{VerifyVote}$ and their tracking codes. Other algorithms are as usual. If we write $\mathsf{ProcessBallot}(\mathsf{b}) = \{\mathsf{ProcessBallot}(\mathsf{b}_i)\}_{i=1}^{n}$, it is easy to see that our syntax is indeed a generalization of [6] which corresponds to the particular case of $n = 1$.

3.2 Threshold receipt-freeness

We formalize receipt-freeness in the multi-ballot processing model. Intuitively, the receipt-free threshold $\mathsf{t_{rf}}$ is the maximum number of malicious ballot processing servers tolerated by the system to guarantee receipt freeness.

Definition 6 (Threshold receipt-freeness). *A voting system* $\mathcal{V}$ *with* n *ballot processing severs has receipt-freeness with threshold* $\mathsf{t_{rf}} \leq n$ *if*

1. *There exists an algorithm* $\mathsf{Deceive}$ *such that, for every PPT adversary* $\mathcal{A}$*,* $\Pr[\mathsf{Exp}^{\mathsf{deceive}}_{\mathcal{A},\mathcal{V},\mathsf{t_{rf}}}(\lambda) = 1]$ *negligible in* λ*. (The experiment is defined in Figure 2.)*
2. *There exist algorithms* $\mathsf{SimSetupElection}$ *and* $\mathsf{SimProof}$ *such that, for every* PPT *adversary* $\mathcal{A}$*, the following advantage is negligible in* λ

$$\mathsf{Adv}^{\mathsf{rf},\mathsf{t_{rf}}}_{\mathcal{A},\mathcal{V}}(1^\lambda) = \left|\Pr\left[\mathsf{Exp}^{\mathsf{rf},\mathsf{t_{rf}},0}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{rf},\mathsf{t_{rf}},1}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right]\right|,$$

where the experiment $\mathsf{Exp}^{\mathsf{rf},\mathsf{t_{rf}},\beta}_{\mathcal{A},\mathcal{V}}(\lambda)$ *is defined in Figure 3.*

The experiment $\mathsf{Exp}^{\mathsf{deceive}}_{\mathcal{A},\mathcal{V},\mathsf{t_{rf}}}(\lambda)$**.** The setup of the election creates the public and secret keys $(\mathsf{pk}, \mathsf{sk})$ and pk is given to $\mathcal{A}$. Then $\mathcal{A}$ chooses at most $\mathsf{t_{rf}}$ indexes of the corrupt ballot processing servers I. It also chooses the votes v_0, v_1 and the random coin ρ in the hope that ρ is a receipt ensuring that the computed ballot b contains v_0. However, $\mathcal{A}$ only sees the at most $\mathsf{t_{rf}}$ ballot pieces $(\mathsf{b}_i)_{i\in I}$ and the public tracking codes for which the $\mathsf{Deceive}$ algorithm allows computing the complement ballot pieces so that b is a valid vote for v_1. (See Figure 2.)

$\mathsf{Exp}^{\mathsf{deceive},\mathsf{t_{rf}}}_{\mathcal{A},\mathcal{V}}(\lambda)$

$(\mathsf{pk},\mathsf{sk}) \leftarrow\!\$\ \mathsf{SetupElection}(1^\lambda)$
$(v_0, v_1, \rho, I) \leftarrow\!\$\ \mathcal{A}(\mathsf{pk})$
if $I \not\subset [n]$ or $|I| > \mathsf{t_{rf}}$ **then return** 0
$\mathsf{b}_0 \leftarrow \mathsf{Vote}(\mathsf{id}, v_0, \rho)$
$\mathsf{b}_1 \leftarrow \mathsf{Deceive}(\mathsf{id}, v_0, v_1, \rho, I)$
if $\{\mathsf{b}_0^i\}_{i\in I} \neq \{\mathsf{b}_1^i\}_{i\in I}$ or $\mathsf{b}_1 \notin \mathsf{Vote}(\mathsf{id}, v_1)$
or $\mathsf{TraceBallot}(\mathsf{b}_0) \neq \mathsf{TraceBallot}(\mathsf{b}_1)$
then return 1
return 0

Fig. 2. Deceive experiment.

The experiment $\mathsf{Exp}^{\mathsf{rf},\mathsf{t_{rf}},\beta}_{\mathcal{A},\mathcal{V}}(\lambda)$**.** The experiment given in Figure 3 is parameterized by a bit β, and the adversary has access to the following oracles:

- $\mathcal{O}$init: Is called a single time and generates the secret and public keys for the election. The public key is shared with the adversary. When $\beta = 1$, depending on the computational model, a simulated setup may be performed. This setup provides trapdoor information that can be used to produce a simulated tally correctness proof. Two empty ballot boxes BB_0 and BB_1 are initialized. The adversary will only have access to $\mathsf{Publish}(\mathsf{BB}_\beta)$ that is updated throughout the experiment.
- $\mathcal{O}$receiptLR: Allows the adversary to cast valid ballots and query the honest ballot processing servers to process their respective ballot pieces so that, on input $(\mathsf{id}, \mathsf{B}_0, \mathsf{B}_1, \mathsf{B}_2)$ for voter id, the oracle runs ProcessBallot on both sets B_0 and B_1 of valid ballot pieces if they share the same traces for the same index and gets B'_0 and B'_1. As long as $|B_0 \cup B_2| = |B_1 \cup B_2| \leq n$ and $|B_2| \leq \mathsf{t_{rf}}$, $\mathsf{b}_0 = \mathsf{B}'_0||\mathsf{B}_2$ is appended to BB_0 and $\mathsf{b}_1 = \mathsf{B}'_1||\mathsf{B}_2$ is appended to BB_1. Up to reordering, we can always assume that the first servers are honest. B_2 represents the ballot pieces for which the malicious vote seller and the corrupt servers together know their whole content.
- $\mathcal{O}$board: Returns the view $\mathsf{Publish}(\mathsf{BB}_\beta)$ of the public bulletin board.
- $\mathcal{O}$tally: Allows the adversary to see the result of the election obtained by tallying valid BB_0, as well as a proof of correctness of the tally. If $\beta = 1$, this proof is simulated with respect to the content derived from BB_1.

The adversary first calls $\mathcal{O}$init. Following this, it can call the oracles $\mathcal{O}$receiptLR and $\mathcal{O}$board in any order and any number of times. Finally, the adversary calls the oracle $\mathcal{O}$tally to receive the (simulated) result of the election. It must then return its guess of the bit β, which is the output of the experiment. If $n = 1$ and $\mathsf{t_{rf}} = 0$, we get back the receipt freeness definition of [6].

$\mathcal{O}\mathsf{init}(\lambda)$

if $\beta = 0$ **then** $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SetupElection}(1^\lambda)$
else $(\mathsf{pk}, \mathsf{sk}, \tau) \leftarrow \mathsf{SimSetupElection}(1^\lambda)$
$\mathsf{BB}_0 \leftarrow \bot; \mathsf{BB}_1 \leftarrow \bot$
return pk

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

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

$\mathcal{O}\mathsf{receiptLR}(\mathsf{id}, \mathsf{B}_0, \mathsf{B}_1, \mathsf{B}_2)$

if $|B_0| \neq |B_1|$ **or** $|B_1| + |B_2| > n$ **or** $|B_2| > \mathsf{t_{rf}}$
then return $\bot$
if $\mathsf{TraceBallot}(\mathsf{B}_0||\mathsf{B}_2) \neq \mathsf{TraceBallot}(\mathsf{B}_1||\mathsf{B}_2)$ or
$\mathsf{Valid}(\mathsf{B}_0||\mathsf{B}_2) = 0$ or $\mathsf{Valid}(\mathsf{B}_1||\mathsf{B}_2) = 0$
then return $\bot$
else $\mathsf{Append}(\mathsf{BB}_0, \mathsf{ProcessBallot}(\mathsf{B}_0)||\mathsf{B}_2)$ and
$\mathsf{Append}(\mathsf{BB}_1, \mathsf{ProcessBallot}(\mathsf{B}_1)||\mathsf{B}_2)$

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

return $\mathsf{Publish}(\mathsf{BB}_\beta)$

Fig. 3. Threshold receipt freeness oracles from experiment $\mathsf{Exp}^{\mathsf{rf},\mathsf{t_{rf}},\beta}_{\mathcal{A},\mathcal{V}}(\lambda)$, for $\beta = 0, 1$.

3.3 Threshold correctness

We formalize correctness in the multi-ballot processing model. Intuitively, the correctness threshold $\mathsf{t_{corr}}$ is the maximum number of ballot pieces of any ballot that could not reach the bulletin board while still allowing to include the corresponding underlying voter's intention in the tally.

Definition 7 (Theshold correctness). *A voting system $\mathcal{V}$ with n ballot processing servers satisfies correctness with threshold $\mathsf{t_{corr}} < n$ if for every PPT adversary $\mathcal{A}$, $\Pr[\mathsf{Exp}^{\mathsf{corr},\mathsf{t_{corr}}}_{\mathcal{A},\mathcal{V}}(\lambda) = 1]$ is negligible in λ. (The experiment is defined in Figure 4.)*

The experiment $\mathsf{Exp}^{\mathsf{corr},\mathsf{t_{corr}}}_{\mathcal{A},\mathcal{V}}(\lambda)$. The experiment is given in Figure 4, and the adversary is given access to the following oracles:

- $\mathcal{O}\mathsf{init}$: Generates and returns both the secret and public keys of the election.
- $\mathcal{O}\mathsf{vote}$: Takes a potential vote v for a user id, honestly produces and outputs a ballot b using Vote. The tracking code $\mathsf{TraceBallot}_\mathsf{b}$ is stored in the list $\mathcal{L}$. This represents ballots and tracing information from honest voters.
- $\mathcal{O}\mathsf{append}$: Allows the adversary to select two valid ballots b_0 and b_1 of at least $n - \mathsf{t_{corr}}$ valid ballot pieces as if there were potentially maliciously processed from honest ballots output by $\mathcal{O}\mathsf{vote}$. That is, both ballots must have their respective sets of tracking codes included in a single set of codes from $\mathcal{L}$. Both ballots are then respectively appended to BB_0 and BB_1. It represents maliciously processed ballot pieces that use the same tracing information as honest ballots but for which the adversary tries to change the content of the votes and blocks at most $\mathsf{t_{corr}}$ pieces of them.
- $\mathcal{O}\mathsf{cast}$: Allows the adversary to cast a malicious valid ballot b in BB_0 and BB_1.
- $\mathcal{O}\mathsf{board}$: Returns the views $\mathsf{Publish}(\mathsf{BB}_0)$ and $\mathsf{Publish}(\mathsf{BB}_1)$ of the public bulletin boards.
- $\mathcal{O}\mathsf{tally}$: Allows the adversary to get the results of the election obtained by honestly tallying BB_0 and BB_1.

The adversary $\mathcal{A}$ initiates the experiment by invoking the oracle $\mathcal{O}\mathsf{init}$, and the experiment terminates when it calls $\mathcal{O}\mathsf{tally}$. The output of the experiment is a bit that is equal to 1 only if the adversary managed to compute valid ballots providing distinct outcomes from both ballot boxes. To attempt creating such ballots, the adversary can query all the other oracles many times in many orders.

$\mathcal{O}\mathsf{init}(\lambda)$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SetupElection}(1^\lambda)$
$\mathsf{BB}_0 \leftarrow \bot; \mathsf{BB}_1 \leftarrow \bot; \mathcal{L} \leftarrow \bot$
return $(\mathsf{pk}, \mathsf{sk})$

$\mathcal{O}\mathsf{cast}(\mathsf{id}, \mathsf{b})$

if $\mathsf{Valid}(\mathsf{b}) = 0$ or $|\mathsf{b}| < n - \mathsf{t}_{\mathsf{corr}}$
then return $\bot$
else $\mathsf{Append}(\mathsf{BB}_0, \mathsf{b}); \mathsf{Append}(\mathsf{BB}_1, \mathsf{b})$

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

$(\mathsf{r}_0, \Pi_0) \leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathsf{sk})$
$(\mathsf{r}_1, \Pi_1) \leftarrow \mathsf{Tally}(\mathsf{BB}_1, \mathsf{sk})$
return $(\mathsf{r}_0, \mathsf{r}_1)$

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

$\mathsf{b} = \mathsf{Vote}(\mathsf{id}, \mathsf{v})$
$\mathcal{L} \xleftarrow{\cup} \{\mathsf{TraceBallot}(\mathsf{b})\}$
return b

$\mathcal{O}\mathsf{append}(\mathsf{b}_0, \mathsf{b}_1)$

if $\nexists T \in \mathcal{L} : \mathsf{TraceBallot}(\mathsf{b}_i) \subset T$, **for** $i = 0, 1$
or $\mathsf{Valid}(\mathsf{b}_0) = 0$ **or** $\mathsf{Valid}(\mathsf{b}_1) = 0$
or $|b_0| < n - \mathsf{t}_{\mathsf{corr}}$ **or** $|b_1| < n - \mathsf{t}_{\mathsf{corr}}$
then return $\bot$
$\mathsf{Append}(\mathsf{BB}_0, \mathsf{b}_0)$; $\mathsf{Append}(\mathsf{BB}_1, \mathsf{b}_1)$

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

return $\mathsf{Publish}(\mathsf{BB}_0); \mathsf{Publish}(\mathsf{BB}_1)$

Fig. 4. Threshold correctness experiment $\mathsf{Exp}^{\mathsf{corr},\mathsf{t}_{\mathsf{corr}}}_{\mathcal{A},\mathcal{V}}(\lambda)$ which outputs 1 only if $\mathsf{r}_0 \neq \mathsf{r}_1$.

Remarks. We leave it open to define a stronger threshold correctness notion where the adversary would remain unable to modify the result of the election by selecting distinct ballot pieces from a maliciously generated ballot. Here malicious ballots are equally processed on both ballot boxes. In our construction, this is not an issue as we implicitly define the vote as the one contained in the recombination of all the stripped ciphertexts included in the ballot pieces of each voter. The recombined ciphertexts are then tallied based on a mixnet. Achieving the stronger notion in a construction with an homomorphic tally requires designing new malleable proofs that can be adapted through all the randomness used by the n servers. Indeed, if the vote v is represented with a bit, the randomizable shared proof that $v(1 - v) = 0$ is hardly adaptable locally by the n servers as their adaptation must depend on the others' randomizing factors.

4 Our construction based on TREnc

Our voting system is based on a TREnc. As recalled in Section 1, a TREnc consists of various algorithms including Gen, Enc, Trace, Rand, Ver, and Dec in particular. We will use these algorithms to demonstrate how to design a voting system that can provide the properties of threshold receipt-freeness and correctness as previously described in Sections 3.2 and 3.3.

The message (or the vote) v is divided into n message shares using a t-out-of-n threshold secret sharing scheme, originally proposed by Shamir [14], where at least t message pieces must be combined to reconstruct the message. Moreover, any $t-1$ pieces remain independent of v. To generate the ballot b, the voter encrypts each share of his message by calling the Enc function of TREnc, which produces a ballot share b_i. The process is repeated n times for n message shares, resulting in the creation of a full ballot consisting of n ballot shares, i.e., $\mathsf{b} = \{\mathsf{b}_i\}_{i=1}^{n}$. Each share is processed by a specific ballot processing server. For instance, server i exclusively receives the ballot share b_i, and does not have access to any other share b_j where $j \neq i$. The instantiation of our voting scheme is illustrated in Figure 5, where n is the implicit input of all the algorithms.

SetupElection (λ)

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{TREnc.Gen}(1^{\lambda})$
return pk

Vote(id, v[, aux])

$M = \mathsf{Share}(\mathsf{pk}, t, \mathsf{v}) = \{x_1, \ldots, x_n\}$
if aux is empty **for** $i = 1$ to n **do**
  $\mathsf{lk}_i \leftarrow\$ \mathsf{TREnc.LGen}(\mathsf{pk})$
else $\{\mathsf{lk}_i\}_{i=1}^{n} \leftarrow \mathsf{aux}$
for $i = 1$ to n **do**
  $\mathsf{b}_i \leftarrow \mathsf{TREnc.LEnc}(\mathsf{pk}, \mathsf{lk}_i, x_i)$
return $\mathsf{b} = \{\mathsf{b}_i\}_{i=1}^{n}, \mathsf{aux} = \mathsf{lk} = \{\mathsf{lk}_i\}_{i=1}^{n}$

TraceBallot(b)

return $\{\mathsf{TREnc.Trace}(\mathsf{b}_i)\}_{i=1}^{|\mathsf{b}|}$

ProcessBallot(b_i)

return $\mathsf{TREnc.Rand}(\mathsf{pk}, \mathsf{b}_i)$

Valid(BB, b)

if $\exists \mathsf{b}' \in \mathsf{BB} : \mathsf{TraceBallot}(\mathsf{b}') = \mathsf{TraceBallot}(\mathsf{b})$
then return $\perp$
$k = 0$
for $i = 1$ to $|\mathsf{b}|$ do
  if $\mathsf{TREnc.Ver}(\mathsf{b}_i) = 1$
    $k \leftarrow k + 1$
if $k \geq t$ **then return** 1 **else return** 0

VerifyVote(PBB, τ)

if $\exists \mathsf{b} \in \mathsf{PBB} : \mathsf{Valid}(\mathsf{b}) \wedge \tau == \mathsf{TraceBallot}(\mathsf{b})$
then return 1 **else return** 0

Fig. 5. Instantiation of our voting scheme.

First, EA runs the $\mathsf{SetupElection}$ algorithm to generate the public and secret keys of the election. This is achieved by running the key generation algorithm from TREnc, denoted by $\mathsf{TREnc.Gen}$. The public key pk is published and stored on the PBB, while sk is only known by the talliers (sk can be securely generated in a distributed way in our prime-order groups using standard techniques). For the sake of simplicity, we consider a model with a single tallier.

When a voter wants to cast a vote v, he prepares a ballot using the Vote algorithm. First, the Share is executed to implement the t-out-of-n threshold secret-sharing scheme. This function takes as input the number of ballot processing parties n, the threshold t, and the secret v to output the shares $M = \{x_1, \ldots, x_n\}$. Then, it calls the encryption function Enc of TREnc (which includes $\mathsf{LGen}(\mathsf{pk}) = \mathsf{lk}$ and $\mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, \mathsf{v})$) on each message share x_i to produce a ballot share b_i for $i = 1$ to n. In the end, Vote returns a ballot b and $\mathsf{aux} = \mathsf{lk}$.

Each b_i is then submitted to a distinct ballot processing server, while the full trace $\tau = \mathsf{TraceBallot}(\mathsf{b})$ is sent directly to the PBB. We have that $\tau = \{\tau_i\}_{i=1}^{|\mathsf{b}|}$ including all traces of available ballot shares in b, where $\tau_i = \mathsf{TREnc.Trace}(\mathsf{b}_i)$.

When a server receives a b_i, it extracts its trace using $\mathsf{TREnc.Trace}(\mathsf{b}_i)$ and verifies that no shares with the same trace were recorded before. If b_i is valid, i.e., $\mathsf{TREnc.Ver}(\mathsf{b}_i) = 1$, it will be re-randomized with the help of $\mathsf{ProcessBallot}(\mathsf{b}_i) = \mathsf{TREnc.Rand}(\mathsf{b}_i)$, while invalid shares are discarded. Although each randomizer processes only one share of each voter at a time, the ProcessBallot algorithm can be executed in parallel for n number of randomizers. The resulting ballot share is then made available on the BB.

The tallying process commences with the gathering of ballot shares from the same ballot. To do this, the tallier compares traces received on BB to the full trace τ posted by each voter on the public board earlier. Subsequently, the validity of each ballot is verified using the function TREnc.Ver on each ballot share. The function $\mathsf{Valid}(\mathsf{BB}, \mathsf{b})$ in Figure 5 returns 1 if there are at least t valid ballot shares of b on the public board, and 0 otherwise. Consequently, the combination algorithm Combine takes as input the homomorphic parts of all valid ballot shares available, which may be at most n, and outputs the combined encryption of the original message v. The resulting (CPA-secure) ciphertext contains the reconstruction of the vote from the encrypted shares. According to the principles of Lagrange polynomial interpolation, the final message remains unaltered even when more than t valid shares are combined.

The validity of the vote The purpose of the Valid function is to ensure that the input of the tally is valid. However, this function does not guarantee that the encrypted vote is within the range of the vote space. To address this issue, a verifiable *mixnet* will be added after the combination process is completed, which disassociates the ballots from their corresponding voters. This anonymization process will be carried out by shuffling the resulting combined encryptions through multiple shuffling centers, after which they will be decrypted individually. Invalid votes are discarded. The tallier returns the result of the election r along with proof of its correctness Π, which can be verified by anyone with VerifyResult. Voters also can verify the presence of their vote on the bulletin board by utilizing $\mathsf{VerifyVote}(\mathsf{PBB}, \tau)$ to confirm if any entry in PBB matches τ.

Correlation between t_{rf} *and* $\mathsf{t}_{\mathsf{corr}}$. In accordance with the threshold secret sharing scheme used, the voting system requires that at least t of a ballot's valid shares are available on the BB to reconstruct the ballot. By definition, $\mathsf{t}_{\mathsf{corr}} = n - t$. In terms of receipt-freeness, since a b_i is computed as a TREnc ciphertext, first, the TREnc's traceability ensures that as long as the lk remains secret, it is infeasible for anyone to change the message share x_i while keeping the trace τ_i unchanged even with sk. Additionally, knowing the link keys $\mathsf{lk} = \{\mathsf{lk}_i\}_{i=1}^{n}$ allows voting for any voting shares even for fixed traces (that an RF adversary would like to see on PBB) and the secret sharing properties allows computing shares of any vote v even with $t-1$ chosen shares. Moreover, the TCCA security of TREnc guarantees that an honest randomization will erase all malicious information that can be introduced in the ballot, rendering it infeasible for the individual who generated it to prove to anyone that it is the ballot of a particular vote. Consequently, as long as the combination includes at least *one* honestly re-randomized ballot

share if $t-1$ ciphertexts are corrupt, RF is achieved. As a result, the number of allowed malicious randomizers, i.e., process ballot servers should be no greater than $\mathsf{t_{rf}} = t-1$. See section 5.1 for the proof.

Given that $1 \leq \mathsf{t_{corr}} \leq n-t$, the proposed voting system offers a solution for scenarios where some randomizers are not functioning properly, whether intentionally or unintentionally. In such cases, incomplete ballots will nevertheless be proceeded to the tally. This ensures that the system can provide flexibility by handling unflavored cases and maintaining the integrity of the election process. In a specific case where a n-out-of-n threshold secret-sharing scheme is employed, $\mathsf{t_{corr}} = 0$ and $\mathsf{t_{rf}} = n-1$. That means, provided that all n valid ballot shares arrive on the BB, the system achieves RF even when only one honest randomizing server exists. To the best of our knowledge, this represents the strongest security notion for single-pass RF that has been proposed in the literature to date.

5 Security of the voting scheme

In this section, we prove that our generic voting scheme described in the previous section is threshold receipt-free (Section 5.1) and threshold correct (Section 5.2).

5.1 Threshold receipt-freeness

Theorem 1. *If the* TREnc *used in the voting scheme is TCCA secure and verifiable, and if the proof system used to prove the correctness of the tally is zero-knowledge, our voting scheme has threshold receipt freeness. More precisely, for a t-out-of-n secret sharing of the vote,* $\mathsf{t_{rf}} = t-1$, $\Pr[\mathsf{Exp}^{\mathsf{deceive}}_{\mathcal{A},\mathcal{V},\mathsf{t_{rf}}}(\lambda) = 1] \leq \varepsilon_{\text{verif}}$, *where* $\varepsilon_{\text{verif}}$ *is the verifiability advantage of any adversary against* TREnc, *and* $\mathsf{Adv}^{\mathsf{rf},\mathsf{t_{rf}}}_{\mathcal{A},\mathcal{V}}(1^\lambda) \leq \varepsilon_{\text{ZK}} + q(n-\mathsf{t_{rf}})\varepsilon_{\text{tcca}}$, *where* ε_{ZK} *is the zero-knowledge advantage of any adversary against the proof system,* $\varepsilon_{\text{tcca}}$ *is the TCCA advantage of any adversary against* TREnc *(recalled in Definition 4), and q is the number of queries made to* $\mathcal{O}\mathsf{receiptLR}$ *made by the adversary* $\mathcal{A}$.

Proof. **The experiment** $\mathsf{Exp}^{\mathsf{deceive},\mathsf{t_{rf}}}_{\mathcal{A},\mathcal{V}}(\lambda)$**.** Given $\rho = \{r_{ss}, r_{enc}\}$, where r_{ss} contains all random coins for the secret sharing scheme Share, i.e., the coefficients of the polynomial f in the Shamir's t-out-of-n threshold scheme, while r_{enc} contains lk and all the random coins in $\mathsf{TREnc.LEnc}$, the $\mathsf{Deceive}$ function aborts and outputs 0 if either v_0, v_1, ρ, or I fail to parse correctly. Otherwise,

1. It selects r_{ss} from ρ to output $\{v_0^i\}_{i\in I} = \mathsf{Share}(\mathsf{pk}, n, t, v_0; r_{ss})$, where each v_0^i is an evaluation of a degree-$(t-1)$ polynomial f at a point i. The adversary will be able to verify if $\{v_0^i\}_{i\in I}$ are in $\{\mathsf{b}_0^i\}_{i\in I}$ since these will be processed by dishonest randomizers and with known randomness r_{enc}.
2. To secretly share v_1, it produces another polynomial g of degree $t-1$ such that $g(0) = v_1$. To this end, the evaluation of g at a point i is fixed to be v_0^i for all $i \in I$, thereby providing at most t linear conditions. If $|I| < \mathsf{t_{rf}} = t-1$, add random input-output evaluation pairs to the interpolation to reach the appropriate degree. Subsequently, evaluate g at the point $i \in [n]\backslash I$ to compute v_1^i.

3. To generate b_1, it selects r_{enc} from ρ to run $\mathsf{TREnc.LEnc}$ on message shares $\{v_0^i\}_{i\in I}$ and $\{v_1^i\}_{i\in[n]\setminus I}$, which keeps the traces as $\mathsf{lk} \in r_{enc}$.

Since the Share operation based on the polynomial g is carried out honestly, any combination of t ballot shares in b_1 will result in the decryption of the same encrypted message. Eventually, even if the random coin r_{enc} are maliciously distributed, the TREnc verifiability guarantees that $\mathsf{b}_1 \in \mathsf{Vote}(\mathsf{id}, v_1)$.

The experiment $\mathsf{Exp}^{\mathsf{rf},\mathsf{t_{rf}},\beta}_{\mathcal{A},\mathcal{V}}(\lambda)$**.** To proceed with the $\mathcal{O}\mathsf{receiptLR}$ oracle defined in Figure 3, an attacker must produce two valid ballots, namely $\mathsf{b}_0 = \mathsf{B}_0||\mathsf{B}_2$ and $\mathsf{b}_1 = \mathsf{B}_1||\mathsf{B}_2$. The oracle then verifies that both ballots have identical traces. More precisely, the conditions $\mathsf{Valid}(\mathsf{b}_0) = \mathsf{Valid}(\mathsf{b}_1) = 1$ and $\mathsf{TraceBallot}(\mathsf{b}_0) = \mathsf{TraceBallot}(\mathsf{b}_1)$ must be met before processing them.
The proof involves a series of indistinguishable games, starting with the experiment $\mathsf{Exp}^{\mathsf{rf},\mathsf{t_{rf}},0}_{\mathcal{A},\mathcal{V}}(\lambda)$ ($\beta = 0$) and ending with $\mathsf{Exp}^{\mathsf{rf},\mathsf{t_{rf}},1}_{\mathcal{A},\mathcal{V}}(\lambda)$ ($\beta = 1$).

$\mathsf{Game}_1(\lambda)$**:** This is the experiment $\mathsf{Exp}^{\mathsf{rf},\mathsf{t_{rf}},0}_{\mathcal{A},\mathcal{V}}(\lambda)$ given in Figure 3 with $\beta = 0$. By definition, $\Pr[S_1] = \Pr[\mathsf{Exp}^{\mathsf{rf},\mathsf{t_{rf}},0}_{\mathcal{A},\mathcal{V}}(\lambda) = 1]$.

$\mathsf{Game}_2(\lambda)$**:** This game is as Game 1, with the exception that the keys of the election are generated using $\mathsf{SimSetupElection}$ which still produces the secret key sk of TREnc but also creates the additional trapdoor key to simulate proof for the tally. Moreover, we still honestly run Tally to get the result of the election but we erase the proof and instead run $\mathsf{SetupElection}$ on input the honest result. Since the proof system of the tally is zero-knowledge, we have $|\Pr[S_1] - \Pr[S_2]| \leq \varepsilon_{\mathrm{ZK}}$.

$\mathsf{Game}_3(\lambda)$**:** This game is as the previous game, except that the decryption is executed on-the-fly using $\mathsf{Dec}(\mathsf{sk}, \cdot)$ of TREnc in each call to $\mathcal{O}\mathsf{receiptLR}$ before the (perfect) randomization in $\mathsf{ProcessBallot}$. The result of the election is then computed from the resulting function. Since the view in Game 2 is identical to that of Game 1, we have $\Pr[S_2] = \Pr[S_3]$.

$\mathsf{Game}_4(\lambda)$**:** In this game, we introduce the following modification in the way we answer to the $\mathcal{O}\mathsf{receiptLR}$ queries made by the adversary, by gradually replacing processed ballots from B_0 by those of B_1 in BB_0.

$\mathsf{Game}_{4,1}(\lambda)$**:** In order to compute B'_0, instead of re-randomizing b_0^1, we re-randomize b_1^1. In other words, $\mathsf{B}'_0 = (\mathsf{b}_1^{1'}, \mathsf{b}_0^{2'}, \ldots, \mathsf{b}_0^{\mathsf{n}-\mathsf{t_{rf}}'})$. The probability that $\mathcal{A}$ distinguishes the difference after this modification is the probability that one is able to distinguish whether the first element of B'_0 is the randomization of b_0^1 or b_1^1. Since each ballot share is a valid TREnc ciphertext and that Valid rejects replaying the same traces, the TCCA challenger can call its own TCCA decryption oracle to compute the result as in Game 3 before the challenge phase since the corresponding trace has never been involved in an earlier query in another ballot piece, it follows that $|\Pr[S_3] - \Pr[S_{4,1}]| \leq \varepsilon_{\mathrm{tcca}}$.

$\mathsf{Game}_{4,\mathsf{i}}(\lambda)$**:** Keep doing the same way, we replace each element of B'_0 by a corresponding element in B'_1. Hence, $|\Pr[S_{4,i-1}] - \Pr[S_{4,i}]| \leq \varepsilon_{\mathrm{tcca}}$.

At the end of Game 4, we have $\mathsf{B}'_0 = (\mathsf{b}_1^{1'}, \ldots, \mathsf{b}_1^{\mathsf{n}-\mathsf{t_{rf}}'})$, which is identical to B'_1. Consequently, for the first query to $\mathcal{O}\mathsf{receiptLR}$ query, we have

$|\Pr[S_3]-\Pr[S_4]| \leq (n-\mathsf{t_{rf}})\varepsilon_{\mathrm{tcca}}$. Then, by an hybrid argument on all the q queries made by the adversary, we get $|\Pr[S_3]-\Pr[S_4]| \leq q(n-\mathsf{t_{rf}})\varepsilon_{\mathrm{tcca}}$.

The view of $\mathcal{A}$ in Game 4 is exactly its view in $\mathsf{Exp}^{\mathsf{rf},\mathsf{t_{rf}},1}_{\mathcal{A},\mathcal{V}}(\lambda)$. We thus have $\mathsf{Adv}^{\mathsf{rf},\mathsf{t_{rf}}}_{\mathcal{A},\mathcal{V}}(1^\lambda) \leq \varepsilon_{\mathrm{ZK}} + q(n-\mathsf{t_{rf}})\varepsilon_{\mathrm{tcca}}$. □

5.2 Threshold correctness

Theorem 2. *If the* TREnc *used in the voting scheme is traceable and verifiable, then the proposed voting scheme has threshold correctness. More precisely, for a t-out-of-n secret sharing of the vote, we have $\mathsf{t_{corr}} = t-1$ and for any efficient adversary $\mathcal{A}$, $\Pr[\mathsf{Exp}^{\mathsf{corr},\mathsf{t_{corr}}}_{\mathcal{A},\mathcal{V}}(\lambda)=1] \leq qn\varepsilon_{\mathrm{trace}}$, where $\varepsilon_{\mathrm{trace}}$ is the advantage of any adversary against traceability of* TREnc *(recalled in Definition 3), and q is the number of $\mathcal{O}$append queries.*

Proof. In the experiment $\mathsf{Exp}^{\mathsf{corr},\mathsf{t_{corr}}}_{\mathcal{A},\mathcal{V}}(\lambda)$, to append ballots to BB_1 and BB_0, $\mathcal{A}$ can make $\mathcal{O}$cast and $\mathcal{O}$append queries.

- $\mathcal{O}$cast appends the same malicious valid ballot to both bulletin boards. Since the identical ballot is published to both bulletin boards, this will not assist the adversary in winning the game, as it will not result in any difference in the tally results. That is also because the recombination is made deterministically on all the valid pieces.
- $\mathcal{O}$append records two valid ballots, b_0 and b_1, which are maliciously processed from an honestly generated ballot b, to the bulletin board BB_0 and BB_1 respectively. As defined, $\mathsf{b} = \{\mathsf{b}^i\}_{i=1}^n = \mathsf{Vote}(\mathsf{id},\mathsf{v})$, $T = \mathsf{TraceBallot}(\mathsf{b})$, and $\mathsf{TraceBallot}(\mathsf{b}_0), \mathsf{TraceBallot}(\mathsf{b}_1) \subset T$. Let us call I_0, I_1 two subset indexes of $[n]$, we denote $\mathsf{b}_0 = \{\mathsf{b}_0^i\}_{i\in I_0}$ and $\mathsf{b}_1 = \{\mathsf{b}_1^i\}_{i\in I_1}$ with $|I_0|, |I_1| \geq n - \mathsf{t_{corr}}$. As a result, $\mathsf{TraceBallot}(\mathsf{b}_j^i) \in T$ for all $j = 0,1$ and $i \in I_j$.
  First, since each ballot share is a TREnc ciphertext and TREnc is traceable, it is infeasible for anyone to produce another ciphertext that traces to the same trace and would decrypt to a different message. We thus have $\mathsf{Dec}(\mathsf{sk}, \mathsf{Combine}(\{\mathsf{b}_j^i\})) = \mathsf{Dec}(\mathsf{sk}, \mathsf{Combine}(\{\mathsf{b}^i\}))$ for all $j = 0,1$ and for each $i \in I_j$, except with a negligible probability $\varepsilon_{\mathrm{trace}}$.
  Second, since b is generated honestly, a combination of any subgroup of at least $n - \mathsf{t_{corr}}$ shares returns the same message v. Consequently, it always holds that $\mathsf{Dec}(\mathsf{sk}, \mathsf{Combine}(\{\mathsf{b}^i\}_{i\in I_0})) = \mathsf{Dec}(\mathsf{sk}, \mathsf{Combine}(\{\mathsf{b}^i\}_{i\in I_1}))$.
  The first and second points above infer that $\mathsf{Dec}(\mathsf{sk}, \mathsf{Combine}(\{\mathsf{b}_0^i\}_{i\in I_0})) = \mathsf{Dec}(\mathsf{sk}, \mathsf{Combine}(\{\mathsf{b}_1^i\}_{i\in I_1}))$ but with a negligible probability.

It is easy to see that with sk we can always decrypt each ballot pieces and figure it out whether the adversary managed to compute a TREnc ciphertext that reuses a trace but does not contain the original honest voting shares as message. Hence, $\Pr[\mathsf{Exp}^{\mathsf{corr},\mathsf{t_{corr}}}_{\mathcal{A},\mathcal{V}}(\lambda)=1] \leq qn\varepsilon_{\mathrm{trace}}$ by a standard guessing technique. □

As previously discussed, the current threshold correctness notion does not account for scenarios where an adversary introduces a malicious ballot with distinct ballot pieces. In such cases, different combinations of ballot shares could

yield different combined messages. However, since all valid ballot shares on the bulletin board (possibly more than t) are combined, only one message can be reconstructed. As the adversary cannot predict which ballot shares will be available due to any potential non-operational randomizers, it should produce truthful ballots to ensure the resulting vote accurately reflects its intent.

Verifiability. Our voting scheme satisfies both individual and universal verifiability. Individual verifiability is guaranteed through the VerifyVote steps, which allow voters to confirm the accurate recording of their votes and ensure that their preference remains unaltered. Universal verifiability is upheld by the verifiable mixnet, ensuring accurate tally computation and trustworthy final results.

6 Conclusion

We propose the notion threshold receipt-freeness, an extension of receipt-freeness that involves multiple ballot processing servers, removing a single point of failure of previous methodologies. Apart from preventing a malicious voter from providing proof of their vote to any third party, the novel definition allows an honest voter to cast their vote for their preferred candidate, even if some servers are compromised, provided that the adversary cannot vote on their behalf.

Additionally, we develop a generic construction of a single-pass verifiable voting system, based on traceable receipt-free encryption. Given any number n of ballot processing servers and $1 \leq t \leq n$, the resulting system maintains receipt-freeness and correctness as long as only t valid processed ballot pieces reach the bulletin board and even if $t-1$ of them were processed by the malicious servers who may reveal their random coins. Moreover, the trustworthy servers might differ for each voter. Practically, one may setup an election with only 3 servers so that any voter trusts at least any 2 of them, and their intentions will be taken into account if any 2 ciphertexts among the 3 computed are posted on the bulletin board after re-randomization.

Since the ballots are composed of n TREnc ciphertexts, their size is linear in the number of servers. We think that designing ballots of sub-linear size in n would require new techniques. Moreover, we leave it open to design a threshold receipt-free and correct election system compatible with homomorphic tally. Designing randomizable proof in a single-pass that maintains validity of the vote, for instance in an approval voting scenario, through the n servers which do not interact together seems to be challenging.

Acknowledgments

Thomas Peters is a research associate of the Belgian Fund for Scientific Research (F.R.S.-FNRS). This work has been funded in part by the Walloon Region through the project CyberExcellence (convention number 2110186).

References

1. Benaloh, J., Tuinstra, D.: Receipt-free secret ballot elections (extended abstract). In: Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, STOC 94. pp. 544–553. ACM (1994)
2. Bernhard, D., Cortier, V., Pereira, O., Smyth, B., Warinschi, B.: Adapting helios for provable ballot privacy. In: Computer Security–ESORICS 2011. LNCS, vol. 6879, pp. 335–354. Springer (2011)
3. Blazy, O., Fuchsbauer, G., Pointcheval, D., Vergnaud, D.: Signatures on randomizable ciphertexts. In: Public Key Cryptography–PKC 2011. LNCS, vol. 6571, pp. 403–422. Springer (2011)
4. Chaidos, P., Cortier, V., Fuchsbauer, G., Galindo, D.: BeleniosRF: A noninteractive receipt-free electronic voting scheme. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1614–1625. ACM (2016)
5. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Advances in Cryptology - CRYPTO '92. LNCS, vol. 740, pp. 89–105. Springer, Springer (1992)
6. Devillez, H., Pereira, O., Peters, T.: Traceable receipt-free encryption. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology - ASIACRYPT 2022. LNCS, vol. 13793, pp. 273–303. Springer (2022)
7. Devillez, H., Pereira, O., Peters, T., Yang, Q.: Can we cast a ballot as intended and be receipt free? In: 2024 IEEE Symposium on Security and Privacy (SP). IEEE Computer Society (2024)
8. Doan, T.V.T., Pereira, O., Peters, T.: Encryption mechanisms for receipt-free and perfectly private verifiable elections. In: International Conference on Applied Cryptography and Network Security–ACNS 2024. LNCS, vol. 14583, pp. 257–287. Springer (2024)
9. Hirt, M.: Multi party computation: Efficient protocols, general adversaries, and voting. Diss. ETH No. 14376 (2001)
10. Lee, B., Kim, K.: Receipt-free electronic voting through collaboration of voter and honest verifier. In: Japan-Korea Joint Workshop on Information Security and Cryptology (JW-ISC2000). pp. 101–108 (2000)
11. Lee, B., Kim, K.: Receipt-free electronic voting scheme with a tamper-resistant randomizer. In: Information Security and Cryptology—ICISC 2002. pp. 389–406. Springer (2003)
12. Libert, B., Peters, T., Joye, M., Yung, M.: Linearly homomorphic structure-preserving signatures and their applications. Designs, Codes and Cryptography **77**(2-3), 441–477 (2015)
13. Magkos, E., Burmester, M., Chrissikopoulos, V.: Receipt-freeness in large-scale elections without untappable channels. In: Towards The E-Society: E-Commerce, E-Business, and E-Government (I3E 2001). IFIP Conference Proceedings, vol. 202, pp. 683–693. Kluwer (2001)
14. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)