# Expanding the Toolbox: Coercion and Vote-Selling at Vote-Casting Revisited

Tamara Finogina[1]*, Javier Herranz[2], and Peter B. Rønne[3,4][0000−0002−2785−8301]

[1] Internxt
tamara@internxt.com
[2] Dept. Matemàtiques, Universitat Politècnica de Catalunya
Barcelona, Spain
javier.herranz@upc.edu
[3] University of Luxembourg, Esch-sur-Alzette, Luxembourg
peter.roenne@gmail.com
[4] CNRS, LORIA, Université de Lorraine, Nancy, France

**Abstract.** Coercion is a challenging and multi-faceted threat that prevents people from expressing their will freely. Similarly, vote-buying does threaten to undermine the foundation of free democratic elections. These threats are especially dire for remote electronic voting, which relies on voters to express their political will freely but happens in an uncontrolled environment outside the polling station and the protection of the ballot booth. However, electronic voting in general, both in-booth and remote, faces a major challenge, namely to ensure that voters can verify that their intent is captured correctly without providing a receipt of the cast vote to the coercer or vote buyer.

Even though there are known techniques to resist or partially mitigate coercion and vote-buying, we explicitly demonstrate that they generally underestimate the power of malicious actors by not accounting for current technological tools that could support coercion and vote-selling.

In this paper, we give several examples of how a coercer can force voters to comply with his demands or how voters can prove how they voted. To do so, we use tools like blockchains, delay encryption, privacy-preserving smart contracts, or trusted hardware. Since some of the successful coercion attacks occur on voting schemes that were supposed/claimed/proven to be coercion-resistant or receipt-free, the main conclusion of this work is that the coercion models should be re-evaluated, and new definitions of coercion and receipt-freeness are necessary. We propose such new definitions as part of this paper and investigate their implications.

## 1 Introduction

Coercion is one of those notions that is easier to understand than formally define, as it comes in many different shapes and forms. Generally, coercion incorporates

* Most of the work was done while the author worked at Scytl Election Technologies

all kinds of duress that can prevent people from acting freely and expressing their will. There are possibilities for coercive attacks ranging from persuasion and blackmail to intimidation and physical threats. Yet, all of them, regardless of the specifics, have one thing in common: the coercer wants to minimize the possibility for people to disobey undetectably.

The threat is especially relevant for verifiable voting and dire for remote electronic voting, which should facilitate voters to express their political will freely, but happens in an uncontrolled and potentially coercive environment. Hence, this area offer many distinct proposals for resisting, mitigating, or hampering the coercion threat - such as multiple voting, simulatable credentials, interactive proofs, etc. All those methods allow voters to appear obedient while casting their intended vote. The main requirement is that the coercer cannot constantly control the voter nor intercept information sent over secure channels.

The assumption that a coercer cannot constantly control the voter is necessary for the voter to have a chance to express her own will at some point during the voting phase. However, new tools like immutable blockchains, delay functions, time-based encryption, secret-input MPC smart contracts, trusted hardware, etc., have been developed to enforce certain types of honest behavior of participants. Our idea in this paper is to demonstrate how such tools in the hands of a coercer, in turn, can be used to ensure that the coerced voter follows the instructions of the coercer and cannot evade via anti-coercion strategies. This means that the coercer does not have to constantly monitor or interact with a voter, and opens the possibility of powerful coercion attacks where a single malicious entity can coerce many voters simultaneously. Going further, these new tools can also be used by a voter to self-impose certain behaviors to extract proof of the cast vote, thereby opening the door to efficient vote-selling.

The reason for the hardness of designing coercion-resistant or receipt-free e-voting protocols lies in the fear of a misbehaving computer undetectably modifying the voter's intention. This requires some assurance that the voting device has not altered a voter's vote – a check known as cast-as-intended verification – and is facilitated, e.g., via tracking numbers, return codes, QR codes containing encryption randomness, zero-knowledge proofs of plaintext correctness or other techniques. However, the cast-as-intended verification's output needs to only convince the voter, not a coercer or a vote-buyer, e.g. by being deniable. This also means that all current state-of-the-art verifiable e-voting solutions do not simply provide correctness proof for a final cast vote (e.g., in the form of the random coins used in generating the ballot) but also a simulation strategy.

Several studies focus on this contradiction and offer potential solutions, see, e.g., [18] and references therein. However, to our knowledge, no paper has thoroughly studied the possibility of the coercer utilizing new cryptographic tools - blockchain, delay functions, time-lock encryption, etc. - to prevent the voter from simulating an alternative proof.

To see an example of how a coercer can utilize new tools for forcing voters to vote for a specific candidate, consider a voting system that relies on so-called Benaloh challenges [4] for cast-as-intended verification: A voter enters her chosen options into their voting device, then the voting device prepares the ballot, commits to it, and asks the voter whether to cast it or audit it. In case of an audit, the voting device must reveal the encryption randomness, which allows the voter to verify the ballot on another device. Otherwise, the vote is submitted without any verification. The point is that the voter will never hold the randomness of the submitted ballot (assuming the software does not leak this) and hence have no direct receipt. Of course, the number of audits before casting should be unpredictable, otherwise, the voting device would know when to cheat.

Now we show how, with the aid of a blockchain, a remote coercer can always force voters to cast a ballot for a given option. To do so, the coercer tells voters to post the ballot's commitment on the blockchain, e.g., Bitcoin, before deciding whether to audit or cast the vote. If the next block starts with a bit 0, the voter must press audit and post the corresponding randomness on the blockchain. Otherwise, the voter casts the ballot. Since the coercer can see everything posted on the blockchain, he can always check if the voter behaved. Theoretically, the voter can disobey and commit to a ballot with a different candidate from the coercer's preference. However, with the probability of 1/2, the disobedience would get caught because the voter can't show randomness that corresponds to the committed ballot and encrypts the coercer's option.

The tools we consider are blockchain technology to enforce order, a non-malleable commitment of data, and unpredictable randomness. Delay functions and time-lock encryption, which recently regained popularity, can help the coercer ensure that a voter does not learn a secret until after some time, which can be used to prevent the simulation of proofs. Privacy-preserving smart contracts, e.g., MPC-based [3,32], Trusted Execution Environments, and other trusted hardware can ensure the voter never knows a secret key required for coercion mitigation. These tools can act in place of the coercer and interact with the voter during the vote-casting phase, while the coercer or vote-buyer only verifies all the evidence at the end of the election.

Since the new tools allow an adversary to control the voter without observing her continuously, coercion can be done at a large scale without substantial costs. Hence, it is critical to evaluate and discuss these new attack vectors.

### 1.1 Related Work

Exploiting ballot verification mechanisms for coercion is not new, and we here present related work. An attack similar to the attack mentioned in the introduction on Benaloh challenges was presented [11], but without using blockchain technology. As we explain in Section 3.1, the attack in[11] can be mitigated (i.e., the voter still can disobey and conceal this fact), whereas ours is not.

An interesting coercion attack utilizing scratch-off cards was proposed in [22] against the Punchscan [31] two-part ballots. The number revealed after scratching will force the voter to reveal a certain ballot part - much like the attack on Benaloh challenges. However, scratch-off cards require a physical delivery and support only a limited range of options. Thus, it would be infeasible for many digital ballots. Going further, our attack also works against receipt-freeness, as we explain below, whereas for the code sheets this would require a voter to obliviously create scratch codes on behalf of a vote-buyer, which seems hard to achieve in practice.

Many voting schemes have been proposed using various blockchain primitives to achieve different forms of security, perhaps most famously [25] used smart-contracts to prevent denial-of-service to a decentralized voting scheme, and in [7] smart contracts were used to disincentivize vote-selling.

On the other hand, many schemes have been proposed that naively implement blockchain technology and claim security without properly understanding possible pitfalls and the alignment of incentives [27,28].

Trusted hardware has also been suggested to ensure cast-as-intended verifiability for voting schemes. Quite independently, we here demonstrate that it can be a tool for coercion in normal voting schemes. However, we point out that there is a quite different alignment of trust assumptions. Generally, we would like not to trust the vendor's hardware for cast-as-intended verifiability. The reason is that (according to the consensus in the e-voting community) trust should be avoided for verifiability. For example, the vendor could be hacked to leak the secret keys used in the trusted hardware, and the election result could then be manipulated. On the other hand, a voter coerced via trusted hardware could use the secret key to evade the coercion, but it would be hard for the vendor to assess such a situation and it hence unlike to provide the secret key to the voter. An interesting perspective is presented in [35] where the trusted hardware is used to achieve coercion-resistance.

Also note that parallel to our contribution is a blog post on vote buying in special DAOs [1] using SGX.

Finally, we note that we only consider coercion- and vote-buying rising from the vote-casting procedure. There has recently been improvements on the state-of-the-art for definitions of covering the full election, especially coercion attacks during the tally phase of JCJ [15]. See also [20] for recent definitions of receipt-freeness. However, this is out of scope for this paper.

### 1.2 Contribution and Organization of the Paper

We study unexplored coercion and vote-buying attacks based on new cryptographic primitives such as blockchain, delay function, time-lock encryption, privacy-preserving smart contracts, trusted hardware, etc. We give examples of new tools usage by showing how they help the coercer to force voters to comply or the voter to obtain a (probabilistic) receipt for vote selling. We will also present some attacks inspired by these, which will work even without these tools. Our last contribution will be the proposal of new security definitions for coercion-resistance and receipt-freeness that take into account the possibility that both the coercer and the voter can use such new tools.

We start by stating our model and trust assumptions in Section 2. First, we describe and list expectations for the new tools available to the coercer in Section 2.3 and categorize the coercion attack types in Section 2.4. Then, in Section 3, we proceed with attacks on the known e-voting verification methods and schemes. In Section 4, we explain why deniable re-voting or vote updates could be analysed in our framework, or needs trust for cast-as-intended verifiability. Section 5 contains new security definitions for the notions of coercion-resistance and receipt-freeness, and some relations between them. Finally, in Section 6, we summarize our observations and we briefly state some impossibility results we encountered and should be considered in future work.

## 2 Voter and Adversary Model

### 2.1 Parties

The parties involved in our protocol are the following:

$\mathcal{EA}$: Denotes the election authority. It is trusted for privacy and hence for coercion-resistance, but the trust is usually distributed among several parties. For our purpose, it is sufficient to consider a single authority. It should not be trusted for verifiability, as discussed in the introduction.

$\mathcal{BB}$: Denotes the bulletin board. It is used to collect ballots and verifiably derive the tally result.

$\mathcal{V}$: Denotes voter. We are only concerned about the verifiability and coercion-resistance of the vote-casting procedure; thus, we consider only a single voter. There are privacy, coercion, and verifiability threats arising from the broader election process involving all voters, but this is out of the scope of this paper.

$\mathcal{C}$: Denotes the adversary against coercion-resistance.

$\mathcal{A}_{\text{ver}}$ Denotes the adversary against verifiability.

$\mathcal{VD}$: Denotes the voting device. It helps the voter to prepare the ballot and sends it to $\mathcal{BB}$. It is assumed to be corrupted by $\mathcal{A}_{\text{ver}}$ but not colluding with $\mathcal{C}$.

### 2.2 Voter Computation and Communication Model

In our model, we consider a voter without any pre-shared with $\mathcal{EA}$ knowledge except public election parameters available on $\mathcal{BB}$. We assume $\mathcal{C}$ can observe the ballot that $\mathcal{V}$ sends to $\mathcal{EA}$, e.g., because it is directly published on $\mathcal{BB}$.

We here do not consider a bound on the voter's computational ability except being PPT. Of course, this is not a realistic model, but it rather models a voter having access to a device that might leak random coins, etc., which the voter can give to $\mathcal{C}$ or a vote buyer.

We will assume that $\mathcal{C}$ is not present during the vote casting, but can give instructions before and get information after the session to verify if the voter followed instructions.

### 2.3 The Coercer's Toolbox

We consider different cryptographic means that the coercer or vote buyer can use to control the voter without being present in the vote-casting situation. We assume the voter wants to vote for $\mathsf{Cand}_{\mathcal{V}}$ and the coercer expects $\mathsf{Cand}_{\mathcal{C}}$.

- $\mathsf{Instr}$: Instructions that $\mathcal{C}$ gives to $\mathcal{V}$ before voting. We assume $\mathcal{V}$ already knows the preferred candidate of $\mathcal{C}$, but $\mathsf{Instr}$ will provide more details.
- $\mathsf{CC}$: Chain of commitments. The committed values are add-only and immutable. This can be done, for example, via a blockchain or a hardware device that stores input from the voters.
- $\mathsf{CC\text{-}PRF}$: Chain of commitments with (pseudo-)random output between commitments. One example could be Bitcoin, with the hash pointers treated as pseudorandom output. Another option is a hardware device taking inputs $x_i$ and returning $y_i = \mathsf{H}(x_i||y_{i-1}||\mathsf{sk})$, where $\mathsf{H}$ is a cryptographic hash function, and $\mathsf{sk}$ is a secret key only known to $\mathcal{C}$. Knowing the last output and inputs, $\mathcal{C}$ can verify the entire transcript without asking the hardware token back.
- $\mathsf{Timed\text{-}CC}$, $\mathsf{Timed\text{-}CC\text{-}PRF}$: Timed chain of commitments without/with pseudorandom output. Similar to $\mathsf{CC}$ and $\mathsf{CC\text{-}PRF}$, but also commitments are timestamped. A blockchain or a hardware token with timings would suffice.
- $\mathsf{Timed\text{-}Enc}$: Timed release of secrets. It can be done via Time-Lock-Puzzles [33], Delay Encryption [9], Homomorphic Time-Lock Puzzles [24], etc.
- $\mathsf{Token}$: Tamper-proof hardware token. It gets inputs from the voter and can give outputs, record timings, store secret values known only by the coercer, and generate public keys while keeping private keys safe in the module. The coercer can ask the voter for the full transcript of inputs and outputs from the device and verify everything without receiving the token back. This can be done, via a Trusted Platform Module (available on most modern laptops, PCs, and smartphones), a Trusted Execution Environment, general trusted hardware, or privacy-preserving smart contracts (e.g., MPC-based versions).

### 2.4 Coercion Attack-Types

We also classify different types of attacks according to their severity and difficulty

- $\mathsf{Attack{:}Precision}$: An attack we can carry out with a probability that can be made close to 1.
- $\mathsf{Attack{:}Probabilistic}$: An attack where the coercer has a certain probability to carry it out, but this probability is not close to 1.
- $\mathsf{Attack{:}Complex}$: An attack where the coercer has to estimate a bound on the computational power of $\mathcal{V}$, e.g., for the delay time in the primitives in $\mathsf{Timed\text{-}Enc}$ or the number of devices that $\mathcal{V}$ has.

### 2.5 Security Properties

We informally define, below, the properties of cast-as-intended verifiability, coercion-resistance, and receipt-freeness.

*Cast-as-Intended Verifiability (CAI)* A precise game-based definition of CAI verifiability can be found in [34], and is included in Appendix A for completeness. The idea of the definition is that the voter $\mathcal{V}$ interacts with the adversary $\mathcal{A}$ that fully controls the voter's voting device, $\mathcal{VD}$, to create a ballot. The voter can verify the submitted ballots, proofs on the bulletin board, etc. using some trusted device or via a proxy verifier. If the verification fails, the game will be abandoned. The adversary wins if the cast ballot, accepted by the voter, is not in the image of the voting algorithm using the intended vote, i.e. the ballot has been invalidated or is for another candidate.

*Coercion-Resistance for the Vote-Casting Phase* A formal game-based definition can be found in [18]. The point of this definition is to consider coercion-resistance limited to the vote-casting phase. This gives a lighter definition than full coercion-resistance which has to consider possible coercion threats during the tally phase and in particular, unavoidably, from the tally result. Basically, a protocol is then coercion-resistant if the coercer cannot distinguish a world where the voter follows the instructions of the coercer and honestly outputs its view of the communication with the voting device $\mathcal{VD}$ from a world where the voter casts a ballot according to her own choice and simulates the view.

*Receipt-Freeness* Finally, receipt-freeness can be defined via a game where for any voter algorithm voting for a candidate $A$ and outputting its view, there exists a simulator voting for candidate $B$ with a simulated view, and any adversarial algorithm (vote buyer) cannot distinguish the two worlds. Note that contrary to coercion-resistance, the vote buyer does not issue instructions directly to the voter before the election. However, he might publicly commit to the preferred candidate and (maybe) some specific conditions the ballot must have.

In Section 5, we will propose precise game-based definitions for these two notions. The new tools (in particular blockchains or time-lock encryption) are modeled as oracles accessible to the parties. Coercion-resistance can be defined in different forms depending on the type of instructions the coercer has to give to the voter. Importantly, the new tools will give rise to novel relations between security properties. As an example, whereas coercion-resistance always implies receipt-freeness, however, with access to timed commitments we can also relate receipt-freeness to a weaker form of coercion-resistance (Theorem 1 below).

## 3 Attacks

We investigate how a coercer can use the above tools to attack some CAI mechanisms in the e-voting literature. The attack impact varies from completely breaking privacy to computationally penalizing voters for using disobedience strategies. The attacks are categorized based on the type and the coercer's tools.

### 3.1 Benaloh Challenges

As mentioned in the introduction, the Benaloh challenges are a perfect example to demonstrate the different coercer tools and estimate how easily a coercer can attack multiple voters. Below, in Figure 1, we give a simplified graphic

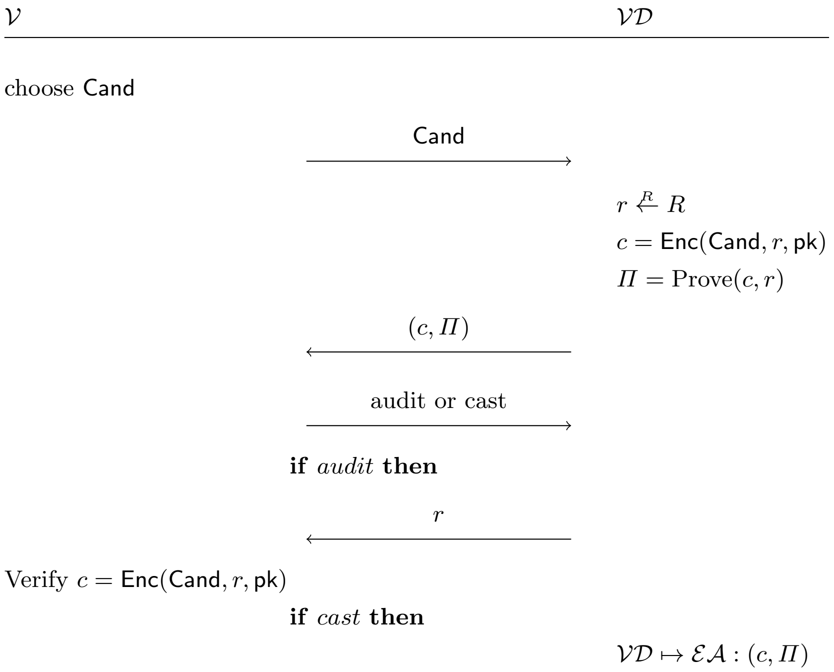
$\mathcal{V}$
$\mathcal{VD}$
choose $\mathsf{Cand}$
$\mathsf{Cand}$
$r \overset{R}{\leftarrow} R$
$c = \mathsf{Enc}(\mathsf{Cand}, r, \mathsf{pk})$
$\mathit{\Pi} = \mathrm{Prove}(c, r)$
$(c, \mathit{\Pi})$
audit or cast
**if** *audit* **then**
$r$
Verify $c = \mathsf{Enc}(\mathsf{Cand}, r, \mathsf{pk})$
**if** *cast* **then**
$\mathcal{VD} \mapsto \mathcal{EA} : (c, \mathit{\Pi})$

**Fig. 1.** A simplified diagram of a Banaloh challenges verification mechanism.

description of the protocol. It aims to facilitate understanding of the attacks we propose. For a detailed description please refer to [2,30].

Whereas Benaloh challenges were never claimed to be coercion-resistant[5] these attacks were not considered earlier. Even further, it is generally believed that Helios is receipt-free if the software does not leak the random coins, see e.g. page 3 of [10], which could e.g. be enforced using a hardware root of trust.

*(*$\mathsf{Attack{:}Complex};\mathsf{Instr}$*)* An attack fitting our narratives was proposed for a polling booth- Helios [11]. We believe it would work for remote voting: $\mathcal{C}$ tells the voter to vote only if the receipt hash $h$ fulfills some predicate $P(h)$ (e.g., the number of leading null bits which happens with some probability $p$) and audit otherwise. Then $\mathcal{C}$ demands to see all audited receipts and random coins. The attack can be avoided but requires double effort: the voter first uses the coercer's choice to obtain verification data, then (instead of casting the vote when receipt permits it) switches to the preferred option and re-runs the voting process until receipt allows vote-casting. Of course, all audit material corresponding to the voter's choice must be destroyed.

*(*$\mathsf{Attack{:}Precision};\mathsf{Instr},\mathsf{CC\text{-}PRF}$*)* The coercer instructs the voter to use the $\mathsf{Cand}_{\mathcal{C}}$, then add a commitment to the ballot that the voting device shows to $\mathsf{CC\text{-}PRF}$ and only cast if the $\mathsf{CC\text{-}PRF}$ output starts with 1 (alternatively: 0 or more complex predicate). The expectation is that the voter cannot predict when the $\mathsf{CC\text{-}PRF}$ will allow casting the ballot; thus, she does not know when it's safe to misbehave and use her preference. The coercer can always check that the commitment of the casting vote was added to the $\mathsf{CC\text{-}PRF}$ and resulted in the output indicating the case. Therefore, the voter has a high risk of being caught in the case of disobedience. In case of just checking the first bit this probability is $p = 1/2$, but the coercer can increase this to a general probability $p$ the cost of the voter having to do $1/(1-p)$ vote cast attempts on average.

Note that a voter with a $\mathsf{CC\text{-}PRF}$, e.g. access to Bitcoin, also can use this to get a receipt of the vote, i.e. Helios is not receipt-free even with trusted software.

On a high level, the first previous attack (suggested in [11]) looks very similar to the second one (proposed by ourselves), with the only distinction being the use of blockchain. However, we claim this is not the case. To see why, one should observe that in the polling-booth-Helios the voter receives an electronic hash of her receipt as a commitment from the machine. This commitment is not publicly posted or stored anywhere. It is given to the voter in the privacy of the voting booth. Therefore, a realistic coercer (i.e. one who cannot compute the exact amount of time spent by the voter during vote casting) would have no way of knowing exactly how many hash commitments the voter received and would not notice if a few were not used. Thus, the voter can destroy receipts indicating audit and only show the coercer the receipt that allows casting. Unfortunately, omitting some of the receipts would be impossible with the blockchain attack as it is specifically designed to preserve the immutability of records.

[5] An early version of Helios had a "coerce-me" button to point to the danger of coercion in remote e-voting which handed out the random coin.

### 3.2 STAR-Vote

For CAI verification, STAR-Vote [5] offers a novel variant of Benaloh's challenge: the voter either deposits the ballot in the ballot box or not. First, the voter makes selections on a terminal, which prints the paper ballot in human-readable form with a random serial number and a corresponding receipt that the voter might take home. The voting terminal also sends the encrypted vote and the corresponding receipt to the judge station and publishes the commitment to the ballot on a publicly accessible bulletin board. If the voter chooses to cast the vote, she takes the paper ballot to the ballot scanner, which reads the ballot's serial number and marks it as complete. If the voter decides to spoil the vote, she should return to a poll worker, who scans the vote and indicates it is spoiled. Such a vote would be decrypted as such during the tally. The verification mechanism works like the original Benaloh challenge: the voting terminal commits to the ballot before it knows whether the voter decides to cast or spoil it. However, the procedure itself is adapted to a polling place voting.

*(*Attack:Probabilistic; Instr*)* The coercer tells the voter to cast their ballot only if the printed receipt starts with some predicate, say a bit '0'. Otherwise, the voter must spoil the vote and give the receipt to the coercer. For our example, the chance of an audit is 1/2, but it can vary depending on the complexity of the predicate. Regardless, the voter cannot predict when the vote-casting happens and thus must take a risk or obey. However, if the voter disobeys and the receipt indicates spoiling the ballot, the coercer can trivially detect misbehavior by checking the decrypted spoiled ballot.

### 3.3 Belenios-CaI

Belenios [14] is built upon the Helios and recently obtained CAI verifiability [13]. After the voter selects a vote $v$, she receives two random integers $a$ and $b$ such that $b = v + a(\text{mod}\mu)$ for some positive $\mu$ larger than the biggest possible $v$. Then, the ballot is formed as three ciphertexts encrypting values $v$, $a$, and $b$, plus a zero-knowledge proof that $b = v + a(\bmod \mu)$. After that, the voting device commits to the ballot and asks the voter to choose if the ciphertext encrypting $b$ or $a$ should be opened. The selected ciphertext is publicly opened. To modify $v$ and create a convincing zero-knowledge proof, one has to change both $v$ and one of the values $a$ or $b$; therefore, the voter will detect it with probability 1/2. Below, in Figure 2, we give a simplified graphic description of the protocol. It aims to facilitate understanding of the attacks we propose. For a detailed description please refer to [13].

We stress that, as far as we know, BeleniosCAI has never claimed to enjoy receipt-freeness. It only highlights that revealing only one of two values does not affect the privacy of the vote but says nothing about vote-selling or coercion. Mostly, this is because the Belenios voting family defines receipt-freeness in the strong sense, where the voter can forcefully extract randomness from the voting device to facilitate vote-selling. However, in our model, we trust $\mathcal{VD}$.

| $\mathcal{V}$ | | $\mathcal{VD}$ |
|---|---|---|
| choose $\mathsf{Cand}$ | | |
| | $\xrightarrow{\mathsf{Cand}}$ | |
| | | $r, s_0, r_0, r_1, \overset{R}{\leftarrow} R$<br>$s_1 = (\mathsf{Cand} - s) \bmod k$<br>$c = \mathsf{Enc}(\mathsf{Cand}, r, \mathsf{pk})$<br>$c_0 = \mathsf{Enc}(s_0, r_0, \mathsf{pk})$<br>$c_1 = \mathsf{Enc}(s_1, r_1, \mathsf{pk})$<br>$\pi \leftarrow \{\mathsf{Dec}(c) = \mathsf{Dec}(c_0 \cdot c_1)\}$<br>$\sigma \leftarrow \mathsf{Sign}(c, c_0, c_1; \mathsf{sk})$<br>$\mathcal{VD} \mapsto \mathcal{BB} : (c, c_1, c_2, \pi, \sigma)$ |
| | $\xleftarrow{(c, c_1, c_2, s_0, s_1)}$ | |
| Verify $\mathsf{Cand} = s_0 + s_1 \bmod k$<br>Verify $(c, c_1, c_2) \in \mathcal{BB}$<br>choose $\alpha \in \{0, 1\}$ | | |
| | $\xrightarrow{\alpha}$ | |
| | | $\mathcal{VD} \mapsto \mathcal{BB} : (s_\alpha, r_\alpha)$ |
| Verify $s_\alpha$ | | |

**Fig. 2.** A simplified diagram of the BeleniosCAI verification mechanism.

($\mathsf{Attack{:}Probabilistic; Instr, CC\text{-}PRF}$) The coercer $\mathcal{C}$ instructs the voter $\mathcal{V}$ to use the $v = \mathsf{Cand}_{\mathcal{C}}$, commit all values generated by $\mathcal{VD}$ to $\mathsf{CC\text{-}PRF}$, and choose between $a$ or $b$ based on the $\mathsf{CC\text{-}PRF}$'s output. Theoretically, voter can receive $(c_v, c_a, c_b, a, b)$ corresponding to $\mathsf{Cand}$, then set $b^* = (\mathsf{Cand}_{\mathcal{C}} - a)$ and post $(c_v, c_a, c_b, a, b*)$ on the $\mathsf{CC\text{-}PRF}$. However, if the output of the $\mathsf{CC\text{-}PRF}$ indicates to open $c_b$, then the coercer would notice the disobedience. Again this attack can also means there is no receipt-freeness for a voter with access to $\mathsf{CC\text{-}PRF}$.

### 3.4 Themis

Closely related to BeleniosCAI is the in-person voting scheme Themis [6], which uses the same idea of splitting the candidate number $v$, which is always odd, into randoms $a$ and $b$, ensuring that $v = a + b \bmod 2n$ ($n$ is the number of candidates) and verifying the encryption of one of the numbers. However, the voter gets this splitting on a printed ballot and chooses which side to audit.

($\mathsf{Attack{:}Probabilistic; Instr}$) Assume the voter can compute a boolean function $f$ in the head. The voter in the booth computes $f(a, b)$ in the head and audits the left or right side according to the value. For example, assume that $f = 0$ indicates opening $a$ while $f = 1$ says audit $b$. If the voter votes for $v = \mathsf{Cand}_{\mathcal{V}}$ and gets $a$ and $b$ such that $v \equiv a+b$ but then claims to have selected $v^* = \mathsf{Cand}_{\mathcal{C}}$, she needs to fake $a$ and make sure that $f(a, v^* - b) = 0$ or fake $b$ and ensure that $f(v^* - b, b) = 1$. If $f$ is random, then it can happen with probability 1/4. It might not be high, but it is an interesting observation and could be enough to have a monetary incentive for a vote buyer.

($\mathsf{Attack{:}Probabilistic; Instr}$) A better and easier attack is as follows. The possibilities for $a$ and $b$ such that $a + b = v \bmod 2n$ are depending on $v$. Consider a simple case of $n = 2$ candidates (e.g., "A" and "B") with assigned numbers 1 and 3. Then for the candidate A the possible codes are $(0, 1)$ and $(2, 3)$, and for B – $(0, 3)$ and $(2, 1)$. If the coercer demands the audited number to be 0 or 1, voting for B always allows compliance with the demand. However, voting for A would result in $(a, b) = (0, 1)$ only in 1/2 of cases. Thus, if the voter votes for A, the coercer will find out with the probability of 1/2. Note that the attack can scale to more candidates if the coercer demands computing numbers modulo 4.

### 3.5 Proof of Correct (Re-)Encryption

Voting schemes often offer the voter a proof of correct encryption or re-encryption (of the ciphertext that contains the chosen option) as a CAI verification method. Of course, such proof should be interactive, or else the coercer can demand to see it. Moreover, the proof should have full zero-knowledge and not merely honest-verifier zero-knowledge if one wants to avoid coercion. Below, we present and attack three different proposals for verification based on (re-)encryption correctness.

**3.5.1 Commitment to the Challenge.** A paper [19] analyzes a $\Sigma$-protocol for proving encryption correctness with a commitment to a challenge and concludes it's both coercion-resistant and CAI. However, we will show how such a scheme can be attacked using new coercion tools.

Below, in Figure 3, we give a simplified graphic description of the protocol. It aims to facilitate understanding of the attacks we propose.

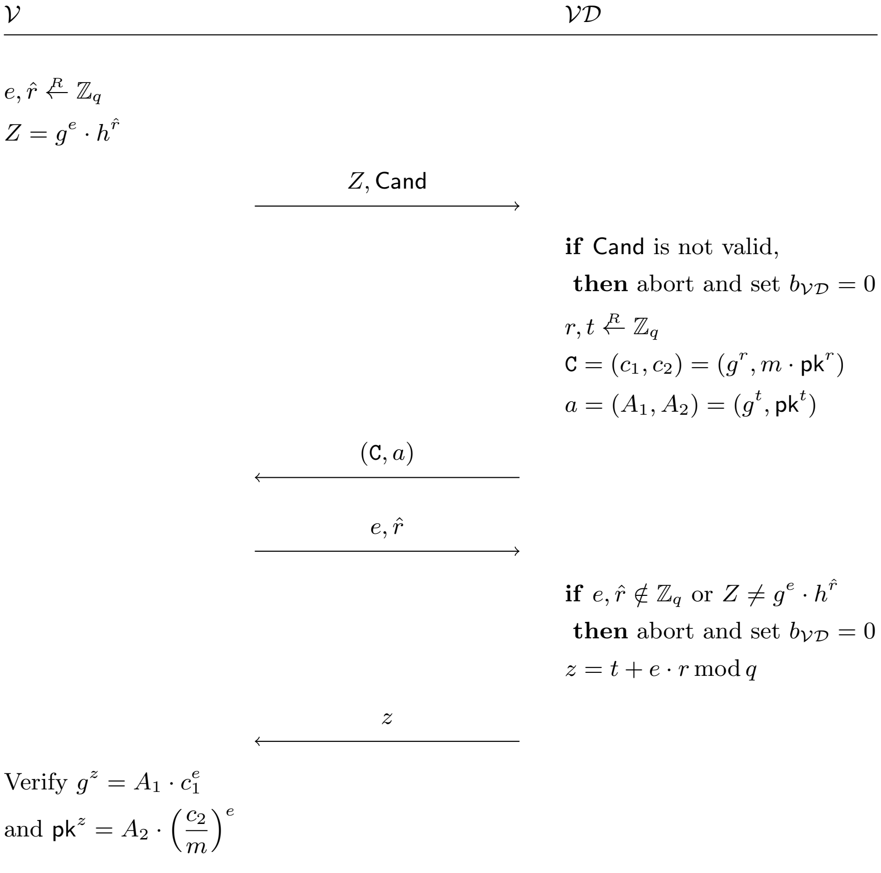
$\mathcal{V}$
$\mathcal{VD}$
$e, \hat{r} \xleftarrow{R} \mathbb{Z}_q$
$Z = g^e \cdot h^{\hat{r}}$
$Z, \mathsf{Cand}$
**if** $\mathsf{Cand}$ is not valid,
**then** abort and set $b_{\mathcal{VD}} = 0$
$r, t \xleftarrow{R} \mathbb{Z}_q$
$\mathsf{C} = (c_1, c_2) = (g^r, m \cdot \mathsf{pk}^r)$
$a = (A_1, A_2) = (g^t, \mathsf{pk}^t)$
$(\mathsf{C}, a)$
$e, \hat{r}$
**if** $e, \hat{r} \notin \mathbb{Z}_q$ or $Z \neq g^e \cdot h^{\hat{r}}$
**then** abort and set $b_{\mathcal{VD}} = 0$
$z = t + e \cdot r \bmod q$
$z$
Verify $g^z = A_1 \cdot c_1^e$
and $\mathsf{pk}^z = A_2 \cdot \left(\frac{c_2}{m}\right)^e$

**Fig. 3.** A simplified diagram of the verification mechanism based on the Commitment to the Challenge.

The public parameters of the election system must contain elements $(q, G, g, h)$ such that $G = \langle g \rangle = \langle h \rangle$ has prime order $q$. To commit to the challenge, the perfectly hiding Pedersen commitment scheme [29] is used. In Step 1, voter samples $e, \hat{r} \xleftarrow{R} \mathbb{Z}_q$, computes $Z = g^e \cdot h^{\hat{r}}$ and sends $(Z, \mathsf{Cand})$ to $\mathcal{VD}$. In Step 2, $\mathcal{VD}$ samples $r, t \xleftarrow{R} \mathbb{Z}_q$, computes $\mathsf{C} = (c_1, c_2) = (g^r, \mathsf{Cand} \cdot \mathsf{pk}^r)$ and $a = (A_1, A_2) = (g^t, \mathsf{pk}^t)$, so that values $(\mathsf{C}, a)$ are sent back to $\mathcal{V}$. In Step 3, $\mathcal{V}$ replies with $(e, \hat{r})$. Finally, $\mathcal{VD}$ checks that $Z = g^e \cdot h^{\hat{r}}$, computes $z = t + e \cdot r \bmod q$ and sends $z$ to $\mathcal{V}$, who can verify that both $g^z = A_1 \cdot c_1^e$ and $\mathsf{pk}^z = A_2 \cdot \left(\frac{c_2}{\mathsf{Cand}}\right)^e$ hold.

*(*Attack:Complex; Instr, Timed-CC-PRF, Timed-Enc*)* Shortly before the voting phase, the coercer $\mathcal{C}$ gives the voter $\mathcal{V}$ commitments $Z = g^e h^{\hat{r}}$ and the corresponding openings under delay encryption $X = \mathsf{Delay}(\hat{r}||e)$, which can be opened only after time $\mathcal{T}$. The voter is ordered to commit to the ciphertext $\mathsf{C} = (c_1, c_2)$ and the first move of the sigma protocol $a = (A_1, A_2)$ using timed commitment chain of the coercer's choice Timed-CC before time $\mathcal{T}$. We note that to prevent precomputation by the voter, the coercer could use timed encryption like [16] to release the puzzle at a precise time.

One can consider a modified protocol (with five rounds, started by $\mathcal{VD}$) where (i) the generator $h$ for Pedersen commitments is not fixed in the public parameters, but instead chosen by $\mathcal{VD}$ in step 1 of the protocol, and (ii) the commitment sent by $\mathcal{V}$ in step 2 is defined as $Z = g^{\hat{r}} \cdot h^e$ instead. This is actually the specific instantiation of the protocol proposed in the Appendix of [26], discussed in the upcoming Section 3.5.3. The coercion strategy based on combining a blockchain and a delay function does not seem to work against this protocol modification.

**3.5.2 Designated Verifier Proof.** Another way to construct an interactive full zero-knowledge proof of encryption correctness proposed in [19] is to use a (possibly non-interactive) OR proof "statement is true or I know the solution $x$ (also known as trapdoor) of a hard problem $y$" combined with a proof of the trapdoor $x$ knowledge. Note that proving $x$ knowledge is critical because, otherwise, a coercer can force $\mathcal{V}$ to use a value $y$ without revealing the corresponding solution $x$. Moreover, this proof cannot be non-interactive because the coercer might instruct the voter to use a pre-made proof otherwise.

The final protocol is roughly the following: in Step 1, $\mathcal{V}$ samples $x, t \xleftarrow{R} \mathbb{Z}_q$, computes $Y = g^x$, $T = g^t$ and sends the pair $(Y, T)$ to $\mathcal{VD}$. In Step 2, $\mathcal{VD}$ samples and sends a challenge $e \xleftarrow{R} \mathbb{Z}_q$. In Step 3, $\mathcal{V}$ computes and sends the answer $s = t + x \cdot e \bmod q$, which is used by $\mathcal{VD}$ to check the correctness of this interactive proof: $g^s = T \cdot Y^e$. If so, $\mathcal{VD}$ finishes the protocol by computing a non-interactive zero-knowledge OR proof, as described in the previous paragraphs.

*(*Attack:Complex; Instr, Timed-CC-PRF, Timed-Enc*)* The coercer gives the voter $y, \mathsf{Delay}(x), a, \mathsf{Delay}(s)$, where Delay is a homomorphic function that allows recovering the hidden secret in time $\mathcal{T}$. Then, he instructs the voter to vote for $\mathsf{Cand}_{\mathcal{C}}$, use $y$ and $a$ in the non-interactive proof, and return with the OR-proof until $\mathcal{T}_{\mathcal{C}} < 2\mathcal{T}$. An obeying voter can finish the proof $\pi_1$ by combining puzzles and obtaining a puzzle for $\mathsf{Delay}(s + ex) = \mathsf{Delay}(z)$, which can be opened in time $\approx \mathcal{T}$. However, a disobeying voter who selected $\mathsf{Cand} \neq \mathsf{Cand}_{\mathcal{C}}$ has to compute another OR-proof to avoid being caught by the coercer, which requires knowing both $x$ and $s$ hidden by the delay function. Hence, cheating requires opening two

values hidden by the delay function $\mathsf{Delay}$ instead of just one (as it was for an obedient voter). Assuming the voter has only one device, she would need time $\mathcal{T}_{\mathcal{V}} \approx 2\mathcal{T}$ to compute both $z$ and $x$, which would not fit the time frame $\mathcal{T}_{\mathcal{C}}$ that the coercer demanded.

A possible modification is for $\mathcal{VD}$ to choose and send the generator $g$ in step 1 of the protocol instead of using a fixed $g$. With this modification, it would even be possible that the zero-knowledge proof of knowledge $\pi_1$ of trapdoor $x$ such that $y = g^x$ was computed non-interactively by $\mathcal{V}$, which would send $y, \pi_1$ and voting option $\mathsf{Cand}$ in step 2, leaving the NIZK OR proof, computed by $\mathcal{VD}$, for the last step 3. Once again, we do not know how a coercer could use blockchains and delay functions to coerce a voter in this modified protocol.

**3.5.3 Proof of Correct Re-Randomization of a Ciphertext.** Muller and Truderung in [26] propose a re-encryption-based protocol similar to the previous two constructions. The difference is that the zero-knowledge proof is not for proving that a ciphertext $\mathtt{C}$ decrypts to a voting option $\mathsf{Cand}$, but for proving that a ciphertext $\mathtt{C}'$ is a correct re-randomization of another ciphertext $\mathtt{C}$. All new coercion attacks we discuss on the two solutions employing commitments and OR proofs also apply to this protocol.

### 3.6 Civitas

Civitas [12] is a modification of the JCJ electronic voting protocol proposed by Juels, Catalano, and Jakobsson [21]. They are considered two of the voting schemes enjoying the strongest level of coercion-resistance. The coercion-evading strategy is based on the fact that a voter can compute and show fake credentials to the coercer, whereas he uses real credentials for the desired vote casting. One of the novelties of Civitas compared to JCJ is how these credentials are generated in a registration phase. Fake and real credentials are indistinguishable because real credentials are computed using a designated verifier technique, which takes as input an ElGamal public designation key $K_{V_E}$ of the voter (different from the voter's registration key used, among others, for authentication purposes). The voter can use the secret key $k_{V_E}$ to compute the (indistinguishable from real) fake credentials. However, if a coercer can force a voter to use a specific public key $K_{V_E}$ without knowing the matching secret trapdoor $k_{V_E}$, then the voter cannot resist coercion.

The situation is similar to the one in Section 3.5.2: even a modification of Civitas where the voter is requested to prove, in zero-knowledge, that he knows the trapdoor $k_{V_E}$ could be vulnerable to new coercion tools based on blockchains, homomorphic delay functions, and tamper-proof tokens. Moreover, these attacks do not seem to contradict Trust Assumption 1 of Civitas: *The adversary cannot simulate a voter during Registration.* On the one hand, the attacks we propose are off-line: the coercer gives $K_{V_E}$ to the voter before Registration starts. On the other hand, coercion involves only the designation keys and not the registration keys (which are the focus of all the discussion about this Assumption 1 in [12]).

### 3.7 Voting Based on Trusted Computing

Smart and Ritter proposed a coercion-resistant protocol [35] based on trusted computations (specifically, the TPM and Direct Anonymous Attestation protocol). It consists of three phases: registration, where the voter has to prove their identity in person; joining, where the voter uses a trusted TPM to receive a certificate confirming eligibility; and signing, where the trusted TPM signs the vote. The authorities re-encrypt the ballot before publishing and send the voter a designated proof of re-encryption. If the voter is coerced and does not want to send the coercer's ballot, she can send a different ballot instead and use her designated key to simulate the re-encryption proof for the coercer.

*(*Attack:Complex; Instr, Timed-CC-PRF, Timed-Enc*)* As a part of the protocol, the voter (not a trusted TPM!) is supposed to generate a fresh Elgamal key pair $(s_v, h_v = g^{s_v})$, which is her designated key. Without $s_v$, the voter cannot simulate a re-encryption proof, which is why it is a crucial component of the coercion-resistance strategy. However, with the new tools, the coercer can give the voter a pre-generated pair $(\mathsf{Delay}(s_v), h_v)$, hidden by the delay function $\mathsf{Delay}$ that cannot be opened before time $T$, and demand the re-encryption proof before time $T$. The voter will have no choice but to obey.

A similar attack applies to a version of BeleniosRF [10] where voters generate their signing keys and register the public part with the registrar. As a side observation, we think an untrusted election authority generating public parameters $pp$ can undetectably modify ballots of this particular version of BeleniosRF.[6]

## 4 Deniable Vote Updates and Re-Voting

Many schemes achieve receipt-freeness or some level of coercion-resistance using deniable re-voting or vote updates, e.g., re-randomization as in Belenios-RF [10].

We now note that most systems either can be analysed in our setting or will have CaI verifiability based on the assumption that a secret key, e.g. a signing kay, is not being leaked to $\mathcal{A}_{\text{ver}}$ or using some trusted party. To see why, suppose that the voting device $\mathcal{VD}$ generates ballot $b$ while a public bulletin board $\mathcal{BB}$ posts ballot $b'$. If $b$ is identical to $b'$, then we are in our setting. Alternatively, if $b$ is different from $b'$, the voter needs to be able to verify her vote somehow. She can do it either by a) being able to link the public ballot on $\mathcal{BB}$ to the ballot from $\mathcal{VD}$, which basically would bring us back to our situation in terms of coercion-resistance because we can consider $b'$ to be the ballot produced by $\mathcal{V}$ and $\mathcal{VD}$. Or b) the voter cannot relate $b$ and $b'$, but then $\mathcal{VD}$ and $\mathcal{EA}$ can cheat and change $b'$ to another contain another vote, unless $\mathcal{V}$ knows some secret that prevents them from this, or alternatively we trust some party.

[6] A dishonest election authority, instead of selecting $z$ randomly from $G_1$, sets $z = g_1^v$ for some $v$ in $\mathsf{Setup}(1^\lambda, 1^k)$. Now, the re-randomization server can compute $X_1^v = (g_1^x)^v = (g_1^v)^x = z^x = Y$ (i.e., the voter's private signing key) and sign any ballot.

One might also suggest that we should allow unobservable re-voting to prevent coercion attacks. This basically corresponds to the analysis above with $b'$ being empty and hence relies on trust or a secret key.

# 5 New Security Definitions

The attacks presented in this paper have demonstrated that it is necessary to make a more general definition of receipt-freeness and coercion-resistance for the vote-casting phase to take into account the new tools for coercers and vote-buyers. We will first give a game-based definition without the new tools and then introduce these as oracles that can be used by the coercer and voter. A formal definition of cast-as-intended verifiability can be found in App. A.

## 5.1 Vote Casting Phase Coercion-Resistance

We consider Coercion-Resistance for the Vote Casting Phase, $\mathsf{VC\text{-}CR}$, which is a necessary condition for achieving coercion-resistance for the full voting system considering vote submissions from all voters and information leaks from the tally.

In the definition, the coercer, $\mathcal{A}$, can give instructions, $\mathsf{Instr}$, to the voter before vote-casting. Vote-casting is done using a vote-device $\mathcal{VD}$. To be general, this is modeled as an oracle $\mathcal{O}_{\mathsf{state}_{\mathcal{VD}}}\mathcal{VD}$ with a state $\mathsf{state}_{\mathcal{VD}}$ which is updated during the interaction between the voter and device. We assume that the instruction $\mathsf{Instr}$ uniquely defines an algorithm $\mathcal{V}_{\mathsf{Instr}}^{\mathcal{O}_{\mathsf{state}_{\mathcal{VD}}}\mathcal{VD}}$ which models what the voter does when completely following the instructions of the adversary. The adversary has to distinguish the output from this compared to the case where the voter casts her own vote using some coercion-evasion strategy, denoted $\mathcal{V}$, which we will assume is public, normally given as part of the voting scheme. In both cases, the voter can output a message $\mathsf{msg}$ to the coercer, which can include the (faked) $\mathtt{View}$ between the voter and the voting device plus auxiliary information such as random coins used by the voter.

We have kept $\mathsf{Instr}$ and $\mathsf{msg}$ very abstract here since they can depend on the voting protocol and values obtained when accessing the new tools. When proving security for a specific protocol and attacker model they can be made more specific to facilitate easier proofs.

As mentioned above the coercer will get access to the ballot $\mathsf{ballot}$ produced by $\mathcal{VD}$ in the end. We get this ballot from the final state of $\mathcal{VD}$ using the algorithm $\mathsf{Vote}$. Finally, we extract the underlying vote enclosed in the ballot using the algorithm $\mathsf{Extract}$. We use this to ensure that the voter following the coercion-evasion strategy really casts the preferred vote $\mathsf{Vote}_{\mathcal{V}}$, and we require that the voter following instructions casts the coercer's choice $\mathsf{Vote}_{\mathsf{coerc}}$, i.e., we do not consider randomisation attacks or forced abstention. The latter is impossible to protect against when the coercer sees the output ballot. The ballot randomisation attacks are interesting but outside the scope of this paper, however, they could be modelled using a similar type of definition where $\mathsf{Vote}_{\mathsf{coerc}} \neq \mathsf{Vote}_{\mathcal{V}}$ up to a bounded probability.

```
$\mathsf{Exp}^{\mathsf{VC\text{-}CR},b}_{\mathcal{A},\mathcal{V}}(\lambda)$

$(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{Setup}(\lambda)$
$\mathsf{state}_{\mathcal{VD}} \leftarrow \mathsf{empty}$
$(\mathsf{Instr},\mathsf{state}_{\mathcal{A}}) \leftarrow \mathcal{A}_1(\mathsf{pk},\mathsf{Vote}_{\mathcal{V}},\mathsf{Vote}_{\mathcal{C}})$
if $b = 0$
    $\mathsf{msg} \leftarrow \mathcal{V}^{\mathcal{O}_{\mathsf{state}_{\mathcal{VD}}}\mathcal{VD}}(\mathsf{pk},\mathsf{Instr},\mathsf{Vote}_{\mathcal{V}})$
    $\mathsf{ballot} \leftarrow \mathsf{Vote}(\mathsf{state}_{\mathcal{VD}})$
    $\mathsf{Promise\ Extract}(\mathsf{ballot},\mathsf{sk}) = \mathsf{Vote}_{\mathcal{V}}$
if $b = 1$
    $\mathsf{msg} \leftarrow \mathcal{V}_{\mathsf{Instr}}^{\mathcal{O}_{\mathsf{state}_{\mathcal{VD}}}\mathcal{VD}}(\mathsf{pk})$
    $\mathsf{ballot} \leftarrow \mathsf{Vote}(\mathsf{state}_{\mathcal{VD}})$
    $\mathsf{Require\ Extract}(\mathsf{ballot},\mathsf{sk}) = \mathsf{Vote}_{\mathcal{C}}$
$b' \leftarrow \mathcal{A}_2(\mathsf{pk},\mathsf{msg},\mathsf{ballot},\mathsf{state}_{\mathcal{A}},\mathsf{Vote}_{\mathcal{V}},\mathsf{Vote}_{\mathcal{C}})$
return $b' = b$
```

```
$\mathcal{O}_{\mathsf{state}_{\mathcal{VD}}}\mathcal{VD}(m)$

$(\mathsf{state}_{\mathcal{VD}}, m_{out}) \leftarrow \mathsf{VoteDev}(\mathsf{pk},\mathsf{state}_{\mathcal{VD}}, m)$
return $m_{out}$
```

**Fig. 4.** The experiment for Coercion-Resistance for the Vote Casting Phase, VC-CR.

We use abbreviations for the constraints in the game code on the vote choices using Require · which stands for ‘ if not · then Stop with $\perp$ ’ and Promise · which stands for ‘ if not · then Stop with $\top$ ’.

**Definition 1 (Vote Casting Phase Coercion-Resistance).** *The protocol Vote enjoys Coercion-Resistance for the Vote Casting Phase,* VC-CR*, if there exists a PPT voter algorithm $\mathcal{V}$ such that for all vote choices $\mathsf{Vote}_{\mathcal{V}} \neq \mathsf{Vote}_{\mathcal{C}}$ and for any polynomial-time adversary $\mathcal{A}$ we have that*

$$\mathsf{Adv}^{\mathsf{vc\text{-}cr}}_{\mathcal{A}}(\lambda) = \left|\Pr\left[\mathsf{Exp}^{\mathsf{VC\text{-}CR},0}_{\mathcal{A},\mathcal{V}}(\lambda)\right] - \Pr\left[\mathsf{Exp}^{\mathsf{VC\text{-}CR},1}_{\mathcal{A},\mathcal{V}}(\lambda)\right]\right|$$

*is a negligible function of the security parameter $\lambda$.*

In the definition of VC-CR we state that there exists an overall voter coercion-evasion algorithm $\mathcal{V}$ that works for all coercers (by taking as input the coercer's instructions). A natural alternative is a definition that turns the $\exists\forall$ into a $\forall\exists$ by saying that for all coercer algorithms $\mathcal{A}$ there exists a voter mitigation strategy $\mathcal{V}$. Indeed, this is weaker since it would be implied by VC-CR. Hence, we denote it weak coercion-resistance Weak-VC-CR and will keep VC-CR as our preferred definition. This is because the usability is much better if there exists a general public coercion-evasion strategy, whereas for Weak-VC-CR it is not clear how the voter can effectively find the correct strategy, and might even need knowledge about the coercer's intention.

We can also consider other natural variations of the definition

– Minimal Instructions, VC-CR(Min-Instr): Here the coercer just needs to output the desired vote $\mathsf{Vote}_{\mathcal{C}}$ as instruction.

- Known coerced vote, $\mathsf{VC\text{-}CR(Known\text{-}Vote)}$: Note that $\mathsf{Vote}_{\mathsf{coerc}}$ is not explicitly given to the voter algorithm $\mathcal{V}$ to model that the voter might just get some instructions without knowing what the desired vote of the coercer is. A weaker definition can be made where $\mathsf{Vote}_{\mathsf{coerc}}$ is known to the voter.
- No secret in instruction, $\mathsf{VC\text{-}CR(No\text{-}Secret\text{-}in\text{-}Instr)}$: In this case it is not necessary for the coercer to keep secrets from the coerced voter. We model this as $\mathcal{A}_1$ being a deterministic algorithm using some random tape $r$ and which is included as part of the instructions $\mathsf{Instr}$ together with the algorithm $\mathcal{A}_1$. In this case $\mathsf{state}_{\mathcal{A}}$ used by $\mathcal{A}_2$ can just be $\mathsf{Instr}$.
- No secret for classifier, $\mathsf{VC\text{-}CR(No\text{-}Secret\text{-}in\text{-}Classifier)}$: We can make a weaker definition where the coercer does not need to remember a secret used in the instructions to classify whether the coerced voter follows instructions or not. We model this by not giving the $\mathsf{state}_{\mathcal{A}}$ to the algorithm $\mathcal{A}_2$ where adversary decides which world he is in, but only the instructions $\mathsf{Instr}$. This means the verification could be done by any party just knowing the instructions from the coercer. We denote this $\mathsf{VC\text{-}CR(No\text{-}Secret\text{-}in\text{-}Classifier)}$, and clearly $\mathsf{VC\text{-}CR}$ implies $\mathsf{VC\text{-}CR(No\text{-}Secret\text{-}in\text{-}Classifier)}$.
- We can go further and also remove $\mathsf{Instr}$ from the distinguisher $\mathcal{A}_2$ to achieve an even weaker definition. We denote this $\mathsf{VC\text{-}CR(Minimal\text{-}Classifier)}$
- Note that we have given the coercer access to both the coercer's choice of vote, and the desired vote of the voter in order to make sure that the definition is not too weak in the sense that we would not want coercion-resistance to rely on the coercer not being able to guess or somehow know $\mathsf{Vote}_{\mathcal{V}}$. A weaker version of the definitions above can be made where the adversary does not have access to $\mathsf{Vote}_{\mathcal{V}}$. We denote this with $\mathsf{Unknown\text{-}Intent}$
- Finally, we note that we can combine the above definitions. E.g. we can have $\mathsf{VC\text{-}CR(Known\text{-}Vote, No\text{-}Secret\text{-}in\text{-}Instr, Minimal\text{-}Classifier)}$ for the case where the voter knows the coercer's vote choice, there are no secrets in the instruction, and no secret or instructions used in the classifier.

As we have seen in the first part of this paper (attacks), the coercer might force the voter to use some cryptographic means for controlling the voter's behavior without being present during the voting. For example, the coercer might ask the voter to commit values on the blockchain or utilize TPM for keypair generation. Thus, definitions for coercion-resistance (and for receipt-freeness) must consider this possibility. The way of doing this is by slightly modifying all the security games as follows: both the coercer and the voter will have access to some subset $\mathcal{O}$ of the family of oracles listed in the following sub-section.

### 5.1.1 Oracles for the Coercer's Toolbox

- $\mathcal{O}_{\mathsf{Clock}}$ - an oracle that upon **RequestTime** gives the value of the global time clock [8,17]. No one can alter the global time. In practice this could e.g. be instantiated from blockchains, see for instance [23].
- $\mathcal{O}_{\mathsf{CC}}$ - an oracle that captures an append-only immutable chain of commitments. It starts with an empty list $\mathbf{List}_0$ and supports two functions

**Commit**$(x)$ and **Return**. The function **Commit**$(x)$, upon being called an $i$-th time with the input $x$, appends $x$ to the list $\mathbf{List}_i \leftarrow \mathbf{List}_{i-1} \cup \{x\}$. The function **Return**, upon being called an $i$-th time, returns the list $\mathbf{List}_i$. No one can remove or modify the value from the internal state after it has been committed. In practice, it can be realized with a blockchain.

– $\mathcal{O}_{\mathsf{CC\text{-}PRF}}$ - an oracle that captures an append-only immutable chain of commitments with the possibility to request pseudo-random values. In other words, it supports not only **Commit**$(x)$ and **Return** as $\mathcal{O}_{\mathsf{CC}}$, but also a function **PRF**, which upon being called an $i$-th time, derives and outputs a pseudorandom value $y_i$ from the internal state of the oracle $\mathbf{List}_i$. No one can remove or modify the value from the internal state after it has been committed, nor obtain the pseudorandom value different from the derived from the internal state value.

– $\mathcal{O}_{\mathsf{Timed\text{-}CC}}$ - is an oracle that captures an append-only immutable chain of commitments with time stamps. In other words, it supports **Return** function as $\mathcal{O}_{\mathsf{CC}}$ and a function **TimedCommit**$(x)$, upon being called an $i$-th time with the input $x$, calls the function **RequestTime** of $\mathcal{O}_{\mathsf{Clock}}$ oracle to get timestamp $t$ and then calls **Commit**$(\{x||t\})$ of the $\mathcal{O}_{\mathsf{CC}}$ oracle. No one can remove or modify the value from the internal state after it has been committed or alter the timestamp.

– $\mathcal{O}_{\mathsf{Timed\text{-}CC\text{-}PRF}}$: - is an oracle that captures an append-only immutable chain of commitments with time stamps with the possibility to request pseudorandom values. In other words, it supports not only **TimedCommit**$(x)$ and **Return** as $\mathcal{O}_{\mathsf{Timed\text{-}CC}}$, but also a function **PRF**, which upon being called an $i$-th time, derives and outputs a pseudorandom value $y_i$ from the internal state of the oracle $\mathbf{List}_i$. No one can remove or modify the value from the internal state after it has been committed, nor obtain the pseudorandom value different from the derived from the internal state value.

– $\mathcal{O}_{\mathsf{Timed\text{-}Enc}}$ - an oracle that captures functionalities of secure storage that does not release the secret until time $\mathcal{T}$ (according to oracle $\mathcal{O}_{\mathsf{Clock}}$) has passed. It supports function $\mathsf{Delay}(x, \mathcal{T})$, which takes as input secret $x$ and desired time $\mathcal{T}$ and outputs a puzzle $Z$. No one can open $Z$ until **RequestTime** from $\mathcal{O}_{\mathsf{Clock}}$ returns time, which is greater or equal to $\mathcal{T}$.

– $\mathcal{O}_{\mathsf{Token}}$ - an oracle that captures functionalities of tamper-proof secret storage and outputs the result of a pre-defined cryptographic operation. It supports two functions: **Init**$(x)$ and **EvaluateFunction**$(F)$. The function **Init**$(x)$ can only be run once; it receives the input $x$ and, if and only if the initial state is empty, initializes the internal state with $x$. The function **EvaluateFunction**$(F)$ receives a one-way function $F$ as an input and outputs the $F(x)$. No one can access the state or alter it after it is set.

– $\mathcal{O}_{\mathsf{TPM}}$ - an oracle that captures functionalities of a trusted platform module. The exact interface depends on the particular emulated module, but it is expected to run any coercer's program without leaking an internal secret.

### 5.1.2 Implications and Separations between Different CR Notions

On the one hand, we can now state some simple reductions between these definitions, considering access to the same subset of oracles.

- $\mathsf{VC\text{-}CR}$ implies $\mathsf{VC\text{-}CR(Known\text{-}Vote)}$ which, in turn, implies $\mathsf{VC\text{-}CR(Min\text{-}Instr)}$
- $\mathsf{VC\text{-}CR}$ implies $\mathsf{VC\text{-}CR(No\text{-}Secret\text{-}in\text{-}Classifier)}$ which, in turn, implies respectively $\mathsf{VC\text{-}CR(No\text{-}Secret\text{-}in\text{-}Instr)}$ and $\mathsf{VC\text{-}CR(Minimal\text{-}Classifier)}$ both of which finally implies $\mathsf{VC\text{-}CR(Min\text{-}Instr)}$.

**Comment 1** *If a voting protocol fulfils cast-as-intended verifiability as in Def. 3, and we only have a small number of candidate choices, one might presume that* $\mathsf{VC\text{-}CR(Known\text{-}Vote)}$ *implies* $\mathsf{VC\text{-}CR}$*, since one could use the cast-as-intended verification mechanism to discover the vote the coercer wants. Unfortunately, cast-as-intended only ensures that a voter following protocol instructions enjoys verifiability. However, a coercer might issue instructions which prevents the coerced voter from verifying her vote without the coercer detecting this.*

On the other hand, some protocols can be used to show that there are strict separations among (some of) the CR notions defined in Section 5.1.

- Both $\mathsf{VC\text{-}CR(Min\text{-}Instr)} \not\Rightarrow \mathsf{VC\text{-}CR(No\text{-}Secret\text{-}in\text{-}Instr)}$ and the statement $\mathsf{VC\text{-}CR(Min\text{-}Instr)} \not\Rightarrow \mathsf{VC\text{-}CR(Known\text{-}Vote)}$ can be proven by considering the interactive 4-rounds protocol where (1) $\mathcal{V}$ sends the chosen vote, (2) $\mathcal{VD}$ encrypts the vote in ciphertext $C$ and sends the first message $a$ of a Sigma protocol to prove that $C$ encrypts the chosen vote, (3) $\mathcal{V}$ sends a challenge $c$ for this Sigma protocol, (4) $\mathcal{VD}$ sends the last message $z$ of the Sigma protocol.
  This protocol enjoys $\mathsf{VC\text{-}CR(Min\text{-}Instr)}$, but as long as the coercer can use as $\mathsf{Instr}$ not only its chosen voting option, but also the fact that the challenge $c$ is a deterministic function of the first message $a$, that is $c = f(a)$, it is not known how a voter can cheat such a coercer.
- $\mathsf{VC\text{-}CR(No\text{-}Secret\text{-}in\text{-}Classifier)} \not\Rightarrow \mathsf{VC\text{-}CR}$ can be proven with a protocol where (1) $\mathcal{V}$ sends the chosen vote and a public key $\mathsf{pk}'$ different to the public key $\mathsf{pk}$ of the election, (2) $\mathcal{VD}$ encrypts twice the vote, once with $\mathsf{pk}$ and one with $\mathsf{pk}'$, and adds a non-interactive zero-knowledge proof that the two ciphertexts encrypt the same plaintext.
  A coercer who forgets the secrets used in the instructions can be cheated by the voter, who can present two ciphertexts of the same voting option (the one chosen by the voter, instead of the one chosen by the coercer). However, if the coerced remembers the secret key $\mathsf{sk}'$, it cannot cheated in this way (and in no way) because it can use $\mathsf{sk}'$ to check that one of the ciphertexts encrypts its chosen option.

### 5.2 Receipt-Freeness

We can now define receipt-freeness at vote-casting time, in contrast to other definitions of receipt-freeness which are more global and take into account all

```
$\mathsf{Exp}^{\mathsf{VC\text{-}RF},b}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}(\lambda)$
$(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{Setup}(\lambda)$
$\mathsf{state}_{\mathcal{VD}} \leftarrow \mathsf{empty}$
$\textbf{if } b = 0$
    $\mathsf{msg} \leftarrow \mathcal{V}^{\mathcal{O}_{\mathsf{state}_{\mathcal{VD}}}{}^{\mathcal{VD}}}(\mathsf{pk},\mathsf{Vote}_{\mathsf{sell}})$
    $\mathsf{ballot} \leftarrow \mathsf{Vote}(\mathsf{state}_{\mathcal{VD}})$
    $\mathsf{Require\ Extract}(\mathsf{ballot},\mathsf{sk}) = \mathsf{Vote}_{\mathsf{sell}}$
$\textbf{if } b = 1$
    $\mathsf{msg} \leftarrow \mathsf{Sim}_{\mathcal{V}}^{\mathcal{O}_{\mathsf{state}_{\mathcal{VD}}}{}^{\mathcal{VD}}}(\mathsf{pk},\mathsf{Vote}_{\mathsf{own}},\mathsf{Vote}_{\mathsf{sell}})$
    $\mathsf{ballot} \leftarrow \mathsf{Vote}(\mathsf{state}_{\mathcal{VD}})$
    $\mathsf{Promise\ Extract}(\mathsf{ballot},\mathsf{sk}) = \mathsf{Vote}_{\mathsf{own}}$
$b' \leftarrow \mathcal{A}(\mathsf{pk},\mathsf{msg},\mathsf{ballot},\mathsf{Vote}_{\mathsf{own}},\mathsf{Vote}_{\mathsf{sell}})$
$\textbf{return } b' = b$
```

**Fig. 5.** The experiment for Receipt-Freeness for the Vote Casting Phase, VC-RF. For VC-RF the randomness in parenthesis (rand) is not shared between the voter algorithm $\mathcal{V}$ and simulator Sim whereas for VC-RF-Rand it is.

the phases of a voting system (see for instance [20]). The point of our definition is that for any (malicious) voter algorithm $\mathcal{V}$ trying to obtain a receipt in the form of some information msg for her vote $\mathsf{Vote}_{\mathsf{sell}}$, there exists a simulator that casts a vote for another choice $\mathsf{Vote}_{\mathsf{own}}$ but gives information msg which is indistinguishable from the claimed receipt. It models that the voter tries to cheat a coercer or vote buyer by voting for another vote option. In principle, the simulator could have access to both the algorithm $\mathcal{V}$ and to the random tape rand used. This is because the voter and the simulator are essentially the same person and cannot hide information it uses from itself (which would make it easier to obtain a receipt). As an example, if the voter creates a designated verifier key and claims to delete it, the adversary cannot be sure whether the voter still knows it and uses it to create a fake proof.

**Definition 2 (Vote Casting Phase Receipt-Freness).** *The protocol Vote enjoys Receipt-Freeness for the Vote Casting Phase,* VC-RF*, if there exists a simulator Sim such that for all vote choices $\mathsf{Vote}_{\mathsf{sell}} \neq \mathsf{Vote}_{\mathsf{own}}$, for all PPT algorithms $\mathcal{V}$ (corresponding to a malicious voter trying to obtain an receipt) and for all polynomial-time adversaries $\mathcal{A}$ we have that*

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

*is a negligible function of the security parameter $\lambda$.*

Again we can define a weaker variant, denoted Weak-VC-RF , as for coercion-resistance, which involves the same security experiment, but has an $\forall\exists$ definition, i.e. of the form: for all PPT algorithms $\mathcal{V}$ there exists a simulator Sim.

In most cases receipt-freeness can be achieved without the simulator having access to the random tape of the voter since we can achieve a computational indistinguishability without it. We hence defined $\mathsf{VC\text{-}RF}$ without access to the randomness, but an alternative definition $\mathsf{VC\text{-}RF\text{-}Rand}$ can be defined where the simulator has access to the random tape of the voter. $\mathsf{VC\text{-}RF}$ clearly implies $\mathsf{VC\text{-}RF\text{-}Rand}$ since the simulator in the latter has more information.

Note that we have also given the attacker access to $\mathsf{Vote}_{\mathsf{own}}$ when trying to distinguish, similar to our coercion definition. This models that the vote-buyer might know this choice. In case of high-entropy vote choices and where the vote buyer might have less knowledge about the vote seller this could be changed.

Finally, we here present a theorem and informal proof to demonstrate the usefulness of the different definitions.

**Theorem 1.** *Receipt-freeness* $\mathsf{VC\text{-}RF}$ *implies the constrained coercion-resistance* $\mathsf{VC\text{-}CR}(\mathsf{Known\text{-}Vote}, \mathsf{No\text{-}Secret\text{-}in\text{-}Instr}, \mathsf{Minimal\text{-}Classifier})$ *and with access to a timed commitment oracle* $\mathcal{O}_{\mathsf{Timed\text{-}CC}}$ *(see Sec. 5.1.1) they are equivalent.*

The proof follows by relating the simulator, $\mathsf{Sim}$, in $\mathsf{VC\text{-}RF}$ to the coercion-mitigation algorithm $\mathcal{V}$ in $\mathsf{VC\text{-}CR}$, and correspondingly relate the voter algorithm $\mathcal{V}$ in $\mathsf{VC\text{-}RF}$ to $\mathcal{V}_{\mathsf{Instr}}$ in $\mathsf{VC\text{-}CR}$. The constrained form of $\mathsf{VC\text{-}CR}$ ensures that this mapping can be done and the advantages will be the same in the two experiments. We need the timed commitment oracle since in the coercion game, the adversary gives instructions before voting. However, in the vote-seller experiment the output message is produced after voting which could allow the vote-seller to choose a favourable coercion instruction after voting which fits with the view of the voter interaction.

*Proof (sketch).* As a first step, we rewrite the receipt-freeness experiment by swapping $b = 0, 1$, renaming $\mathsf{Vote}_{\mathsf{sell}}$ as $\mathsf{Vote}_{\mathcal{C}}$ and $\mathsf{Vote}_{\mathsf{own}}$ as $\mathsf{Vote}_{\mathcal{V}}$. We also denote the malicious voter algorithm as $\mathcal{V}_{RF}$. This gives the hybrid $\mathsf{Exp}^{\mathsf{hybrid}_0,b}_{\mathcal{A},\mathcal{V}_{RF},\mathsf{Sim}}(\lambda)$ shown in Fig. 6. For comparison, the experiment for the coercion-resistance $\mathsf{Exp}^{\mathsf{VC\text{-}CR}(\mathsf{Known\text{-}Vote},\mathsf{No\text{-}Secret\text{-}in\text{-}Instr},\mathsf{Minimal\text{-}Classifier}),b}_{\mathcal{A},\mathcal{V}}(\lambda)$ is displayed as well where it is made explicit that the $\mathsf{Instr}$ contains $\mathcal{A}_1$, its random coins, the coercer's vote choice and some auxiliary data $\mathsf{aux}$.

We will first prove that the coercion-resistance variant implies receipt-freeness. Accordingly, we assume there exists an attack on $\mathsf{Exp}^{\mathsf{hybrid}_0,b}_{\mathcal{A},\mathcal{V}_{RF},\mathsf{Sim}}(\lambda)$. Now for the coercion-resistance we assume a given coercion-mitigation algorithm, $\mathcal{V}$, and a given key setup. For simplicity, we assume $\mathcal{V}$ always produces a correct ballot for $\mathsf{Vote}_{\mathcal{V}}$ as demanded by the promise statement. We can now feed this as the public key setup and simulator, $\mathsf{Sim}_{\mathcal{V}_{RF}}$, into the attacker on the hybrid $\mathsf{VC\text{-}RF}$. Here $\mathcal{V}_{RF}$ and $\mathsf{Vote}_{\mathcal{C}}$ in $\mathsf{Sim}$ are given to $\mathcal{V}$ by embedding them into $\mathsf{Instr}$. Now we get a distinguisher algorithm $\mathcal{A}$ whose output we will use as the output of $\mathcal{A}_2$. We will assume that the attack also outputs the algorithm $\mathcal{V}_{RF}$. We then define $\mathcal{A}_1$ to output $\mathsf{Vote}_{\mathcal{C}}$ and the description of $\mathcal{V}_{RF}$. No randomness is needed for this and hence does not have to be output. Now $\mathcal{V}$ and $\mathsf{Sim}$ are equivalent in the two games, and $\mathcal{V}_{RF}$ can also be used in the $\mathsf{VC\text{-}CR}$ game as $\mathcal{V}_{\mathsf{Instr}}$ includ-

```
$\mathsf{Exp}^{\mathsf{hybrid}_0,b}_{\mathcal{A},\mathcal{V}_{RF},\mathsf{Sim}}(\lambda)$

$(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{Setup}(\lambda)$
$\mathsf{state}_{\mathcal{VD}} \leftarrow \mathsf{empty}$
if $b = 0$
    $\mathsf{msg} \leftarrow \mathsf{Sim}_{\mathcal{V}_{RF}}^{\mathcal{O}_{\mathsf{state}_{\mathcal{VD}}}{}^{\mathcal{VD}}}(\mathsf{pk},\mathsf{Vote}_{\mathcal{V}},\mathsf{Vote}_{\mathcal{C}})$
    $\mathsf{ballot} \leftarrow \mathsf{Vote}(\mathsf{state}_{\mathcal{VD}})$
    $\mathsf{Promise\ Extract}(\mathsf{ballot},\mathsf{sk}) = \mathsf{Vote}_{\mathcal{V}}$
if $b = 1$
    $\mathsf{msg} \leftarrow \mathcal{V}_{RF}^{\mathcal{O}_{\mathsf{state}_{\mathcal{VD}}}{}^{\mathcal{VD}}}(\mathsf{pk},\mathsf{Vote}_{\mathcal{C}})$
    $\mathsf{ballot} \leftarrow \mathsf{Vote}(\mathsf{state}_{\mathcal{VD}})$
    $\mathsf{Require\ Extract}(\mathsf{ballot},\mathsf{sk}) = \mathsf{Vote}_{\mathcal{C}}$
$b' \leftarrow \mathcal{A}(\mathsf{pk},\mathsf{msg},\mathsf{ballot},\mathsf{Vote}_{\mathcal{V}},\mathsf{Vote}_{\mathcal{C}})$
return $b' = b$
```

```
$\mathsf{Exp}^{\mathsf{VC\text{-}CR(Known\text{-}Vote,No\text{-}Secret\text{-}in\text{-}Instr,Minimal\text{-}Classifier)},b}_{\mathcal{A},\mathcal{V}}(\lambda)$

$(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{Setup}(\lambda)$
$\mathsf{state}_{\mathcal{VD}} \leftarrow \mathsf{empty}$
$\mathsf{Instr} = (\mathsf{rand},\mathcal{A}_1,\mathsf{Vote}_{\mathcal{C}},\mathsf{aux})$
    $\leftarrow \mathcal{A}_1(\mathsf{pk},\mathsf{Vote}_{\mathcal{V}},\mathsf{Vote}_{\mathcal{C}},\mathsf{rand})$
if $b = 0$
    $\mathsf{msg} \leftarrow \mathcal{V}^{\mathcal{O}_{\mathsf{state}_{\mathcal{VD}}}{}^{\mathcal{VD}}}(\mathsf{pk},\mathsf{Instr},\mathsf{Vote}_{\mathcal{V}})$
    $\mathsf{ballot} \leftarrow \mathsf{Vote}(\mathsf{state}_{\mathcal{VD}})$
    $\mathsf{Promise\ Extract}(\mathsf{ballot},\mathsf{sk}) = \mathsf{Vote}_{\mathcal{V}}$
if $b = 1$
    $\mathsf{msg} \leftarrow \mathcal{V}_{\mathsf{Instr}}^{\mathcal{O}_{\mathsf{state}_{\mathcal{VD}}}{}^{\mathcal{VD}}}(\mathsf{pk})$
    $\mathsf{ballot} \leftarrow \mathsf{Vote}(\mathsf{state}_{\mathcal{VD}})$
    $\mathsf{Require\ Extract}(\mathsf{ballot},\mathsf{sk}) = \mathsf{Vote}_{\mathcal{C}}$
$b' \leftarrow \mathcal{A}_2(\mathsf{pk},\mathsf{msg},\mathsf{ballot},\mathsf{Vote}_{\mathcal{V}},\mathsf{Vote}_{\mathcal{C}})$
return $b' = b$
```

**Fig. 6.** The hybrid experiment $\mathsf{Exp}^{\mathsf{hybrid}_0}_{\mathcal{A},\mathcal{V}_{\mathsf{Instr}},\mathsf{Sim}}$ together with the variant of VC-CR.

ing knowledge of $\mathsf{Vote}_{\mathcal{C}}$, consistent with how Sim was defined. Thus, we have constructed an adversary with the same advantage as for VC-RF.

We now prove that receipt-freeness implies the constrained coercion-resistance using a timed commitment oracle. Thus, we assume that we have an attack on the constrained VC-CR definition (this also holds if this also has access to the timed commitment oracle). From the VC-RF experiment we can then get a key setup and choose a general simulator, $\mathsf{Sim}_{\mathcal{V}_{RF}}$. The simulator fulfills the form for $\mathcal{V}$ in VC-CR since $\mathsf{Vote}_{\mathcal{C}}$ is included in Instr and we use $\mathcal{V}_{\mathsf{Instr}}$ as the $\mathcal{V}_{RF}$ input to $\mathsf{Sim}_{\mathcal{V}_{RF}}$. Hence we can get Instr from $\mathcal{A}_1$ and thus $\mathcal{V}_{\mathsf{Instr}}$, which has a non-negligible advantage in winning the VC-CR game when using the output of $\mathcal{A}_2$. We can now let $\mathcal{V}_{RF}$ be $\mathcal{V}_{\mathsf{Instr}}$ and for the distinguisher $\mathcal{A}$ we use the output of $\mathcal{A}_2$. However, to be able to win the game on the VC-CR side we extend $\mathcal{V}_{RF}$ to start by using the timed commitment oracle and commit to Instr which includes $\mathcal{A}_1, \mathsf{rand}$. We will assume that it is clear that this happens before voting as in the VC-CR definition, e.g. by doing this early enough. Finally, $\mathcal{A}_2$ will also check that the commitment has been made, to enforce that Sim also calls $\mathcal{V}_{RF}$ to do this, otherwise it is clear that $b = 0$. If this test passes, it uses the output from $\mathcal{A}_2$. The winning probability is going to be the same even though Sim is itself creating the instructions and hence knows all secrets there in contrast to the general VC-CR definition. This is the reason we use the No-Secret-in-Instr where the simulator $\mathcal{V}$ also gets access to $\mathcal{A}_1$ and its random tape and hence could have created Instr itself. □

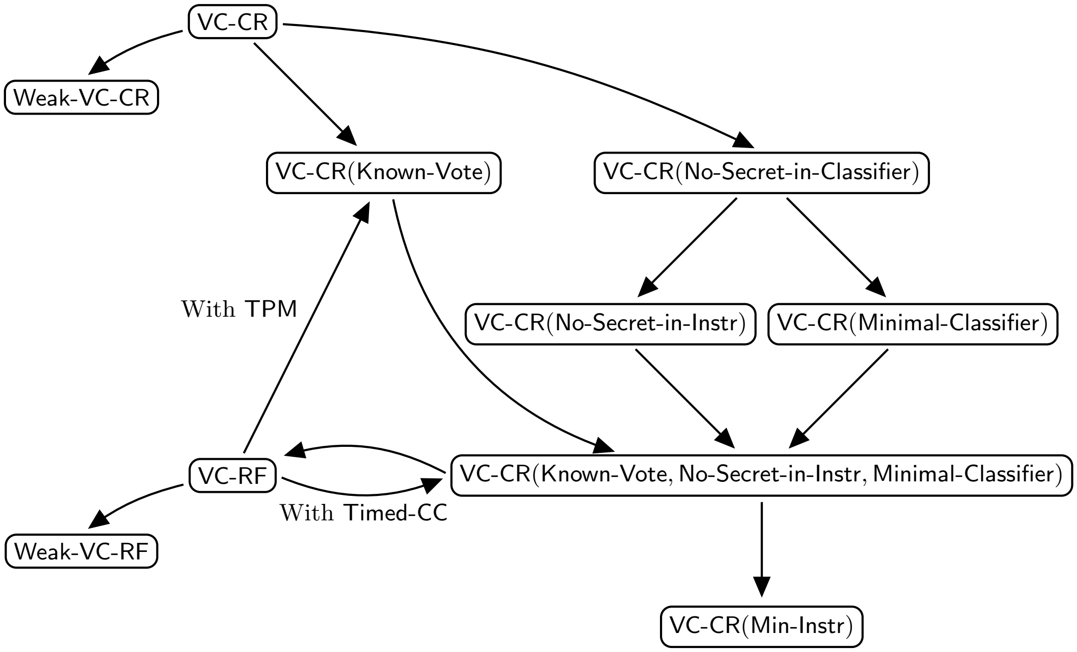
VC-CR
Weak-VC-CR
VC-CR(Known-Vote)
VC-CR(No-Secret-in-Classifier)
With TPM
VC-CR(No-Secret-in-Instr)
VC-CR(Minimal-Classifier)
VC-RF
VC-CR(Known-Vote, No-Secret-in-Instr, Minimal-Classifier)
With Timed-CC
Weak-VC-RF
VC-CR(Min-Instr)

**Fig. 7.** Implications between the new security definitions

We also conjecture that VC-RF implies VC-CR(Known-Vote) with access to the trusted hardware module $\mathcal{O}_{\mathsf{TPM}}$, since the vote seller can commit to a coercion instruction, let the module output the coercer instructions and in the end output all secrets to the distinguisher including an attestation of what it was running.

In Fig. 7 we present the relations between the security definitions in a diagram.

## 6 Conclusion and Future Work

We studied previously unexplored coercion and vote-selling attacks based on new cryptographic primitives such as blockchains, delay functions, time-lock encryption, etc. Our investigation showed many examples of how the coercer can force voters to comply with his demands by relying on those new tools. We described some of the possible attacks and sketched others. Since some successful coercion attacks occur on voting schemes that were supposed/claimed/proved to be coercion resistance or receipt-free, the main conclusion of the first part of the work was that the coercion models should be re-evaluated, and new definitions were required. Such definitions, that are presented in Section 5, lead to some interesting lines of future work; we list below three impossibility results that (we believe) could be proved to hold in our setting (this is ongoing work).

1) We cannot have receipt-free voting without a trusted $\mathcal{VD}$ in our model. This is because the voter could otherwise prove how the output ballot (which can be seen by the adversary in our model) was created. We thus need $\mathcal{VD}$ to generate the ballot obliviously to the voter.

2) If the coercer $\mathcal{C}$ directly interacts with a voter during vote casting in a system satisfying CAI then it cannot be coercion-resistant. This is because $\mathcal{C}$ can use CAI to verify the cast vote with the voter acting as a proxy. If the voter could cheat the attacker here, then the voting device could use the voter's strategy to cheat the voter and break CAI verifiability.

3) No scheme can be coercion-resistant if the coercer uses $\mathsf{Token}$. This follows from the result above since the coercer can simply let the $\mathsf{Token}$ simulate the direct interaction.

## References

1. On-chain vote buying and the rise of dark daos. https://hackingdistributed.com/2018/07/02/on-chain-vote-buying/, (Accessed on 12/13/2023)
2. Adida, B.: Helios: Web-based Open-Audit Voting. In: USENIX Security'0817th USENIX Security Symposium. pp. 335–348 (2008)
3. Baum, C., Chiang, J.H.y., David, B., Frederiksen, T.K.: Eagle: Efficient privacy preserving smart contracts. In: International Conference on Financial Cryptography and Data Security. pp. 270–288. Springer (2023)
4. Benaloh, J.: Simple Verifiable Elections. In: EVT'06 Electronic Voting Technology Workshop (2006)
5. Benaloh, J., Byrne, M., Kortum, P., McBurnett, N., Pereira, O., Stark, P.B., Wallach, D.S.: Star-vote: A secure, transparent, auditable, and reliable voting system. arXiv preprint arXiv:1211.1904 (2012)
6. Bougon, M., Chabanne, H., Cortier, V., Debant, A., Dottax, E., Dreier, J., Gaudry, P., Turuani, M.: Themis: an on-site voting system with systematic cast-as-intended verification and partial accountability. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 397–410 (2022)
7. Boyd, C., Haines, T., Rønne, P.B.: Vote selling resistant voting. In: Financial Cryptography and Data Security: FC 2020 International Workshops, AsiaUSEC, CoDeFi, VOTING, and WTSC, Kota Kinabalu, Malaysia, February 14, 2020, Revised Selected Papers 24. pp. 345–359. Springer (2020)
8. Brihaye, T., Laroussinie, F., Markey, N., Oreiby, G.: Timed concurrent game structures. In: International Conference on Concurrency Theory. pp. 445–459. Springer (2007)
9. Burdges, J., De Feo, L.: Delay encryption. In: Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I. pp. 302–326. Springer (2021)
10. Chaidos, P., Cortier, V., Fuchsbauer, G., Galindo, D.: BeleniosRF: A non-interactive receipt-free electronic voting scheme. In: Proceedings of CCS'2016. pp. 1614–1625 (2016)
11. Chaum, D., Florescu, A., Nandi, M., Popoveniuc, S., Rubio, J., Vora, P.L., Zagórski, F.: Paperless independently-verifiable voting. In: International Conference on E-Voting and Identity. pp. 140–157. Springer (2011)

12. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a secure voting system. In: 2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA. pp. 354–368. IEEE Computer Society (2008)
13. Cortier, V., Debant, A., Gaudry, P., Glondu, S.: Belenios with cast as intended. In: Voting 2023 - 8th Workshop on Advances in Secure Electronic Voting, May 2023, Bol, Brac, Croatia. Lecture Notes in Computer Science, Springer (2023), `https://universite-paris-saclay.hal.science/LORIA-ALGO/hal-04020110v1`
14. Cortier, V., Gaudry, P., Glondu, S.: Belenios: a simple private and verifiable electronic voting system. Foundations of Security, Protocols, and Equational Reasoning: Essays Dedicated to Catherine A. Meadows pp. 214–238 (2019)
15. Cortier, V., Gaudry, P., Yang, Q.: Is the jcj voting system really coercion-resistant? Cryptology ePrint Archive (2022)
16. Döttling, N., Hanzlik, L., Magri, B., Wohnig, S.: Mcfly: Verifiable encryption to the future made practical. In: International Conference on Financial Cryptography and Data Security. pp. 252–269. Springer (2023)
17. Eldefrawy, K., Terner, B., Yung, M.: Composing timed cryptographic protocols: Foundations and applications. Cryptology ePrint Archive (2024)
18. Finogina, T., Herranz, J.: On remote electronic voting with both coercion resistance and cast-as-intended verifiability. Journal of Information Security and Applications **76**, 103554 (2023)
19. Finogina, T., Herranz, J., Larraia, E.: How (not) to achieve both coercion resistance and cast as intended verifiability in remote evoting. In: International Conference on Cryptology and Network Security. pp. 483–491. Springer (2021)
20. Fraser, A., Quaglia, E.A., Smyth, B.: A critique of game-based definitions of receipt-freeness for voting. In: Provable Security: 13th International Conference, ProvSec 2019, Cairns, QLD, Australia, October 1–4, 2019, Proceedings 13. pp. 189–205. Springer (2019)
21. Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant electronic elections. In: Proceedings of the 2005 ACM workshop on Privacy in the electronic society. pp. 61–70 (2005)
22. Kelsey, J., Regenscheid, A., Moran, T., Chaum, D.: Attacking paper-based e2e voting systems. In: Towards Trustworthy Elections: New Directions in Electronic Voting, pp. 370–387. Springer (2010)
23. Liu, J., Jager, T., Kakvi, S.A., Warinschi, B.: How to build time-lock encryption. Designs, Codes and Cryptography **86**, 2549–2586 (2018)
24. Malavolta, G., Thyagarajan, S.A.K.: Homomorphic time-lock puzzles and applications. In: Proceedings of CRYPTO 2019 (volume I). pp. 620–649. Springer (2019)
25. McCorry, P., Shahandashti, S.F., Hao, F.: A smart contract for boardroom voting with maximum voter privacy. In: Proceedings of Financial Cryptography Conference 2017. pp. 357–375. Springer (2017)
26. Müller, J., Truderung, T.: Caised: A protocol for cast-as-intended verifiability with a second device. In: International Joint Conference on Electronic Voting. pp. 123–139. Springer (2023)
27. Nasser, Y., Okoye, C., Clark, J., Ryan, P.Y.: Blockchains and voting: Somewhere between hype and a panacea. White Paper (2018)
28. Park, S., Specter, M., Narula, N., Rivest, R.L.: Going from bad to worse: from internet voting to blockchain voting. Journal of Cybersecurity **7**(1) (2021)
29. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Proceedings of CRYPTO 1991
30. Pereira, O., Adida, B., de Marneffe, O.: Bringing open audit elections into practice: Real world uses of helios. swiss e-voting workshop (2010)

31. Popoveniuc, S., Hosp, B.: An introduction to punchscan. In: Towards Trustworthy Elections: New Directions in Electronic Voting, pp. 242–259. Springer (2010)
32. Qi, H., Xu, M., Yu, D., Cheng, X.: Sok: Privacy-preserving smart contract. High-Confidence Computing p. 100183 (2023)
33. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
34. Rønne, P.B., Ryan, P.Y., Smyth, B.: Cast-as-intended: A formal definition and case studies. In: Financial Cryptography and Data Security. FC 2021 International Workshops: CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers 25. pp. 251–262. Springer (2021)
35. Smart, M., Ritter, E.: True trustworthy elections: remote electronic voting using trusted computing. In: International Conference on Autonomic and Trusted Computing. pp. 187–202. Springer (2011)

## A Cast-as-Intended Verifiability

We here recall the definition of cast-as-intended from [34]. Here, the voter $\mathcal{V}$ interacts with the adversary $\mathcal{A}$ completely controlling the voting device, $\mathcal{VD}$, to create a ballot. The voter can use $\mathcal{TV}$, which stands for Trusted Verification and can verify the submitted ballots, proofs on the bulletin board etc. In [34], $\mathcal{TV}$ was given as a trusted device, but here it could also be a proxy verifier, etc. If the verification fails, the game will be abandoned. $\mathsf{Vote}$ stands for the vote casting algorithm, and the adversary wins if the cast ballot is not in the image of the vote algorithm using the intended vote.

**Definition 3 (Cast-as-intended).** *Let $\mathcal{V}$, $\mathcal{TV}$, $\mathsf{Vote}$, and $\mathcal{A}$ be probabilistic polynomial-time algorithms, $\kappa$ and $\hat{\kappa}$ be security parameters, and $\mathsf{Exp}^{\mathsf{CaI}}_{\mathcal{A},\mathcal{V},\mathcal{TV},\mathsf{Vote}}(\kappa,\hat{\kappa})$ be the experiment in Fig. 8.*
*We say $\mathcal{V},\mathcal{TV},\mathsf{Vote}$ satisfies $\delta(\hat{\kappa})$-*cast-as-intended*, if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a function $\delta$ and for all security parameters $\kappa$ and $\hat{\kappa}$, we have $\mathsf{Succ}(\mathsf{Exp}^{\mathsf{CaI}}_{\mathcal{A},\mathcal{V},\mathcal{TV},\mathsf{Vote}}(\kappa,\hat{\kappa})) \leq \mathsf{negl}(\hat{\kappa}) + \delta(\hat{\kappa}) + \mathsf{negl}(\kappa)$.*

```
$\mathsf{Exp}^{\mathsf{CaI}}_{\mathcal{A},\mathcal{V},\mathcal{TV},\mathsf{Vote}}(\kappa,\hat{\kappa})$
  $(\mathsf{pk}, v, nc, \mathsf{state}) \leftarrow \mathcal{A}(\kappa,\hat{\kappa})$
  $\mathsf{ballot} \leftarrow \mathcal{V}^{\mathcal{TV}(\mathsf{pk},nc,\kappa),\mathcal{VD}}(v,\hat{\kappa})$
  return $\forall r \,.\, \mathsf{ballot} \neq \mathsf{Vote}[\mathsf{pk}, v, nc, \kappa; r]$
          $\wedge\ \mathsf{ballot} \neq \perp \wedge 1 \leq v \leq nc$
```

```
$\mathcal{O}_{\mathsf{state}}\mathcal{VD}(m)$
  $(\mathsf{state}, m_{out}) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{state}, m)$
  return $m_{out}$
```

**Fig. 8.** The experiment for Cast-as-Intended, $\mathsf{CaI}$.

To make precise what $\mathcal{TV}$ checks in this paper, we will denote the verification algorithm by $\mathsf{Verify}$ and the voters view by $\mathtt{View}$. Then $\mathcal{TV}$ runs $\mathsf{Verify}(\mathtt{View}, b, \mathcal{BB})$ and the experiments aborts if this checks fails.

In most protocols, $\mathsf{Verify}$ would be split into two parts. Firstly, a private part is used by the voter to run a private (cast-as-intended) verification $\mathsf{Verify}_{\mathcal{V}}(\texttt{View}, b)$, which takes the whole $\texttt{View}$ into account but does not access $\mathcal{BB}$. Secondly, a public part (ballot correctness and recorded-as-cast verification) $\mathsf{Verify}_{\text{pub}}(\texttt{View}_{\text{pub}}, \mathcal{BB})$ which takes a subset, or function, of $\texttt{View}$ into account and compares it with data on $\mathcal{BB}$. The latter is often hiding the actual vote choice to allow proxy verification, and the former is kept as simple as possible to increase usability.