Is the JCJ voting system really coercion-resistant?

Véronique Cortier, Pierrick Gaudry, Quentin Yang

Is the JCJ voting system really coercion-resistant?

Véronique Cortier, Pierrick Gaudry, Quentin Yang

Université de Lorraine, Inria, CNRS
April 2022

***Abstract*—Coercion-resistance is a security property of electronic voting, often considered as a must-have for high-stake elections. The JCJ voting scheme, proposed in 2005 by Juels, Catalon and Jakobsson, is still the reference paradigm when designing a coercion-resistant protocol. We highlight a weakness in JCJ that is also present in all the systems following its general structure. This comes from the procedure that precedes the tally, where the trustees remove the ballots that should not be counted. This phase leaks more information than necessary, leading to potential threats for the coerced voters. Fixing this leads to the notion of *cleansing-hiding*, that we apply to form a variant of JCJ that we call CHide. One reason for the problem not being seen before is the fact that the associated formal definition of coercion-resistance was too weak. We therefore propose a definition that can take into accounts more behaviors such as revoting or the addition of fake ballots by authorities. We then prove that CHide is coercion-resistant for this definition, and that JCJ is coercion-resistant for a slightly weakened version of our definition, that models the leakage of information in JCJ.**

***Index Terms*—E-voting, coercion-resistance, JCJ protocol.**

I. Introduction

Internet voting allows to take part into an election without being physically present at a polling station. It can be used for many reasons, such as providing people with low mobility or expatriates with a way to vote beside postal voting, or as a necessary alternative during the pandemic. As of today, electronic voting has been used for politically-binding elections in several countries (Australia, Switzerland, Estonia, to name but a few). For such high-stakes contexts, coercion may be an important threat. Coercion occurs when an attacker forces a voter to vote in a specific way, using a threat or a reward. This phenomenon is known to exist in real-world elections, with traditional voting at polling stations. An electronic voting solution which is not designed to tackle coercion could allow the attacker to coerce a larger number of voters, or to gain a more convincing evidence that the coerced voters actually obeyed. Since Internet voting is a remote voting process, this introduces new attacks compared to voting at polling station. The most basic attack is when the coercer asks the voter to give them all the voting material that they received. The classical verifiability mechanisms will then provide a proof to the coercer that the voter did not cheat.

The JCJ protocol. A famous protocol with the aim to counter this coercion threat was proposed in 2005 by Juels, Catalano and Jakobsson [14]; they also provide a formalization of the notion, allowing to give security arguments. This is now called the JCJ protocol and remains the reference for the research on coercion-resistance in electronic voting. The key idea of JCJ is that voters can give fake voting material (a fake credential) to the coercer, and pretend that it is genuine. The coercer, who votes with the provided credential, should have no way to detect whether the credential is valid or not. In order to guarantee this, during the voting phase, ballots are accepted in the ballot box regardless of the validity of their credentials; those which use an invalid credential or a duplicate credential are removed later, during a *cleansing* phase. The output of this cleansing phase is a set of ballots that is tallied in the usual way. The main security feature is that given a credential and all the publicly available information, the coercer is unable to tell whether the credential is real or fake. At the same time, for the legitimate voters, verifiability is preserved.

For coercion-resistance, the cleansing phase is critical. In JCJ, there is some unavoidable information that leaks, namely the number of the ballots sent to the public board (the input of the cleansing phase) and the number of ballots sent to the tallying procedure (the output of the cleansing phase). It is well known that the difference Δ between these two numbers can reveal some information to the coercer. Therefore it is recognized as important to ensure some "noise", coming from revotes or dummy ballots, that would mask the action of a voter resisting a coercer. Still, depending on its exact definition, the cleansing phase could leak more than just this difference. For instance, in [23], the authors propose a protocol where the coercer can deduce the number of ballots which pretend to be from each voter, and exploit this additional information (which is not available in the original JCJ). To protect from such an exploit, they propose that the authorities add a random number of dummy ballots for each voter, which mitigates the impact of the leakage.

A weakness in JCJ. In fact, we unveil that even in the JCJ original protocol, the cleansing step leaks more than the difference Δ between the sizes of its input and output. Indeed, first, the ballots with the same credentials (*i.e.* revotes) are handled, keeping only one ballot per credential. Second, the ballots corresponding to invalid credentials are eliminated. The size of the intermediate ballot box is leaked. More than that, anyone can observe the distribution of the number of revotes. We provide a few examples where this allows the attacker to fully break coercion-resistance, even if dummy ballots are used. While these are extreme scenarios that are unlikely to occur in practice, this highlights that in realistic scenarios, the attacker could still gain a non-negligible advantage by exploiting the leakage, in the sense that even though the attacker is not 100% sure that the voter cheated, a hint is

given that should not exist if the cleansing procedure were leaking only Δ. All the variants and improvements on JCJ that we know of are also affected by this vulnerability.

A cleansing-hiding protocol. We propose a modification of JCJ, that we call CHide, and that is not subject to this weakness. The key modification is the introduction of a *cleansing hiding* procedure, that replaces the original cleansing phase. JCJ was using plaintext equivalence tests (PET) as a main cryptographic tool. CHide relies on slightly more complex MPC building blocks to be run between the trustees who hold shares of the decryption key. Instead of PETs that return a bit telling whether two ciphertexts represent the same cleartext, we use a primitive `Eq` that returns an encrypted version of this bit, in order to hide which ballots are removed and for which reason (revote or fake credential). Then a few logical gates primitives `Or`, `And`, *etc.* must be operated on encrypted bits. At the end of the cleansing, for each (mixed and re-randomized) input ballot, a bit is decrypted, and indicates whether the ballot must be discarded (without revealing the reason). As a consequence, in CHide, the adversary can only learn minimal information from the cleansing phase, namely the number Δ of ballots that have been removed. Of course, each step comes with a zero-knowledge proof that the expected operation has been performed, so that anyone can check that the result of the election is correct.

A stronger notion of coercion-resistance. The leakage of JCJ was not noticed before because the original JCJ definition of coercion-resistance was too weak. It was already remarked in [11] that JCJ's original definition was flawed in the sense that it could never be realized. The fix proposed in [11] repairs this flaw but both definitions do not consider the case where voters may revote and hence miss the situation where JCJ leaks too much information. These definitions also model the addition of ballots with fake credentials in a contrieved way in the sense that each fake ballot must be cast by a voter that sacrifices her right to vote.

We propose a more general definition of coercion-resistance that accounts for revoting as well as the addition of fake ballots by authorities. We prove that CHide is coercion-resistant according to this definition, and that JCJ is not, thus showing that we indeed capture the weakness of the leakage during the cleansing phase.

While JCJ does not verify our definition of coercion-resistance, it does provide a certain level of security. We identify the exact nature of the leakage in JCJ and propose an alternative (weaker) definition of coercion-resistance that is achieved by JCJ. Beyond giving a concrete description of the leakage that occurs in the JCJ protocol, this demonstrates that there are several shades of coercion-resistance depending on the leakage which is considered acceptable. More generally, for any variant of JCJ in the literature, if the leakage during the cleansing is different from the one in JCJ, our definition of coercion-resistance could (in principle) be weakened to capture exactly this leakage. Still, CHide shows that a better coercion-resistance can be achieved.

Summary of the contributions.

- We discovered a vulnerability in JCJ's scheme, which shows that it is not perfectly coercion-resistant, due to a leakage during the cleansing phase, before the tally.
- We propose CHide, a cleansing-hiding variant of JCJ, that is not subject to this problem.
- We propose a new definition of coercion-resistance, which properly takes revoting and dummy ballots into account.
- We prove that CHide is coercion-resistant for this definition, while JCJ is not, thus showing the definition is precise enough to capture the leakage during cleansing.
- We explain how our definition can be weakened to describe precisely the level of coercion-resistance of JCJ-like schemes, in particular the original JCJ.

Related work. To provide coercion-resistance, many authors used the core idea of JCJ to allow the voters to provide the coercer with a fake credential. Civitas [8] is one of the most notable examples. It is widely considered as an important step towards a practical version of JCJ. Among other things, Civitas introduces the notion of ballot blocks, in order to mitigate the cost of the cleansing phase that is quadratic in the original JCJ. This reduces somehow its coercion-resistance, and this reduction can be captured with our definition.

Other attempts were made to improve the efficiency of JCJ. In [23], Spycher *et al.* claim a linear time cleansing, but this gain of efficiency comes with a deterioration of the coercion-resistance, as explained above. Later on, the same authors proposed other schemes with a clear trade-off between efficiency and coercion-resistance, thanks to anonymity sets [22].

Other improvements of JCJ include [1], where Araújo *et al.* propose a way to perform the cleansing phase in linear time, and [7], where Clark and Hengartner introduce the idea of *over-the-shoulder* coercion-resistance. Both schemes would suffer from the same cleansing leakage as JCJ, but it can be observed directly from the public board, when the adversary can still submit ballots. In contrast, the leakage in JCJ can only be observed during the cleansing phase, when the adversary can no longer submit ballots.

Apart from the fake credential paradigm, a natural approach to address coercion is through *deniable revoting*, where the coerced voter complies with the coercer but revotes later when they have a moment of privacy. The ballot cast under coercion is invalidated by the subsequent ballot, in such a way that the coercer cannot tell whether the coerced voter revoted or not. The Estonian voting system [19] completely relies on revoting to mitigate coercion, and examples of recent academic proposals based on revoting are VoteAgain by Lueks *et al.* [18] and the scheme of Locher *et al.* [17]. This approach assumes a weaker adversary than in the JCJ family since the adversary can no longer submit ballots passed a certain point.

All these schemes (including JCJ) do not address the so-called Italian attacks. Such coercion attacks exist independently of the use of electronic voting. They are based on the information given by the tally and can occur when the ballots

are complex enough, so that voters can "sign" their ballot using a specific pattern on the low-stake parts of the answers. When using electronic voting, the typical way to prevent such attacks is *tally hiding*, *i.e.* to decrypt only the winners of the election, without decrypting the individual ballots. The challenge is then to preserve verifiability. Therefore tally-hiding usually relies on homomorphic encryption, or more generally on multiparty computation (MPC) techniques [3], [9], [15]. Designing a tally-hiding scheme is out of scope of this paper but interestingly, CHide could be easily coupled with such tally-hiding schemes, right after the cleansing phase.

Regarding formal definition of coercion-resistance, we already mentioned the recent work of Haines and Smyth [11] that attempts to survey and unify the various definitions of the literature, starting with the one of JCJ where the adversary must distinguish between a real and an ideal game modeling the protocol.

Another approach to define coercion-resistance is given by Küsters *et al.* in [16]. The authors define δ-coercion-resistance with two conditions: first, the coerced voter must have a strategy to meet their objective with overwhelming probability (*i.e.* they can actually evade coercion and vote for the desired candidate); second, the adversary cannot distinguish the case when the voter uses their evasion strategy from the case where the voter forwards all received messages to the adversary, with an advantage greater than δ. This is therefore a way towards a quantitative definition of coercion-resistance, giving more than a yes/no answer.

II. Unveiling a Shortcoming in JCJ

We provide a high level description of the JCJ protocol and show why coercion-resistance is undermined in case of revoting.

A. Overview of JCJ

Setup. During the setup, the Election Trustees jointly generate the election public key pk_T, that is sent to the public board. Then, the Registrars jointly compute one credential σ for each voter. Each credential is sent privately to the voter, possibly with designated zero-knowledge proofs to guarantee voters that their credential is valid. The Registrars send to the public board the list $\boldsymbol{R} = \{\mathsf{enc}(\sigma_1, \mathsf{pk}_T), \ldots, \mathsf{enc}(\sigma_n, \mathsf{pk}_T)\}$ of encrypted credentials, in some random order.

Voting phase. To cast a vote, a voter encrypts her vote ν with the election public key pk_T. She also encrypts her credential σ and proves knowledge of ν, of σ, and proves that the two encryptions are linked, yielding a proof π. The resulting ballot $b = (\mathsf{enc}(\nu, \mathsf{pk}_T), \mathsf{enc}(\sigma, \mathsf{pk}_T), \pi)$ is sent anonymously to the bulletin board.

The voting phase is depicted in Figure 1.

Tally phase. The tally phase is the key part of the JCJ protocol. It consists of four steps.

1) Duplicates are removed using Plaintext Equivalence Tests (PET). In case several ballots contain the same credential, only one is kept, according to the revote policy. Typically, the last submitted ballot is kept.
2) The trustees mix the ballots.
3) PETs are used again, to detect ballots that do not have a valid credential, that is, that do not have a credential that appears in the list $\boldsymbol{R}$ of encrypted credentials.
4) Finally, each remaining ballot is decrypted so that the result can be computed.

Each step includes zero-knowledge proofs that the right operation is performed.

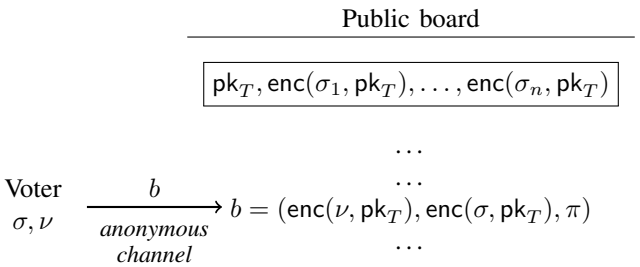

Fig. 1. Voting phase in JCJ.

Evasion strategy. If Alice is under coercion, she simply provides her coercer with a random (and fake) credential σ'. Ballots containing σ' will be removed at Step 3 of the tally phase. Thanks to the mixing phase, the coercer will be unable to learn that his ballot has been suppressed. Alice then uses her real credential σ to cast her vote when she is not under the control of her coercer.

B. Leakage in case of revoting

For a verifiable voting system, it seems unavoidable to leak the total number of received ballots since ballots are visible as soon as there is some public board. The number of valid ballots is also leaked unless more sophisticated tally methods are used such as tally-hiding schemes [3], [15]

However, JCJ leaks much more information in case of revoting. Indeed, JCJ leaks:

- n_b the total number of received ballots
- n_v the total number of valid (and counted) ballots
- n_r the total number of revotes
- and even the complete distribution of revotes, per (encrypted) credential.

This can be exploited by a coercer to detect when a coerced voter disobeys. Observe first that there is no reason to assume that revoting is independent from the choice of candidate. Indeed, revoting is often due to voters changing their mind between candidates, or even opting for some candidate C due to some late announcements about C, in the press.

Attack against coercion-resistance. We consider an extreme case, with two candidates A and B and such that voters voting for A do not revote while voters voting for B always revote, exactly once. We denote:

- r_A the number of votes for A,

- r_B the number of votes for B.

Due to the distribution of voting behaviors that we consider, the number of revotes corresponds to the number of votes for B sent by the honest voters.

Assume now that Alice wants to vote for B but is instructed by her coercer to vote for A (or abstain).

- If Alice obeys, the coercer will observe $r_B = n_r$.
- If Alice disobeys and casts one ballot for B, the coercer will observe that $r_B = n_r + 1$.

Hence the coercer will detect that Alice has disobeyed, which breaks coercion-resistance.

One could argue that Alice, our coerced voter, should follow a different evasion strategy and cast one ballot if she votes for A and two ballots if she votes for B. This does not work either. Indeed, assume now that Alice wants to vote for A, but is instructed to vote for B (resp. abstain).

- If she obeys, she will provide her coercer with her real credential σ. The coercer will then cast *exactly one* ballot for B using σ (resp. zero).
- If she disobeys, she will provide a fake credential σ', that the coercer will use to vote for B (resp. not use). Alice will vote for A using σ.

In the first case, $r_B = n_r + 1$, while in the second case, $r_B = n_r$. Once again, the coercer is able to detect that Alice disobeyed and coercion-resistance is broken.

Discussion. One possibility to correct this flaw is to define other evasion strategies in case of revoting. The evasion strategy devised in JCJ (vote once, with the correct credential) is too weak. However, it seems very hard in practice to instruct voters to use revoting, according to a certain distribution, when they are under coercion. This is made even harder by the fact that the strategy may evolve during time depending on new events that could change the revoting distribution of honest voters. We believe that this will not lead to any realistic scheme.

Hence we propose another option, in Section III, that consists in renforcing JCJ in case of revoting, such that there is no leakage besides the total number of ballots and the number of valid ballots.

One could also argue that the distribution considered for the attack (voters always vote twice when they vote for B) is very contrived, which is certainly true. But as soon as the distribution of revotes is not independent from the distribution of votes per candidate, the coercer will learn some information, and hence will detect when a voter disobeys with some non negligible probability. This can be amplified if the coercer coerces several voters at the same time. Hence coercion-resistance remains broken.

C. More noise is needed

A known issue of JCJ is the fact that fake ballots should be randomly added, in order to hide to a coercer that her ballot has been removed. Indeed, imagine a situation where in several elections, absolutely no ballots with fake credentials are removed at Step 3. Then in a new election with the same set of voters, a coercer asks Alice to provide her credential and observes that exactly one ballot is removed at step 3. Very likely, Alice has disobeyed.

Hence, it is necessary that a non-deterministic number of ballots is removed during the tally. In JCJ, this "noise" comes from honest voters sending ballots with an invalid credential (dummy ballots), but this source alone may not be sufficient. A natural approach is to have the authorities add a random number of dummies. This solution is proposed in [23], where it is used to mitigate a leakage during the tally. This noise made of fake ballots should however be calibrated carefully since the computation overhead of additional ballots is important. In a context where revoting is a well spread behavior, it could be judicious to rely on revoting, at least partially, as an additional source of noise. This is however not possible in JCJ since these two sources of noise can be distinguished.

III. CHide: A Cleansing-Hiding Variant of JCJ

We propose a modification of JCJ. During the tally phase, the trustees will perform the same tasks of cleansing, mixing and decrypting, but in a hidden way, so that the coercer (or anyone) does not learn how many ballots were deleted because they correspond to revotes or to invalid credentials. For this, we propose a novel cleansing algorithm based on MPC primitives.

A. Cyrptographic primitives

ElGamal encryption scheme. We use the ElGamal encryption scheme on elliptic curves, which is convenient for its efficiency and its homomorphic property. If g is the group generator and pk the public key of the recipient, in order to encrypt a message m, one chooses a random exponent r and compute $\mathtt{Enc}(m, \mathsf{pk}) = (g^r, g^m\, \mathsf{pk}^r)$. We impose the message m to be taken in a small list of valid mesages, so that decryption is feasible (in general recovering m from g^m would require to solve a discrete logarithm problem). An important special case is when m is a bit, because this is the kind of inputs used by the MPC primitives we mention below. For a general message m, we call bit-wise encryption of m the list of the encryptions of the bits in a binary encoding of m.

Distributed key generation / threshold decryption. A DKG [10] is a protocol which allows the participants to generate an ElGamal public key pk in such a way that each participant gets a share of the secret key. The protocol also generates a public commitment to each participant's share. A threshold t is set, with the following properties. If t or less participants collude, they can not deduce any information about the secret key, nor about any ciphertext. If $t + 1$ or more participants collaborate, they can combine their shares to recover the secret key. Furthermore, they can jointly run a threshold decryption protocol, so that, using their shares, they can jointly decrypt a ciphertext encrypted with pk, without actually recovering the secret key nor revealing any information about their secret share. The resulting decryption comes with a zero-knowledge proof (ZKP) of correct decryption that anyone can verify.

Designated Verifier Zero-Knowledge Proof (DVZKP). In complement to traditional zero-knowledge proofs that anyone can verify, the JCJ protocol requires a DVZKP [13]. A verifier V holds a key pair. Using the public key, any prover can produce a proof for a statement, so that only V is convinced that the statement is true. This is achieved by allowing the verifier to forge fake but valid DVZKPs for any statement, using their private key.

Verifiable mixnets. A mixnet [24] allows the participants to shuffle and reencrypt a list of ciphertexts. The participants are not required to know the (shares of the) decryption key. As soon as one participant is honest, the permutation remains secret. The result comes with a zero-knowledge proof that the resulting ciphertexts are re-encryptions of the ones given in input, up to permutation.

Distributed random bit generation. A set of participants can follow a random bit generation protocol `RandBit`, such that they jointly produce an encrypted bit $\texttt{Enc}(b, \mathsf{pk})$. Each participant gets a share b_i of the bit b, in such a way that knowing all these partial informations allows to reconstruct b. Furthermore, the transcript of the communications of `RandBit` serves as a proof that $\texttt{Enc}(b, \mathsf{pk})$ is indeed the encryption of a bit, that is unknown as soon as one of the participants is honest (assuming the decryption key is unknown).

Logical operations on encrypted bits. There are MPC protocols that allow the owners of the shares of the decryption key to jointly perform logical operations on encrypted bits, based on their threshold decryption protocol. This is done without revealing the cleartexts to anyone, and in a verifiable manner (the threshold decryption is not applied directly to the inputs). The main building block we use is the `CGate` protocol [21], that allows to compute an encryption of a logical and (a conjunction) of the encrypted bits given in input. Combining this with the homomorphic property of ElGamal encryption, one can design various protocols for all the logical operations on bits (*e.g.* negation, disjunction, exclusive or).

In the cleansing phase of our protocol, we will use specifically the `Eq` (equality test) and the `Or` (disjunction) protocols that work on encrypted bits. The `Eq` protocol is extended to bit-wise encrypted data, by computing the conjunction of all the equality tests on encrypted bits.

In the appendix, we recall for completeness the cryptographic primitives that were not already present in the original JCJ protocol, namely the distributed random bit generation and the logical operations on encrypted bits.

The security of all the primitives relies on the Decisional Diffie-Hellman assumption, and all proofs are made in the Random Oracle Model, in particular due to the use of the Fiat-Shamir tranformation to get non-interactive ZKPs.

B. Description of the CHide protocol

Participants. The CHide protocol is similar to JCJ, and the list of participants is the same. We recall them and emphasize the trust assumptions on them.

- The **public board**. This is an append-only list of data where all the other participants can write. At any time, the content of the board can be read by anyone, and the view is the same. The board is assumed to be honest. See [12] for a possible realization of this.
- The **auditors**. Any entity can perform checks about the consistency of the data present on the board. This includes the validity of all the zero-knowledge proofs. We assume that there is at least one honest auditor that will reveal any problem that can be detected by reading the public board.
- The **registrars**. A set of n_R participants play the role of registrars. They send voting material to the legitimate voters. For privacy and verifiability, it is assumed that at least one registrar is honest. For coercion resistance, it is assumed that all of them are honest (no registrar collaborates with the coercer).
- The **election trustees**. A set of n_T participants play the role of election trustees. They hold the key shares and perform the cleansing and the tally. For privacy and coercion-resistance, it is assumed that at most t dishonest trustees can collaborate, where $t < n_T$ is the threshold used in the DKG. For verifiability, there is no trust assumption on them.
- The **voters**. There are n_V voters. Some of them may be dishonest and collude with the attacker. Honest voters may be subject to a coercion attempt by the attacker.

Setup phase `Setup`. A security parameter κ is chosen. The election trustees jointly run the DKG protocol, yielding a public key pk guaranteeing security at this level. The DKG also produces a list of commitments h_i, one for each of the n_T trustees. All this public data $(h_i)_{1\leq i\leq n_T}$, together with pk is sent to the public board. The secret shares are denoted by $(s_i)_{1\leq i\leq n_T}$.

Registration phase `Register`. In JCJ, the public board contains the encrypted credentials. The main difference in CHide is that the credentials are now encrypted *bitwise*. The registrars produce credentials as follows. For all $1 \leq i \leq n_V$, they run κ times the `RandBit` protocol. The resulting encrypted bits are denoted $\mathcal{R}_i = (R_{i,1}, \ldots, R_{i,\kappa})$, and the proofs that they have been correctly generated are denoted by $\Pi_i^{\texttt{Cred}}$. The registrars send to the board the list $\boldsymbol{R} = (\mathcal{R}_i, \Pi_i^{\texttt{Cred}})_{1\leq i\leq n_V}$. Each registrar keeps the random values they used in the `RandBit` protocol.

The voters authenticate themselves with the registrars. For each voter, the registrars choose a random element in the list $\boldsymbol{R}$ which was not chosen previously, and send to the voter its index j in $\boldsymbol{R}$ and their shares of the bits. Each registrar also produces and sends a DVZKP that their shares are correct. We denote by $\Delta_{i,j} = (\Delta_{i,j,1}, \ldots, \Delta_{i,j,\kappa})$ the DVZKPs for the data coming from registar i and sent to the voter who receives the credential with index j, using an untapable channel.

Then, the voters combine the shares to recover the plaintexts $\sigma = (\sigma_1, \ldots, \sigma_\kappa)$, which form their credentials. They read their entries in the public list $\boldsymbol{R}$, and, via the DVZKPs, they verify that these plaintexts correspond to their entries.

Voting phase `Vote`. During the voting phase, in order

to cast a vote for the option ν, a voter computes $C_1 = \texttt{Enc}(\nu, \mathsf{pk})$, and produces a ZKP π_1 that the randomness of the encryption is known and that ν is a valid voting option. They also compute a bitwise encryption of their credential $C_2 = (\texttt{Enc}(\sigma_1, \mathsf{pk}), \ldots, \texttt{Enc}(\sigma_\kappa, \mathsf{pk}))$. An additional ZKP π_2 is produced, that proves that C_2 contain encryptions of bits. It also links C_1 and C_2, so that the pair (C_1, C_2) is non-malleable (concretely, since we use ZKPs based on Fiat-Shamir, the data hashed to create the challenge contains this pair). The ballot is (C_1, C_2, π_1, π_2). It is sent to the public board by the voter, using an anonymous channel.

The voters check that their ballot is indeed present on the public board.

The auditors verify that the ZKPs are valid and that all the C_1 and C_2 parts of the ballots present in the public board are unique.

Cleansing and tallying phase. Once the voting phase is finished, the election trustees get the list of ballots published on the board. They run the following cleansing procedure `Cleanse` on them, where all the intermediate data and verification transcripts are sent to the public board.

1) Discard all the ballots marked as invalid by the audit procedure. Denote by $B = (C_1^i, C_2^i)_i$ the remaining ballots, without the ZKPs. Let n_b be the size of this list.
2) For $1 \leq i \leq n_b$ and for all $j > i$, run the bitwise equality test protocol $\texttt{Eq}(C_2^i, C_2^j)$. This produces encrypted boolean $D_{i,j}$ and verification transcripts $\Pi_{i,j}^{\texttt{Eq}}$.
3) For $1 \leq i \leq n_b$, run the logical disjunction protocol $\texttt{Or}(D_{i,i+1}, \cdots, D_{i,n_b})$. This produces encrypted booleans D_i and verification transcripts $\Pi_i^{\texttt{Or}}$. The ciphertext D_i encrypts 1 if there is a ballot after ballot i with the same credential, meaning that ballot i should be removed since only the last ballot of a voter is kept.
4) For $1 \leq i \leq n_b$, and for $1 \leq j \leq n_V$, run the bitwise equality test protocol $\texttt{Eq}(C_2^i, R_j)$. This produces encrypted booleans $T_{i,j}$ and verification transcripts $\tilde{\Pi}_{i,j}^{\texttt{Eq}}$. Then run the disjunction protocol $\texttt{Or}(T_{i,1}, \cdots, T_{i,n_v})$ and produce the booleans T_i with verification transcripts $\tilde{\Pi}_i^{\texttt{Or}}$. The ciphertext T_i encrypts 1 if the credential of ballot i appears in the R_j. Finally, for $1 \leq i \leq n_b$, compute $F_i = \texttt{Not}(T_i)$ (this is done homomorphically, and does not require a ZKP). If F_i encrypts 1 then ballot i should be removed since it contains an invalid credential (since it does not appear in any R_j).
5) For $1 \leq i \leq n_b$, run the disjunction protocol $\texttt{Or}(D_i, F_i)$ to obtain I^i and Π_i. The ballots i such that I^i encrypts 1 are the invalid ballots.
6) Apply the mixnet protocol on the tuples $(C_1^i, I^i)_i$ to produce $(C_1'^i, I'^i)$ and a verification transcript $\Pi^{\texttt{Mixnet}}$.
7) Decrypt all the I'^i's. This reveals the plaintext z_i' and a ZKP of correct decryption $\Pi_{i,1}^{\texttt{Dec}}$.

The bit z_i' tells whether the encrypted vote $C_1'^i$ corresponds to an invalid ballots, without revealing whether a ballot has been discarded because of a revote or an invalid credential.

The election trustees finally decrypt all the $C_1'^i$ such that $z_i' = 0$, and produce ZKPs of correct decryption $\Pi_{i,2}^{\texttt{Dec}}$.

The auditors check the validity of all the data published on the board during this phase.

Evading coercion. When under coercion, a voter can lie about the registration as follows. First, they generate $n_r\kappa$ shares at random in $\{0, 1\}$ and use the ciphertexts $\mathcal{R}_j$ for the index j received during registration. Then they forge a fake DVZKP that the ciphertexts are encryptions of the shares and send the fake shares, the real index j, the real ciphertexts $\mathcal{R}_j$ and the fake DVZKP to the coercer. We denote `Fakecred` this procedure.

Assumptions on the communication channels. As in the original JCJ protocol, the communications between the voters and the registrars must be untapable, and the communication between the voters and the public board must be anonymous. This is required only for the coercion-resistance property. Verifiability and vote secrecy do not rely on non-standard communication channels assumptions.

For verifiability, we assume that honest voters check that their ballots appear on the public board and complain as soon as something unexpected happens. For weaker voters that check their ballots only when they vote, the ballot order should be enforced *e.g.* using chain of hash (see e.g. [2]).

Efficiency considerations. In terms of computational and communication costs, the CHide system is slightly less efficient, but still in the same ballpark than JCJ. The encrypted credentials are now formed by κ ciphertexts instead of a single one. This κ factor is probably affordable by the authorities since the task is highly parallel. For the voters, the computational load increases but the total cost for realistic parameters is around a thousand exponentiations, which should be a matter of seconds with a standard implementation in Javascript running within a modern browser.

For the talliers, the cleansing phase is more complex than the one in JCJ, but still requires a number of exponentiations that grows quadratically with the number of ballots received on the board. The main difference is that due to the MPC tools, the number of communication rounds between them is no longer constant, but becomes logarithmic in the number of ballots. This assumes that in steps 3 and 4 of the cleansing, the implementation of the `Or` with multiple inputs is done with a binary tree, each node corresponding to a bivariate `Or`.

IV. Defining coercion-resistance

One of the reasons why the flaw was not discovered in the original proposition of the JCJ protocol is the fact that the definition of coercion-resistance itself was flawed, in the sense that no scheme can be proved secure w.r.t. this definition. We recall here this definition and discuss its issues, as well as the ones of the variant proposed in [11]. We then introduce our own definition, that is used to analyse the security of CHide.

A. Voting system

A *voting system* is a tuple of five functions or protocols $(P_{\texttt{Setup}}, \texttt{Register}, \texttt{Vote}, \texttt{Fakecred}, P_{\texttt{tally}})$ such that:

- $P_{\texttt{Setup}}(\kappa, n_T, t)$ is a protocol run by n_T authorities. They take as input the security parameter κ and the threshold t of authorities for decryption. The protocol computes the public key of the election pk, as well as the secret and public shares s_i, h_i for each authority.
- $\texttt{Register}(\kappa, \mathsf{pk}, n_V)$ generates a private credential c_i for each voter $\in [1, n_V]$. It also returns some public information $\boldsymbol{R}$ that contains the public part of the credentials, and any necessary transcript for proving correctness.
- the function $\texttt{Vote}(c, \nu, \mathsf{pk})$ takes as input the public key of the election pk, a credential c and a vote ν and returns a ballot. The public key is often omitted for simplicity.
- $\texttt{Fakecred}(c)$ takes as input a valid credential c and returns a fake one $\tilde{c}$. Note that $\texttt{Fakecred}$ may also take as input some long term secret keys of the voters, as well as some public information (e.g. $\boldsymbol{R}$) as in JCJ but this is left implicit for the sake of readability.
- $P_{\texttt{tally}}(BB, \mathsf{pk}, \{h_i, s_i\}, t)$ is a protocol run by the authorities that possess the secret shares $\{s_i\}$. It takes as input a board BB, that is, a list of ballots, the public key of the election pk, the threshold t and returns the result $\boldsymbol{X}$ of the election together with a proof Π of correct tally.

Note that the registration phase is actually a protocol since it is run by several authorities. However, for coercion-resistance, we assume all the registrars to be honest hence we consider a single function for registration.

B. The original definition of JCJ

The intuition of the JCJ definition of coercion-resistance is that an adversary must not guess whether a coerced voter obeyed or evaded coercion. When the voter obeys ($b = 1$ in the definition), she gives her real credential and abstains from doing any other action. Note that a coercer may ask the voter to cast some specific vote or to perform some specific computations, but this is not considered in the definition as the adversary might as well do it herself. When the voter evades ($b = 0$ in the definition), she gives a fake credential and casts a vote for the desired voting option (or abstains, depending on her personal choice).

This yields the game $\mathsf{Real}_{\mathsf{JCJ}}$ presented in Algorithm 1, where κ is the security parameter, n_V denotes the total number of voters while n_A is the number of corrupted voters, under the control of the adversary. Voting choices are represented as integers between 1 and n_C. The symbol ϕ represents the choice to abstain. The adversary selects V, the set of corrupted voters. She is given the corresponding private credentials as well as all the public information $\boldsymbol{R}$ such as the set of encrypted credentials in the JCJ protocol. The adversary then choses (j, β). The number j designates the voter under coercion. Voter j will try to vote β when she disobeys. The evasion strategy is modeled in lines 11 and 12: when the voter disobeys, she creates a fake credential and casts a vote for β. Otherwise, she gives her real credential.

Honest voters votes according to a distribution $\mathcal{D}$. The distribution $\mathcal{D}$ takes as parameter the number of voting option n_C and returns a value that may be:

- any valid vote $\nu \in [1, n_C]$,
- ϕ, which represents abstention,
- λ, which represents the case where a voter casts a vote with an invalid credential.

By convention, we extend the $\texttt{Vote}$ function to votes equal to λ as follows

$$\texttt{Vote}(c, \lambda, \mathsf{pk}) = \texttt{Vote}(\tilde{c}, \nu, \mathsf{pk})$$

where $\tilde{c} = \texttt{Fakecred}(c)$ and ν is sampled from $[1, n_C]$.

It is worth noting that the advantage of an adversary in game $\mathsf{Real}_{\mathsf{JCJ}}$ will always be non negligible since the adversary can always compare the result of the tally with the expected result, given the distribution $\mathcal{D}$ of the voting intentions. For example, if the adversary wants to cast a vote for a very unlikely candidate c, he may observe cases where her candidate c does not appear in the result, which is a clear indication that the coerced voter disobeyed. Hence, the JCJ definition compares the advantage of an adversary in game $\mathsf{Real}_{\mathsf{JCJ}}$ with the advantage in an ideal game $\mathsf{Ideal}_{\mathsf{JCJ}}$, where the adversary has no other information than what is unavoidably leaked, that is, the information leaked by the result. The game $\mathsf{Ideal}_{\mathsf{JCJ}}$ is presented in Algorithm 2. Compared to the original definition, we present a slightly modified version that reasons on the clear votes only. This simplifies the definition, focusing on the information given to the adversary. All our claims and remarks hold on the original definition as well.

Definition 1 (adapted from [14])**.** *A voting system is* JCJ-coercion resistant *if for all PPT adversary $\mathbb{A}$, for all parameters n_T, t, n_V, n_A, n_C, and for all distributions $\mathcal{D}$, there exists a PPT adversary $\mathbb{B}$ and a negligible function μ such that*

$$\begin{aligned}&|\Pr(\mathsf{Ideal}_{\mathsf{JCJ}}(\mathbb{B}, \kappa, n_V, n_A, n_C, \mathcal{D}) = 1)\\ &- \Pr(\mathsf{Real}_{\mathsf{JCJ}}(\mathbb{A}, \kappa, n_T, t, n_V, n_A, n_C, \mathcal{D}) = 1)| \leq \mu(\kappa)\,.\end{aligned}$$

As noticed in [11], this definition cannot be realized by a scheme which uses a public board. The reason is that, in the real game $\mathsf{Real}_{\mathsf{JCJ}}$, the adversary observes the length n_b of the board which correspond to the total number of ballots cast by non-corrupted voters. Then the adversary learns the result and in particular its size n_v, that is the number of valid ballots counted. Hence the adversary learns the total number $\Delta = n_b - n_v$ of ballots discarded, which is not available in the ideal game $\mathsf{Ideal}_{\mathsf{JCJ}}$. The value of Δ can be compared with its expected number, according to the distribution $\mathcal{D}$. Since there is an additional ballot discarded (the one of the coercer) when the voter evades coercion, the adversary has a non-negligible advantage in the real game. For instance, if $\mathcal{D}$ is such that no (or very few) voters cast a ballot with an invalid credential, either $n_v = n_b + 1$ which means that the adversary's ballot has been counted, or $n_v = n_b$ meaning that the adversary's ballot has been discarded and that the voter has disobeyed. Of course, the same issue applies to the JCJ definition as stated in [14].

The authors of [11] proposed a patch to the issue they discovered: the length of the board should be given to the ad-

Algorithm 1: $\mathsf{Real}_{\mathsf{JCJ}}$

Require: $\mathbb{A}, \kappa, n_T, t, n_V, n_A, n_C, \mathcal{D}$

1 $BB \longleftarrow \varnothing$
2 $\mathsf{pk}, s_1, h_1, \cdots, s_{n_T}, h_{n_T} \longleftarrow P^{\mathbb{A}}_{\mathtt{Setup}}(\kappa, n_T, t)$
3 $V \longleftarrow \mathbb{A}()$
4 $\{c_i; i \in [1, n_V]\}, \boldsymbol{R} \longleftarrow \mathtt{Register}(\kappa, \mathsf{pk}, n_V)$
5 $(j, \beta) \longleftarrow \mathbb{A}(\{c_i; i \in V\}, \boldsymbol{R})$
6 **if** $|V| \neq n_A \vee j \notin [1, n_V] \backslash V \vee \beta \notin [1, n_C] \bigcup \{\phi\}$ **then**
7 Return 0
8 $b \xleftarrow{\$} \{0, 1\}$
9 $\tilde{c} \longleftarrow c_j$
10 **if** $b == 0$ **then**
11 $\tilde{c} \longleftarrow \mathtt{Fakecred}(c_j)$
12 $BB \longleftarrow BB \bigcup \{\mathtt{Vote}(c_j, \beta)\}$
13 **for** $i \in [1, n_V] \backslash (V \bigcup \{j\})$ **do**
14 $\nu_i \longleftarrow \mathcal{D}_{n_C}()$
15 **if** $\nu_i \notin \{\phi, \lambda\}$ **then**
16 $BB \longleftarrow BB \bigcup \{\mathtt{Vote}(c_i, \nu_i)\}$
17 $BB \longleftarrow BB \bigcup \mathbb{A}(\tilde{c}, BB)$
18
19
20 $\boldsymbol{X}, \Pi \longleftarrow P^{\mathbb{A}}_{\mathtt{tally}}(BB, \boldsymbol{R}, \mathsf{pk}, \{h_i, s_i\}, t)$
21 $b' \longleftarrow \mathbb{A}()$
22 Return 1 **if** $b' == b$ **else** 0

Algorithm 2: $\mathsf{Ideal}_{\mathsf{JCJ}}$

Require: $\mathbb{A}, \kappa, n_V, n_A, n_C, \mathcal{D}$

1 $D \longleftarrow \varnothing$
2
3 $V \longleftarrow \mathbb{A}(\kappa)$
4
5 $(j, \beta) \longleftarrow \mathbb{A}()$
6 **if** $|V| \neq n_A \vee j \notin [1, n_V] \backslash V \vee \beta \notin [1, n_C] \bigcup \{\phi\}$ **then**
7 Return 0
8 $b \xleftarrow{\$} \{0, 1\}$
9
10 **if** $b == 0$ **then**
11 $D = D \bigcup \{\beta\}$
12
13 **for** $i \in [1, n_V] \backslash (V \bigcup \{j\})$ **do**
14 $\nu_i \longleftarrow \mathcal{D}_{n_C}()$
15 **if** $\nu_i \neq \phi$ **then**
16 $D = D \bigcup \{\nu_i\}$
17 $(\nu_i)_{i \in V}, \beta' \longleftarrow \mathbb{A}()$
18 **if** $b == 1$ **then**
19 $D = D \bigcup \{\beta'\}$
20 $\boldsymbol{X} \longleftarrow \mathtt{result}(D \bigcup (\nu_i)_{i \in V})$
21 $b' \longleftarrow \mathbb{A}(\boldsymbol{X})$
22 Return 1 **if** $b' == b$ **else** 0

Fig. 2. JCJ definition of coercion resistance. κ is the security parameter, n_T the number of talliers, t the threshold, n_V the number of voters, n_A the number of corrupted voters, n_C the number of voting options and $\mathcal{D}$ the distribution of votes.

versary in the ideal game as well. Intuitively, this corresponds to simply rewriting line 17 of Algorithm 2 as follows:

$$(\nu_i)_{i \in V}, \beta' \longleftarrow \mathbb{A}(|D|)$$

However, this patch still does not allow to detect the leakage of the JCJ protocol during the tally. Indeed, the distribution $\mathcal{D}$ fails to model several aspects, which in turn, renders the comparison between the real and the idea games too weak: there are cases where the cryptographic protocol may differs from the ideal case and such cases are not explored by $\mathcal{D}$. More precisely, the distribution $\mathcal{D}$ fails to models two main aspects.

- First, the addition of ballots with fake credentials is badly modeled since it happens only when an honest voter uses a fake credential, sacrificing her own vote. This is quite unlikely in practice. In addition, $\mathcal{D}$ does not model the case where ballots with fake credentials are added directly by authorities, even if they do not own a valid credential.
- Second, revoting is not considered at all in $\mathcal{D}$, which explains why the leakage of the JCJ protocol was not detected.

C. Taking the public board into account

If we compare the advantage of the adversary in the real game with her advantage in the ideal one, we need to cover a large family of vote distributions, otherwise, we may miss security flaws. In particular, we need to cover cases explicitly planned by the protocol such as revote and addition of ballots with fake credentials.

Therefore, we consider a distribution $\mathcal{B}$ of sequence of pairs of positive integers (j, ν) where $\nu \in [1, n_c]$ represents a vote and j represents either a valid voter (when $j \in [1, n_V]$ or a fake voter, with a fake credential. The distribution $\mathcal{B}$ models both revoting and the addition of fake votes, typically by authorities:

- revoting is reflected in $\mathcal{B}$ by the fact that a voter may appear several time in the same sequence;
- additional fake voters are modeled by pairs (j, ν) where j is not a valid voter ($j \notin [1, n_V]$). The additional fake voters may be either added by authorities or voters that wish to play with the protocol. Note that $\mathcal{B}$ even models the case where authorities (or voters) revote with fake credentials.

For example, the sequence $(1,1), (2,1)(1,2)(3,2), (1,1)$ with $n_V = 2$ represents a situation with two voters V_1 and V_2. V_1 first votes 1, V_2 votes 1 as well, then V_1 revotes for 2, then a fake vote for 2 is added, then V_1 changes back her vote to 1.

Our Real game is similar to $\mathsf{Real}_{\mathsf{JCJ}}$. Votes are drawn according to $\mathcal{B}$, yielding a sequence B. The sequence B typically contains pairs (i, ν) with $i \notin [1, n_V]$. This corresponds to the addition of fake credentials by authorities. We therefore

Algorithm 3: Real

Require: $\mathbb{A}, \kappa, n_T, t, n_V, n_A, n_C, \mathcal{B}$

1 $BB \longleftarrow \varnothing$
2 $\mathsf{pk}, s_1, h_1, \cdots, s_{n_T}, h_{n_T} \longleftarrow P^{\mathbb{A}}_{\texttt{Setup}}(\kappa, n_T, t)$
3 $V \longleftarrow \mathbb{A}()$
4 $\{c_i; i \in [1, n_V]\}, \boldsymbol{R} \longleftarrow \texttt{Register}(\kappa, \mathsf{pk}, n_V)$
5 $(j, \beta) \longleftarrow \mathbb{A}(\{c_i; i \in V\}, \boldsymbol{R})$
6 **if** $|V| \neq n_A \vee j \notin [1, n_V] \backslash V \vee \beta \notin [1, n_C] \bigcup \{\phi\}$ **then**
7 Return 0
8 $B \longleftarrow \mathcal{B}(n_V - n_A, n_C)$
9 **for** $(i, *) \in B$, $i \notin [1, n_V]$ **do**
10 $c_i \longleftarrow \texttt{Fakecred}(c_1)$
11 $b \xleftarrow{\$} \{0, 1\}$
12 $\tilde{c} \longleftarrow c_j$
13 **if** $b == 1$ **then**
14 Remove all $(j, *) \in B$
15 **else**
16 Remove all $(j, *) \in B$ but the last, which is replaced by (j, β)
17 $\tilde{c} \longleftarrow \texttt{Fakecred}(c_j)$
18 $\mathbb{A}(\tilde{c})$
19 **for** $(i, \alpha) \in B$ *(in this order)* **do**
20 $\boldsymbol{M} \longleftarrow \mathbb{A}(BB)$
21 $BB \longleftarrow BB \bigcup \{m \in \boldsymbol{M} \mid m \texttt{ is valid}\}$
22 $BB \longleftarrow \{\texttt{Vote}(c_i, \alpha, \mathsf{pk})\}$
23 $\boldsymbol{M} \longleftarrow \mathbb{A}(BB)$
24 $BB \longleftarrow BB \bigcup \{m \in \boldsymbol{M} \mid m \texttt{ is valid}\}$
25 $\boldsymbol{X}, \Pi \longleftarrow P^{\mathbb{A}}_{\texttt{tally}}(BB, \boldsymbol{R}, \mathsf{pk}, \{h_i, s_i\}, t)$
26 $b' \longleftarrow \mathbb{A}()$
27 Return 1 **if** $b' == b$ **else** 0

Algorithm 4: Ideal

Require: $\mathbb{A}, \kappa, n_V, n_A, n_C, \mathcal{B}$

1
2
3 $V \longleftarrow \mathbb{A}(\kappa)$
4
5 $(j, \beta) \longleftarrow \mathbb{A}()$
6 **if** $|V| \neq n_A \vee j \notin [1, n_V] \backslash V \vee \beta \notin [1, n_C] \bigcup \{\phi\}$ **then**
7 Return 0
8 $B \longleftarrow \mathcal{B}(n_V - n_A, n_C)$
9
10
11 $b \xleftarrow{\$} \{0, 1\}$
12
13 **if** $b == 1$ **then**
14 Remove all $(j, *) \in B$
15 **else**
16 Remove all $(j, *) \in B$ but the last, which is replaced by (j, β)
17
18
19 $(\nu_i)_{i \in V}, \beta' \longleftarrow \mathbb{A}(|B|)$
20 **if** $(b == 1) \wedge (\beta' \neq \phi)$ **then**
21 $B \longleftarrow B \bigcup \{(j, \beta')\}$
22 $B \longleftarrow B \bigcup \{(i, \nu_i); i \in V, \nu_i \in [1, n_C]\}$
23
24
25 $\boldsymbol{X} \longleftarrow \texttt{result}(\texttt{cleanse}(B))$
26 $b' \longleftarrow \mathbb{A}(\boldsymbol{X})$
27 Return 1 **if** $b' == b$ **else** 0

Fig. 3. Definition of coercion-resistance. κ is the security parameter, n_T the number of talliers, t the threshold, n_V the number of voters, n_A the number of corrupted voters, n_C the number of voting options and $\mathcal{B}$ the distribution of the sequence of votes.

generate a fake credential for each such i at lines 9-10. If $b = 0$, the coercer voter j obeys, hence any vote from j is removed from B and the real credential of the voter is provided to the adversary. If $b = 1$, the voter follows the evasion strategy, namely he casts one vote for β and provides a fake credential. Hence the votes from j in $\mathcal{B}$ are replaced by a single vote for β. Then ballots are added according to $\mathcal{B}$. They correspond either to real votes (or revotes) or to fake votes added by the authorities Compared to the original JCJ definition, we also slightly improve the power of the adversary by letting her observe the board after each vote and add ballots if she wants too. This better reflects the reality.

Again, the advantage of the adversary needs to be compared with a corresponding ideal game Ideal where the adversary simply observes:

- the total number of ballots B,
- the result of the election.

We assume a function `cleanse` that removes votes from invalid voters $j \notin [1, n_V]$ and that takes care of revotes according to the policy (typically, the last vote is kept). The function `result`, given a set of valid votes, returns the result of the election.

Definition 2. *A voting system is* coercion resistant *if for all PPT adversary $\mathbb{A}$, for all parameters n_T, t, n_V, n_A, n_C, and for all distribution $\mathcal{B}$, there exists a PPT adversary $\mathbb{B}$ and a negligible function μ such that*

$$|\Pr(\mathsf{Ideal}(\mathbb{B}, \kappa, n_V, n_A, n_C, \mathcal{B}) = 1) - \Pr(\mathsf{Real}(\mathbb{A}, \kappa, n_T, t, n_V, n_A, n_C, \mathcal{B}) = 1)| \leq \mu(\kappa)\,.$$

The main difference between our definition and the original definition is the fact that we consider a larger family of distributions, which allows us to analyze a protocol in the context of revotes and/or addition of fake ballots.

Another difference is the fact that we request that an adversary does not gain any advantage for *any* distribution $\mathcal{B}$, while the JCJ definition defines coercion-resistance *with respect to* a particular distribution. We believe that our approach is

preferable since intuitively, a protocol should be as secure as the ideal one, whatever the considered distribution. It would be quite counter-intuitive to design a cryptographic protocol that resists only for particular distributions. Then of course, it makes sense to analyze the exact advantage of the adversary in the ideal game, for a particular distribution and devise whether voters are reasonably protected in that case of not. But the cryptographic protocol itself should be as solid as the ideal one nevertheless.

D. CHide is coercion-resistant

Thanks to our modified cleansing procedure, we can show that our CHide protocol achieves coercion-resistance. This also shows that our definition can be realized.

Theorem 1. *Under the DDH assumption and in the Random Oracle Model, the voting system CHide is coercion-resistant.*

Proof (sketch). We recall the notations of the definition: n_T is the number of trustees, t is the threshold, n_V is the number of voters, n_C the number of voting options, and n_A the number of dishonest voters.

Let $\mathcal{B}$ be a distribution, and let $\mathbb{A}$ be an adversary that plays the real game. We give to $\mathbb{A}$ the power to impersonate the last t talliers. We construct an adversary $\mathbb{B}$ that plays the ideal game, and that can interact with $\mathbb{A}$ by simulating the real game.

First of all, $\mathbb{B}$ jointly runs the setup with $\mathbb{A}$ to generate the public key pk, the secret shares $s_1, \cdots, s_{n_T}$ and the public commitments $h_1, \cdots, h_{n_T}$. From these n_T shares, $\mathbb{B}$ reconstructs the secret key sk.

Then $\mathbb{B}$ calls $\mathbb{A}$ as in line 3 of the real game. $\mathbb{B}$ gets the set V of dishonest voters chosen by $\mathbb{A}$.

The next step for $\mathbb{B}$ is to call the `Register` function to get the n_V secret parts of the credentials $c_1, \cdots, c_{n_v}$ and the corresponding public list $\boldsymbol{R}$. The secret credentials of the corrupted voters are given to $\mathbb{A}$, as in line 5 of the real game. $\mathbb{B}$ gets in return the voter j that $\mathbb{A}$ chooses to coerce, and the voting option β that models the initial intention of the coerced voter.

In the ideal game, $\mathbb{B}$ can then send the same choices for V, j and β.

Afterwards, $\mathbb{B}$ calls $\mathbb{A}$ as in line 18 of the real game and gives it the real credential $\tilde{c} = c_j$ of the coerced voter. From the ideal game, $\mathbb{B}$ gets the size of the ideal board $|B|$ and will start to emulate the board BB for the real game, initialized to the empty set.

To simulate the for loop in lines 20-22 of the real game, $\mathbb{B}$ performs the following steps, $|B|$ times.

- $\mathbb{B}$ calls $\mathbb{A}$ with input the current BB to get $\boldsymbol{M}$.
- For all valid ballots in $\boldsymbol{M}$, $\mathbb{B}$ decrypts the voting option ν and the credential c using sk and adds the ballot to BB. If c corresponds to some credential c_i, it sets ν_i to ν (erasing any preexisting value).
- $\mathbb{B}$ chooses a random voter and a random valid voting option and adds a ballot to BB using `Vote`.

The lines 23 and 24 are simulated in the same way.

In line 19 of the ideal game, $\mathbb{B}$ returns $\{\nu_i; i > n_V - n_A\}$, all taken from the real game, and β', which models the voting option that the coercer wants to enforce. It can be read by $\mathbb{B}$ in the real game by taking $\beta' = \nu_j$. Then, still in the ideal game, $\mathbb{B}$ gets $\boldsymbol{X}$ (line 25), which is the result of the tally. We denote by $|\boldsymbol{X}|$ the size of this tally.

At this point, in the real game, $\mathbb{B}$ must simulate the cleansing phase to $\mathbb{A}$. We refer to the full description of CHide in Section III, where this phase is decomposed into 7 elementary steps.

The first five steps consists of a sequence of calls to `CGate`. The calls can be simulated thanks to the fact that `CGate` is a SUC-secure protocol, as shown in [9]. SUC refers to a Simpler version of the Universally Composable framework [6], a composition framework inspired from the Universally Composable framework [5] and well-suited for MPC protocols. The principle is that an MPC protocol π that is SUC-secure w.r.t. a functionality Φ can be emulated simply by accessing the ideal functionality Φ. The strength of this approach is that protocols can then easily be composed, preserving security w.r.t. the composed functionalities. More precisely, we consider $T_{\texttt{CGate}}$ be the ideal functionality which takes as input two encryptions of either 0 or 1, decrypts them and outputs a random encryption of their conjunction. It is shown in [9] that there exists $\mathcal{S}$ which can simulate (in the SUC setting) the real `CGate` protocol to $\mathbb{A}$, given oracle access to $T_{\texttt{CGate}}$. Since $\mathbb{B}$ has the secret key sk, it can act as $T_{\texttt{CGate}}$ and hence simulate the `CGate` protocols using $\mathcal{S}$.

Then comes a mixnet step, that $\mathbb{B}$ can simply run for the honest authorities, letting $\mathbb{A}$ play the dishonest trustees.

To simulate the seventh and last cleansing step, $\mathbb{B}$ simply chooses $|\boldsymbol{X}|$ entries at random among all the $(C_1'^i, I'^i)$ and simulate their partial decryption so that I'^i is decrypted into 0, and simulate the other partial decryptions so that I'^i is decrypted into 1. This way there is exactly $|\boldsymbol{X}|$ valid ballots in the simulated game, as in the ideal game.

Finally, to simulate the tally, $\mathbb{B}$ simulates all decryptions so that the result is exactly $\boldsymbol{X}$. This result and the corresponding simulated proofs are sent to $\mathbb{A}$ which gives a guess bit in return. This guess bit is forwarded by $\mathbb{B}$ in the ideal game.

We briefly explain why $\mathbb{B}$'s simulation is indistinguishable from the real game. The first difference is that in line 18 of the real game, $\mathbb{A}$ gets the real κ-bit credential and the real DVZKP when $b = 1$, and a random κ-bit credential and a fake DVZKP when $b = 0$. In the simulation, $\mathbb{A}$ always has the real credential and the real DVZKP. Since the real DVZKP is indistinguishable from the fake and since $\tilde{c}$ is uniformly random in both cases, $\mathbb{A}$ can distinguish the simulation from the real game if and only if it can tell whether $\tilde{c}$ is the plaintext of one of the encrypted credential or not, which is unfeasible under DDH assumption. (Recall that the ElGamal encryption scheme is IND-CPA under DDH assumption, even against an adversary who can corrupt up to a threshold of key holders.)

The second difference is that, during the loop where the voters submit their votes, $\mathbb{B}$ adds a ballot from a random

voter for a random voting option while, in the real game, the ballot should correspond to the board B. Since the voter's credential and the voting option are encrypted, $\mathbb{A}$ cannot tell the difference under DDH assumption.

A third difference is that, during the cleansing phase, $\mathbb{B}$ uses the SUC simulator instead of the real `CGate` protocol. However, by SUC-security, the simulator is indistinguishable from the real protocol.

Another difference is that, in the simulation, the result consists of all the final valid ballots cast by each honest voter and the adversary. In the real game, the result of the tally can be different if the last valid ballot cast by an honest voter is discarded during the cleansing phase, which can only occur if the adversary manages to send a valid ballot with the credential of an honest voter. However, to submit a valid ballot, the adversary has to create a valid proof of knowledge of the credential used. Under DDH assumption and in the ROM, the encryption scheme ElGamal + PoK [4] is NM-CPA, so that submitting a valid ballot which has the same credential as an honest voter is unfeasible.

Finally, the only remaining differences is that $\mathbb{B}$ simulates the result of the decryption protocols, at the end of the cleansing and during the final tally. This difference is not noticeable in the ROM under the DDH assumption. □

We can also check that JCJ is not coercion-resistant according to our definition.

Proposition 1. *The JCJ protocol is not coercion-resistant w.r.t. Definition 2.*

This is again due to the fact that in the real game, among the total number of discarded ballots, the attacker can observe the number of ballots removed due to an invalid credential or due to revote.

V. A weaker definition of coercion-resistance

It may seem unfair to declare JCJ not coercion-resistant "just" because the adversary can have a better information on the number of ballots with fake credentials and hence she may better guess whether the coerced voter has disobeyed or not. Interestingly, our definition can easily be adapted to tolerate some leakage, when there is a consensus that such a leakage is acceptable by the voters. In particular, if we believe that it is unimportant that the adversary learns the sequence of revotes, we can simply pass this information in the ideal game.

In Definition 3, we define a weaker notion of coercion-resistance. The real game is left unchanged and is repeated just to help the comparison. In the ideal game, we generate a pseudonym c_i for each voter i occurring in B. The first voter who votes is given pseudonym 1, the second one is given pseudonym 2, and so on. The rest of the game is left unchanged except that in line 20, we leak in $\boldsymbol{I}$ which ballots correspond to the same credential (invalid or not). More formally, $\boldsymbol{I}$ is a sequence of pseudonyms, each pseudonym corresponding to the voter that has voted at this step. This is exactly what can be observed in JCJ.

A voting system is weakly coercion-resistant if an adversary does not have a better advantage in the real game than in the modified ideal one.

Definition 3. *A voting system is* weakly coercion resistant *if for all PPT adversary $\mathbb{A}$, for all parameters n_T, t, n_V, n_A, n_C, and for all distribution $\mathcal{B}$, there exists a PPT adversary $\mathbb{B}$ and a negligible function μ such that*

$$|\Pr(\mathsf{Ideal}_{\mathsf{W}}(\mathbb{B}, \kappa, n_V, n_A, n_C, \mathcal{B}) = 1) - \Pr(\mathsf{Real}(\mathbb{A}, \kappa, n_T, t, n_V, n_A, n_C, \mathcal{B}) = 1)| \leq \mu(\kappa)\,.$$

We can prove that JCJ satisfies this relaxed version of coercion-resistance.

Theorem 2. *Under the DDH assumption and in the Random Oracle Model, the JCJ protocol is weakly coercion-resistant.*

The proof follows the same lines than the proof of Theorem 1 and is given in appendix.

VI. Discussion

We conclude by discussing how our refined notion of coercion-resistance can be adapted, in principle, to emphasize the various quality of coercion-resistance provided by JCJ-like schemes.

To illustrate our purpose, we start with the coercion-resistant scheme presented by Araújo, Foulle and Traoré (AFT) in [1]. This is a scheme whose main feature is that it has a linear time complexity for the cleansing and tallying phase. While they use different cryptographic primitive from JCJ, their scheme has a similar structure: voters are given credentials to vote with, and can provide a fake credential to a coercer. Assuming that the cryptography used in their scheme is perfect, let us analyze the leakage and compare it with that of JCJ.

During the tally, both the number of duplicates and the number of ballots which use a fake credential are revealed, just as in JCJ. In addition, it is possible to deduce, by observing the board, how many revotes each ballot has. In JCJ, this information is only available during the tally, when it is no longer possible for the adversary to submit a ballot. In the AFT scheme, this information is available on the fly, during the whole voting phase, and the adversary may exploit it to submit ballots in a specific way. Consequently, the AFT scheme provides a coercion-resistance level which is similar to Definition 3, but where $\boldsymbol{I}$ is given to $\mathbb{A}$ at line 19 instead of line 26 in the ideal game. This definition is slightly (but strictly) weaker.

Another interesting example is Civitas [8], a scheme considered as an implementation of JCJ, which has a similar level of security regarding coercion-resistance. Among the few differences that could have an impact, let us concentrate again on the leakage during the cleansing and tallying phase. Interestingly, Civitas actually leaks more information than JCJ. First, Civitas provides the same leakage as the AFT protocol: the number of revotes for each ballot can be directly deduced from the board. Furthermore, in order to reduce the (quadratic) number of PETs, Civitas proposes to group voters by blocks:

Algorithm 5: Real

Require: $\mathbb{A}, \kappa, n_T, t, n_V, n_A, n_C, \mathcal{B}$

$BB \longleftarrow \varnothing$
$pk, s_1, h_1, \cdots, s_{n_T}, h_{n_T} \longleftarrow P^{\mathbb{A}}_{\texttt{Setup}}(\kappa, n_T, t)$
$V \longleftarrow \mathbb{A}()$
$\{c_i; i \in [1, n_V]\}, \boldsymbol{R} \longleftarrow \texttt{Register}(\kappa, \mathsf{pk}, n_V)$
$(j, \beta) \longleftarrow \mathbb{A}(\{c_i; i \in V\}, \boldsymbol{R})$
if $|V| \neq n_A \vee j \notin [1, n_V] \backslash V \vee \beta \notin [1, n_C] \bigcup \{\phi\}$ **then**
  Return 0
$B \longleftarrow \mathcal{B}(n_V - n_A, n_C)$
for $(i, *) \in B$, $i \notin [1, n_V]$ **do**
  $c_i \longleftarrow \texttt{Fakecred}(c_1)$
$b \xleftarrow{\$} \{0, 1\}$
$\tilde{c} \longleftarrow c_j$
if $b == 1$ **then**
  Remove all $(j, *) \in B$
else
  Remove all $(j, *) \in B$ but the last, which is replaced by (j, β)
  $\tilde{c} \longleftarrow \texttt{Fakecred}(c_j)$
$\mathbb{A}(\tilde{c})$
for $(i, \alpha) \in B$ *(in this order)* **do**
  $\boldsymbol{M} \longleftarrow \mathbb{A}(BB)$
  $BB \longleftarrow BB \bigcup \{m \in \boldsymbol{M} \mid m \texttt{ is valid}\}$
  $BB \longleftarrow \{\texttt{Vote}(c_i, \alpha, \mathsf{pk})\}$
$\boldsymbol{M} \longleftarrow \mathbb{A}(BB)$
$BB \longleftarrow BB \bigcup \{m \in \boldsymbol{M} \mid m \texttt{ is valid}\}$
$\boldsymbol{X}, \Pi \longleftarrow P^{\mathbb{A}}_{\texttt{tally}}(BB, \boldsymbol{R}, \mathsf{pk}, \{h_i, s_i\}, t)$
$b' \longleftarrow \mathbb{A}()$
Return 1 **if** $b' == b$ **else** 0

Algorithm 6: $\mathsf{Ideal}_{\mathsf{W}}$

Require: $\mathbb{A}, \kappa, n_V, n_A, n_C, \mathcal{B}$

$k \longleftarrow 1$
$V \longleftarrow \mathbb{A}()$

$(j, \beta) \longleftarrow \mathbb{A}()$
if $|V| \neq n_A \vee j \notin [1, n_V] \backslash V \vee \beta \notin [1, n_C] \bigcup \{\phi\}$ **then**
  Return 0
$B \longleftarrow \mathcal{B}(n_V - n_A, n_C)$
for $(i, *) \in B$ **do**
  if $c_i == \perp$ **then** $c_i \longleftarrow k$; $k \longleftarrow k + 1$
$b \xleftarrow{\$} \{0, 1\}$

if $b == 1$ **then**
  Remove all $(j, *) \in B$
else
  Remove all $(j, *) \in B$ but the last, which is replaced by (j, β)

$(\nu_i)_{i \in V}, \beta' \longleftarrow \mathbb{A}(|B|)$
$\boldsymbol{I} \longleftarrow \{c_i; (i, *) \in B\}$ in this order, with duplicates
if $(b == 1) \wedge (\beta' \neq \phi)$ **then**
  $B \longleftarrow B \bigcup \{(j, \beta')\}$
$B \longleftarrow B \bigcup \{(i, \nu_i); i \in V, \nu_i \in [1, n_C]\}$

$\boldsymbol{X} \longleftarrow \texttt{result}(\texttt{cleanse}(B))$
$b' \longleftarrow \mathbb{A}(\boldsymbol{X}, \boldsymbol{I})$
Return 1 **if** $b' == b$ **else** 0

Fig. 4. Definition of weak coercion-resistance. κ is the security parameter, n_T the number of talliers, t the threshold, n_V the number of voters, n_A the number of corrupted voters, n_C the number of voting options and $\mathcal{B}$ the distribution of the sequence of votes.

each credential is publicly assigned to one block, and the voter indicates their block in clear when casting their ballot. To summarize, compared to JCJ, the adversary still learns how many revotes each ballot has and how many invalid ballots there is, but also has access to this information block by block. Modelling the exact security of Civitas would require to weaken the coercion-resistance definition compared to the one we sketched for AFT. In particular, the definition would have to take the number of blocks as a parameter, so that the ideal game could leak a list of K information sets similar to $\boldsymbol{I}$.

Finally, for voting schemes that are not based on JCJ, the adaptation is less immediate. For instance, in the VoteAgain system [18], the paradigm for coercion-resistance is different, since the voters are assumed to be able to vote after the coercer. The idea of revoting is key to the security and needs to be reflected in the definition of coercion-resistance by preventing the adversary to vote at any time. Even though the situation is too different from what we have presented in our work to be applied directly, the amount of information revealed during the cleansing phase should also be carefully assessed when analysing its resistance to coercion.

References

[1] R. Araújo, S. Foulle, and J. Traoré. A practical and secure coercion-resistant scheme for remote elections. In *Frontiers of Electronic Voting*. IBFI, 2007.

[2] S. Baloglu, S. Bursuc, S. Mauw, and J. Pang. Provably Improving Election Verifiability in Belenios. In *6th International Joint Conference on Electronic Voting (E-Vote-ID'21)*. Springer, 2021.

[3] J. Benaloh, T. Moran, L. Naish, K. Ramchen, and V. Teague. Shuffle-sum: coercion-resistant verifiable tallying for STV voting. *IEEE Trans. Inf. Forensics Secur.*, 4(4):685–698, 2009.

[4] D. Bernhard, O. Pereira, and B. Warinschi. How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. In *Advances in Cryptology (ASIACRYPT'12)*. Springer, 2012.

[5] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd Annual Symposium on Foundations of Computer Science (FOCS'01)*. IEEE Computer Society, 2001.

[6] R. Canetti, A. Cohen, and Y. Lindell. A simpler variant of universally composable security for standard multiparty computation. In *35th Annual Cryptology Conference (CRYPTO'15)*, volume 9216 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2015.

[7] J. Clark and U. Hengartner. Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance. In *15th International Conference on Financial Cryptography and Data Security (FC'11)*. Springer, 2011.
[8] M. Clarkson, S. Chong, and A. Myers. Civitas: Toward a Secure Voting System. In *IEEE Symposium on Security and Privacy (S&P'08)*. IEEE Computer Society, 2008.
[9] V. Cortier, P. Gaudry, and Q. Yang. A toolbox for verifiable tally-hiding e-voting systems. Cryptology ePrint Archive, Report 2021/491, 2021. https://ia.cr/2021/491.
[10] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT'99)*. Springer, 1999.
[11] T. Haines and B. Smyth. Surveying definitions of coercion resistance. Cryptology ePrint Archive, Report 2019/822, 2019. https://ia.cr/2019/822.
[12] L. Hirshi, L. Schmid, and D. Basin. Fixing the Achilles Heel of E-Voting: The Bulletin Board. In *34th IEEE Computer Security Foundations Symposium (CSF'21)*. IEEE Computer Society, 2021.
[13] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated Verifier Proofs and Their Applications. In *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT'96)*. Springer, 1996.
[14] A. Juels, D. Catalano, and M. Jakobsson. Coercion-Resistant Electronic Elections. In *ACM Workshop on Privacy in the Electronic Society (WPES'05)*. ACM, 2005.
[15] R. Küsters, J. Liedtke, J. Müller, D. Rausch, and A. Vogt. Ordinos: A Verifiable Tally-Hiding E-Voting System. In *IEEE European Symposium on Security and Privacy (EuroS&P'20)*. IEEE Computer Society, 2020.
[16] R. Küsters, T. Truderung, and A. Vogt. A Game-Based Definition of Coercion-Resistance and Its Applications. In *23rd IEEE Computer Security Foundations Symposium (CSF'10)*. IEEE Computer Society, 2010.
[17] P. Locher, R. Haenni, and R. E. Koenig. Coercion-Resistant Internet Voting with Everlasting Privacy. In *20th International Conference on Financial Cryptography and Data Security (FC'16)*. Springer, 2016.
[18] W. Lueks, I. Querejeta-Azurmendi, and C. Troncoso. Voteagain: A scalable coercion-resistant voting system. In *USENIX Security Symposium (USENIX'20)*, 2020.
[19] Ü. Madise and T. Martens. E-voting in Estonia 2005. The first Practice of Country-wide binding Internet Voting in the World. In *2nd International Workshop in Electronic Voting (EVOTE'06)*. GI, 2006.
[20] E. McMurtry, O. Pereira, and V. Teague. When is a test not a proof? In *25th European Symposium on Research in Computer Security (ESORICS'20)*, pages 23–41. Springer, 2020.
[21] B. Schoenmakers and P. Tuyls. Practical Two-Party Computation Based on the Conditional Gate. In *Advances in Cryptology (ASIACRYPT'04)*. Springer, 2004.
[22] O. Spycher, R. Koenig, R. Haenni, and M. Schläpfer. Achieving Meaningful Efficiency in Coercion-Resistant, Verifiable Internet Voting. In *5th International Conference on Electronic Voting (EVOTE'12)*. GI, 2012.
[23] O. Spycher, R. E. Koenig, R. Haenni, and M. Schläpfer. A New Approach towards Coercion-Resistant Remote E-Voting in Linear Time. In *15th International Conference on Financial Cryptography and Data Security (FC'11)*. Springer, 2011.
[24] D. Wikström. A Commitment-Consistent Proof of a Shuffle. In *Information Security and Privacy (ACISP'09)*. Springer, 2009.

Appendix A: Cryptographic primitives

The encryption scheme used is ElGamal in a group of order q. A message m is encrypted as: $\texttt{Enc}(m, r, \mathsf{pk}) = (g^r, g^m\,\mathsf{pk}^r)$, where r is a random value in $\mathbb{Z}_q$ and pk is the public key. The randomness can be left implicit by using the notation $\texttt{Enc}(m, \mathsf{pk})$. We denote by E_0, E_1 and E_{-1} the encryptions of 0, 1 and -1 using the current public key pk and the randomness 0.

Re-encryption can be done as usual by multiplying by an encryption of 0, using the homomorphy property of ElGamal. We therefore denote $\texttt{ReEnc}(X, r, \mathsf{pk}) = X\,\texttt{Enc}(0, r, \mathsf{pk})$, where X should be a ciphertext $X = \texttt{Enc}(m, r_0, \mathsf{pk})$. The result is equivalent to $\texttt{Enc}(m, r + r_0, \mathsf{pk})$.

We assume that pk has been produced by a DKG, and that the participants who own the shared decryption key can apply a decryption protocol `Dec`, that additionally produces a ZKP of correct decryption.

The framework in which we do the security proof is universal composability. This leads to technical constraints and therefore the protocols often finish with a joint re-randomization step performed by the participants, which guarantees that no single participant can force any specific form of the output.

In all the following algorithms, we denote by a the number of participants and pk the public encryption key. Typically, a is the number of registrars n_R in the `RandBit` protocol, and the number of talliers n_T in the `CGate` protocol. All the algorithms require $O(a)$ exponentiations and $O(1)$ rounds of communications between the participants. The size of the verification transcript is in $O(a)$ times the size of a group element.

Algorithm 7: `RandBit`
The participants don't need the shared decryption key.

1 $X_0 \longleftarrow E_0$
2 **for** $i = 1$ *to* a **do**
3 Participant i chooses $r_i \xleftarrow{\$} \mathbb{Z}_q$ and $s_i \xleftarrow{\$} \{0, 1\}$
4 They compute $X_i = X_{i-1}(E_1/X_{i-1}^2)^{s_i}\texttt{Enc}(0, r_i, \mathsf{pk})$
5 They reveal X_i and a ZKP π_i of well-formedness
6 Each participant verifies all the proofs
7 The participants jointly rerandomize X_a into X', which produces a transcript $\Pi^{\texttt{ReEnc}}$
8 **return** X' and verification transcript $\Pi^{\texttt{ReEnc}}||(X_a, \pi_a)||\cdots||(X_1, \pi_1)$

In the context of the CHide protocol, the registrars use `RandBit` to produce the credentials. Therefore, they keep the values r_i and s_i picked at line 3. The s_i are sent to the voter, who can then reconstruct the cleartext bit corresponding to X' by computing the exclusive or of all the s_i.

The registrars also send a DVZKP that the public transcript corresponds to the values s_i. This works as follows. Each registrar will produce a ZKP that disjunction of two statements is true. The first one is the knowledge of the secret key of the voter, and the second one is the knowledge of r_i such that the statement on line 4 is correct, for the given value of s_i. The registrar can produce the proof because they know the value r_i, and the voter can produce the proof because they know their secret key. We do not give further details, since this is a rather classical Chaum-Pedersen ZKP, very similar to the one used in JCJ and Civitas.

The `CGate` protocol that we present in Algorithm 8 is taken from [21], in the re-reandomized version of [9]. It allows to compute the conjunction (`And`) of two encrypted bits. The `Not` operator can be implemented by homomorphy:

Algorithm 8: `CGate`
The participants need the shared decryption key.

Require: X, Y such that X (resp. Y) is an encryption of x (resp. y), with $y \in \{0, 1\}$
Ensure: Z, an encryption of xy
1 Compute $Y_0 = E_{-1}Y^2$, set X_0 at X
2 **for** $i = 1$ *to* a **do**
3 Participant i chooses $r_1, r_2 \in_r \mathbb{Z}_q$ and $s \in_r \{-1, 1\}$
4 They compute $X_i = \mathtt{ReEnc}(X_{i-1}^s, r_1, pk)$ and $Y_i = \mathtt{ReEnc}(Y_{i-1}^s, r_2, \mathsf{pk})$
5 They reveal X_i, Y_i and a ZKP π_i that X_i and Y_i are well formed
6 We denote $\Pi = (X_a, Y_a, \pi_a)||\cdots||(X_1, Y_1, \pi_1)$
7 Each participant verifies all the proofs
8 The participants jointly rerandomize X_a and Y_a into X' and Y', which produces a transcript $\Pi^{\mathtt{ReEnc}}$
9 They collectively compute $y_a = \mathtt{Dec}(Y')$ and a transcript $\Pi^{\mathtt{Dec}}$.
10 **return** $Z = (XX'^{y_a})^{\frac{1}{2}}$ and verification transcript $(y_a, \Pi^{\mathtt{Dec}})||(X', Y', \Pi^{\mathtt{ReEnc}})||\Pi$

$\mathtt{Not}(X) = E_1/X$. The disjunction (`Or`) can then be easily deduced as $\mathtt{Or}(X, Y) = \mathtt{Not}(\mathtt{CGate}(\mathtt{Not}(X), \mathtt{Not}(Y)))$.

The equality-test `Eq` between two encrypted bits is the negation of the exclusive or. Again, it can be deduced by combining homomorphic properties and the `And`. We propose the following: $\mathtt{Eq}(X, Y) = \mathtt{Not}(XY/\mathtt{CGate}(X, Y)^2)$.

The other cryptographic primitives used in CHide were already present in JCJ and we do not recall them here.

Appendix B: Proof that JCJ is weakly coercion-resistant

We provide here a proof of Theorem 2, namely that the original JCJ scheme is weakly coercion-resistant, according to Definition 3, with the real and ideal games given in Figure 4. This is very similar to the proof of Theorem 1 and some parts are verbatim copies of the proof that CHide is coercion-resistant.

We recall the notations of the definition: n_T is the number of trustees, t is the threshold, n_V is the number of voters, n_C the number of voting options, and n_A the number of dishonest voters.

Let $\mathcal{B}$ be a distribution, and let $\mathbb{A}$ be an adversary that plays the real game. We give to $\mathbb{A}$ the power to impersonate the last t talliers. We construct an adversary $\mathbb{B}$ that plays the ideal game, and that can interact with $\mathbb{A}$ by simulating the real game.

First of all, $\mathbb{B}$ jointly runs the setup with $\mathbb{A}$ to generate the public key pk, the secret shares $s_1, \cdots, s_{n_T}$ and the public commitments $h_1, \cdots, h_{n_T}$. From these shares, $\mathbb{B}$ reconstructs the secret key sk.

Then $\mathbb{B}$ calls $\mathbb{A}$ as in line 3 of the real game. $\mathbb{B}$ gets the set V of dishonest voters chosen by $\mathbb{A}$.

The next step for $\mathbb{B}$ is to call the `Register` function to generate the n_V secret parts of the credentials $c_1, \cdots, c_{n_v}$ and the corresponding public list $\boldsymbol{R}$. The secret credentials of the corrupted voters are given to $\mathbb{A}$, as in line 5 of the real game. $\mathbb{B}$ gets in return the voter j that $\mathbb{A}$ chooses to coerce, and the voting option β that models the initial intention of the coerced voter.

In the ideal game, $\mathbb{B}$ can then send the same choices for V, j and β.

Afterwards, $\mathbb{B}$ calls $\mathbb{A}$ as in line 18 of the real game and gives it the real credential $\tilde{c} = c_j$ of the coerced voter. From the ideal game, $\mathbb{B}$ gets the size of the ideal board $|B|$ and will start to emulate the board BB for the real game, initialized to the empty set.

To simulate the for loop in lines 20-22 of the real game, $\mathbb{B}$ performs the following steps, $|B|$ times.

- $\mathbb{B}$ calls $\mathbb{A}$ with input the current BB to get $\boldsymbol{M}$.
- For all valid ballots in $\boldsymbol{M}$, $\mathbb{B}$ decrypts the voting option ν and the credential c using sk and adds the ballot to BB. If c corresponds to some credential c_i, it sets ν_i to ν (erasing any preexisting value).
- $\mathbb{B}$ chooses a random voter and a random valid voting option and adds a ballot to BB using `Vote`.

The lines 23 and 24 are simulated in the same way.

In line 19 of the ideal game, $\mathbb{B}$ returns $\{\nu_i; i > n_V - n_A\}$, all taken from the real game, and β', which models the voting option that the coercer wants to enforce. It can be read by $\mathbb{B}$ in the real game by taking $\beta' = \nu_j$. Then, still in the ideal game, $\mathbb{B}$ gets $\boldsymbol{X}$ and $\boldsymbol{I}$ (line 25), which is the result of the tally, together with the leakage due to the cleansing phase. We denote by $|\boldsymbol{X}|$ the size of the tally.

Now, $\mathbb{B}$ must simulate the cleansing phase. The first step of the cleansing phase consists of pair-wise PETs. To simulate a PET, $\mathbb{B}$ run the protocol honestly but simulate the decryption so that $\mathbb{B}$ can decide whether the result is 1 (which stands for an equality of the plaintexts), or random (which stands for different plaintexts). To decide the result of a PET, $\mathbb{B}$ denotes a and b the indexes of the credentials to be compared, in BB. The algorithm is as follow.

- If the two corresponding ballots were added by the adversary, $\mathbb{B}$ performs the decryption honestly (the adversary knows the real result of these PETs, so that the simulator cannot lie).
- If a (resp. b) was added by the adversary while b (resp. a) was added by a non-corrupted voter, $\mathbb{B}$ simulates the decryption so that the result of the PET is a random.
- If a and b were added by a non-corrupted voter, $\mathbb{B}$ looks at the indexes a' and b' that they would have in the board if the adversary did not send any ballot. It then simulates the decryption to match the equality test between $I_{a'}$ and $I_{b'}$, so that the result of the PET is exactly the same as in the ideal game.

Then comes a mixnet step, that $\mathbb{B}$ can simply run for the honest authorities, letting $\mathbb{A}$ play the dishonest trustees.

The last step of the cleansing phase consists of pairwise PETs between the ballots of the board and the encrypted

credentials $\boldsymbol{R}$. $\mathbb{B}$ simulates this phase by choosing $|\boldsymbol{X}|$ random ballots of the board which will be valid, and simulate the PETs so that those ballots are marked as valid while the others are marked as invalid.

Finally, $\mathbb{B}$ simulates the tally by simulating the decryptions, so that the result is exactly $\boldsymbol{X}$.

The argument to prove that $\mathbb{B}$'s simulation is indistinguishable from the real game is similar as for the proof of Theorem 1, without the complication due to the use of `CGate`.

We remark that the original PET mechanism used in JCJ had some weakness. For the proof, we assume that the version of [20] is used; the reduction would not be possible with the original one.