A toolbox for verifiable tally-hiding e-voting systems

Véronique Cortier, Pierrick Gaudry, Quentin Yang

Université de Lorraine, Inria, CNRS

March 2022

ABSTRACT

In most verifiable electronic voting schemes, one key step is the tally phase, where the election result is computed from the encrypted ballots. A generic technique consists in first applying (verifiable) mixnets to the ballots and then revealing all the votes in the clear. This however discloses much more information than the result of the election itself (that is, the winners) and may offer the possibility to coerce voters.

In this paper, we present a collection of building blocks for designing tally-hiding schemes based on multi-party computations. As an application, we propose the first tally-hiding schemes with no leakage for three important counting functions: D'Hondt, STV, and Majority Judgment. We prove that they can be used to design a private and verifiable voting scheme. We also unveil unknown flaws or leakage in several previously proposed tally-hiding schemes.

1 INTRODUCTION

Electronic voting is used in many countries and various contexts, from major politically binding elections to small elections among scientific councils. It allows voters to vote from any place and is often used as a replacement of postal voting. Moreover, it enables complex tally processes where voters express their preference by ranking their candidates (preferential voting). In such cases, the votes are counted using the prescribed procedure (*e.g.* Single Transferable Vote or Condorcet), which can be tedious to conduct by hand but can be easily handled by a computer.

Numerous electronic voting protocols have been proposed such as Helios [6], Civitas [18], or CHVote [24]. They all intend to guarantee at least two security properties: *vote secrecy* (no one should know how I voted) and *verifiability*. Vote secrecy is typically achieved through asymmetric encryption: election trustees jointly compute an election public key that is used to encrypt the votes. The trustees take part in the tally, to compute the election result. Only a coalition of dishonest trustees (set to some threshold) can decrypt a ballot and violate vote secrecy. Verifiability typically guarantees that a voter can check that her vote has been properly recorded and that an external auditor can check that the result corresponds to the received votes. Then, depending on the protocol, additional properties can be achieved such as coercion-resistance or cast-as-intended. Various techniques are used to achieve such properties but one common key step is the tally: from the set of encrypted ballots, it is necessary to compute the result of the election, in a verifiable manner.

There are two main approaches for tallying an election in the context of electronic voting. The first one is the *homomorphic tally*. Thanks to the homomorphic property of the encryption scheme (typically ElGamal), the ballots are combined to compute the (encrypted) sum of the votes. Then only the resulting ciphertext needs to be decrypted to reveal the election result, without leaking the individual votes. For verifiability, each trustee produces a zero-knowledge proof of correct (partial) decryption so that anyone can check that the result indeed corresponds to the encrypted ballots. The second main approach is based on *verifiable mixnets*. The encrypted ballots are shuffled and re-randomized such that the resulting ballots cannot be linked to the original ones [24, 47]. A zero-knowledge proof of correct mixing is produced to guarantee that no ballot has been removed nor added. Several mixers are successively used and then each (rerandomized) ballot is decrypted, yielding the original votes in clear, in a random order.

Homomorphic tally can only be applied to simple vote counting functions, where voters select one or several candidates among a list and the result of the election is the sum of the votes, for each candidate. We note that even in this simple case, the tally reveals more information than just the winner(s) of the election. Mixnet-based tally can be used for any vote counting function since it reveals the (multi)set of the initial votes. On the other hand, this is much more information than the result itself and is subject to so-called Italian attacks. Indeed, when voters rank their candidates by order of preference, the number of possible choices can be higher than the number of voters. Hence a voter can be coerced to vote in a certain way by first selecting the first candidates as desired by the coercer and then "signing" her ballot with some very particular order of candidates, as prescribed by the coercer. The coercer will check at the end of the election that such a ballot appears.

Recent work have explored the possibility to design tally-hiding schemes, that compute the result of the election from a set of encrypted ballots, without leaking any other information. This can be seen as an instance of Multi-Party Computation (MPC), but the context of voting adds some constraints. First, a voter should only produce one encrypted ballot that should remain of reasonable size and be computed with low resources (*e.g.* in JavaScript). The trustees can be assumed to have more resources. Yet, it is important to minimize the number of communications and the computation cost, whenever possible. In particular, voters should not wait for weeks before obtaining the result. Moreover, all proofs produced by the authorities need to be downloaded and verified by external, independent auditors. It is important that verifying an election remains affordable.

Related work. Even when the winner(s) of the election is simply the one(s) that received the most votes, leaking the scores of each candidate can be embarrassing and even lower vote privacy. This is discussed in [28] where the authors propose a protocol called Ordinos that computes the candidate who received the most votes, without any extra information. In case of preferential voting, where voters rank candidates, several methods can be applied to determine the winner(s). Two popular methods are Single Transferable Vote (STV) and Condorcet. STV is used in politically binding elections in several countries like Australia, Ireland, or UK. Condorcet has several variants and the Schulze variant is popular among several associations like Ubuntu or GnuGP. These are the counting methods offered by the voting platform CIVS [1] and used in many elections. Literature for tally-hiding schemes includes [25] which shows how to compute the result in Condorcet, while [44] and [10] provide several methods for STV. They all leak some partial information, but much less than the complete set of votes. Ordinos has been extended [27] to cover various counting functions that include Borda, Hare-Niemeyer, Condorcet, and Instant-Runoff Voting (IRV, a particular case of STV, where there is only one seat). This shows the flexibility of Ordinos, yet at a cost: ballots are of size cubic in the number of candidates for Condorcet-Schulze and even super-exponential for IRV. The last system we study, Majority Judgment (MJ) is a vote system where voters give a grade to each candidate (typically between 1 and 6). The winner is, roughly, the candidate with the highest median rating. Since typically several candidates have the same median, the winner is determined by a complex algorithm that iteratively compares the highest median, then the second one and so on (see [7] for the full details). In [15], the authors show how to compute Majority Judgment in MPC. All these approaches except [25] rely on Paillier encryption since it is better suited than ElGamal for the arithmetic comparison of the content of two ciphertexts.

Our contribution. First, we revisit the existing work, exhibiting weaknesses and even flaws for some of them. Second, we provide new algorithms for computing vote counting functions, decreasing both the complexity and the leakage or proposing other trade-offs regarding the load for the voters and the trustees. In particular, we propose the first tally-hiding schemes with no leakage for three major counting functions: D'Hondt, Majority Judgment, and STV. For Condorcet-Schulze, we propose the first tally-hiding scheme that allows candidates to be ranked at equality. We summarize our main contributions in the following table.

Single vote	- Fix shortcoming in [28] in case of equality - Adaptation to D'Hondt method
Majority Judgment	- Fix the fact that [15] fails in not-so-rare cases - Complete leakage-free algorithm, based on ElGamal
Condorcet - Schulze	- Fix privacy issue in [25] - Candidates can be ranked at equality (unlike [27]) - Quasi-linear complexity for voters (vs cubic in [27]) - Several efficiency/leakage compromises - Complete leakage-free algorithm
STV	- Ideal STV has exponential worst-case complexity - Complete leakage-free algorithm, with fast arithmetic

One of our first findings is that even for complex counting functions, it is possible to use ElGamal encryption instead of Paillier. This offers several advantages. ElGamal encryption can be implemented on elliptic curves, yielding building blocks of much lower complexity than Paillier's encryption for the same security level. Moreover, in the context of voting, it is important to split the decryption key among several trustees so that no single authority can break vote privacy. It is easy to set up threshold decryption in ElGamal, with an arbitrary threshold of trustees needed for decryption [19]. The situation is more complex in Paillier. The general threshold key distribution scheme [26] is of high complexity. A more efficient scheme exists [34], but only with a honest majority.

Another reason for preferring ElGamal could be that the underlying security assumption (Decisional Diffie Hellman) can be considered as more standard than the one for Paillier (Decisional n-Residuosity). Finally, from a practical point of view, it is also easier to find standard software libraries that include support for ElGamal encryption.

We have considered several families of counting methods, that include complex ones (e.g. STV, MJ), to demonstrate that it is possible to build efficient MPC schemes for such vote counting functions, using standard ElGamal encryption.

Single vote. A first class of counting functions applies to the case where voters simply select one (or several) candidate(s). The typical way to determine the s winners is to count the number of votes for each candidate and select the s candidates with the most votes. This is exactly the case covered by Ordinos [28]. There is however a shortcoming in case of equalities: the function implemented in Ordinos may return more winners than the number of seats. We correct this by providing an algorithm that computes exactly the winners according to the election rule, without leaking any extra information. Moreover, we show that it is possible to rely on ElGamal with the associated benefits discussed earlier, thanks to an adapted algorithm. This lowers the size of a ballot for voters at a higher cost for the authorities, which can be preferred in practice.

Things get more complex when voters select a candidate list instead of a candidate. The seats need to be shared among the candidates of the different lists, according to number of votes received. One popular technique is the D'Hondt method, that can be adapted to several variants depending on whether the election system wishes to favor big or small parties. We extend the approach initiated by Ordinos to the case of D'Hondt, building on two main ideas: the use of a more advanced algorithm and a more efficient primitive for comparison, inspired from circuits. We propose two different compromises in terms of computations and communications cost. We study the cost of relying on either Paillier and ElGamal and in this case, ElGamal is a key ingredient for designing a practical tally-hiding scheme.

Majority Judgment. The idea of Majority Judgment [7] is that candidates should not be ranked but instead should each be judged independently. We found out that [15] actually only implements a simplified version of the Majority Judgment method, called majority gauge. When the majority gauge returns a winner, then it is indeed a MJ winner. Unfortunately, in small elections, there is a rather high probability that the simplified algorithm does not provide any result. For example, in an election with 100 voters, [15] would fail with probability 20%, which not only is inconvenient (imagine an election that must be canceled because no winner is declared!) but also leaks some information (there is no winner according to the majority gauge).

To repair the approach, one issue is that the complexity of the MJ algorithm depends (linearly) in the number of voters, which may be large. Hence, [7] devises an alternative (complex) algorithm that no longer depends on the number of voters. We propose a variant of this algorithm and use it as a basis to derive a tally-hiding procedure for MJ. Our resulting algorithm remains of a complexity similar to [15] while they implement a much simpler algorithm. Then we show that it is actually possible to adapt our algorithm to ElGamal encryption. Interestingly, the format remains unchanged for the voter (hence the resulting ballot is even easier to compute).

A key idea is to work in the bit-encoding of integers, which allows us to perform all the needed operations (additions, comparisons) on ElGamal encryptions. The load for the trustees increases (since comparisons are more complex) but our study shows that it remains reasonable since the extra operations are more or less compensated by the fact that computations are faster in ElGamal.

Condorcet. A Condorcet winner is a candidate that would win against each of her opponents. In some cases, there is no Condorcet winner, and several variants exist to further determine a winner in such a case. In [25] a tally-hiding algorithm is proposed for the Condorcet-Schultze variant. However, we found out that [25] is subject to a major privacy flaw. Indeed, everyone learns, for each voter, how many candidates have been placed at equality. Hence Alice completely loses her vote privacy if she votes blank, which cannot be acceptable. This flaw has been acknowledged by the authors. [27] simply disallows equalities while this is widely used in Condorcet elections. Moreover, the authors propose a ballot format that is cubic in the size of candidates.

To cover the fact that candidates can be ranked equal, we had to solve a difficult question: voters need to prove, at a reasonable cost, that they encrypted a meaningful ballot, that is, a ballot that corresponds to an order. We considered two main ingredients here. First, we devised a new encoding for ballots. Second, we used mixnet in an original way: a voter proves that her ballot is valid by showing that her ballot can be obtained as a permutation of a valid (public) ballot. Here, the permutation encodes the voter's choice and the voter is her own mixer. We then devise several algorithms (all based on ElGamal) with different compromises in terms of load balance between the voters and the trustees and in terms of leakage.

STV. In a first round of STV, if a candidate has been ranked in the first place sufficiently often (more than a quota), then the candidate obtains a seat. However, if she obtained more votes than the quota, the exceeding votes should not be lost. Instead, they should be transferred to the next candidate. Hence a fraction of votes (which corresponds to the exceeding votes) is transferred to the second preferred candidate of each voter. The process is repeated until all the seats are filled. Since it is not easy to compute by hand the fractions that need to be transferred, many variants of STV exist where the fractions can be rounded or where the votes can be transferred to randomly selected ballots.

Our first goal was to implement a tally-hiding algorithm for the ideal STV. However, we discovered that even without any cryptography, the pure STV algorithm is exponential and far from being practical. On real data elections from the South New Wales election in Australia [3], the pure STV algorithm (on clear votes) would take about one month on a personal computer to compute the result. We believe that this issue was not well understood.

Given that ideal STV cannot be efficiently computed even in the clear, we considered a variant with rounding. In [10, 44], there are in total three techniques to compute the STV winners, all with some leakage (for example, the current score of the selected candidate). Note that [44] computes the ideal STV (with no rounding) but probably because the authors did not realize that it would quickly be impractical. [27, 38] cover a particular case of STV where only one candidate is elected (IRV). Note that [27] uses a naive encoding of the possible choices: if there are c candidates, they view the $c!$ possible orders as $c!$ possible "candidates" from which a voter makes a selection, yielding a ballot of super-exponential size, while the ballot size is $O(c^2)$ in [38]. We propose a fully tally-hiding algorithm for STV, with no leakage, at a cost similar to [10, 44]. To keep the cost reasonable, we re-used techniques of hardware circuits (*e.g.* to reduce the length of the carry-chains in additions).

Security proof and implementation. The Paillier setting of our toolbox builds upon the same low-level primitive as previous works. However, in the ElGamal setting that we found to be highly relevant, the core ingredient is a different primitive called `CGate` (that conditionally sets a component to 0). An important contribution of our work is to formally prove that it is UC-secure and verifiable. Concentrating on this ElGamal setting, this allows us to prove vote secrecy and verifiability of a voting scheme that embeds our tally-hiding protocol.

With the same goal of validating our ElGamal approach, we have implemented our building blocks in a library in this setting. As a proof of concept, we have combined them to form the tally-hiding scheme that corresponds to Condorcet-Schulze. Our experiments show a reasonable execution time. Authorities need a couple of minutes to perform the tally for 5 candidates, and about 9 hours for 20 candidates (and 1024 voters). In contrast, the code [27] developed in the Paillier setting, needed more than 9 days for 20 candidates (and was almost insensitive to the number of voters).

Finally, we emphasize that our toolbox should be suitable to implement any realistic counting method, in addition to the ones we have explicitly studied. For completeness, a rather long Appendix gives fully detailed algorithms and security proofs, and our source code for the implementation is available [4].

2 BUILDING BLOCKS

We focus on the tally phase, common to most voting schemes. We assume a public ballot box that contains the list of encrypted ballots where all the traditional issues up to here have been handled: eligibility, validity of ballots, revoting policy if applicable, and so on. We concentrate on the counted-as-recorded property. We do not assume that the encrypted ballots are anonymous: for example, they could be signed by voters.

Our goal is to compute the winners of the election, while preserving the privacy of the voters, namely with no additional leakage of information about the tally. The decryption key is assumed to be shared among a trustees, with a threshold scheme, and we wish the procedure to produce a transcript such that: 1) if at least a threshold of t trustees is honest, the result will be obtained and only the result is known (no side-information); 2) even if all the a trustees are dishonest, the result is guaranteed to be correct.

This does not come for free and usually involve heavy computations and communications between trustees.

2.1 MPC toolbox

The MPC implementation of counting functions relies on several common building blocks that we define below, such as addition, multiplication, comparison. For each of them, we study their cost. All these costs are summarized in Figure 6 in Appendix. Regarding the computation cost, we count the number of exponentiations. For the communications, sometimes all the trustees need to broadcast their share of the computations, and sometimes they need to perform a round of communications, where one trustee contributes to the data they receive from the previous one in the loop. We will count these two types of communications separately. An important information is also the size of the transcript that is created during the process and that should be checked, for example by auditors, to guarantee that the result is correct.

Beyond defining our needed building blocks, we believe that this study is of independent interest since it could be used in other contexts than voting. It has required to study a rich literature, first on zero-knowledge proofs [14, 30, 37, 47] and MPC [8, 31, 39–41] but also on hardware circuits [13]. Interestingly, we distinguish between the functionality (*e.g.* addition) and the algorithm that realizes it since different algorithms may be considered, leading to different trade-offs in terms of communications and computations. For a few building blocks, we even propose our own algorithms, that improve existing propositions.

Homomorphic property. Both Paillier and ElGamal are homomorphic encryption schemes. This means that multiplication or division of ciphertexts correspond to addition or subtraction of the corresponding cleartexts. We denote these functions Add and Sub; they cost no communications nor exponentiations. This allows re-randomization, by multiplying with $\mathtt{Enc}(0)$. If the encrypted value is a bit, by dividing $\mathtt{Enc}(1)$ by it, this allows to flip the encrypted bit. We denote $\mathtt{Not}(B)$ this function that is essentially for free.

Encoding of encrypted integers. An integer can be directly encrypted. This is simple and we call this *natural encoding* in this paper. It allows to directly add and subtract encrypted values. However, in the ElGamal setting, most of the other operations (comparison, multiplication, …) are more difficult, or even impossible.

The alternative is to encrypt each bit of the integer separately; we call this the *bit-encoding* of an encrypted integer and we denote it $X^{\text{bits}} = (X_0, \ldots, X_{m-1})$, where 2^m is a bound on the integer represented by X, and X_i is the encryption of the i-th bit of the binary expansion (index 0 for the least significant bit). Converting an integer in bit-encoding to natural encoding is easily done using the homomorphic property and the Horner scheme. The other direction is harder (in the Paillier setting) or impossible (in the ElGamal setting).

Branch-free tools. In MPC, the algorithms must be implemented in a branch-free setting, because the result of a test cannot be revealed (unless we allow a partial leakage). The classical building-blocks for this are conditional operations.

- $\mathtt{CondSetZero}(X, B), \mathtt{CondSetZero}^{\text{bits}}(X^{\text{bits}}, B)$: conditionally set to zero. This function returns (a re-encryption of) X if B is an encryption of 1, or $\mathtt{Enc}(0)$ if B is an encryption of 0. In the bit-encoding setting, each bit of X is re-encrypted or set to zero.
- $\mathtt{Select}(X, Y, B)$, $\mathtt{Select}^{\text{bits}}(X^{\text{bits}}, Y^{\text{bits}}, B)$: select according to bit. This function returns (a re-encryption of) X if B is an encryption of 0 and Y if B is an encryption of 1.
- $\mathtt{SelectInd}([X_i], [B_i])$: select in array according to bits. This function returns (a re-encryption of) X_i such that B_i is an encryption of 1. This requires that the encrypted bit array $[B_i]$ is such that that there is only one index i for which B_i is 1.

The CondSetZero function is the main primitive from which all the others can be easily derived using the homomorphic property. For instance, $\mathtt{Select}(X, Y, B)$ can be implemented as $\mathtt{Add}(\mathtt{CondSetZero}(\mathtt{Sub}(Y, X), B), X)$. In the context of ElGamal encryption, it costs one round of communication at each use.

Arithmetic. As already said, by homomorphy, addition and subtraction of encrypted values are built-in functionalities when the natural encoding is used. However the same operations with the bit-encoding become more involved. Several variants can be considered, the most classical being to have all the operations defined modulo 2^m where m is the number of encrypted bits. Sometimes it is useful to return the final carry / borrow bits. Comparison of two integers is denoted by LT. In bit-encoding, it can be seen as a subtraction where only the final borrow is needed, but in the natural-encoding, the borrow is not available, and a dedicated algorithm must be designed, only available in the Paillier scheme. Similarly we define the Mul function that can be applied to integers in both encoding, with the exception that the natural-encoding multiplication is available only in the Paillier scheme. Finally, a frequent operation is to compute the sum of many encrypted values each containing a bit, typically to get the total number of votes for a given option. We call this operation Aggreg. Again with homomorphic encryption, this is very cheap. However, especially in the ElGamal setting, it could be that the result is wanted in the bit-encoding format. Then a dedicated tree-based algorithm, with variable bit-precision can be designed to improve the complexity compared to naively using the Add function with the maximum precision.

- $\mathtt{Add}(X, Y)$, $\mathtt{Add}^{\text{bits}}(X^{\text{bits}}, Y^{\text{bits}})$: addition of X and Y, in any encoding. In the bit-encoding, the result is taken modulo 2^m.
  $\mathtt{Sub}(X, Y)$, $\mathtt{Sub}^{\text{bits}}(X^{\text{bits}}, Y^{\text{bits}})$: subtraction of X and Y.
- $\mathtt{Aggreg}([X_i])$, $\mathtt{Aggreg}^{\text{bits}}([X_i])$: sum of n binary values X_i. In the bit-encoding, the output contains $\log n$ encrypted bits.
- $\mathtt{LT}(X, Y)$, $\mathtt{LT}^{\text{bits}}(X^{\text{bits}}, Y^{\text{bits}})$: comparison of X and Y in any encoding. Only in the Paillier setting for the natural-encoding.
- $\mathtt{EQ}(X, Y)$, $\mathtt{EQ}^{\text{bits}}(X^{\text{bits}}, Y^{\text{bits}})$: equality test of X and Y in any encoding. Only in the Paillier setting for the natural-encoding.
- $\mathtt{BinExpand}(X)$: binary expansion of X. This function returns X^{bits}. This is available only in the Paillier setting.
- $\mathtt{Mul}(X, Y)$, $\mathtt{Mul}^{\text{bits}}(X^{\text{bits}}, Y^{\text{bits}})$: multiplication of X and Y. Only in the Paillier setting for the natural-encoding.

Since $\mathtt{CondSetZero}(X, B)$ can be seen as an And gate when X is just a bit, with the additional homomorphic operations (Add and Not), this allows to build any arithmetic circuit with bits as input and output. Building all the arithmetic functions with the bit-encoding is therefore a matter of optimizing the circuit design with respect to the number of exponentiations and communications. We discuss more thoroughly these optimizations in Section 6.

In the ElGamal setting, we use the CGate protocol (adapted from [39]) to achieve the CondSetZero functionality (see Algorithm 1). During this protocol, each authority produces a Zero Knowledge Proof (ZKP) that guarantees that the correct computations were performed. The ZKP of all authorities can later form a transcript which can be used to verify the output of the protocol. Since all our arithmetic and logic protocols are based on

CondSetZero, a transcript for verifiability can be obtained for all our MPC protocols. The Appendix C.1 contains more detains on the CGate protocol.

Algorithm 1: CGate

Require: X, Y such that X, Y are encryptions of $x, y \in \{0, 1\}$
Ensure: $Z = \mathtt{Enc}(xy)$
Compute $Y_0 = \mathtt{Enc}(-1)Y^2$, set X_0 at X
for $i = 1$ *to a, for the authority i,* **do**
  Choose $r_1, r_2 \in_r \mathbb{Z}_q$ and $s \in_r \{-1, 1\}$
  Compute $X_i = \mathtt{ReEnc}(X_{i-1}^s, r_1)$ and $Y_i = \mathtt{ReEnc}(Y_{i-1}^s, r_2)$
  Reveal X_i, Y_i and a ZKP that X_i and Y_i are well formed
Each authority verifies the proof of the other authorities
They collectively rerandomize X_a and Y_a into X' and Y'
They collectively compute $y_a = \mathtt{Dec}(Y')$
Return $Z = (XX'^{y_a})^{\frac{1}{2}}$

In the natural-encoding, the strategy is different and available only in the Paillier setting, where it is possible to extract the bits of naturally-encoded integers with an MPC procedure based on masking [40]. This gives an algorithm for BinExpand. Hence, using this conversion, it is possible to compute all the arithmetic operations even if the input are in natural-encoding.

Shuffle and mixnet. A tool that is of great use in our context is verifiable shuffling, leading to mixnets. In electronic voting, the typical use of a verifiable mixnet is during the tally phase, just before decrypting all the ballots, one by one. Our tally-hiding schemes actually makes a thorough use of mixnets, not only on the trustees side but also on the voter's side, as shown in Section 5.

The first building block is Shuffle($[X_i]$). It takes as input an array of encrypted values and output the same (re-encrypted) values in another order that remains secret, together with a zero-knowledge proof that everything was done correctly. As such, this is not an MPC primitive: this is an operation done by just one entity. Chaining a sequence of applications of this procedure by all the trustees, in turn, leads to the Mixnet($[X_i]$) function, that outputs an array of the same re-encrypted values in an order that is secret as soon as at least one trustee is honest.

A variant is to shuffle ballots containing a pairwise comparison matrix. Then, the (secret) permutation used to shuffle the columns should be the same as the one used to shuffle the rows. This leads to the ShuffleMatrix($[M_{i,j}]$) and the MixnetMatrix($[M_{i,j}]$) procedures, and their variants in the bit-encoding.

2.2 UC security

We consider the well-known UC-framework [16] to prove security. A composable framework is particularly suitable to analyze the security of our MPC protocols since we provide building blocks that we combine. We actually use the composition framework from [17], which is a Simpler version of the Universally Composable framework (SUC), shown to imply UC-security. Participants of a protocol P are modeled as Polynomial Probabilistic Turing Machines (PPT). Each of the a participants has a single input and output communication tape, and interacts with a router, which in turn interacts with an adversary $\mathbb{A}$. The adversary interacts with the router and the environment $\mathcal{Z}$. The adversary can corrupt a subset C of participants of size at most t, where $t \le a$ is some threshold. Non-corrupted participants are honest and follow the protocol, while corrupted participants are fully impersonated by the adversary and give away any secret they have. The process terminates when $\mathcal{Z}$ writes on its output tape. We denote $\mathtt{REAL}_{P,\mathbb{A},\mathcal{Z}}(\kappa, z)$ the output, where κ is a security parameter and z is an arbitrary auxiliary input.

The security of the process is guaranteed by a comparison with an ideal process, in which each party hands over their inputs to a trusted party T which honestly performs the desired computation. Corrupted parties may send arbitrary outputs as instructed by the adversary, and the adversary can block or delay communications with the trusted party. Intuitively, T computes some ideal function f, such as Add but it cannot be just a function. Indeed, T additionally takes care of failure cases (for example, when too many parties return inconsistent data). We denote $\mathtt{IDEAL}_{T,\mathcal{S},\mathcal{Z}}(\kappa, z)$ the output of the environment in the ideal process, when it interacts with the adversary $\mathcal{S}$. Intuitively, a protocol is SUC-secure if, for all adversary $\mathbb{A}$ in the real process, there exists a simulator $\mathcal{S}$ in the ideal process such that no PPT environment $\mathcal{Z}$ can tell whether they are interacting with the adversary in the real process or with the simulator in the ideal process.

Definition 2.1 (Secure computation [17]). Let P be a protocol, T some trusted party. We say that P *securely computes* T if, for all PPT $\mathbb{A}$, there exists a PPT $\mathcal{S}$ such that, for all PPT $\mathcal{Z}$, there exists a negligible function μ such that for all κ and all z polynomial in κ,

$$|\Pr(\mathtt{IDEAL}_{T,\mathcal{S},\mathcal{Z}}(\kappa, z) = 1) - \Pr(\mathtt{REAL}_{P,\mathbb{A},\mathcal{Z}}(\kappa, z) = 1)| \le \mu(\kappa).$$

All our building blocks (except shuffle and mixnets, that are handled separately) rely on a single block, namely CondSetZero in the sense that they can all be derived as composition of this function, possibly with intermediate operations using only the homomorphic property of encryption. To compute CondSetZero, we consider an MPC protocol CGate [39] based on ElGamal, and we adapt it in order to prove, in the SUC framework, that CGate securely computes the trusted party $T_{\mathtt{CGate}}$, that behaves as CondSetZero except when parties do not answer, in which case it returns an error. The CGate protocol also produces a transcript which acts as a ZKP that the protocol was performed correctly. The SUC security of the other building blocks then follows by composition. Actually, as detailed in [17], SUC-security is not directly composable but instead requires to introduce intermediary (composable) hybrid models, where participants have magically access to some ideal trusted parties. We could prove by composition of the hybrid models that each of our building blocks securely computes its corresponding ideal trusted party. However, this would require some extra work since our building blocks compute a re-encryption of the desired function (e.g. addition) and hence is not a deterministic function. Instead, we use a different proof strategy: we show that any composition of CGate, followed by a final decryption, is SUC-secure, which corresponds exactly to our needs when applied to tally-hiding schemes. All the precise definitions and proofs are provided in appendix (Part II).

2.3 Paillier vs elliptic ElGamal

As discussed in introduction, when ElGamal encryption can be used, it offers several avantages over Paillier. First, popular elliptic curves like NIST P-256 or Curve25519 are now ubiquitous in cryptographic libraries, while there is in general no support for Paillier. Moreover, threshold key generation is much simpler in ElGamal. Also, ElGamal relies on a well understood security assumption (DDH). In general, an algorithm based on the Paillier scheme requires less exponentiations that when based on ElGamal; however, exponentiations are more costly. In this paper, we will provide the complexity of our algorithm measured by the number of exponentiations. These figures should be compared having in mind the respective cost in ElGamal and in Paillier, that we estimate in this section.

Parameter sizes and cost of operations. For a voting system, a 128-bit level of security seems to be a reasonable choice. While 112-bit level is probably acceptable for the next decade, many certification bodies will ask for 128 bits or more. In the case of an elliptic ElGamal this translates readily into a curve over a base field of 256 bits. Furthermore, base fields that are prime finite fields are usually preferred.

For the Paillier scheme, the security relies on a supposedly hard problem that it not harder than integer factorization of an RSA number n. The complexity of the best known factoring algorithm, the Number Field Sieve, being hard to evaluate, there is no strict consensus about the size of n giving a 128-bit security level, but generally this goes around 3072 bits. We provide below estimates based on a medium level of optimization, for a native implementation on a modern processor (based on OpenSSL and using RSA for Paillier emulation), and for a Javascript implementation running in a modern web browser (based on `libsodium.js` and JS BigInt).

	Paillier	Elliptic ElGamal	Ratio
Native (server-side)	200	10,000	50
In browser (voter-side)	2	5,000	2,500

3 SINGLE-CHOICE VOTING

Context. Voters give their choice among a list of k possibilities. The choices that get the more votes get the seats. Sometimes voters can select more than one choice, specially when the number s of seats is large. The basic situation is when choices are precisely the candidates. Another frequent situation is when the voter's choices are lists of candidates. Then one needs a rule to decide how to assign the seats according to the number of votes obtained by each list. For this later case, we will study the D'Hondt method since it is widely used in practice for politically binding elections.

Basic counting. The s winners are the first s candidates who obtained the most votes. This is the situation covered by Ordinos [28], but in case of equality between several values, more than s candidates can be elected by Ordinos. Assume for example that there are 10 seats but that the 10th and 11th candidates have received exactly the same number of votes. The toolkit of Ordinos will either outputs all the 11 candidates, with no information on who are the two last ones, or outputs the ordered list of the candidates, which leaks more information than needed.

We propose alternative algorithms, both in the ElGamal and in the Paillier settings, where we properly handle equality of candidates, according to the election rules. We also extend this approch to list voting, where seats are distributed among list of candidates.

List-voting. The method of D'Hondt parametrized by a sequence of distinct weights $w_1, \ldots, w_s$ proceeds as follows. Each voter votes for one list among k lists. At the end of the election, each list i has received c_i votes. Then the coefficients (c_i/w_j) for $1 \leq i \leq k$ and $1 \leq j \leq s$ are computed and the s largest values are selected so that the seats can be assigned accordingly: a list i gets one seat for each selected coefficient of the form (c_i/w_j). We define the function $f_{\mathtt{DHondt}}([c_1, \ldots, c_k], s)$ that returns $[s_1, \ldots, s_k]$ where s_i is the number of seats allocated to the list i. A ballot contains an encrypted bit for each choice (and a proof that the number of set bits follows the rules of the election). Then the ballots are aggregated to get the encrypted values of $c_1, \ldots, c_k$.

Various MPC algorithms. Comparing two encrypted integers can be done with various algorithms, depending on the ElGamal or Paillier setting, and whether the inputs are in the bit-encoding format or not. In Appendix, we present a list of cost vs communications trade-offs, that we select depending on our needs.

The question of how to handle fractions in the D'Hondt method must also be addressed. A textbook approach using pairs of integers to store the numerators and the denominators would require to compute products each time we want to make a comparison. One option to avoid this additional cost is to compute the $(s'_{i,j} = c_i w_j)$'s instead of $(s_{i,j} = c_i/w_j)$'s. Then, the boolean $s_{i_1,j_1} < s_{i_2,j_2}$ is exactly $s'_{i_1,j_2} < s'_{i_2,j_1}$. In a quadratic setting where all the comparisons are made, this is a nice solution. Otherwise, this would require to keep track of the indexes to be compared and would leak information. To design a sub-quadratic algorithm, we multiply all the $s_{i,j}$ by the least common multiple (lcm) of the weights w_j, to have only integers. In the case where the weights are just $1, 2, 3, \ldots, s$, this lcm grows like $\exp(s(1 + o(1)))$, so this adds $O(s)$ bits to the integers to manipulate. If s is of a size comparable to the logarithm of the number of voters, this is probably faster than to deal with the numerators and denominators separately.

Security. We denote by $P_{\mathtt{DHondt}}$ the resulting MPC protocol, in the ElGamal setting and we consider $T_{\mathtt{DHondt}}$ the trusted party that implements $f_{\mathtt{DHondt}}$ in the SUC framework, taking care of failure cases, as explained in Section 2.2.

THEOREM 3.1. *$P_{\mathtt{DHondt}}$ securely computes $T_{\mathtt{DHondt}}$ under the DDH assumption and the random oracle model (ROM).*

Summary. The choices of the algorithms to use depend on many practical questions and it is impossible to propose a universally best solution. A first element to consider is the choice between ElGamal and Paillier. If many voters are involved, then, with ElGamal, the aggregation of the ballots become very costly for the trustees both in computations and in communications, and Paillier might be the only realistic solution. Otherwise, ElGamal is very attractive for the reasons mentioned in Section 2.3 and the much easier key generation step (DKG).

In Figure 1 we propose two choices for basic counting, one with ElGamal and one with Paillier, in order to compare to Ordinos [28].

Version	Leak-age	EG/P	Voters # exp.	Authorities # exp.	Authorities # comm.	Transcript size
Basic counting [28]	[i]	P	$5k$	$4kn + (27m + 146 \log m)k^2a$	$(4R + 26B) \log m$	$6kn + (28m + 50 \log m)k^2a$
ours (exact winners)	$\emptyset$	P	$5k$	$4kn + \left(\frac{27}{2}m_2 + 73 \log m_1\right) k^2a$	$(4R + 26B) \log m_1$	$6kn + (14m_1 + 25 \log m_1)k^2a$
ours (exact winners)	$\emptyset$	EG	$8k$	$87(nk + m_1(2k - s)s)a$	$(m^2 + 2sm_1 \log m_1)R$	$93(nk + m_1(2k - s)s)a$
D'Hondt (comm.)	$\emptyset$	P	$5k$	$4kn + (27m_2 + 146 \log m_2)(ks)^2a$	$(4R + 26B) \log m_2$	$6kn + (28m_2 + 50 \log m_2)a(ks)^2$
D'Hondt (comm.)	$\emptyset$	EG	$8k$	$29ka(3n + 2m_2ks^2)$	$(m(m + \log s) + \log(ks)^2)R$	$93nak + 62m_2(ks)^2a$
D'Hondt (comp.)	$\emptyset$	EG	$8k$	$29ka(3n + mss' + 6m_3s^2)$	$(m^2 + ms' + 2s \log(ks) \log m_3)R$	$31ka(3n + mss' + 6m_3s^2)$

[i] More than s winners can be output in case of tie.

Figure 1: Leading terms of the cost of MPC implementations of various single choice systems. s: number of seats, k: number of lists, a: number of authorities, n: number of voters, $m = \lceil \log(n + 1) \rceil$, $m_1 = m + \log k$, $m_2 = m + \log(sk)$, $m_3 = m_1 + \log(\text{lcm}(2, \cdots, s))$, $s' = \log(\text{lcm}(2, \cdots, s))$, R: round of communications, B: broadcasts.

The cost of the DKG is not included, even though it is not negligible in the case of Paillier. In both cases, we assume that the number of candidates is small enough, so that the quadratic algorithm for selecting the s best is appropriate. For these figures as well as all the following ones, we only count the *leading terms of the cost*. For example, we neglect ak^2 if there is a term of the form a^2k^2. The unit of the *transcript size* is the key length, typically 3072 bits in Paillier and 256 bits in ElGamal.

Next, we propose three options for computing a D'Hondt tally with weights $1, 2, \ldots, s$. The first option is with Paillier, and with the objective of reducing the amount of communications. Therefore, we use the quadratic selection, and a communication-efficient comparison function. The second option is a simple adaptation to the ElGamal setting, which use a communication-efficient integer-comparison primitive. Finally, the third option also uses ElGamal, but uses a less naive algorithm to compute the winners. For comparing the fractions, we use the idea of crossing the indexes for the first two options, while in the third we multiply by the lcm.

4 MAJORITY JUDGMENT

In the Majority Judgment (MJ) approach [7], voters give a grade to each candidate, such as Excellent, Very Good, Poor, *etc.* Each grade is translated into a numerical value, typically from 1 to 6, where 1 is the highest grade. At the end of the election, each candidate c has received a list L_c of grades. The *list of medians* $\text{med}(c)$ associated to candidate c is the sequence formed by first the median grade m of L_c, *i.e.* the highest grade m such that at least half of the grades are greater or equal to m, then the median of $L_c \backslash \{m\}$ and so on. For example, if Alice received $1, 2, 2, 4, 4, 5$, her list of medians is $2, 4, 2, 4, 1, 5$. Then candidate c_1 is ranked above candidate c_2 if $\text{med}(c_1) < \text{med}(c_2)$ in the lexicographical order. Intuitively, c_1 wins over c_2 if she has a lower median, or, in case of a draw, a lower second median, *etc.* This defines a strict order, and therefore a winner: two candidates are ranked equal only if they received exactly the same grades.

A simplified algorithm. While the algorithm to determine the MJ winner(s) is simple, its naive implementation yields a complexity that depends on the number of voters, which could be very costly when done in MPC. Hence, the authors of [15] propose an MPC implementation of a simplification of the MJ algorithm, where whenever two candidates have the same median, only their number of grades higher and smaller than the median are compared. It has been shown that this technique is sound [7]: if a winner can be determined with this approach, it is indeed a MJ winner. However, it may also fail to conclude. In case the number of candidates is small and if the distribution of votes is uniform, then the probability of failure raises up to 22%, as shown in the table below. In any case, the approach of [15] leaks more information about the ballots than just the result, with non negligible probability, since it reveals whether the result can be determined with the simplified algorithm.

Number of voters	10	100	1000
uniform distribution over 5 candidates	0.384	0.220	0.080
political distribution [7]	N/A	0.001	N/A

Failure probabilities in [15].

MPC with Paillier. Our first contribution is an algorithm that computes MJ winner(s) on the clear votes, with a complexity that does not depend on the number of voters. Another algorithm was also proposed in [7] but our algorithm is easier to adapt in MPC and we prove it to correctly implement the MJ definition.

We assume that each voter produces a ballot formed of a matrix of encrypted 0 and 1, that encodes her choice, together with a zero-knowledge proof that each line contains exactly one 1. Thanks to the homomorphic property of Paillier encryption, the (encrypted) aggregated matrix, that is the sum of all the votes, can easily be obtained from the encrypted ballots. Then our algorithm essentially consists of comparisons, selections, additions or subtractions, and has been written in order to ease the conversion to an MPC algorithm, using the building blocks described in Section 2. Interestingly, the cost is similar to the (leaky) MPC implementation of [15], except for the number of communications that increases (see Figure 2).

MPC with ElGamal. The encoding of ballots remains unchanged for voters: each voter produces the matrix of her encrypted choices. Hence the cost is even lower for the voter since ElGamal encryption is cheaper. Then we compute the bit-encoding of the aggregated matrix using $\texttt{Add}^{\text{bits}}$. This part is linear in the number of voters but could be done on-the-fly during the election. Then the same algorithm can be used, on the bit-encoding, yielding a similar complexity than the Paillier's version, with the advantages of ElGamal as discussed in Section 2.3. Hence not only it remains practical to

Version	Leak-age	Voters # exp.	Authorities # exp.	Authorities # comm.	Transcript size
[15]	[i]	$5kd$	$4nkd + kma(224k + 58d)$	$(4m + d)R$	$6nkd + kma(280k + 62d)$
ours (P)	$\varnothing$	$5kd$	$4nkd + kda(75m + 146 \log m + 20d)$	$d(2R + 13B) \log m \log k$	$6nkd + kda(78m + 50 \log m + 22d)$
ours (EG)	$\varnothing$	$8kd$	$87nkda + 58kmda(10 + d)$	$m^2 + d(6m + 2 \log k \log m)$	$31kda(3n + (20 + 2d)m)$

[i] [15] leaks whether the winner can be determined with the simplified algorithm.

Figure 2: Leading terms of the cost of MPC implementations of Majority Judgment. n**: number of voters,** $m = \lceil \log(n+1) \rceil$**,** k**: number of candidates,** d**: number of grades,** a**: number of authorities.**

implement the full MJ function in MPC but surprisingly, the simple ElGamal encryption is well suited in this case.

Security. We denote by $P_{\mathtt{MJ}}$ the resulting MPC protocol, in the ElGamal setting and $f_{\mathtt{MJ}}$ the function that returns the MJ winners. $T_{\mathtt{MJ}}$ is the trusted party that implements $f_{\mathtt{MJ}}$ in the SUC framework.

Theorem 4.1. *$P_{\mathtt{MJ}}$ securely computes $T_{\mathtt{MJ}}$ under the DDH assumption and the random oracle model (ROM).*

5 CONDORCET-SCHULZE

The Condorcet approach is one popular technique to determine a winner when voters rank candidates by order of preference, possibly with equalities. A Condorcet winner is a candidate that is preferred to every other candidate by a majority of voters. More formally, we consider the *matrix of pairwise preferences d* where $d_{i,j}$ is the number of voters that prefer (strictly) candidate i over j. Then a Condorcet winner is a candidate i such that $d_{i,j} > d_{j,i}$ for all $j \neq i$. Such a Condorcet winner may not exist. In that case, several variants can be applied to compute the winner. We focus here on the Schulze method, used for example for Ubuntu elections [2]. It first considers by "how much" a candidate is preferred, which can be reflected into the *adjacency matrix a* defined as

$$a_{i,j} = \begin{cases} d_{i,j} - d_{j,i} & \text{if } d_{i,j} > d_{j,i}, \\ 0 & \text{otherwise.} \end{cases}$$

Then a weighted directed graph is derived from the adjacency matrix, where each candidate i is associated to a node and there is an edge from i to j with weight $a_{i,j}$. This itself induces an order relation between the candidates by comparing the "strength" of the paths between i and j. The exact algorithm can be found in [42]. Note that there may be several winners according to Condorcet-Schulze. We denote by f_{Cond} the function that returns the winners.

We propose several MPC implementations of Condorcet-Schulze, depending on the accepted leakage and on the load balance between the voters and the authorities. The different approaches are summarized in Figure 3.

Ballots as matrices. For each candidate i, let c_i be an integer that represents the order of preference, possibly with equality. A first approach is to encode the vote as a *preference matrix m* where

$$m_{i,j} = \begin{cases} 1 & \text{if } c_i < c_j \\ 0 & \text{if } c_i = c_j \\ -1 & \text{otherwise} \end{cases}$$

The voters then simply encode their ballot as an encrypted preference matrix M. Note that this requires k^2 encryptions (one encryption for each coefficient of the matrix). Voters also need to prove that their (encrypted) matrix is well-formed, that is, corresponds to a total order (with equalities). This requires *e.g.* to prove that if the voter prefers i over j and j over k then she prefers i over k:

$$(m_{i,j} = 1) \wedge (m_{j,k} = 1) \Rightarrow (m_{i,k} = 1)$$

and similar relations when $m_{i,j}$ and $m_{j,k}$ are equal to 0 or -1, yielding $O(k^3)$ statements.

Previous work. To discharge the voter from such a proof effort, in [25] the authorities shuffle each preference matrix in blocks (using $\mathtt{ShuffleMatrix}([M_{i,j}])$) and then decrypt it to check that it was indeed well formed. However, this yields a privacy breach, unnoticed by the authors, as explained in introduction: for each voter, everyone learns the number of candidates placed at equality. In particular, everyone learns who voted blank since in that case all candidates are placed at equality. A costly way to repair [25] is to let the voters prove the relations with zero-knowledge proofs, yielding a cost of $O(k^3)$ exponentiations to build and to check a ballot. This is roughly the approach of [27], that also assumes that voters do not place candidates at equality (the case $c_i = c_j$ is forbidden).

Our approach. We propose an alternative approach in $O(k^2)$. Assume first that a voter prefers candidate 1 over candidate 2, that is preferred over candidate 3 and so on. Then the corresponding preference matrix is:

$$m^{\mathsf{init}} = \begin{pmatrix} 0 & 1 & \cdots & 1 \\ -1 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ -1 & \cdots & -1 & 0 \end{pmatrix}.$$

We consider a fixed encryption M^{init} of this matrix:

$$M^{\mathsf{init}}_{i,j} = \begin{cases} E_1 & \text{if } i < j \\ E_0 & \text{if } i = j \\ E_{-1} & \text{otherwise} \end{cases}$$

where E_α is the ElGamal encryption of α with "randomness" 0. Everyone can check that M^{init} is formed as prescribed, at no cost, since we use a constant "randomness".

Assume now that a voter wishes to rank the candidates in some order, which is a permutation σ of $1, 2, \ldots, k$. Then our core idea is that the voter can simply shuffle M^{init} (using $\mathtt{ShuffleMatrix}$) using permutation σ. The associated zero-knowledge proof guarantees that the resulting matrix is indeed a permutation of M^{init}, hence is well formed. Interestingly the secret vote σ is not encoded in the initial matrix but in the permutation used to shuffle it. Applying [47], this requires $O(k^2)$ exponentiations for the voter. To account for candidates that have an equal rank, the voter still shuffles M^{init}

Version	Leakage	EG/P	Voters # exp.	Authorities # exp.	Authorities # comm.	Size of the transcript
[25]	adj. matrix **privacy breach** [i]	EG	$10k^2$	$18ank^2$	2	$10ank^2$
[27] [ii][iii]	∅	P	$5k^3$	$6nak^3 + (54m + 292\log m)ak^3$	$4k\log m$	$9nk^3 + (56m + 100\log m)ak^3$
ballots as list of integers (partial MPC)	adj. matrix	EG	$8k\log k$	$\frac{87}{2}nak^2\log k$	$2\log k$	$\frac{93}{2}nak^2\log k$
ballots as list of integers (full MPC)	∅	EG	$8k\log k$	$\frac{29}{2}nak^2(3\log k + 5m)$ $+174mak^3$	$m(m+4k)$	$\frac{31}{2}nak^2(3\log k + 5m)$ $+186mak^3$
ballots as matrices	adj. matrix	EG	$\frac{43}{2}k^2$	$\frac{47}{2}nk^2$	0	$\frac{85}{2}nk^2$

[i] [25] leaks, for each ballot, the number of candidates ranked at equality. In particular, who voted blank is known to everyone.
[ii] [27] does not allow voters to give the same rank to several candidates.
[iii] [27] originally does not take into account the cost of verifying the ZKP provided by the voters.

Figure 3: Leading terms of the cost of MPC implementations for Condorcet-Schulze. n: number of voters, $m = \lceil\log(n+1)\rceil$, k: number of candidates, a: number of authorities.

according to a permutation σ, consistent with her preference order, that is such that $\sigma(i) < \sigma(j)$ implies that $c_i \leq c_j$. But beforehand, she sends an additional vector B of encrypted bits (b_i), where $b_i = 1$ if candidates $\sigma^{-1}(i)$ and $\sigma^{-1}(i+1)$ have equal rank and $b_i = 0$ otherwise. The voter will then modify the matrix M^{init} into a transformed matrix M', using B, so that M' corresponds to her preference matrix. The resulting cost is still in $O(k^2)$ (since k^2 coefficients need to be updated) instead of $O(k^3)$ for [27] (that, yet, does not consider equalities).

Then the (encrypted) adjacency matrix can be computed by simply multiplying all ballots. This matrix is then (provably) decrypted by the authorities and Condorcet-Schulze as well as many variants can be applied. The main cost for the authorities lies in the verification of the proofs for each ballot. We could also avoid leaking the adjacency matrix by computing the Condorcet-Schulze winner(s) in MPC. However, the cost for the authorities would be in $O(k^3)$. If this is considered as affordable, then we can further alleviate the charge of the voters, as we shall explain now.

Ballots as list of integers. To minimize computations on the voter's side, we can ask them to simply encrypt the list of integers (c_i) representing their preference. To allow for ElGamal encryption, we will directly use the bit representation of each integer and encrypt each bit separately. If there are k candidates, we need $\log k$ bits to encode each candidate, hence a ballot will contain $k\log k$ ciphertexts, together with zero-knowledge proofs that the ciphertexts encrypt only 0 or 1. This is to be compared with the k^2 encryptions when ballots are encoded as a preference matrix.

Our first goal is to transform back each ballot into a preference matrix. We consider the *positive preference matrix*, obtained from the preference matrix by setting negative coefficients to 0. If C_i denotes the bitwise encryption of c_i then the encrypted positive preference matrix M can be computed by the authorities as:

$$M_{i,j} = \mathsf{LT}^{\text{bits}}(C_i, C_j).$$

Summing up the (encrypted) matrix M_v for each voter v, we immediately obtain the (encrypted) pairwise preferences matrix D. This matrix can be decrypted, or the authorities may apply the Schulze method in MPC from D. Despite the fact that the Schulze method is a complex algorithm on graphs, it can be implemented with an algorithm from Floyd-Warshall [23, 43], that mostly consists in computations of min/max. This can be translated into an MPC algorithm using the building blocks presented in Section 2. We denote by P_{Cond} the corresponding MPC protocol.

The advantage of this solution is that the load for voters remains very reasonable, with $O(k\log k)$ exponentiations in total. However, transforming each ballot into the (encrypted) preference matrix M_v is of cost $O(k^2\log k)$ per voter, for each authority.

Security. We denote by T_{Cond} the trusted party that implements f_{Cond} in the SUC framework.

Theorem 5.1. *P_{Cond} securely computes T_{Cond} under the DDH assumption and the random oracle model (ROM).*

We have actually presented two other variants, depending on the ballot encoding. They both leak the adjacency matrix and hence do not securely compute T_{Cond}. Instead, we can show that they securely compute the trusted party T_{adj} that returns the adjacency matrix corresponding to the submitted ballots.

To summarize, when the number of candidates and voters remain reasonable, it is actually possible to compute the Condorcet winners with no leakage. Interestingly, the costly operations performed by the trustees can be done on-the-fly, while voters submit their ballots. Note that unless the number of candidates is really large w.r.t. the number of voters, a fully-hiding tally scheme is not really more expensive than schemes leaking the adjacency matrix.

6 SINGLE TRANSFERABLE VOTE

Choosing one version of STV. Many flavors of STV election methods exist. In all of them, a ballot cast by a voter contains an ordered list of candidates, starting with the most preferred one. Along the counting process, if the candidate in the first line has been selected to get a seat or eliminated, then it should be erased from the ballot, so that the candidate on the second line becomes the most preferred at this stage. However, when a candidate gets a seat, this must "consume" some of the ballots who voted for him. From this comes the notion of quota and the transfer mechanism. In our case, we used the so-called Droop quota, which sets the

Version	Leakage	P/EG	Voters # exp.	Authorities # exp.	Authorities # comm.	Transcript size
[10, Sec. II]	[i]	EG	$10k^2$	$62nak^2$	$9kR$	$19nak^2$
[44]	[ii]	P	$5k^2$	$22nk^2am$	$2nkmR$	$11nak^2m$
[10, Sec. III.B]	[iii]	EG	$10k^2$	$62nak^2$	$9kR$	N/A
ours (naive arith.)	$\varnothing$	EG	$9k\log k$	$29nak^2(4\log k +3m'(r+1))$	$k(2m'(m+2r+\log k) +k\log\log k))R$	$31nak^2(4\log k +3m'(r+1))$
ours (optimized arith.)	$\varnothing$	EG	$9k\log k$	$\frac{29}{2}nak^2(9\log k +3m'(r+1)\log m')$	$k(2m'(\log k+2\log m') +(\log k)^2)R$	$\frac{31}{2}nak(9k\log k +3m'(r+1)\log m')$

i Score of all candidates at each turn

ii Score of selected candidates at each turn

iii Selected or eliminated candidates and approximation of transfer coefficient at each turn. Trustees learn the score of all candidates at each turn

Figure 4: Leading terms of the cost of MPC implementations of STV. n**: number of voters,** k**: number of candidates,** $m = \lceil\log(n+1)\rceil$**,** a**: number of authorities,** r**: precision in power of** 2**,** $m' = m + r$**,** $k' = k + r$**.**

value of a seat at $q = \lceil n/(s+1)\rceil + 1$. Here s is the number of seats, and n is the number of valid ballots. If a candidate is in the first line of a (weighted) number of ballots that is larger than q, then she gets elected. Otherwise, we take the candidate that gets the least votes and we eliminate her. In case of equality, we use a predefined arbitrary ordering (as in Section 3). The transfer is implemented as follows: each ballot starts with a weight set to one. When a candidate is elected, the surplus of votes is transferred to the next candidates. Namely, all the ballots where this candidate was listed first have their weight multiplied by a transfer coefficient $t = (c - q)/c$ where c is the sum of the weights of such ballots.

Fractions vs approximations. All along the STV algorithm, the weights of the ballots and the transfer coefficient are rational numbers that can be stored as pairs of integers. While this looks as the cleanest approach, we noticed that this leads to an exponential worse-case complexity. Indeed, the transfer coefficient t_i at a round i is a fraction whose height typically doubles at each round where a candidate is selected, and we get a complexity that is exponential in the number of seats.

This observation is a major problem in an MPC setting where the worst complexity must always be done, in order to hide every side-information. However, we realized that this is also a problem outside any cryptographic consideration. For instance, we ran the ideal STV algorithm on the publicly available ballots of the 2019 Legislative Council of New South Wales in Australia [3]. There are 21 seats, 346 candidates and 3.5 millions of ballots were cast. Our basic implementation using Sagemath shows that indeed, the size of the fractions roughly doubles at each selection, so that one would require about 30 GB of central memory for storing all of them. In real elections, and due to the fact that elections were initially counted by hand, approximations of fractions are used instead. We therefore represent fractions with a fix-point arithmetic, allowing r binary digits after the radix point. We denote by f_{STV} the corresponding function that returns the STV winners.

To leak or not to leak. The two main approaches toward a tally-hiding STV algorithm in the literature are [44] and [10]. In [44], mixnets are applied between each round of the algorithm, so that some information can be decrypted and revealed, without disclosing the list of the complete original ballots. The information that is leaked is whether the round was a selection or an elimination, and in the latter case, the score of the selected candidates. We remark also that their technique involves a very sequential first phase with a number of communications that is proportional to the number of ballots. In [10], some information is also revealed between each round of the STV algorithm, in particular the score of all the candidates, which is much more than in [44]. It is however highly efficient. The authors acknowledge that revealing the intermediates scores might be too much; in particular, they propose realistic scenarios where a coercer could successfully use this information. In [10], a variant is proposed where the most crucial information is leaked only to the trustees. For external observers, their approach leaks essentially the same information as [44], and also an approximation of the transfer coefficient at each round.

Thanks to our toolbox, we can simply follow the standard STV algorithm (with rounding) and derive a leakage-free tally scheme.

Arithmetic optimizations. To improve the complexity of the resulting scheme, we had to carefully implement bit-wise addition and multiplication. Let r be the number of binary digits after the radix point, so that all our computations are done with a fix-point precision of 2^{-r}. A bound on the real numbers manipulated during the algorithm is given by the number of voters n, so that we need $m = \lceil\log(n+1)\rceil$ bits for the mantissa. Hence, the operations reduce to integer arithmetic with $m' = m + r$ bits. While this looks small (a few dozens of bits), using textbook algorithms with a naive carry-propagation would lead to a number of rounds of communications that grow linearly with m' for additions and quadratically with m' for multiplications.

For carry-propagation during additions, this is a classical problem in hardware arithmetic circuits. The depth of the circuit translates more or less immediately into the number of communications in our MPC setting. An important difference with hardware considerations is that bounding the fan-in / fan-out of the gates is not relevant for us. The general idea is to rewrite the addition (or subtraction, or comparison) with the help of an associative operator acting on bits, so that a tree of height $\log(m')$ can be constructed.

The Appendix contains the details of how, following this strategy, we managed to strongly reduce the communication rounds at cost of a moderate increase in terms of exponentiations. This also yields big savings for multiplications and divisions, since they are built upon additions.

Security. We denote by P_{STV} the MPC protocol with arithmetic optimizations and T_{STV} the trusted party that implements f_{STV} in the SUC framework.

THEOREM 6.1. *P_{STV} securely computes T_{STV} under the DDH assumption and the random oracle model (ROM).*

Efficiency considerations. In Figure 4, we give a summary of the various costs for our algorithm and the ones from the literature. Comparing the two last lines demonstrates the advantages of optimizing the arithmetic, since the last one is a very good compromise. While it is difficult to draw conclusions without knowing the context, we consider that with our algorithm, requiring a perfectly tally-hiding is not the criterion that will make the solution turns from practical to impractical. In fact, from the voter's side, our scheme is more efficient than existing solutions, with a quasi-linear number of exponentiations instead of quadratic. The costs for the authorities is certainly terribly high and is not yet realistic for a large scale election, but we consider that this is not much more than the previous solutions which leak partial information.

7 APPLICATION TO E-VOTING SECURITY

We show that our tally-hiding schemes can be used for e-voting, preserving vote secrecy and verifiability. We consider a mini-voting scheme, TH-voting, where we assume that voters have an authenticated channel with the voting server. Similarly to Ordinos [28], voters simply encrypt their vote following the expected format and the MPC protocol is used for tallying.

Definitions. A *voting scheme* consists of four algorithms and one MPC protocol $(\mathtt{Setup}, \mathtt{vote}, \mathtt{isValid}, P_{\mathtt{tally}}, \mathtt{Verify})$ where:

- $\mathtt{Setup}(\kappa, a, t)$ takes as input the security parameter κ, the number of authorities a and a threshold t. It returns $sk, pk, s_1, h_1, \cdots, s_a, h_a$ respectively a key pair sk, pk and the corresponding private and public shares s_i, h_i for the authorities.
- $\mathtt{vote}(pk, v)$ takes a public key pk, a vote v, and returns a ballot.
- $\mathtt{isValid}(BB, B)$ takes as input a ballot B and a ballot box BB and returns a boolean that states whether B is valid w.r.t. BB.
- $P_{\mathtt{tally}}(a, t) = P_1, \cdots, P_a$ is an MPC protocol run by authorities to compute the tally.
- $\mathtt{Verify}(r, \Pi, BB)$ takes as input a result r, a transcript Π and a ballot box BB and returns a boolean that states whether r is correct w.r.t. BB and Π. This check is typically run by external auditors.

In [29], a quantitative definition of privacy is proposed, where a voting system is said δ-private for some δ. This definition can be turned into a qualitative one when δ is shown to be minimal, in a sense that an ideal protocol achieves δ'-privacy with a negligible $|\delta - \delta'|$. Hence, a natural definition of privacy is to compare the probability of success of the adversary in a real and in an ideal protocol, and to show that the difference is negligible. Just as in [29], we consider a definition where the adversary tries to guess the vote of a single voter. We consider a fixed set V of valid *voting options* and the games defined respectively in Algorithms 2 and 3, where the differences are highlighted in blue.

Definition 7.1 (vote privacy). We say that a voting protocol $(\mathtt{Setup}, \mathtt{vote}, \mathtt{isValid}, P_{\mathtt{tally}}, \mathtt{Verify})$ guarantees *vote privacy* w.r.t a result function $\mathtt{tally}$ if, for all parameters t, a, n, n_c with $t < a$ and $n_c \leq n$, for all $C \subset [1, a]$ of size at most t, for all adversary $\mathbb{A}$, there exists an adversary $\mathbb{B}$ and a negligible function μ such that for all voting options $v_2, \cdots, v_n \in V$,

$$|Pr(\mathtt{Real}^{\mathtt{Priv}}_{\mathbb{A}, P_{\mathtt{tally}}}(\kappa, n, n_c, a, t, C, V, v_2, \cdots, v_n) = 1)$$
$$- \Pr(\mathtt{Ideal}^{\mathtt{Priv}}_{\mathbb{B}, \mathtt{tally}}(\kappa, n, n_c, a, t, C, V, v_2, \cdots, v_n) = 1)| \leq \mu(\kappa).$$

Algorithm 2: $\mathrm{Real}^{\mathtt{Priv}}_{\mathbb{A}, P_{tally}}(\kappa, n, n_c, a, t, C, V, v_2, \cdots, v_n)$

$sk, pk, s_1, h_1, \cdots, s_a, h_a := \mathtt{Setup}(\kappa, a, t)$
$b \in_r \{0, 1\}$; $par = pk, h_1, \cdots, h_a$
$v_0, v_1 := \mathbb{A}(\kappa, par, (s_i)_{i \in C})$
$BB := \{\mathtt{vote}(pk, v_b)\}$
for $i = 2$ to $n - n_c$ **do** $BB := BB \bigcup \{\mathtt{vote}(pk, v_i)\}$
$(X_i)_{i > n - n_c} := \mathbb{A}(BB)$
for $i > n - n_c$ **do**
  if $\mathtt{isValid}(BB, X_i)$ **then** $BB := BB \bigcup \{X_i\}$
$r := \mathbb{A} ||_{i \in [1,a] \setminus C} P_i(s_i, par, BB)$
$b' := \mathbb{A}()$
Return $(b == b') \wedge (v_0, v_1 \in V)$

Algorithm 3: $\mathrm{Ideal}^{\mathtt{Priv}}_{\mathbb{B}, \mathtt{tally}}(\kappa, n, n_c, a, t, C, V, v_2, \cdots, v_n)$

$sk, pk, s_1, h_1, \cdots, s_a, h_a := \mathtt{Setup}(\kappa, a, t)$
$b \in_r \{0, 1\}$; $par = pk, h_1, \cdots, h_a$
$v_0, v_1 := \mathbb{B}(\kappa, par, (s_i)_{i \in C})$
$BB := \{\mathtt{vote}(pk, v_b)\}$
for $i = 2$ to $n - n_c$ **do** $BB := BB \bigcup \{\mathtt{vote}(pk, v_i)\}$
$(X_i)_{i > n - n_c} := \mathbb{B}()$
for $i > n - n_c$ **do**
  if $\mathtt{isValid}(BB, X_i)$ **then** $BB := BB \bigcup \{X_i\}$
$r := \mathtt{tally}((\mathtt{Extract}_{sk}(B))_{B \in BB})$
$b' := \mathbb{B}(r)$
Return $(b == b') \wedge (v_0, v_1 \in V)$

TH-voting. We define a voting protocol $V_{\mathtt{tally}}$ for each tally function $\mathtt{tally}$ covered in our paper (D'Hondt, Majority Judgment, Condorcet-Schulze, and STV), with $P_{\mathtt{tally}}$ the corresponding tally-hiding protocol, in the ElGamal setting. The algorithm $\mathtt{vote}_{\mathtt{tally}}$ returns an encrypted ballot following the encoding devised in the corresponding section, and a ZKP that the ballot is correctly formed. The algorithm $\mathtt{isValid}_{\mathtt{tally}}$ checks the ZKP and additionally ensures that the ballot is not already on the board. As explained in Section 2, the $\mathtt{CGate}$ protocol produces a transcript which acts as a ZKP that the protocol was performed correctly. By concatenating the transcripts of all $\mathtt{CGate}$ and the transcript of the threshold decryption, the participants produce a ZKP Π that $P_{\mathtt{tally}}$ has been performed correctly. This also defines a $\mathtt{Verify}_{\mathtt{tally}}$ algorithm

voters	5 candidates	10 candidates	20 candidates
64	1m50s / 49 MB	8m30s / 0.30 GB	45m / 1.8 GB
128	2m40s / 87 MB	12m / 0.51 GB	1h27m / 2.9 GB
256	4m35s / 160 MB	20m / 0.88 GB	2h37m / 4.8 GB
512	8m10s / 305 MB	34m / 1.6 GB	4h43m / 8.6 GB
1024	15m / 595 MB	1h05m / 3.1 GB	8h50m / 16 GB

Figure 5: Benchmark (wall-clock time and transcript size) of fully tally-hiding Condorcet-Schulze MPC computation.

which simply consists of verifying all the ZKP. Finally, we consider an ideal $\texttt{Setup}(\kappa, a, t)$ that picks a group G corresponding to the security parameter κ, picks randomly a generator g and returns $sk, pk, s_1, h_1, \cdots, s_a, h_a$ where the (s_i, h_i) are distributed following Shamir's scheme with a authorities and a threshold t; sk is the corresponding secret key and $pk = (g, g^{sk})$. The setup can be further refined with a UC-secure DKG (see e.g. [45]).

Theorem 7.2. *Let* `tally` *be one of the previously defined tally functions (D'Hondt, Majority Judgment, Condorcet-Schulze, and STV). Assuming DDH,* $V_{\texttt{tally}}$ *is private w.r.t.* `tally`.

The proof can be found in Appendix J.3. We also prove that $V_{\texttt{tally}}$ is verifiable for a notion of verifiability similar to [20]. Note that the key step is the fact that our tally-hiding schemes guarantees universal verifiability: auditors can check the result is valid. Individual verifiability is straightforward in our setting since we implicitly assume that all voters verify their vote. How to achieve individual verifiability in practice is beyond the scope of this work.

8 IMPLEMENTATION

In order to validate our approach, we have written a prototype implementation. In the literature, most of such prototypes are based on Paillier encryption. Here, we concentrate on the ElGamal-based setting, in order to evaluate its practical feasibility. The `libsodium` library is used for randomness and all elliptic curve and hashing operations. The rest is implemented as a standalone C++ program. It is available as a companion artefact of this paper [4] and is published as free software. Most of the primitives of our toolbox have been implemented, and as a proof of concept, we have written a fully tally-hiding protocol for Condorcet-Schulze (ballots as list of integers, and no leakage, in Figure 3).

We ran our software on various sets of parameters. In order to compare to [27], we also consider 3 trustees (and no threshold). Our experimental setting is a single server hosting two 16-core AMD EPYC 7282 processors and 128 GB or RAM. Each of the 3 trustees runs 4 computing threads and a few scheduling and I/O threads. The communication between the trustees is emulated via the loopback network interface. Thus, all the network system calls are indeed performed by the program, even though this is just a simulation. The verification of the validity of the ballots is a non-MPC computation that takes a negligible time, compared to the tally. In Figure 5, we summarize the cost in terms of wall-clock time and the size of the transcript, measured by the program.

This experiment demonstrates that the approach is sound and in the realm of practicability, for moderate-sized elections. With this choice of ballot representation, which is very cheap from the voter's point of view, the agglomeration of the preference matrices has to be done in MPC, and therefore the cost for the trustees grows quasi-linearly in the number of voters. Therefore, at some point, the approach of [27] using Paillier encryption becomes preferable, since the aggregation is for free, and the MPC cost is essentially independent of the number of voters. Still, their benchmark gives more than 9 days of MPC computation for tallying a 20-candidates Condorcet-Schulze election, which is more than what we provide for 1024 voters. This is mostly due to the efficiency of elliptic-curve based ElGamal encryption.

9 LESSONS LEARNED

Our study shows that it is possible to compute the result of an election without leaking any additional information on the original ballots, often at a realistic cost. This requires however to carefully design the corresponding algorithm for each different tally function. We have provided in this paper several techniques that can reduce the cost. This was applied to several well-known complex voting systems, and we developed a toolbox that can be re-used in other contexts. We list here the main questions that a designer should consider when implementing another counting function.

Think ElGamal. While Paillier is the Swiss-Army knife for MPC implementations, our study has shown that ElGamal can often suffice, even when encrypted integers need to be compared or multiplied. This can be a big advantage in terms of efficiency and availability of software libraries.

Rethink the encoding of ballots. The encoding of a ballot can have a huge impact on the cost of the rest of the procedure. For example, encoding integers in their bit representation adds an initial cost that can later save a lot of computation. It can allow to use ElGamal rather than Paillier. The encoding of ballots also typically offers different tradeoffs in terms of load balance between voters and authorities, as seen for example for the Condorcet voting function where a more complex ballot can alleviate the authorities task.

Verifiable mixnets are a versatile tool. The typical use of a mixnet is to mix and re-randomize encrypted ballots before decryption and application of a counting function on the cleartexts. However, verifiable mixnets are also useful to discharge some computations (*e.g.* verifications) on the cleartexts. More advanced mixing can be used to ensure for example that the same permutation is applied to several components. We have proposed an original usage of mixnet in the context of Condorcet, where each voter uses a verifiable shuffle to encode their vote as a (secret) permutation of a fixed public matrix, proving well-formedness.

Consider the full algorithmic toolbox. When designing an MPC algorithm, the constraints are rather non standard. The worst case always needs to be considered, and all branches need to be always visited, like in the circuit complexity model. In fact, this circuit point of view is highly relevant, and we borrowed some algorithms from the hardware literature. The depth of the circuit is related to the number of communication rounds; but limits on the fan-in or fan-out of a gate are irrelevant.

Some rather advanced algorithms like the MJ counting functions or the Floyd-Warshall shortest path algorithm can be translated rather easily. On the other hand, some basic tasks can be way too costly if one chooses the wrong algorithm for them. For instance, sorting a list of integers becomes quadratic for more than a few quasi-linear classical algorithms when converted to MPC. Indeed, many classical algorithms assume that accessing the i^{th} value of an array $T[i]$ takes constant time, even when i is a computed value, while in MPC this requires a linear time to pass through all the values and hide the value of i. Another typical example is addition of encrypted integers, where carry propagation can generate a chain of dependencies that translates into a linear number of communication rounds. Breaking the chain of carries as done in hardware circuits allows to reduce this to a logarithmic number of rounds.

REFERENCES

[1] 2003. Condorcet Internet Voting Service (CIVS). https://civs.cs.cornell.edu/. (2003). Accessed: 04/01/2022.

[2] 2012. Ubuntu IRC Council Position. https://lists.ubuntu.com/archives/ubuntu-irc/2012-May/001538.html. (2012). Accessed: 04/01/2022.

[3] 2019. NSWEC – Election Results. NSW Electoral Commision, https://pastvtr.elections.nsw.gov.au/SG1901/LC/State/preferences. (2019). Accessed: 2020-08-05.

[4] 2022. Source code of prototype implementation of Section 8. Available at https://gitlab.inria.fr/gaudry/THproto. (2022).

[5] Masayuki Abe and Serge Fehr. 2004. Adaptively Secure Feldman VSS and Applications to Universally-Composable Threshold Cryptography. In *Advances in Cryptology - CRYPTO 2004 (Lecture Notes in Computer Science)*, Matthew K. Franklin (Ed.), Vol. 3152. Springer, 317–334.

[6] Ben Adida. 2008. Helios: Web-based Open-Audit Voting. In *17th USENIX Security Symposium (Usenix'08)*.

[7] Michel Balinski and Rida Laraki. 2010. *Majority Judgment: Measuring Ranking and Electing.* MIT Press. https://hal.archives-ouvertes.fr/hal-01533476

[8] J. Bar-Ilan and D. Beaver. 1989. Non-Cryptographic Fault-Tolerant Computing in Constant Number of Rounds of Interaction. In *Annual ACM Symposium on Principles of Distributed Computing (PODC'89)*. 9. https://doi.org/10.1145/72981.72995

[9] Mihir Bellare and Amit Sahai. 1999. Non-malleable Encryption: Equivalence between Two Notions, and an Indistinguishability-Based Characterization. In *Advances in Cryptology - CRYPTO '99 (Lecture Notes in Computer Science)*, Michael J. Wiener (Ed.), Vol. 1666. Springer, 519–536.

[10] Josh Benaloh, Tal Moran, Lee Naish, Kim Ramchen, and Vanessa Teague. 2010. Shuffle-Sum: Coercion-Resistant Verifiable Tallying for STV Voting. *IEEE Transactions on Information Forensics and Security* (2010). https://doi.org/10.1109/TIFS.2009.2033757

[11] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. 2015. SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 499–516. https://doi.org/10.1109/SP.2015.37

[12] David Bernhard, Olivier Pereira, and Bogdan Warinschi. 2012. How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. In *Advances in Cryptology - ASIACRYPT 2012 (Lecture Notes in Computer Science)*, Vol. 7658. Springer, 626–643.

[13] Brent and Kung. 1982. A Regular Layout for Parallel Adders. *IEEE Trans. Comput.* C-31, 3 (1982).

[14] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. 2018. Bulletproofs: Short Proofs for Confidential Transactions and More. In *IEEE Symposium on Security and Privacy (S&P'18)*. https://doi.org/10.1109/SP.2018.00020

[15] Sébastien Canard, David Pointcheval, Quentin Santos, and Jacques Traoré. 2018. Practical Strategy-Resistant Privacy-Preserving Elections. In *European Symposium on Research in Computer Security (ESORICS'18)*. Springer.

[16] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001*. IEEE Computer Society, 136–145.

[17] Ran Canetti, Asaf Cohen, and Yehuda Lindell. 2015. A Simpler Variant of Universally Composable Security for Standard Multiparty Computation. In *Advances in Cryptology - CRYPTO 2015 (Lecture Notes in Computer Science)*, Vol. 9216. Springer, 3–22.

[18] M. R. Clarkson, S. Chong, and A. C. Myers. 2008. Civitas: Toward a Secure Voting System. In *IEEE Symposium on Security and Privacy (S&P'08)*.

[19] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachene. 2013. Distributed ElGamal à la Pedersen - Application to Helios. In *Workshop on Privacy in the Electronic Society (WPES'13)*.

[20] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachene. 2014. Election Verifiability for Helios under Weaker Trust Assumptions. In *Proceedings of the 19th European Symposium on Research in Computer Security (ESORICS'14) (LNCS)*, Vol. 8713. Springer, Wroclaw, Poland, 327–344.

[21] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. 1994. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *CRYPTO'94*. Springer.

[22] Ivan Damgård and Mads Jurik. 2001. A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In *Public Key Cryptography (PKC'01)*. Springer.

[23] Robert W. Floyd. 1962. Algorithm 97: Shortest Path. *Commun. ACM* 5, 6 (1962).

[24] Rolf Haenni, Reto E. Koenig, Philipp Locher, and Eric Dubuis. 2017. CHVote System Specification. Cryptology ePrint Archive, Report 2017/325. (2017).

[25] Thomas Haines, Dirk Pattinson, and Mukesh Tiwari. 2019. Verifiable Homomorphic Tallying for the Schulze Vote Counting Scheme. In *Verified Software. Theories, Tools, and Experiments (VSTTE'19)*. Springer.

[26] Carmit Hazay, Gert Mikkelsen, Tal Rabin, and Tomas Toft. 2019. Efficient RSA Key Generation and Threshold Paillier in the Two-Party Setting. *Journal of Cryptology* (2019).

[27] Fabian Hertel, Nicolas Huber, Jonas Kittelberger, Ralf Kuesters, Julian Liedtke, and Daniel Rausch. 2021. Extending the Tally-Hiding Ordinos System: Implementations for Borda, Hare-Niemeyer, Condorcet, and Instant-Runoff Voting. In *Proceedings E-Vote-ID 2021*. University of Tartu Press, 269–284.

[28] Ralf Kuesters, Julian Liedtke, Johannes Mueller, Daniel Rausch, and Andreas Vogt. 2020. Ordinos: A Verifiable Tally-Hiding E-Voting System. In *IEEE European Symposium on Security and Privacy (EuroS&P'20)*.

[29] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. 2011. Verifiability, Privacy, and Coercion-Resistance: New Insights from a Case Study. In *32nd IEEE Symposium on Security and Privacy, S&P 2011*. IEEE Computer Society, 538–553.

[30] Helger Lipmaa. 2003. On Diophantine Complexity and Statistical Zero-Knowledge Arguments. In *ASIACRYPT'03*. Springer.

[31] Helger Lipmaa and Tomas Toft. 2013. Secure Equality and Greater-Than Tests with Sublinear Online Complexity. In *Automata, Languages, and Programming (ICALP'13)*. Springer.

[32] B. L. Meek. 1969. Une nouvelle approche du scrutin transférable. *Mathématiques et Sciences humaines* 25 (1969). http://www.numdam.org/item/MSH_1969__25__13_0

[33] Jesper Buus Nielsen. 2003. *On Protocol Security in the Cryptographic Model.* Ph.D. Dissertation. University of Aarhus.

[34] Takashi Nishide and Kouichi Sakurai. 2010. Distributed Paillier Cryptosystem without Trusted Dealer. In *Information Security Applications (WISA 2010)*. Springer.

[35] Torben Pryds Pedersen. 1991. A Threshold Cryptosystem without a Trusted Party. In *EUROCRYPT'91*. Springer.

[36] Maurice Pollack. 1960. The Maximum Capacity through a Network. *Operations Research* 8, 5 (1960). http://www.jstor.org/stable/167387

[37] Guillaume Poupard and Jacques Stern. 1998. Security analysis of a practical "on the fly" authentication and signature generation. In *EUROCRYPT'98*. Springer.

[38] Kim Ramchen, Chris Culnane, Olivier Pereira, and Vanessa Teague. 2019. Universally Verifiable MPC and IRV Ballot Counting. In *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers (Lecture Notes in Computer Science)*, Ian Goldberg and Tyler Moore (Eds.), Vol. 11598. Springer, 301–319. https://doi.org/10.1007/978-3-030-32101-7_19

[39] Berry Schoenmakers and Pim Tuyls. 2004. Practical Two-Party Computation Based on the Conditional Gate. In *ASIACRYPT'04*. Springer.

[40] Berry Schoenmakers and Pim Tuyls. 2006. Efficient Binary Conversion for Paillier Encrypted Values. In *EUROCRYPT'06*. Springer.

[41] Berry Schoenmakers and Meilof Veeningen. 2015. Universally Verifiable Multiparty Computation from Threshold Homomorphic Cryptosystems. In *Applied Cryptography and Network Security (ACNS'15)*. Springer.

[42] Markus Schulze. 2011. A New Monotonic, Clone-independent, Reversal Symmetric, and Condorcet-consistent Single-winner Election Method. *Social Choice and Welfare* 36 (2011). https://doi.org/10.1007/s00355-010-0475-4

[43] Stephen Warshall. 1962. A Theorem on Boolean Matrices. *J. ACM* 9, 1 (1962), 2. https://doi.org/10.1145/321105.321107

[44] Roland Wen and Richard Buckland. 2008. *Mix and Test Counting in Preferential Electoral Systems.* Technical Report. University of New South Wales.

[45] Douglas Wikström. 2004. Universally Composable DKG with Linear Number of Exponentiations. In *Security in Communication Networks, 4th International Conference, SCN 2004 (Lecture Notes in Computer Science)*, Carlo Blundo and Stelvio Cimato (Eds.), Vol. 3352. Springer, 263–277.

[46] Douglas Wikström. 2005. A Sender Verifiable Mix-Net and a New Proof of a Shuffle. In *Advances in Cryptology - ASIACRYPT 2005 (Lecture Notes in Computer Science)*, Bimal K. Roy (Ed.), Vol. 3788. Springer, 273–292.

[47] Douglas Wikström. 2009. A Commitment-Consistent Proof of a Shuffle. In *Information Security and Privacy (ACISP'09)*. Springer.

Appendices

Part I: Building blocks.

Functionality	Option	Algorithm	Exp per trustee	Comm. cost	Transcript size
`Dec`	P/EG	`Dec`	$5a$	B	$4a$
`RandBit`	P/EG	`RandBit`	$3a+2$	R	$6a$
`CSZ`	EG	`CGate` [39]	$29a$	$R+4B$	$31a$
	P	`Mul` [41]	$10a$	$2B$	$11a$
`Select`	P/EG	`Select`	CSZ	CSZ	CSZ
`SelectInd`	P/EG	`SelectInd`	nCSZ	CSZ	nCSZ
$\texttt{Neg}^{\text{bits}}$	P/EG	$\texttt{Neg}^{\text{bits}}$	$(m-1)$CSZ	$(m-1)$CSZ	$(m-1)$CSZ
$\texttt{Add}^{\text{bits}}$	P/EG	$\texttt{Add}^{\text{bits}}$ [39]	$(2m-1)$CSZ	$(2m-1)$CSZ	$(2m-1)$CSZ
	Sublinear P/EG	$\texttt{UFCAdd}^{\text{bits}}$	$m(\frac{3}{2}\log m+2)$CSZ	$2(\log m+1)$CSZ	$m(\frac{3}{2}\log m+2)$CSZ
$\texttt{Sub}^{\text{bits}}$	P/EG	$\texttt{Sub}^{\text{bits}}$	$(2m-1)$CSZ	$(2m-1)$CSZ	$(2m-1)$CSZ
	LT P/EG	$\texttt{SubLT}^{\text{bits}}$	$(2m-1)$CSZ	$(2m-1)$CSZ	$(2m-1)$CSZ
	LT+EQ P/EG	$\texttt{SubLT}^{\text{bits}}$	$(3m-2)$CSZ	$(2m+\log m)$CSZ	$(3m-2)$CSZ
	Sublinear P/EG	$\texttt{UFCSub}^{\text{bits}}$	$m(\frac{3}{2}\log m+2)$CSZ	$2(\log m+1)$CSZ	$m(\frac{3}{2}\log m+2)$CSZ
$\texttt{LT}^{\text{bits}}$	LT P/EG	$\texttt{SubLT}^{\text{bits}}$	$(2m-1)$CSZ	$(2m-1)$CSZ	$(2m-1)$CSZ
	LT+EQ P/EG	$\texttt{SubLT}^{\text{bits}}$	$(3m-2)$CSZ	$(2m+\log m)$CSZ	$(3m-2)$CSZ
	Sublinear P/EG	$\texttt{CLT}^{\text{bits}}$	$(4m-3)$CSZ	$2(\log m+1)$CSZ	$(4m-3)$CSZ
	Sublinear+EQ P/EG	$\texttt{CLT}^{\text{bits}}$	$(5m-4)$CSZ	$2(\log m+1)$CSZ	$(5m-4)$CSZ
$\texttt{EQ}^{\text{bits}}$	P/EG	$\texttt{EQ}^{\text{bits}}$	$(2m-1)$CSZ	$(\log m+1)$CSZ	$(2m-1)$CSZ
`EQ`	Precomp P	`EQH` [31]	$21ma+75a$ $+4(m+1)$	$R+8B$	$(22m+28)a$
`GT`	Precomp P	`GTH` [31]	$(27m+146\log m)a$ $+8m+9a+5\log m$	$(2R+13B)\log m$	$(28m+50\log m)a$ $+6a$
`BinExpand`	P	`BinExpand` [40]	$12ma+53a+3m$	$R+2mB$	$(17m+21)a$
$\texttt{Aggreg}^{\text{bits}}$	EG	$\texttt{Aggreg}^{\text{bits}}$	$3n$CSZ	$(\log n+1)\log n$CSZ	$3n$CSZ
$\texttt{Mul}^{\text{bits}}$	P/EG	$\texttt{Mul}^{\text{bits}}$	$3m^2$CSZ	$2m^2$CSZ	$3m^2$CSZ
$\texttt{Div}^{\text{bits}}$	P/EG	$\texttt{Div}^{\text{bits}}$	$(3m-1)r$CSZ	$2mr$CSZ	$(3m-1)r$CSZ
$\texttt{MinMax}^{\text{bits}}$	naive P/EG	$\texttt{MinMax}^{\text{bits}}$	$(8m-2)n$CSZ	$2m\log n$CSZ	$(8m-2)n$CSZ
	sublinear P/EG	$\texttt{MinMax}^{\text{bits}}$	$(12m-6)n$CSZ	$2\log n(\log m+2)$CSZ	$(12m-6)n$CSZ
`Mixnet`	EG	[47]	$(9n+11)a$ $+n-6$	R	$10(n+1)a$
	P	[47]	$(8n+10)a$	R	$10(n+1)a$

Figure 6: Cost of various MPC primitives: basic functionalities for logic, integer arithmetic, and a few advanced functions. The Option column includes whether this is available in Paillier (P) or ElGamal (EG). The notations are a for the number of authorities, m for the bit-length of the operands, n for the number of operands, r for the precision (in the division). All logarithms are in base 2. The communication costs are expressed in terms of broadcast (denoted B) and full-rounds (denoted R). The unit of the transcript size is the key length. This corresponds to half the size of a ciphertext in both Paillier (typically 3072 bits) and ElGamal (typically 256 bits) settings.

In this Appendix, we give details about our cryptographic primitives. This includes the MPC building blocks that we present in Appendix C and which are summed up in Figure 6, but also more basic recalls about the ElGamal and Paillier cryptosystems in Appendix A and the Zero Knowledge Proofs (ZKP) in Appendix B. Afterwards, we explain how we use our MPC toolbox to perform the tally for single choice voting (in Appendix D), Majority Judgment (in Appendix E), the Condorcet methods (in Appendix F) and Single Transferable Choice (in Appendix G). All the security aspects are addressed in the second part of the Appendix, which begins with Appendix H.

A ELGAMAL AND PAILLIER CRYPTOSYSTEMS

In this section, we recall the encryption and decryption algorithms in the Paillier and ElGamal cryptosystems. Both are additively homomorphic. This allows efficient addition, subtraction, negation (flipping an encrypted bit) and re-encryption, without resorting to MPC. These are extremely useful for various uses. We sum up their complexity in Figure 7.

A.1 ElGamal and Paillier encryptions

ElGamal encryption.

In the ElGamal setting, G is a group of prime order q and public generators g and h. The public encryption key is (g, h), while the discrete logarithm of h in base g is the corresponding decryption key. To encrypt a message $m \in \mathbb{Z}_q$ under h, one chooses $r \in_r \mathbb{Z}_q$ and compute

$$\mathsf{Enc}(m, r) = (g^r, g^m h^r).$$

Note that this is different from the textbook ElGamal cryptosystem, since we encrypt g^m instead of m. Therefore, decrypting will require to solve a discrete logarithm problem and only small values of m can be efficiently decrypted. Hence we assume that computing g^m has a negligible complexity compared to that of the two other exponentiations. This modification grants the ElGamal cryptosystem the desired homomorphic property.

Consequently, Add and Sub are simply point-wise multiplication and division of ciphertexts, and we will often just use the multiplication or division symbols in our algorithms, without explicitly mentioning that they encode Add or Sub. We also have an almost free Not operation (divide an encryption of 1 by the operand) and a cheap ReEnc primitive (multiply the operand by an encryption of 0). Note that Not can use a fixed (trivial) encryption of 1, while ReEnc needs a fresh encryption of 0. Therefore Add, Sub and Not are essentially for free, while ReEnc costs two exponentiations.

Paillier encryption.

In the Paillier setting, n is a RSA integer, coprime with it's Euler's totient value $\phi(n)$. In addition, $g \in \mathbb{Z}_{n^2}$ is an element of order n, for instance $g = 1 + n$. To encrypt a message $m \in \mathbb{Z}_n$ under the public key (n, g), one chooses $r \in \mathbb{Z}_n^\times$ and computes

$$\mathsf{Enc}(m, r) = g^m r^n \mod n^2.$$

This encryption scheme is naturally homomorphic, which allows to derive the Add, Sub, Not and ReEnc primitives as above. Note that when m is small, computing an encryption of m only costs 1 exponentiation, as the other is either negligible or precomputable.

Functionality	Option	Exponentiations
Enc	P	1 or 2
	EG	2
Not	P/EG	0
Add/Sub	P/EG	0
ReEnc	P	1
	EG	2

Figure 7: Cost of non-MPC homomorphic operations. In the first line, when the plaintext is a small integer, the cost is only 1 exponentiation as the other is either precomputable or negligible.

A.2 Threshold decryption

We recall the distributed algorithms for threshold decryption in the Paillier and ElGamal setting. While threshold ElGamal is standard, there are several algorithms for threshold decryption in Paillier, and it is not straightforward to decide which one is the best. In the following, we consider the work of [22]. In both cases, the overall cost of the Dec decryption function is $5a$ exponentiations per authority (where a is the number of authorities), and it requires a single broadcast per authority.

ElGamal decryption.

In the ElGamal setting, the secret s such that $h = g^s$ is shared between the authorities using a Shamir secret sharing scheme, such as Pedersen's distribution scheme [35]. More precisely, there exists a polynomial P of degree t (where t is the threshold) such that $P(0) = s$, while authority i's share is $s_i = P(i)$. Each authority has a public commitment $h_i = g^{s_i}$ to their share, which allows to provide proofs of correct decryption.

In order to decrypt a ciphertext (x, y), each authority computes $w_i = x^{s_i}$ and provides a Zero Knowledge proof that $\log_x(w_i) = \log_g(h_i)$. The w_i are referred to as the *partial decryptions*. From any $t + 1$ valid partial decryptions, the value x^s can be recovered using Lagrange's interpolation. Finally, the plaintext is $m = log_g(y/x^s)$. The operations performed by authority i are described in Algorithm 4, which assumes that at least $t + 1$ valid partial decryptions are available after the for loop.

Algorithm 4: Decryption algorithm for authority i in the ElGamal setting

Require: (g, h), $(h_1, \cdots, h_a)$, hash, s_i, (x, y)
Ensure: m, a decryption of (x, y)

$w_i = x^{s_i}$
$\alpha_i \in_r \mathbb{Z}_q$
$e_{1,i} = g^{\alpha_i}$, $e_{2,i} = x^{\alpha_i}$
$d_i = \mathsf{hash}(g||h||h_1|| \cdots ||h_a||x||y||w_i||e_{1,i}||e_{2,i})$
$r_i = \alpha_i + s_i d_i$
for $j = 1$ *to* a $(j \neq i)$ **do**
  $d_j = \mathsf{hash}(g, h, h_1, \cdots, h_a, x, y, w_j, e_{1,j}, e_{2,j})$
  $b_j = (g^{r_j} h_j^{-d_j} == e_{1,j})$
  $b_j = b_j \wedge (g^{r_j} w_j^{-d_j} == e_{2,j})$
$S = \{i\} \bigcup \{j \in [1, a] \mid b_j = 1\}$
Compute Lagrange coefficients λ_k for set $S = \{j_1, \cdots, j_{t+1}\}$ (* keep only the first elements of S if it's larger *)
Return $\log_g \left(y \left(\prod_{k=1}^{t+1} w_{j_k}^{-\lambda_k} \right) \right)$

Paillier decryption.

In the Paillier setting, we use the approach from [22] as it provides a decryption algorithm which is similar to that of the ElGamal setting. In what follows, $g = (1 + n)$. Recall that n and $\phi(n)$ are coprime, so there exists a unique integer s in $\mathbb{Z}_{n\phi(n)}$ such that s is congruent to 1 modulo n and 0 modulo $\phi(n)$. This integer s is shared among the authorities using a Shamir secret sharing scheme, for instance using the work of [26], which can be generalized for an arbitrary number of authorities. Finally, we assume that a public random group element $g' \in_r \mathbb{Z}_{n^2}$ has been chosen, and that each authority has a public commitment $h_i = (g')^{s_i}$ to their share.

To decrypt a ciphertext C, each authority computes $w_i = C^{s_i}$ and provides a Zero Knowledge proof that w_i is well-formed (using the proof from [37]). Let $\Delta = a!$, where a is the number of authorities. For any $t + 1$ valid partial tallies, the value $D = C^{\Delta s}$ can be recovered using Lagrange's interpolation. Note that the Lagrange coefficients are multiplied by Δ because inverting an integer is infeasible modulo $n\phi(n)$, as $\phi(n)$ is unknown. Since Δ and n are coprime, Δ is invertible modulo n and we denote $u = \Delta^{-1} \bmod n$. We compute $D' = D^u \bmod n^2$ and cast D' into $\mathbb{Z}$ in order to derive $m = (D' - 1)/n$. The resulting m is the desired plaintext.

Note that another threshold scheme for the Paillier cryptosystem can be found in [34]. It is less similar to the ElGamal threshold scheme, and it requires an honest majority of authorities.

B ZERO KNOWLEDGE PROOFS

Zero Knowledge proofs are ubiquitous in our algorithms. Already, the decryption algorithms we have just mentioned include proofs of correct decryption for the partial tallies. Not trying to be exhaustive, we recall two standard zero knowledge proofs, and give their complexity in terms of exponentiations for the prover, the verifier, as well as the size of the transcript. All our Zero Knowledge proofs are made non-interactive with the Fiat-Shamir transformation, which requires a hash function hash. We decided to incorporate this function as an argument of the algorithms as some specific prefixes should be incorporated into the hash depending on the context (typically any public parameter). The precise specification of how the hashes should be prefixed to provide the correct level of security is out of the scope of our work, but still needs to be mentioned.

Standard encryption of 0**.**

Proving that a ciphertext is an encryption of 0 is useful to prove that two ciphertexts encrypt the same plaintext and, ultimately, to prove that a ciphertext has been correctly reencrypted. We give Algorithm 5, which is a standard way to obtain such a ZKP. To verify a ZKP obtained with this algorithm, simply compute $d = \mathsf{hash}(X||e)$ and check that $e = \mathsf{Enc}(0, a)X^{-d}$.

Standard $0/1$ **encryption.**

In all our algorithms, in particular on the voter side, it is extremely common to prove that some encryption is an encryption of either 0 or 1. We give Algorithm 6 which allows to produce such a proof given a bit b, a randomness r and an encryption $X = \mathsf{Enc}(b, r)$. This proof

Algorithm 5: ZKP0

Require: R, hash, X, r such that $X = \mathsf{Enc}(0, r)$
(* R is $\mathbb{Z}_q$ for ElGamal encryption, $\mathbb{Z}_n^\times$ for Paillier encryption *)
Ensure: π^0, a ZKP that X is a encryption of 0.

$w \in_r R$
$e = \mathsf{Enc}(0, w)$
$d = \mathsf{hash}(X||e)$
$a = w + rd$
Return $\pi^0 = (e, a)$

ZKP	P/EG	Exp. for Prover	Exp. for Verifier	Transcript size
π^0	EG	2	4	3
	P	1	2	3
$\pi^{0/1}$	EG	6	8	8
	P	4	4	8
π^{Shuffle}	EG	$10n + 5$	$9n + 11$	$10n + 10$
	P	$8n + 4$	$8n + 10$	$10n + 10$

Figure 8: Cost of the Zero Knowledge Proofs for $0/1$ encryptions and shuffles.

has the form $\pi^{0/1} = (e_1, e_2, \sigma_1, \rho_1, \sigma_2, \rho_2)$. To verify such a proof, one can simply compute $d = \mathsf{hash}(X||e_1||e_2)$ and check that the following equations are verified:

$$\begin{aligned}
&\sigma_1 + \sigma_2 = d\\
&\mathsf{Enc}(0, \rho_1)(X/E_1)^{-\sigma_1} = e_1\\
&\mathsf{Enc}(0, \rho_0)(X/E_0)^{-\sigma_0} = e_0.
\end{aligned}$$

Algorithm 6: ZKP01

Require: $R, \mathsf{hash}, X, r, b \in \{0, 1\}$ such that $X = \mathsf{Enc}(b, r)$
(* R is $\mathbb{Z}_q$ for ElGamal encryption, $\mathbb{Z}_n^\times$ for Paillier encryption *)
Ensure: $\pi^{0/1}$, a Zero Knowledge proof that X is an encryption of 0 or 1

$w \in_r R$
$e_b = \mathsf{Enc}(0, w)$
$\sigma_{1-b}, \rho_{1-b} \in_r R$
$e_{1-b} = \mathsf{Enc}(0, \rho_{1-b})(X/E_{1-b})^{-\sigma_{1-b}}$
$d = \mathsf{hash}(X||e_1||e_2)$
$\sigma_b = d - \sigma_{1-b}$
$\rho_b = w + r\sigma_b$
Return $\pi^{0/1} = (e_1, e_2, \sigma_1, \rho_1, \sigma_2, \rho_2)$

Proof of a shuffle.

We consider a prover P which is given a list of ciphertexts $C_1, \cdots, C_n$. The prover wants to shuffle the ciphertexts and output $C'_1, \cdots, C'_n$, while providing a proof π^{Shuffle} that there exists a permutation σ such that for all i, $C'_i = \mathsf{ReEnc}(C_{\sigma(i)})$. To do so, one can for instance apply the protocol from [47], which is a very standard approach in the ElGamal setting. This approach can be adapted in the Paillier setting. Note that a mixnet procedure can be derived from a proof of a shuffle, using a round of communication.

C OUR MPC TOOLBOX FOR EFFICIENT TALLY-HIDING

From now, we go on with proper MPC primitives and sum up their complexities in Figure 6. We will explain thoroughly how they can be obtained.

We stress that each functionality can be implemented in several manners, depending on the context. Choosing the implementation that best suits their need is left to the readers, and may imply implementations which are not presented in this section, as a more efficient replacement could exist in some specific context. For instance, we give a generic algorithm for adding two bit-wise encrypted integers (Algorithm 11). But when one of the operand is known in the clear, another implementation is available, which is twice as efficient (Algorithm 21). Such an optimisation is often possible for a very specific use, but we cannot anticipate every single specific situation.

A fundamental choice is however to decide how to encode integers. As explained in the main body of the article, in the ElGamal setting it is not possible to perform advanced arithmetic (in particular multiplication or comparison) if the integer is encrypted in the natural way (m is directly the integer to be dealt with). The *bitwise encryption* means that each bit of the integer m is encrypted individually. We recall that everything with an exponent $^{\text{bits}}$ means that this method is used. In the Paillier setting, the `BinExpand` function allows to convert an encrypted integer into its bitwise encryption. We postpone the description of this conversion; we will discuss it with other Paillier-specific algorithms (see Appendix C.5).

C.1 `CondSetZero` (abbreviated as `CSZ`) and selectors

The `CondSetZero` functionality is the basis of many other MPC primitives. Given two ciphertexts X and Y which encrypt x and y respectively, where $y \in \{0, 1\}$, it returns an encryption of xy. This algorithm is the basis of virtually all of our MPC algorithm in the ElGamal setting, but could be used in the Paillier setting as well. We present two algorithms for it. Algorithm 1, that is presented in Section 2.1 and reproduced below for conveniance, is adapted from [39], where it is referred to as the *conditional gate*. In the Paillier case, there exists a more efficient and more general algorithm [41], that we present as Algorithm 8: this is a general multiplication algorithm that does not require y to be a bit. The costs of these two variants are given in Figure 6.

We remark that the `CGate` algorithm requires raising the ciphertext to the power $1/2$. This can be done by raising to the power $(q+1)/2$ in ElGamal, or $(n+1)/2$ in Paillier. In the latter case, this works because while the full-group order is unknown, the cleartexts belong to $\mathbb{Z}_n$. Therefore, even though `Mul` is a faster implementation of `CSZ` in the Paillier setting, `CGate` could be used as well.

Algorithm 7: `CGate`

Require: X, Y such that X (resp. Y) is an encryption of x (resp. y), with $y \in \{0, 1\}$
Ensure: $Z = \mathsf{Enc}(xy)$
1 Compute $Y_0 = E_{-1}Y^2$, set X_0 at X
2 **for** $i = 1$ *to* a **do**
3   Authority i chooses $r_1, r_2 \in_r \mathbb{Z}_q$ and $s \in_r \{-1, 1\}$
4   She computes $X_i = \mathsf{ReEnc}(X_{i-1}^s, r_1)$ and $Y_i = \mathsf{ReEnc}(Y_{i-1}^s, r_2)$
5   She reveals X_i, Y_i and a zero knowledge proof that X_i and Y_i are well formed
6 Each authority verifies the proof of the other authorities
7 The authorities collectively rerandomize X_a and Y_a into X' and Y'
8 They collectively compute $y_a = \mathsf{Dec}(Y')$.
9 Return $Z = (XX'^{y_a})^{\frac{1}{2}}$

In line 5, `CGate` includes a Zero-Knowledge proof that X_i, Y_i are well formed. This proof guarantees that there exists $s \in_r \{-1, 1\}$ such that X_i (resp. Y_i) is a reencryption of X_{i-1}^s (resp. Y_{i-1}^s). To obtain such a proof, the authority i first chooses $s \in_r \{-1, 1\}$, $r_1, r_2 \in_r \mathbb{Z}_q$, computes X_i and Y_i, then proves that

$$X_i = \mathsf{ReEnc}(X_{i-1}) \wedge Y_i = \mathsf{ReEnc}(Y_{i-1})$$
$$\vee$$
$$X_i = \mathsf{ReEnc}(X_{i-1}^{-1}) \wedge Y_i = \mathsf{ReEnc}(Y_{i-1}^{-1}).$$

This can be done as follows, where we recall that s, r_1, r_2 are such that $X_i = \mathsf{ReEnc}(X_{i-1}^s, r_1)$ while $Y_i = \mathsf{ReEnc}(Y_{i-1}^s, r_2)$.

- Choose $\alpha, \beta \in_r \mathbb{Z}_q$ and compute $e_{s,X} = \mathsf{Enc}(0, \alpha)$, $e_{s,Y} = \mathsf{Enc}(0, \beta)$.
- Choose $\sigma_{-s}, \rho_{-s,X}, \rho_{-s,Y} \in_r \mathbb{Z}_q$.
- Compute $e_{-s,X} = \mathsf{Enc}(0, \rho_{-s,X})(X_i X_{i-1}^s)^{-\sigma_{-s}}$, $e_{-s,Y} = \mathsf{Enc}(0, \rho_{-s,Y})(Y_i Y_{i-1}^s)^{-\sigma_{-s}}$.
- Compute $d = \mathsf{hash}(g||h||X_{i-1}||Y_{i-1}||X_i||Y_i||e_{1,X}||e_{1,Y}||e_{-1,X}||e_{-1,Y})$.
- Compute $\sigma_s = d - \sigma_{-s}$, $\rho_{s,X} = \alpha + r_1\sigma_s$ and $\rho_{s,Y} = \beta + r_2\sigma_s$.
- Return $(e_{1,X}, e_{1,Y}, e_{-1,X}, e_{-1,Y}, \sigma_1, \sigma_{-1}, \rho_{1,X}, \rho_{1,Y}, \rho_{-1,X}, \rho_{-1,Y})$.

To verify the proof, compute $d = \mathsf{hash}(g||h||X_{i-1}||Y_{i-1}||X_i||Y_i||e_{1,X}||e_{1,Y}||e_{-1,X}||e_{-1,Y})$ and check that

$$\begin{aligned}
\sigma_1 + \sigma_{-1} &= d \\
\mathsf{Enc}(0, \rho_{1,X})(X_i/X_{i-1})^{-\sigma_1} &= e_{1,X} \\
\mathsf{Enc}(0, \rho_{1,Y})(Y_i/Y_{i-1})^{-\sigma_1} &= e_{1,Y} \\
\mathsf{Enc}(0, \rho_{-1,X})(X_i X_{i-1})^{-\sigma_{-1}} &= e_{-1,X} \\
\mathsf{Enc}(0, \rho_{-1,Y})(Y_i Y_{i-1})^{-\sigma_{-1}} &= e_{-1,Y}
\end{aligned}$$

In line 7, `CGate` includes a rerandomization phase. During this phase, the behavior of the authority i is as follows.

- Choose random $\alpha_i, \beta_i \in \mathbb{Z}_q$ and computes $A_i = \mathsf{Enc}(0, \alpha_X)$ and $B_i = \mathsf{Enc}(0, \alpha_Y)$. Broadcast $c_i = \mathsf{hash}(g||h||X_a||Y_a||A_i||B_i)$.
- When a c_j is received from all other authorities, reveal A_i, B_i, along with ZKP π_A^0 and π_B^0 that they are encryptions of 0.
- For each other authority j, check that $c_j = \mathsf{hash}(g||h||X_a||Y_a||A_j||B_j)$ and check the validity of the ZKP.
- Finally, set $X' = X_a \prod_j A_j$ and $Y' = Y_a \prod_j B_j$.

Since each authority has to check all the other authorities' proofs, this algorithm costs approximately $29a$ exponentiations per authority, where a is the number of authorities. The real value depends on the threshold, since a threshold decryption is needed in line 8. The value $29a$ is a reasonable upper-bound. The communication cost is one round of communication and four broadcasts.

Algorithm 8: `Mul` (Paillier only)

Require: X, Y, Paillier encryptions of $x, y \in \mathbb{Z}_n$
Ensure: Z, an encryption of xy

Authority i chooses $s_i \in_r \mathbb{Z}_n$ and $r_i \in_r \mathbb{Z}_n$
The authorities simultaneously reveal $S_i = \mathsf{Enc}(s_i, r_i)$, $Y_i = Y^{s_i}$ as well as a Zero Knowledge proof that S_i and Y_i are well formed
Each authority check the proof of the other authorities
$x' = \mathsf{Dec}(X \prod_i S_i)$ (* $x' = x + \sum_i s_i$ *)
They compute $Z' = Y^{x'}$, then $Z = Z' / \prod_i Y_i$

In Algorithm 8, there is also a Zero Knowledge proof required for the well-formedness of Y_i, S_i. The authority i can proceed as follows.

- Choose $\alpha, \beta \in_r \mathbb{Z}_n$ and compute $e_1 = \mathsf{Enc}(\alpha, \beta)$ and $e_2 = Y^{\alpha}$
- Compute $d = \mathsf{hash}(g, n, Y, Y_i, S_i, e_1, e_2)$, $a_1 = \alpha + d s_i$ and $a_2 = \beta r_i^d$
- Return (e_1, e_2, a_1, a_2)

To verify the proof, one can simply compute $d = \mathsf{hash}(g, n, Y, Y_i, S_i, e_1, e_2)$ and check that

$$\begin{aligned}
\mathsf{Enc}(a_1, a_2) S_i^{-d} &= e_1 \\
Y^{a_1} Y_i^{-d} &= e_2.
\end{aligned}$$

Since each authority has to check all the other authorities' proofs, the overall cost of the procedure is approximately $9a + 3$ exponentiation, where a is the number of authorities. The communication is also lower than in Algorithm 1 since it only requires broadcasts.

Selectors derived from `CSZ`

From the `CSZ` functionality, it is easy to build the `Select` function that allows to select a ciphertext among two, according to the value of an encrypted bit: Given two ciphertexts X and Y, and B an encryption of a bit b, this will return a reencryption of X if $b = 0$, of Y otherwise. This allows to remove branching by computing both branches and keeping only the relevant one, without revealing which one. An algorithm for `Select` is given in Algorithm 9. This can be generalized for bit-wise encrypted integers X^{bits}, Y^{bits} (simply return $(\mathtt{Select}(X_i, Y_i, B))_i$), or for wider branches with more than two possibilities (see Algorithm 10). The cost of `Select` is the same as the cost of `CSZ`.

Algorithm 9: `Select`

Require: X, Y, B encryptions of x, y and b with $b \in \{0, 1\}$
Ensure: Z, an encryption of x if $b = 0$, of y otherwise.

Return $Z = X\mathtt{CSZ}(Y/X, B)$

Universal verifiability for the `CSZ` protocol

Both algorithms `CGate` and `Mul` have the nice property that the participants are able to produce a transcript T which can be verified by an external auditor. For instance, in `CGate`, $T = T_1||T_2||T_3$, where $T_1 = (X_i, Y_i, \pi_i)_{1 \le i \le a}$, with π_i is a ZKP that X_i, Y_i are well-formed with respect to X_{i-1}, Y_{i-1}. For T_2, we have $T_2 = (A_i, B_i, \pi^0_{A_i}, \pi^0_{B_i})_i$, where, for all i, $\pi^0_{A_i}$ (resp. $\pi^0_{B_i}$) is a ZKP that A_i (resp. B_i) is an encryption of 0.

Algorithm 10: `SelectInd`

Require: $[X_i]$, an array of ciphertexts, $[B_i]$, an array of ciphertexts of the same size, such that one of them is an encryption of 1 while the others are encryptions of 0.
Ensure: Z, an encryption of x_i such that $b_i = 1$.
Return $Z = \prod_i \mathtt{CSZ}(X_i, B_i)$

Finally, $T_3 = (w_i, \pi_i^{\mathsf{Dec}})_i$ where, for all i, $\pi_{\mathsf{Dec},i}$ is a ZKP that w_i is a correct partial decryption of Y'. Note that since T only consists of ZKP, this transcript is Zero Knowledge (*i.e.* it does not leak any information about the initial inputs).

To verify a transcript T, an auditor first gets the public elements $pk, h_1, \cdots, h_a$, the input X, Y of the protocol and its output Z. Finally, the auditor computes $X_0 = X$, $Y_0 = E_{-1}Y^2$, verifies the ZKP π_i for all i, verify the ZKP $\pi^0_{A_i}$ and $\pi^0_{B_i}$ for all i, computes $X' = X_a \prod_i A_i$ and $Y' = Y_a \prod_i B_i$, verifies the ZKP π_i^{Dec} for all i, computes y_a from Y' and the w_i's, and checks that $Z = (XX'^{y_a})^{\frac{1}{2}}$.

If all the checks are successful, the auditor is guaranteed that, except with negligible probability, Z is an encryption of xy, where x (resp. y) is the plaintext for X (resp. Y). In this sense, the `CGate` protocol is universally verifiable. A similar result holds for the `Mul` protocol.

C.2 Basic integer arithmetic: addition, subtraction, comparison

Due to the homomorphic property, `Add` and `Sub` can be simply implemented by multiplication and division of the ciphertexts, when we want to work in the natural encoding. In bit-encoding, however, we need to build appropriate algorithms for these. We remark readily that comparing integers can be done with a subtraction, where we return the final borrow bit. However, we can sometimes do better.

Linear addition and subtraction.

Suppose that we have as input the (encrypted) bits X^{bits} and Y^{bits} of x and y, where x and y are the m-bit plaintexts associated with X^{bits} and Y^{bits} respectively. For addition, *i.e.* computing Z^{bits}, an encryption of $x + y$ modulo 2^m, we reproduce in Algorithm 11 the method found in [39]. The idea is simply to reproduce the schoolbook algorithm for the addition, with four variables X_i, Y_i, Z_i and R which represent (encryptions of) the i^{th} bit of X and Y, the i^{th} bit of the sum and the current value of the carry. The value of z_i, the plaintext associated with Z_i, is simply $x_i \oplus y_i \oplus r_i$, and the new value of R can be obtained with a truth table from the three other variables.

Algorithm 11: $\mathtt{Add}^{\text{bits}}$

Require: $(X_0, \cdots, X_{m-1})$, $(Y_0, \cdots, Y_{m-1})$ bit-wise encryptions of x and y
Ensure: $Z_0, \cdots, Z_{m-1}$, bitwise encryption of $x + y$ modulo 2^m
$R = \mathtt{CSZ}(X_0, Y_0)$
$Z_0 = X_0 Y_0 / R^2$ (* $x_0 \oplus y_0$ *)
for $i = 1$ *to* $m - 1$ **do**
  $A = X_i Y_i / \mathtt{CSZ}(X_i, Y_i)^2$ (* $x_i \oplus y_i$ *)
  $Z_i = AR / \mathtt{CSZ}(A, R)^2$ (* $x_i \oplus y_i \oplus r$ *)
  $R = (X_i Y_i R / Z_i)^{\frac{1}{2}}$
Return $Z_0, \cdots, Z_{m-1}$

A first approach for writing a subtraction algorithm that returns an encryption of $x - y \bmod 2^m$ is to modify Algorithm 11 as follows. Computing $x - y \bmod 2^m$ is the same as computing $x + (-y) \bmod 2^m$. Turning y to $-y \bmod 2^m$ is performed by flipping each bit (replacing y_i by $1 - y_i$) then adding 1. This gives Algorithm 12.

Algorithm 12 is interesting for its similarity with Algorithm 11, but another way to perform the subtraction is also to use the schoolbook algorithm, just as for Algorithm 11. The advantage is that the carry is then the classical borrow of the subtraction, and not an artificial carry in an equivalent addition modulo 2^m. Hence, if the last borrow bit is required in order to get a comparison algorithm from the subtraction, Algorithm 13 must be preferred.

When the required comparison is just an equality test, there is a simpler and cheaper approach. Indeed, testing whether two integers are equal is the same as testing whether all of their bits are equal, therefore the associativity of the logical $\wedge$ operator can be exploited to parallelize the procedure. This gives Algorithm 14, the cost of which is $(2m - 1)\mathtt{CSZ}$ in term of transcript size and exponentiations per authority, but only $(1 + \log m)\mathtt{CSZ}$ in term of communication cost, using a tree structure.

Therefore, adding, subtracting or comparing two m-bit integers have roughly the same cost of $(2m - 1)\mathtt{CSZ}$. The similarity between these algorithms can be exploited to build specialized algorithm which do several operations altogether.

For instance, if one need an algorithm that computes both the subtraction and the full comparison as a ternary value (1 if $x > y$, 0 if $x = y$ or -1 if $x < y$), we can combine these operations by first calling Algorithm 13, then using a $\vee$ composition to test whether all the bits of the output are 0. This leads to a cost of about $3m\mathtt{CSZ}$ instead of $4m\mathtt{CSZ}$ if done separately. This is useful, for instance in our version of Condorcet.

Algorithm 12: $\mathsf{Sub}^{\text{bits}}$

Require: $(X_0, \cdots, X_{m-1})$, $(Y_0, \cdots, Y_{m-1})$, bit-wise encryptions of x and y.
Ensure: $(Z_0, \cdots, Z_{m-1})$, bit-wise encryption of $x - y$ modulo 2^m.

```
A = CSZ(X_0, Y_0)
Z_0 = (X_0 Y_0)/A^2 (* x_0 ⊕ (1 − y_0) ⊕ 1 *)
R = A Not(Y_0) (* x_0 ∨ ¬y_0 *)
for k = 1 to m − 1 do
    A = X_k Not(Y_k)/CSZ(X_k, Not(Y_k))^2
    Z_k = AR/CSZ(A, R)^2
    R = (X_k Not(Y_i) R / Z_k)^(1/2)
Return Z_0, ··· , Z_{m−1}
```

Algorithm 13: $\mathsf{SubLT}^{\text{bits}}$

Require: $(X_0, \cdots, X_{m-1})$, $(Y_0, \cdots, Y_{m-1})$, bit-wise encryption of x and y.
Ensure: $(Z_0, \cdots, Z_{m-1}), R$ where Z_i are bit-wise encryption of $x - y$ modulo 2^m and $R = \mathsf{Enc}(x < y)$.

```
A = CSZ(X_0, Y_0)
Z_0 = X_0 Y_0/A^2 (* x_0 ⊕ y_0 *)
R = Y_0/A (* y_0 ∧ ¬x_0 *)
for k = 1 to m − 1 do
    A = CSZ(Y_k, R)
    B = Y_k R/A^2 (* y_k ⊕ r *)
    C = CSZ(X_k, B)
    Z_k = X_k B/C^2 (* x_k ⊕ y_k ⊕ r *)
    R = Y_k R/(AC) (* (y_k ∧ r) ∨ [(y_k ∨ r) ∧ ¬x_k] *)
Return (Z_0, ··· , Z_{m−1}), R
```

Algorithm 14: $\mathsf{EQ}^{\text{bits}}$

Require: $X_0, \cdots, X_m$, $Y_0, \cdots, Y_m$ bit-wise encryptions of x and y.
Ensure: $Z = \mathsf{Enc}(x == y)$, an encryption of 1 if $x = y$, of 0 otherwise.

```
For all i (in parallel), compute A_i = CSZ(X_i, Y_i).
For all i (in parallel), compute B_i = E_1 A_i^2/(X_i Y_i) (* 1 − x_i ⊕ y_i *)
Return Z = CSZ(B_0, ··· , B_{m−1})
```

Finally, we remark that computing the opposite $-x$ of an integer x modulo 2^m can be done faster than using the subtraction algorithm between 0 and x. Algorithm 15 simply flips all bits and then add 1; this is simply a special case of Algorithm 21 which be introduced later on.

Algorithm 15: $\mathsf{Neg}^{\text{bits}}$

Require: $(X_0, \cdots, X_{m-1})$, a bit-wise encryption of x
Ensure: $(Z_0, \cdots, Z_{m-1})$, a bit-wise encryption of $-x \bmod 2^m$

```
Z_0 = X_0
R_0 = Not(X_0)
for i = 1 to m − 1 do
    R_i = CSZ(Not(X_i), R_{i−1})
    Z_i = Not(X_i) R_{i−1}/R_i^2
Return Z_0, ··· , Z_{m−1}
```

C.3 Arithmetic with sublinear communication complexity

Apart from the equality test, all the previous arithmetic algorithms in the bit-encoding require a number of communication rounds that is proportional to the bit-size of the input integers. This is mostly due to carry and borrow propagations. In order to reduce the number of communication rounds, our idea is to use more sophisticated adder circuits, following the (now classical) approach of Brent and Kung [13]. We do not reproduce their full algorithm here but we sketch the key idea and give the resulting algorithms and their complexity (summarized in Figure 6).

Recall that the i^{th} bit of $x + y$ is simply $z_i = x_i \oplus y_i \oplus r_i$, where r_i is the i^{th} carry bit. The idea is to first compute all the $x_i \oplus y_i$ in parallel, then to compute all the r_i in parallel, so as to deduce the result. To perform the second step efficiently, Brent and Kung's approach consists of computing the variables (p_i, g_i) where $p_i = x_i \vee y_i$ and $g_i = x_i \wedge y_i$. Those variable are used to encode elements of a set $\Sigma = \{P, G, K\}$, where P is encoded by $(1, 0)$, K by $(0, 0)$ and G by $(0, 1)$ and $(1, 1)$. They represent the fact that the carry bit will be propagated, generated of killed in the i^{th} position. They define an operation $\circ$ as follows (which we slightly modify into an equivalent operation for the sake of presentation).

$$\begin{aligned} P \circ P &= P \\ G \circ P &= G \\ K \circ P &= K \\ x \circ G &= G \\ x \circ K &= K. \end{aligned}$$

In the boolean representation, the $\circ$ law can be computed with the following formula:

$$(p, g) \circ (p', g') = (p \wedge p', g' \vee (p' \wedge g)).$$

It is easy to show that $\circ$ is associative [13], which enables tree-based parallelism for computing all the prefixes of $(p_0, g_0) \circ \cdots \circ (p_{m-1}, g_{m-1})$, which gives essentially the i^{th} carry bit for all i. From here onward, we diverge from [13]'s work since we are not interested in designing hardware, so the unbounded fan-in is not an issue. We deduce the Unbounded Fan-in Composition algorithm, which can be instantiated to compute the addition (Algorithm 16). Algorithm 16 is highly efficient in term of communication since it only requires about $\log(m)$ times more round communications than the one required for $\circ$. However, this comes with an increase in term of computation as the number of calls to $\circ$ is about $\frac{1}{2} m \log(m)$, so the linear approach could be preferable in some cases. To evaluate the complexity, note that the worst-case scenario in term of computational cost is when m is a power of 2, in which case the number of calls to CSZ is easy to derive.

Algorithm 16: $\mathsf{UFCAdd}^{\text{bits}}$

Require: $(X_0, \cdots, X_{m-1})$, $(Y_0, \cdots, Y_{m-1})$, bit-wise encryptions of x and y.
Ensure: $(Z_0, \cdots, Z_{m-1})$, bit-wise encryption of $x + y \bmod 2^m$

for $i = 0$ *to* $m - 1$ *(* in parallel *)* **do**
  $A_i = \mathsf{CSZ}(X_i, Y_i)$
  $B_i = X_i Y_i / A_i^2$ (* $x_i \oplus y_i$ *)
  $P_i = X_i Y_i / A_i$ (* $x_i \vee y_i$ *)
  $G_i = A_i$ (* $x_i \wedge y_i$ *)
$C_{i,j} = (P_j, G_j)$ for all $1 \leq i \leq \lceil \log m \rceil$ and $0 \leq j \leq m - 1$
for $i = 1$ *to* $\lceil \log m \rceil$ **do**
  for $j = 0$ *to* $\lceil m/2^i \rceil - 1$ *(in parallel)* **do**
    for $k = 1$ *to* 2^{i-1} *(in parallel)* **do**
      $(P, G) = C_{i-1, j2^i + 2^{i-1}}$
      $(P', G') = C_{i-1, j2^i + 2^{i-1} + k}$ (* do not proceed for this k if $j2^i + 2^{i-1} + k \geq m$ *)
      $T = \mathsf{CSZ}(P', G)$
      $C_{i, j2^i + 2^{i-1} + k} = (\mathsf{CSZ}(P, P'), TG' / \mathsf{CSZ}(T, G'))$
$Z_0 = B_0$
for $i = 1$ *to* $m - 1$ *(in parallel)* **do**
  $(\_, G_i) = C_{\lceil \log(i+1) \rceil, i+1}$
  $Z_i = B_i G_i / \mathsf{CSZ}(B_i, G_i)^2$
Return $Z_0, \cdots, Z_{m-1}$

The same algorithm can be used for computing subtraction; it only requires to change the initialization of the p_i and g_i. Indeed, we have initially $p_i = x_i \oplus y_i$ and $g_i = y_i \wedge \neg x_i$, so to obtain $\texttt{UFCSub}^{\text{bits}}$, one can just replace line 4 by $P_i = B_i$ and line 5 by $G_i = Y_i / A_i$ in Algorithm 16.

When it comes to comparing two integers, only the last carry bit is of interest so we do not need to compute all the prefixes. In this case, a much simpler algorithm exists and allows to compute the comparison with $m - 1$ calls to $\circ$ but a communication cost which remains of the order of $\log(m)$. We call this algorithm Chained Lesser-Than (see Algorithm 17). Note that this algorithm returns an additional bit R which tells whether the two inputs are equal. If this bit is not needed, some computations can be saved (remove lines 11 and 20).

Algorithm 17: $\texttt{CLT}^{\text{bits}}$

Require: $(X_0, \cdots, X_{m-1}), (Y_0, \cdots, Y_{m-1})$ bit-wise encryption of x and y.
Ensure: Z, R such that $Z = \texttt{Enc}(x < y)$ and $R = \texttt{Enc}(x = y)$

1 Let $(m_j)_{j=0}^{l-1}$ be the binary representation of m, such that $m = \sum_{j=0}^{l-1} m_j 2^j$
2 **for** $i = 0$ *to* $m - 1$ *(in parallel)* **do**
3     $A_i = \texttt{CSZ}(X_i, Y_i)$
4     $P_i = X_i Y_i / A_i^2$
5     $G_i = Y_i / A_i$
6     $B_{i,0} = (P_i, G_i)$, $A_{i,0} = P_i$.
7 $r = 0$ (* A boolean which tells whether there is a remainder *)
8 $R_B = (E_1, E_0)$, $R_A = E_1$ (* initialize the remainders as neutral element *)
9 **for** $j = 1$ *to* l **do**
10   **for** $i = 0$ *to* $\lfloor l/2^j \rfloor - 1$ *(in parallel)* **do**
11         $A_{i,j} = \texttt{CSZ}(A_{2i,j-1}, A_{2i+1,j-1})$
12         $(P, G) = B_{2i,j-1}$
13         $(P', G') = B_{2i+1,j-1}$
14         $T = \texttt{CSZ}(P', G)$
15         $B_{i,j} = (\texttt{CSZ}(P, P'), TG'/\texttt{CSZ}(T, G'))$
16   **if** $m_{j-1} \wedge \neg r$ *(in parallel)* **then**
17         $R_B = B_{2\lfloor l/2^j \rfloor, j-1}$, $R_A = A_{2\lfloor l/2^j \rfloor, j-1}$
18         $r = 1$
19   **if** $m_{j-1} \wedge r$ *(in parallel)* **then**
20         $A_{2\lfloor l/2^j \rfloor, j-1} = \texttt{CSZ}(A_{2\lfloor l/2^j \rfloor, j-1}, R_A)$
21         $(P, G) = B_{2\lfloor l/2^j \rfloor, j-1}$
22         $(P', G') = R_B$
23         $T = \texttt{CSZ}(P', G)$
24         $B_{2\lfloor l/2^j \rfloor, j} = (\texttt{CSZ}(P, P'), TG'/\texttt{CSZ}(T, G'))$
25         $r = 0$, $R_B = (\texttt{Enc}(1), \texttt{Enc}(0))$, $R_A = \texttt{Enc}(1)$
26 $(\_, G) = B_{0,l}$
27 Return $G, A_{0,l}$

C.4 An example: Searching the minimum and the maximum

As an illustration of the previous functionalities, we describe the $\texttt{MinMax}$ algorithm which takes as input a list of bit-wise encrypted integers ${X_1}^{\text{bits}}, \cdots, {X_n}^{\text{bits}}$ and returns two bit-wise encrypted integers $Z^{\text{bits}}, T^{\text{bits}}$ which correspond to the minimum and the maximum of the list, as well as bit-wise encryptions of their indexes in the list ${X_1}^{\text{bits}}, \cdots, {X_n}^{\text{bits}}$. A straightforward implementation would be to linearly scan the list, using a comparison algorithm. However, the min and max operators are associative and as such, allows tree-based parallelization. This gives Algorithm 18, in which $\texttt{LT}^{\text{bits}}$ is either $\texttt{CLT}^{\text{bits}}$ or the second output of $\texttt{SubLT}^{\text{bits}}$, depending on the focus in optimization (use $\texttt{CLT}^{\text{bits}}$ for a better communication cost, and $\texttt{SubLT}^{\text{bits}}$ for a better computation cost). We give the two corresponding costs in Figure 6.

C.5 Paillier-specific algorithms

Conversion from natural to bit-encoding (Paillier only)

Algorithm 18: $\texttt{MinMax}^{\text{bits}}$

Require: $({X_1}^{\text{bits}}, \cdots, {X_n}^{\text{bits}})$ bit-wise encryptions of $x_1, \cdots, x_n$ and y
Ensure: $Z^{\text{bits}}, T^{\text{bits}}, I^{\text{bits}}, J^{\text{bits}}$, bitwise encryptions of $\min_{i=1}^{n}(x_i)$ and $\max_{i=1}^{n}(y_i)$, as well as their indexes in the input vector

Let $(n_j)_{j=0}^{l-1}$ be the binary representation of n, such that $n = \sum_{j=0}^{l-1} n_j 2^j$
${Z_{i,0}}^{\text{bits}} = {T_{i,0}}^{\text{bits}} = {X_i}^{\text{bits}}$ for all i
${I_{i,0}}^{\text{bits}} = {J_{i,0}}^{\text{bits}} = i^{\text{bits}}$ for all i (* trivial bit-wise encryption of i *)
$r = 0$ (* A boolean which tells whether there is a remainder *)
for $j = 1$ *to* l **do**
  for $i = 1$ *to* $\lfloor l/2^j \rfloor$ *(in parallel)* **do**
    (* Both operations are performed in parallel *)
    $B_Z = \texttt{LT}^{\text{bits}}({Z_{2i-1,j-1}}^{\text{bits}}, {Z_{2i,j-1}}^{\text{bits}})$
    $B_T = \texttt{LT}^{\text{bits}}({T_{2i,j-1}}^{\text{bits}}, {T_{2i-1,j-1}}^{\text{bits}})$
    (* All four operations are performed in parallel *)
    ${Z_{i,j}}^{\text{bits}} = \texttt{Select}^{\text{bits}}({Z_{2i,j-1}}^{\text{bits}}, {Z_{2i-1,j-1}}^{\text{bits}}, B_Z)$
    ${T_{i,j}}^{\text{bits}} = \texttt{Select}^{\text{bits}}({T_{2i,j-1}}^{\text{bits}}, {T_{2i-1,j-1}}^{\text{bits}}, B_T)$
    ${I_{i,j}}^{\text{bits}} = \texttt{Select}^{\text{bits}}({I_{2i,j-1}}^{\text{bits}}, {I_{2i-1,j-1}}^{\text{bits}}, B_Z)$
    ${J_{i,j}}^{\text{bits}} = \texttt{Select}^{\text{bits}}({J_{2i,j-1}}^{\text{bits}}, {J_{2i-1,j-1}}^{\text{bits}}, B_T)$
  if $m_{j-1} \wedge \neg r$ **then**
    $R_Z = Z_{2\lfloor l/2^j \rfloor, j-1}$, $R_T = T_{2\lfloor l/2^j \rfloor, j-1}$
    $R_I = I_{2\lfloor l/2^j \rfloor, j-1}$, $R_J = J_{2\lfloor l/2^j \rfloor, j-1}$
    $r = 1$
  if $m_{j-1} \wedge r$ **then**
    $i = \lfloor l/2^j \rfloor + 1$
    $B_Z = \texttt{LT}^{\text{bits}}({Z_{2i-1,j-1}}^{\text{bits}}, {R_Z}^{\text{bits}})$
    $B_T = \texttt{LT}^{\text{bits}}(R_T, {T_{2i-1,j-1}}^{\text{bits}})$
    ${Z_{i,j}}^{\text{bits}} = \texttt{Select}^{\text{bits}}({R_Z}^{\text{bits}}, {Z_{2i-1,j-1}}^{\text{bits}}, B_Z)$
    ${T_{i,j}}^{\text{bits}} = \texttt{Select}^{\text{bits}}({R_T}^{\text{bits}}, {T_{2i-1,j-1}}^{\text{bits}}, B_T)$
    ${I_{i,j}}^{\text{bits}} = \texttt{Select}^{\text{bits}}({R_I}^{\text{bits}}, {I_{2i-1,j-1}}^{\text{bits}}, B_Z)$
    ${J_{i,j}}^{\text{bits}} = \texttt{Select}^{\text{bits}}({R_J}^{\text{bits}}, {J_{2i-1,j-1}}^{\text{bits}}, B_T)$
    $r = 0$
Return ${Z_{1,l}}^{\text{bits}}, {T_{1,l}}^{\text{bits}}, {I_{1,l}}^{\text{bits}}, {J_{1,l}}^{\text{bits}}$

We recall here the work from [40] which allows to get the bit-encoding representation from a Paillier-encrypted integer. While the homomorphic property of the Paillier cryptosystem allows extremely efficient solutions for the addition and the subtraction, the comparison is not so easy to perform. Therefore, it is important to provide an algorithm which converts to the bit-encoding.

The idea is to use the mask-and-decrypt paradigm, which consists of applying a random mask r to the encrypted value x, which gives an encryption of $x - r$, to decrypt $y = x - r$ then to perform the relevant operation (here, an addition with r) to deduce the (encrypted) result. The overall process that we call `BinExpand` is described by Algorithm 22. To create a mask, we use Algorithm 20 from [40] which requires Zero Knowledge Ranged proofs, such as the ones from [30] or more recently [14]. We do not dig too deep into the details as the security, correction and complexity is fully discussed in [40]. We emphasize that these Paillier-specific algorithms use a `RandBit` function given by Algorithm 19 which is also available in ElGamal. The same holds for the $\texttt{AddKnown}^{\text{bits}}$ function of Algorithm 21 which is a variant of $\texttt{Add}^{\text{bits}}$ where one operand is a cleartext. While we did not need them for the tally function that we studied, they might prove useful in other contexts.

Integer comparison with precomputation and sublinear online complexity (Paillier only)
We mention here a work from [31], which is only exploitable in the Paillier setting. They present some algorithms for the equality test and the comparison which are mostly precomputable. We do not go into all the details here and refer to [31] for a more complete description. To compare two m bits integers x and y, Lipmaa and Toft suggest to create the unique polynomial P_m such that $P_m(1) = 1$ and $P_m(k) = 0$ for $k \in \{2, \cdots, m+1\}$. Their strategy is to first compute the Hamming weight h of $x - y$, then to evaluate P_m on $1 + h$, from which they derive the result. To do so, they use some classical primitives in MPC (Algorithms 19, 23 and 24). The overall process is presented in Algorithm 25.

Algorithm 19: `RandBit`

Ensure: Z, an encryption of $b \in_r \{0,1\}$

$Z_0 = \mathsf{Enc}(1)$
for $i = 1$ *to* a **do**
    Authority i chooses $s_i \in_r \{-1,1\}$ and $r \in_r \mathbb{Z}_n$
    She reveals $Z_i = Z_{i-1}^{s_i}\mathsf{Enc}(0,r)$, as well as a Zero Knowledge proof of well-formedness
The authorities check each others' proofs
Return $(E_1 Z_a)^{\frac{1}{2}}$

Algorithm 20: `RandBits` (Paillier only)

Require: m, a number of bits
Ensure: R, $(R_0, \cdots, R_{m-1})$ such that R is an encryption of $r \in_r \mathbb{Z}_n$, while $(R_0, \cdots, R_{m-1})$ are encryptions of the m first (least significant) bits of r

for $i = 0$ *to* $m-1$ **do**
    $R_i = \mathsf{RandBit}()$
Each authority i chooses $r_{*,i} \in_r [0, 2^{m+\kappa-1}-1]$ and publishes $R_{*,i} = \mathsf{Enc}(r_{*,i})$, along with a Zero Knowledge Ranged Proof
$R = \prod_{i=1}^{a} R_{*,i}$
for $i = m-1$ *to* 0 **do**
    $R = R^2$
    $R = RR_i$
Return R, $(R_0, \cdots, R_{m-1})$

Algorithm 21: $\mathsf{AddKnown}^{\text{bits}}$

Require: $(X_0, \cdots, X_{m-1})$ bit-wise encryptions of x and bits $(y_0, \cdots, y_{m-1})$
Ensure: $Z_0, \cdots, Z_{m-1}$, bitwise encryption of $x+y$ modulo 2^m

$R = X_0^{y_0}$
$Z_0 = X_0\mathsf{Enc}(y_0,1)/R^2$ (* $x_0 \oplus y_0$ *)
for $i = 1$ *to* $m-1$ **do**
    $A = X_i\mathsf{Enc}(y_i,1)/(X_i^{y_i})^2$ (* $x_i \oplus y_i$ *)
    $Z_i = AR/\mathsf{CSZ}(A,R)^2$ (* $x_i \oplus y_i \oplus r$ *)
    $R = (X_i\mathsf{Enc}(y_i,1)R/Z_i)^{\frac{1}{2}}$
Return $Z_0, \cdots, Z_{m-1}$

Algorithm 22: `BinExpand` (Paillier only)

Require: X, an encryption of $x < 2^m$
Ensure: $X_0, \cdots, X_{m-1}$, the bit-wise encryption of x

$R, (R_0, \cdots, R_{m-1}) = \mathsf{RandBits}(m)$
$Y = X/R$
$y' = \mathsf{Dec}(Y)$ (* $y' = x - r$ modulo n *)
Let $y = y' - n$ modulo 2^m and $(y_0, \cdots, y_{m-1})$ the bits of y
Return $\mathsf{AddKnown}^{\text{bits}}((R_0, \cdots, R_{m-1}), (y_0, \cdots, y_{m-1}))$

The advantage of this approach is that the procedure can be precomputed so that only a small part has to be done *online*, after the operands are known. Compared to Algorithm 14, Algorithm 25 does not require the costly binary expansion as the inputs do not have to be bit-wise encrypted. The complexity of the procedure is less dependent in m, the bit size of the integers to compare, but is of the same order. When m is small, which might be the case in an e-voting setting, the complexity is much higher due to some constant overloads. However, since most of the procedure can be precomputed, the approach is of interest even when m is small.

Algorithm 23: `RandInv` (Paillier only)

Ensure: R, R', encryptions of $r \in_r \mathbb{Z}_n^x$ and $r' \in \mathbb{Z}_n$ such that $r' = r^{-1}$

The authorities (simultaneously) display two ciphertexts A_i, B_i

$A = \prod_i A_i$, $B = \prod_i B_i$, $C = \mathtt{Mul}(A, B)$

$c = \mathtt{Dec}(C)$

$R = A$, $R' = B^{c^{-1}}$.

Return R, R'.

Algorithm 24: `Prefixes` (Paillier only)

Require: $M_1, \cdots, M_m$ encryptions of $m_1, \cdots, m_m$, each coprime with n

Ensure: $Z_1, \cdots, Z_m$, encryptions of $m_1, m_1 m_2, \cdots, \prod_i m_i$

for $i = 1$ *to* m *(in parallel)* **do**

  $R_i, R'_i = \mathtt{RandInv}()$

  $S_i = \mathtt{Mul}(R_{i-1}, M_i)$ (* with $R_0 = 1$ *)

  $S_i = \mathtt{Mul}(S_i, R'_i)$ (* $s_i = r_{i-1} m_i r_i^{-1}$ *)

  The authority decrypt S_i to get s_i.

for $i = 2$ *to* m *(in parallel)* **do**

  $a_i = \prod_{j=1}^{i} s_j$

  $Z_i = R_i^{a_i}$

Return $Z_1, \cdots, Z_m$ (* with $Z_1 = M_1$ *)

Algorithm 25: `EQH` (Paillier only)

Require: X, m, P_m, where X is an encryption of an integer $x << n$ and P_m the unique polynomial of degree m such that $P_m(1) = 1$ and $P_m(k) = 0$ for $k \in \{2, \cdots, m+1\}$

Ensure: Z, an encryption of 1 if $x = 0 \bmod 2^m$, of 0 otherwise

$R, R_{m-1}, \cdots, R_0 = \mathtt{RandBits}(m)$

$M, M' = \mathtt{RandInv}()$

$M_1, \cdots, M_m = \mathtt{Prefixes}(M, \cdots, M)$

$A = X/R$

$a = \mathtt{Dec}(A)$

Let $a_0, \cdots, a_{m-1}$ be the bit representation of $a - n$ modulo 2^m

$H = \mathtt{Enc}(1) \prod_{i=0}^{m-1} \mathtt{Enc}(a_i) R_i^{1-2a_i}$ (* $h = 1 + \sum_{i=0}^{m-1} a_i \oplus r_i$ *)

$M_H = \mathtt{Mul}(M', H)$

$m_H = \mathtt{Dec}(M_H)$

for $i = 0$ *to* m *(in parallel)* **do**

  $H_i = M_i^{(m_H)^i}$

Return $Z = \prod_{i=0}^{m} H_i^{\alpha_i}$ (* where the α_i are the coefficients of P_m *)

The `RandInv` Algorithm 23 (adapted from [8]) allows to (collectively) generate two ciphertexts R, R', which encrypt respectively r and r'. The plaintext r is a random invertible integer (modulo n, the Paillier public key), while $r' = r^{-1}$.

The `Prefixes` Algorithm 24 (adapted from [8]) takes as input m ciphertexts $M_1, \cdots, M_m$ which are encryptions of $m_1, \cdots, m_m$. This algorithm returns ciphertexts $Z_1, \cdots, Z_m$ such that Z_i is an encryption of $\prod_{1 \le j \le i} m_j$.

From these, in [31], the authors present two algorithms for the inequality test in the Paillier setting, but we will only present one of them. The idea is to use a recursive algorithm which first tests the equality of the most significant halves of x and y, using Algorithm 25. If they are equals, we recursively compare the integers represented by the other halves. If not, we recursively compare the integers represented by the most significant halves. The main process is given in Algorithm 26. Note that at line 5, we took the liberty to denote $R_\top, R_\bot$ the result of `BinExpand` while `BinExpand` returns encryptions of the form $R, (R_0, \cdots, R_{l-1})$. We can derive $R_\bot$ as $\prod_{i<l/2} (R_i)^{2^i}$ and $R_\top$ in a similar manner.

Algorithm 26: GTH (Paillier only)

Require: X, Y, l, two encryptions of l-bit integers x and y
Ensure: Z, T, encryptions of $(x \geq y)$ and $(x = y)$

```
if l = 1 then
    A = Mul(X, Y)
    T = E_1 A^2/(XY) (* ¬(x ⊕ y) *)
    A Not(Y) (* ¬(y ∧ ¬x) *)
R, R_⊤, R_⊥ = BinExpand(l)
W = Enc(2^l)X/Y
M = WR
m = Dec(M)
m_⊤ = m mod 2^{l/2}, m_⊥ = ⌊m/2^{l/2}⌋ mod 2^{l/2}
B = EQH(Enc(m_⊤), R_⊤) (* x_⊤ = y_⊤ *)
C = Mul(B, Enc(m_⊥ − m_⊤))Enc(m_⊤) (* m_⊥ if b = 1, m_⊤ otherwise *)
D = Mul(B, R_⊥/R⊤)R_⊤ (* r_⊥ if b = 1, r_⊤ otherwise *)
F = Enc(1)/GTH(C, D)
W′ = F^{2^l} Enc(m mod 2^l)/(R_⊤^{l/2} R⊥) (* w mod 2^l *)
Return Z = (W/W′)^{1/2^l}, T
```

C.6 Advanced arithmetic: aggregation, multiplication and division

In the Paillier setting, integers can be represented in the natural encoding, and we already gave a multiplication algorithm Mul as a way to implement the CSZ functionality. We now come to the more difficult question of doing multiplication and other arithmetic operations in the bit-encoding, in order to have them available in the ElGamal setting.

Aggregation of several encrypted bits

This $\mathsf{Aggreg}^{\text{bits}}$ operation is ubiquitous in e-voting. More often than not, the ballots of the voters are encoded as a sequence of encrypted bits and the first step of the tally is to aggregate some of them, *i.e.* counting all the bits that are set at a given position. The resulting encrypted integers should be in the bit-encoding format, so as to be able to perform comparisons (for instance). The algorithm for that is pretty simple: we just use repeatedly the addition algorithm 11, each time with the minimal value of m, the bit length of the operands.

For simplicity in Algorithm 27, we give the process when the number of bits to aggregate (denoted n) is a power of 2. In this algorithm, we took the liberty to denote $\mathsf{Add}^{\text{bits}}$ an addition algorithm which returns $m+1$ encrypted bits when the operands' bitsize is m (the last bit is the carry bit, so we just add $Z_m = R$ in Algorithm 11). Note that at line 4, the $n/2^i$ calls to $\mathsf{Add}^{\text{bits}}$ are made with inputs of length i, so that the cost is exactly $(2i-1)\mathsf{CSZ}$. Therefore the cost of the procedure is

$$\begin{aligned}\sum_{i=1}^{\log n} \frac{n}{2^i}(2i-1)\mathsf{CSZ} &\leq \sum_{i=1}^{\infty} \frac{2i-1}{2^i} n\mathsf{CSZ} \\ &\leq 3n\mathsf{CSZ}\end{aligned}$$

As for the communication cost, the process can be parallelized with a classical tree-based approach, since the addition is an associative operation.

Algorithm 27: $\mathsf{Aggreg}^{\text{bits}}$

Require: $B_1, \cdots, B_n$ such that for all i, $B_i = E(b_i)$ with $b_i \in \{0,1\}$.
Ensure: $S_0, \cdots, S_{\log n - 1}$ such that for all i, $S_i = E(s_i)$ with $s_i \in \{0,1\}$ and $\sum_{i=0}^{\log n - 1} s_i 2^i = \sum_i b_i$

```
1 B_{0,1}, ···, B_{0,n} = B_1, ···, B_n
2 for i = 1 to log n do
3     for k = 1 to n/2^i (in parallel) do
4         B_{i,k} = Add^bits(B_{i−1,2k−1}, B_{i−1,2k})
5 Return B_{log n,1}
```

Multiplication of integers in the bit-encoding.

In Algorithm 28, we detail the schoolbook algorithm for multiplication. This procedure is quite costly, as it requires about $3m^2$CSZ for the computation cost and transcript size, and $2m^2$CSZ for the communication cost, where m is the bitsize of the input integers.

Algorithm 28: $\mathtt{Mul}^{\text{bits}}$

Require: $(X_0, \cdots, X_{m_x-1}), (Y_0, \cdots, Y_{m_y-1})$, bitwise encryptions of x and y
Ensure: $Z_0, \cdots, Z_{m_x+m_y-1}$, bitwise encryption of xy

for $i \in [0, m_x - 1], j \in [0, m_y - 1]$ *(in parallel)* **do**
  $A_{i,j} = \mathtt{CSZ}(X_i, Y_j)$
$Z_0 = A_{0,0}$
$(T_0, \cdots, T_{m_y-1}) = (A_{0,1}, \cdots, A_{0,m_y-1}, E_0)$
for $i = 1$ *to* $m_x - 1$ **do**
  $(T_0, \cdots, T_{m_y}) = \mathtt{Add}^{\text{bits}}((T_0, \cdots, T_{m_y-1}), (A_{i,0}, \cdots, A_{i,m_y-1}))$
  $Z_i = T_0$
  for $j = 0$ *to* $m_y - 1$ **do**
    $T_j = T_{j+1}$
for $i = m_x$ *to* $m_x + m_y - 1$ **do**
  $Z_i = T_{i-m_x}$
Return $Z_0, \cdots, Z_{m_x+m_y-1}$

Schoolbook division algorithm.

For the Single Transferable Vote (see Section 6), we chose to represent fractions with a fixed number of binary places so that a fraction is encoded and encrypted as an integer. This allows to re-use most of the primitives from this section, while providing a certain degree of precision and generality. From the schoolbook division algorithm, we derive Algorithm 29, which takes as inputs bit-wise encryptions of x and y with $y > x$ and return the r first binary places of x/y. This algorithm could be generalised for any pair (x, y) (*i.e.* the condition $y > x$ is not necessary), but the restriction is useful in the special case of STV, and gives a simpler description.

Algorithm 29: $\mathtt{Div}^{\text{bits}}$

Require: $(X_0, \cdots, X_{m-1}), (Y_0, \cdots, Y_{m-1}), r$, bit-wise encryptions of integers $0 \le x < y$, and a precision r
Ensure: $Z_0, \cdots, Z_{r-1}$, encryptions of the first r binary places of x/y (in reverse order: z_0 is the least significant bit)

$A^{\text{bits}} = X^{\text{bits}}$
for $i = 0$ *to* $r - 1$ **do**
  $B^{\text{bits}}, R_i = \mathtt{SubLT}(A^{\text{bits}}, Y^{\text{bits}})$
  $A^{\text{bits}} = \mathtt{Select}^{\text{bits}}(B^{\text{bits}}, A^{\text{bits}}, R_i)$
  $Z_i = \mathtt{Not}(R_i)$
Return $Z_0, \cdots, Z_{r-1}$

D SINGLE CHOICE VOTING, ALGORITHMS FOR FINDING THE S LARGEST VALUES

In Section 3, we explained that the basic single choice voting or the more advanced list-voting à la D'Hondt reduce both to finding efficiently the s highest value in an array of encrypted fractions. In Section D.1, we recall the difficulty that arises in case of an equality and we propose a simple solution. This allows us to consider that all the values are distinct in Section D.2, where we give several options to find the s greatest values from a list of encrypted integers, which solves the case of basic single choice voting. In Section D.3, we discuss on how to represent the encrypted fractions in the context of D'Hondt voting, which completes the description of how we can perform the tally in the D'Hondt setting. This leads to Section D.4, where we explain how Figure 1 was obtained. The security of the protocol is addressed in Appendix I.3.

D.1 Breaking ties

Recall that selecting the candidates who have the s highest number of votes can lead to selecting more than s candidates in the case when two candidates have the same number of votes. Consequently, if we want a perfectly tally-hiding protocol which supports every corner case, we need a way to break the ties.

The idea is simply to force the scores of the candidates to be distinct, even if two candidates received the same number of voices. To do so, the election administrators must agree on an arbitrary ordering of the candidates, which allows to break ties. For instance, it can be decided that, in case of a tie, candidate i wins over candidates j if $i > j$. To put this into act, we simply modify the scores s_i of each candidate to turn it into $s'_i = 2^l s_i + i$, where $l = \lceil \log(k+1) \rceil$, assuming the candidates are labelled from 0 to $k-1$. This can be done with a very little extra cost, which comes from the fact that the number of bits increased in every procedure. Therefore, if m is the number of bits required to encode the scores (typically, m is logarithmic in the number of voters), the overall process is $\frac{m+\log k}{m}$ more expensive in terms of computations, while the impact on communication is about a factor $\frac{\log(m+\log k)}{\log m}$. In general, the impact is smaller than a factor 2 as the number of candidates is smaller than the number of voters. Note that the modification of the scores itself is basically free, and can be performed as follows.

- In natural encoding, $S'_i = (S_i)^{2^l} E_i$.
- In bit-encoding, $S'_i = i^{\text{bits}} || S_i$, where i^{bits} is a bit-encoding of i and $||$ stands for the concatenation.

Note that if the agreed ordering of the candidates has to be kept secret, the authorities can first run a reencryption mixnet to shuffle $E_0, \cdots, E_{k-1}$ (in either bit- or natural encoding) to obtain $(p_0, \cdots, p_{k-1})$. Afterwards, p_i can be used instead of E_i in natural encoding (resp. i^{bits} in bit-encoding).

From now on, we can assume that all the scores are distinct.

D.2 Various algorithm for the s largest values.

Algorithm	Exp per trustee	Comm. cost	Transcript size
`BubbleSort`	$\frac{1}{2}K^2\mathsf{LT} + \mathsf{Mixnet}(s)$	$\frac{1}{2}K^2\mathsf{LT} + \mathsf{Mixnet}(s)$	$\frac{1}{2}K^2\mathsf{LT} + \mathsf{Mixnet}(s)$
`NaiveSMax`	$\frac{1}{2}K^2\mathsf{LT} + K\mathsf{Dec}$	$2\mathsf{LT} + \mathsf{Dec}$	$\frac{1}{2}K^2\mathsf{LT} + K\mathsf{Dec}$
`s-Select`	$\frac{1}{2}s(2K-s)\mathsf{LT} + \mathsf{Mixnet}(s)$	$s \log K\mathsf{LT} + \mathsf{Mixnet}(s)$	$\frac{1}{2}s(2K-s)\mathsf{LT} + \mathsf{Mixnet}(s)$

Algorithm	Option	Exp per trustee	Comm. cost	Transcript size
`BubbleSort`	P,GTH	$\frac{1}{2}K^2(27m + 146\log m)a$	$\frac{1}{2}K^2(2R + 13B)\log m$	$K^2(14m + 25\log m)a$
`NaiveSMax`	P,GTH	$\frac{1}{2}K^2(27m + 146\log m)a$	$(4R + 26B)\log m$	$K^2(14m + 25\log m)a$
`s-Select`	EG,CLT	$87s(2K-s)ma$	$2s\log m \log KR$	$93s(2K-s)ma$

Figure 9: Leading terms of the cost of various s-max algorithms. We express the costs in terms of `Dec`, `LT` and `Mixnet`. The latter denotes the cost of a decryption Mixnet, while `LT` denotes the cost of any implementation of $\mathsf{LT}^{\text{bits}}$ or `GT` (see Figure 6). In the table at the bottom, we propose a few instantiations, for various choices of `LT`. In both tables, K is the number of operands, m is the bitsize of the operands, a is the number of trustees, R denotes a round of communication and B a broadcast.

The sorting approach.

A straightforward way to process is simply to sort the input list and return the indexes of the s first elements of the sorted list. This can be done using the elementary Bubble-Sort algorithm, which is straightforward to adapt in MPC. Note that this approach would leak the ordering of the selected candidates, while we may only want to reveal the unordered list of the selected candidates. In this case, a decryption mixnet can be used to reveal the indexes in some random order (for an instance of a UC-secure decryption mix-nets whose complexity is linear in the size of the list to shuffle, see [46]). The main drawback of this approach is that the bubble-sort algorithm is extremely inefficient, both in terms of communications and comparisons. Adapting a more efficient sorting algorithm in MPC is left to future work.

The naive approach.

Following [28], it is possible to return the indexes of the s best candidates (or list of candidates) using a quadratic number of comparisons. We call it the naive approach, but it could be efficient in practice due to the low communication cost. We recall it for completeness in Algorithm 30 which is a slightly modified version of the one in [28].

The s-selection approach.

Another natural way to proceed is to iteratively select the s largest values, just as a selection sort would do. Compared to the bubble sort, this approach would not require a quadratic number of comparisons. Moreover, finding the maximal value in a list is an operation which only requires a logarithmic number of rounds, so that so cost in communication is way lower. This can be done in MPC using Algorithm 18, which also finds the minimum value of the input list. Note that since this extra output is not relevant here, the cost of the primitive would actually be halved. Just as for the sorting approach, a decryption mixnet can be used to reveal the indexes of the candidates in some random order, rather than in the sorted order.

D.3 Representing fractions

The algorithm presented in Section D.2 can be directly applied to basic single choice voting. But when it comes to the D'Hondt method, we need to find the s largest values out of a list of fractions. A natural way to represent a fraction is to use a numerator and a denominator. This representation is consistent with our integer arithmetic toolbox and allows a straightforward adaptation of the algorithms from Section D.2: we simly have to change the comparison algorithm to allow it to handle fractions. Another advantage is that it is trivial to get a natural

Algorithm 30: `NaiveSMax`

Require: $X_0, \cdots, X_{n-1}$, encryptions of integers $x_0, \cdots, x_{n-1}$
Ensure: $i_0, \cdots, i_{s-1}$, indexes of the encryptions of the s largest integers

for $i = 0$ *to* $n-1$ *(in parallel)* **do**
  for $j = i+1$ *to* $n-1$ *(in parallel)* **do**
    $B_{i,j} = \mathtt{LT}(X_i, X_j)$
  $B_{i,i} = \mathtt{Enc}(0)$
  for $j = 0$ *to* $i-1$ *(in parallel)* **do**
    $B_{i,j} = \mathtt{Not}(B_{j,i})$
for $i = 0$ *to* $n-1$ *(in parallel)* **do**
  $S_i = \prod_{j=0}^{n-1} B_{i,j}$
  $g_i = \mathtt{Dec}(\mathtt{LT}(S_i, s))$
Return $\{i \mid g_i = 1\}$

representation of (c_i/w_j) from the score c_i of candidate i and the weight w_j. However, the comparisons become more expensive since they require two additional multiplications.

To mitigate this drawback, a straightforward approach is to precompute all the multiplications, so that the same multiplication is not computed multiple times. Indeed, for all i, j, i', j', $c_i/w_j < c_{i'}/w_{j'}$ if and only if $c_i w_{j'} < c_{i'} w_j$ so that we only have to compute all the $c_i w_j$. We refer to this representation as the implicit representation. This representation is strictly better than the natural representation in terms of computations and communications, but can only applied to the naive approach where the encrypted elements are never swapped. In Algorithm 31, we give an efficient way to precompute all the $c_i w_j$ in the specific case where $w_j = j$ for all j.

Algorithm 31: AddChain

Require:
- C^{bits}, a bit-wise encryption of some integers c of size m.
- s, a positive integer.

Ensure: ${C_1}^{\text{bits}}, \cdots, {C_s}^{\text{bits}}$, such that ${C_i}^{\text{bits}}$ is a bitwise encryption of iC for all i.

${C_1}^{\text{bits}} := C^{\text{bits}}$; ${C_2}^{\text{bits}} := E_0 || {C_1}^{\text{bits}}$
for $i = 2$ *to* $\lceil \log(s) \rceil$ **do**
  for $j = 1$ *to* 2^{i-2} *(in parallel)* **do**
    ${C_{2^{i-1}+2j-1}}^{\text{bits}} := \mathtt{Add}^{\text{bits}}({C_{2^{i-2}+j-1}}^{\text{bits}}, {C_{2^{i-2}+j}}^{\text{bits}})$
    ${C_{2^{i-1}+2j}}^{\text{bits}} := E_0 || {C_{2^{i-2}+j}}^{\text{bits}}$
Return ${C_1}^{\text{bits}}, \cdots, {C_s}^{\text{bits}}$

Finally, one can simply multiply every fraction by the least common multiple (lcm) and use an integer representation. The drawback is that the precomputation becomes more expansive and that the size of the integers to be compared would greatly increase, but it allows to use a more efficient algorithm than the naive one, such as `s-Select`. Since one of the operands of the multiplication is known, we can use a slightly optimized multiplication algorithm, such as Algorithm 32.

D.4 Summing-up the costs

We finally explain how to obtain the costs given in Figure 1 in Section 3. The first line is taken from [28] and is for the basic single choice setting. The second and third lines are our versions of the same tally function, except that we always return exactly s winners, using the technique from Section D.1, which adds $\log k$ additional bits to the size of the integers, hence the size m_1. The second line is in the Paillier setting, with the naive approach and the `GTH` comparison, and the third line is in the ElGamal setting, with the `s-Select` approach and the `CLT` comparison. Note that in the Paillier setting, the cost of verifying the ZKP of each ballot is taken into account, while in the ElGamal setting, the additional cost of the `Aggreg` procedure is predominant (see Section C.6). Also, the cost of the mixnet is proportional to sam_1 in terms of exponentiations, and consists of a round of communication. It is therefore negligible compared to the rest of the procedure.

The other two lines are for the D'Hondt tally function.

First, we give the most efficient trade-off in terms of communications. It simply consists of an adaptation of [28] in the D'Hondt setting. We fix the shortcoming in case of an equality using the technique from Section D.1, which adds $\log k$ bits to the bitsize of the integers. Finally, we use the implicit representation for the fraction, which adds an additional $\log s$ bits to the size of the integers in the case when $w_j = j$ for

Algorithm 32: `MulKnown`$^{\text{bits}}$

Require: $(X_0, \cdots, X_{m_x-1})$, $(y_0, \cdots, y_{m_y-1})$, bitwise encryptions of x, and a (public) bitwise representation of y

Ensure: $Z_0, \cdots, Z_{m_x+m_y-1}$, bitwise encryption of xy

for $i \in [0, m_x - 1]$, $j \in [0, m_y - 1]$ *(in parallel)* **do**
  $A_{i,j} = X_i^{y_j}$
$Z_0 = A_{0,0}$
$(T_0, \cdots, T_{m_y-2}) = (A_{0,1}, \cdots, A_{0,m_y-1})$
for $i = 1$ *to* $m_x - 1$ **do**
  $(T_0, \cdots, T_{m_y-1}) = \mathtt{Add}^{\text{bits}}((T_0, \cdots, T_{m_y-2}), (A_{i,0}, \cdots, A_{i,m_y-2}))$
  $T_{m_y} = \mathtt{CSZ}(T_{m_y-1}, A_{i,m_y-1})$
  $T_{m_y-1} = T_{m_y-1}A_{i,m_y-1}/T_{m_y}^2$
  $Z_i = T_0$ **for** $j = 0$ *to* $m_y - 1$ **do**
    $T_j = T_{j+1}$
for $i = m_x$ *to* $m_x + m_y - 1$ **do**
  $Z_i = T_{i-m_x}$
Return $Z_0, \cdots, Z_{m_x+m_y-1}$

- Let pk be the public encryption key and v the chosen voting option.
- Encode v as a vector of k bits, where k is the number of candidates. The ith bit is set if and only if i is the choice of the voter.
- Encrypt the vector into $B_1, \cdots, B_k$, using pk.
- For all i, produce a ZKP $\pi_i^{0/1}$ that B_i is an encryption of 0 or 1.
- Produce a ZKP $\pi_0^{0/1}$ that the product $B_1 \cdots B_k$ is an encryption of 0 or 1.
- Return $(B_i)_{1 \leq i \leq k}$, $(\pi_i^{0/1})_{0 \leq i \leq k}$.

Figure 10: `vote` procedure for the D'Hondt method

all j. Note that in the Paillier setting, multiplying by a public value is performed with a single exponentiation which is extremely cheap compared to the remaining of the procedure. Hence, cost is the same as in the basic single choice setting, except that k is replaced by ks and that m is replaced by m_1.

Second, we give the cost of the same algorithm, but in the ElGamal setting. To get the bitwise encryptions of the number of votes for each candidate, we need to compute k `Aggreg` (Algorithm 27 in parallel first. Then we also need to compute k `AddChain` (Algorithm 31 to compute the implicit representation of the fractions. Finally, we can use the naive algorithm to find the s largest values. Recall that the naive algorithm (see Algorithm 30) consists on comparing each value to all the other values and to check whether the number of "losses" is lower than s. To perform this comparison on encrypted data without revealing the number of losses, we need a bitwise encryption for both operands, which requires an additional call to `Aggreg`. The cost of this procedure would not alter the leading term of the number of exponentiations, but affects the number of communications. Note that the computational cost of `AddChain` is about $(sm + s\log s)$`CGate`, which is negligible compared to the remaining of the process. The communication cost, however, is about $m \log s$ rounds, which may not be negligible.

Finally, the last line uses the `s-Select` method. As before, it requires first to obtain the bitwise encryptions of the number c_i of votes obtained by each candidate i, which is done with `Aggreg`. Afterwards, for all i, j, we compute a bitwise encryption of $d_{i,j} = c_i \frac{\mathrm{lcm}(2,\cdots,s)}{j}$ using the `MulKnown` protocol (Algorithm 32). Note that the bit length of $\mathrm{lcm}(2, \cdots, s)$ is s', which is approximately $s\log(e)$, where $e = \exp(1)$. Once the integer representations are computed, we can use the `s-Select` procedure. Adding up the complexity of the three protocols yields the claimed complexity.

D.5 D'Hondt voting, the bottom-line

To improve readability, we give again the details that are necessary to use our tally-hiding protocol inside of a voting protocol. First, to submit a ballot, a voter can simply use the `vote` procedure, which is summed up in Figure 10. This allows the voter to choose up to one candidate among the k options (choosing no candidate is possible and corresponds to a blank vote). Finally, to proceed with the tally, the authorities use the protocol $P_{\mathtt{DHondt}}$, defined in Algorithm 33.

Algorithm 33: DHondt

Require:

- n, the number of ballots,
- k, the number of candidates,
- s, the number of seats,
- $\boldsymbol{B}$, the list of n encrypted ballots such that, for all i, j, $B_{i,j}$ is an encryption of 0 or 1 and is 1 for at most one out of the k candidates

Ensure: $s_1, \cdots, s_k$, the number of seats given to each candidate

```
for j = 1 to k (in parallel) do
    C_j := Aggreg(B_{1,j}, ..., B_{n,j})
    C_{1,j}, ..., C_{s,j} := AddChain(C_j, s) for i = 1 to s do
        C_{i,j} := j^bits || C_{i,j}
for all (i, j, i', j') s.t. (i, j) < (i', j') (in parallel) do
    D_{i,j,i',j'} := LT(C_{i,j'}, C_{i',j})
    D_{i',j',i,j} := Not(D_{i,j,i',j'})
for all (i, j) (in parallel) do
    L_{i,j} := Aggreg((D_{i,j,i',j'})_{i',j'})
    S_{i,j} := LT(L_{i,j}, s^bits)
for all j (in parallel) do
    S_j := ∏_{i=1}^{s} S_{i,j}  s_j := Dec(S_j)
Return s_1, ..., s_k
```

E MAJORITY JUDGEMENT

In this appendix, we will give the details of what is sketched in Section 4: we start with a precise definition of Majority Judgement (MJ), then we discuss the contribution of citeCPST-Esorics08 and explain why it is not acceptable. We then present our algorithm for computing the winners and give a complete proof of its correctness. Finally we explain how to adapt it for MPC for both Paillier and ElGamal settings. The security aspects are addressed in Appendix I.3.

E.1 Definition

In a MJ protocol, there are k candidates and a set of d grades, which is totally ordered. For instance, the set could be {Excellent,Good,Medium,Bad,Reject}. For the computations, we represent grades with integers and the tradition in MJ is to use a reversed ordering (*i.e.* 1 is a better / higher grade than 2). Each voter has to grade each candidate with a single grade. Hence, if n is the number of voters (who did not abstain or vote blank), each candidate has a list of n grades. For simplicity, we assume that the lists are sorted in decreasing order (highest grades first). Thus, we consider that each candidate has a sorted n-tuple. Note that two n-tuples are equal if and only if the candidates received exactly the same number of each grade. Given a sorted n-tuple $u_1, \cdots, u_n$ the *median* of u is simply $\mathsf{med}(u) = u_{\lceil n/2 \rceil}$. We denote $\hat{u}$ the $(n-1)$-tuple $u_1, \cdots, u_{\lceil n/2 \rceil - 1}, u_{\lceil n/2 \rceil + 1}, \cdots, u_n$; that is, the tuple u in which the median element has been removed. Finally, we define the $\leq_{maj}$ relation as follows, where $<$ stands for the grade-wise comparison (which is the opposite of the natural comparison of integers).

Definition E.1 (The relation $\leq_{maj}$). Let u and v be grade n-tuples sorted in decreasing order. If $n = 1$, $u <_{maj} v$ if $u_1 < v_1$. Else, $u <_{maj} v$ if one of the following conditions holds:

- $\mathsf{med}(u) < \mathsf{med}(v)$,
- $\mathsf{med}(u) = \mathsf{med}(v)$ and $\hat{u} <_{maj} \hat{v}$.

Finally, $u \leq_{maj} v$ if $u = v$ or $u <_{maj} v$.

It is straightforward to show that $\leq_{maj}$ is a total order. The majority judgement declares as winner any candidate whose grades form a maximal n-tuple (once sorted) according to $\leq_{maj}$.

E.2 The approach of [15]

While the algorithm to determine the MJ winner(s) is simple, its naive implementation yields a complexity that depends on the number of voters, which could be very costly when done in MPC. Hence, the authors of [15] propose an MPC implementation of a simplification of the MJ algorithm, where whenever two candidates have the same median, only their number of grades higher and smaller than the median are compared. It has been shown that this technique is sound [7]: if a winner can be determined with this approach, it is indeed a MJ winner. However, it may also fail to conclude.

Number of voters	10	100	1000
uniform distribution over 5 candidates	0.384	0.220	0.080
political distribution [7]	N/A	0.001	N/A

Figure 11: Estimated probability that the algorithm of [15] fails to determine the MJ winner(s).

An experiment run in [7] on real ballots of a political election with 12 candidates is reassuring: the simplified approach fails only with probability 0,001 for an election of 100 voters. However, this is due to the fact that in this political election, there was a high correlation between candidates (if a voter likes a candidate, he is likely to also like other candidates from similar political parties).

In case the number of candidates is smaller and if the distribution of votes is uniform, then the probability of failure raises up to 22%, as shown in Figure 11. In any case, the approach of [15] leaks more information about the ballots than just the result, with non negligible probability, since it reveals whether the result can be determined with the simplified algorithm.

E.3 Our simplified algorithm for MJ

Algorithm 34: Majority Judgement

Require: a the aggregated matrix, d the number of grades, n the number of voters
Ensure: C the set of MJ winner(s)

Let $m = \max\{m_i \mid m_i \text{ is the median of candidate } i\}$
Let C be the set of candidates with m as median grade.
Let $I^- = 1$ and $I^+ = 1$ be counters.
Let $s = 1$.
for $i \in C$ **do**
  $p_i = \sum_{j=1}^{m-1} a_{i,j}$, $q_i = \sum_{j=m+1}^{d} a_{i,j}$,
  $m_i^- = \lfloor \frac{n}{2} \rfloor - p_i$, $m_i^+ = \lfloor \frac{n}{2} \rfloor - q_i$
while $(\mid C \mid > 1) \wedge (s \neq 0)$ **do**
  for $i \in C$ **do**
    if $m_i^- \leq m_i^+$ **then**
      $s_i = p_i$
    else
      $s_i = -q_i$
  $s = \max\{s_i \mid i \in C\}$
  $C = \{i \in C \mid s_i = s\}$.
  if $s \geq 0$ **then**
    for $i \in C$ **do**
      $m_i^+ = m_i^+ - m_i^-$, $m_i^- = a_{i,m-I^-}$
      $p_i = p_i - a_{i,m-I^-}$
    $I^- = I^- + 1$
  else
    for $i \in C$ **do**
      $m_i^- = m_i^- - m_i^+$, $m_i^+ = a_{i,m+I^+}$
      $q_i = q_i - a_{i,m+I^+}$
    $I^+ = I^+ + 1$
Return C.

First, we give Algorithm 34, our simplified algorithm for Majority Judgment. It takes as input the *aggregated matrix a* such that, for all candidate i and grade j, $a_{i,j}$ is the number of j received by i. It outputs the set of the winners according to the Majority Judgement. To prove its correctness, we first give Definition E.2. From this definition and Definition E.1, it is straightforward to show that $\leq_{maj}$ is the lexicographic order for the median sequences. Hence, it is important to describe the behavior of the median sequence, which is done in Lemma E.3.

Definition E.2 (The median sequence). The median sequence of a sorted n-tuple u, denoted $m(u)$ is the sequence formed by $\mathrm{med}(u)$ followed by $m(\hat{u})$.

Lemma E.3. *Let u a sorted n-tuple. The k^{th} element of the median sequence of u is the element of index $m + (-1)^{k+n}\lfloor k/2 \rfloor$, where $m = \lceil \frac{n}{2} \rceil$.*

Proof. We distinguish the cases where n is even or odd and give a recurrence in k.

Case 1: n is even. The first element of the median sequence is u_m by definition. Let $k \geq 1$. Suppose that for $i \in [1, k]$, the i^{th} element of the median sequence is $u_{m+(-1)^i \lfloor i/2 \rfloor}$. By definition, the $(k+1)^{th}$ element of the median sequence is the element of index $\left\lceil \frac{n-k}{2} \right\rceil$ of some $(n-k)$-tuple, obtained by removing the first k elements of the median sequence of u.

If k is even, by recurrence hypothesis, the removed elements have indexes $m, m+1, m-1, \cdots, m-(k/2-1), m+k/2$ thus the remaining elements are

$$(u_1, \cdots, u_{m-k/2}, u_{m+k/2+1}, \cdots, u_n).$$

As n and k are even, $\left\lceil \frac{n-k}{2} \right\rceil = m - k/2$. Therefore, the $(k+1)^{th}$ element of the median sequence is $u_{m-k/2}$, and since k is even, $m - k/2 = m + (-1)^{k+1} \left\lfloor \frac{k+1}{2} \right\rfloor$.

If k is odd, by recurrence hypothesis, the removed elements have indexes $m, m+1, m-1, \cdots, m+(k-1)/2, m-(k-1)/2$ so the remaining elements are

$$(u_1, \cdots, u_{m-(k+1)/2}, u_{m+(k+1)/2}, \cdots, u_n).$$

Since n is even while k odd, $\left\lceil \frac{n-k}{2} \right\rceil = m - (k-1)/2$, so the $(k+1)^{th}$ element of the median sequence is the one following $u_{m-(k+1)/2}$ in the above list, namely $u_{m+(k+1)/2}$, with $m + (k+1)/2 = m + (-1)^{k+1} \left\lfloor \frac{k+1}{2} \right\rfloor$.

Case 2: n is odd. The first element of the median sequence is u_m by definition. Let $k \geq 1$. Suppose that for $i \in [1, k]$, the i^{th} element of the median sequence is $u_{m-(-1)^i \lfloor i/2 \rfloor}$. By definition, the $(k+1)^{th}$ element of the median sequence is the element of index $\left\lceil \frac{n-k}{2} \right\rceil$ of some $(n-k)$-tuple, obtained by removing the first k elements of the median sequence of u.

If k is even, by recurrence hypothesis, the removed elements have indexes $m, m-1, m+1, \cdots, m+(k/2-1), m-k/2$ so the remaining elements are

$$(u_1, \cdots, u_{m-k/2-1}, u_{m+k/2}, \cdots, u_n).$$

As n is odd and k even, $\left\lceil \frac{n-k}{2} \right\rceil = m - k/2$. Therefore the $(k+1)^{th}$ element of the median sequence is the one following $u_{m-k/2-1}$ in the above list, namely $u_{m+k/2}$ with $m + k/2 = m - (-1)^{k+1} \left\lfloor \frac{k+1}{2} \right\rfloor$.

If k is odd, by recurrence hypothesis, the removed elements have indexes $m, m-1, m+1, \cdots, m-(k-1)/2, m+(k-1)/2$ so the remaining elements are

$$(u_1, \cdots, u_{m-(k+1)/2}, u_{m+(k+1)/2}, \cdots, u_n).$$

As n and k are odds, $\left\lceil \frac{n-k}{2} \right\rceil = m - (k+1)/2$. Hence the $(k+1)^{th}$ element of the median sequence is $u_{m-(k+1)/2}$, with $m - (k+1)/2 = m - (-1)^{k+1} \left\lfloor \frac{k+1}{2} \right\rfloor$.

□

In order to prove the correctness of Algorithm 34, we exhibit the following loop invariants, where a sum indexed with the empty set is 0 and $g_{i,1}, \cdots, g_{i,n}$ denote the list of grades received by candidate i, sorted in decreasing order. Note that m is used to denote the best median (line 1), and not $\lceil \frac{n}{2} \rceil$ as in the previous lemma.

Lemma E.4. *In Algorithm 34, the following loop invariants hold at the beginning of the loop (line 8) and at the end of the loop (line 25).*

(1) For all $i \in C$, $p_i + m_i^- = m_i^+ + q_i$, and this value is the same for all i.
(2) For all $i \in C$, $m_i^+ \geq 0$ and $m_i^- \geq 0$.
(3) For all $i \in C$, $p_i = \sum_{j=1}^{m-I^-} a_{i,j}$. Hence $p_i \geq 0$.
(4) For all $i \in C$, $q_i = \sum_{j=m+I^+}^{d} a_{i,j}$. Hence, $q_i \geq 0$.
(5) Let $L + p_i + m_i^- + m_i^+ + q_i$. The $n - L$ first elements of the median sequence are identical for all $i \in C$.
(6) For all $i \in C$, for all $j \in [1, m_i^-]$, $g_{i,p_i+j} = m - I^- + 1$ and, for all $j \in [1, m_i^+]$, $g_{i,n-q_i-j+1} = m + I^+ - 1$.
(7) C contains all the MJ winners.

Proof. **Initialization.** First of all, we verify that the loop invariants are true after line 7.

Invariants 1 to 4:

We have $p_i + m_i^- = \lfloor n/2 \rfloor = m_i^+ + q_i$.

Moreover p_i is the number of grades strictly greater than the median, so by definition of the median, $p_i \leq \lfloor n/2 \rfloor$ hence $m_i^- = \lfloor n/2 \rfloor - p_i \geq 0$. Similarly, q_i is the number of grades strictly worse than the median, so by definition of the median, $q_i \leq \lfloor n/2 \rfloor$ hence $m_i^+ = \lfloor n/2 \rfloor - q_i \geq 0$.

Finally, Equalities 3 and 4 are true with $I^- = I^+ = 1$.

Invariant 5:

Initially, $L = p_i + m_i^- + m_i^+ + q_i = 2\lfloor n/2 \rfloor$ so if n is even, $n - L = 0$. Else, $n - L = 1$. As the first element of the median sequence is the median, the $n - L$ first elements are the same for all candidates in C after line 7.

Invariant 6:

After line 7, p_i is the number of grades strictly greater than the median for candidate i so, for all $j \geq 1$, $g_{i,p_i+j} \geq m$. Moreover m_i^- is lower than the number of grades equal to the median received by i. So for all $j \leq m_i^-$, $g_{i,p_i+j} \leq m$. Hence, for all $j \in [1, m_i^-]$, $g_{i,p_i+j} = m$. Similarly, for all $j \in [1, m_i^+]$, $g_{i,n-q_i-j+1} = m$.

Invariant 7:

After line 7, C contains the candidates who have the best median, thus contains the winners.

Heredity. Assume that the loop invariants are verified at the beginning of the loop, we show that they are preserved at the end of the loop.

We first show the following result, which is a consequence of loop invariants 1 to 4.

Sub-lemma. For all candidates i, $s_i \geq 0$ if and only if $m_i^- \leq m_i^+$.

Let i be a candidate. Suppose $s_i \geq 0$ and $m_i^- > m_i^+$. Then $0 \leq s_i = -q_i \leq 0$ so $q_i = 0$ and as $p_i + m_i^- = m_i^+ + q_i$, we have $p_i + m_i^- = m_i^+$, which contradicts $p_i \geq 0$. Conversely, if $m_i^- \leq m_i^+$, $s_i = p_i \geq 0$.

To show that the loop invariants are preserved, we denote C_1 the set C at the beginning of the loop and C_2 the set C at the end of the loop. Let $i \in C_2$. Let $i \in C_2$, then $i \in C_1$ so the loop invariants hold at the beginning of the loop, for all $i \in C_2$. We denote p_1 the value of p_i at the beginning of the loop and p_2 at the end, and the same for all other variable m_i^-, m_i^+, q_i, I^-, I^+ and L.

Invariants 1 to 4: Let $s = \max\{s_i \mid i \in C\}$. $C_2 = \{i \mid s_i = s\}$.

If $s \geq 0$, then $s_i = s \geq 0$ so $m_1^- \leq m_1^+$ by the sub-lemma. Hence $m_2^+ = m_1^+ - m_1^- \geq 0$, $m_2^- = a_{i,m-I_1^-} \geq 0$.

In addition, $p_2 = p_1 - a_{i,m-I_1^-}$ and $q_2 = q_1$. Therefore $p_2 + m_2^- = p_1 = s_i = s$, which is the same for all i. Moreover $m_2^+ + q_2 = m_1^+ - m_1^- + q_1 = p_1 + m_1^- - m_1^- = p_1 = S$.

Finally, line 20 together with line 21 and loop invariant 3 give $p_2 = \sum_{j=1}^{m-I_2^-} a_{i,j}$, which shows that invariant 3 is preserved. (Invariant 4 is also preserved because $q_2 = q_1$ and $I_2^+ = I_1^+$.)

If $s < 0$, then $s_i = s < 0$ so $m_1^- > m_1^+$ by the sub-lemma. Hence $m_2^- = m_1^- - m_1^+ \geq 0$, $m_2^+ = a_{i,m+I_1^+} \geq 0$, $q_2 = q_1 - a_{i,m+I_1^+}$ et $p_2 = p_1$. So $m_2^+ + q_2 = q_1 = -S_i = -S$, which is the same for all i. In addition $p_2 + m_2^- = p_1 + m_1^- - m_1^+ = m_1^+ + q_1 - m_1^+ = q_1 = -S$. Finally line 26 together with line 27 and loop invariant 4 give $q_2 = \sum_{j=m+I_2^+}^{c} a_{i,j}$, so that invariant 4 is preserved. (Invariant 3 is also preserved because $p_2 = p_1$ and $I_2^- = I_1^-$.)

Invariant 5:

If $s \geq 0$, $m_1^- \leq m_1^+$. Consequently, $p_1 = s_i = s$ and since $p_1 + m_1^-$ is the same for all i, we deduce that m_1^- is the same for all i. In addition we have $p_2 + m_2^- = p_1$ (lines 19 and 20), $m_2^+ = m_1^+ - m_1^-$ (line 18) and $q_2 = q_1$, so

$$\begin{aligned} L_2 &= p_2 + m_2^- + m_2^+ + q_2 \\ &= p_1 + m_1^+ - m_1^- + q_1 \\ &= p_1 + m_1^- + m_1^+ + q_1 - 2m_1^- = L_1 - 2m_1^-, \end{aligned}$$

and since the $n - L_1$ first elements of the median sequence are the same for all candidates in C_1, we only have to show that the $2m_1^-$ next elements are the same for all candidates in C_2. For this purpose, we remark that loop invariant 1 implies that L_1 is even and we suppose $m_1^- > 0$. (If $m_1^- = 0$, our job is already done.)

By Lemma E.3, the elements of indexes $n - L_1 + 1, \cdots, n - L_1 + 2m_1^-$ of the median sequence are the elements

$$g_{i,\lceil n/2 \rceil + (-1)^{2n-L_1+1} \lfloor (n-L_1+1)/2 \rfloor}, g_{i,\lceil n/2 \rceil + (-1)^{2n-L_1+2} \lfloor (n-L_1+2)/2 \rfloor}, \cdots, g_{i,\lceil n/2 \rceil + (-1)^{2n-L_1+2m_1^-} \lfloor (n-L_1+2m_1^-)/2 \rfloor};$$

which are also

$$g_{i,\lceil n/2 \rceil - \lfloor (n+1)/2 \rfloor + L_1/2}, g_{i,\lceil n/2 \rceil + \lfloor n/2 \rfloor - L_1/2+1}, \cdots, g_{i,\lceil n/2 \rceil - \lfloor (n-1)/2 \rfloor + L_1/2 - m_1^-}, g_{i,\lceil n/2 \rceil + \lfloor n/2 \rfloor - L_1/2 + m_1^-}.$$

But $L_1 = p_1 + m_1^- + m_1^+ + q_1$ so, by invariant 1, $L_1/2 = p_1 + m_1^- = m_1^+ + q_1$. Since $\lceil n/2 \rceil = \lfloor (n+1)/2 \rfloor$ and $\lceil n/2 \rceil + \lfloor n/2 \rfloor = n$ for all n, we can rewrite them as

$$g_{i,p_1+m_1^-}, g_{i,n-q_1-m_1^++1}, \cdots, g_{i,p_1+1}, g_{i,n-q_1-m_1^++m_1^-}.$$

In what follow, we prove that for all $j \in [1, m_i^-]$, $g_{i,n-q_1-m_1^++j} = m + I_1^+ - 1$. Indeed, $n - q_1 - m_1^+ + j = n - q_1 - (m_1^+ - j + 1) + 1$ and since $m_1^+ \geq m_1^- > 0$, $m_1^+ - j + 1 \in [1, m_1^+]$ for all $j \in [1, m_1^-]$, which allows to prove our claim by invariant 6.

In addition, $g_{i,p_1+j} = m - I_1^- + 1$ for all $j \in [1, m_1^-]$ by invariant 6, so the elements listed above are equal to $m - I_1^- + 1, m + I_1^+ - 1, \cdots, m - I_1^- + 1, m + I_1^+ - 1$ and therefore are the same for all $i \in C_2$, which shows that invariant 5 is preserved.

If $s < 0$, $m_1^- > m_1^+$. Consequently, $q_1 = -s_i = -s$ and since $m_1^+ + q_1$ is the same for all i, so is m_1^+. Moreover $m_2^+ + q_2 = q_1$ (lines 25 and 26), $m_2^- = m_1^- - m_1^+$ (line 24) and $p_2 = p_1$ so

$$\begin{aligned} L_2 &= p_2 + m_2^- + m_2^+ + q_2 \\ &= p_1 + m_1^- - m_1^+ + q_1 \\ &= p_1 + m_1^- + m_1^+ + q_1 - 2m_1^+ = L_1 - 2m_1^+, \end{aligned}$$

and since the $n - L_1$ first elements of the median sequence are the same for all candidates in C_1, we only have to show that the $2m_1^+$ next elements are the same for all candidates in C_2. For this purpose, we remark that invariant 1 implies that L_1 is even and we suppose that $m_1^+ > 0$. (If $m_1^+ = 0$, our job is done.)

By Lemma E.3, the elements of indexes $n - L_1 + 1, \cdots, n - L_1 + 2m_1^+$ of the median sequence are

$$g_{i,\lceil n/2\rceil+(-1)^{2n-L_1+1}\lfloor(n-L_1+1)/2\rfloor}, g_{i,\lceil n/2\rceil+(-1)^{2n-L_1+2}\lfloor(n-L_1+2)/2\rfloor}, \cdots, g_{i,\lceil n/2\rceil+(-1)^{2n-L_1+2m_1^+}\lfloor(n-L_1+2m_1^+)/2\rfloor};$$

which are also

$$g_{i,\lceil n/2\rceil-\lfloor(n+1)/2\rfloor+L_1/2}, g_{i,\lceil n/2\rceil+\lfloor n/2\rfloor-L_1/2+1}, \cdots, g_{i,\lceil n/2\rceil-\lfloor(n-1)/2\rfloor+L_1/2-m_1^+}, g_{i,\lceil n/2\rceil+\lfloor n/2\rfloor-L_1/2+m_1^+}.$$

But $L_1 = p_1 + m_1^- + m_1^+ + q_1$ so, by invariant 1, $L_1/2 = p_1 + m_1^- = m_1^+ + q_1$. Since $\lceil n/2\rceil = \lfloor(n+1)/2\rfloor$ et $\lceil n/2\rceil + \lfloor n/2\rfloor = n$ for all n, we can rewrite them as

$$g_{i,p_1+m_1^-}, g_{n-q_1-m_1^++1}, \cdots, g_{i,p_1+m_1^--m_1^++1}, g_{i,n-q_1}.$$

We now show that for all $j \in [1, m_i^+]$, $g_{i,p_1+m_1^--j+1} = m - I_1^- + 1$. Indeed, $p_1 + m_1^- - j + 1 = p_1 + (m_1^- - j + 1)$ and since $m_1^- > m_1^+ > 0$, $(m_1^- - j + 1) \in [1, m_1^-]$ for all $j \in [1, m_1^+]$, which allows to prove our claim by invariant 6.

In addition, $g_{i,n-q_1-j+1} = m + I_1^+ - 1$ for all $j \in [1, m_1^+]$ by invariant 6, so the elements listed above are equal to $m - I_1^- + 1, m + I_1^+ - 1, \cdots, m - I_1^- + 1, m + I_1^+ - 1$ and therefore are the same for all $i \in C_2$, which shows that invariant 5 is preserved.

Invariant 6:

If $s \geq 0$, $m_1^- \leq m_1^+$ so $p_2 = p_1 - a_{i,m-I_1^-}$ and $m_2^- = a_{i,m-I_1^-}$. But $p_1 = \sum_{j=1}^{m-I_1^-} a_{i,j}$, which is exactly the number of grades strictly greater than $m - I_1^- + 1$ received by i so by definition of $a_{i,m-I_1^-}$, p_2 is the number of grades strictly greater than $m - I_1^-$. Therefore g_{i,p_2+1} is lower than $m - I_1^-$ and as there are $a_{i,m-I_1^-} = m_2^-$ grades equal to $m - I_1^-$, we deduce that $g_{i,p_2+j} = m - I_1^- = m - (I^- + 1) + 1 = m - I_2^- + 1$ for all $j \in [1, m_2^-]$. In addition, for all $j \in [1, m_1^+]$, $g_{i,n-q_1-j+1} = m + I_1^+ - 1$ so, *a fortiori*, for all $j \in [1, m_1^+ - m_1^-]$, $g_{i,n-q_1-j+1} = m + I_2^+ - 1$.

If $s < 0$, $m_1^- > m_1^+$ so $q_2 = q_1 - a_{i,m+I_1^+}$ and $m_2^+ = a_{i,m+I_1^+}$. But $q_1 = \sum_{j=m+I_1^+}^{c} a_{i,j}$, which is exactly the number of grades strictly worse than $m + I_1^+ - 1$ so by definition of $a_{i,m+I_1^+}$, q_2 is the number of grades strictly worse than $m + I_1^+$. Therefore $g_{i,n-q_2}$ is greater than $m + I_1^+$ and as there are $a_{i,m+I_1^+} = m_2^+$ grades equal to $m + I_1^+$, we deduce that $g_{i,n-q_2-j+1} = m + I_1^+ = m + (I^+ + 1) - 1 = m + I_2^+ - 1$ for all $j \in [1, m_2^+]$. In addition, for all $j \in [1, m_1^-]$, $g_{i,p_1+j} = m - I_1^- + 1$ so, *a fortiori*, for all $j \in [1, m_1^+ - m_1^-]$, $g_{i,p_1+j} = m - I_2^- + 1$.

Invariant 7:

Let $b \in C_2$, (namely $b \in C_1$ such that $s_b = s$). We show that for all $a \in C_1 \backslash C_2$, (namely for all $a \in C_1$ such that $s_a < s$), $a <_{maj} b$.

Positive case. Suppose that $s \geq 0$. Let $a \in C_1$ such that $s_a < s$.

Positive-negative case. We first assume that $s_a < 0$. Therefore $s_a < 0 \leq s = s_b$. By the sub-lemma, we have $m_a^- > m_a^+$ and $m_b^- \leq m_b^+$.

Suppose that $m_a^+ < m_b^-$. With the same reasoning as in the proof of invariant 6, we show that the elements of indexes 1 to $n - L + 2m_a^+$ of the median sequence of a and b are the same. Since $m_a^- > m_a^+$, by Lemma E.3 and loop invariant 1 and 6, the $n - L + 2m_a^+ + 1$ th elements of the median sequence of a and b are respectively

$$\begin{aligned} &g_{a,p_a+m_a^--m_a^+} = m - I^- + 1 \text{ and} \\ &g_{b,p_a+m_a^--m_a^+} = g_{b,p_b+m_b^--m_a^+} = m - I^- + 1. \end{aligned}$$

However, the $n - L + 2m_a^+ + 2$ th element of the median sequence of a is

$$g_{a,n-q_a+1} < g_{a,n-q_a} = m + I^+ - 1,$$

while b's is

$$g_{b,n-q_a-m_a^++m_a^++1} = g_{b,n-q_b-(m_b^+-m_a^+)+1} = m + I^+ - 1.$$

Therefore $b >_{maj} a$.

Now suppose that $m_a^+ \geq m_b^-$. As above, the $n - L + 2m_b^-$ first elements of the median sequence of a and b are the same. The elements of index $n - L + 2m_b^- + 1$ are respectively

$$g_{a,p_a+m_a^- - m_b^-} = g_{a,p_a+(m_a^- - m_a^+)+(m_a^+ - m_b^-)} = m - I^- + 1^- \text{ and}$$
$$g_{b,p_a+m_a^- - m_b^-} = g_{b,p_b} > m - I^- + 1.$$

Therefore $b >_{maj} a$.

Positive-positive case. Now suppose that $0 \leq s_a$. By the sub-lemma, $m_b^- \leq m_b^+$, $m_a^- \leq m_a^+$. Consequently $s_a = p_a$ and $s_b = p_b$ and since $s_a < s_b$, by invariant 1, we have $m_a^- > m_b^-$. Then again, we deduce that the $n - L + 2m_b^-$ first elements of the median sequence are the same and that b wins over a thanks to the next element.

Negative case. Finally, suppose that $s < 0$. Then $s_a < s_b = s < 0$ so, by the sub-lemma, $m_a^- > m_a^+$ and $m_b^- > m_b^+$. Consequently $s_a = -q_a$ and $s_b = -q_b$ and since $s_a < s_b$, by invariant 1, we have $m_b^+ > m_a^+$. Then again, we deduce that the $n - L + 2m_a^+$ first elements of the median sequence are the same. In addition $m_a^- > m_a^+$, so by Lemma E.3 and invariants 1 and 6, the $n - L + 2m_a^+ + 1$ th elements of the median sequence of a and b are

$$g_{a,p_a+m_a^- - m_a^+} = m - I^- + 1 \text{ and}$$
$$g_{b,p_a+m_a^- - m_a^+} = g_{b,p_b+m_b^- - m_a^+} = m - I^- + 1.$$

However, the $n - L + 2m_a^+ + 2$ th element for a is

$$g_{a,n-q_a+1} > g_{a,n-q_a} = m + I^+ - 1,$$

while b's is

$$g_{b,n-q_a-m_a^+ + m_a^+ + 1} = g_{b,n-q_b-(m_b^+ - m_a^+)+1} = m + I^+ - 1.$$

Therefore $b >_{maj} a$. □

Once the loop invariants are established, it is straightforward to show the correctness of our algorithm (Theorem E.5).

Theorem E.5. *Algorithm 34 returns the set of maxima according to $\leq_{maj}$ in $O(kd)$ comparisons between grades.*

Proof. **Complexity.** By Lemma E.4, $p_i = \sum_{j=1}^{m-I^-} a_{i,j}$ and $q_i = \sum_{j=m+I^+}^{c} a_{i,j}$. But at each iteration, we subtract $a_{i,m-I^-}$ to p_i or $a_{i,m+I^+}$ to q_i so there cannot be more than d iterations before both are equal to 0. When $p_i = q_i = 0$ for all i, $s = 0$, which terminates the loop. Hence the Algorithm terminates en $O(kd)$ comparisons.

Correctness. If the algorithm terminates because $|C| = 1$, C contains only one element and since C contains the winners, C is the set of winners. Otherwise, $s = 0$. Recall that s is the maximum of s_i and let i such that $s_i = s$. If $m_i^- > m_i^+$, we have $s_i = -q_i$ thus $q_i = 0$, which contradicts $p_i + m_i^- = m_i^+ + q_i$ and $p_i \geq 0$ so $m_i^- \leq m_i^+$ and $p_i = s_i = s = 0$. But $m_i^- \leq m_i^+$ and $p_i + m_i^- = m_i^+ + q_i$. Since $q_i \geq 0$, $q_i = 0$ thus $m_i^- = m_i^+$. Hence, by invariants 6 and 7, each candidate in C are equal with respect to $\leq_{maj}$. Since C contains the winners, C is the set of winners. □

E.4 An adaptation in MPC in the Paillier setting

In this section, we show how to adapt Algorithm 34 in MPC in the Paillier setting. Since we only focus on the tallying phase and since obtaining (an element-wise encryption of) the aggregated matrix from the ballots is easy in the Paillier setting, we consider that (an element-wise encryption of) the aggregated matrix is available. We first rewrite the algorithm into Algorithm 37 and prove that the new algorithm is equivalent to Algorithm 34. Using the building blocks from Section 2.1, it is easy to implement Algorithm 37 in MPC (see Algorithm 41).

We first provide Algorithm 35 which returns the grade vector as defined in [15]. The grade vector is a (term-by-term) encryption of g such that $g_j = 1$ if j is strictly greater than the best median m, and $g_j = 0$ otherwise. It will be useful to initialize p_i, m_i^-, m_i^+, q_i, $m - I^-$ and $m + I^+$.

The idea of this algorithm is that, for all candidate i and grade j, j is strictly greater than the best median if and only if the number of grades greater than j is strictly lower than half the number of grades. This translates into the formula $2\sum_{l=1}^{j} a_{i,l} < n = \sum_{l=1}^{d} a_{i,l}$, which allows to compute $c_{i,j}$ for all (i, j), where $c_{i,j} = 1$ if j is strictly greater than i's median. To deduce the grade vector, we compute the logical conjunction column by column.

Once the grade vector is computed, we can initialize p_i, m_i^-, m_i^+ and q_i with Algorithm 36, which is adapted from [15].

The idea is that p_i can be obtained from G thanks to $p_i = \sum_{j=1}^{d} a_{i,j} g_j$ while q_i can be obtained similarly with a right shift of G's negation. Indeed, $\mathtt{Not}(G)$ is the vector of encryptions of 1 if j is worse than the best median, of 0 otherwise. Its right shift is therefore encryptions of 1 if j is strictly worse than the best median, of 0 otherwise.

At this point, we remark that we can replace C as defined in line 2 of Algorithm 34 by the whole set of candidates, this without affecting the result, (see Lemma E.6). In what follows, we call Algorithm 34.E.6 the Algorithm 34 in which this transformation has been done.

Algorithm 35: Grade (Paillier setting)

Require: A such that, for all (i, j), $A_{i,j}$ is an encryption of the number of grades j given to candidate i

Ensure: G, such that for all j, G_j is an encryption of 1 if j is strictly greater than the best median, of 0 otherwise.

$V = \prod_{j=1}^{d} A_{1j}$

for $i = 1$ *to* k *(in parallel)* **do**

  for $j = 1$ *to* d *(in parallel)* **do**

    $B = \left(\prod_{l=1}^{j} A_{il}\right)^2$

    $C_{ij} = \mathtt{Not}(\mathtt{GTH}(B, V))$

for $j = 1$ *to* d *(in parallel)* **do**

  $G_j = C_{1j}$

  for $i = 2$ *to* k *(tree-based parallelisation is possible)* **do**

    $G_j = \mathtt{Mul}(G_j, C_{ij})$

Return G

Algorithm 36: InitD (Paillier setting)

Require: $(A_{ij}), G, n$ such that $A_{i,j}$ is an encryption of the number of j grades given to candidate i, while G is the grade vector and n the number of voters.

Ensure: P, M^-, M^+, Q where, for all i,

- P_i is an encryption of p_i, the number of grades received by i which are strictly greater than the best median,
- M_i^- is an encryption of $\lfloor n/2 \rfloor - p_i$,
- Q_i is an encryption of the number q_i of grades received by i which are strictly worse than the best median,
- M_i^+ is an encryption of $\lfloor n/2 \rfloor - q_i$.

for $i = 1$ *to* k **do**

  $P_i = \prod_{j=1}^{d} \mathtt{Mul}(A_{ij}, Gj)$

  $M_i^- = \mathtt{Enc}(\lfloor n/2 \rfloor)/P_i$

  $Q_i = \prod_{j=2}^{d} \mathtt{Mul}(A_{ij}, \mathtt{Not}(G_{j-1}))$

  $M_i^+ = \mathtt{Enc}(\lfloor n/2 \rfloor)/Q_i$

Lemma E.6. *In Algorithm 34, replacing line 2 by "Let C be the set of all candidates" will not alter the output.*

Proof. We show that after the first iteration of the loop, the C sets of both algorithms are the same, which shows that invariants from Lemma E.4 are verified at the beginning of the second iteration of the loop, if any (if not the output is correct as well since the sets are the same).

Let m be the best median, and a and b be two candidates such that $\mathsf{med}(b) < \mathsf{med}(a) = m$. For all i, after line 7 in both algorithms, p_i is the number of grades strictly better than m received by candidate i while q_i is the number of grades strictly worse than m received by candidate i. By definition of the median, we have $q_a \leq \lfloor n/2 \rfloor$. On the other hand, $p_b \leq \lfloor n/2 \rfloor < q_b$. But after line 7, we have $m_i^- + p_i = m_i^+ + q_i = \lfloor n/2 \rfloor$ for all i so $m_b^- > m_b^+$ and $S_b = -q_b$ after line 13. As $S_a \in \{p_a, -q_a\}$ with $p_a \geq -q_a \geq -\lfloor n/2 \rfloor > -q_b$, we have $S_b < S_a$. Therefore b is discarded from C at line 15. □

Lemma E.6 allows to initialize p_i, m_i^-, m_i^+ and q_i for all candidate i with no care of whether i's median is m or not. Now we explain how to run the while loop in MPC without revealing the number of iterations, nor the number of candidates which remain at any given point (see Lemma E.7).

Lemma E.7. *In Algorithm 34.E.6, we can replace line 8 by a for loop on d iterations, without affecting the result. Moreover, invariants from Lemma E.4 are still preserved.*

Proof. Following the proof of Lemma E.4, we remark that the proof does not depend on the number of iterations, so the loop invariants are preserved even if additional iterations are performed. Since the number of iterations is at most d as explained in the proof of Theorem E.5, this concludes the proof. □

Algorithm 37: MJ; version with a fixed number of loops, and an array of bits (indicator) instead of a set.

Require: a, the aggregated matrix.
Ensure: c, the indicator of the set of MJ winners.

1 Let m be the best median among all candidates
2 Let c such that for $i \in [1, k]$, $c_i = 1$
3 Let $I^- = 1$ and $I^+ = 1$ be counters
4 **for** $i = 1$ *to* k **do**
5     $p_i = \sum_{j=1}^{m-1} a_{i,j}$, $q_i = \sum_{j=m+1}^{d} a_{i,j}$
6     $m_i^- = \lfloor \frac{n}{2} \rfloor - p_i$, $m_i^+ = \lfloor \frac{n}{2} \rfloor - q_i$
7 **for** $j = 1$ *to* d **do**
8   **for** $i = 1$ *to* k **do**
9   **if** $m_i^- \leq m_i^+$ **then**
10             $s_i = p_i$
11   **else**
12             $s_i = -q_i$
13   **if** $c_i = 0$ **then**
14         $s_i = -n$ (* Already eliminated candidates are given a fake score *)
15   Let $s = \max\{s_i \mid i \in [1, k]\}$
16   **for** $i = 1$ *to* k **do**
17         $c_i = c_i \wedge (s_i == s)$
18   **if** $s \geq 0$ **then**
19   **for** $i = 1$ *to* k **do**
20             $m_i^+ = m_i^+ - m_i^-$
21             $m_i^- = a_{i,m-I^-}$
22             $p_i = p_i - a_{i,m-I^-}$
23         $I^- = I^- + 1$
24   **else**
25   **for** $i = 1$ *to* k **do**
26             $m_i^- = m_i^- - m_i^+$
27             $m_i^+ = a_{i,m+I^+}$
28             $q_i = q_i - a_{i,m+I^+}$
29         $I^+ = I^+ + 1$
30 Return c.

In what follows, we denote Algorithm 34.E.7 the Algorithm 34.E.6 in which line 8 is replaced by "for $j = 1$ to d do".

To encode C, we use its indicator (which we also denote C). To show the implied modification, we explicitly give Algorithm 37, where the transformations induced by Lemmas E.6 and E.7 have been made. To prove its correctness, we give the following lemma.

Lemma E.8. *In Algorithm 37, c is the indicator of C from Algorithm 34.E.7.*

Proof. We verify that this property holds as a loop invariant.

Initialisation. Before the first loop iteration, we have $c_i = 1$ for all $i \in [1, k]$ and $C = [1, k]$ so c is C's indicator.

Heredity. Suppose that before the j^{th} iteration in Algorithm 37, c is the indicator of the set C such as before the j^{th} iteration in Algorithm 34.E.7. Then for $i \in C$, $c_i = 1$ so s_i is the same in both algorithms. On the other hand, for $i \notin C$, $c_i = 0$ so $s_i = -n$ in Algorithm 37. By Lemma E.6, after the first loop iteration in Algorithm 34.E.6, C only contains candidates of median m. They therefore have at least a grade equal to m, so for all $i \in C$, $q_i \leq n - 1 < n$ after the first iteration. Since q_i can only decrease, we always have $p_i \geq -q_i > -n$ for $i \in C$, hence $s_i > -n$. Therefore, for $i \in C$ and $j \notin C$, $s_i > s_j$. This is also true in Algorithm 34.E.7, so s is the same in both algorithms after line 15. □

Now we explain how to get $a_{i,m-I^-}$ and $a_{i,m+I^+}$ without revealing $m - I^-$ et $m + I^+$. We use two vectors L and R of size d such that L_j is an encryption of 1 if $j = m - I^-$, of 0 otherwise, while R_j is an encryption of 1 if $j = m + I^+$, of 0 otherwise. This way $a_{i,m-I^-}$ and $a_{i,m+I^+}$ can be obtained with `SelectInd`. To initialize L and R, we use Algorithm 38 which uses the grade matrix g such that $g_j = 1$ if $j < m$, where

m is the best median, and $g_j = 0$ otherwise. The idea is that $m-1$ is the last index for which $g_j = 1$, so that $l_j = g_j - g_{j+1}$. Note that an initialization of R is obtained from L, with two right shifts. The only difficulty is when the best median is equal to the best possible grade, in which case g and l are null, while $r_2 = 1$. In any other case, $g_0 = 1$ and $r_2 = 0$, so we have $r_2 = 1 - g_0$.

Algorithm 38: InitP (Paillier setting)

Require: G, the grade matrix
Ensure: L, R, two vectors such that, for all i,
- L_i is an encryption of $i == m-1$,
- R_i is an encryption of $i == m+1$.

for $i = 1$ *to* $d-1$ **do**
  $L_i = G_i / G_{i+1}$
$L_d = \mathtt{Enc}(0)$
for $i = 3$ *to* d **do**
  $R_i = L_{i-2}$
$R_1 = \mathtt{Enc}(0)$, $R_2 = \mathtt{Not}(G_0)$
Return L, R

Algorithm 39: ConditionalLeftShift (CLS)

Require: V, B where V is a vector of $n-1$ ciphertexts and B an encryption of a bit b.
Ensure: Return a (reencrypted) left shift of V if $b = 1$, a reencryption of V otherwise.
for $j = 1$ *to* $n-1$ *(in parallel)* **do**
  $V'_j = \mathtt{Select}(V_j, V_{j+1}, B)$ (* $V_n = \mathtt{Enc}(0)$ *)
Return V'

Algorithm 40: ConditionalRightShift (CRS)

Require: V, B where V is a vector of $n-1$ ciphertexts and B an encryption of a bit b.
Ensure: Return a (reencrypted) right shift of V if $b = 1$, a reencryption of V otherwise.
for $j = 2$ *to* $n-1$ *(in parallel)* **do**
  $V_j = \mathtt{Select}(V_j, V_{j-1}, B)$ (* $V_1 = \mathtt{Enc}(0)$ *)
Return V

In order to increment I^- and I^+, we use the simple Algorithms 39 and 40. Note that we always have $L_d = \mathtt{Enc}(0)$ while $R_d = \mathtt{Enc}(0)$, so L and R can be processed as vectors of $d-1$ ciphertexts.

The complete procedure is given in Algorithm 41, whose correctness is the claim of Theorem E.9. In this Algorithm, we add the constant n (the number of voters) to the candidates' scores at line 15, so that each integers to be compared are non-negative. The comparison requires therefore one additional bit but only for the first loop iteration. In the remaining iterations, we have $q_i \le \lfloor n/2 \rfloor$ so that we can add $\lfloor n/2 \rfloor$ instead of n. Since $p_i \le \lfloor n/2 \rfloor$, we no longer need an extra bit. For simplicity, we did not explicitly write this optimization in Algorithm 41. Another notable difference compared to Algorithm 37 is that instead of computing $m_i^- \le m_i^+$, we compute $p_i \ge q_i$ (which is equivalent by invariant 1 from Lemma E.4) since p_i and q_i are non-negative, while m_i^+ and m_i^- could be negative during the first loop iteration.

Theorem E.9. *Algorithm 41 is correct.*

Proof. See Lemmas E.6, E.7, E.8 and E.4, as well as Lemma E.10 below. ◻

Lemma E.10. *In Algorithm 41, after the i^{th} loop iteration, L and R are such that L_j is an encryption of $j == m - I^-$, while R_j is an encryption $j == m + I^+$.*

Algorithm 41: MJ: MPC version (Paillier setting)

Require: A, n, the (encrypted) aggregated matrix and the number of voters.
Ensure: c, the indicator of the set of winners.

for $i = 1$ *to* k **do**
  $C_i = \mathtt{Enc}(1)$
$G = \mathtt{Grade}(A)$
$P, M^-, M^+, Q = \mathtt{InitD}(A, G, n)$
$L, R = \mathtt{InitP}(G)$
for $j = 1$ *to* d **do**
  (* scores computation *)
  for $i = 1$ *to* k *(in parallel)* **do**
    $B_1 = \mathtt{GTH}(P_i, Q_i)$ (* $p_i \geq q_i$ *)
    $S_i = \mathtt{Select}(1/Q_i, P_i, B_1)$ (* p_i if $p_i \geq q_i$, $-q_i$ otherwise *)
    $S_i = \mathtt{Select}(\mathtt{Enc}(-n), S_i, C_i)$ (* eliminated candidates get the fake $-n$ score*)
    $S_i = \mathtt{Enc}(n) S_i$ (* $s_i = s_i + n$ *)
  $S = S_1$ (* research of the best score *)
  for $i = 2$ *to* k *(tree-based parallelisation is possible)* **do**
    $B_2 = \mathtt{GTH}(S_i, S)$
    $S = \mathtt{Select}(S, S_i, B_2)$ (* s_i is $s_i \geq s$, s otherwise *)
  for $i = 1$ *to* k *(in parallel)* **do**
    $B_3 = \mathtt{EQH}(S, S_i)$
    $C_i = \mathtt{Mul}(C_i, B_3)$ (* elimination of candidates who do not have the best score*)
  $B_4 = \mathtt{GTH}(S, \mathtt{Enc}(n))$
  for $i = 1$ *to* k *(in parallel)* **do**
    $A'_{i,m-I^-} = \mathtt{SelectInd}((A_{i,1}, \cdots, A_{i,d-1}), L)$
    $A'_{i,m+I^+} = \mathtt{SelectInd}((A_{i,2}, \cdots, A_{i,d}), R)$
    $T^+ = \mathtt{Select}(A'_{i,m+I^+}, M_i^+/M_i^-, B_4)$ (* $m_i^+ - m_i^-$ if $b_4 = 1$, $a_{i,m+I^+}$ otherwise *)
    $T^- = \mathtt{Select}(M_i^-/M_i^+, A'_{i,m-I^-}, B_4)$ (* $a_{i,m+-I^-}$ if $b_4 = 1$, $m_i^- - m_i^+$ otherwise *)
    $P_i = \mathtt{Select}(P_i, P_i/A'_{i,m-I^-}, B_4)$ (* $p_i - a_{i,m-I^-}$ if $b_4 = 1$, p_i otherwise *)
    $M_i^- = T^-$
    $M_i^+ = T^+$
    $Q_i = \mathtt{Select}(Q_i/A'_{i,m+I^+}, Q_i, B_4)$ (* q_i if $b_4 = 1$, $q_i - a_{i,m+I^+}$ otherwise *)
  $L = \mathtt{CLS}(L, B_4)$, $R = \mathtt{CRS}(R, \mathtt{Not}(B_4))$
$c = \mathtt{Dec}(C)$ (* bit-wise decryption *)
Return c

E.5 An adaptation in MPC in the ElGamal setting

In the previous section, we gave an adaptation in MPC of the MJ tally function in the Paillier setting. As explained in Section 2.3, it is interesting to consider ElGamal encryptions to obtain a better computational complexity, especially at the voter-side. Note that most of Algorithm 41 is easy to adapt in the ElGamal setting thanks to the toolbox we provide. In this setting, the (encrypted) aggregated matrix must be encrypted in bit-encoding, so that obtaining the aggregated matrix from the list of encrypted ballots is no longer straightforward, but requires kd parallel calls to $\mathtt{Aggreg}^{\mathrm{bits}}$, which is the main drawback of this approach. Even if those computations can be made on the fly while the voters submit their ballot, if nkd is too large, the Paillier setting might be preferable as this phase would be too expensive.

Another difference is that in the Paillier setting, some procedures were performed thanks to the homomorphic property while they need the $\mathtt{Add}^{\mathrm{bits}}$ algorithm in the ElGamal setting. As replacing each multiplication of two ciphertexts in Algorithm 41 by a call to Algorithm 11 might deteriorate the complexity too much, we made a few modifications listed below.

First, we give Algorithm 42 which allows to initialize p_i, m_i^-, m_i^+ and q_i, just as Algorithm 36, but also initialize L and R as in Algorithm 38. Finally Algorithm 42 also initializes C as the indicator of the candidates whose median is the best median. In what follows, we use bold characters to denote a matrix of elements. For instance, $\boldsymbol{A}^{\mathrm{bits}}$ stands for a matrix of size kd, whose elements are bit-encoded encrypted integers. By abuse of notation, we use $\lfloor n/2 \rfloor$ or n instead of bit-encoded encryption of the said integer.

Algorithm 42: InitALL (ElGamal setting)

Require: $\boldsymbol{A}^{\text{bits}}$, such that, for all (i, j), ${A_{i,j}}^{\text{bits}}$ is a bit-encdoded encryption of $a_{i,j}$ from the aggregated matrix.

Ensure: $\boldsymbol{P}^{\text{bits}}, \boldsymbol{M}^{-\text{bits}}, \boldsymbol{M}^{+\text{bits}}, \boldsymbol{Q}^{\text{bits}}, L, R, C$ where, for all $i \in [1, k]$,

- ${P_i}^{\text{bits}}$ is a bit-wise encryption of p_i, the number of grades received by candidate i which are strictly greater than the best median,
- $M_i^{-\text{bits}}$ is a bit-wise encryption of $\lfloor n/2 \rfloor - p_i$,
- ${Q_i}^{\text{bits}}$ is a bit-wise encryption of q_i, the number of grades received by candidate i which are strictly worse than the best median,
- $M_i^{+\text{bits}}$ is a bit-wise encryption of $\lfloor n/2 \rfloor - q_i$,
- L_j is an encryption of $j == N - 1$ for all j, where N is the best median,
- R_j is an encryption of $j == N + 1$ for all j, where N is the best median,
- C_i is an encryption of 1 if i's median is N, of 0 otherwise.

1 **for** $i = 1$ *to* k *(in parallel)* **do**
2     ${S_{i,1}}^{\text{bits}} = {A_{i,1}}^{\text{bits}}$
3   **for** $j = 1$ *to* $d - 2$ **do**
4         $D_{i,j} = \mathtt{LT}({S_{i,j}}^{\text{bits}}, \lceil n/2 \rceil)$
5         ${S_{i,j+1}}^{\text{bits}} = \mathtt{Add}^{\text{bits}}({S_{i,j}}^{\text{bits}}, {A_{i,j+1}}^{\text{bits}})$ (* $s_{i,j} = \sum_{k=1}^{j} a_{i,j}$ *)
6     $D_{i,d-1} = \mathtt{LT}({S_{i,d-1}}^{\text{bits}}, \lceil n/2 \rceil)$
7     $S_i, d^{\text{bits}} = n$
8 **for** $j = 1$ *to* $d - 1$*(in parallel)* **do**
9     $G_j = D_{1,j}$
10   **for** $i = 2$ *to* k *(tree-based parallelisation is possible)* **do**
11         $G_j = \mathtt{CGate}(G_j, D_{i,j})$
12 **for** $i = 1$ *to* k *(in parallel)* **do**
13     $X = G_1 D_{i,1} / \mathtt{CGate}(G_1, D_{i,1})^2$ (* $g_1 \oplus d_{i,1}$ *)
14     $C_i = \mathtt{Not}(X)$ (* $g_1 == d_{i,1}$ *)
15   **for** $j = 2$ *to* $d - 1$ *(tree-based parallelisation is possible)* **do**
16         $X = G_j D_{i,j} / \mathtt{CGate}(G_j, D_{i,j})^2, C_i = \mathtt{CGate}(C_i, \mathtt{Not}(X))$ (* $g_j = d_{i,j}$ for all j *)
17 $L, R = \mathtt{InitP}(G)$
18 **for** $i = 1$ *to* k *(in parallel)* **do**
19     ${P_i}^{\text{bits}} = \prod_{j=1}^{d-1} \mathtt{CGate}({S_{i,j}}^{\text{bits}}, L_j)$ (* Bit-wise product and $\mathtt{CGate}$, as in $\mathtt{SelectInd}^{\text{bits}}$ *)
20     ${Q_i}^{\text{bits}} = \prod_{j=2}^{d} \mathtt{CGate}({S_{i,j}}^{\text{bits}}, L_{j-1})$ (* same as above *)
21     ${Q_i}^{\text{bits}} = \mathtt{Sub}^{\text{bits}}(n, {Q_i}^{\text{bits}})$
22     $M_i^{-\text{bits}} = \mathtt{Sub}^{\text{bits}}(\lfloor n/2 \rfloor, {P_i}^{\text{bits}}), M_i^{+\text{bits}} = \mathtt{Sub}^{\text{bits}}(\lfloor n/2 \rfloor, {Q_i}^{\text{bits}})$
23 Return $(\boldsymbol{P}^{\text{bits}}, \boldsymbol{M}^{-\text{bits}}, \boldsymbol{M}^{+\text{bits}}, \boldsymbol{Q}^{\text{bits}}, L, R, C)$

Algorithm 42 is a merger of Algorithms 35, 36 and 38. Merging all three algorithms together allows to exploit common intermediate computations. Note that at line 4, we compute $\lceil n/2 \rceil > s_{i,j}$ instead of $n > 2s_{i,j}$, so as to use one bit fewer. (See Lemma E.11 which states that the two comparisons are equivalent.)

Lemma E.11. *For all $n, s \in \mathbb{Z}$, we have $n > 2s$ if and only if $\lceil n/2 \rceil > s$.*

Proof. Let n, s be integers. If $n > 2s$, $\lceil n/2 \rceil \geq n/2 > s$. Conversely, suppose that $\lceil n/2 \rceil > s$. We first consider the case where n is even. Then $n/2 = \lceil n/2 \rceil$ so $n = 2\lceil n/2 \rceil > 2s$. If n is odd, we have $\lceil n/2 \rceil = (n+1)/2$ so $n + 1 > 2s$, therefore $n + 1 \geq 2s + 1$, hence $n \geq 2s$. Since n is odd, $n \neq 2s$, thus $n > 2s$. ◻

In Algorithm 41, we did not have to initialize C (see Lemma E.6). However, as the variables could be negative, we decided to add a constant. This would not be that easy in the ElGamal setting since adding a constant to a bit-encoded encrypted integers would require a non-trivial operations. In this case, eliminating the candidates who do not have the best median right away so as to initialize C consistently with Algorithm 34 has approximately the same computational cost. Afterwards, for all i, we have $\mid s_i \mid \leq \lfloor n/2 \rfloor$ so we can add the constant 2^{m-1}

Algorithm 43: MJ: MPC version (ElGamal setting)

Require: $\boldsymbol{B}$, the n encrypted ballots
Ensure: c, the indicator of the set of winners.

for $i = 1$ *to* k *(in parallel)* **do**
  for $j = 1$ *to* d *(in parallel)* **do**
    ${A_{i,j}}^{\text{bits}} = \texttt{Aggreg}^{\text{bits}}(B_{i,j,1}, \cdots, B_{i,j,n})$
$\boldsymbol{P}^{\text{bits}}, \boldsymbol{M}^{-\text{bits}}, \boldsymbol{M}^{+\text{bits}}, \boldsymbol{Q}^{\text{bits}}, L, R, C = \texttt{InitALL}(\boldsymbol{A}^{\text{bits}})$
for $j = 1$ *to* d **do**
  for $i = 1$ *to* k *(in parallel)* **do**
    $B_1 = \texttt{Not}(\texttt{LT}({P_i}^{\text{bits}}, {Q_i}^{\text{bits}}))$
    $P^{+\text{bits}} = P_{i,0}, \cdots, P_{i,m-2}, E(1)$ (* $2^{m-1} + p_i$ *)
    $Q^{+\text{bits}} = \texttt{Neg}({Q_i}^{\text{bits}})$ (* $2^{m-1} - q_i$ *)
    ${S_i}^{\text{bits}} = \texttt{Select}^{\text{bits}}(Q^{+\text{bits}}, P^{+\text{bits}}, B_1)$
    ${S_i}^{\text{bits}} = \texttt{CGate}(S_{i,0}, C_i), \cdots, \texttt{CGate}(S_{i,m-1}, C_i)$ (* give the fake score 0 to already eliminated candidates *)
  $S^{\text{bits}} = {S_1}^{\text{bits}}$
  for $i = 2$ *to* k *(tree-base parallelisation is possible)* **do**
    $B_2 = \texttt{LT}(S^{\text{bits}}, {S_i}^{\text{bits}})$
    $S^{\text{bits}} = \texttt{Select}^{\text{bits}}(S^{\text{bits}}, {S_i}^{\text{bits}})$
  for $i = 1$ *to* k *(in parallel)* **do**
    $B_3 = \texttt{EQ}^{\text{bits}}(S^{\text{bits}}, {S_i}^{\text{bits}})$
    $C_i = \texttt{CGate}(C_i, B_3)$
  $B_4 = S_{m-1}$ (* the most significant bit of s tells whether $s \geq 2^{m-1}$ *)
  for $i = 1$ *to* k *(in parallel)* **do**
    ${A'_{i,m-I^-}}^{\text{bits}} = \prod_{j=1}^{d-1} \texttt{CGate}({A_{i,j}}^{\text{bits}}, L_j)$ (* bit-wise product and CGate *)
    ${A'_{i,m+I^+}}^{\text{bits}} = \prod_{j=2}^{d} \texttt{CGate}({A_{i,j}}^{\text{bits}}, R_j)$ (* same as above *)
    ${M_{+-}}^{\text{bits}} = \texttt{Sub}^{\text{bits}}(M_i^{+\text{bits}}, M_i^{-\text{bits}})$
    ${M_{-+}}^{\text{bits}} = \texttt{Neg}({M_{+-}}^{\text{bits}})$
    $T^{+\text{bits}} = \texttt{Select}^{\text{bits}}({A'_{i,m+I^+}}^{\text{bits}}, {M_{+-}}^{\text{bits}}, B_4)$
    $T^{-\text{bits}} = \texttt{Select}^{\text{bits}}({M_{-+}}^{\text{bits}}, {A'_{i,m-I^-}}^{\text{bits}}, B_4)$
    ${P_i}^{\text{bits}} = \texttt{Select}^{\text{bits}}({P_i}^{\text{bits}}, \texttt{Sub}^{\text{bits}}({P_i}^{\text{bits}}, {A'_{i,m-I^-}}^{\text{bits}^{\text{bits}}}, B_4))$
    $M_i^{-\text{bits}} = T^{-\text{bits}}$
    $M_i^{+\text{bits}} = T^{+\text{bits}}$
    ${Q_i}^{\text{bits}} = \texttt{Select}^{\text{bits}}(\texttt{Sub}^{\text{bits}}({Q_i}^{\text{bits}}, {A'_{i,m+I^+}}^{\text{bits}}), {Q_i}^{\text{bits}}, B_4)$
  $\texttt{CLS}(L, B_4), \texttt{CRS}(R, \texttt{Not}(B_4))$
$c = \texttt{Dec}(C)$ (* bit-wise decryption *) Return c

instead, where m is the bit length of the integers. Indeed, $2^{m-1} > \lfloor n/2 \rfloor \geq q_i$ and $2^{m-1} + p_i \leq 2^{m-1} + \lfloor n/2 \rfloor < 2^m$. This is of interest because computing $2^{m-1} + p_i$ is completely free (just add $\texttt{Enc}(1)$ as the most significant bit); so we just have to call $\texttt{Neg}$ once (to compute $2^{m-1} - q_i$) instead of calling twice $\texttt{Add}^{\text{bits}}$.

Finally, we obtain Algorithm 43 for our ElGamal version of a fully-hiding tallying of MJ.

E.6 Majority Judgment, the bottom-line

To improve readability, we give again the details that are necessary to use our tally-hiding protocol inside of a voting protocol. First, to submit a ballot, a voter can simply use the `vote` procedure, which is summed up in Figure 12. This allows the voter to either give a grade to each candidate, either vote blank. Finally, to proceed with the tally, the authorities use the protocol $P_{\texttt{MJ}}$, defined in Algorithm 43.

- Let pk be the public encryption key and v the chosen voting option.
- Encode v as a matrix a of kd bits, where k is the number of candidates and d is the number of grades. The bit $a_{i,j}$ is set if and only if the grade j is given to candidate i.
- Encrypt the matrix into $(A_{i,j})_{i,j}$, using pk.
- For all i, j, produce a ZKP $\pi^{0/1}_{i,j}$ that $A_{i,j}$ is an encryption of 0 or 1.
- For all i, produce a ZKP $\pi^{0/1}_{i}$ that the product $A_{i,1} \cdots A_{i,d}$ is an encryption of 0 or 1.
- Produce a ZKP $\pi^{0/k}$ that the product $\prod_{i,j} A_{i,j}$ is an encryption of 0 or k.
- Return A, $(\pi^{0/1}_{i,j})_{i,j}$, $(\pi^{0/1}_{i})_{i}$, $\pi^{0/k}$.

Figure 12: `vote` procedure for the Majority Judgment

F CONDORCET METHODS, SCHULZE AND RANKED-PAIRS VARIANTS

In this Section, we give details about our approach to handle the Condorcet tally function that was only sketched in Section 5. While only the Condorcet-Schulze variant is mentioned in the main body of the article, we also cover here the ranked-pairs method. We refer to [1] for a discussion and a comparison of the many Condorcet variants, Schulze and ranked pairs being only two of them.

After recalling the notion of adjacency matrix, we define with more details the Schulze and ranked pairs variants and explain how they can be processed in MPC once the adjacency matrix is known. We then focus on how to compute this matrix from the encrypted ballots. The security notions are addressed in Appendix I.3.

F.1 Schulze and ranked pairs from the adjacency matrix

In the Condorcet methods, voters are asked to rank each candidate, potentially with ties (several candidates may have the same rank). The Condorcet winner is the candidate which is preferred to every other candidate by a majority of voters. Schulze and ranked pairs differ when there is no Condorcet winner. Like in many versions of Condorcet, only the adjacency matrix, which is defined in Definition F.1, is needed to compute the winners. In all what follows, we denote $d_{i,j}$ the number of voters who prefers (strictly) candidate i over candidate j.

Definition F.1 (Adjacency matrix). The adjacency matrix is the matrix $(a_{i,j})$ defined by

$$a_{i,j} = \begin{cases} d_{i,j} - d_{j,i} & \text{if } d_{i,j} \geq d_{j,i} \\ 0 & \text{otherwise.} \end{cases}$$

The Schulze variant.

The Schulze variant consists of several steps. First, compute $d_{i,j}$ for all (i, j). Second, compute $b_{i,j} = d_{i,j} - d_{j,i}$ for all (i, j). For all pair of candidates (u, v), a path p of length l from u to v is a finite sequence of $l + 1$ candidates such that $u = p_0$ and $v = p_l$. We say that $(i, j) \in p$ if there exists an index $0 \leq k < l$ such that $i = p_k$ and $j = p_{k+1}$. The strength of a path p is defined as $s(p) = \min_{(i,j)\in p} b_{i,j}$. The third step of the Schulze method is to compute $f_{i,j} = \max_{\sigma\in[i\rightsquigarrow j]} s(\sigma)$, where $[i \rightsquigarrow j]$ denotes the set of all paths from i to j. Finally, i is a winner by the Schulze method if $f_{i,j} \geq f_{j,i}$ for all j.

If a is the adjacency matrix, a Schulze tally can be derived from a (see Lemma F.2). When a is seen as the adjacency matrix of a graph, the Schulze method is well known to be equivalent to the shortest path problem [36], that can be solved with standard algorithms [23, 43].

Lemma F.2. *A Schulze tally can be performed from the adjacency matrix, by using $a_{i,j} = \max\{0, b_{i,j}\}$ instead of $b_{i,j} = d_{i,j} - d_{j,i}$, where $d_{i,j}$ is the number of voters who prefers i over j.*

Proof. For all path p, we denote $s(p) = \min_{(i,j)\in p} b_{i,j}$ and $s'(p) = \min_{(i,j)\in p} a_{i,j}$. For all (i, j), we denote

$$f_{i,j} = \max_{\sigma\in[i\rightsquigarrow j]} \min_{(u,v)\in\sigma} b_{i,j}$$
$$f'_{i,j} = \max_{\sigma\in[i\rightsquigarrow j]} \min_{(u,v)\in\sigma} a_{i,j}.$$

With these notations, the statement of the lemma becomes

$$\forall i, (\forall j, f_{i,j} \geq f_{j,i}) \iff (\forall j, f'_{i,j} \geq f'_{j,i}).$$

Let i be a candidate, suppose that for all j, $f_{i,j} \geq f_{j,i}$ (*i.e.* i is a Schulze winner). Let j be any candidate. If $j = i$, clearly $f'_{i,j} \geq f'_{j,i}$, so we assume that $j \neq i$. Since $j \neq i$, there is no path from i to j (nor from j to i) of length 0. As $f_{i,j} \geq f_{j,i}$, there exists a path p from i to j (of length $n > 0$) such that for all path p' from j to i (of length $n' > 0$), there exists $k' < n'$ such that for all $k < n$, $b_{p'_{k'},p'_{k'+1}} \leq b_{p_k,p_{k+1}}$. We consider two cases.

First, if $b_{p_k,p_{k+1}} < 0$ for some k, then for all p', $b_{p'_{k'},p'_{k'+1}} < 0$ for all k', hence $a_{p'_{k'},p'_{k'+1}} = 0$ for all k', thus $s'(p') = 0 \leq s'(p)$. Since this is holds for all p', $f'_{j,i} = 0 \leq f'_{i,j}$.

Second, if $b_{p_k,p_{k+1}} \geq 0$ for all k, then for all k, $a_{p_k,p_{k+1}} = b_{p_k,p_{k+1}}$. Now consider any path p' (of length $n' > 0$) from j to i. If $b_{p'_{k'},p'_{k'+1}} \geq 0$ for all k', then $s'(p') = s(p') \leq f_{j,i} \leq f_{i,j} = s(p) = s'(p) \leq f'_{i,j}$. If there exists k' such that $b_{p'_{k'},p'_{k'+1}} < 0$, then $s'(p') = 0 \leq f'_{i,j}$. Therefore $f'_{j,i} \leq f'_{i,j}$.

Conversely, let i such that $f'_{j,i} \leq f'_{i,j}$ for all j. Let j be any candidate (as above, w.l.o.g. we assume that $i \neq j$). We consider three cases.

First, suppose that $f_{i,j} < 0$. Then for all path p from i to j, there exists $(u,v) \in p$ such that $b_{u,v} < 0$ (we call this proposition $*$). In particular, $b_{i,j} < 0$, so $b_{j,i} = -b_{i,j} > 0$, hence $b_{j,i} = a_{j,i}$ and $f'_{j,i} \geq s'_{j,i} = a_{j,i} = b_{j,i}$. In addition, $s_{j,i} = b_{j,i} > 0$, so $f'_{j,i} \geq s_{j,i} > 0$. On the other hand, by $*$ we have $f'_{i,j} = 0$, which contradicts $f'_{j,i} \leq f'_{i,j} < 0$. Therefore $f_{i,j} \geq 0$.

Second, suppose that $f_{i,j} = 0$. Then for all path p from i to j, there exists $(u,v) \in p$ such that $b_{u,v} \leq 0$, hence $f'_{i,j} = 0$. Let p' be a path from j to i (of length $n' > 0$). Suppose that for all $(u,v) \in p'$, $b_{u,v} > 0$. Then $0 < s'(p) \leq f'_{j,i}$, which contradicts $f'_{j,i} \leq f'_{i,j}$. Consequently, there exists $(u,v) \in p'$ such that $b_{u,v} \leq 0$, therefore $s(p') \leq 0 = f_{i,j}$. This holds for all p' so $f j, i \leq f_{i,j}$.

Finally, suppose that $f_{i,j} > 0$. Let p' be a path from j to i. If there exists $(u,v) \in p'$ such that $b_{u,v} \leq 0$, then $s(p') \leq 0 < f_{i,j}$. Otherwise, for all $(u,v) \in p'$, $b_{u,v} > 0$ so $s(p') = s'(p') \leq f'_{j,i} \leq f'_{i,j}$, so we just have to show that $f_{i,j} \geq f'_{i,j}$.

Let p be a path from i to j. If there exists $(u,v) \in p$ such that $b_{u,v} \leq 0$, $s'(p) = 0 < f_{i,j}$. Otherwise, for all $(u,v) \in p$, $b_{u,v} > 0$ so $s'(p) = s(p) \leq f_{i,j}$, which concludes the proof. □

From this lemma, the Schulze tally can be derived by a simple Floyd-Warshall algorithm and we give it in Algorithm 45 for completeness. This has a cost that is cubic in the number of candidates (here, this number is denoted n).

Algorithm 44: FW (Floyd-Warshall algorithm)

```
Require: P, the encrypted adjacency matrix
Ensure: S, such that S_{i,j} is an encryption of the strength of the strongest path from i to j
(* n is the number of candidates *)
S = P
for k = 1 to n do
    for i = 1 to n (in parallel) do
        for j = 1 to n (in parallel) do
            (* proceed only if (i ≠ j) *)
            A_{i,j} = Select(S_{k,j}, S_{i,k}, LT(S_{i,k}, S_{k,j}))
            B_{i,j} = Select(S_{i,j}, A_{i,j}, LT(S_{i,j}, A_{i,j}))
    S_{i,j} = B_{i,j} for all (i ≠ j)
Return S
```

Algorithm 45: Schulze (from adjacency matrix)

```
Require: A, the encrypted adjacency matrix
Ensure: c, the indicator of the Schulze winners
(* n is the number of candidates *)
S = FW(A)
for i = 1 to n (in parallel) do
    for j ≠ i (in parallel) do
        b_j = Not(LT(S_{i,j}, S_{j,i}))
    C_i = CSZ_{j≠i}(b_j) (* use tree-based parallelization to compute the conjunction of all b_j *)
Return c = Dec(C)
```

The ranked pairs variant.

The ranked pairs is another algorithm which allows to break ties when there is no Condorcet winner. In this method, the adjacency matrix is seen as the adjacency matrix of a graph G. The Ranked Pairs protocol consists of three steps. First, sort the edges of G in decreasing order of weights. Let G' be the graph which consists of k vertices (where k is the number of candidates) and no edge. Second, for all edge of G taken in decreasing order, if this edge does not create a cycle in G', add this edge in G'. Finally, as G' is an oriented graph without cycle, G' is the graph of a partial order over the candidates. The sources of the graph are the winners according to the Ranked Pairs protocol.

Assuming the adjacency matrix is known, an MPC version of the ranked pairs method goes as follows. First, to shuffle the edges, we can use the bubble-sort algorithm. The edges can be encoded with three ciphertexts, one for the source, one for the destination and one for the weight. Then, the main procedure is to update a matrix $B_{i,j} = \mathtt{Enc}(b_{i,j})$, where $b_{i,j} = 1$ if there is a path from i to j, and 0 otherwise. Initially, B is simply an encryption of the identity matrix. To add the edge (i, j) simply compute $b'_{s,t}$ for all (s, t), as follows:

$$b'_{s,t} = b_{s,t} \vee (b_{s,i} \wedge b_{j,t}).$$

The edge will create a cycle if and only $b'_{s,t} = b'_{t,s} = 1$ for some (s, t), hence we compute the encryption of the boolean

$$c = \vee_{s \neq t}(b'_{s,t} \wedge b'_{t,s}).$$

Finally, we can update $b_{i,j}$ using `Select` and c.

The problem that remains is that (i, j) is unknown, since the edges are encrypted. A simple solution is to perform the test $u == i$ and $v == j$ for all (u, v), using the known (u, v) and the encryptions of (i, j), and to update each $b_{u,v}$ using `Select`, so as to hide the results of both tests (only one $b_{u,v}$ will be modified, while the others will be re-encrypted). This leads to an additional $O(k^2 \log k)\mathtt{CGate}$, as `EQ` requires $O(\log k)\mathtt{CSZ}$. Finally, finding the source of the graph can be done by exhaustive search on the final B, which cost $O(k^2\mathtt{CSZ})$. The whole process can be performed in $O(k^4 \log k)\mathtt{CSZ}$ in terms of computation, communication and transcript size.

F.2 How to obtain the adjacency matrix from the voters' ballots

The preference matrices.

The choice of a voter can be modelled by a preference matrix. We consider two types of such matrices (see Figure 13). The m_a preference matrix format is antisymmetric, therefore only $k(k-1)/2$ elements need to be considered. The m_p preference matrix has only non-negative integers, which can also be an advantage.

$$m_a[i,j] = \begin{cases} 1 & \text{if } i \text{ is preferred over } j \\ -1 & \text{if } j \text{ is preferred over } i \\ 0 & \text{otherwise.} \end{cases} \qquad m_p[i,j] = \begin{cases} 1 & \text{if } i \text{ is preferred over } j \\ 0 & \text{otherwise.} \end{cases}$$

Figure 13: Two types of preference matrix

Deducing the adjacency matrix from the set of preference matrices of each voter boils down to aggregating with a bit more details as explained below. Using the homomorphic property, this is straightforward from the m_a type, but this requires either to be in the Paillier setting or to reveal the adjacency matrix. Otherwise, the bitwise encryption is required, and then the m_p matrix format is better suited.

Ballots encoded as list of integers.

We assume here that each ballot consists of $k\lceil\log(k+1)\rceil$ ciphertexts, along with zero knowledge proofs that they are encryptions of 0 or 1. Those ciphertexts are interpreted as k bit-encoded integers, which encrypt integers in $[0, 2^L - 1]$, where 2^L is the first power of 2 greater than k. This way the voter can give each candidate a rank (which is not necessarily between 0 and $k-1$), and can give the same rank to several candidates without any restriction.

First, we consider the easy case where only the m_a preference matrix in the natural encoding is needed, because the adjacency matrix will be revealed. In that case, we simply use a variant of `LT` which returns an additional bit for the equality test (see Section C.2). Let ${C_i}^{\text{bits}}$ and ${C_j}^{\text{bits}}$ be the bitwise encrypted rank of candidates i and j for some ballot. Let $Z, T = \mathtt{LT}({C_i}^{\text{bits}}, {C_j}^{\text{bits}})$. Then $M_a[i,j] = Z^2T/\mathtt{Enc}(-1)$, and $M_a[j,i] = 1/M_a[i,j]$. Therefore the preference matrix m_a can be obtained in $k(k-1)/2$ calls of `LT`, which accounts for $\frac{3}{2}k(k-1)\log k\,\mathtt{CSZ}$ in computation and transcript size, and $2\log k\,\mathtt{CSZ}$ in communication since all $m_a[i,j]$ can be computed in parallel.

For a full tally-hiding procedure, we need the result to be bitwise encrypted and the m_p preference matrix is better suited. Similarly, we use a variant of `LT` which returns an additional bit. This additional bit allows to derive $m_p[j,i]$ from $m_p[i,j]$ using `Not` and `CSZ`. Hence the preference matrix is obtained with $\frac{3}{2}k(k-1)\log k\,\mathtt{CSZ}$ in computation and transcript size, and $2\log k\,\mathtt{CSZ}$ in communication just as in the previous case. The aggregation requires to call $\mathtt{Aggreg}^{\text{bits}}$ to obtain a matrix D. By construction, $D_{i,j}$ is a bit-wise encryption of the number $d_{i,j}$ of voters who prefers i over j. For all $i < j$, we can then use `SubLT` to compute (bit-encoded encryptions of) $b_{i,j} = d_{i,j} - d_{j,i}$, as well as an additional bit $(b_{i,j} < 0)$. This bit allows to derive the adjacency matrix by setting all negative values to zero using `CSZ`, and by computing $b_{j,i}$ from $b_{i,j}$ using `Neg` and `CSZ`.

Ballots encoded as preference matrices (quadratic algorithm).

We explain now how the voters can directly encode their choice as a preference matrix of the m_a type. The difficulty is for the voter to prove in zero-knowledge that the matrix encoded in their ballot is indeed a preference matrix, *i.e.* that it corresponds to an ordering of the candidates. This is of great interest if one is ready to leak the adjacency matrix, because then the tally can be done by the authorities without any MPC protocol apart from the decryption.

We start by explaining our method in the cleartexts. Suppose that Alice wants to vote the ordering $(1, \cdots, k)$ (*i.e.* the candidate number i is ranked i^{th}). Then her preference matrix would be as follows.

$$m_{\mathsf{init}}[i,j] = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{if } i < j \\ -1 & \text{otherwise.} \end{cases}$$

Now assume that Alice wants to rank $\sigma(i)^{th}$ the candidate number i, for some permutation σ that encodes her choice. If the candidate number i were numbered $\sigma(i)$ instead, Alice could have voted with m_{init} as above. This means that the preference matrix of Alice m_a is such that $m_a[\sigma^{-1}(i), \sigma^{-1}(j)] = m_{\mathsf{init}}[i,j]$ for all (i,j). Therefore m_a can be obtained by using the permutation σ to shuffle m_{init} (using the permutation on the rows, then on the columns, with the `ShuffleMatrix` function).

So far, Alice can only choose a strict ordering of the candidates. Assume that she wants to give the same rank to several candidates and let r_i be the rank of candidate i according to her. Alice first sorts the candidates according to their rank, in increasing order. Let σ be the permutation used for sorting. At this point, σ is an arbitrary permutation such that $\sigma(i) < \sigma(j) \implies r_i \leq r_j$. To obtain her preference matrix m_a from m_{init}, Alice will first transform m_{init} into m_σ, such that $m_\sigma[i,j] = m_a[\sigma^{-1}(i), \sigma^{-1}(j)]$. For this purpose, she computes a vector b of size $k-1$ such that for all i, $b_i = 1$ if $r_{\sigma^{-1}(i)} = r_{\sigma^{-1}(i+1)}$, and 0 otherwise. Afterwards, Alice modifies m_{init} diagonal by diagonal, so as to indicate that some candidates are ranked equal. For the first diagonal, we have $m_{\mathsf{init}}[i, i+1] = 1$ while we would like $m_\sigma[i, i+1] = 1 - b_i$. This can be done easily using the homomorphic property.

For the $(j+1)^{th}$ diagonal $(i, i+j+1)_i$, assume that the previous diagonal is correct. Then, as the candidates are sorted in order of preference, we have

$$m_\sigma[i, i+j+1] = \begin{cases} 0 & \text{if } (m_\sigma[i, i+j] = 0) \wedge (m_\sigma[i+1, i+j+1] = 0), \\ 1 & \text{otherwise.} \end{cases}$$

Therefore, Alice can apply an iterative algorithm, using the following formula:

$$\begin{aligned} m_\sigma[i, i+j+1] &= 1 - (1 - m_\sigma[i, i+j])(1 - m_\sigma[i+1, i+j+1]) \\ &= m_\sigma[i, i+j] + m_\sigma[i+1, i+j+1] - m_\sigma[i, i+j] m_\sigma[i+1, i+j+1]. \end{aligned}$$

Once m_σ is obtained, Alice can finally derive m_a by shuffling the rows and the columns, using the permutation σ and the `ShuffleMatrix` function.

The algorithm that we sketched above is interesting because it requires only a quadratic number of steps and it only uses transformations for which there is a standard zero knowledge proof. Indeed, a public and canonical encryption of m_{init} is available so Alice does not have to prove that m_{init} is well-formed. For the first diagonal, Alice simply has to provide $(k-1)$ ciphertexts and $0/1$ zero knowledge proofs. For the remaining diagonals, Alice has to provide an encryption Z of $m_\sigma[i, i+j] m_\sigma[i+1, i+j+1]$, as well as zero knowledge proof of well-formedness. For this purpose, Alice uses Algorithm 46 which produces a transcript π_{mul} of the form $(e_1, e_2, a_1, a_2, a_3)$. To verify the proof, one computes $d = \mathsf{hash}(X||Y||Z||e_1||e_2)$ where X is the encryption of $m_\sigma[i, i+j]$ and Y the encryption of $m_\sigma[i+1, i+j+1]$, and checks that the following equations are verified:

$$\begin{aligned} Y^{a_3} \mathsf{Enc}(0, a_1) Z^{-d} &= e_1 \\ \mathsf{Enc}(a_3, a_2) X^{-d} &= e_2. \end{aligned}$$

Finally, the shuffle can be performed with a standard proof of a shuffle.

Algorithm 46: ZKmult

Require: $\mathsf{hash}, X, Y, x, r_x$, such that $X = \mathsf{Enc}(x, r_x)$ and Y is any ciphertext

Ensure: Z, π_{mul}, such that $Z = \mathsf{ReEnc}(Y^x)$ and π_{mul} is a zero knowledge proof of well-formedness

```
α, r_1, r_2, w ∈_r Z_q, Z = Y^x Enc(0, α)
e_1 = Y^w Enc(0, r_1), e_2 = Enc(w, r_2)
d = hash(X||Y||Z||e_1||e_2)
a_1 = r_1 + αd, a_2 = r_2 + r_x d, a_3 = w + xd
π_mul = (e_1, e_2, a_1, a_2, a_3)
Return Z, π_mul
```

To summarize our construction, we recap the procedure to provide a ballot and prove its well-formedness in Figure 14. The proof can be verified by first verifying all the ZKP $\pi_i^{0/1}$. Then, using the B_i's, M_init and the $Z_{i,i+j+1}$'s, the verifier computes the matrix M_σ. She checks that it is well-formed by verifying all the ZKP $\pi_{i,i+j+1}^{mul}$ using M_σ and the $Z_{i,i+j+1}$'s. Finally, she verifies the proof of a shuffle using π^{Shuffle} and M_a. We denote `Verify` this verification algorithm.

- Let σ be (any) permutation such that $\sigma(i) < \sigma(j)$ $r_i \leq r_j$.
- For all $1 \leq i < k$, let $b_i = 1$ if $r_{\sigma^{-1}(i)} = r_{\sigma^{-1}(i+1)}$, 0 otherwise.
- For all $1 \leq i < k$, compute B_i, an encryption of b_i, and $\pi_i^{0/1}$, a ZKP that B_i is an encryption of either 0 or 1.
- Let $M_\sigma[i, i] = E_0$ and $M_\sigma[i, i+1] = E_1/B_i$ for all $1 \leq i \leq k$.
- For all $1 \leq j < k$,
  - For all $1 \leq i \leq k - j - 1$,
    * Obtain $Z_{i,i+j+1}$, $\pi^{mul}_{i,i+j+1}$ with algorithm ZKmult,
    * Compute $M_\sigma[i, i+j+1] = M_\sigma[i, i+j]M_\sigma[i+1, i+j+1]/Z_{i,i+j+1}$.
- Let $M_\sigma[j, i] = 1/M_\sigma[i, j]$ for all $i < j$.
- Use `ShuffleMatrix` to shuffle M_σ into M_a, and produce the ZKP of a shuffle $\pi^{\mathtt{Shuffle}}$.
- Return M_a, $\pi^{\mathtt{Shuffle}}$, $Z_{i,i+j+1}$, $\pi^{mul}_{i,i+j+1}$ for $1 \leq j < k$ and $1 \leq i \leq k - j - 1$, as well as B_i, $\pi_i^{0/1}$ for $1 \leq i < k$.

Figure 14: Voter's procedure to vote with the ranks $r_1, \cdots, r_k$

Claim 1. *Let* `Prove` *be the algorithm defined in Figure 14 and* `Verify` *the above verification process. Then* `Prove`, $Verify$ *is a Non-Interactive Zero Knowledge Proof system for the set of valid encrypted ballots M which verifies the following proposition.*

$$\exists r_1, \cdots, r_k \text{ s.t. } \forall (i, j), \mathtt{Dec}(M[i, j]) = \begin{cases} 1 & \text{if } r_i < r_j \\ 0 & \text{if } r_i = r_j \\ -1 & \text{otherwise.} \end{cases}$$

Proof sketch. **Completness.** Clearly, an honest voter will always have her ballot accepted by the verifier.

Zero Knowledge. Apart from the encrypted ballot M_a, only the B_i's, the $Z_{i,i+j+1}$'s and ZKP are published, hence the proof is Zero Knowledge.

Soundness. The soundness comes directly from the soundness of the ZKP used. Indeed, the soundness of the 0/1 ZKP guarantees that all the B_i's are encryption of 0 or 1, and the soundness of the private multiplication ZKP (see Algorithm 46) guarantees that the $Z_{i,i+j+1}$ are well-formed, and therefore that M_σ is a valid encrypted ballot. Finally, the soundness of the proof of a shuffle $\pi^{\mathtt{Shuffle}}$ guarantees that M_a is obtained from M_σ using some permutation σ'. Since any of these transformations preserve the set of the valid ballots, it follows that M_a is a valid encrypted ballot, even if $\sigma' \neq \sigma$. □

Ballots encoded as preference matrices (cubic algorithm).

For comparison, we present a naive approach to prove that an encrypted m_a preference matrix is well-formed. To do so, the voter can provide two proofs:

- A proof that each element is an encryption of either 0, 1 or −1,
- A proof of transitivity.

The proof of transitivity must prove the following statements, for all (i, j, k) and $u \in \{-1, 0, 1\}$

$$(m_a[i, k] = u) \wedge (m_a[k, j] = u) \implies m_a[i, j] = u.$$

Since the voter also provides a proof that each $m_a[i, j]$ is in $\{-1, 0, 1\}$, this is equivalent to proving that, for all (i, j, k), $m_a[i, k] = m_a[k, j] \implies m_a[i, j] = m_a[i, k]$, which is equivalent to proving that the following statement is true:

$$(m_a[i, k] \neq m_a[k, j]) \vee (m_a[i, j] = m_a[i, k]).$$

To prove that $m_a[i, k] \neq m_a[k, j]$, one can prove that $m_a[i, k] - m_a[k, j] \in \{-2, -1, 1, 2\}$ and to prove that $m_a[i, j] = m_a[i, k]$, we prove that the difference is 0. Overall, the voter has to prove that, for all (i, j, k),

$$(m_a[i, k] - m_a[k, j] = -2) \vee (m_a[i, k] - m_a[k, j] = -1) \vee (m_a[i, k] - m_a[k, j] = 1) \vee (m_a[i, k] - m_a[k, j] = 2) \vee (m_a[i, j] - m_a[i, k] = 0).$$

The proof of the disjunction can be obtained with the process of [21] (see Algorithm 47 below). To verify such a proof, simply compute $d = \mathsf{hash}(A_1|| \cdots ||A_5||e_1|| \cdots ||e_5)$ and check that $e_j = \mathsf{Enc}(0, \rho_j)(A_j/\mathsf{Enc}(b_j, 0))^{-\sigma_j}$ for all j. Overall, the zero knowledge proof requires about $18k^3$ for the prover and $20k^3$ for the verifier.

In [27], the authors use a similar approach, but for the m_p preference matrix. We stress that while their approach is more efficient than the above naive approach, it does not apply to the case where some candidates have the same rank.

Algorithm 47: ZKP of a 5-disjunction

Require: $\mathsf{hash}, A_1, \cdots, A_5, a_1, \cdots, a_5, r_1, \cdots, r_5, b_1, \cdots, b_5$, such that

- for all i, $A_i = \mathsf{Enc}(a_i, r_i)$
- there exists i such that $a_i = b_i$

Ensure: $(e_1, \cdots, e_5, \sigma_1, \cdots, \sigma_5, \rho_1, \cdots, \rho_5)$, a Zero Knowledge proof that there exists i such that $a_i = b_i$.

Let i such that $a_i = b_i$
$w \in_r \mathbb{Z}_q$, $e_i = \mathsf{Enc}(0, w)$
for $j \neq i$ **do**
    $\sigma_j, \rho_j \in_r \mathbb{Z}_q$
    $e_j = \mathsf{Enc}(0, \rho_j)(A_j/\mathsf{Enc}(b_j, 0))^{-\sigma_j}$
$d = \mathsf{hash}(A_1||\cdots||A_5||e_1||\cdots||e_5)$
$\sigma_i = d - \sum_{j \neq i} \sigma_i$
$\rho_i = w + \sigma_i r_i$
Return $(e_1, \cdots, e_5, \sigma_1, \cdots, \sigma_5, \rho_1, \cdots, \rho_5)$

- Let pk be the public encryption key and v the chosen voting option.
- Encode v as a vector of k integers, where k is the number of candidates. The ith integer is the desired rank for candidate i.
- Encrypt the vector into $B_1, \cdots, B_k$, using pk and a bitwise encryption for each integer (hence each B_i is in fact $\lceil \log k \rceil$ encryptions of either 0 or 1).
- For all i, produce $\lceil \log k \rceil$ ZKP $(\pi^{0/1}_{i,j})_{1 \leq j \leq \lceil \log k \rceil}$ that $B_{i,j}$ is an encryption of 0 or 1.
- Return $(B_i)_{1 \leq i \leq k}, (\pi^{0/1}_{i,j})_{i,j}$.

Figure 15: `vote` procedure for the D'Hondt method

F.3 Condorcet-Schulze method, the bottom-line

To improve readability, we give again the details that are necessary to use our tally-hiding protocol inside of a voting protocol. First, to submit a ballot, a voter can simply use the `vote` procedure, which is summed up in Figure 15. This allows the voter to freely give a rank to each candidate, among the $2^{\lceil \log k \rceil}$ possible ranks, where k is the number of candidates. Note that only the ordering of the candidate with the given rank is of interest, so ranking three candidates 1, 1 and 2 is the same as ranking them 0, 0 and 3. Finally, to proceeds with the tally, the authorities use the protocol P_{Cond}, defined in Algorithm 48.

G SINGLE TRANSFERABLE VOTE

Section 6 contains a sketch of our results on Single Transferable Vote (STV). We give here more material about this: we recall the general idea of STV and some variants, then explain in details the algorithms to use for each step of an MPC implementation, and finally explain how the costs given in table 4 were obtained. The security notions are addressed in Appendix I.3.

G.1 Overview of STV

STV consists of the following algorithm, where s is the number of seats to be attributed. First, each voter chooses a subset of candidates (any other candidate is not deemed of interest by the voter) and rank them in a strict order. For instance, if there are four candidates, Alice can vote $(1, 3)$ while Bob can vote $(4, 1, 2)$. Each ballot is attributed a weight, which is initially 1. Once all the ballots are cast, the tallying process consists of several rounds. During each round, each ballot grants a number of votes (equal to the ballot's weight) to the first candidate mentioned in the ballot. If some candidates meet a certain quota q (which is fixed during the whole process), the one with the greatest number of votes is selected. The selected candidates keep q votes for themselves and transfer each of their ballot to the next candidate on the ballot, with a transfer coefficient $t = (v - q)/v$, where v is the number of votes of the selected candidate (note that v might not be an integer). In other words, the name of the selected candidate is removed from the ballot and the weight is multiplied by t. The eliminated candidates transfer their ballot to the next candidate in the ballot, but with the same weight. The process terminates when s candidates are elected, or when the number of candidates that remain is equal to the number of (still) available seats.

There are several versions of STV. In the version that we chose to consider, the tallying process consists of several rounds, and in each round, exactly one candidate is either selected or eliminated. In some other versions, several candidates can be selected or eliminated simultaneously, if some conditions are met. This comes with two problems. First, for an MPC tally, revealing no more than the result also means not to reveal the number of candidates which were selected or eliminated in any round, so having a non-constant number of eliminations or selections is quite difficult. Second, if several candidates are selected simultaneously, the exact way in which the transfer

Algorithm 48: Condorcet-Schulze

Require: B, the n encrypted ballots
Ensure: c, the indicator of the set of winners

for $p = 1$ *to* n *(in parallel)* **do**
  for $i = 1$ *to* k *(in parallel)* **do**
    for $j = i + 1$ *to* k *(in parallel)* **do**
      $\_, T, C := \texttt{SubLT}(B_i, B_j)$ (* use a variant that returns an additional bit for the equality test *);
      $M_p[i, j] := T$ (* the candidate with the lowest rank is preferred *);
      $M_p[j, i] := \texttt{CSZ}(\texttt{Not}(T), \texttt{Not}(C))$
    $M_p[i, i] := E_0$ (* trivial encryption of 0 *)
for $i = 1$ *to* k *(in parallel)* **do**
  for $j = 1$ *to* k *(in parallel)* **do**
    ${M_{i,j}}^{\text{bits}} := \texttt{Aggreg}(M_1[i, j], \cdots, M_n[i, j])$
for $i = 1$ *to* k *(in parallel)* **do**
  for $j = i + 1$ *to* k, *(in parallel)* **do**
    $D^{\text{bits}}, N := \texttt{SubLT}(M_{i,j}, M_{j,i})$;
    $F^{\text{bits}} := \texttt{Neg}(D)$;
    ${A_{i,j}}^{\text{bits}} := \texttt{CSZ}(D, \texttt{Not}(N))$;
    ${A_{j,i}}^{\text{bits}} := \texttt{CSZ}(F, N)$
  $A_{i,i} := {E_0}^{\text{bits}}$
$S := \texttt{FW}(A)$;
for $i = 1$ *to* k *(in parallel)* **do**
  for $j = 1$ *to* k *(in parallel)* **do**
    $W_{i,j} := \texttt{Not}(\texttt{LT}(S_{i,j}, S_{j,i}))$
  $C_i := \texttt{CSZ}_j(W_{i,j})$ (* use tree-base parallelization to compute the conjunction of all w_j *);
  $c_i := \texttt{Dec}(C_i)$
Return c

should occur is not clear. Indeed, suppose that candidates a and b are selected. For each ballot possessed by a, a has to transfer a certain proportion t of the ballot to the second candidate mentioned in the ballot, but t depends on the number of votes possessed by a. So what if a must transfer some votes to b while b must transfer some votes to a? Which transfer coefficient should be used? While some variants of STV choose to ignore the selected candidates in the transfer process (don't transfer to b but to the next candidate that is not already selected), some other variants require to solve a system of c equations of degree c, where c is the number of candidates selected simultaneously [32].

G.2 A tally-hiding algorithm for STV

In what follows, we will only consider the ElGamal setting with bit-encoding, but a similar approach could be used in the Paillier setting as well (some procedures would become easier). Each ballot consists of $(k + 1)$ bit-wise encrypted integers, which are obtained by shuffling a public vector which contains bit-encoded encryptions of $(0, \cdots, k)$, where k is the number of candidates. The candidate 0 is an artificial candidate: any candidate ranked after 0 should be ignored. Also, we represent rational numbers with an approximation in the first r binary places, where r is fixed by the election administrator.

First, we initialize a data structure as follows (recall that q is the quota, s the number of seats, k the number of candidates and n the number of voters).

- H is the *hopeful* vector. It contains k encryptions of bits (initially E_1, the public encryption of 1).
- W is the *winner* vector. It contains k encryptions of bits (initially E_0, the public encryption of 0).
- S is the *score* vector. It contains k bit-encoded encrypted integers of size $m + r$, where $m = \lceil \log(n + 1) \rceil$.
- B is the *ballots* matrix. For all $i \in [1, n]$, B_i consists of a weight V_i (a bit-encoded encrypted integer of size $r + 1$, initially $(E_0, \cdots, E_1)$, which stands for the bit-encoded encryption of 1; the r less significant bits represent the r binary places) and $k + 1$ candidates $B_i[0], \cdots, B_i[k]$ (candidates are represented as bit-encoded encrypted integers, of size $\lceil \log(k + 1) \rceil$ bits).

In what follows, if P is a MPC procedure that requires two (bit-encoded) inputs, we denote P_k the procedure P in which the second input is known in the clear. If m is the bitsize of the inputs, P_k costs generally $m\,\texttt{CSZ}$ less than P, which often leads to a good improvement (a third or a half of the computations is saved, see Algorithm 21 for an example). Our P_{STV} protocol consists of $k-1$ rounds, which themselves consists of the following procedures.

(1) **Finished?** (Algorithm 49.) From the candidate data structure, compute the number of candidates (apart from candidate 0) that got a seat or are still in the running. If this is equal to the number of available seats s, then mark as selected all the candidates that were still in the running.
(2) **Count votes.** (Algorithm 50.) For each ballot B, take the candidate in the first rank, and add the weight of the ballot to the number of votes S_i of this candidate. In MPC, this is done with a loop on all candidates i, and conditionally adding the weight of the ballot to S_i, depending on whether B_0 is equal to i.
(3) **Search for min and max.** (Algorithm 51.) Compute i and j the indexes such that $S_i = \max(s_k)$ and $S_j = \min(s_k)$. If the candidate i gets a seat, *i.e.* $S_i \geq q$, set e to 1, c to i and the transfer ratio t to $(S_i - q)/S_i$. Otherwise, the candidate j will be eliminated and set e to 0, c to j, and t to 1.
(4) **Select, delete, transfer.** (Algorithm 52.) Mark the candidate number c as selected or eliminated: set $H_c = 0$, and if e is 1, then set $W_c = 1$. Also, for all ballots, remove the candidate c. This is done in one pass over the list of preferences of each ballot. At the time, remember for each ballot if c was in first position. For each ballot for which c was in first position, multiply its weight by the transfer value t.

At the very end, the vector W is decrypted into w, and the elected candidates i are such that $w_i = 1$.

Algorithm 49: Finished

Require: s, t, H, W, where t the round index (initially 0)
Ensure: Modify W
$N^{\text{bits}} = \mathtt{Aggreg}^{\text{bits}}(W_1, \cdots, W_k)$ (* bit-wise encryption of the number of selected candidates *)
$F = \mathtt{EQ}_k(N, s-k+t)$ (* when one of the operand is known in the clear, the procedure is cheaper *)
for $i = 1$ *to* k *(in parallel)* **do**
    $H_i = \mathtt{CSZ}(H_i, F)$
    $W_i = \mathtt{Select}(W_i, H_i, F)$

In STV, the procedure should stop when s candidates have been selected or when the number of candidates that remain is equal to the number of seats that remain. If s candidates are selected, adding some additional rounds will not modify the result as it is not possible for $(s+1)$ or more candidates to reach the quota (*i.e.* no subsequent selection would occur, therefore W will no longer be modified). However, if the number of candidates that remain is equal to the number of seats that remain, adding an additional round may lead to an elimination if no candidate reach the quota, so it is important to select all candidates right away. Since a candidate is either selected or eliminated each round, the round index t is such that the number of candidates that remain is equal to $k - t$. Moreover, the number of seats that remain is simply s minus the number of selected candidates. So we compute the latter (say n') and we test if $n' = s - k + t$, which is equivalent to $n' = s - k + t$. Rewriting the test this way allows a slightly more efficient equality test as one operand is known.

Note that we do not want to reveal *when* the procedure stop so, in MPC, the procedure should actually continue. In what follows, we explain why the result (the decryption of W) will not be modified if subsequent iterations are run. First, once this test returns true, n' becomes s and since $t < k$ (there are $k-1$ rounds), the test can no longer return true, so this modification will occur only once. Afterwards, only selection and elimination would occur and since selecting a candidate which is already selected does not change anything, the outcome is not altered by the subsequent rounds.

Algorithm 50: CountVotes

Require: B, S
Ensure: Modify S
for $i = 1$ *to* n *(in parallel)* **do**
    for $j = 1$ *to* k *(in parallel)* **do**
        $C_{i,j} = \mathtt{EQ}_k(B_i[0], j)$
for $j = 1$ *to* k *(in parallel)* **do**
    ${S_j}^{\text{bits}} = 0$
    for $i = 1$ *to* n *(tree-based parallelisation is possible)* **do**
        ${S_j}^{\text{bits}} = \mathtt{Select}^{\text{bits}}(S_j, \mathtt{Add}^{\text{bits}}(S_j, V_i), C_{i,j})$

In the procedure CountVotes, we mention that tree-based parallelisation is possible. Indeed, it is possible to compute all $\mathtt{Select}^{\text{bits}}(0, V_i, C_{i,j})$ in parallel, then to add them together using a tree-based algorithm. Hence the communication cost of this step is $O(\log(n)\mathtt{Add})$, where $\mathtt{Add}$ is the communication cost of an addition.

The last two procedures, namely SearchMinMax, and SelectDeleteTransfer are self-explanatory.

Algorithm 51: SearchMinMax

Require: S, q
Ensure: $D, C^{\text{bits}}, T^{\text{bits}}$, where

- D is an encryption of a bit d ($d = 1$ for a selection, 0 for an elimination)
- C^{bits} is a bit-wise encryption of the index of some candidate (with $\lceil \log(k+1) \rceil$ bits)
- T^{bits}, is a bit-wise encryption of the transfer coefficient (with $r + 1$ bits)

```
_, M^bits, I^bits, J^bits = MinMax^bits(S_1^bits, ··· , S_k^bits)
Δ^bits, D = SubLT_k(M^bits, q), D=Not (D)
T^bits = Div(Δ^bits, M^bits)
T^bits = Select^bits(1, T^bits, D) (* use a bit-wise encryption of 1 *)
C^bits = Select^bits(I^bits, J^bits, D)
Return D, C^bits, T^bits
```

Algorithm 52: SelectDeleteTransfer

Require: $D, C^{\text{bits}}, T^{\text{bits}}, W, H, B$
Ensure: Modify W, H, B

```
for i = 1 to k (in parallel) do
    Z = EQ_k(C^bits, i)
    H_k = CSZ(H_k, Z)
    W_k = Select(W_k, Enc(1), Z)
for i = 1 to n (in parallel) do
    A = EQ(B_i[0], C^bits)
    F = A
    for j = 0 to k − 1 do
        B_i[j] = Select^bits(B_i[j], B_i[j + 1], F)
        Z = EQ(B_i[j + 1], C^bits)
        F = FZ/CSZ(F, Z) (* f = 1 iff the candidate c has been found in the list *)
    B_i[k] = Select^bits(B_i[k], 0, F) (* use a bit-wise encryption of 0 *)
    V_i = Select^bits(V_i, Mul^bits(V_i, T^bits), A)
```

G.3 Complexity analysis

Naive approach

Recall that k is the number of candidates, n the number of voters, s the number of seats, $m = \lceil \log(m+1) \rceil$ and r the precision in terms of binary places. First, assume that we use the naive version for each algorithm.

The complexity of `Finished` can be derived directly from Figure 6. Since we use $\texttt{Aggreg}^{\text{bits}}$ with k operands, one `EQ` for two operands of size $\log k$ and $2k$`CSZ`, the complexity of this step is $(5k + \log k)$`CSZ` in terms of computation and transcript size, and $((\log k)^2 + \log\log k + 2)$ rounds in terms of communications. (For simplicity we will only keep the leading terms, here $5k$ and $(\log k)^2$.)

The complexity of `CountVotes` can also be derived from Figure 6. There are nk calls to `EQ` for inputs of size $\log k$ and nk calls of $\texttt{Add}^{\text{bits}}$ and $\texttt{Select}^{\text{bits}}$ for inputs of size $(m+r)$. Therefore the cost is $nk(\log k + 3(m+r))$`CSZ` in terms of computation and transcript size. However, as a tree-based parallelisation is possible, the communication cost is about $2(m+r)m$ rounds, as $m \approx \log n$.

The complexity of `SearchMinMax` is also obtained from Figure 6. As there are k operands of size $m+r$, `MinMax` costs $8k(m+r)$`CSZ` in terms of computation and transcript size, and $2(m+r)\log k$ rounds of communication. The remaining of the procedure consists (mainly) of a call to `Div` and `SubLT` (the two `Select` cost $O(\log k)$ and $O(r)$ in computations, and 1 round each). Overall, the cost of this step is about $(m+r)(3r+8k)$`CSZ` in terms of computation and transcript size and $2(m+r)(r+\log k)$ rounds of communication.

In `SelectDeleteTransfer`, there are k calls to $\texttt{EQ}_k$ which costs $\log k$`CSZ` each (the subsequent `CSZ` and `Select` use 1`CSZ` each. This part is negligible in terms of both computations and communications ($O(\log\log k)$ rounds). Afterwards, there are nk calls to `EQ` and $\texttt{Select}^{\text{bits}}$ for inputs of size $\log k$, which accounts to $3nk\log k$`CSZ` in terms of computation, and $k \log\log k$ rounds of communication. Finally, we multiply

- Let k be the number of candidates and $c_0, \cdots, c_k$ be the $k+1$ trivial bitwise encryptions of $0, \cdots, k$ with $\lceil \log(k+1) \rceil$ bits.
- Let $u_1, \cdots, u_b$ be the $b \leq k$ candidates selected by the voter, in this order.
- The voter chooses a permutation σ so that $\sigma(i) = u_{i+1}$ for all $0 \leq i < b$, and $\sigma(b) = 0$.
- She shuffles $c_0, \cdots, c_k$ with σ to obtain $c'_0, \cdots, c'_k$ and produces a proof of a shuffle π^{Shuffle}.
- The ballot is $(c'_0, \cdots, c'_k), \pi^{\mathsf{Shuffle}}$.

Figure 16: The vote procedure in STV

two inputs of size r and $m+r$ and use $\mathsf{Select}^{\text{bits}}$ for inputs of size $(m+r)$, which accounts to $3nr(m+r)\mathsf{CSZ}$ in terms of computation and transcript size and $2r(m+r)$ rounds. Overall, the complexity is about $3n(k \log k + r(m+r))\mathsf{CSZ}$ in terms of computation and transcript size, and $(k \log\log k + 2r(m+r))$ rounds.

Overall, since there are $k-1$ rounds, the leading terms of the complexity are

- $nk^2(4\log k + 3(m+r)(r+1))\mathsf{CSZ}$ in terms of computation and transcript size,
- $k(2(m+r)(m+2r+\log k) + k \log\log k)$ rounds of communications.

Advanced approach

The complexity of our algorithm is satisfying in terms of computations: recall from Section 6 that we aim for $O(nk^2)$ operations; and the $\log k$ and $(m+r)$ terms seems unavoidable as they are the bitsize of some operands. However, the number of rounds is quadratic in m, r and k. While m, as the logarithm of n, is not expected to grow too much, the strong dependency in k and r can be problematic. In what follows, we use the arithmetic of Section C.3 to explain how to avoid this quadratic number of rounds. For this purpose, it is crucial to identify which processes need a quadratic number of rounds. From the analysis above, we identify four terms which contribute to this.

- In `CountVotes`, we use the associativity of the addition to sketch a tree-based parallelisation of the loop which leads to $2(m+r)m$ rounds of communications. To mitigate this quadratic cost, we can use Algorithm 16 for the addition instead of Algorithm 11. This allows to perform the same step in $2m\log(m+r)$ rounds instead, but requires $\frac{3}{2}nk^2(m+r)\log(m+r)\mathsf{CSZ}$ instead of $2nk^2(m+r)\mathsf{CSZ}$.
- In `SearchMinMax`, the computation of the transfer coefficient implies a division, which leads to a quadratic number of rounds $2(m+r)r$. By replacing `SubLT` calls by the equivalent Unbounded Fan-in composition (the subtraction can be obtained similarly with the same complexity), the division can be performed in $2r\log(m+r)$ rounds, but the complexity increases slightly (it becomes $\frac{3}{2}r(m+r)\log(m+r)\mathsf{CSZ}$ instead of $3r(m+r)\mathsf{CSZ}$). Note that the complexity of this phase in terms of computation is still negligible compared to the rest of the algorithm.
- In `SelectDeleteTransfer`, the multiplications can all be computed in parallel, but each still requires a quadratic number $2r(m+r)$ of rounds. Just as above, using Algorithm 16 instead of the naive $\mathsf{Add}^{\text{bits}}$ allows to reduce the number of rounds to $2r\log(m+r)$, but the computation costs increases from $3nr(m+r)\mathsf{CSZ}$ to $\frac{3}{2}nr(m+r)\log(m+r)\mathsf{CSZ}$.
- In `SelectDeleteTransfer` again, there is a for loop in k which imposes a round complexity of $O(k\log\log k)$ (testing the equality of two integers of $\log k$ bits takes $\log\log k$ rounds). As the procedure is repeated $k-1$ times, this leads to a quadratic number of rounds in k. Once again, we can use the strategy from Section C.3 to solve this problem. First, compute all equality tests in parallel (denote the result $b_0, b_1, \cdots, b_k$. Then use an Unbounded Fan-in circuit to compute all the prefixes $b_0, b_0 \vee b_1, \cdots, b_0 \vee \cdots \vee b_k$. (Since the operation $\vee$ is associative, the same technique can be applied.) Finally for all i in parallel, compute the updated $B_i[j]$ as $\mathsf{Select}^{\text{bits}}(B_i[j], B_i[j+1], b_0 \vee \cdots \vee b_j)$, where $B_i[k+1]$ is a bit-wise encryption of 0 for all i. This time the number of rounds decreases to $k\log k\log\log k$ (from $k^2\log\log k$), while the computation cost increases slightly (from $3nk^2\log k\mathsf{CSZ}$ to $\frac{7}{2}nk^2\log k\mathsf{CSZ}$). Note that interestingly, the communication cost of this step becomes negligible before the aggregation process in the `Finished` procedure, which was negligible in the naive approach.

Using the modifications sketched above, we arrive to a good communication / computation trade-off: the impact on the computation is minimal, but the number of rounds is no longer quadratic in any variable.

G.4 STV, the bottom-line

Finally, we recap all that is necessary to use our tally-hiding protocol for STV. First, to submit a ballot, a voter can use the `vote` procedure that is detailed in Figure 16. It simply consists of shuffling a public representation of the k candidates to obtain the desired ordering. Since the voter may not rank all candidate, a dummy candidate 0 is added and means that the subsequent candidates should not be taken into account. This way all ballots have the same size, even if they do not rank the same number of candidates.

To verify a ballot, an auditor can simply check the zero knowledge proof of a shuffle. To tally a list of ballots B, the authorities use the protocol P_{STV} described in Section G.2.

Part II: Security in the SUC-framework.

In this appendix, we give all the notions that are necessary to establish the security of our MPC toolbox as well as our tally-hiding protocols. This begins with the SUC security framework that we introduce in Appendix H. In Appendix I, we prove that our toolbox is SUC-secure. In Appendix J.3, we deduce that the resulting electronic voting system is private and we address verifiability in Appendix J.4.

H A SECURITY FRAMEWORK FOR OUR MPC PROTOCOLS

H.1 Introduction to the framework

To analyze the security of our MPC protocols, we use the composition framework from [17], which is a Simpler version of the Universally Composable framework (SUC). Although less expressive than the more general UC framework [16], it is sufficient to analyze standard MPC protocols, and it is shown that protocols secure in the SUC framework are also secure in the UC framework. In the SUC framework, the participants of a protocol are modeled as Interactive Turing Machines (ITM) which have input / output tapes, a random tape, a working tape and some input / output communication tapes. Two ITM A and B can interact with each other if A (resp. B) has an output communication tape which share the same name as an input communication tape of B (reps. A). A *process* is simply a concurrent execution of several connected participants. It can invoke several sub-processes in parallel, in order to execute some sub-protocols. In the SUC framework, the participants of the sub-processes are the same as the participants of the main process. To invoke a sub-process, they simply allocate enough space in their memory and run the corresponding algorithm. They may run several sub-protocols in parallel, using time-sharing.

We analyze the security of such a process against a malicious and static adversary, which can corrupt some parties, but only before the execution of the process. Corrupted parties can be impersonated by the adversary and give away any secret that they have. In addition, the adversary has a full control over the communication network. It can read, block and deliver messages at will. However, we consider ideally authenticated messages, meaning that the adversary cannot forge, change or replay a message sent by an honest party. To model this, we consider that the participants can interact with a router in a star network. In addition of the adversary, the SUC framework considers another adversarial PPT, the environment $\mathcal{Z}$. It serves as an "interactive distinguisher" and interacts with the adversary.

The security of the process is guaranteed by a comparison with an ideal process, in which each party hands over their inputs to a trusted party which honestly performs the desired computation. However, the corrupted parties may send a different input (resp. output) than their real input (resp. output), and the adversary can block or delay the communication with the trusted party. Intuitively, a protocol is SUC-secure if, for all adversary in the real process, there exists a simulator in the ideal process such that no PPT environment can tell whether they are interacting with the adversary in the real process or with the simulator in the ideal process.

H.2 Secure functionality computation

We now give a more formal description of the framework. For this purpose, we suppose that there is a fixed number a of participants defined by a set of ITM P. Each participants has a single input and output communication tape, and interacts with a router, which in turn interacts with the adversary. The adversary interacts with the router and the environment. The adversary can corrupt a subset C of participants of size at most t, where $t < a$ is some threshold. Non-corrupted participants are honest and follow the protocol, while corrupted participants are fully impersonated by the adversary and give away any secret that they have. In the real process, the participants, the environment, the adversary and the router interact as follows.

The environment. Upon activation, the environment $\mathcal{Z}$ can write on the input tapes of each participant, read their outputs and send a message to the adversary, which is activated next.

Participants. Upon activation, an honest participant reads its input and input communication tapes. It runs an algorithm which is specified by the protocol and may write on its output tape. It then submits any number of queries $\mathtt{send}(i, j, m)$ to the router, where i is the sender, j the receiver and m the message. The router is activated next.

The adversary. Upon activation, the adversary can read the memory of the router, write a message to the environment, read the tapes of a corrupted participant or write on their output tape. Finally, it can choose one of the following.

- Send a query $\mathtt{deliver}(i, j, m)$ to the router, which is activated next.
- Activate the environment.
- Activate a participant.

The router. Upon activation, the router look for send queries. For each send query $\mathtt{send}(i, j, m)$, it checks that i is consistent with the sender of the query and that j is another participant. If so, it stores (i, j, m) in memory and the adversary is activated next. If there is no valid send query, the router looks for a deliver query (i, j, m) and checks that either i is corrupted (in which case the message is delivered to j) or (i, j, m) is stored in memory (in which case the message is delivered to j and one copy of (i, j, m) is erased from memory. If a message is delivered this way, the receiver is activated next). Otherwise, the adversary is activated.

The process terminates when $\mathcal{Z}$ writes an output on its output tape. We denote $\mathsf{REAL}_{P,\mathbb{A},\mathcal{Z}}(\kappa, z)$ this output, where κ is a security parameter and z is an arbitrary auxiliary input. For the ideal process, we consider a trusted party T which is also modeled as an ITM, and can interact with the router and the adversary. The trusted party performs an algorithm which is specified by the protocol and which aims to realize some ideal functionality. The participants, the environment, the adversary, the router and the trusted party interact as follows.

Participants. Upon activation, an honest participant i looks for any new input from T on its communication tape and copies it on its output tape. It also reads any new input I on its input tape and sends a query $\mathtt{send}(i, T, s_{id}||I)$ to the router, where s_{id} is the number of send query that the honest participant sent before. It serves as a session identifier, so that different queries made by the same participant are handled independently by T. The router is activated next.

The trusted party. The trusted party interacts with both the router and the adversary and hence has two input (and output) communication tapes. Upon activation, the trusted party looks for a new message (i, T, I) in its router (resp. adversary) input communication tape, perform some local computations and may send any query $\mathtt{send}(T, j, O)$ to the router or answer directly to the adversary. The router is activated next.

The router. The router behaves the same as in the real process, except that is dos not let the adversary read the messages exchanged between a participant and the trusted party. (The adversary still knows that a message was sent, knows the sender and the receiver, and can still decide when to deliver it.)

The adversary. Same as in the real process, except that it can write directly on the communication tape of the trusted party.

The environment. Same as in the real process.

We denote $\mathtt{IDEAL}_{T,\mathcal{S},\mathcal{Z}}(\kappa, z)$ the output of the environment in the ideal process, when it interacts with the adversary $\mathcal{S}$.

Definition H.1 (Secure computation [17]). Let P be a protocol, T some trusted party. We say that P *securely computes* T if, for all PPT $\mathbb{A}$, there exists a PPT $\mathcal{S}$ such that, for all PPT $\mathcal{Z}$, there exists a negligible function μ such that for all κ and all z polynomial in κ,

$$|\Pr(\mathtt{IDEAL}_{T,\mathcal{S},\mathcal{Z}}(\kappa, z) = 1) - \Pr(\mathtt{REAL}_{P,\mathbb{A},\mathcal{Z}}(\kappa, z) = 1)| \leq \mu(\kappa).$$

Note that in [16], Canetti defines the hardest adversary to simulate, which is the *dummy* adversary $\mathcal{D}$. When activated, the dummy adversary simply forwards its view to the environment and activates it. The dummy adversary can also handle requests of the form $\mathtt{deliver}(i, j, m)$ from the environment. On such requests, it forwards them to the router. Similarly, it can handle requests of the form $\mathtt{write}(i, m)$ by writing message m in the output tape of corrupted party i. Finally, it can handle $\mathtt{activate}(i)$ requests by activating participant i. It is shown that if there exists a simulator for the dummy adversary, then there exists a simulator for all adversary.

H.3 The composition theorem

In the previous section, we explained the notion of security from [17]. This notion is very convenient when it comes to MPC protocols because of its modularity. For example, suppose that we want to evaluate an algorithm g which can be expressed as the composition of m algorithms $f_1, \cdots, f_m$. Suppose that, for each algorithm f_i, we have an MPC protocol ρ_i that securely computes f_i. Then it is natural to construct a protocol $P^{\rho_1,\cdots,\rho_m}$ which will in turn invoke the sub-protocols $\rho_1, \cdots, \rho_m$ so as to evaluate g. For convenience, we denote $f = f_1, \cdots, f_m$ and $\rho = \rho_1, \cdots, \rho_m$. We restrict ourselves to a fixed number m of functions and protocols, with a polynomial number of sub-protocol invocations. Compared to the real protocol, the composed protocol is similar, except that participants can invoke the sub-protocols in addition to the other actions.

We also define a f-hybrid process in which the sub-protocols are replaced by calls to the appropriate trusted party: just as in the ideal process, the participants can query a trusted party to evaluate a function in f, but the query does not necessarily consists of the inputs of the participant. In addition, the participant proceeds with the output of the trusted party according to the protocol and does not simply copy it on its output tape. For more details, see [17]. In the hybrid process, the output of the environment $\mathcal{Z}$ interacting with an adversary $\mathbb{A}$ for a protocol P with ideal calls to the functions f is denoted $\mathtt{HYBRID}^{f}_{P,\mathbb{A},\mathcal{Z}}(\kappa, z)$.

Definition H.2 (Secure hybrid computation). Let $m \geq 1$, g and $f = f_1, \cdots, f_m$ be some functions and P be a protocol in the f-hybrid model. We say that P securely computes g in the f-hybrid model if, for all PPT $\mathbb{A}$, there exists a PPT $\mathcal{S}$ such that, for all PPT $\mathcal{Z}$, there exists a negligible function μ such that for all κ and all $z \in \{0,1\}^*$ of polynomial length,

$$|\Pr(\mathtt{IDEAL}_{g,\mathcal{S},\mathcal{Z}}(\kappa, z) = 1) - \Pr(\mathtt{HYBRID}^{f}_{P,\mathbb{A},\mathcal{Z}}(\kappa, z) = 1)| \leq \mu(\kappa).$$

We now formulate the composition theorem as Theorem H.3.

Theorem H.3 (The composition theorem [17]). *Let $m \geq 1$ be some fixed integer, $\rho = \rho_1, \cdots, \rho_m$ be protocols and $f = f_1, \cdots, f_m$ and $g_1, \cdots, g_m$ be some functions. Suppose that for all i, ρ_i securely computes f_i in the g-hybrid model. Then, for all protocol P in the f-hybrid model, the composed protocol P^ρ obtained by replacing calls to f by an invocation of ρ is such that for all PPT $\mathbb{A}$, there exists a PPT $\mathcal{S}$ such that for every PPT environment $\mathcal{Z}$, there exists a negligible function μ such that for all κ and $z \in \{0,1\}^*$ of polynomial length,*

$$|\Pr(\mathtt{HYBRID}^{g}_{P^\rho,\mathcal{S},\mathcal{Z}}(\kappa, z) = 1) - \Pr(\mathtt{HYBRID}^{f}_{P,\mathbb{A},\mathcal{Z}}(\kappa, z) = 1)| \leq \mu(\kappa).$$

H.4 Persistent memory in the SUC framework

In the SUC framework, a same participant can run several sub-protocols and take part in different sub-processes in parallel. It is reasonable to assume that it has access to some long term, persistent memory. For this reason, we place ourselves in the $\mathcal{F}_M$-hybrid model, where $\mathcal{F}_M$ is defined in Figure 17. In this hybrid model, a trusted party F_M updates a hashmap $\mathtt{value}_i$ for each participant i. In the remaining of the section, whenever a participant uses a variable which is neither in its local memory nor in any of its tapes, we consider that it sends a $\mathtt{get}$ query to F_M. If the variable is not set, the participant would run a dedicated algorithm to decide the initial value and send a $\mathtt{set}$ query to F_M.

The typical use of this ideal functionality is to get access to the secret shares, the public encryption key and the public commitment to each participant's share. Note that to avoid race conditions, an honest participant will never send a `set` query twice for the same key *var*.

- On message $(i, F_M, \texttt{set}(var, val))$ from participant i:
  - If $var \notin \texttt{value}_i$, adds an entry of value val with key var in the hashmap $\texttt{value}_i$.
  - Otherwise, replace the value associated with key var by val.
  - Reply with message $(F_M, i, \texttt{set}(var, val))$.
- On message $(i, F_M, \texttt{get}(var))$ from participant i:
  - If $var \in \texttt{value}_i$, reply with message $(F_M, i, var||\texttt{value}_i(var))$.
  - Otherwise, reply with message $(F_M, i, var||\perp)$.

Figure 17: The ideal memory access

H.5 Restricted I/O behavior

In the SUC framework, security is assured against an environment which can choose the inputs of the participants and read the outputs. Such an adversary is too strong in the context of electronic voting, where the inputs of the MPC protocols do not come from the wild but are rather given as the output of another protocol or with a ZKP that they have the correct format. For instance, the `CGate` protocol expects the inputs to be encryptions of 0 or 1, so that the SUC-security of the protocol is extremely difficult to prove against an environment which is able to give inputs which are not encryptions of 0 or 1. In [33] (Section 3.5), Nielsen considers a slightly weakened framework where we quantify over environments which respect some I/O restrictions. It is shown that we can restrict our analysis to environments which chooses the inputs of the party from any decidable language (in our case, the ciphertexts provided as input must be encryptions of 0 or 1), and proved that the composition theorem still holds for such restricted environments (see Theorem 3.5 of [33]). In what follows, we only consider environments which choose encryptions of 0 or 1 as inputs, and which give the same inputs to all participants. Such environements are said *restricted*.

I THE SUC-SECURITY OF OUR BUILDING BLOCKS

To prove the security of our protocols in the SUC framework, we idealized some of their natural features. In Section H.4, we presented our model for persistent memory, which uses a relevant ideal functionality $\mathcal{F}_M$. In Section H.5, we introduced the notion of restricted environments. Now, we are ready to prove the security of our building blocks. As explained in Section H.3, the SUC framework is convenient for its compatibility with composition. In our toolbox, most of our protocols are based on the `CGate` protocol and can be expressed as a sequential composition of different conditional gates. Therefore, we first prove in Section I.1 that `CGate` is SUC-secure. Afterwards, we extend the SUC-security to the other MPC primitives in Section I.2, using the composition theorem. Finaly, we use this result to establish the SUC-security of our tally-hiding protocols in Section I.3.

I.1 The SUC-security of `CGate`

First of all, we cast the `CGate` protocol into the SUC framework. For this purpose, we implicitly use the $\mathcal{F}_M$-hybrid model as explained in Section H.4. This assumes that the Distributed Key Generation (DKG) has already been performed in a secure way. For instance, the protocols of [45] or [5] compute a UC-secure DKG. In our model, we not only assume that the DKG created a share for each participant, but that it also created a public encryption key pk and a public commitment to the share of each participant. The public commitment of participant i is denoted h_i and is used to create and verify the ZKP of correct partial decryption. We assume that each (honest) participant is given the same view of $pk, h_1, \cdots, h_a$, and can access to any of these values with a memory access.

Since the secret share and the public elements are not given as input by the environment, the only remaining input fed to the participants are the ciphertexts. In the ideal process, the participants forwards their inputs to the ideal functionality and nothing else. Consequently, we consider that the ideal functionality can query the value `share` to F_M for each participant. The honest participants are therefore required to store their secret share in the variable `share`.

In addition, we use the Programmable Random Oracle Model, which means that we consider an ideal functionality $\mathcal{F}_{\mathcal{RO}}$ which is queried by participants instead of the local computation of hash values (this is the random oracle model). In addition, the simulator is allowed to see the queries to $\mathcal{F}_{\mathcal{RO}}$, and to answer them instead of $\mathcal{F}_{\mathcal{RO}}$ (hence the programmable adjective).

Also, we restrict ourselves to environments which only choose encryptions of 0 or 1 as inputs, and give the same inputs to each participant, as explained in Section H.5.

We now give the complete specification of the `CGate` protocol, which is unfold from Algorithm 1. In this protocol, each participant i has an input (X, Y) and stores the variables $(X_j, Y_j, \pi_j, b_j, w_j, c_j, A_j, B_j)_{j=0}^{a}$ as well as y_a and r. Initially, $X_j = Y_j = \pi_j = w_j = c_j = A_j = B_j = y_a = \perp$ and $b_j = 0$ for all j, except that $X_0 = X$, $Y_0 = E_{-1}Y^2$ and $b_0 = 1$. The boolean r is initially 0. When participant i is activated, it performs the algorithm described in Figure 18.

Also, we consider a trusted party $T_{\texttt{CGate}}$. When activated, it performs the algorithm described in Figure 19.

- **Rounds of communication.** For all new message of the form $(j, i, X||Y||\pi)$ in the input communication tape, do:
  - If $X_j = Y_j = \pi_j = \perp$, set them to X, Y and π respectively, otherwise ignore the message.
  - If $X_{j-1} \neq \perp$, check π using $X_{j-1}, Y_{j-1}, X_j, Y_j$. If the proof valid, set b_j to 1.
  - If $b_{i-1} = 1$, compute X_i, Y_i, π_i as in Algorithm 1, set b_i to 1 and make a send query $(i, j, X_i||Y_i||\pi_i)$ to all $j \neq i$. Recall that π is a ZKP that there exists $s \in \{-1, 1\}$ s.t. X_i (resp. Y_i) is a reencryption of X_{i-1}^s (resp. Y_{i-1}^s).
- **Reencryption.** If $X_a \neq \perp$, if $b_j = 1$ for all j and if $r = 0$, do:
  - Choose two random α and β, compute $A_i = \mathsf{Enc}(0, \alpha)$, $B_i = \mathsf{Enc}(0, \beta)$ and $c_i = \mathsf{hash}(pk||X_a||Y_a||A_i||B_i)$. Make a send query (i, j, c_i) for all j. Set r to 1.
  - For all message of the form (j, i, c) if $c_j = \perp$, set c_j to c.
  - When all c_j are received, compute $\pi^0(A_i, \alpha)$ and $\pi^0(B_i, \beta)$, two ZKP that A_i and B_i are encryptions of 0, and send $(i, j, c_1|| \cdots ||c_a||A_i||B_i||\pi^0(A_i, \alpha)||\pi^0(B_i, \beta))$ to all j
  - For all message of the form $(j, i, c_1^j|| \cdots ||c_a^j||A||B||\pi_A^0||\pi_B^0)$, if the ZKP are valid, if $c_i^j = c_i$ for all i, and if $c_j = \mathsf{hash}(pk||X_a||Y_a||A_j||B_j)$, set A_j to A and B_j to B.
  - When all A_j are set, compute $X' = X_a \prod_{i=1}^a A_i$ and $Y' = Y_a \prod_{i=1}^a B_i$.
- **Threshold decryption.** If X' and Y' are set, do:
  - Compute a partial decryption w_i of Y' with the secret share, as well as a proof of correct partial decryption π_i^{Dec}. Make a send query $(i, j, w_i||\pi_i^{\mathsf{Dec}})$ for all j.
  - For all new message of the form $(j, i, w||\pi^{\mathsf{Dec}})$, if $w_j = \perp$ and if the proof is correct, set w_j to w.
  - If $y_a = \perp$ and if there is at least $t+1$ $w_j \neq \perp$, set y_a to the plaintext which is obtained by combining the $t+1$ partial decryptions.
    * If $y_a \notin \{-1, 1\}$, terminate by writing $\perp$ in the output tape.
    * Otherwise, compute $Z = (XX'^{y_a})^{1/2}$ and output Z in the output tape.

Figure 18: Specification of the honest participant in CGate

- If there is a new message from the adversary in the input communication tape, it sends $(T_{\mathsf{CGate}}, i, \perp)$ to all participant i.
- If there is a new message of the form $(i, T_{\mathsf{CGate}}, s_{id}||X||Y)$ in the input communication tape, it looks for a session with the same session identifier s_{id}. If such a session exists, it queries the value s of share for i and adds X, Y, s to the FIFO queue q_i of the session. If there is no such session, it creates a new session with identifier s_{id}, creates a FIFO queue q_j for all participant j and adds X, Y, s to q_i. Note that if $s = \perp$, it ignores the query.
- If there is a session in which all queues are non-empty, it removes their first elements $(X_i, Y_i, s_i)_{i=1}^a$ and do the following.
  - If there is i, i' such that $X_i \neq X_{i'}$ or $Y_i \neq Y_{i'}$, make a send query $(T_{\mathsf{CGate}}, i, \perp)$ for all participant i.
  - Otherwise, decrypt X and Y using the shares, check that the respective plaintexts x and y belong to $\{0, 1\}$ and create a fresh, random encryption Z of xy with the public key pk. Make a send query $(T_{\mathsf{CGate}}, i, Z)$ for all i. If $x, y \notin \{0, 1\}$, make a send query $(T_{\mathsf{CGate}}, i, \perp)$ for all i.

Figure 19: Specification of T_{CGate}

We now prove Theorem I.4, under DDH assumption. Note that in Lemma I.2, we show that the ElGamal threshold encryption scheme is ZK-TCPA (as of Definition I.1) under DDH assumption.

Definition I.1 (ZK-TCPA). Let $\Pi = (\mathsf{DKG}, \mathsf{Enc}, \mathsf{PartDec})$ be a threshold encryption scheme. We say that Π is ZK-TCPA secure if, for all number of trustees $a \geq 2$, for all threshold $t < a$, for all set C of size at most t, there exists a simulator $\mathsf{Sim}_{t,a}$ such that there is no adversary $\mathbb{A}$ which can win game $\mathsf{Exp}^{ZK-TCPA}$ defined in Algorithm 53 with non-negligible advantage.

Lemma I.2. *Under DDH assumption, the exponential ElGamal encryption scheme with Shamir's secret sharing is ZK-TCPA secure.*

Proof. Recall that the exponential ElGamal encryption with public (g, pk) of a plaintext m consists of a couple $(g^r, g^m pk^r)$ with a random r. Recall that the decryption of a ciphertext (x, y) with the secret key sk is the discrete logarithm m of yx^{-sk}, which can be efficiently computed when m is small. Recall that in the ElGamal setting, Shamir's secret sharing with parameters κ, t, a consists of the following steps.

- Chooses a group G of prime order $q = \Theta(2^\kappa)$ and a generator g.
- Chooses a random polynomial $Q \in \mathbb{Z}_q[X]$ of degree t.
- For all $i \in [0, a]$, let $s_i = Q(i)$ and $h_i = g^{s_i}$.
- Return $g, s_0, h_0, s_1, h_1, \cdots, s_a, h_a$.

Also, recall that the partial decryption of a ciphertext $C = (x, y)$ with the share s is simply $w = x^s$.

Finally, by abuse of notations, we will exhibit simulators which take as inputs a ciphertext $Y = (x, y)$, a cleartext m, a set C and the partial decryptions $(w_i)_{i \in C}$ instead of the secret shares $(s_i)_{i \in C}$. Clearly a simulator which has the secret shares and the ciphertext $Y = (x, y)$ can

Algorithm 53: $\mathtt{Exp}_{\mathbb{A},\Pi}^{ZK-TCPA}(\kappa)$

Require: κ, t, a, C
$sk, pk, s_1, h_1 \cdots, s_a, h_a := \mathtt{DKG}(\kappa, t, a)$
$b \in_r \{0, 1\}$
$m := \mathbb{A}_1(\kappa, g, pk, (h_i)_{i=1}^a, (s_i)_{i \in C})$
$Y := \mathtt{Enc}_{pk}(m)$
$S_1 := (\mathtt{PartDec}(Y, s_i))_{i \notin C}$
$S_0 := \mathtt{Sim}_{t,a}(Y, m, C, (s_i)_{i \in C})$
$b' := \mathbb{A}_2(Y, S_b)$
Return $b == b'$

compute the partial decryptions $w_i = x^{s_i}$ for all $i \in C$, so this is not a loss of generality. In this proof, we implicitly incorporate this tweak when we refer to the ZK-TCPA game.

First, we describe the simulator $\mathtt{Sim}_{t,a}^t$ which simulates the partial decryptions when $|C| = t$. It first computes the Lagrange's coefficients $\lambda_{i,j}^S = \prod_{k \in S \setminus \{j\}} \frac{i-k}{j-k}$ for all $i \in [1, a] \setminus C$ and $j \in S$, where $S = C \bigcup \{0\}$. Then, it outputs $w_i = \prod_{j \in S} (w_j)^{\lambda_{i,j}^S}$ for $i \in [1, a] \setminus C$, where $w_0 = y/m$. Remark that by Lagrange's interpolation, this simulator plays a perfect simulation of the partial decryptions.

Now, let $k \in [2, t]$. Suppose that for all set C of size k, there exists a simulator $\mathtt{Sim}_{t,a}^k$ such that no PPT adversary which can win the ZK-TCPA game with non-negligible advantage. As shown above, this is true for $k = t$. We construct $\mathtt{Sim}_{t,a}^{k-1}$ as follows. First, the simulator picks the smallest $i' \in [1, a] \setminus C$, sets $C' = C \bigcup \{i'\}$, sets $w_{i'}$ as a random group element and return $\mathtt{Sim}_{t,a}^k$'s output. Suppose that there exists C of size $k - 1$ and an adversary $\mathbb{A} = \mathbb{A}_1, \mathbb{A}_2$ which wins the ZK-TCPA game with non-negligible advantage.

We consider $\mathbb{B}$, an adversary for DDH. It gets a DDH challenge g_1, g_2, g_3, g_4, which is either a DDH tuple such that $\log_{g_1}(g_2) = \log_{g_3}(g_4)$, either a random tuple. $\mathbb{B}$ sets $g = g_1$, chooses a random secret key s_0 and computes $pk = g^{s_0}$. Afterwards, $\mathbb{B}$ chooses random shares s_i for $i \in C$, computes the corresponding $h_i = g^{s_i}$, picks the smallest $i' \in [1, a] \setminus C$ and sets $h_{i'} = g_2$. We denote $C' = C \bigcup \{i'\}$. Now, it picks a random subset $I \subset [1, a] \setminus C'$ of size $t - k$ and chooses a random group element h_i for $i \in I$. Let $h_0 = pk$ and $J = C' \bigcup I \bigcup \{0\}$. For $i \in [1, a] \setminus J$, $\mathbb{B}$ computes $h_i = \prod_{j \in J} (h_j)^{\prod_{k \in J \setminus \{j\}} \frac{i-k}{j-k}}$. It can now call $\mathbb{A}_1$ with inputs $(g, pk, h_1, \cdots, h_a, (s_i)_{i \in C})$ to get m. To compute a random ciphertext (x, y) of m, $\mathbb{B}$ sets $x = g_3$ and $y = mx^{s_0}$. Finally, it computes the partial decryptions $(w_i)_{i \notin C}$ by querying $\mathtt{Sim}_{t,a}^k$ with inputs (x, y), m and $(w_i)_{i \in C'}$. For this purpose, it computes $w_i = x^{s_i}$ for $i \in C$ and sets $w_{i'} = g_4$.

If g_1, g_2, g_3, g_4 is a random tuple, $\mathbb{B}$ played a perfect simulation of the ZK-TCPA game with simulator $\mathtt{Sim}_{t,a}^{k-1}$ to $\mathbb{A}$ with the set C, and hence wins with a non-negligible advantage. If g_1, g_2, g_3, g_4 is a DDH tuple, $\mathbb{B}$ played a perfect simulation of the ZK-TCPA game with simulator $\mathtt{Sim}_{t,a}^k$ to $\mathbb{A}$ with the set C', and hence wins with a negligible advantage (by recursion). Overall $\mathbb{B}$ wins with a non-negligible advantage. □

Lemma I.3. *Let $\Pi = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec})$ be a homomorphic, IND-CPA encryption scheme. Then there is no PPT adversary $\mathbb{A} = \mathbb{A}_1, \mathbb{A}_2$ which satisfies the two following properties, where the game G_{xy} is defined in Algorithm 56.*

- *For all κ, $G_{xy}(\mathbb{A}, \kappa) \neq \perp$.*
- *The advantage $|\Pr(G_{xy}(\mathbb{A}, \kappa) = 1) - 1/2|$ is non-negligible in κ.*

We refer by restricted the adversaries which satisfy the first property.

Proof. Let $\mathbb{A} = \mathbb{A}_1, \mathbb{A}_2$ be a restricted adversary for game G_{xy}. On random key pair sk, pk, we consider the random variable x_0, y_0, where $x_0 = \mathtt{Dec}_{sk}(X_0)$, $y_0 = \mathtt{Dec}_{sk}(Y_0)$ and $X_0, Y_0, P_1 = \mathbb{A}_1(pk)$. Note that since $\mathbb{A}$ is restricted, $x_0 \in \{0, 1\}$ and $y_0 \in \{-1, 1\}$. We denote $p_{0,\pm 1} = \Pr(x_0 = 0)$, $p_{1,1} = \Pr(x_0 = 1, y_0 = 1)$ and $p_{1,-1} = \Pr(x_0 = 1, y_0 = -1)$. For any adversary $\mathbb{B}$ and $a \in \{(0, \pm 1), (1, 1), (1, -1)\}$, $p(\mathbb{B}|e_a)$ the conditional probability that $\mathbb{B}$ wins G_{xy} when $x_0 = 0$ (for $a = 0, \pm 1$), $x_0 = 1$ and $y_0 = 1$ (for $a = 1, 1$) and $x_0 = 1$ and $y_0 = -1$ (for $a = 1, -1$). Finally, we denote $p(\mathbb{B})$ the probability that $\mathbb{B}$ wins G_{xy}. Hence, it follows that for all $\mathbb{B}$,

$$p(\mathbb{B}) = p_{0,\pm 1} p(\mathbb{B}|e_{0,\pm 1}) + p_{1,1} p(\mathbb{B}|e_{1,1}) + p_{1,-1} p(\mathbb{B}|e_{1,-1}).$$

We consider an adversary $\mathbb{B}_1$ for IND-CPA which proceeds as follows. First, $\mathbb{B}_1$ gets pk from the IND-CPA game and submits the plaintexts 0 and 1. The IND-CPA game gives a random encryption C of either 0 or 1. Then $\mathbb{B}_1$ calls $\mathbb{A}_1$ with input pk to get X_0, Y_0, P_1. It chooses two random exponents α, β and computes $Z_1 = (X_0/Y_0)^\alpha C^\beta E_1$ and Z_0, a random ciphertext. It chooses $\gamma \in \{0, 1\}$ at random and queries $\mathbb{A}_2(P_1, Z_\delta)$. If $\mathbb{A}_2$ guesses the correct value of γ, $\mathbb{B}_1$ outputs 0. Otherwise, it outputs 1.

Note that when $x_0 = y_0 = 1$, if C is an encryption of 0, then $\mathbb{B}_1$ plays a perfect simulation of G_{xy} to $\mathbb{A}$ and hence wins with probability $p(\mathbb{A}|e_{1,1})$. If C is an encryption of 1, then Z_1 and Z_0 are equally distributed and $\mathbb{A}_2$ guesses delta with probability $1/2$, therefore $\mathbb{B}_1$ wins with probability $1/2$. In all other cases (*i.e.* when $x_0 = 0$ and when $x_0 = 1$ while $y_0 = -1$), Z_1 and Z_0 are equally distributed and $\mathbb{B}_1$ wins with probability 1/2. In addition, under IND-CPA assumption, the overall probability that $\mathbb{B}_1$ wins is approximately 1/2. Consequently, we have

$$\frac{1}{2} \approx \frac{1}{2}p_{0,\pm 1} + \frac{1}{2}p_{1,1}\left(\frac{1}{2} + p(\mathbb{A}|e_{1,1})\right) + \frac{1}{2}p_{1,-1}.$$

Now, we consider an adversary $\mathbb{B}_0$ for IND-CPA which proceeds as follows. First, $\mathbb{B}_0$ gets pk from the IND-CPA game and submits the plaintexts 0 and 1. The IND-CPA game gives a random encryption C of either 0 or 1. Then $\mathbb{B}_1$ calls $\mathbb{A}_1$ with input pk to get X_0, Y_0, P_1. It chooses two random exponents α, β and computes $Z_1 = X_0^{\alpha} C^{\beta}$ and Z_0, a random ciphertext. It chooses $\gamma \in \{0, 1\}$ at random and queries $\mathbb{A}_2(P_1, Z_\delta)$. If $\mathbb{A}_2$ guesses the correct value of γ, $\mathbb{B}_0$ outputs 0. Otherwise, it outputs 1.

Note that when $x_0 = 0$, if C is an encryption of 0, then $\mathbb{B}_0$ plays a perfect simulation of G_{xy} to $\mathbb{A}$ and hence wins with probability $p(\mathbb{A}|e_{0,\pm 1})$. If C is an encryption of 1, then Z_1 and Z_0 are equally distributed and $\mathbb{A}_2$ guesses delta with probability 1/2, therefore $\mathbb{B}_0$ wins with probability 1/2. In all other cases (*i.e.* when $x_0 = y_0 = 1$ and when $x_0 = 1$ while $y_0 = -1$), Z_1 and Z_0 are equally distributed and $\mathbb{B}_0$ wins with probability 1/2. In addition, under IND-CPA assumption, the overall probability that $\mathbb{B}_0$ wins is approximately 1/2. Consequently, we have

$$\frac{1}{2} \approx \frac{1}{2}p_{0,\pm 1}\left(\frac{1}{2} + p(\mathbb{A}|e_{0,\pm 1})\right) + \frac{1}{2}p_{1,1} + \frac{1}{2}p_{1,-1}.$$

Finally, we consider an adversary $\mathbb{B}_{-1}$ for IND-CPA which proceeds as follows. First, $\mathbb{B}_{-1}$ gets pk from the IND-CPA game and submits the plaintexts 0 and 1. The IND-CPA game gives a random encryption C of either 0 or 1. Then $\mathbb{B}_{-1}$ calls $\mathbb{A}_1$ with input pk to get X_0, Y_0, P_1. It chooses two random exponents α, β and computes $Z_1 = (X_0 Y_0)^{\alpha} C^{\beta}$ and Z_0, a random ciphertext. It chooses $\gamma \in \{0, 1\}$ at random and queries $\mathbb{A}_2(P_1, Z_\delta)$. If $\mathbb{A}_2$ guesses the correct value of γ, $\mathbb{B}_{-1}$ outputs 0. Otherwise, it outputs 1.

Note that when $x_0 = 1$ while $y_0 = -1$, if C is an encryption of 0, then $\mathbb{B}_{-1}$ plays a perfect simulation of G_{xy} to $\mathbb{A}$ and hence wins with probability $p(\mathbb{A}|e_{1,-1})$. If C is an encryption of 1, then Z_1 and Z_0 are equally distributed and $\mathbb{A}_2$ guesses delta with probability 1/2, therefore $\mathbb{B}_{-1}$ wins with probability 1/2. In all other cases (*i.e.* when $x_0 = y_0 = 1$ and when $x_0 = 0$), Z_1 and Z_0 are equally distributed and $\mathbb{B}_{-1}$ wins with probability 1/2. In addition, under IND-CPA assumption, the overall probability that $\mathbb{B}_0$ wins is approximately 1/2. Consequently, we have

$$\frac{1}{2} \approx \frac{1}{2}p_{0,\pm 1} + \frac{1}{2}p_{1,1} + \frac{1}{2}p_{1,-1}\left(\frac{1}{2} + p(\mathbb{A}|e_{1,-1})\right).$$

By computing the average of the three equations, we have

$$\begin{aligned}\frac{1}{2} &\approx \frac{1}{3}\left(\frac{5}{4}(p_{0,\pm 1} + p_{1,1} + p_{1,-1}) + \frac{1}{2}(p_{0,\pm 1}p(\mathbb{A}|e_{0,\pm 1}) + p_{1,1}p(\mathbb{A}|e_{1,1}) + p_{1,-1}p(\mathbb{A}|e_{1,-1}))\right)\\ &= \frac{1}{3}\left(\frac{5}{4} + \frac{1}{2}p(\mathbb{A})\right).\end{aligned}$$

Consequently, we deduce that $p(\mathbb{A}) \approx 1/2$. □

Theorem I.4. *Under DDH assumption, the* CGate *protocol, used in the ElGamal setting, SUC-securely computes the* T_{CGate} *functionality in the* $\mathcal{F}_{\mathcal{RO}} - \mathcal{F}_M$*-hybrid model, with respect to environments which only set encryptions of* 0 *or* 1 *as inputs, and give the same input to all participants.*

Proof. We construct a simulator $\mathcal{S}$ which emulates the honest parties, the router and the $\mathcal{RO}$ functionality. The simulator will keep in memory a session id sid_i for all honest participant i. Initially, $sid_i = 0$ for all i. When an honest participant is activated, $\mathcal{S}$ increments its session id and checks if there is already a simulation running with this id. If so, $\mathcal{S}$ will have the participant join this session. Otherwise, $\mathcal{S}$ will create a new, independent simulation, with an independent copy of the ideal functionality. In what follows, we explain how each simulation is handled. Assuming that there is at least one honest party and one corrupted party (otherwise the result is trivial), we denote j_0 the last honest party (for all $i > j_0$, i is corrupted).

Before simulating the real process, $\mathcal{S}$ first forwards the queries of all honest participants (in the ideal process) to the ideal functionality to get the answer Z_f. Recall that the environment sets a common input X, Y to each participant, where X and Y are encryptions of $x, y \in \{0, 1\}$. Then, Z_f is simply a random encryption of xy. The simulator blocks the answer to all participant (it may deliver it later), except for one corrupted party, which allows $\mathcal{S}$ to get Z_f without it being written on any output tape. Once Z_f is known, $\mathcal{S}$ picks $y_a \in \{-1, 1\}$ at random and simulates the real process as follows.

Phase 1: Rounds of communication. Recall that in the real process, there is a first phase in which the participants each choose $s \in_r \{-1, 1\}$ and compute X_i (resp. Y_i), a reencryption of X_{i-1}^s (resp. Y_{i-1}^s), and a ZKP π_i that X_i (resp. Y_i) is well-formed, where $X_0 = X$ and $Y_0 = E_{-1}Y^2$.

Since this phase does not require a secret and does not cause the (honest) participants to write on their output tape, $\mathcal{S}$ can give a perfect simulation to this phase using the algorithm of the honest participants.

Phase 2: Reencryption. Recall that in the real process, there is a second phase in which X_a and Y_a are reencrypted into X' and Y'. Once again, the simulator can give a perfect simulation for all participants using their algorithm. However, it will have participant j_0 behave differently. For completion, we reproduce the five steps of the honest algorithm.

(1) Choose two random α and β, compute $A_i = \mathsf{Enc}(0, \alpha)$, $B_i = \mathsf{Enc}(0, \beta)$ and $c_i = \mathsf{hash}(pk||X_a||Y_a||A_i||B_i)$. Make a send query (i, j, c_i) for all j. Set r to 1.

(2) For all message of the form (j, i, c) if $c_j = \perp$, set c_j to c.
(3) When all c_j are received, compute $\pi^0(A_i, \alpha)$ and $\pi^0(B_i, \beta)$, two ZKP that A_i and B_i are encryptions of 0, and send $(i, j, c_1|| \cdots ||c_a||A_i||B_i||\pi^0(A_i, \alpha)||\pi$ to all j.
(4) For all message of the form $(j, i, c_1^j|| \cdots ||c_a^j||A||B||\pi_A^0||\pi_B^0)$, if the ZKP are valid, if $c_i^j = c_i$ for all i, and if $c_j = \mathsf{hash}(pk||X_a||Y_a||A_j||B_j)$, set A_j to A and B_j to B.
(5) When all A_j are set, compute $X' = X_a \prod_{i=1}^{a} A_i$ and $Y' = Y_a \prod_{i=1}^{a} B_i$.

For participant j_0, the simulator runs the two first steps honestly but, after that j_0 received all c_j (step 3) $\mathcal{S}$ looks back at the $\mathcal{RO}$ queries it answered. If, for all j, there exists a unique query of the form $pk||X_a||Y_a||A_j||B_j$ which was answered with c_j, $\mathcal{S}$ computes $A_{j_0} = (Z_f^2/X)^{1/y_a}/(X_a \prod_{i \neq j_0} A_i)$, computes a random encryption C of y_a and computes $B_{j_0} = C/(Y_a \prod_{i \neq j_0} B_i)$. It computes a perfect simulation of the ZKP π_A^0 and π_B^0 by computing the challenges in advance, running the simulator of the ZKP and answering any relevant subsequent query to the random oracle with the chosen challenges. Afterwards, $\mathcal{S}$ answers to any subsequent query to the $\mathcal{RO}$ with input $pk||X_a||Y_a||A_{j_0}||B_{j_0}$ with c_{j_0} and runs the last two steps honestly.

If there exists j such that no query of the form $pk||X_a||Y_a||A_j||B_j$ was answered with c_j, $\mathcal{S}$ simply runs the honest algorithm of j_0. Remark that in this case, participant j will fail (with overwhelming probability) to provide $pk||X_a||Y_a||A_j||B_j$ such that the random oracle answers with c_j, hence j_0 will get stuck and eventually, every honest participant will get stuck (we show below that every honest participant gets the same view of c_j). When every honest participant is stuck, the simulation is trivial since $\mathcal{S}$ only needs to block all messages from and to the honest participants. Consequently we assume that this does not happen, and therefore we have $X' = (Z_f^2/X)^{1/y_a}$ and $Y' = C$.

Phase 3: Threshold decryption. Recall that in the real process, there is a last phase in which the honest participants must compute a partial decryption of Y', and generate a ZKP of correct partial decryption. To simulate the partial decryptions of the honest participants, $\mathcal{S}$ uses the simulator $\mathsf{Sim}_{t,a}$ with the secret shares of the corrupted participants, the ciphertext Y' and the plaintext y_a, where $\mathsf{Sim}_{t,a}$ exists by ZK-TCPA assumption (see Definition I.1, and Lemma I.2 for a proof that the ElGamal threshold encryption scheme is ZK-TCPA secure under DDH assumption).

In what follows, we denote $p^{\mathcal{S}} = \Pr(\mathsf{IDEAL}(\mathcal{T}_{\mathsf{CGate}}, \mathcal{S}, \mathcal{Z}) = 1)$ and $p = \Pr(\mathsf{REAL}(\mathsf{CGate}, D, \mathcal{Z}) = 1)$, where D is the dummy adversary.

A few remarks.

Remark that when an honest participant j begins the reencryption phase, it means that it received some (X_i^j, Y_i^j, π_i) for all i, where π_i is a valid ZKP that there exists some $s_i \in \{-1, 1\}$ such that X_i (resp. Y_i) is a reencryption of $X_{i-1}^{s_i}$ (resp. $Y_{i-1}^{s_i}$). Hence, except with negligible probability, this proposition is true for all i and there exists some $s \in \{-1, 1\}$ such that X_a is a reencryption of X_0^s while Y_a is a reencryption of Y_0^s. In addition, the environment is restricted to giving encryptions of 0 or 1 as input so that Y is an encryption of $y \in \{0, 1\}$. Hence, by the homomorphic property of the encryption scheme, Y_0 is an encryption of $2y - 1 \in \{-1, 1\}$. Let $x \in \{0, 1\}$ such that X is an encryption of x. It follows that X_a is an encryption of sx while Y_a is an encryption of $s(2y - 1)$.

Now, remark that when a honest participant j sets a value to X' and Y', then, for all i, it received a message of the form $(i, j, c_1^j|| \cdots ||c_a^j||A^j||B^j||\pi_A^0||\pi_B^0)$ with $c_i^j = c_i$ for all i. In particular, it means that for all honest participant j, j' we have $c_i^j = c_i^{j'} = c_i$ for all i. Now, suppose that for two honest participants j_1 and j_2, there are two distinct local values $X_a^{j_1}$ and $X_a^{j_2}$ (resp. $Y_a^{j_1}$ and $Y_a^{j_2}$) for the variable X_a (reps. Y_a). Then, with overwhelming probability, we have

$$\mathsf{hash}(pk||X_a^{j_1}||Y_a^{j_1}||A_{j_2}||B_{j_2}) \neq \mathsf{hash}(pk||X_a^{j_2}||Y_a^{j_2}||A_{j_2}||B_{j_2}) = c_{j_2}.$$

Hence participant j_1 will not accept j_2's message. Since j_2 is honest, it will never again send to j_1 a message with the correct format - except in another independent session which will therefore use a distinct c_{j_2} with overwhelming probability. Hence j_1 is stuck and eventually all the honest participants will get stuck. Consequently, we can assume that X_a and Y_a are the same for all honest participant. Similarly, since the c_i's are the same for all honest participants, we can also assume that all participants received the same values for A_i and B_i, for all i. Hence they compute two common values X' and Y'.

Finally, remark that the only way for a honest participant to begin the reencryption part is that $X_a \neq \perp$. During the reencryption phase, the honest participants wait for all c_j to be received, therefore if one honest participant do not take part in the phase due to some asynchronism, the other participants will be stuck. In particular, if a corrupted participant i sends (X_i, Y_i, π) to an honest participant j, with an invalid ZKP (due to j's algorithm, i can only send one message of this form to j ; the others are ignored) the participant will not take part in the reencryption phase since $b_i \neq 1$, and therefore all honest participants will get stuck. In this case the simulation is trivial, so that we can assume that all ZKP π sent by the corrupted participants are valid.

Indistinguishability.

Now, let $\mathcal{Z}$ be a restricted environment such that $p - p^{\mathcal{S}}$ is non-negligible. Without loss of generality, we consider that $\mathcal{Z}$'s output is always either 0 or 1. We consider game G_1 defined by Algorithm 54, where DKG is an ideal ElGamal DKG which uses Shamir's secret sharing. We show that there exists a restricted adversary $\mathbb{A}^1$ which wins G_1 with non-negligible advantage, where, by restricted, we mean that in instance of G_1 with $\mathbb{A}_1$, $x_0 \in \{0, 1\}$, $y_0 \in \{-1, 1\}$ and π is valid.

For this purpose, we consider the adversary $\mathbb{A}^1$ which emulate the ideal functionalities. It interacts with $\mathcal{Z}$ and simulates the participants. First, it gets X, Y from $\mathcal{Z}$ and deduces $X_0 = X$ and $Y_0 = E_{-1}Y^2$. The game then gives back X_{j_0} and Y_{j_0}. Then $\mathbb{A}^1$ interacts with $\mathcal{Z}$ by simulating the rounds of communication until j_0's turn, in which case $\mathbb{A}^1$ plays X_{j_0} and Y_{j_0}. Since $\mathbb{A}^1$ emulates $\mathcal{F}_{\mathcal{RO}}$, it can give a perfect simulation of the ZKP. Note that the soundness of the ZKP provided by $\mathcal{Z}$ during the rounds of communication guarantees that, except with negligible probability, the ciphertexts X_{j_0}, Y_{j_0} played by $\mathbb{A}^1$ follow the same distribution as in the real and the simulated processes. Since j_0 was the last honest participant, $\mathcal{Z}$ will afterwards send a succession of X_i, Y_i, π_i for $i > j_0$. To produce the ZKP π in game G_1, $\mathbb{A}^1$ extracts the secret $s_i, r_{1,i}, r_{2,i}$ such that $X_i = \mathsf{ReEnc}(X_{i-1}^{s_i}, r_{1,i})$ while $Y_i = \mathsf{ReEnc}(Y_{i-1}^{s_i}, r_{2,i})$ from the ZKP of $\mathcal{Z}$ and uses the respective witnesses. Then $\mathbb{A}^1$ is given $X'_\delta, Y'_\delta, A_\delta, B_\delta, S_\delta$ in game G_1. In a similar way as $\mathcal{S}$ uses a specific value for A_{j_0}, B_{j_0} during the reencryption phase, $\mathbb{A}^1$ plays A_δ and B_δ during this phase. During the threshold decryption phase, $\mathbb{A}^1$ uses the partial decryptions given by S_δ as the partial decryption of the honest participants. Finally, $\mathbb{A}^1$ outputs $\mathcal{Z}$'s output. Clearly, if $\delta = 1$, $\mathcal{Z}$ gets a perfect simulation of the interactions in the real process while, if $\delta = 0$, $\mathcal{Z}$ gets a perfect simulation of the interactions in the simulated process. Hence, $\mathbb{A}^1$'s advantage in winning G_1 is $1/2(p + 1 - p^{\mathcal{S}}) - 1/2 = 1/2(p - p^{\mathcal{S}})$, which is non-negligible.

Algorithm 54: G_1

Require: κ, t, a, C

1 $sk, pk, s_1, h_1 \cdots, s_a, h_a := \mathsf{DKG}(\kappa, t, a)$
2 $X_0, Y_0, P_1 := \mathbb{A}_1(pk, (h_i)_{i=1}^a, (s_i)_{i \in C})$
3 $x_0 := \mathsf{Dec}_{sk}(X_0); y_0 := \mathsf{Dec}_{sk}(Y_0)$
4 $\alpha \in_r \{-1, 1\}; r_1, r_2 \in_r \mathbb{Z}_q$
5 $X_{j_0} := \mathsf{ReEnc}_{pk}(X_0^\alpha, r_1); Y_{j_0} := \mathsf{ReEnc}_{pk}(Y_0^\alpha, r_2)$
6 $X_a, Y_a, \pi, P_2 := \mathbb{A}_2(P_1, X_{j_0}, Y_{j_0})$
7 **if** π *is invalid* **then**
8 Return $\perp$
9 $Z := \mathsf{Enc}_{pk}(x_0(y_0 + 1)/2); Z' := \mathsf{ReEnc}_{pk}(Z)$
10 $X'_1 := \mathsf{ReEnc}_{pk}(X_a); Y'_1 := \mathsf{ReEnc}_{pk}(Y_a)$
11 $\beta \in_r \{-1, 1\}$
12 $Y'_0 := \mathsf{Enc}_{pk}(\beta); X'_0 := (Z^2/X_0)^{1/\beta}$
13 $A_i := X'_i/X_a; B_i := Y'_i/Y_a$ for $i \in \{0, 1\}$
14 $S_0 = \mathsf{Sim}_{t,a}(Y'_0, \beta, (s_i)_{i \in C})$
15 $S_1 := (\mathsf{PartDec}(Y'_1, s_i))_{i \notin C}$
16 $\delta \in_r \{0, 1\}$
17 $g := \mathbb{A}_3(P_2, Z', X'_\delta, Y'_\delta, A_\delta, B_\delta, S_\delta)$
18 Return $g == \delta$

Algorithm 55: G_3

Require: κ

1 $sk, pk := \mathsf{KeyGen}(\kappa)$
2 $X_0, Y_0, P_1 := \mathbb{A}_1(pk)$
3 $x_0 := \mathsf{Dec}_{sk}(X_0); y_0 := \mathsf{Dec}_{sk}(Y_0)$
4 $\alpha \in_r \{-1, 1\}; r_1, r_2 \in_r \mathbb{Z}_q$
5 $X_{j_0} := \mathsf{ReEnc}_{pk}(X_0^\alpha, r_1); Y_{j_0} := \mathsf{ReEnc}_{pk}(Y_0^\alpha, r_2)$
6 $Y_a, \pi, P_2 := \mathbb{A}_2(P_1, X_{j_0}, Y_{j_0})$
7 **if** π *is invalid* **then**
8 Return $\perp$
9 $Z := \mathsf{Enc}_{pk}(x_0(y_0 + 1)/2)$
10
11 $\beta \in_r \{-1, 1\}$
12
13
14 $y_0 := \beta$
15 $y_1 := \mathsf{Dec}_{sk}(Y_a)$
16 $\delta \in_r \{0, 1\}$
17 $g := \mathbb{A}_3(P_2, Z, y_\delta)$
18 Return $g == \delta$

Now, we consider a first game hop G_2 where S_δ in line 17 is replaced by $\mathsf{Dec}_{sk}(Y'_\delta)$. Note that whatever the value of δ, Y'_δ is uniformly distributed among the ciphertexts of $\mathsf{Dec}_{sk}(Y'_\delta)$. Consequently, by the ZK-TCPA security, there exists a restricted adversary $\mathbb{A}^2$ which wins G_2 with non-negligible advantage. Simply, $\mathbb{A}^2$ uses $\mathsf{Sim}_{t,a}$ to simulate the shares of the honest participants, and returns $\mathbb{A}^1$'s answer. Afterwards, we use a second game hop G_3 defined in Algorithm 55, where KeyGen consists of choosing a random group generator g, a random secret key sk, and setting $pk = g, g^{sk}$. We construct a restricted $\mathbb{A}^3$ which wins G_3 with non-negligible advantage.

For this purpose, $\mathbb{A}^3$ first gets pk from game G_3, computes random shares $(s_i)_{i \in C}$ and their respective commitments h_i, compute enough random commitments h_i for some honest i to have exactly t commitments in total, and uses pk to generate the remaining commitments. (Recall that t, a and C are fixed). In the ElGamal setting, this is possible by using Lagrange's interpolation "in the exponents", which allows to generate a view for $\mathbb{A}^2_1$ which is perfectly indistinguishable from $\mathbb{A}^2_1$'s view in Game G_2. See the proof of Lemma I.2 for more details. Afterwards, $\mathbb{A}^3$ gets X_0, Y_0 from $\mathbb{A}^2$ and X_{j_0}, Y_{j_0} from G_3. It forwards it to $\mathbb{A}^2$ which answers with X_a, Y_a, π. $\mathbb{A}^3$ extract s, r_2 such that $Y_a = \mathsf{ReEnc}(Y_{j_0}^s, r_2)$ from the ZKP and uses the witnesses to provide Y_a, π in game G_3. Finally, $\mathbb{A}^3$ gets Z and y_δ from G_3. It computes $Y'_\delta = \mathsf{Enc}_{pk}(y_\delta)$, $X'_\delta = (Z^2/X)^{1/y_\delta}$, $A_\delta = X'_\delta/X_a$, $B_\delta = Y'_\delta/Y_a$ and $Z' = \mathsf{ReEnc}_{pk}(Z)$ and outputs $\mathbb{A}^2$'s answer.

Remark that in game G_2, we also have $A_\delta = X'_\delta/X_a$ and $B_\delta = Y'_\delta/Y_a$. We also have that $Y'_\delta = \mathsf{Enc}(y_\delta)$ and that Z' is a random reencryption of Z. Finally, when $\delta = 0$, we have $y_\delta = \beta$ and $X'_\delta = (Z^2/X_0)^{1/y_\delta}$. When $\delta = 1$, X'_a, Y'_a are reencryptions of X_a and Y_a, which are reencryptions of X_a^s and Y_a^s, which are reencryptions of $X_0^{\alpha s}$ and $Y_0^{\alpha s}$. Also, when $\delta = 1$, $y_\delta = \mathsf{Dec}(Y_a) = \alpha s y_0$. Hence, $X_0 {X'}_\delta^{y_\delta}$ is an encryption of $x_0 + x_0 y_0$, and hence a reencryption of Z^2. It shows that $Z', X'_\delta, Y'_\delta, A_\delta, B_\delta, y_\delta$ follows the same distribution as in game G_2, and therefore $\mathbb{A}^3$ wins with G_3 with the same advantage as $\mathbb{A}^2$ wins G_2.

Now we consider game G_4 in which the adversary is not given Z at line 17. We consider the restricted adversary $\mathbb{A}^4$ which completes the view of $\mathbb{A}^3_3$ with a random reencryption of $Z' = (X_0 X_{j_0}^{y_\delta})^{1/2}$ (*i.e.* it sets Z as a random reencryption of Z' and outputs $\mathbb{A}^3_3$'s answer). Similarly, we consider $\mathbb{A}_r$ which completes the view of $\mathbb{A}^3_3$ using a random ciphertext.

Let $p_4(\mathbb{A}|\delta = b)$ (resp. $p_3(\mathbb{A}|\delta = b)$) be the conditional probability that $\mathbb{A}$ wins G_4 (resp. G_3) when $\delta = b$, and $p_4(\mathbb{A})$ (resp. $p_3(\mathbb{A})$) the probability that $\mathbb{A}$ wins game G_3 (resp. G_4). Note that $p_4(\mathbb{A}^4|\delta = 1) = p_3(\mathbb{A}^3|\delta = 1)$ so that

$$p_4(\mathbb{A}^4) - p_3(\mathbb{A}^3) = \frac{1}{2}(p_4(\mathbb{A}^4|\delta = 0) - p_3(\mathbb{A}^3|\delta = 0)).$$

We consider an adversary $\mathbb{B}$ for IND-CPA. The adversary gets pk and uses it to get X_0, Y_0 from $\mathbb{A}^3$. It chooses $\alpha \in_r \{-1, 1\}$ and X_{j_0}, Y_{j_0} as in game G_3, gives them to $\mathbb{A}^3$ which answers with Y_a, π. Afterwards, $\mathbb{B}$ computes $y_\delta \in_r \{-1, 1\}$ and $Z' = (X_0 X_{j_0}^{y_\delta})^{1/2}$. It finally queries a ciphertext C from the IND-CPA game with plaintexts 0 or 1, chooses a random exponent z and sends $Z = Z'C^z$ to $\mathbb{A}^3$. Finally, it outputs $\mathbb{A}^3$'s guess. Note that when C is an encryption of 0, $\mathbb{B}$ played $\mathbb{A}^4$'s simulation to $\mathbb{A}^3$ and wins the IND-CPA game if it outputs 0 so that it wins with probability $p_4(\mathbb{A}^4|\delta = 0)$. When C is an encryption of 1, $\mathbb{B}$ played $\mathbb{A}_r$'s simulation to $\mathbb{A}^3$ and wins if it outputs 1, which happens with probability $1 - p_4(\mathbb{A}_r|\delta = 0)$. Hence $\mathbb{B}$'s advantage in winning the IND-CPA game is $1/2(p_4(\mathbb{A}^4|e_{0,\pm 1}) - p_4(\mathbb{A}_r|e_{0,\pm 1}))$. Recall that under DDH assumption, the ElGamal encryption scheme is IND-CPA secure. It follows that

$$p_4(\mathbb{A}^4|\delta = 0) \approx p_4(\mathbb{A}_r|\delta = 0).$$

Now, we consider game G_{xy} defined by Algorithm 56. We consider adversary $\mathbb{B}$ which forwards pk to $\mathbb{A}^3$ to get X_0, Y_0, then computes X_{j_0}, Y_{j_0} as in game G_3 to get Y_a. Finally, $\mathbb{B}$ gets Z_δ from game G_{xy}, sets $y_\delta \in_r \{-1, 1\}$ and outputs $\mathbb{A}^3$'s outputs. When $\gamma = 1$, $\mathbb{B}$ plays a perfect simulation of G_3 when $\delta = 0$ to $\mathbb{A}^3$ and wins with probability $1 - p_3(\mathbb{A}^3|\delta = 0)$. When $\gamma = 0$, $\mathbb{B}$ plays $\mathbb{A}_r$'s simulation to $\mathbb{A}^3$ and wins with probability $p_4(\mathbb{A}_r|\delta = 0)$. Hence $\mathbb{B}$'s advantage in winning game G is $1/2(p_4(\mathbb{A}_r|\delta = 0) - p_3(\mathbb{A}^3|\delta = 0))$. By Lemma I.3, under IND-CPA assumption, $\mathbb{B}$ wins G_{xy} with negligible advantage. Therefore, we have $p_3(\mathbb{A}^3|\delta = 0) \approx p_4(\mathbb{A}_r|\delta = 0) \approx p_4(\mathbb{A}^4|\delta = 0)$, hence $p_4(\mathbb{A}^4) \approx p_3(\mathbb{A}^3)$.

Algorithm 56: G_{xy}

Require: $\mathbb{A}, \kappa$
$sk, pk := \mathsf{KeyGen}(\kappa)$
$X_0, Y_0, P_1 := \mathbb{A}_1(pk)$
$x_0 := \mathsf{Dec}_{sk}(X_0)$; $y_0 := \mathsf{Dec}_{sk}(Y_0)$
if $(x_0 \notin \{0, 1\})$ *or* $y_0 \notin \{-1, 1\}$ **then**
  Return $\perp$
$\alpha \in_r \mathbb{Z}_q$; $\gamma \in_r \{0, 1\}$
$Z_1 := \mathsf{Enc}_{pk}(x_0(y_0 + 1)/2)$; $Z_0 := \mathsf{Enc}_{pk}(\alpha)$
$g := \mathbb{A}_2(P_1, Z_\gamma)$
Return $g == \gamma$

Now, we consider game G_5 defined by Algorithm 57. Assume that there exists an adversary $\mathbb{A}^5$ which wins G_5 with non-negligible probability. Then, we construct $\mathbb{B}$ for IND-CPA. It gets pk and C, an encryption of either 0 or 1 from the IND-CPA game. It chooses $\alpha \in_r \{-1, 1\}$ and computes $Y_{j_0} = \mathsf{Enc}(\alpha)C^a$ for some random exponent a. Using this Y_{j_0}, it queries Y_a from $\mathbb{A}^5$ and from the ZKP extracts $s \in \{-1, 1\}$ such that Y_a is a reencryption of $Y_{j_0}^s$. It then chooses $\beta \in_r \{-1, 1\}$ and $\delta \in_r \{0, 1\}$, sets $y_0 = \beta$ and $y_1 = s\alpha$ and output 0 if $\mathbb{A}_3^5$ outputs δ. Note that when C is an encryption of 0, $\mathbb{A}_3^5$ has a perfect simulation of game G_5 and $\mathbb{B}$ wins with a non-negligible advantage. When C is an encryption of 1, Y_{j_0}, y_δ is uniformly random, so that $\mathbb{B}$ wins with probability $1/2$. Overall, $\mathbb{B}$ wins with a non-negligible advantage.

Algorithm 57: G_5

Require: κ
$sk, pk := \mathsf{KeyGen}(\kappa)$
$\alpha \in_r \{-1, 1\}$; $Y_{j_0} := \mathsf{Enc}_{pk}(\alpha)$
$Y_a, \pi, P_1 := \mathbb{A}_1(pk, Y_{j_0})$
if π *is invalid* **then**
  Return $\perp$
$\beta \in_r \{-1, 1\}$, $\delta \in_r \{0, 1\}$ $y_1 := \mathsf{Dec}_{sk}(Y_a)$; $y_0 := \beta$
$g := \mathbb{A}_2(P_1, y_\delta)$
Return $g == \delta$

From the above paragraph and under DDH assumption (which is equivalent to IND-CPA security of the ElGamal encryption scheme), it follows that for all adversary $\mathbb{A}$, the probability $p_5(\mathbb{A})$ that $\mathbb{A}$ wins G_5 is approximately $1/2$. We will exhibit a contradiction with the fact that $\mathbb{A}^4$ wins G_4 with non-negligible advantage, and therefore conclude the proof. For this purpose, let X_0, Y_0, P_1 be the output of $\mathbb{A}^4$ when given a random pk with secret key sk. We denote $p_{0,\pm 1} = \Pr(\mathtt{Dec}_{sk}(X_0) = 0)$, $p_{1,1} = \Pr(\mathtt{Dec}_{sk}(X_0) = 1, \mathtt{Dec}_{sk}(Y_0) = 1)$ and $p_{1,-1} = \Pr(\mathtt{Dec}_{sk}(X_0) = 1, \mathtt{Dec}_{sk}(Y_0) = -1)$. For adversary $\mathbb{A}$ and game $k \in \{4, 5\}$, for $a \in \{(0, \pm 1), (1, 1), (1, -1)\}$, we denote $p_k(\mathbb{A}|e_a)$ the conditional probability that $\mathbb{A}$ wins game G_k when X_0 is an encryption of 0 (for $a = 0, \pm 1$), when X_0 is an encryption of 1 while Y_0 is an encryption of 1 (for $a = 1, 1$) and when X_0 is an encryption of 1 while Y_0 is an encryption of -1 (for $a = 1, -1$). Note that since $\mathbb{A}^4$ is restricted, for all $\mathbb{A}$, we have

$$p_5(\mathbb{A}) = p_{0,\pm 1} p_5(\mathbb{A}|e_{0,\pm 1}) + p_{1,1} p_5(\mathbb{A}|e_{1,1}) + p_{1,-1} p_5(\mathbb{A}|e_{1,-1}).$$

We consider adversary $\mathbb{A}_r$ for game G_5 which sets X_{j_0} as a random cipherext, gets Y_{j_0} from the game and forwards X_{j_0}, Y_{j_0} to $\mathbb{A}^4$. Observe that, in this case, X_{j_0}, Y_{j_0} is independent from X_0, Y_0 so that $p_5(\mathbb{A}_r|e_{0,\pm 1}) = p_5(\mathbb{A}_r|e_{1,1}) = p_5(\mathbb{A}_r|e_{1,-1}) = p_5(\mathbb{A}_r)$. In addition, we have $1/2 \approx p_5(\mathbb{A}_r)$. Hence,

$$p_5(\mathbb{A}_r|e_{0,\pm 1}) = p_5(\mathbb{A}_r|e_{1,1}) = p_5(\mathbb{A}_r|e_{1,-1}) = p_5(\mathbb{A}_r) \approx 1/2.$$

Now, we consider adversary $\mathbb{A}_0$ which sets X_{j_0} as a random reencryption of X_0^{α}, for a random α. When X_0 is an encryption of 0, $\mathbb{A}_0$ plays a perfect simulation of game G_4 to $\mathbb{A}^4$ and hence wins with the same probability. When X_0 is an encryption of 1, $\mathbb{A}_0$ plays $\mathbb{A}_r$'s simulation and hence wins with probability approximately $1/2$. Consequently we have that

$$\begin{aligned}
\frac{1}{2} \approx p_5(\mathbb{A}_0) &= p_{0,\pm 1} p_5(\mathbb{A}_0|e_{0,\pm 1}) + p_{1,1} p_5(\mathbb{A}_0|e_{1,1}) + p_{1,-1} p_5(\mathbb{A}_0|e_{1,-1}) \\
&= p_{0,\pm 1} p_4(\mathbb{A}^4|e_{0,\pm 1}) + p_{1,1} p_5(\mathbb{A}_r|e_{1,1}) + p_{1,-1} p_5(\mathbb{A}_r|e_{1,-1}) \\
&\approx p_{0,\pm 1} p_4(\mathbb{A}^4|e_{0,\pm 1}) + \frac{1}{2} p_{1,1} + \frac{1}{2} p_{1,-1}.
\end{aligned}$$

Similarly, we consider adversary $\mathbb{A}_1$ which sets $X_{j_0} = Y_{j_0}(X_0/Y_0)^{\alpha}$ for some random α. If $\mathtt{Dec}_{sk}(X_0) = \mathtt{Dec}_{sk}(Y_0) = 1$, $\mathbb{A}_1$ plays a perfect simulation of game G_4 to $\mathbb{A}^4$ and therefore wins with the same probability. Otherwise, $\mathbb{A}_1$ plays $\mathbb{A}_r$'s simulation and wins with probability approximately $1/2$. Hence, we have

$$\begin{aligned}
\frac{1}{2} \approx p_5(\mathbb{A}_1) &= p_{0,\pm 1} p_5(\mathbb{A}_1|e_{0,\pm 1}) + p_{1,1} p_5(\mathbb{A}_1|e_{1,1}) + p_{1,-1} p_5(\mathbb{A}_1|e_{1,-1}) \\
&= p_{0,\pm 1} p_5(\mathbb{A}_r|e_{0,\pm 1}) + p_{1,1} p_4(\mathbb{A}^4|e_{1,1}) + p_{1,-1} p_5(\mathbb{A}_r|e_{1,-1}) \\
&\approx \frac{1}{2} p_{0,\pm 1} + p_{1,1} p_4(\mathbb{A}^4|e_{1,1}) + \frac{1}{2} p_{1,-1}.
\end{aligned}$$

Finally, we consider adversary $\mathbb{A}_{-1}$ which sets $X_{j_0} = (X_0 Y_0)^{\alpha}/Y_{j_0}$ for some random α. If $\mathtt{Dec}_{sk}(X_0) = 1$ while $\mathtt{Dec}_{sk}(Y_0) = -1$, $\mathbb{A}_{-1}$ plays a perfect simulation of game G_4. Otherwise, it plays $\mathbb{A}_r$'s simulation. Therefore,

$$\begin{aligned}
\frac{1}{2} \approx p_5(\mathbb{A}_{-1}) &= p_{0,\pm 1} p_5(\mathbb{A}_{-1}|e_{0,\pm 1}) + p_{1,1} p_5(\mathbb{A}_{-1}|e_{1,1}) + p_{1,-1} p_5(\mathbb{A}_{-1}|e_{1,-1}) \\
&= p_{0,\pm 1} p_5(\mathbb{A}_r|e_{0,\pm 1}) + p_{1,1} p_5(\mathbb{A}_r|e_{1,1}) + p_{1,-1} p_4(\mathbb{A}^4|e_{1,-1}) \\
&\approx \frac{1}{2} p_{0,\pm 1} + \frac{1}{2} p_{1,1} + p_{1,-1} p_4(\mathbb{A}^4|e_{1,-1}).
\end{aligned}$$

By computing the average of the three formulas, we have

$$\begin{aligned}
\frac{1}{2} &\approx \frac{1}{3} \left(p_{0,\pm 1} p_4(\mathbb{A}^4|e_{0,\pm 1}) + \frac{1}{2} p_{1,1} + \frac{1}{2} p_{1,-1} + \frac{1}{2} p_{0,\pm 1} + p_{1,1} p_4(\mathbb{A}^4|e_{1,1}) + \frac{1}{2} p_{1,-1} + \frac{1}{2} p_{0,\pm 1} + \frac{1}{2} p_{1,1} + p_{1,-1} p_4(\mathbb{A}^4|e_{1,-1}) \right) \\
&= \frac{1}{3} (p_{0,\pm 1} + p_{1,1} + p_{1,-1} + p_{0,\pm 1} p_4(\mathbb{A}^4|e_{0,\pm 1}) + p_{1,1} p_4(\mathbb{A}^4|e_{1,1}) + p_{1,-1} p_4(\mathbb{A}^4|e_{1,-1})) \\
&= \frac{1}{3} (1 + p_4(\mathbb{A}^4)).
\end{aligned}$$

Hence $p_4(\mathbb{A}^4) \approx 1/2$, which contradicts the fact that $\mathbb{A}^4$ wins G_4 with non-negligible probability. □

I.2 Extending the SUC-security of `CGate` to all the MPC toolbox

In Section I.1, we proved Theorem I.4 that states the SUC-security of the `CGate` protocol. As explained in Appendix H, this framework is extremely convenient due to its composability, which is expressed in Theorem H.3. As all our MPC building blocks are simply compositions of `CGate`, this allows us to generalize the SUC security of the `CGate` protocol to the other MPC protocols of our toolbox.

For this purpose, for all protocol P_f presented in Part I., Appendix C, we define the corresponding functionality f in the plaintext. In addition, we consider the trusted party T_f which is given the inputs I of P_f, decrypts them and returns the evaluation of f on the decrypted inputs (see Figure 20). Note that in case of a failure, the trusted party can output $\perp$ to all participants. Finally, we denote $\tilde{P}_f$ the protocol which consists of running P_f, then decrypting its output using a threshold decryption protocol. We show in Theorem I.5 that $\tilde{P}_f$ securely computes T_f.

- If there is a new message from the adversary in the input communication tape, it sends $(T_f, i, \perp)$ to all participant i.
- If there is a new message of the form $(i, T_f, s_{id}||I)$ in the input communication tape, it looks for a session with the same session identifier s_{id}. If such a session exists, it queries the value s of `share` for i and adds I, s to the FIFO queue q_i of the session. If there is no such session, it creates a new session with identifier s_{id}, creates a FIFO queue q_j for all participant j and adds I, s to q_i. Note that if $s = \perp$, it ignores the query.
- If there is a session in which all queues are non-empty, it removes their first elements $(I_i, s_i)_i$ and do the following.
  - If there is i, i' such that $I_i \neq I_{i'}$, make a send query $(T_f, i, \perp)$ for all participant i.
  - Otherwise, decrypt I using the shares, check that the plaintexts is valid with respect to f (if it is not, make a send query $(T_f, i, \perp)$ for all i) and compute the result r by applying f on the plaintexts (by nature, r itself is a plaintext). Make a send query (T_f, i, r) for all i.

Figure 20: Specification of T_f in the SUC-framework

Theorem I.5. *Under DDH assumtion, for all protocol P_f presented in Part I., Appendix C (except the Paillier-specific protocols), $\tilde{P}_f$ securely computes T_f in the $\mathcal{F}_M - \mathcal{F}_{\mathcal{RO}}$-hybrid model, with respect to environments which only set encryptions of* 0 *or* 1 *as inputs, and give the same input to all participants.*

To prove Theorem I.5, we first formalize what we mean by *composition* of `CGate`. Intuitively, a protocol is a composition of `CGate` (Definition I.6) if all the communications between the participants are part of a `CGate` protocol. Clearly, in the ElGamal setting, all of our building blocks from Part I., Appendix C are compositions of `CGate`. In this case, the SUC-security follows by inspection (see Lemma I.7).

Definition I.6. Let $P_f = P_1, \cdots, P_a$ be a protocol for a participants. We denote $P_f^{\mathcal{F}_{\mathtt{CGate}}}$ the protocol obtained by replacing each call to the sub-protocol `CGate` in P_f by a call to an oracle $\mathcal{F}_{\mathtt{CGate}}$ which takes as input the public encryption key and two encryptions X and Y of x and y, and return an encryption Z of xy if $x, y \in \{0, 1\}$, and $\perp$ otherwise.

We say that P_f is a composition of `CGate` if there is no send query in $P_f^{\mathcal{F}_{\mathtt{CGate}}}$.

Lemma I.7. *Let $f : \mathcal{I} \longrightarrow O$ be some function, P_f be a composition of* `CGate`*, $P_f^{\mathcal{F}_{\mathtt{CGate}}}$ the protocol obtained by the transformation in Definition I.6 and $P_f^{T_{\mathtt{CGate}}}$ the protocol obtained by replacing the calls of the* `CGate` *sub-protocols in P_f by an ideal call to $T_{\mathtt{CGate}}$. We denote $\tilde{P}_f$, $\tilde{P}_f^{\mathcal{F}_{\mathtt{CGate}}}$ and $\tilde{P}_f^{T_{\mathtt{CGate}}}$ the protocols obtained by running P_f, $P_f^{\mathcal{F}_{\mathtt{CGate}}}$ and $P_f^{T_{\mathtt{CGate}}}$, then decrypting the result.*

Suppose that for all input I such that decrypting I results in an element $x \in \mathcal{I}$, $\tilde{P}_f^{\mathcal{F}_{\mathtt{CGate}}}(I) = f(x)$.

Then, under DDH assumption, $\tilde{P}_f^{T_{\mathtt{CGate}}}$ securely computes T_f in the $T_{\mathtt{CGate}}$-hybrid model, with respect to environments which only set encryptions of 0 *or* 1 *as inputs, and give the same input to all participants.*

Proof. We construct a simulator `Sim` which simulates all the participants and their communications. Since P_f is a composition of `CGate`, `Sim` can perform a perfect simulation of P_f in the $T_{\mathtt{CGate}}$-hybrid model. Indeed, there is no communication to simulate and `Sim` can answer $\mathcal{F}_{\mathtt{CGate}}$ queries by making queries to the trusted party $T_{\mathtt{CGate}}$. Hence, the only part which `Sim` has to actually simulate is the threshold decryption.

For this purpose, `Sim` simply forwards the queries of all participants to T_f which answers with some r. If $r = \perp$, `Sim` aborts the protocol by asking T_f to send $\perp$ to all participants. Otherwise, `Sim` simulates the threshold decryption to have the output of $\tilde{P}_f^{T_{\mathtt{CGate}}}$ be r. Note that since $r \neq \perp$, the inputs I decrypt to some valid $x \in \mathcal{I}$, and we have $\tilde{P}_f^{\mathcal{F}_{\mathtt{CGate}}}(I) = f(x) = r$. Hence, the environment cannot make the difference by observing the output. In addition, under DDH assumption the ElGamal threshold cryptosystem is ZK-TCPA (see Lemma I.2) so that the simulation is indistinguishable from the real threshold decryption. □

Proof of Theorem I.5. By Theorem I.4, under DDH assumption, `CGate` securely computes $T_{\mathtt{CGate}}$ in the $\mathcal{F}_M - \mathcal{F}_{\mathcal{RO}}$-hybrid model. Hence, by Theorem H.3, it is sufficient to show that $\tilde{P}_f^{T_{\mathtt{CGate}}}$, securely computes T_f in the $T_{\mathtt{CGate}}$-hybrid model. By Lemma I.7, it is sufficient to check that the protocol $\tilde{P}_f^{\mathcal{F}_{\mathtt{CGate}}}$ is correct, which can be done by inspection. □

I.3 The SUC-security of our tally-hiding protocols

In Section I.2, we proved that our toolbox is SUC-secure. However, we considered a slightly degraded toolbox, where each building block would end with a decryption phase. This looks regrettable since the security of a more complex protocol would have to be proven from scratch, and could not be obtained from composition. Nonetheless, recall that we proved Lemma I.7 which gives a simple criteria to establish SUC-security: a protocol which is correct (the output is correct when all participants are honest) and which is a composition of `CGate` followed by a threshold decryption is automatically SUC-secure. This gives Theorem I.8, where $T_{\mathtt{tally}}$ is an instance of T_f (see Figure 20).

Theorem I.8. *Let* tally *be one of the considered tally function in this article (D'Hondt, Majority Judgment, Condorcet- Schulze or STV), let* $P_{\texttt{tally}}$ *be the corresponding tally-hiding protocol and* $T_{\texttt{tally}}$ *the corresponding trusted party. Then, under DDH assumption,* $P_{\texttt{tally}}$ *securely computes* $T_{\texttt{tally}}$ *in the* $\mathcal{F}_M - \mathcal{F}_{RO}$*- hybrid model, with respect to environments which only set encryptions of* 0 *or* 1 *as inputs, and give the same input to all participants.*

Proof. With the notation of Lemma I.7, we remark that $P_{\texttt{tally}}$ can be expressed as $\tilde{\pi}_{\texttt{tally}}$, where π verifies the hypotheses of Lemma I.7 with the function tally. We conclude using Theorem I.4 and Theorem H.3. □

Note that Theorem 3.1, Theorem 4.1, Theorem 5.1, Theorem 6.1 follow directly from Theorem I.8.

J THE SECURITY OF OUR VOTING SCHEMES

Informally, an electronic voting protocol aims at evaluating the tally function on the voting options chosen by the voters, without compromising privacy and in a verifiable manner. In this appendix, we consider TH-voting, a familly of voting schemes, that only reveals the result of an election. We define the notions of privacy and verifiability, and prove that our voting schemes $V_{\texttt{tally}}$ satisfy these properties for all the considered tally functions (D'Hondt, Majority Judgment, Condorcet-Schulze and Single Transferable Vote).

J.1 Completeness and correctness of a voting scheme

As defined in Section 7, a *voting scheme* consists of four algorithms and one MPC protocol $(\texttt{Setup}, \texttt{vote}, \texttt{isValid}, P_{\texttt{tally}}, \texttt{Verify})$.

We first define some basic properties that a voting scheme must satisfy in order to be meaningful. First, isValid and $P_{\texttt{tally}}$ must be complete, which means that an honestly generated ballot or tally must be treated as valid (Definition J.1). Second, we need the tally protocol to be consistent with the tally function, which means that it produces the correct result if the participants are honest (see Definition J.2, which is inspired from [11]).

Definition J.1 (Completeness). A voting scheme $(\texttt{Setup}, \texttt{vote}, \texttt{isValid}, P_{\texttt{tally}}, \texttt{Verify})$ is complete if, for all valid voting option v, for all ballot board BB, for all public encryption key pk, for all (a, t) such that $t < a$,

- $\texttt{isValid}(BB, \texttt{vote}(pk, v)) = 1$ with overwhelming probability.
- If r, Π is the output of $P_{\texttt{tally}}(a, t)$ run with honest participants, then $\texttt{Verify}(r, \Pi, BB) = 1$.

Definition J.2 (Consistency). A voting scheme $\mathcal{V} = (\texttt{Setup}, \texttt{vote}, \texttt{isValid}, P_{\texttt{tally}}, \texttt{Verify})$ is consistent w.r.t. a tally function tally if the two following properties are met.

- There exists an Extract algorithm which takes as input the secret decryption key sk and a ballot B, and outputs a valid voting option. Extract is such that, for all valid voting option v and for all key pair (sk, pk), we have $\texttt{Extract}_{sk}(\texttt{vote}(pk, v)) = v$.
- For all PPT adversary $\mathbb{A}$, for all number of participant a and all threshold $t < a$, then the probability $\Pr(\texttt{Cons}_{\mathbb{A},\mathcal{V},\texttt{tally}}(\kappa, a, t) = 1)$ is negligible in κ, where Cons is defined in Algorithm 58, where $P_{\texttt{tally}}(B_1, \cdots, B_n)$ denotes the output of $P_{\texttt{tally}}(a, t)$ run with honest participants.

Algorithm 58: $\texttt{Cons}_{\mathbb{A},\mathcal{V},\texttt{tally}}(\kappa)$

$(sk, pk, \_) := \texttt{Setup}(\kappa, a, t)$;
$B_1, \cdots, B_n := \mathbb{A}(\kappa)$;
if *there exists i such that* $\texttt{isValid}((B_1, \cdots, B_{i-1}), B_i) = 0$ **then**
    Return 0
$(r, \Pi) := P_{\texttt{tally}}(B_1, \cdots, B_n)$;
Return $r \neq \texttt{tally}(\texttt{Extract}_{sk}(B_1), \cdots, \texttt{Extract}_{sk}(B_n))$

In what follows, we will only consider complete and consistent voting schemes.

J.2 Definition of TH-voting

We define a tally-hiding voting protocol $V_{\texttt{tally}} = (\texttt{Setup}_{\texttt{tally}}, \texttt{vote}_{\texttt{tally}}, \texttt{isValid}_{\texttt{tally}}, P_{\texttt{tally}}, \texttt{Verify}_{\texttt{tally}})$ for each tally function tally considered in our paper (D'Hondt, Majority Judgment, Condorcet-Schulze, and STV).

- First, the setup does not depend on the tally function and is described in Algorithm 59.
- As for the voting procedure, we consider $\texttt{vote}_{\texttt{tally}}(pk, v)$, that consists of a bitwise encryption of the relevant encoding of v using ElGamal with public encryption key pk, and a ZKP that v is a valid voting option and that the randomness used to produce the ciphertext is known. As it it shown in [12], this allows to define a NM-CPA encryption scheme. For more precision about how a voting option is encoded or how the ZKP is produced, refer to the corresponding section in Part I. of this appendix.
- The algorithm $\texttt{isValid}_{\texttt{tally}}$ simply consists of verifying the ZKP and checking that the ballot is not already on the board.

Algorithm 59: `Setup`

Require: (κ, a, t)

Choose a group of prime order q for the security parameter κ
Choose a random generator g
Choose a random polynomial Q of degree t
for $i = 0$ *to* a **do**
  $s_i \longleftarrow Q(i)$
  $h_i \longleftarrow g^{s_i}$
$sk \longleftarrow s_0; pk \longleftarrow (g, h_0)$
Return $(sk, pk, s_1, h_1, \cdots, s_a, h_a)$

- In Sections 3, 4, 5 and 6, we explained how to design a MPC protocol $P_{\mathtt{tally}}$ for each of these tally functions, and give a complete algorithmic description in Part I. of the appendix.
- As explained in Section C.1, the `CGate` protocol produces a transcript which acts as a ZKP that the protocol was performed correctly. By concatenating the transcripts of all `CGate` and the transcript of the threshold decryption, the participants can produce a ZKP Π that $P_{\mathtt{tally}}$ has been performed correctly. This also defines a $\mathtt{Verify}_{\mathtt{tally}}$ algorithm which simply consists of verifying all the ZKP.

The completeness of $V_{\mathtt{tally}}$ comes from the completeness of the ZKP. As for the consistency, the `Extract` algorithm can be directly derived from the decryption algorithm, and the consistency comes from the correctness of the protocol and the soundness of the ZKP (that guarantees that each ballot of the board contains a valid voting option).

J.3 Privacy of TH-voting

We first show that our schemes provides privacy. In [29], a quantitative definition of privacy is proposed, where a voting system is said δ-private for some δ. This definition can be turned into a qualitative one when δ is shown to be minimal, in a sense that an ideal protocol does not achieve δ'-privacy with a non-negligible $|\delta - \delta'|$. Hence, a natural definition of privacy is to compare the probability of success of the adversary in a real and in an ideal protocol, and to show that the difference is negligible. Just as in [29], we consider a definition where the adversary would only try to guess the vote of a single voter. To establish our result of privacy, we exploit the fact that the number of valid voting options is small compared to the size of all possible plaintexts.

Valid voting options. We consider a small fixed set V of valid voting options. Remark that in this case, for all $v \in V$, for all ElGamal key pair (sk, pk), for all $sk' \neq sk$, the probability that $\mathtt{Dec}_{sk'}(\mathtt{Enc}_{pk}(v)) \in V$ is negligible. We refer to this result as the *small voting set* setting.

Definition J.3 (Privacy). We say that a voting scheme $(\mathtt{Setup}, \mathtt{vote}, \mathtt{isValid}, P_{\mathtt{tally}}, \mathtt{Verify})$ for the result function $\mathtt{tally} : V^* \longrightarrow O$ is private if, for all number of participants $a \geq 2$, for all threshold $t < a$, for all parameters n, n_c with $n_c \leq n$, for all $C \subset [1, a]$ of size at most t, for all adversary $\mathbb{A}$ which can impersonate the participants of $(P_i)_{i \in C}$, where $P_{\mathtt{tally}}(a, t) = P_1, \cdots, P_a$, there exists an adversary $\mathbb{B}$ and a negligible function μ such that, for all voting options $v_2, \cdots, v_n \in \mathcal{I}$,

$$|Pr(\mathtt{Real}^{\mathtt{Priv}}_{\mathbb{A}, P_{\mathtt{tally}}}(\kappa, n, n_c, a, t, C, V, v_2, \cdots, v_n) = 1) - \Pr(\mathtt{Ideal}^{\mathtt{Priv}}_{\mathbb{B}}(\kappa, n, n_c, a, t, C, V, v_2, \cdots, v_n) = 1)| \leq \mu(\kappa).$$

Algorithm 60: $\mathrm{Real}^{\mathtt{Priv}}_{\mathbb{A}, P_{\mathtt{tally}}}(\kappa, n, n_c, a, t, C, V, v_2, \cdots, v_n)$

$sk, pk, s_1, h_1, \cdots, s_a, h_a := \mathtt{Setup}(\kappa, a, t)$
$par := pk, h_1, \cdots, h_a$
$v_0, v_1 := \mathbb{A}(\kappa, par, (s_i)_{i \in C})$
$b \in_r \{0, 1\}$
$BB := \{\mathtt{vote}_{pk}(v_b)\}$
for $i = 2$ *to* $n - n_c$ **do**
  $BB := BB \bigcup \{\mathtt{vote}(pk, v_i)\}$
$(X_i)_{i > n - n_c} := \mathbb{A}(BB)$
for $i > n - n_c$ **do**
  if $\mathtt{isValid}(BB, X_i)$ **then**
    $BB := BB \bigcup \{X_i\}$
$r, \Pi := \mathbb{A} ||_{i \in [1,a] \setminus C} P_i(s_i, par, BB)$
$b' := \mathbb{A}()$
Return $(b == b') \wedge (v_0, v_1 \in V)$

Algorithm 61: $\mathrm{Ideal}^{\mathtt{Priv}}_{\mathbb{B}}(\kappa, n, n_c, a, t, C, V, v_2, \cdots, v_n)$

$sk, pk, s_1, h_1, \cdots, s_a, h_a := \mathtt{Setup}(\kappa, a, t)$
$par := pk, h_1, \cdots, h_a$
$v_0, v_1 := \mathbb{A}(\kappa, par, (s_i)_{i \in C})$
$b \in_r \{0, 1\}$
$BB := \{\mathtt{vote}(pk, v_b)\}$
for $i = 2$ *to* $n - n_c$ **do**
  $BB := BB \bigcup \{\mathtt{vote}_{pk}(v_i)\}$
$(X_i)_{i > n - n_c} := \mathbb{A}()$
for $i > n - n_c$ **do**
  if $\mathtt{isValid}(BB, X_i)$ **then**
    $BB := BB \bigcup \{X_i\}$
$r := \mathtt{tally}((\mathtt{Extract}_{sk}(B))_{B \in BB})$
$b' := \mathbb{A}(r)$
Return $(b == b') \wedge (v_0, v_1 \in V)$

To prove that privacy is achieved, we use a notion of non-malleability given in Definition J.4. Although this definition may seem different to what the reader is accustomed with, [9] proves that it is equivalent to the standard notion of non-malleability. The proof of privacy is given in Theorem 7.2.

Definition J.4 (IND-PA0 [9]). Let $\Pi = (K, \mathsf{Enc}, \mathsf{Dec})$ be three algorithms of key generation, encryption and decryption of an encryption scheme. We say that Π is non-malleable if, for all PPT $\mathbb{A}$, there exists a negligible function μ such that

$$|\Pr(\mathsf{Exp}_{\mathbb{A},\Pi}^{IND-PA0}(\kappa) = 1) - \frac{1}{2}| \leq \mu(\kappa).$$

Algorithm 62: $\mathsf{Exp}_{\mathbb{A},\Pi}^{IND-PA0}$

Require: κ
$(pk, sk) := K(\kappa)$
$(x_0, x_1, s_1) := \mathbb{A}_1(pk)$
$b \in_r \{0, 1\}$
$y := \mathsf{Enc}_{pk}(x_b)$
$\boldsymbol{c}, s_2 := \mathbb{A}_2(s_1, y)$
$\boldsymbol{p} := \mathsf{Dec}_{sk}(\boldsymbol{c})$
$g := \mathbb{A}_3(s_2, \boldsymbol{p})$
Return $(y \notin \boldsymbol{c}) \wedge (g == b)$

Theorem 7.2. *Let* $\mathtt{tally}$ *be one of the previously defined tally functions (D'Hondt, Majority Judgment, Condorcet-Schulze, and STV). Assuming DDH,* $V_{\mathtt{tally}}$ *is private w.r.t.* $\mathtt{tally}$.

Proof. Let $a \geq 2$, $t < a$ and $0 \leq n_c \leq n$ some fixed integers. Let $C \subset [1, a]$ of size at most t. We use a first game hop H_1 where the line 12 of the Real game is replaced by the execution of an ideal process where all honest party hands over their shares to $T_{\mathtt{tally}}$ (which is an instance of Figure 20 with the $\mathtt{tally}$ function). Following the notation of the SUC framework, we denote $\mathsf{IDEAL}_{\mathtt{tally},\mathbb{A},\mathcal{Z}}(\kappa, z)$ the output of the environment $\mathcal{Z}$ in such a process, when it interacts with adversary $\mathbb{A}$.

Consider an adversary $\mathbb{A}^0$ for the Real game. For convenience, we split $\mathbb{A}^0$ into $\mathbb{A}_1^0, \mathbb{A}_2^0, \mathbb{A}_3^0, \mathbb{A}_4^0$ according to the four invocations of $\mathbb{A}^0$ during the Real game. By Theorem I.8, $P_{\mathtt{tally}}$ securely computes $T_{\mathtt{tally}}$ in the $\mathcal{F}_M - \mathcal{F}_{\mathcal{RO}}$-hybrid model, with respect to environments which only set encryptions of 0 or 1 as inputs, and give the same input to all participants (we refer to such environments as *restricted*). By definition, there exists a simulator $\mathcal{S}$ such that for all restricted environment $\mathcal{Z}$, there exists a negligible function μ such that for all κ and $z \in \{0, 1\}^*$ of polynomial size,

$$|\Pr(\mathsf{IDEAL}_{\mathtt{tally},\mathcal{S},\mathcal{Z}}(\kappa, z) = 1) - \Pr(\mathsf{REAL}_{P,\mathbb{A}_3,\mathcal{Z}}(\kappa, z) = 1)| \leq \mu(\kappa).$$

In particular, we consider the environment which chooses $v_0, \cdots, v_n$, performs the DKG, picks b at random, encrypts the honest voters' ballots, accept valid ballots from the adversary and checks that $b' == b$. This shows that the adversary $\mathbb{A}^1 = \mathbb{A}_1^0, A_2^0, \mathcal{S}, \mathbb{A}_4^0$ has the same probability to win H_1 as $\mathbb{A}^0$ has to win the Real game, with up to a negligible difference.

Now, we consider a second hop H_2 where lines 12 and 13 of H_1 are replaced by lines 12 and 13 of the Ideal game. We construct an adversary $\mathbb{A}^2$ for game H_2 as follows. First, $\mathbb{A}^2$ calls $\mathbb{A}_1^0$ and $\mathbb{A}_2^0$ to get v_0, v_1 and X. Then, $\mathbb{A}^2$ interacts with $\mathcal{S}$ by emulating the honest and the trusted parties in the ideal process. To emulate the honest parties, $\mathbb{A}^2$ simply sends empty queries to the trusted party (recall that $\mathcal{S}$ cannot see the content of the messages to the trusted party, so that it cannot tell the difference). When the trusted party gets a query m from a participant i, $\mathbb{A}^2$ adds m in the FIFO queue q_i. Each time $t + 1$ queues are non-empty, $\mathbb{A}^2$ removes one query from each of them and form the list $(id_i, m_i)_{i=1}^{a}$ from the removed queries, where id stands from the querier and m stands for the message.

$\mathbb{A}^2$ then checks that each m_i such that $i \in C$ corresponds to the share s_i. If not, $\mathbb{A}^2$ aborts and returns $\perp$. If the check succeeds, $\mathbb{A}^2$ queries the tally r from the game H_2 and returns r. Finally, $\mathbb{A}^2$ returns $\mathbb{A}_4^0$'s output. Remark that if $\mathcal{S}$ makes a query to the trusted party on behalf of a corrupted party i, but with a different input than the share s_i, then the trusted party will decrypt the ballots using a different secret key than the real secret key sk. Hence the ballots of the honest voters would decrypt to an invalid voting option with overwhelming probability, by the small voting set assumption. In this case, the trusted party would also abort. Hence, $\mathbb{A}^2$ plays a computationally indistinguishable simulation of H_1 to $\mathbb{A}^1$, so that it wins H_2 with the same probability as $\mathbb{A}^1$ wins H_1, with up to a negligible difference.

The third step of the proof is to consider $n - n_c + 1$ hops H_3^i for i going from $n - n_c$ down to 1. In game H_3^i, the adversary is given the first i elements of BB at line 8 of the game, instead of the whole BB. For all adversary $\mathbb{B}$, we denote $p_{\mathbb{B}}^i(\kappa, par) = \Pr(H_3^i(\kappa, \mathbb{B}, par) = 1)$ for all parameters $par = n, n_c, a, t, C, v_2, \cdots, v_n$. When the context is clear, we drop the parameters.

Suppose that there exists i and an adversary $\mathbb{B}_{i+1}$ for game H_3^{i+1} such that for all adversary $\mathbb{B}$ for game H_3^i, there exists a choice of parameter par such that $p_{\mathbb{B}_{i+1}}^{i+1}(\kappa, par) - p_{\mathbb{B}}^i(\kappa, par)$ is non-negligible. In particular, we consider the following adversary $\mathbb{B}_i$ for game H_3^i, which forwards its inputs to $\mathbb{B}_{i+1}$ and outputs $\mathbb{B}_{i+1}$'s outputs. For this purpose, $\mathbb{B}_i$ needs to complete its view of BB with an additional valid ballot for a fixed voting option ε. By assumption, there exists a choice of parameter $par = n, n_c, a, t, C, v_2, \cdots, v_n$ such that $p^{i+1}_{\mathbb{B}_{i+1}}(\kappa, par) - p^{i}_{\mathbb{B}_i}(\kappa, par)$ is non-negligible.

Then we construct an adversary $\mathbb{B}$ which breaks the IND-PA0 security from [9]. First, on input pk, it generates the shares $(s_i)_{i \in C}$ at random and queries v_0, v_1 from $\mathbb{B}_{i+1}$. It picks a random bit b and sets BB to $\{\mathtt{vote}(v_b)\}$. For j going from 2 up to i, it adds $\{\mathtt{vote}_{pk}(v_i)\}$ to BB. Then, it outputs (ε, v_{i+1}) to the IND-PA0 game, where ε is the arbitrary fixed voting option chosen by $\mathbb{B}_i$. The challenger answers with some y. Then $\mathbb{B}$ adds $\{y\}$ to BB and gives it to $\mathbb{B}_{i+1}$ which answers with some $(X_i)_{i > n - n_c}$. $\mathbb{B}$ adds to BB the valid ciphertexts which are not already on the board. In the IND-PA0 game, $\mathbb{B}$ sets $\boldsymbol{c}$ to the subset of the (X_i)'s which were added to BB. It gets back the decryption of the elements of $\boldsymbol{c}$ as $\boldsymbol{p}$. It computes the tally r from the plaintexts $\boldsymbol{p} \bigcup \{v_b\} \bigcup \{v_j; j \in [2, n - n_c]\}$. It query $\mathbb{B}_{i+1}$'s guess b' and outputs 1 if $b' = b$, 0 otherwise.

Now, with probability $1/2$, the IND-PA0 game picked v_{i+1}, so that $\mathbb{B}$ wins if it outputs 1. In this case, it played a perfect simulation of game H_3^{i+1} to adversary $\mathbb{B}_{i+1}$, and hence wins with probability $p^{i+1}_{\mathbb{B}_{i+1}}(\kappa)$. With probability $1/2$, the IND-PA0 game piked ε and $\mathbb{B}$ wins if it outputs 0. In this case, it played $\mathbb{B}_i$'s simulation of game H_3^{i+1} to $\mathbb{B}_{i+1}$, so that it wins with probability $1 - p^{i}_{\mathbb{B}_i}(\kappa)$. Hence it wins the IND-PA0 game with the advantage $1/2(p^{i+1}_{\mathbb{B}_{i+1}}(\kappa) - p^{i}_{\mathbb{B}_i}(\kappa))$, which is non-negligible.

However, the `vote` procedure defines a NM-CPA encryption scheme, as shown in [12], and the IND-PA0 game is shown equivalent to the NM-CPA game in [9]. We deduce that there exists an adversary $\mathbb{B}_1$ for game H_3^1 which wins with the same probability as $\mathbb{A}^2$ wins H_2.

Finally, we use a final game hop which is identical to the ideal game. More precisely, the adversary is no longer given the ballot $\mathtt{vote}_{pk}(v_b)$ at line 8. With a similar reduction to IND-PA0 as above, it is straightforward to show that there exists an adversary $\mathbb{A}^4$ which wins the ideal game with the same advantage as $\mathbb{B}_1$ wins H_3^1. □

J.4 Universal verifiability of TH-voting

We now show that our TH-voting schemes satisfy universal verifiability. In our setting, this property is expressed by Definition J.5, where O stands for oracle access to Algorithms 65, 64 and 63.

Definition J.5. We say that an electronic voting protocol $(\mathtt{Setup}, \mathtt{vote}, \mathtt{isValid}, P_{\mathtt{tally}}, \mathtt{Verify})$ is universally verifiable for the result function `tally` if for all PPT adversary $\mathbb{A}$, for all t, a, the probability $\Pr(\mathtt{IdealV}(\mathbb{A}, \kappa, t, a) = 1)$ is negligible in κ, where `IdealV` is defined in Algorithm 66.

Algorithm 63: Ovote

Require: v

if *v is a valid voting option* **then**

  $BB := BB \bigcup \{\mathtt{vote}(pk, v)\}$

Algorithm 64: Ocast

Require: B

if $\mathtt{isValid}(BB, B)$ **then**

  $BB := BB \bigcup \{B\}$

Algorithm 65: Oboard

Return BB

In our setting, the universal verifiability of our schemes comes directly from the soundness of the ZKP and the correctness of the tally-hiding protocol.

Individual verifiability is straightforward in our setting since we implicitly assume that all voters verify their vote. How to achieve individual verifiability in practice is beyond the scope of this work.

Algorithm 66: IdealV

Require: $\mathbb{A}, \kappa, t, a$

$sk, pk, s_1, h_1, \cdots, s_a, h_a :=$ `Setup`(κ, t, a);
$r, \Pi := \mathbb{A}^{O}(\kappa, g, sk, pk, s_1, h_1, \cdots, s_a, h_a)$;
if `Verify`(r, Π, BB)) **then**
  $r' :=$ `tally`$((\texttt{Extract}_{sk}(B))_{B \in BB})$;
  Return $r' \neq r$
else
  Return 0