A Toolbox for Verifiable Tally-Hiding E-Voting Systems

Véronique Cortier, Pierrick Gaudry(✉), and Quentin Yang

Université de Lorraine, CNRS, Inria, Metz, France
pierrick.gaudry@loria.fr

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, plus possibly some information required by law) 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. From these building blocks, we design a fully tally-hiding scheme for Condorcet elections. Our implementation shows that the approach is practical, at least for medium-size elections. Similarly, we provide 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 some 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 for 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 is tedious by hand but easy for a computer.

Numerous electronic voting protocols have been proposed such as Helios [6], Civitas [15], or CHVote [21]. 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.

V. Atluri et al. (Eds.): ESORICS 2022, LNCS 13555, pp. 631–652, 2022.
https://doi.org/10.1007/978-3-031-17146-8_31

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. 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 is 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 re-encryption mixnets*. The encrypted ballots are shuffled and re-randomized such that the resulting ballots cannot be linked to the original ones [21,40]. 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 such systems can be subject to 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 [25] 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, including 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 [22] which shows how to compute the result in Condorcet, while [37] and [9] provide several methods for STV. They all leak some partial information, but much less than the complete set of votes. Ordinos has been extended [24] to cover various counting functions that include Borda, Hare-Niemeyer, Condorcet, and Instant-Runoff Voting (IRV, which is STV with 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 [12], the authors show how to compute Majority Judgment in MPC. All these approaches except [22] rely on Paillier encryption since it is better suited than ElGamal for the arithmetic comparison of the content of two ciphertexts.

Our Contributions. First, we revisit the existing work, exhibiting weaknesses and even flaws for some of them. For example, we discovered that the scheme proposed in [22] for Condorcet breaks vote privacy for each voter that voted blank. Moreover, we found out that the approach developed in [12] for Majority Judgement fails in not-so-rare cases.

Our second and main contribution is the design of a toolbox of MPC primitives well suited for tally-hiding schemes. We provide a precise cost analysis, with various tradeoffs in terms of message size, number of communications, and computational costs. We believe this study could be useful in other settings. As an application of our toolbox, 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. One of our first findings is that even for complex counting functions, it is possible to use Exponential ElGamal encryption instead of Paillier. This offers a much better tool support as well as new tradeoffs in terms of computational costs.

As counting functions, we first consider Condorcet-Schulze and propose the first tally-hiding scheme that allows candidates to be ranked at equality, with a quasi-linear complexity for voters (vs cubic in [24]). We also devise several efficiency/leakage compromises. We continue by considering three major counting functions: D'Hondt, Majority Judgment, and STV. For each of them, we propose the first tally-hiding schemes with no leakage.

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 the `CGate` protocol (that conditionally sets a component to 0). An important contribution of our work is to formally prove that this primitive 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 h for 20 candidates (and 1024 voters). In contrast, the code [24] 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. For example, we assumed here that the desired result of the election is exactly the set of winners but our toolbox could be used to reveal more information if needed (for example, it could tell that candidate A receives between 55% and 60% of the votes).

Outline of the Paper. We start (Sect. 2) by explaining how to obtain all basic arithmetic operations in MPC on encrypted integers, using El Gamal encryption and we show that it is UC-secure. Figure 1 in Appendix provides the cost of each basic function, that allows to derive the cost of any complex function, obtained by composition. In Sect. 3, we apply our toolbox to the Condorcet-Schulze tally function and we provide a detailed computational cost analysis, and compare it with previous approaches (one of them suffering from a privacy breach). Due to space constraints, we overview in Sect. 4 how our toolbox can be applied to single voting, STV and Majority judgement, again comparing our approach to previous techniques. The exact cost of each tally function is given in Appendix. We show in Sect. 5 that, in all these cases, we can derive a privacy preserving voting protocol.

A companion report [18] provides a more detailed overview on how our toolbox can be applied to build MPC secure tally functions for Condorcet, single voting, STV, and Majority judgement. It also contains all the detailed algorithms and security proofs. Our source code for the implementation is available in [4].

2 Description of the Tally-Hiding Toolbox

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.

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 $t+1$ trustees is honest, the result will be obtained; 2) if at most t trustees are corrupted, only the result is known (no side-information is leaked); 3) even if all a trustees are dishonest, if the transcript is valid then the result is guaranteed to be correct.

2.1 Encryption Scheme: Paillier vs ElGamal

Paillier and Exponential ElGamal are the most popular asymmetric encryption schemes that are homomorphic, where multiplication or division of ciphertexts correspond to addition or subtraction of the corresponding cleartexts. They therefore allow re-encryption, by multiplying by an encryption of 0. These are properties at the heart of the MPC protocols.

When Exponential ElGamal encryption can be used, it offers several advantages 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, in our context, 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 [16]. The situation is more complex in Paillier. The general threshold key distribution scheme [23] is of high complexity. A more efficient scheme exists [29], but only with a honest majority. Another reason for preferring ElGamal is that the underlying security assumption (Decisional Diffie Hellman) can be considered as more standard than the one for Paillier (Decisional n-Residuosity).

On the other hand, Paillier offers more possibilities when it comes to MPC. Therefore, in general, an algorithm based on the Paillier scheme requires less exponentiations than when based on ElGamal; however, exponentiations are more costly. Later on, we will provide the complexities of our algorithms measured by the number of exponentiations. When comparing these figures, one should remember the respective costs in ElGamal and in Paillier, that we estimate now.

Table 1. Estimation of the number of exponentiations per second in Paillier and ElGamal settings.

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

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, and usually prime files are preferred.

For the Paillier scheme, the security relies on a problem that is not harder than integer factorization of an RSA number n. Since the complexity of the best known factoring algorithm is hard to evaluate, there is no strict consensus about the size of n for a 128-bit security level. Generally, this goes around 3072 bits. In Table 1, we estimate the number of exponentiations per second, based on a medium level of optimization, for a native implementation on a modern processor (based on OpenSSL, using RSA for Paillier emulation), and for a JavaScript implementation in a browser (based on `libsodium.js` and JavaScript BigInt).

2.2 Key Elements of ElGamal-Based MPC

Our toolbox contains subroutines for both ElGamal and Paillier, but in this description, we concentrate on ElGamal, since in the end we find it more suitable for e-voting. In ElGamal-based MPC, some operations seem to be impossible to be performed efficiently, for instance comparing two encrypted integers. In order to evaluate any counting function, we will therefore restrict ourselves to manipulating encrypted bits. By the homomorphic property, dividing an encryption of 1 by a ciphertext provides an easy and cheap `Not` gate. The main workhorse of our toolbox is a primitive from [32] called conditional gate, that provides an `And` gate. We readily deduce that a `Nand` gate is available, which is complete, and therefore any function can be implemented by working on encrypted bits.

Algorithm 1: `CGate`

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

Conditional Gates. A conditional gate [32] is a protocol which allows to compute, from two encryptions of x and y, an encryption of xy. It is named this way because y needs to lie in a known binary domain. We propose the `CGate` protocol (Algorithm 1), adapted from [32] so that we could prove its security in the SUC framework (see Sect. 2.4). This protocol is the main building block of our MPC protocols, which consist of `CGate` protocols and homomorphic operations. Note that each participant of a `CGate` protocol produces a Zero Knowledge Proof (ZKP) that guarantees that the correct computations were performed (including at steps 7 and 8 for example). Those ZKP can later form a transcript which

can be used to verify the output of the protocol. Their exact description can be found in [18]. By concatenating the transcripts of all the `CGate` subprotocols, a transcript for verifiability can be obtained for all our MPC protocols.

Encrypting an Integer. When ElGamal is used for a homomorphic tally, the result is an integer that is directly encrypted thanks to a *natural encoding*. We can still add and subtract encrypted values, but most other operations (comparison, multiplication, ...) are more difficult, or even impossible. Therefore, in our protocol we will keep intermediate integer values encrypted in the *bit-encoding*, where each bit of the integer is separately encrypted. 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 done using the homomorphic property and the Horner scheme. The other direction is impossible in the ElGamal setting. However, if the Paillier scheme is used, converting from the natural to the bit-encoding is still possible [33].

2.3 MPC Toolbox

We now present the building blocks that constitute our toolbox, such as addition, multiplication and comparison. Those building blocks can be combined to evaluate any counting function without leaking anything but the result. For each of them, we study their cost, which are summarized in the Fig. 1 of the Appendix. The computation cost is the number of exponentiations, but for the communications, we distinguish the broadcast and the rounds of communications. 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.

We believe that this toolbox is of independent interest and could be used in contexts beyond tally-hiding protocols. This gathers results from various domains, first on ZKP [11,27,30,40] and MPC [8,19,28,32–34] but also on hardware circuits [10]. We distinguish between the functionality (*e.g.* addition) and the protocol that realizes it since different options may be considered, leading to different trade-offs in terms of communications and computations. For some building blocks, we propose our own protocols, improving existing propositions.

Branch-Free Tools. In MPC, the algorithms must be implemented in a branch-free setting, because the result of a test cannot be revealed. We consider the following conditional operations, where B is an encrypted bit.

- $\texttt{CondSetZero}(X, B)$, $\texttt{CondSetZero}^{\text{bits}}(X^{\text{bits}}, B)$: conditionally sets to zero by outputting a re-encryption of X if B is an encryption of 1, or of $\texttt{Enc}(0)$ otherwise. In the bit-encoding setting, each bit of X is treated separately.
- $\texttt{Select}(X, Y, B)$, $\texttt{Select}^{\text{bits}}(X^{\text{bits}}, Y^{\text{bits}}, B)$: selects according to bit by outputting a re-encryption of X if B is an encryption of 0, or of Y otherwise.

– `SelectInd`$([X_i],[B_i])$: selects in array according to bits by outputting a re-encryption of X_i for the i such that B_i is an encryption of 1. This requires that $[B_i]$ is such that there is only one index i for which B_i is `Enc`(1).

The `CondSetZero` functionality is essentially just the `CGate` protocol. The other functionalities can be easily derived using the homomorphic property. If the Paillier setting is used, a more efficient realization is possible [19,34]. More details can be found in [18].

Arithmetic. Thanks to the homomorphic property, additions and subtractions are easily handled with the natural encoding. However, they are more involved with the bit-encoding [32]. We denote the corresponding functionalities `Add` and `Sub`. They can be implemented as we would do for binary circuits.

Comparison of two integers is denoted by `LT`. In bit-encoding, it can be seen as a subtraction where only the final borrow bit is needed. Similarly, we define the `Mul` functionality that can be applied to integers in the bit-encoding, following the schoolbook algorithm for bit-wise encoded integers. Finally, a frequent operation is to compute the sum of many encrypted binary values, typically to get the total number of votes for a given option. We call this operation `Aggreg`. If this is the final result before decryption, the homomorphic property is enough, but in general the result is needed in the bit-encoding format. We therefore designed a dedicated tree-based algorithm with variable precision, which improves the complexity compared to a naive approach.

The cost of many variants of all of these, with different trade-offs, are given in Appendix. We also include algorithms in the Paillier setting for which more operations are available in the natural encoding.

Shuffle and Mixnet. A tool that is of great use in our context is the verifiable shuffle [39,40], leading to mixnets. In electronic voting, the typical use of a 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 shuffle, not only on the trustees side but also on the voter's side, as shown in Sect. 3.

2.4 Security

We consider the well-known UC-framework [13] 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 [14], 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 $\mathcal{Z}$. It can corrupt a subset C of participants of size at most t, where $t \leq a$ 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 $\texttt{REAL}_{P,\mathbb{A},\mathcal{Z}}(\kappa, z)$ 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 one, 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 $\texttt{IDEAL}_{T,\mathcal{S},\mathcal{Z}}(\kappa, z)$ 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 1 (Secure computation [14]). *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,*

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

All our building blocks (except shuffle and mixnets, that are handled separately) rely on `CondSetZero` in the sense that they can all be derived as composition of this function, possibly with intermediate operations using only the homomorphic property. To compute `CondSetZero`, we consider the MPC protocol `CGate` [32] based on ElGamal, and we adapt it in order to prove, in the SUC framework, that `CGate` securely computes the trusted party $T_{\texttt{CGate}}$, 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 [14], SUC-security is not directly composable but instead requires to introduce intermediary (composable) hybrid models, where participants have an oracle 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 the full version of this paper [18].

3 Tally-Hiding Schemes for Condorcet-Schulze

The Condorcet approach is a 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 candidate 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 [5]. It first considers by "how much" a candidate is preferred, which can be reflected into the *adjacency matrix* a defined as

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

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 [35]. Note that there may be several winners according to Condorcet-Schulze. We denote by $f_{\texttt{Cond}}$ 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 Table 2.

Table 2. 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.

Version	Leakage	EG/P	Voters	Authorities		Size of the transcript
			# exp.	# exp.	# comm.	
[22]	Adj. matrix **privacy breach** [i]	EG	$5k^2$	$18nak^2$	2	$13nak^2$
[24] [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 [22] leaks, for each ballot, the number of candidates ranked at equality. In particular, who voted blank is known to everyone.
ii [24] does not allow voters to give the same rank to several candidates.
iii [24] originally does not take into account the cost of verifying the ZKP from the voters.

3.1 Ballots as Matrices

A first approach is to encode the vote as a *preference matrix* m. For each candidate i, let c_i be its rank, possibly with equality. Then $m_{i,j}$ is set to 1 if $c_i < c_j$, 0

if $c_i = c_j$ and -1 otherwise. The voters then encode their ballot as an encrypted preference matrix M. They also need to prove that M 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.

To discharge the voter from such a proof effort, in [22] the authorities shuffle each preference matrix in blocks and then decrypt them to check that it was indeed well formed. However, this yields a privacy breach, unnoted in [22]: 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 [22] is to let the voters prove the relations with a ZKP, with a cost of $O(k^3)$ exponentiations to build and to check a ballot, where k is the number of candidates. This is the approach of [24], that also assumes that voters do not place candidates at equality (the case $c_i = c_j$ is forbidden).

We propose an alternative approach in $O(k^2)$ exponentiations for both the voter and the verifier. 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^{init}. We consider a fixed encryption M^{init} of this matrix, 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":

$$m^{\mathsf{init}} = \begin{pmatrix} 0 & 1 & \cdots & 1 \\ -1 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ -1 & \cdots & -1 & 0 \end{pmatrix} \qquad M^{\mathsf{init}}_{i,j} = \begin{cases} E_1 & \text{if } i < j \\ E_0 & \text{if } i = j \\ E_{-1} & \text{otherwise.} \end{cases}$$

Assume now that a voter wishes to rank the candidates in some order, which is a permutation σ of $1, 2, \ldots, k$. Then the voter can simply shuffle M^{init} using σ. The associated proofs of a shuffle guarantee 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 [40], this requires $O(k^2)$ exponentiations for the voter. To account for candidates that have an equal rank, the voter still shuffles M^{init} 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 [24] (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 affordable, then we can further alleviate the charge of the voters, as we shall explain now.

3.2 Ballots as List of Integers

To minimize computations on the voter's side, we simply ask them to encrypt the list of integers (c_i) representing their preference. In the ElGamal setting, we 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 ZKP which prove that they encrypt only 0 or 1. This is to be compared with the k^2 encryptions when ballots are encoded as a preference matrix. To apply the Schulze method, the authorities 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 encryption of c_i then the encrypted positive preference matrix M are computed by the authorities as $M_{i,j} = \mathtt{LT}^{\mathrm{bits}}(C_i, C_j)$.

Summing up the (encrypted) matrix M_v for each voter v, we obtain the (encrypted) pairwise positive preferences matrix D. Then the authorities can apply the Schulze method in MPC from D, which can be implemented from the Floyd-Warshall algorithm [20,36]. Indeed, the latter mostly consists in computations of min/max, and translates into an MPC algorithm using the building blocks presented in Sect. 2. We denote by $P_{\mathtt{Cond}}$ the corresponding MPC protocol.

The advantage of this solution is that the load for voters remains minimal, with $O(k \log k)$ exponentiations in total. However, for the authorities, transforming each ballot into a preference matrix costs $O(k^2 \log k)$ per voter, while computing the Floyd-Warshall algorithm requires $O(k^3)$ exponentiations.

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.

Security. We denote by $T_{\mathtt{Cond}}$ the trusted party that implements $f_{\mathtt{Cond}}$ in the SUC framework. We show that $P_{\mathtt{Cond}}$ securely computes $T_{\mathtt{Cond}}$ (proof in [18]).

Theorem 1. *$P_{\mathtt{Cond}}$ securely computes $T_{\mathtt{Cond}}$ under the DDH assumption and the random oracle model (ROM).*

3.3 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 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 artifact 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 Table 2).

We ran our software on various sets of parameters. In order to compare to [24], 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 of 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 Table 3, we summarize the cost in terms of wall-clock time and the size of the transcript, measured by the program.

Table 3. Benchmark (wall-clock time and transcript size) of fully tally-hiding Condorcet-Schulze MPC computation.

Voters	5 candidates	10 candidates	20 candidates
64	1 min 50 s/49 MB	8 min 30 s/0.30 GB	45 min/1.8 GB
128	2 min 40 s/87 MB	12 m/0.51 GB	1 h 27 min/2.9 GB
256	4 min 35 s/160 MB	20 m/0.88 GB	2 h 37 min/4.8 GB
512	8 min 10 s/305 MB	34 min/1.6 GB	4 h 43 min/8.6 GB
1024	15 min/595 MB	1 h 05 min/3.1 GB	8 h 50 min/16 GB

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 [24] 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.

4 Other Counting Methods

We also provide fully leakage-free tally protocols for D'Hondt, Majority Judgment and Single Transferable Vote. We survey our findings and encodings for each counting functions. Full details are available in [18]. In particular, we prove that our tally protocols are SUC-secure by providing analogs of Theorem 1.

4.1 Single Vote

A first class of counting functions applies to the case where voters simply select some candidate(s). The typical way to determine the s winners is to count the number of votes for each candidate and select the s ones with the most votes. This is the case covered by Ordinos [25], which however suffers from a shortcoming in case of equalities: it may return more winners than the number of seats. We correct this and we show that it is possible to rely on ElGamal, 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 single candidate. Indeed, the seats need to be shared among the candidates of the different lists, according to the number of votes received. One popular technique is the D'Hondt method, which is used in practice for politically-binding elections. 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. In this case, ElGamal is a key ingredient for designing a practical tally-hiding scheme. The analysis in terms of cost is displayed in Fig. 2 of the appendix.

4.2 Majority Judgment

Majority Judgment (MJ) [7] is a method in which candidates are each given a grade, such as Excellent, Good, Poor, *etc.* Then the candidates are compared based on the sequence formed by their median grades *i.e.* the median grade, then the median obtained when the median grade is removed, and so on. It has been recently used by more than 400 000 voters in French primary elections [2]. In [12], an MPC protocol is proposed to realize MJ, but we discovered that it only implements a simplified version, called majority gauge. When the majority gauge returns a winner, then it is indeed a MJ winner but, 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, [12] can fail with probability 20% [18], 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) on 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. Our algorithm has a similar complexity to [12] while they implement a much simpler algorithm. Then we show that it is 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). The resulting computational costs are displayed in Fig. 3 in appendix. This is a good example where working with bit-encoded integers allowed to perform all the needed operations in MPC. The load for the trustees increases 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.

4.3 Single Transferable Vote

In Single Transferable Vote (STV), each voter must give a strict ordering of a subset of candidates. It consists of several rounds, during which each ballot grants a (weighted) number of votes to its first candidate. If a candidate has more votes than a quota, she is selected and any exceeding votes are transferred to the next candidate in each ballot (*i.e.* the weight of the ballot is multiplied by a transfer coefficient and the candidate is removed from all ballots). Otherwise, the candidate with the least votes is eliminated. Many variants of STV exist, depending on the way in which the votes are transferred. We took advise from Australian academics to choose an ideal version of STV, which is easy to analyze.

We discovered that even without any cryptography, the ideal STV algorithm is exponential and far from being practical. The reason is that the numerators and denominators of the fractions grow exponentially with the number of seats. On real data elections from the South New Wales election in Australia [3], it would take about one month on a personal computer to compute the result, and about 30 GB of central memory to store all the fractions.

Given that ideal STV cannot be efficiently computed in the clear, we considered a variant with rounding. In [9,37], there are three techniques to compute the STV winners, all with some leakage. Note that [37] computes the ideal STV (with no rounding) but probably because the authors did not realize that it would quickly be impractical. [24,31] cover a particular case where only one candidate is elected (IRV). Note that [24] 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 [31]. We propose a fully tally-hiding algorithm for STV, with no leakage, at a cost similar to [9,37], as displayed in Fig. 4 in appendix. To keep the cost reasonable, we re-used techniques of hardware circuits to implement efficiently the arithmetic functions.

5 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 [25], voters simply encrypt their vote following the expected format and the MPC protocol is used for tallying.

5.1 Definitions

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

- $\texttt{Setup}(\kappa, a, t)$ takes as input the security parameter κ, the number of authorities a and a threshold t. It returns $sk, pk, (s_i, h_i)_{i=1}^{a}$, respectively a key pair sk, pk and the corresponding private and public shares s_i, h_i for each authorities.
- $\texttt{vote}(pk, v)$ takes a public key pk, a vote v, and returns a ballot.
- $\texttt{isValid}(BB, B)$ 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.
- $P_{\texttt{tally}}(a, t) = P_1, \cdots, P_a$ is an MPC protocol to compute the tally.
- $\texttt{Verify}(r, \Pi, BB)$ 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 [26], 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 [26], 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 2 (vote privacy). *We say that a voting protocol* $(\texttt{Setup}, \texttt{vote}, \texttt{isValid}, P_{tally}, \texttt{Verify})$ *guarantees* vote privacy *w.r.t a result function* `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* $v_2, \cdots, v_n \in V$*,*

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

Algorithm 2: $\text{Real}^{\text{Priv}}_{\mathbb{A},P_{tally}}$

Require: $\kappa, n, n_c, a, t, C, V, v_2, \cdots, v_n$

$sk, pk, (s_i, h_i)_{i=1}^{a} :=$ $\texttt{Setup}(\kappa, a, t)$

$b \in_r \{0,1\}$; $par = pk, h_1, \cdots, h_a$

$v_0, v_1 := \mathbb{A}(\kappa, par, (s_i)_{i\in C})$

$BB := \{\texttt{vote}(pk, v_b)\}$

for $i = 2$ to $n - n_c$ **do**

  $BB := BB \bigcup \{\texttt{vote}(pk, v_i)\}$

$(X_i)_{i>n-n_c} := \mathbb{A}(BB)$

for $i > n - n_c$ **do**

  if $\texttt{isValid}(BB, X_i)$ **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: $\text{Ideal}^{\text{Priv}}_{\mathbb{B},\texttt{tally}}$

Require: $\kappa, n, n_c, a, t, C, V, v_2, \cdots, v_n$

$sk, pk, (s_i, h_i)_{i=1}^{a} := \texttt{Setup}(\kappa, a, t)$

$b \in_r \{0,1\}$; $par = pk, h_1, \cdots, h_a$

$v_0, v_1 := \mathbb{B}(\kappa, par, (s_i)_{i\in C})$

$BB := \{\texttt{vote}(pk, v_b)\}$

for $i = 2$ to $n - n_c$ **do**

  $BB := BB \bigcup \{\texttt{vote}(pk, v_i)\}$

$(X_i)_{i>n-n_c} := \mathbb{B}()$

for $i > n - n_c$ **do**

  if $\texttt{isValid}(BB, X_i)$ **then**

    $BB := BB \bigcup \{X_i\}$

$r := \texttt{tally}((\texttt{Extract}_{sk}(B))_{B\in BB})$

$b' := \mathbb{B}(r)$

Return $(b == b') \wedge (v_0, v_1 \in V)$

5.2 TH-voting

We define a voting protocol $V_{\texttt{tally}}$ for each tally function `tally` covered in our work (D'Hondt, Majority Judgment, Condorcet-Schulze, and STV), with $P_{\texttt{tally}}$ the corresponding tally-hiding protocol, in the ElGamal setting. The algorithm $\texttt{vote}_{\texttt{tally}}$ returns an encrypted ballot following the devised encoding, and a ZKP that the ballot is correctly formed. The algorithm $\texttt{isValid}_{\texttt{tally}}$ checks the ZKP and additionally ensures that the ballot is not already on the board. As explained in Sect. 2, 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 produce a ZKP Π that $P_{\texttt{tally}}$ has been performed correctly. This also defines a $\texttt{Verify}_{\texttt{tally}}$ algorithm 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.* [38]).

Theorem 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 [18]. We also prove that $V_{\texttt{tally}}$ is verifiable for a notion of verifiability similar to [17]. Note that the key step is the fact that our tally-hiding schemes guarantees universal verifiability: auditors can check that 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.

Appendix

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 [32]	$29a$	$R+4B$	$31a$
	P	Mul [34]	$10a$	$2B$	$11a$
Select	P/EG	Select	CSZ	CSZ	CSZ
SelectInd	P/EG	SelectInd	nCSZ	CSZ	nCSZ
Neg^{bits}	P/EG	Neg^{bits}	$(m-1)$CSZ	$(m-1)$CSZ	$(m-1)$CSZ
Add^{bits}	P/EG	Add^{bits} [32]	$(2m-1)$CSZ	$(2m-1)$CSZ	$(2m-1)$CSZ
	Sublinear P/EG	$\text{UFCAdd}^{\text{bits}}$	$m(\frac{3}{2}\log m+2)$CSZ	$2(\log m+1)$CSZ	$m(\frac{3}{2}\log m+2)$CSZ
Sub^{bits}	P/EG	Sub^{bits}	$(2m-1)$CSZ	$(2m-1)$CSZ	$(2m-1)$CSZ
	LT P/EG	$\text{SubLT}^{\text{bits}}$	$(2m-1)$CSZ	$(2m-1)$CSZ	$(2m-1)$CSZ
	LT+EQ P/EG	$\text{SubLT}^{\text{bits}}$	$(3m-2)$CSZ	$(2m+\log m)$CSZ	$(3m-2)$CSZ
	Sublinear P/EG	$\text{UFCSub}^{\text{bits}}$	$m(\frac{3}{2}\log m+2)$CSZ	$2(\log m+1)$CSZ	$m(\frac{3}{2}\log m+2)$CSZ
LT^{bits}	LT P/EG	$\text{SubLT}^{\text{bits}}$	$(2m-1)$CSZ	$(2m-1)$CSZ	$(2m-1)$CSZ
	LT+EQ P/EG	$\text{SubLT}^{\text{bits}}$	$(3m-2)$CSZ	$(2m+\log m)$CSZ	$(3m-2)$CSZ
	Sublinear P/EG	CLT^{bits}	$(4m-3)$CSZ	$2(\log m+1)$CSZ	$(4m-3)$CSZ
	Sublinear+EQ P/EG	CLT^{bits}	$(5m-4)$CSZ	$2(\log m+1)$CSZ	$(5m-4)$CSZ
EQ^{bits}	P/EG	EQ^{bits}	$(2m-1)$CSZ	$(\log m+1)$CSZ	$(2m-1)$CSZ
EQ	Precomp	EQH [28]	$21ma+75a$	$R+8B$	$(22m+28)a$
	P		$+4(m+1)$		
GT	Precomp	GTH [28]	$(27m+146\log m)a$	$(2R+13B)\log m$	$(28m+50\log m)a$
	P		$+8m+9a+5\log m$		$+6a$
BinExpand	P	BinExpand [33]	$12ma+53a+3m$	$R+2mB$	$(17m+21)a$
$\text{Aggreg}^{\text{bits}}$	EG	$\text{Aggreg}^{\text{bits}}$	$3n$CSZ	$(\log n+1)\log n$CSZ	$3n$CSZ
Mul^{bits}	P/EG	Mul^{bits}	$3m^2$CSZ	$2m^2$CSZ	$3m^2$CSZ
Div^{bits}	P/EG	Div^{bits}	$(3m-1)r$CSZ	$2mr$CSZ	$(3m-1)r$CSZ
$\text{MinMax}^{\text{bits}}$	naive P/EG	$\text{MinMax}^{\text{bits}}$	$(8m-2)n$CSZ	$2m\log n$CSZ	$(8m-2)n$CSZ
	sublinear P/EG	$\text{MinMax}^{\text{bits}}$	$(12m-6)n$CSZ	$2\log n(\log m+2)$CSZ	$(12m-6)n$CSZ
Mixnet	EG	[40]	$(9n+11)a$	R	$10(n+1)a$
			$+n-6$		
	P	[40]	$(8n+10)a$	R	$10(n+1)a$

Fig. 1. 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.

Version	Leakage	EG/P	Voters # exp.	Authorities # exp.	Authorities # comm.	Transcript size
Basic counting [25]	[i]	P	$5k$	$4kn + (27m + 146 \log m)k^2a$	$(4R + 26B) \log m$	$6kn + (28m + 50 \log m)k^2a$
ours (exact winners)	∅	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)	∅	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.)	∅	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.)	∅	EG	$8k$	$29ka(3n + 2m_2ks^2)$	$(m(m + \log s) + \log(ks)^2)R$	$93nak + 62m_2(ks)^2a$
D'Hondt (comp.)	∅	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.

Fig. 2. Leading terms of the cost of tally-hiding for single choice systems. s: # seats, k: # lists, a: # authorities, n: # voters, $m = \lceil \log(n+1) \rceil$, $m_1 = m + \log k$, $m_2 = m + \log(sk)$, $m_3 = m_1 + \log(\mathrm{lcm}(2, \cdots, s))$, $s' = \log(\mathrm{lcm}(2, \cdots, s))$, R: round of comm., B: broadcasts.

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

[i] [12] leaks whether the winner can be determined with the simplified algorithm.

Fig. 3. Leading terms of the cost of tally-hiding for MJ. n: # voters, $m = \lceil \log(n+1) \rceil$, k: # candidates, d: # grades, a: # authorities.

Version	Leakage	P/EG	Voters # exp.	Authorities # exp.	Authorities # comm.	Transcript size
[9, Sec. II]	[i]	EG	$10k^2$	$62nak^2$	$9kR$	$19nak^2$
[37]	[ii]	P	$5k^2$	$22nk^2am$	$2nkmR$	$11nak^2m$
[9, Sec. III.B]	[iii]	EG	$10k^2$	$62nak^2$	$9kR$	N/A
ours (naive arith.)	∅	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.)	∅	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

Fig. 4. Leading terms of the cost of tally-hiding for STV. n: # voters, k: # candidates, $m = \lceil \log(n+1) \rceil$, a: # authorities, r: precision in power of 2, $m' = m + r$, $k' = k + r$.

References

1. Condorcet Internet Voting Service (CIVS). https://civs.cs.cornell.edu/
2. The Guardian, 30 January. https://www.theguardian.com/world/2022/jan/30/peoples-primary-backs-as-taubira-as-unity-candidate-of-french-left
3. NSWEC - Election results. NSW Electoral Commision. https://pastvtr.elections.nsw.gov.au/SG1901/LC/State/preferences
4. Source code of prototype implementation of Section 3. https://gitlab.inria.fr/gaudry/THproto
5. Ubuntu IRC council position. https://lists.ubuntu.com/archives/ubuntu-irc/2012-May/001538.html
6. Adida, B.: Helios: Web-based Open-Audit Voting. In: USENIX (2008)
7. Balinski, M., Laraki, R.: Majority Judgment: Measuring Ranking and Electing. MIT Press (2010)
8. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In: PODC. ACM (1989)
9. Benaloh, J., Moran, T., Naish, L., Ramchen, K., Teague, V.: Shuffle-Sum: coercion-resistant verifiable tallying for STV voting. IEEE Trans. Inf. Forensics Secur. **4**, 685–698 (2010)
10. Brent, R., Kung, H.: A regular layout for parallel adders. IEEE Trans. Comput. **C-31**(3), 260–264 (1982)
11. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short Proofs for Confidential Transactions and More. In: S&P 2018 (2018)
12. Canard, S., Pointcheval, D., Santos, Q., Traoré, J.: Practical strategy-resistant privacy-preserving elections. In: Lopez, J., Zhou, J., Soriano, M. (eds.) ESORICS 2018. LNCS, vol. 11099, pp. 331–349. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98989-1_17
13. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: FOCS (2001)
14. Canetti, R., Cohen, A., Lindell, Y.: A simpler variant of universally composable security for standard multiparty computation. In: CRYPTO (2015)
15. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: toward a Secure Voting System. In: S&P (2008)
16. Cortier, V., Galindo, D., Glondu, S., Izabachene, M.: Distributed ElGamal à la Pedersen - application to helios. In: WPES (2013)
17. Cortier, V., Galindo, D., Glondu, S., Izabachène, M.: Election verifiability for Helios under weaker trust assumptions. In: Kutyłowski, M., Vaidya, J. (eds.) ESORICS 2014. LNCS, vol. 8713, pp. 327–344. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11212-1_19
18. Cortier, V., Gaudry, P., Yang, Q.: A toolbox for verifiable tally-hiding e-voting systems. Cryptology ePrint Archive, Report 2021/491 (2021)
19. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–300. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_18
20. Floyd, R.W.: Algorithm 97: shortest path. Commun. ACM **5**, 345 (1962)
21. Haenni, R., Koenig, R.E., Locher, P., Dubuis, E.: CHVote System Specification. Cryptology ePrint Archive, Report 2017/325 (2017)

22. Haines, T., Pattinson, D., Tiwari, M.: Verifiable homomorphic tallying for the Schulze vote counting scheme. In: Chakraborty, S., Navas, J.A. (eds.) VSTTE 2019. LNCS, vol. 12031, pp. 36–53. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-41600-3_4
23. Hazay, C., Mikkelsen, G.L., Rabin, T., Toft, T., Nicolosi, A.A.: Efficient RSA Key generation and threshold Paillier in the two-party setting. J. Cryptol. **32**(2), 265–323 (2018). https://doi.org/10.1007/s00145-017-9275-7
24. Hertel, F., Huber, N., Kittelberger, J., Kuesters, R., Liedtke, J., Rausch, D.: 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 (2021)
25. Kuesters, R., Liedtke, J., Mueller, J., Rausch, D., Vogt, A.: Ordinos: a verifiable tally-hiding e-voting system. In: EuroS&P (2020)
26. Küsters, R., Truderung, T., Vogt, A.: Verifiability, privacy, and coercion-resistance: new insights from a case study. In: S&P (2011)
27. Lipmaa, H.: On Diophantine complexity and statistical zero-knowledge arguments. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 398–415. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40061-5_26
28. Lipmaa, H., Toft, T.: Secure equality and greater-than tests with sublinear online complexity. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013. LNCS, vol. 7966, pp. 645–656. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39212-2_56
29. Nishide, T., Sakurai, K.: Distributed Paillier cryptosystem without trusted dealer. In: Chung, Y., Yung, M. (eds.) WISA 2010. LNCS, vol. 6513, pp. 44–60. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-17955-6_4
30. Poupard, G., Stern, J.: Security analysis of a practical "on the fly" authentication and signature generation. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 422–436. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054143
31. Ramchen, K., Culnane, C., Pereira, O., Teague, V.: Universally verifiable MPC and IRV ballot counting. In: Goldberg, I., Moore, T. (eds.) FC 2019. LNCS, vol. 11598, pp. 301–319. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32101-7_19
32. Schoenmakers, B., Tuyls, P.: Practical two-party computation based on the conditional gate. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 119–136. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30539-2_10
33. Schoenmakers, B., Tuyls, P.: Efficient binary conversion for Paillier encrypted values. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 522–537. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_31
34. Schoenmakers, B., Veeningen, M.: Universally verifiable multiparty computation from threshold homomorphic cryptosystems. In: Malkin, T., Kolesnikov, V., Lewko, A.B., Polychronakis, M. (eds.) ACNS 2015. LNCS, vol. 9092, pp. 3–22. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-28166-7_1
35. Schulze, M.: A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method. Soc. Choice Welf. **36**, 267–303 (2011). https://doi.org/10.1007/s00355-010-0475-4
36. Warshall, S.: A theorem on Boolean matrices. J. ACM **9**, 11–12 (1962)
37. Wen, R., Buckland, R.: Mix and Test Counting in Preferential Electoral Systems. University of New South Wales, Technical report (2008)

38. Wikström, D.: Universally composable DKG with linear number of exponentiations. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 263–277. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30598-9_19
39. Wikström, D.: A sender verifiable mix-net and a new proof of a shuffle. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 273–292. Springer, Heidelberg (2005). https://doi.org/10.1007/11593447_15
40. Wikström, D.: A commitment-consistent proof of a shuffle. In: Boyd, C., González Nieto, J. (eds.) ACISP 2009. LNCS, vol. 5594, pp. 407–421. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02620-1_28