Fully Tally-Hiding Verifiable E-Voting for Real-World Elections with Seat-Allocations

Carmen Wabartha[*], Julian Liedtke[×], Nicolas Huber[×], Daniel Rausch[×], and Ralf Küsters[×]

University of Stuttgart
[*]`st161329@stud.uni-stuttgart.de`
[×]`firstname.secondname@sec.uni-stuttgart.de`

Abstract. Modern e-voting systems provide what is called verifiability, i.e., voters are able to check that their votes have actually been counted despite potentially malicious servers and voting authorities. Some of these systems, called tally-hiding systems, provide increased privacy by revealing only the actual election result, e.g., the winner of the election, but no further information that is supposed to be kept secret. However, due to these very strong privacy guarantees, supporting complex voting methods at a real-world scale has proven to be very challenging for tally-hiding systems.
A widespread class of elections, and at the same time, one of the most involved ones is parliamentary election with party-based seat-allocation. These elections are performed for millions of voters, dozens of parties, and hundreds of individual candidates competing for seats; they also use very sophisticated multi-step algorithms to compute the final assignment of seats to candidates based on, e.g., party lists, hundreds of electoral constituencies, possibly additional votes for individual candidates, overhang seats, and special exceptions for minorities. So far, it has not been investigated whether and in how far such elections can be performed in a verifiable tally-hiding manner.
In this work, we design and implement the first verifiable (fully) tally-hiding e-voting system for an election from this class, namely, for the German parliament (Bundestag). As part of this effort, we propose several new tally-hiding building blocks that are of independent interest. We perform benchmarks based on actual election data, which show, perhaps surprisingly, that our proposed system is practical even at a real-world scale. Our work thus serves as a foundational feasibility study for this class of elections.

1 Introduction

E-voting is of rising interest. In order to ensure secure and correct elections, modern e-voting systems are designed to be (end-to-end) verifiable [1, 3, 4, 7–9, 18–20, 22, 26], that is, voters should be able to check that their votes/ballots were submitted correctly, and voters, election officials, and even external observers should be able to check that the election result corresponds to the votes that were cast. A stronger notion of verifiability is accountability, which states that, if the result turns out to be incorrect, then a misbehaving party causing this mistake can be identified and be held accountable. A very common method for election systems to achieve verifiability is by publishing the full tally, which consists of all (potentially aggregated) individual votes, along with additional evidence, such as zero-knowledge proofs (ZKPs), which proves that the tally was computed correctly. With the knowledge of the full tally, everyone is able to compute the actual election result, e.g., the winner of the election, and check whether this corresponds to the claimed election result.

More recently, verifiable tally-hiding e-voting systems (e.g. [5,6,10,14,16,17,21,25,29]), have been proposed that defer from revealing the full tally. They are rather designed to only publish the actual election result, e.g., the winning candidate(s) of an election, and as little further information as possible (ideally none), while the correctness of the election result can still be verified. Following the terminology of [17], tally-hiding systems can be divided into three classes: Fully tally-hiding systems (e.g., [6, 10, 16, 21]) are the strongest ones as they reveal only the election result. Publicly or partially tally-hiding systems (see, e.g., [17, 25]) are more relaxed in that they reveal some information beyond the election result, possibly only to certain parties. As discussed for example in [10, 16, 17, 21], tally-hiding systems offer several attractive features such as improved ballot privacy for voters, avoiding embarrassment or weakening of candidates, protection against a specific class of coercion attacks called Italian attacks [5, 15], and preventing Gerrymandering. So far, it has been shown that simple election schemes can be performed at a large scale, even in a fully tally-hiding manner. However, due to the strong privacy requirements, more complex voting methods have proven to be a challenge for all types of tally-hiding systems, with some types of elections even turning out to be practical only for very few candidates and/or voters (cf. Section 6).

A very important class of elections in practice is *parliamentary election with party-based seat allocation* as carried out by many countries around the world. These are among the most complex types of elections: They usually involve millions of voters, dozens of parties, hundreds of individual candidates, and hundreds of electoral constituencies. In some cases, voters have not just one but multiple votes that they can distribute among parties and possibly also individual candidates. Sophisticated multi-step algorithms are used to compute the election result, i.e., the assignment of seats to individual candidates. An important component for this process is a so-called *seat allocation method*, which takes as input a number of available seats and a set of parties with their total number of votes and then computes the number of seats assigned to each party. While a crucial part this seat allocation method is only a small step in the computation of the actual election result. Additional steps are taken, e.g., to combine the results of different constituencies to distribute seats that are directly allocated to individual candidates instead of just parties, to take into account minimum vote counts for parties before they are assigned any seats, and to include special exceptions for minorities. Furthermore, the seats assigned to each party need to be mapped to individual candidates, typically according to party candidate lists for each constituency and weighted by how many votes a party has obtained in the respective constituency. In some cases, even the size of the parliament is modified while computing the election result, possibly only after the seat allocation method has already been computed to more closely reflect the vote distribution.

Perhaps due to this intimidating complexity, so far, it has not been investigated *whether and in how far this class of elections can be performed in a tally-hiding manner, and whether this is possible even at the same scale in terms of voters, parties, candidates, and constituencies as needed in real-world elections*. There are only a few existing works that propose tally-hiding algorithms for computing certain seat allocation methods, namely, the d'Hondt method [10] and the Hare-Niemeyer method [16]. As explained above, while seat allocation methods are important components, they constitute just a small portion of the entire election scheme, and hence, these prior works do not answer the above question. In this work, we therefore, for the first time, investigate this open research question.

Contributions. More specifically, we design, implement, and benchmark the *first verifiable (and even accountable) fully tally-hiding e-voting system for a major real-world party-based parliamentary election*, namely, the election of the German parliament (Bundestag). Perhaps surprisingly, and as our main insight, with this system, we are able to *show that such a parliamentary election scheme with party-based seat allocation can actually be performed in a verifiable, fully tally-hiding manner at a real-world scale.* Our system supports the strongest level of tally-hiding, namely full tally-hiding. That is, if desired, one can reveal only the allocation of individual candidates to seats and the number of voters who cast a vote and nothing else to anybody. But one can also easily relax the kind of information that is revealed, e.g., by additionally publishing the winners of individual constituencies.

On a technical level, to obtain our voting system, we follow and slightly modify a generic approach for constructing verifiable fully tally-hiding systems, namely the Ordinos framework [21]. The Ordinos framework provides a general blueprint for the structure of such systems. Some components in this blueprint are unspecified and have to be filled in by protocol designers on a case-by-case basis to obtain a concrete instantiation of Ordinos that can perform an election for a specific voting method. It has been shown in [21] that, as long as those components meet specific requirements, the overall system/instantiation is a secure verifiable, fully tally-hiding e-voting system. The main challenge lies in constructing those components for a specific voting method in such a way that they provide all expected security properties while achieving practical performance.

The most important and also most difficult to design component is a publicly verifiable secure multi-party computation (MPC) protocol that computes the election result for the German parliament from the set of (encrypted) ballots. Due to the inherent complexity and scale of this election, this requires special care to obtain not just a theoretically secure but also a practically efficient system. Specifically, we first propose several MPC building blocks, including the first MPC subroutine for computing the Sainte-Laguë seat allocation method used for parliamentary elections, not just in Germany but also in, e.g., Indonesia, New Zealand, Nepal, Sweden, Norway, and Kosovo. Based on these building blocks, we then construct an efficient MPC protocol that performs the entire election evaluation for the German parliament. Along the way, we evaluate different options for designing our algorithms and propose several novel optimizations to improve the overall efficiency. We note that many of our ideas and building blocks, such as our MPC protocol for the Sainte-Laguë method, are of interest also for other parliamentary election schemes since such elections often use similar concepts and components.

The overall practicality of e-voting systems following the Ordinos approach is determined essentially only by the performance of the MPC component. Hence, to evaluate the performance and identify potential limitations of our system, we have implemented our full MPC protocol for electing the German parliament and performed extensive benchmarks based on actual real-world election data consisting of the votes of all respective constituencies. Our solution needs about a day to compute the election result, which is within the usual time frame expected for this election, thus demonstrating that our MPC protocol is practical even for such a complex large-scale political election.

Altogether, our results serve as a foundational feasibility study for (fully) tally-hiding elections for the important class of parliamentary elections with party-based seat allocation. Of course, as can be seen in countries already using or aiming at online elections, establishing and actually deploying a full-fledged ready-to-use system in the real world requires a huge effort beyond studying feasibility. Future deployments can build on our results by considering further aspects of parliamentary elections that are out of scope of this work, such as deciding which parties run the election in a distributed fashion, tackling the risk of voter coercion, establishing procedures for handling voter complaints, etc.

Structure. We recall the Ordinos framework in Section 2. In Section 3, we present novel building blocks that we have constructed to realize the voting methods in this work. In Section 4, we present the Sainte-Laguë method, including a novel variant to compute the Sainte-Laguë seat allocation and different tally-hiding algorithms to compute the Sainte-Laguë method. We present our voting system for the German Bundestag in Section 5. We discuss related work and conclude in Section 6. Our implementation is available at [28].

2 Preliminaries

Notation. We write $[n]$ to denote the set $\{0,\dots,n-1\}$. Let n_{cand} be the number of candidates/parties/choices, and let n_{votes} be the (maximal) number of votes. We will use n_{seats} to denote the number of seats that are being distributed among n_{parties} parties. With n^j_{votes} we denote the number of votes, and with n^j_{seats} the number of seats that party j has received. The format of a plain, i.e., unencrypted, ballot is defined via a finite *choice space* $C \subseteq \mathbb{N}^{n_{\mathsf{cand}}}$, i.e., a ballot assigns each candidate a number subject to constraints defined by C. For example, a single vote election where a plain ballot contains one vote for a single candidate/party/choice can be modeled via the choice space $C_{\mathsf{single}} := \{(b_0,\dots,b_{n_{\mathsf{cand}}-1}) \in \{0,1\}^{n_{\mathsf{cand}}} \mid \sum_i b_i = 1\}$. For voter j we denote her plain ballot by $v^j := (v^j_i)_{i\in[n_{\mathsf{cand}}]} \in C$.

The Ordinos Framework. The Ordinos framework was introduced in [21] as a general blueprint for constructing verifiable, fully tally-hiding e-voting systems. Systems following the Ordinos approach use a voting authority, an arbitrary number of voters, n_t trustees, an authentication server, and an append-only bulletin board (B) and roughly work as follows. In an initial *setup phase*, parameters of the election are generated and published on B, including a public key and corresponding secret key shares for an additively homomorphic t-out-of-n_t threshold public key encryption scheme $\mathcal{E} = (E,D)$. Each trustee has one secret key share and publishes a non-interactive zero-knowledge proof of knowledge (NIZKP) $\pi^{\mathsf{KeyShareGen}}$ to prove knowledge of their key share. The choice space C and the result function f_{res} of the election are published on B as well, where f_{res} takes as input a tally and outputs the corresponding election result, e.g., the candidate with the most votes. In the following *voting phase*, the voters first encrypt their ballots and then publish them on B, authenticating themselves as eligible voters with the help of the authentication server and the authentication server adds a signature to the ballot. An encrypted ballot of voter j has the form $(E_{\mathsf{pk}}(v_i^j))_{i\in[n_{\mathsf{cand}}]}$, i.e., each component of the plain ballot is encrypted separately. The encrypted ballot comes with a NIZKP π^{Enc} that proves validity of the plain ballot, i.e., $v^j = (v_i^j)_{i\in[n_{\mathsf{cand}}]} \in C$. The published encrypted ballots can be homomorphically (and publicly) aggregated to obtain an encryption of the aggregated full tally, i.e., one obtains one ciphertext on each $v_i := \sum_{j\in[n_{\text{votes}}]} v_i^j$, where v_i is the total number of votes/points that candidate/choice i obtained in the election. In the *tallying phase*, the trustees run a publicly verifiable MPC protocol $\mathsf{P_{MPC}}$ to compute f_{res}. This protocol takes as (secret) inputs the secret key shares of the trustees and the (public) encrypted aggregated tally and outputs the election result $\mathit{res} = f_{\mathsf{res}}(v_0, \ldots, v_{n_{\mathsf{cand}}-1})$. This result, along with any material that is needed to allow external parties to verify the MPC computation, is published by the trustees on B. Finally, in the *verification phase*, voters can check that their ballots appear on B, and everyone can verify that the election result res was computed correctly from the encrypted ballots by re-computing the homomorphic aggregation, checking all NIZKPs, and checking the MPC computation (which typically involves additional NIZKP verifications).

Many of the above components are not fixed by the Ordinos framework because they strongly depend on the specific election method that is to be implemented. Specifically, the following parameters and components have to be specified or constructed by a protocol designer to create an instantiation of Ordinos for a concrete election method: *(i)* the choice space C and election result function f_{res}, *(ii)* a threshold encryption scheme $\mathcal{E}$, *(iii)* NIZKPs $\pi^{\mathsf{KeyShareGen}}$ and π^{Enc}, *(iv)* a EUF-CMA-secure signature scheme $\mathcal{S}$, and *(v)* an MPC protocol $\mathsf{P_{MPC}}$ for computing the election result function f_{res}.

Voting systems following the Ordinos approach are intended to provide verifiability and full tally-hiding. As already mentioned, verifiability intuitively means that everyone can check whether the election result returned by the voting system corresponds to the actual votes. Full tally-hiding intuitively means that no one, including attackers, learns anything besides the number of submitted ballots and the final election result; this property, therefore, also implies the security notion of ballot privacy. We refer interested readers to [21] for formal definitions of both verifiability and full tally-hiding. Küsters et al. [21] have shown that if the above components defined by protocol designers meet certain properties, then the resulting Ordinos instance, indeed achieves both security notions:

Theorem 1 (Verifiability and Full Tally Hiding [21], informal). *Let $\mathcal{E}$ be an additively homomorphic threshold public-key encryption scheme $\mathcal{E}$, $\pi^{\mathsf{KeyShareGen}}$ and π^{Enc} be secure NIZKPs for $\mathcal{E}$, $\mathcal{S}$ be an EUF-CMA-secure signature scheme, and $\mathsf{P_{MPC}}$ be a publicly verifiable secure MPC protocol for computing f_{res}, i.e., if the result does not correspond to the input, then this can be detected, and at least one misbehaving trustee can be identified; this must hold even if all trustees running the MPC protocol are malicious. Then, the resulting instance of Ordinos is verifiable and fully tally-hiding.*[1]

Existing building blocks. In this work, we will use a threshold variant of the Paillier encryption scheme [11] to implement $\mathcal{E}$. Given a public key pk for this encryption scheme, there exist publicly verifiable MPC building blocks [11,16,24] that allow the owners of the corresponding secret key shares to compute the following basic operations for $a,b,c \in \mathbb{Z}_n$ (all operations are $\bmod\, n$ where n is determined by pk):

- $E_{\mathsf{pk}}(c) = f_{\mathsf{add}}(E_{\mathsf{pk}}(a), E_{\mathsf{pk}}(b))$ s.t. $c = a + b$; for brevity, we denote this operation by $\oplus$.
- $E_{\mathsf{pk}}(c) = f_{\mathsf{mul}}(E_{\mathsf{pk}}(a), E_{\mathsf{pk}}(b))$ s.t. $c = a \cdot b$, for brevity, we denote this operation by $\odot$.
- $E_{\mathsf{pk}}(c) = f_{\mathsf{gt}}(E_{\mathsf{pk}}(a), E_{\mathsf{pk}}(b))$ s.t. $c = 1$ iff $a \geq b$ and 0 otherwise.
- $E_{\mathsf{pk}}(c) = f_{\mathsf{eq}}(E_{\mathsf{pk}}(a), E_{\mathsf{pk}}(b))$ s.t. $c = 1$ iff $a = b$ and 0 otherwise.
- $(E_{\mathsf{pk}}(s_i))_{i=1}^n = f_{\mathsf{max}}((E_{\mathsf{pk}}(v_i))_{i=1}^n)$ s.t. $s_i \in \{0,1\}$ and $s_i = 1$ means that $v_i = \max_{j\in\{1,\ldots,n\}} v_j \wedge \forall j \in \{i+1,\ldots,n\} : v_j < v_i$.
- $c = f_{\mathsf{dec}}(E_{\mathsf{pk}}(c))$ s.t. $E_{\mathsf{pk}}(c)$ is an encryption of c.

[1] Full tally-hiding requires that at most $t-1$ trustees are malicious, verifiability does not require any honest trustees at all. This theorem uses further standard e-voting assumptions, such as honesty of B. We refer interested readers to [21] for full details.

The MPC building blocks for computing the above operations have a useful property, namely, encrypted outputs from one building block can be used as inputs for another building block such that the resulting combined protocol is still a secure, publicly verifiable MPC protocol [24]. In other words, they allow for building more complex protocols such as $\mathsf{P}_{\mathsf{MPC}}$ for Ordinos that meet the requirements of Theorem 1. We further note that the MPC building blocks for computing $f_{\mathsf{gt}}()$ and $f_{\mathsf{eq}}()$ proposed by [24] offer sublinear runtime as long as an upper bound $< n$ for both input values a and b is known; hence, performance drastically increases as long as a,b are known to remain small. This, in turn, also improves performance of MPC protocols based on these two building blocks, including the MPC building block for computing $f_{\mathsf{max}}()$ [16].

3 New MPC Building Blocks

In this section, based on the primitives introduced in Section 2, we describe several new publicly verifiable MPC building blocks that we need for constructing our $\mathsf{P}_{\mathsf{MPC}}$. We note that these building blocks are of independent interest.

Election methods for parliamentary elections often make use of divisions that produce fractions, which is an issue for encryption schemes and MPC protocols which operate on natural numbers, such as those from Section 2. One common approach [10,16] to deal with this is to multiply all values by the least common multiple of all divisors used in a computation such that divisions are guaranteed to always produce natural numbers. This can drastically increase the size of numbers, which in turn severely reduces the efficiency gain of the sublinear comparisons protocols $f_{\mathsf{gt}}(), f_{\mathsf{eq}}()$ from Section 2. Therefore, we instead take an alternative approach to deal with fractions by representing our values, where needed, as rational numbers consisting of a numerator n and denominator d. Encrypted rational numbers are denoted as $E_{\mathsf{pk}}^{\text{frac}}(a) := (E_{\mathsf{pk}}(a.n), E_{\mathsf{pk}}(a.d))$ where $a.n$ is the numerator and $a.d$ the denominator of a. We denote by $\textsc{Fraction}(n, d)$ the operation that creates an encrypted rational number with numerator n and denominator d (if the inputs n and/or d are not already encrypted, then they are first encrypted with public constant randomness). Based on this representation, we design and implement MPC components for basic arithmetic computations on encrypted rational numbers, including addition, multiplication, and comparisons.

Based on $f_{\mathsf{max}}()$ (see Section 2), we propose the method `getMaxFraction` that takes a list of k encrypted fractions and returns another list of the same length with $E_{\mathsf{pk}}(1)$ at the index of the maximal fraction and $E_{\mathsf{pk}}(0)$ everywhere else, where if there are multiple maxima, only the last one in the list is marked $E_{\mathsf{pk}}(1)$.

Election methods often need to deal with breaking ties. For this purpose, Cortier et al. [10] proposed an algorithm that finds the maximum in a list and additionally takes care of tie-breaking by scaling values and adding small tie-breaking values. While this scaling idea is conceptually simple, care must be taken to obtain a correct implementation. For example, in the method proposed by Cortier et al., for each party j a score is computed as $m_j = 2^{\lceil \log n_{\text{parties}} + 1 \rceil} \cdot s_j + r(j)$ and then the values m_j are compared. Here, $r(j) \in [n_{\text{parties}}]$ denotes a tie-breaking value (or *rank*) uniquely assigned to party j and the values s_j are the values that one would compare without tie-breaking (i.e., if $s_i > s_j$ then party i should always win this comparison). However, if the values s_j are allowed to be fractions, that can be very close to each other (which can occur in our setting), one can easily construct examples where this tie-breaking approach fails, i.e., $s_i > s_j$ but $m_i < m_j$. The explanation for this is that in this situation ranks are taken into account although the values s_i and s_j are different and hence no tie-breaking would be required. These cases can occur in our application described in Section 4. For this reason, we present a new algorithm for breaking ties that does not run into this issue. This algorithm additionally takes encrypted ranks $r = (E_{\mathsf{pk}}(r_i))_{i=1}^{k}$ as input, where the $(r_1, \ldots, r_k)$ form a permutation of $0, \ldots, k-1$, and first scales all ciphertexts q_i by a certain value, adds the encrypted ranking r_i to the scaled q_i, and then continues just as `getMaxFraction`. By the scaling the addition of r_i does not change the output if the q_i are not tied. But, if any of the inputs q_i are equal, then the party with the highest rank r_i will have the greater (encrypted) value after the addition.

We first show that breaking ties similarly to the method proposed in [10] works when only comparing values in $\mathbb{N}$.

```
procedure getMaxFractionByRank($l = (E^{\text{frac}}_{\mathsf{pk}}(v_i))_{i=1}^{k}, k, r = (E_{\mathsf{pk}}(r_i))_{i=1}^{k}$)
    $E^{\text{frac}}_{\mathsf{pk}}(\text{c\_max\_val}) = E^{\text{frac}}_{\mathsf{pk}}(v_1)$
    $E_{\mathsf{pk}}(\text{c\_max\_idx}) = E_{\mathsf{pk}}(1)$
    $E_{\mathsf{pk}}(\text{c\_max\_r}) = E_{\mathsf{pk}}(r_1)$
    for $i = 2, \ldots, k$ do
        $E_{\mathsf{pk}}(m_{\max}) = E^{\text{frac}}_{\mathsf{pk}}(\text{c\_max\_val}) \odot E_{\mathsf{pk}}(\text{c\_max\_val}.d) \odot E_{\mathsf{pk}}(v_i.d) \odot k \oplus E_{\mathsf{pk}}(\text{c\_max\_r})$
        $E_{\mathsf{pk}}(m_i) = E^{\text{frac}}_{\mathsf{pk}}(v_i) \odot E_{\mathsf{pk}}(v_i.d) \odot E_{\mathsf{pk}}(\text{c\_max\_val}.d) \odot k \oplus E_{\mathsf{pk}}(r_i)$
        $E_{\mathsf{pk}}(\text{set}) = f_{\mathsf{gt}}(E_{\mathsf{pk}}(m_i), E_{\mathsf{pk}}(m_{\max}))$
        $E_{\mathsf{pk}}(\text{c\_max\_val}) = E_{\mathsf{pk}}(\text{set}) \odot E_{\mathsf{pk}}(v_i) \oplus (1 - E_{\mathsf{pk}}(\text{set})) \odot E_{\mathsf{pk}}(\text{c\_max\_val})$
        $E_{\mathsf{pk}}(\text{c\_max\_idx}) = E_{\mathsf{pk}}(\text{set}) \odot E_{\mathsf{pk}}(i) \oplus (1 - E_{\mathsf{pk}}(\text{set})) \odot E_{\mathsf{pk}}(\text{c\_max\_idx})$
        $E_{\mathsf{pk}}(\text{c\_max\_r}) = E_{\mathsf{pk}}(\text{set}) \odot E_{\mathsf{pk}}(r_i) \oplus (1 - E_{\mathsf{pk}}(\text{set})) \odot E_{\mathsf{pk}}(\text{c\_max\_r})$
    result = $(f_{\mathsf{eq}}(E_{\mathsf{pk}}(i), E_{\mathsf{pk}}(\text{c\_max\_idx})))_{i=1}^{k}$
    return result
```

Fig. 1: Algorithm to find a maximum in a list of fractions, including tie breaking by rank.

Lemma 1 (Tie Breaking with Natural Numbers). *If $n_{parties}$ parties are given and each party has a value $e_i \in \mathbb{N}$ and a unique rank $r_i \in [n_{parties}]$, then the modified elements $m_i = e_i \cdot n_{parties} + r_i$ preserve the order between unequal elements but create a unique ordering between equal elements according to their ranks.*

Proof. First, it is shown that the order is preserved for different values of e_i and e_j. Without loss of generality, let $e_i < e_j$. Then for the modified elements it holds that $m_i < m_j$:

$$\begin{aligned} m_i = e_i \cdot n_{\text{parties}} + r_i \leq e_i \cdot n_{\text{parties}} + n_{\text{parties}} - 1 = \left(e_i + \frac{n_{\text{parties}} - 1}{n_{\text{parties}}} \right) \cdot n_{\text{parties}} \\ < (e_i + 1) \cdot n_{\text{parties}} \leq e_j \cdot n_{\text{parties}} \leq e_j \cdot n_{\text{parties}} + r_j = m_j \end{aligned}$$

Next, we show that a unique ordering according to the ranks is realized if e_i and e_j are equal. Without loss of generality, let $r_i > r_j$. Then the party with the higher rank gets a larger modified element m_i.

$$m_i = e_i \cdot n_{\text{parties}} + r_i = e_j \cdot n_{\text{parties}} + r_i > e_j \cdot n_{\text{parties}} + r_j = m_j$$

□

Now, we apply this to breaking ties even when considering arbitrary fractions to obtain the tie-breaking method that underlies the algorithm presented in Figure 1.

Theorem 2 (Pairwise Tie-Breaking between Fractions). *For $n_{parties}$ parties, where each party is assigned a fraction $f_i = \frac{a_i}{b_i}$ ($a_i, b_i \in \mathbb{N}$) and a unique rank $r_i \in [n_{parties}]$, tie-breaking between two fractions f_i and f_j using the modified elements $m_{ij} = f_i \cdot b_i \cdot b_j \cdot n_{parties} + r_i$ works.*

Proof. Let $e_{ij} = f_i \cdot b_i \cdot b_j$, $e_{ij} = f_j \cdot b_j \cdot b_i \in \mathbb{N}$. Then, e_{ij} and e_{ji} have the same ordering as f_i and f_j, i.e., $e_{ij} < e_{ji}$ if and only if $f_i < f_j$ and $e_{ij} = e_{ji}$ if and only if $f_j = f_i$. According to the previous theorem, the modified elements $m_{ij} = e_{ij} \cdot n_{\text{parties}} + r_i$ and $m_{ji} = e_{ji} \cdot n_{\text{parties}} + r_i$ preserve the order between unequal elements but create a unique order between equal elements according to the ranks. Since this is valid between every two fractions and the same ranks are used for each pairwise comparison, the statement also holds for the global order. □

Finally, in Figure 2 we introduce a new MPC algorithm for computing the floor division $E_{\mathsf{pk}}(\lfloor \frac{a}{b} \rfloor)$ from two encrypted natural numbers $E_{\mathsf{pk}}(a), E_{\mathsf{pk}}(b)$ and a publicly known upper bound $u \geq \lfloor \frac{a}{b} \rfloor$ of the result. Compared to the floor division MPC algorithm presented in [16], we require u but can be much more efficient by performing a binary search instead of iterating over a full set of values.

```
procedure FloorDivision($E_{\mathsf{pk}}(a)$, $E_{\mathsf{pk}}(b)$, $u$)
    $length = \text{bitLength}(u)$
    $E_{\mathsf{pk}}(lower) = E_{\mathsf{pk}}(0)$
    for $i = 1, \ldots, length$ do
        $E_{\mathsf{pk}}(j) = E_{\mathsf{pk}}(lower) \oplus E_{\mathsf{pk}}(2^{length-i})$
        $E_{\mathsf{pk}}(gt) = f_{\mathsf{gt}}(E_{\mathsf{pk}}(a), E_{\mathsf{pk}}(j) \odot E_{\mathsf{pk}}(b))$
        $E_{\mathsf{pk}}(lower) = E_{\mathsf{pk}}(lower) \oplus (2^{length-i} \odot E_{\mathsf{pk}}(gt))$
    return $E_{\mathsf{pk}}(lower)$
```

Fig. 2: Floor Division to calculate $E_{\mathsf{pk}}(\lfloor \frac{a}{b} \rfloor)$ where u is a known upper bound.

4 MPC Protocol for the Sainte-Laguë Method

The Sainte-Laguë method (also called Webster method) is a seat allocation method, i.e., a procedure that describes how a given number of seats is allocated to a set of parties depending on the number of votes each party has received. The Sainte-Laguë method is used by parliamentary elections in many countries, for example, Indonesia, New Zealand, Nepal, Sweden, Norway, Germany, and Kosovo. As part of computing the election result, these elections run the Sainte-Laguë method multiple times on different inputs. For example, the official evaluation of the final seat distribution of the German Bundestag of the election in 2021 required running the Sainte-Laguë method 23 times (in addition to several other steps, as explained in Section 1). Hence, in order to obtain an efficient tally-hiding voting system for these elections, it is crucial to design a heavily optimized MPC component for computing the Sainte-Laguë method. In this section, we first give both a general overview of the Sainte-Laguë method and then present our efficient tally-hiding MPC algorithm, including several optimizations and variations.

4.1 Computing a Sainte-Laguë Distribution

There are essentially two distinct (but provably equivalent [23]) algorithms for computing the seat allocation following the Sainte-Laguë method, one based on highest quotients and one on finding suitable denominators. Both algorithms take as input the number of seats n_{seats} to be distributed and, for each party $j \in [n_{\text{parties}}]$, the total number of votes n^j_{votes} that party j has received. They return the number of seats assigned to each party.

- **Highest-Quotients.** For $i \in [n_{\text{seats}}], j \in [n_{\text{parties}}]$ compute the quotients $q^j_i := \frac{n^j_{\text{votes}}}{2i+1}$. Let M be the list of the n_{seats} highest quotients. Then party j is assigned k seats, where k is the number of quotients in M that belong to j, i.e., quotients of the form $q^j_i, i \in [n_{\text{seats}}]$.
- **Suitable-Denominator.** Given a *suitable denominator* d, the number n^j_{seats} of seats assigned to party j is computed as $n^j_{\text{seats}} = \lfloor \frac{n^j_{\text{votes}}}{d} \rceil$, where $\lfloor \cdot \rceil$ denotes rounding to the closest integer (rounding of .5 can be chosen to be either up or down and can be chosen differently for each j). A denominator d is *suitable* if the result of this computation leads to the number of desired total seats, i.e., if $\sum_j n^j_{\text{seats}} = n_{\text{seats}}$. To find a suitable denominator, one generally starts with an arbitrary denominator d, e.g., $d = \left\lfloor \frac{\sum_j n^j_{\text{votes}}}{n_{\text{seats}}} \right\rceil$, checks the corresponding number of seats that would be assigned, and then tweaks d until a suitable value has been found.

For both algorithms, there might be ties that would need to be resolved. E.g., in the highest-quotients algorithm, there might be two equal quotients while there is only enough space left in M for one of them. In the suitable-denominator algorithm, it can happen that all suitable denominators are such that the quotients of multiple parties end on .5 and some of which need to be rounded up while others need to be rounded down to achieve an overall sum of n_{seats}. The Sainte-Laguë method does not define any specific tie-breaking mechanism. Instead, elections using this method additionally need to specify how they handle ties.

```
1: procedure AddSeatBasic($(q = E^{\text{frac}}_{\mathsf{pk}}(q^j_{\text{current}}))_{j=0}^{n_{\text{parties}}-1}$, $s = (E_{\mathsf{pk}}(n^j_{\text{seats}}))_{j=0}^{n_{\text{parties}}-1}$)
2:     $t = (E_{\mathsf{pk}}(m_j))_{j=0}^{n_{\text{parties}}-1}$ = getMaxFraction($q$)
3:     for $j \in [n_{\text{parties}}]$ do
4:         $d = E_{\mathsf{pk}}(q_j.d) \oplus 2 \odot t_j$
5:         $q_j$ = Fraction($E_{\mathsf{pk}}(q_j.n), d$)                ▷ Update $q$
6:         $s_j = s_j \oplus t_j$                                   ▷ Update seats ($s$)
7:     return $q, s$
```

Fig. 3: One iteration step of SLQBasic.

4.2 Tally-Hiding MPC Realization of Sainte-Laguë

We want to construct a tally-hiding MPC component that takes as inputs $E_{\mathsf{pk}}(n^j_{\text{votes}})$ for each party as well as publicly known values n_{parties} and n_{seats}, and computes the encrypted Sainte-Laguë seat distribution $E_{\mathsf{pk}}(n^j_{\text{seats}})$. As an initial insight, we observe that basing the MPC protocol on the suitable-denominator algorithm is generally very inefficient: This algorithm has to iterate over several potential denominators d until a suitable one is found. Since the number of iterations required to find d would reveal non-trivial information about the secret inputs, the MPC protocol would rather have to be constructed such that it always uses an apriori fixed number m of iterations (some of which will discard their results if a suitable divisor has already been found by a previous iteration) where m must be chosen sufficiently large such that, for all possible input sequences, a suitable divisor d is guaranteed to always be found. This worst case approximation introduces a lot of additional overhead.

Therefore, we have constructed a basic tally-hiding MPC realization SLQBasic of the Sainte-Laguë method following the highest-quotients approach: each party j is assigned its current quotient q_{current} (see the description of the highest-quotients algorithm) and seats n^j_{seats} thus far. Figure 3 shows this excerpt of a single iteration step. Note that this SLQBasic uses the fast `getMaxFraction` algorithm in all iterations, and hence, breaks ties via a fixed mechanism that always assigns the seat to the party with the highest index j.

Support for Breaking Ties by Lot. Many elections use more involved tie-breaking algorithms than the default one implemented by SLQBasic. For example, for many parliamentary elections, e.g., elections in Indonesia, Sweden, and Germany, whenever several parties are tied for a seat, then a new lot is drawn to resolve the tie. A more general tally-hiding MPC implementation SLQCustomTiebreaking for this election does not only have to support this tie-breaking mechanism but also has to keep secret whether any lots were drawn and what the result was. In particular, to build such a SLQCustomTiebreaking we need to first extend/modify the iteration step AddSeatBasic shown in Figure 3, obtaining a new subroutine AddSeatTieBreaking which takes as additional input an encrypted ranking of parties $r = (E_{\mathsf{pk}}(r_0), \ldots, E_{\mathsf{pk}}(r_{n_{\text{parties}}-1}))$ where r is a uniformly chosen permutation of $0, \ldots, n_{\text{parties}} - 1$, and then resolves ties based on that ranking.

We construct AddSeatTieBreaking by making use of `getMaxFractionByRank` as presented in Section 3. We replace the call to `getMaxFraction` in Line 2 of AddSeatBasic by our algorithm `getMaxFractionByRank` which takes as additional input the ranking r. Based on this AddSeatTieBreaking, we have constructed two versions of a SLQCustomTiebreaking MPC component which implement Sainte-Laguë. In essence, these MPC components first compute, in each iteration, an encrypted ranking r that encodes the result of tie-breaking and then use AddSeatTieBreaking with that r. There are two main optimizations that we introduce in both cases: First, for tie-breaking by lot, we run a distributed randomness generation protocol [24] for each iteration to then compute r based on the results. Since this step is input/vote independent, it can be pre-computed even before the election. Secondly, observe that if a tie occurs between m parties in one iteration of the quotient approach while there are at least m seats to be distributed, then all parties in the tie will obtain a seat in the next m iterations, i.e., it does not actually matter how this tie is broken. Hence, only ties during the last $n_{\text{parties}} - 1$ iterations need to be handled by AddSeatTieBreaking, while otherwise we use the faster AddSeatBasic algorithm.

Secret number of seats. We note that all of our MPC algorithms for the tally-hiding baseline quotient method (as well as for the new method presented next) can also be extended to run with a secret, i.e., encrypted, total number of seats n_{seats} to be assigned, as long as an upper limit of seats is known. For this one uses $\mathsf{AddSeatTieBreaking}$ in every iteration since the number of rounds in which tie-breaking is not necessary is secret. This property is used for the tally-hiding voting system for the German elections, where the Sainte-Laguë method is used, among others, on the number of seats assigned to a specific constituency, which is an intermediate result that might have to remain secret depending on the desired tally-hiding property.

4.3 Sainte-Laguë based on Floor Division

While our MPC algorithms $\mathsf{SLQBasic}$ and $\mathsf{SLQCustomTiebreaking}$ based on the highest-quotients approach are already practical in terms of efficiency, they always use n_{seats} iterations to assign all seats and thus do not scale overly well for elections where a high number of seats n_{seats} needs to be allocated. To improve performance in such cases, we propose a new algorithm for computing the Sainte-Laguë method which we call *floor division method.* Our floor division method is different from the highest-quotient and the suitable-denominator methods and allows us to construct an MPC component, called $\mathsf{SLQFloorDiv}$, that requires n_{parties} instead of n_{seats} many iterations, and thus, is more efficient in the common case that the number of seats exceeds the number of parties. In what follows, we first present the floor division method and then describe our MPC component $\mathsf{SLQFloorDiv}$.

Description of our Method. Intuitively, the main idea of our floor division method is to replace the initial iterations and hence seat assignments of the quotient method by computing an under- and an overestimation of the final seat allocation, and then run only the final (at most n_{parties} many) iterations of the quotient method to add/remove a seat from both of these initial estimations until exactly n_{seats} many seats are assigned. As we prove one of the resulting final seat distributions will be the correct Sainte-Laguë distribution, and it can be determined efficiently which one is correct.

Concretely, for each party j compute $m_j := \lfloor \frac{n^j_{\text{votes}} \cdot n_{\text{seats}}}{n_{\text{votes}}} \rceil$. For the underestimation case, we start by assigning m_j seats to party j. Note that $s^{\mathsf{min}}_{\mathsf{initial}} := \sum_{j \in [n_{\text{parties}}]} m_j$ might be smaller than n_{seats}, but not smaller than $n_{\text{seats}} - n_{\text{parties}}$. Hence, in order to distribute exactly n_{seats} seats in total, we distribute the remaining $n_{\text{seats}} - s^{\mathsf{min}}_{\mathsf{initial}}$ ($\leq n_{\text{parties}}$) seats to the parties by executing the final iterations of the highest-quotients method (and the desired tie-breaking mechanism). That is, starting from the intermediate quotients $q^j_{m_j} := \frac{n^j_{\text{votes}}}{2m_j+1}$ instead of starting from the initial q^j_0 for each party j. For the overestimation case, we start by assigning $m_j + 1$ seats to party j, which might result in at most n_{parties} additional seats being assigned beyond the desired total of n_{seats}. To remove those seats, we use a reverse variant of the highest-quotients algorithm. For this purpose, we again initialize the quotients as $q^j_{m_j}$ and then, in each iteration step, determine the minimal current quotient and remove a seat from the corresponding party (using the desired tie-breaking mechanism). Then, we update the quotient of that party by reducing the denominator by 2.[2] This continues until only a total of n_{seats} seats is distributed.

Finally, to figure out which result is the correct Sainte-Laguë distribution, we evaluate the underestimation case an additional time to compute the next seat that would be assigned. If the corresponding quotient is less than all the initial quotients $q^j_{m_j}$ of the underestimation, then the result computed based on the underestimation is the correct seat distribution. Otherwise, the result computed based on the overestimation is the correct seat distribution. We show the following result:

Lemma 2 (Correctness of $\mathsf{SLQFloorDiv}$). *The algorithm $\mathsf{SLQFloorDiv}$ as presented above is correct, i.e., always outputs the seat allocation according to the Sainte-Laguë method with the desired tie-breaking.*

[2] It might happen that all $m_j + 1$ seats are removed from a party j. In that case, this party is ignored in the following iterations. Note that this special case is non-trivial to implement in our $\mathsf{SLQFloorDiv}$ MPC component since we cannot reveal the values m_j or the quotients.

In order to prove Lemma 2, we first make the connection between the values q_{current} occurring during the execution of SLQCustomTiebreaking and the highest quotients from the generic description of the Sainte-Laguë method explicit. That is, by construction, all of the n_{seats} highest quotients must occur as a maximal value q_{current} at some iteration step during the execution of SLQCustomTiebreaking. On the other hand, if a value $q^j_{\text{current}} = q^j_i$ is not a maximal value in the first n_{seats} iteration steps of the execution of SLQCustomTiebreaking, then it does not belong to the n_{seats} highest quotients and hence party j does not receive the i-th seat.

Next, we will require the following lemma, where we follow Genssler [13]:

Lemma 3 (Over- and Underestimation). *For $j \in [n_{parties}]$ denote $m_j := \lfloor \frac{n^j_{votes} \cdot n_{seats}}{n_{votes}} \rfloor$. If in a Sainte-Laguë distribution a party i is assigned $m_i + 2$ seats, then* all *parties j receive at least m_j seats.*

Proof. We compare the quotients $q^j_{m_j}$ and $q^i_{m_i+2}$. In the case that $m_j = 0$, the statement is trivial. If $m_i = 0$ and $m_j = 1$ we have $n^j_{\text{votes}} > n^i_{\text{votes}}$, so party j must be assigned (at least) as many seats as party i. Thus, if party i is assigned $m_i + 2 = 2$ seats, then also party j is assigned (at least) $2 > 1 = m_j$ seats and the statement holds. In all other cases, we know that $m_j + m_i - 1 > 0$ and can reason as follows:

$$\begin{aligned} m_j > -(m_i+1) &\implies 2m_j m_i + 3m_j > 2m_j m_i - (m_i+1) + 2m_j \\ &\implies m_j \cdot (2m_i+3) > (m_i+1) \cdot (2m_j - 1) \\ &\implies \frac{m_j}{2m_j - 1} > \frac{m_i+1}{2m_i+3} \quad (*) \end{aligned}$$

Now, using that $n^j_{\text{votes}} \cdot \frac{n_{\text{seats}}}{n_{\text{votes}}} = m_j + a_j$ for some $a_j \in [0,1)$ and the fact that $\frac{n_{\text{seats}}}{n_{\text{votes}}} > 0$, we can conclude from $(*)$ as follows:

$$(*) \implies \frac{m_j + a_j}{2m_j - 1} > \frac{m_i + a_i}{2m_i + 3} \implies \frac{n^j_{\text{votes}}}{2m_j - 1} > \frac{n^i_{\text{votes}}}{2_i + 3},$$

i.e., $q^j_{m_j} > q^i_{m_i+2}$ and hence if q_{m_i+2} belongs to the n_{seats} highest quotients, then also all q_{m_j} belong to the n_{seats} highest quotients. □

These preparations now allow for proving the correctness of SLQFloorDiv:

Proof (of Lemma 2). We use that Lemma 3 allows us to distinguish two cases:

(i) There is a party i that in the seat allocation according to the Sainte-Laguë method receives at least $m_i + 2$ seats.
(ii) Each party j receives at most $m_j + 1$ seats in the seat allocation according to the Sainte-Laguë method.

Let us first consider case (i). In this case, by the previous lemma, each party j receives at least m_j seats. Thus, we find that all quotients q^j_i for $i \leq m_j$ must belong to the n_{seats} highest quotients and hence occur as maximal values q^j_{current} during one of the first n_{seats} iteration steps of SLQCustomTiebreaking. Since the order in which seats are allocated in the Sainte-Laguë method does not affect the final result, one can therefore simply start by assigning m_j seats to each party and then allocate the remaining seats using SLQCustomTiebreaking starting with the quotients $q^j_{m_j}$ that match the setting that each party j has been assigned m_j seats. This is exactly what the first subroutine in SLQFloorDiv does. Moreover, since in case (i) all of the values $q^j_{m_j}$ must belong to the n_{seats} largest quotients, we find that - if we follow the SLQFloorDiv algorithm - the maximal value q_{current} at the end of the computation of the first subroutine must be less than[3] all of the values $q^j_{m_j}$, since this value of q_{current} is the first one to not belong to the n_{seats} largest quotients. Hence, in case (i) the seat allocation output by SLQFloorDiv is the result from the first subroutine which as we argued is the

[3] In this comparison we take tie-breaking into account.

correct seat allocation according to the Sainte-Laguë method.

If we fall in case (ii), it can still happen that the maximal value q_{current} at the end of the computation of the first subroutine is less than or equal to all of the values $q^j_{m_j}$. Namely, this happens if all parties j receive m_j or $m_j + 1$ seats. In that case, we can again argue as above to see that SLQFloorDiv returns the correct seat allocation. However, if a party i receives less than m_i seats, then $q^i_{m_i}$ does *not* belong to the n_{seats} highest quotients and thus the check at the end of the first subroutine will fail. In particular, in this case SLQFloorDiv will return the result obtained from the second subroutine. However, since we fall in case (ii), we know that each party j has received at most $m_j + 1$ seats, so if we start by assigning $m_j + 1$ seats to each party j, we know that no party has received too few seats and can then remove the surplus seats.

In this case, if we denote $s^{\mathsf{max}}_{\mathsf{initial}} := \sum_{j\in[n_{\text{parties}}]}(m_j + 1)$ and let M^{over} be the list of quotients q^j_i for $i \leq m_j$, then the list M from the generic description of the highest-quotients method (cf. Section 4.1) consists exactly of the n_{seats} largest entries in M^{over}. Thus, if we remove the $\beta = s^{\mathsf{max}}_{\mathsf{initial}} - n_{\text{seats}}$ seats corresponding to the β smallest quotients (with tie-breaking) in M^{over}, then each party is assigned the correct number of seats according to the Sainte-Laguë method.

This is exactly what SLQFloorDiv does in the second subroutine. More precisely, by starting with the quotients $q^j_{\text{current}} = q^j_{m_j}$, the minimal value q^j_{current} is exactly the minimal value in M^{over}. Removing a seat from the corresponding party and updating the quotient by reducing its denominator by 2 then exactly corresponds to removing the minimal element from M^{over}. Repeating this process until a total of n_{seats} seats remain assigned corresponds exactly to removing the β seats corresponding to the β smallest quotients in M^{over} and hence the result of the second subroutine in this case is exactly the seat allocation according to the Sainte-Laguë method.[4] □

Tally-Hiding MPC Component. Using our building blocks described in Section 3 and the other building blocks from Section 2, most of our tally-hiding MPC protocol for computing the above algorithm for Sainte-Laguë is straightforward. The main issue left to be solved is that the number of iterations that our algorithm needs to add/subtract seats from the MPC protocol reveals initial seat assignment (this would reveal non-trivial information about the inputs/votes). We solve this by always using n_{parties} iterations in our MPC protocol, which is an upper bound on the number of iterations that are needed.

Our benchmarks of single runs of the Sainte-Laguë algorithms show that this variant of the Sainte-Laguë method is indeed faster than SLQCustomTiebreaking for larger numbers of seats and smaller numbers of parties. For example, we have the following runtime for ten parties: To distribute 60 resp. 100 seats using SLQCustomTiebreaking, the runtime is $6.7h$ resp. $12.6h$ while SLQFloorDiv only needs $4.7h$ resp. $5h$. However, for smaller numbers of, say, 20 seats, SLQCustomTiebreaking is faster with $1.6h$ instead of SLQFloorDiv, which needs $4.6h$. Further comparisons are presented in Figure 4. SLQCustomTiebreaking is linear in the number of seats while SLQFloorDiv has a larger initial overhead time but is nearly constant in the number of seats.

5 Election of the German Parliament (Bundestag)

The election of the German federal parliament, the *Bundestag*, which consists of at least 598 seats is a combination of proportional representation and first-past-the-post voting. Each voter has two votes: a constituency vote (called *first vote*) given towards an individual candidate, who is typically but not necessarily also a member of a party, and a vote for state-specific party lists (called *second vote*) which determines the proportions of parties in the parliament. The first votes are evaluated for each of the 299 constituencies individually: The candidate with the most votes wins the constituency and is guaranteed a seat in the parliament, called a *direct mandate*.[5] Each constituency belongs to exactly one of the 16 German states, say state $l \in L$ where L is the set of all states. We denote by $s^{\mathrm{d}}_{j,l}$ the total number of direct mandates that candidates of party j win in state l.

[4] Recall that we ignore quotients for parties j from which $m_j + 1$ seats have already been removed.
[5] Ties for the first place and hence the direct mandate are resolved by lot. Ties in any of the following iterations of the Sainte-Laguë method are also resolved by lot with one exception discussed below.

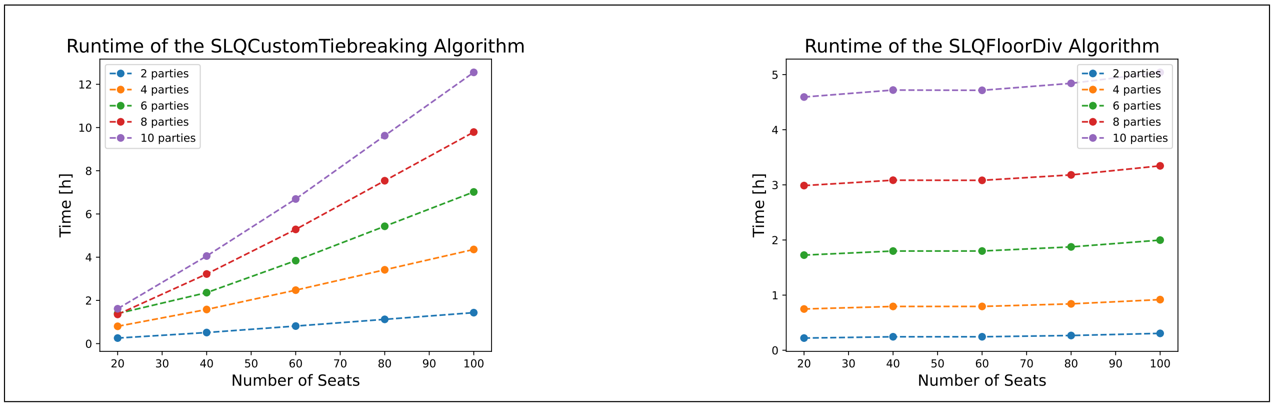

Fig. 4: Benchmarks for one execution of the two Sainte-Laguë algorithms.

Let $v_{j,l}$ be the number of second votes for party j in state l and $v_j := \sum_{l \in L} v_{j,l}$ the total number of second votes for party j. In the next step, the baseline of 598 seats of the parliament are assigned to the states in proportion to their number of inhabitants; we call this the *first top distribution* and refer to these seats as *state seats*. A party j can obtain state seats if v_j is at least 5% of all second votes, j has obtained at least $\sum_{l \in L} s^{\mathrm{d}}_{j,l} \geq 3$ direct mandates, or j represents a special minority. Let $\mathfrak{S}$ be the set of parties that are allowed to obtain state seats. Then, for each state l, the state seats are assigned to parties $j \in \mathfrak{S}$ following the Sainte-Laguë method based on $v_{j,l}$. The resulting seats are called *quota seats*, denoted by $s^{\mathrm{q}}_{j,l}$ for party j and state l. We call this distribution the *first low distribution*.

It usually happens in several states l that a party $j \in \mathfrak{S}$ wins more direct mandates and hence guaranteed seats for their candidates than the party actually receives in terms of quota seats, i.e., $s^{\mathrm{d}}_{j,l} > s^{\mathrm{q}}_{j,l}$. In such cases, the overall size of the parliament is increased, and the seat assignment to parties is updated such that *(i)* parties have enough seats for all their candidates with direct mandates and *(ii)* the number of seats given to party j in the final parliament is "close" to the Sainte-Laguë seat distribution based on v_j (up to 3 additional seats, called *overhang seats*, are tolerated). This is computed via the following procedure.

Let $s^{\min}_j := \sum_{l \in L} \left(\max(\lceil \frac{s^{\mathrm{d}}_{j,l} + s^{\mathrm{q}}_{j,l}}{2} \rceil, s^{\mathrm{d}}_{j,l}) \right)$ be a lower bound for the seats that party $j \in \mathfrak{S}$ will receive. Compute $d_{\mathrm{no}} := \min_{j \in \mathfrak{S}} \left(\frac{v_j}{s^{\min}_j - 0.5} \right)$. Then, for each party $j \in \mathfrak{S}$, it is determined whether there is a state $l \in L$ such that $t(j,l) := s^{\mathrm{d}}_{j,l} - s^{\mathrm{q}}_{j,l} > 0$. This value is also called the *threatening overhang* of party j in state l. Based on these values, one computes a set of divisors:

$$D_{\mathrm{overh}} = \{ \frac{v_j}{s^{\min}_j - i} \mid i \in \{0.5, 1.5, 2.5, 3.5\}, j \in \mathfrak{S} \text{ and } \exists l : \ t(j,l) > 0, t(j,l) + 1 > i \}.$$

Let d_{overh} be the fourth smallest element of D_{overh} and set $d := \min(d_{\mathrm{no}}, d_{\mathrm{overh}})$.

Then, as in the suitable-denominator algorithm (c.f. Section 4), party $j \in \mathfrak{S}$ receives $n^j_{\mathrm{seats}} := \lfloor \frac{n^j_{\mathrm{votes}}}{d} \rceil$ seats, where .5 is always rounded up, i.e., in this step, ties are resolved by giving every tied party a seat. The resulting distribution is called the *second top distribution*. Next, for each party $j \in \mathfrak{S}$, these n^j_{seats} are assigned to individual states following the Sainte-Laguë method weighted by $v_{j,l}$, resulting in the *second low distribution*.

In addition to those n^j_{seats} seats, (some) parties further receive $\alpha_j := s^{\min}_j - n^j_{\mathrm{seats}} (\leq 3)$ overhang seats to cover a possibly remaining surplus of direct mandates. These overhang seats are then also distributed to states according to the smallest α_j many values from the following set:

$$O^j_{\mathrm{overh}} = \{ \frac{v_{j,l}}{s^{\min}_{j,l} - i} \mid i \in \{0.5, 1.5, 2.5\}, l \in L, i < \alpha_j \}.$$

All seats assigned to party $j \in \mathfrak{S}$ in state l are then assigned to candidates as follows. First, candidates of party j that won a direct mandate in a constituency of state l obtain a seat. The remaining seats, if any, are assigned to the party candidate list for that state, starting with the first one and skipping any candidates that have already obtained a direct mandate. Finally, if there are any direct mandates for candidates that do not belong to a party from $\mathfrak{S}$, then each of these candidates receives a seat that is added to the parliament. The resulting set of seats defines the updated size of the parliament.

Our Tally-Hiding Realization. We construct our e-voting system by following the general Ordinos approach, except for one difference. The original Ordinos framework proposed in [21] was designed for elections without electoral constituencies or with just a single constituency where all votes are treated equally. We capture the existence of several constituencies in the German parliamentary elections, where the result also depends on the constituency that a vote was submitted in, via the following small changes: The list of eligible voters that is published during the setup phase now additionally assigns each voter to a constituency. Ballots are extended to additionally contain (in plain) the identifier of the constituency they were cast in such that everyone can check whether ballots were cast for the correct constituency. Encrypted ballots are aggregated per constituency and then evaluated via (the MPC component for) f_{res}. We note that this difference in settings also slightly changes the meaning of full tally-hiding: For elections without electoral constituencies, only the number of submitted votes (since this is public on B) and the final result become known. In the setting with electoral constituencies, only the number of submitted votes *per constituency* and the final result becomes known. As part of our security proof (cf. Theorem 3), we define full tally-hiding for our setting and verify that the original proofs of Theorem 1 carry over in a natural way to our setting using the same preconditions.

We hence instantiate the (modified) Ordinos approach for the German election by using the threshold Paillier encryption scheme $\mathcal{E}$, choice space $C_{\mathsf{single}} \times C_{\mathsf{single}}$, standard NIZKPs $\pi^{\mathsf{KeyShareGen}}$ and π^{Enc} [11] and any standard EUF-CMA-secure signature scheme from the literature, result function $f_{\text{res-Ger}}$ for the German parliamentary election as described above, and importantly our new MPC protocol $\mathsf{P}_{\mathsf{MPC}\text{-Ger}}$ for $f_{\text{res-Ger}}$ described next.

Constructing $\mathsf{P}_{\mathsf{MPC}\text{-Ger}}$. We have constructed $\mathsf{P}_{\mathsf{MPC}\text{-Ger}}$ using the components from Sections 2 and 3 to compute the full evaluation procedure for German parliament, as described above. This includes all small details and special cases, e.g., computing and changing the final parliament size, determining and distributing up to 3 overhang seats per party, and exempting parties from obtaining state seats iff they did not win 5% of the total second votes, won less than 3 direct mandates, and are not representing a special minority.

Of course, capturing the full complexity of the election evaluation of the German parliament in an MPC protocol $\mathsf{P}_{\mathsf{MPC}\text{-Ger}}$ comes at a hefty cost in terms of performance and hence runs the risk of becoming impractical. We have therefore spent considerable effort into carefully optimizing $\mathsf{P}_{\mathsf{MPC}\text{-Ger}}$ by, among others, the following: *(i)* Computing the election result requires multiple iterations of Sainte-Laguë. We use both $\mathsf{SLQCustomTiebreaking}$ and $\mathsf{SLQFloorDiv}$, depending on the number of seats and candidates that has to be processed in the current iteration. *(ii)* We have constructed $\mathsf{P}_{\mathsf{MPC}\text{-Ger}}$ in such a way that, as far as possible, substeps such as repeated state-wise operations can be computed in parallel. We have performed benchmarks for various numbers of threads, which demonstrate that this is a major factor in improving performance, see Table 1. *(iii)* We first compute and reveal the set of parties that will obtain at least one seat in the parliament. This allows us to tailor the following computations to this specific set of parties and thus save time by not having to perform the same operations for (dozens of) parties that will not obtain a seat. As part of Theorem 3, we argue that this construction is still a secure MPC protocol as, intuitively, the intermediate output can be computed from/is part of the final result. *(iv)* By proposing a different algorithm for computing the final number of seats for each party in the German parliament based on an encrypted divisor $d = \min(d_{\text{no}}, d_{\text{overh}})$, we can use an efficient binary search on encrypted data to obtain this value.

In more detail, regarding the last point described above, we can compute a ciphertext of the divisor d, which is then used to compute the final number of seats of the German parliament and the number of seats per party. To determine how many seats party j receives, we would then - as in the suitable-denominator algorithm - compute $n^j_{\text{seats}} = \lfloor \frac{n^j_{\text{votes}}}{d} \rceil$. However, in order to compute this in a tally-hiding way, we would have to perform a floor division, followed by suitable rounding, which is non trivial on encrypted values.

```
procedure TestSeatsMatchDivisor($n^j_{\text{seats}}$, $E_{\mathsf{pk}}(n^j_{\text{votes}})$, $E_{\mathsf{pk}}(d)$)
    $E_{\mathsf{pk}}(q) = \textsc{Fraction}(2 \odot E_{\mathsf{pk}}(n^j_{\text{votes}}), 2 \cdot n^j_{\text{seats}} - 1)$
    $E_{\mathsf{pk}}(i) = f_{\mathsf{gt}}(E_{\mathsf{pk}}(q), E_{\mathsf{pk}}(d))$
    $i = f_{\mathsf{dec}}(E_{\mathsf{pk}}(i))$
    return $1 - i$
```

Fig. 5: Test whether party j receives at least n^j_{seats} seats.

```
procedure DeterminedFinalSeats($(E_{\mathsf{pk}}(n^j_{\text{votes}}))_{j=0}^{n_{\text{parties}}-1}$, $E_{\mathsf{pk}}(d)$)
    for $j \in [n_{\text{parties}}]$ do
        $i = 1$
        $h = 0$
        while $i$ do                                          ▷ First approximation for hundreds
            $i =$ TestSeatsMatchDivisor($h$, $E_{\mathsf{pk}}(d)$, $E_{\mathsf{pk}}(n^j_{\text{votes}})$)
            if $i == 1$ then
                $h = h + 100$
        $l = h - 100$
        $u = h$
        while $l < u$ do                                      ▷ Find number of seats via binary search
            $k = \lfloor l + \frac{1}{2}(u - l) \rfloor$
            $i =$ TestSeatsMatchDivisor($k$, $E_{\mathsf{pk}}(d)$, $E_{\mathsf{pk}}(n^j_{\text{votes}})$)
            if $i == 1$ then
                $l = k$
            else
                $u = k$
            if $l + 1 = u$ then
                break
        $n^j_{\text{seats}} = l$
    return $(n^j_{\text{seats}})_{j=0}^{n_{\text{parties}}-1}$
```

Fig. 6: Computing the final seats of each party in the German Bundestag using the divisor d.

Instead, we use the fact that we can reveal the final number of seats per party, since this is public information (we will discuss this in more detail in the security proof of Theorem 3). The idea is to construct an MPC protocol that can check whether a given number of seats for party j is exceeded in a correct seat distribution based on d. Then one can perform a public binary search and use this MPC component to find the correct number of seats. To realize such a checking functionality, we use the following fact: Consider an execution of the highest-quotients algorithm for distributing n_{seats} seats, where $q_{n_{\text{seats}}}$ is the n_{seats}-th highest quotient and $q_{n_{\text{seats}}+1}$ is the next highest quotient. Then it holds true that a suitable denominator d of an execution of the equivalent suitable-denominator algorithm is in range $(2 \cdot q_{n_{\text{seats}}+1}, 2 \cdot q_{n_{\text{seats}}}]$ [2]. This implies that if a quotient q causes a seat assignment in the highest-quotient algorithm, then the suitable-denominator d of the equivalent suitable-denominator algorithm must be of size at most $2 \cdot q$. We use this fact to construct the algorithm TestSeatsMatchDivisor presented in Figure 5 that for party j takes as input a guess of seats n^j_{seats}, the number of votes $E_{\mathsf{pk}}(n^j_{\text{votes}})$, and the divisor $E_{\mathsf{pk}}(d)$ and checks whether the divisor is less or equal to $\frac{2 \cdot n^j_{\text{votes}}}{2 \cdot n^j_{\text{seats}} - 1}$. If this is not the case, then this divisor d does not assign the seat of this quotient to this party. Then, we update the guess n^j_{seats} and try again. In Figure 6 we make use of TestSeatsMatchDivisor in a binary search.

Theorem 3 (Security). *Let $\mathsf{P}_{\mathsf{MPC}\text{-}Ger}$ be our MPC protocol from above. Then, the Ordinos instance using the aforementioned components is a verifiable (and even accountable) and fully tally-hiding e-voting system for the election of the German parliament.*

Proof Sketch. We first observe that the original Ordinos framework [21] was designed for monolithic elections where nothing beyond the election result and the total number of votes shall be revealed. The election for the German Bundestag is not monolithic but rather based on multiple sub-elections in separate constituencies. This difference is reflected in our system, which publishes the total number of submitted votes *for each constituency* (instead of just the total number of all votes) as well as the final election result. Hence, privacy/full tally-hiding has a slightly different meaning in our setting and existing proofs for Ordinos do not directly apply.

For this reason, we modify the Ordinos framework in the following ways. During the setup phase, where the list id of eligible voters is published, this list is expanded by adding a constituency identifier to each id, such that it is publicly visible which id is allowed to vote in which constituency. Since the ballots published on the bulletin board contain the id, it is public information to which constituency this ballot corresponds. Furthermore, we modify the ideal protocol such that the voter under observation is voting in a specific constituency.

These changes do not affect **accountability** (and thus **verifiability**), and therefore the original proof from the original Ordinos framework for Theorem 1 still applies. However, it is public information to which constituency a ballot corresponds (the id of the voter of a ballot is contained in the ballot, and the id is published during the setup phase alongside the corresponding constituency). These changes naturally lead to a lower level of **privacy** than in the original Ordinos framework. While voting in constituencies might open new options for an adversary, as she can try to change the voting constituencies of specific ballots, she is not able to do so without being caught. Thus, when considering risk avoiding adversaries that aim for not being caught and therefore only manipulate a fixed number of ballots, these adversaries would only manipulate up to a certain number ballots overall (and not per constituency), since the global risk of being caught would be too high. Hence, to a certain extend, the privacy analysis from the original Ordinos framework carries over (with the natural restrictions of public constituency identifiers).

The above discussion allows for proving security similarly to the proofs of the original Ordinos framework. That is, if we can show that all of our cryptographic primitives including $\mathsf{P}_{\mathsf{MPC}}$ meet the requirements stated for Theorem 1 in [21], then it follows analogous to the proof in [21] that our system is (i) accountable and (ii) private/fully tally-hiding in the sense of our setting, i.e., only the final result and the total numbers of submitted votes for each constituency are revealed.

Next, observe that the primitives $\mathcal{E}$, $\pi^{\mathsf{KeyShareGen}}$, and π^{Enc} already fulfill the requirements of Theorem 1. The only thing left to show for Theorem 1 is that our new tallying protocol $\mathsf{P}_{\mathsf{MPC}}$ is secure. That is, we have to show that $\mathsf{P}_{\mathsf{MPC}}$ is a private and publicly accountable implementation of the evaluation of the German Bundestag.

Both properties follow almost directly because our MPC protocol is built from combinations of the basic components presented in Section 2. As mentioned in that section, each of these basic components already guarantees privacy and public accountability on a per-component level. Apart from very few exceptions discussed below, we have that the connections of these components, i.e., the respective inputs and outputs, are all encrypted and published on B. Due to the encryption, these intermediate results do not leak any additional information, neither to internal parties nor to external observers. Also, since the encrypted intermediate results are published, external observers can check that the output of one step is used correctly as the input to the next step. Thus, if some trustee tries to use a different input, she can be held accountable. Hence, also such combinations of the individual components remain accountable and private.

The only special cases and exceptions to the above are that we actually decrypt and reveal which parties enter the Bundestag and how many seats each party receives as an intermediate value in order to facilitate a more efficient evaluation of the following computations. This, however, is still secure: observe that the final result of the election consists of a mapping of seats to candidates. Since the mapping from candidates to parties is unique public knowledge and candidates can only receive a seat if that seat was assigned to their party, it is easy to compute the seats assigned to each party from the final election result. In particular,

# Threads per Trustee	1	2	4	8	16	32
Single-member constituency seats	40.03 h	20.04 h	10.06 h	5.06 h	2.56 h	1.32 h
Determine which parties enter the Bundestag	71 min	36 min	18 min	9 min	5 min	3 min
First low distribution	23.38 h	11.77 h	6.07 h	3.25 h	1.82 h	1.1 h
Minimal number of seats per party	11.68 h	5.84 h	2.93 h	1.46 h	0.74 h	0.36 h
Second top distribution	2.81 h	2.0 h	1.42 h	1.19 h	1.19 h	1.19 h
Second low distribution	77.06 h	38.53 h	19.4 h	12.34 h	6.3 h	5.93 h
Assigning overhang seats	6.67 h	3.33 h	2.2 h	1.14 h	1.14 h	1.14 h
Computing the final result	4 min	2 min	1 min	1 min	0 min	0 min
Total Runtime	163 h	82 h	42 h	24.3 h	13.8 h	11.1 h

Table 1: Benchmarks of the election for the German Bundestag in 2021 using real-world data available at [12]. Total runtime of the evaluation for the German parliament with real-world data from 2021 and with different numbers of available parallel threads for each Trustee.

a simulator for our MPC protocol, who is given only the election result, can compute the corresponding intermediate values that are decrypted/revealed during a run of our MPC protocol. Given this information, the simulator can then use standard techniques as described in [11, 27] for simulating the decryption of ciphertexts to the correct intermediate values. □

5.1 Benchmarks

In this section, we present the benchmarks of our evaluation of the election for the German Bundestag in 2021. We use a Paillier key of size 2048 bits. The setup for our benchmarks consists of three trustees communicating over a local network. Each trustee ran on an ESPRIMO Q957 (64bit, i5-7500T CPU @ 2.70GHz, 16 GB RAM). As in [21], the benchmarks of our MPC protocols start with an already aggregated tally. Küsters et al. [21] showed for the MPC protocols in their Ordinos instances that the number of trustees does not influence the benchmarks in a noticeable way and that, due to the sublinear communication complexity of the comparison protocols, there is no significant difference between a local network and the Internet. Since our MPC protocols are based on the same primitives and basic building blocks as used by [21], the same is also true for our MPC protocols. Our benchmarks therefore focus mostly on the number of candidates/parties which is the main factor for the performance of our protocols.

The election for the German Bundestag in 2021 had 61,181,072 eligible voters, 46,854,508 valid submitted ballots, and 47 parties with 6,211 candidates, distributed over 299 constituencies. We note that, for this, we have used the real-world data available at [12]. In Table 1 we show the benchmarks of our evaluation, including the benchmarks of all steps described in Section 5 individually. The benchmarks are separated by the number of threads that each trustee uses. Since many intermediate results can be computed in parallel (e.g., the intermediate results of the states do not affect each other), allowing the trustees to make use of multiple threads greatly affects the overall runtime of the election evaluation. For example, as shown in the table, the runtime of the second low distribution can already be cut in half by using two threads instead of one.

Each constituency has about 250,000 German inhabitants, and the number of candidates varies between 5 and 18. Therefore, there are some differences in the runtime of the first step of the evaluation, namely the

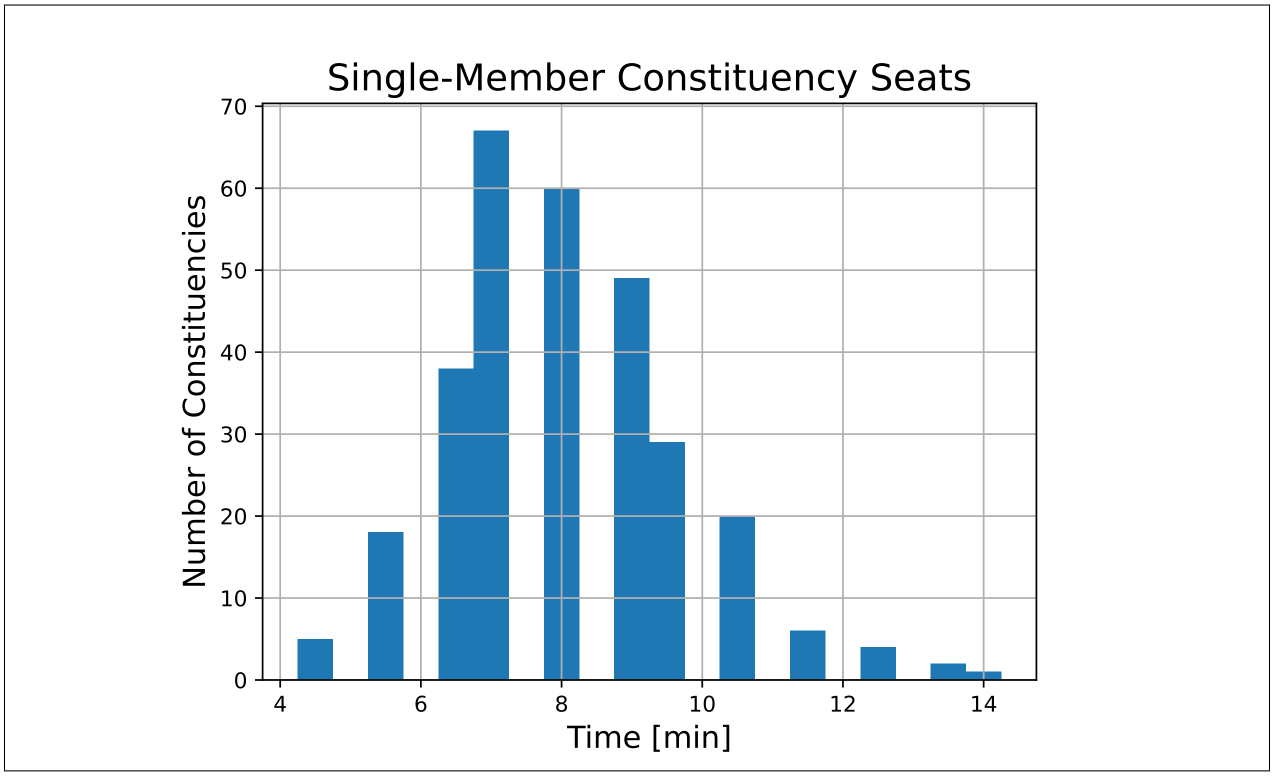

Fig. 7: Runtime of the evaluation of the constituency seats for all constituencies. The y-axis denotes how many of the 299 constituencies took the respective runtime.

computation of the single-member constituency seats. We present the runtime per constituency in Figure 7. The runtime ranges from 4.6 to 14.2 minutes and the mean value is eight minutes.

The first low distribution assigns the seats of each state to the local parties, where all states have six parties except for Schleswig-Holstein (SH) with seven and Saarland (SL) with five parties. The distribution is computed with $\mathsf{SLQFloorDiv}$. The runtime of the substeps of $\mathsf{SLQFloorDiv}$ (floor division, under- and overestimation) is presented in Figure 8. We note that the last two steps can be evaluated in parallel. The runtime also depends on the bit length of the number of seats. Without the use of parallelization, this step takes 23.4 hours. Overall, our benchmarks indicate that the tally-hiding evaluation of the German parliamentary elections are practical for real-world election data.

Verification of an election following the Ordinos approach essentially consists of two main tasks: Firstly, checking the correctness of the ballots submitted to B including verification of the ballot NIZKPs π^{Enc} for the choice space. Secondly, verifying that the MPC protocol $\mathsf{P}_{\mathsf{MPC}}$ was executed correctly.

The first task can be performed on the fly for each new ballot submitted to B while the election is still running. Notably, we use a NIZKP π^{Enc} from [11] that is standard and employed by many e-voting systems since it is very efficient and fast to verify. The second step requires checking certain data, including further NIZKPs, that is published on B while $\mathsf{P}_{\mathsf{MPC\text{-}Ger}}$ is running. Notably, all trustees also perform all of the same verification checks as part of running $\mathsf{P}_{\mathsf{MPC\text{-}Ger}}$. Hence, not only is it possible for an external observer to perform verification of $\mathsf{P}_{\mathsf{MPC\text{-}Ger}}$ in parallel to $\mathsf{P}_{\mathsf{MPC\text{-}Ger}}$ being executed. An external observer will also be done with this verification as soon as the end result is returned by $\mathsf{P}_{\mathsf{MPC\text{-}Ger}}$ because he has to perform strictly less work than the trustees running $\mathsf{P}_{\mathsf{MPC\text{-}Ger}}$. We therefore only had to benchmark the runtime of $\mathsf{P}_{\mathsf{MPC\text{-}Ger}}$ to obtain the overall time for *both computing and verifying* the election result of our system proposed in Section 5.

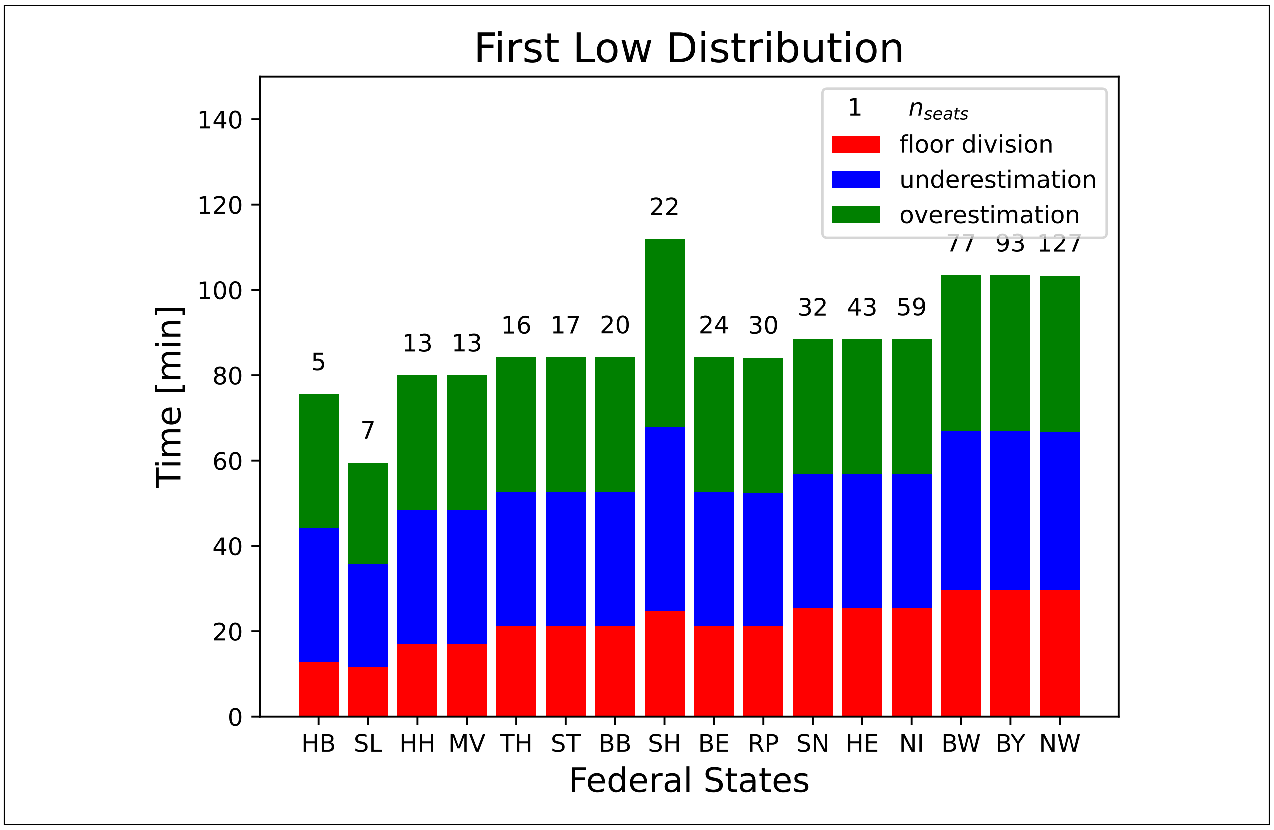

Fig. 8: Runtime of the first low distribution.

6 Related Work and Conclusion

Various tally-hiding e-voting systems for a wide variety of election types have been proposed so far, e.g., [5, 6, 10, 14, 16, 17, 21, 25, 29]. For simple types of elections such as single vote (every voter submits a single vote for the candidate of their choice with the winner(s) being the candidate(s) with the most votes), it has been demonstrated that they can be performed in a verifiable tally-hiding manner, even at a large scale (see, e.g., [10, 16, 17, 21]). However, many real-world elections, notably political ones, are much more complex and have proven to be a challenge for tally-hiding systems.

Preferential Elections: An important class of complex real-world elections are preferential ones. Tally hiding has already been studied for several voting methods from this class, with such systems typically being viable at a small to medium scale but often being impractical for large-scale applications. For example: *(i)* Recent advances in tally-hiding e-voting have managed to support instant-runoff voting (IRV) for small numbers of candidates [17, 25]. However, none of these systems remain practical for more than 6 candidates. *(ii)* Cortier et. al [10] have proposed the first MPC component that can be used to construct a fully tally-hiding voting system for single transferable vote (STV), a preferential voting method somewhat similar to IRV. However, they state that the computational cost of the resulting system would be too high for large-scale elections. *(iii)* For the Condorcet Schulze election scheme, Hertel et.al. [16] proposed an Ordinos instantiation that can handle small numbers of candidates, however, already needs about 9 days to compute an election result for 20 candidates (and essentially arbitrary numbers of voters). Cortier et. al [10] proposed an alternative tally-hiding MPC component for computing Condorcet Schulze, which is faster for small numbers of voters but can be extrapolated to also require 9 days for 20 candidates as soon as there are ∼32.000 voters.

Parliamentary Elections With Party-Based Seat Allocations: As already explained in the introduction, prior to our work, it had not been investigated for any election from this class, whether and in how far, it can be performed in a tally-hiding manner. In this work, we have proposed several new tally-hiding building blocks, as well as the first verifiable tally-hiding voting system for an election from this class, namely, the German parliament. Our results serve as an important foundational feasibility study, which, perhaps surprisingly and for the first time, demonstrates that even such a complex and large-scale real-world election can, in principle be performed in a verifiable fully tally-hiding manner. It is interesting future work to use our building blocks and ideas to construct tally-hiding voting systems for further elections from this class.

Acknowledgements: This research was supported by the DFG through grant KU 1434/11-1, by the Carl Zeiss Foundation, and by the Centre for Integrated Quantum Science and Technology (IQST).

References

1. Adida, B.: Helios: Web-based Open-Audit Voting. In: van Oorschot, P.C. (ed.) Proceedings of the 17th USENIX Security Symposium. pp. 335–348. USENIX Association (2008)
2. Beckmann, B., Trenkler, G., Pukelsheim, F.: Das Landtagswahlsystem in Nordrhein-Westfalen. Ph.D. thesis, Diploma thesis, Universität Dortmund (2006)
3. Bell, S., Benaloh, J., Byrne, M., DeBeauvoir, D., Eakin, B., Fischer, G., Kortum, P., McBurnett, N., Montoya, J., Parker, M., Pereira, O., Stark, P., Wallach, D., Winn, M.: STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System. USENIX Journal of Election Technology and Systems (JETS) **1**, 18–37 (August 2013)
4. Benaloh, J.: Verifiable Secret Ballot Elections. Ph.D. thesis, Yale University (1987)
5. Benaloh, J., Moran, T., Naish, L., Ramchen, K., Teague, V.: Shuffle-Sum: Coercion-Resistant Verifiable Tallying for STV Voting. IEEE Trans. Information Forensics and Security **4**(4), 685–698 (2009)
6. Canard, S., Pointcheval, D., Santos, Q., Traoré, J.: Practical Strategy-Resistant Privacy-Preserving Elections. In: Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018. Lecture Notes in Computer Science, vol. 11099, pp. 331–349. Springer (2018)
7. Chaum, D., Carback, R., Clark, J., Essex, A., Popoveniuc, S., Rivest, R.L., Ryan, P.Y.A., Shen, E., Sherman, A.T.: Scantegrity II: End-to-End Verifiability for Optical Scan Election Systems using Invisible Ink Confirmation Codes. In: USENIX/ACCURATE Electronic Voting Technology (EVT 2008). USENIX Association (2008), see also `http://www.scantegrity.org/elections.php`
8. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a Secure Voting System. In: 2008 IEEE Symposium on Security and Privacy (S&P 2008). pp. 354–368. IEEE Computer Society (2008)
9. Cortier, V., Galindo, D., Glondu, S., Izabachène, M.: Election Verifiability for Helios under Weaker Trust Assumptions. In: Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II. Lecture Notes in Computer Science, vol. 8713, pp. 327–344. Springer (2014)
10. Cortier, V., Gaudry, P., Yang, Q.: A Toolbox for Verifiable Tally-Hiding E-Voting Systems. In: Computer Security - ESORICS 2022 - 27th European Symposium on Research in Computer Security. Lecture Notes in Computer Science, vol. 13555, pp. 631–652. Springer (2022)
11. Damgård, I., Jurik, M., Nielsen, J.B.: A generalization of Paillier's public-key system with applications to electronic voting. Int. J. Inf. Sec. **9**(6), 371–385 (2010)
12. Der Bundeswahlleiter: Wahl zum 20. Deutschen Bundestag am 26. September 2021: Heft 3 Endgültige Ergebnisse nach Wahlkreisen (2021), `https://bundeswahlleiter.de/dam/jcr/cbceef6c-19ec-437b-a894-3611be8ae886/btw21_heft3.pdf`, `https://www.bundeswahlleiter.de/bundestagswahlen/2021/ergebnisse/opendata/csv/`
13. Genssler, G.: Das d'Hondtsche Verfahren und andere Sitzverteilungsverfahren aus mathematischer und verfassungsrechtlicher Sicht. Ph.D. thesis (1984)
14. Haines, T., Pattinson, D., Tiwari, M.: Verifiable Homomorphic Tallying for the Schulze Vote Counting Scheme. In: Verified Software. Theories, Tools, and Experiments - 11th International Conference, VSTTE 2019. Lecture Notes in Computer Science, vol. 12031, pp. 36–53. Springer (2019)
15. Heather, J.: Implementing STV securely in Prêt à Voter. In: IEEE CSF 2007. pp. 157–169 (2007)
16. Hertel, F., Huber, N., Kittelberger, J., Küsters, R., Liedtke, J., Rausch, D.: Extending the Tally-Hiding Ordinos System: Implementations for Borda, Hare-Niemeyer, Condorcet, and Instant-Runoff Voting. In: Proceedings of E-Vote-ID 2021. p. 269–284. University of Tartu Press (2021)
17. Huber, N., Küsters, R., Krips, T., Liedtke, J., Müller, J., Rausch, D., Reisert, P., Vogt, A.: Kryvos: Publicly Tally-Hiding Verifiable E-Voting. In: CCS 2022. pp. 1443–1457. ACM (2022). https://doi.org/10.1145/3548606.3560701, `https://doi.org/10.1145/3548606.3560701`

18. Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant Electronic Elections. In: Proceedings of Workshop on Privacy in the Eletronic Society (WPES 2005). pp. 61–70. ACM Press (2005)
19. Kiayias, A., Zacharias, T., Zhang, B.: DEMOS-2: Scalable E2E Verifiable Elections without Random Oracles. In: CCS 2015. pp. 352–363 (2015)
20. Kiayias, A., Zacharias, T., Zhang, B.: End-to-End Verifiable Elections in the Standard Model. In: EUROCRYPT 2015. pp. 468–498 (2015)
21. Küsters, R., Liedtke, J., Müller, J., Rausch, D., Vogt, A.: Ordinos: A Verifiable Tally-Hiding E-Voting System. In: IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020. pp. 216–235. IEEE (2020)
22. Küsters, R., Müller, J., Scapin, E., Truderung, T.: sElect: A Lightweight Verifiable Remote Voting System. In: IEEE 29th Computer Security Foundations Symposium (CSF 2016). pp. 341–354. IEEE Computer Society (2016). https://doi.org/10.1109/CSF.2016.31
23. Lijphart, A.: Degrees of proportionality of proportional representation formulas. RIVISTA ITALIANA DI SCIENZA POLITICA **13**(2), 295–305 (1983)
24. Lipmaa, H., Toft, T.: Secure Equality and Greater-Than Tests with Sublinear Online Complexity. In: Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013. Lecture Notes in Computer Science, vol. 7966, pp. 645–656. Springer (2013)
25. Ramchen, K., Culnane, C., Pereira, O., Teague, V.: Universally Verifiable MPC and IRV Ballot Counting. In: Financial Cryptography and Data Security - FC 2019, Revised Selected Papers. LNCS, vol. 11598, pp. 301–319. Springer (2019)
26. Ryan, P.Y.A., Rønne, P.B., Iovino, V.: Selene: Voting with transparent verifiability and coercion-mitigation. In: Clark, J., Meiklejohn, S., Ryan, P.Y.A., Wallach, D.S., Brenner, M., Rohloff, K. (eds.) Financial Cryptography and Data Security - FC 2016. Lecture Notes in Computer Science, vol. 9604, pp. 176–192. Springer (2016). https://doi.org/10.1007/978-3-662-53357-4_12, `https://doi.org/10.1007/978-3-662-53357-4\_12`
27. Schoenmakers, B., Veeningen, M.: Universally Verifiable Multiparty Computation from Threshold Homomorphic Cryptosystems. In: Applied Cryptography and Network Security. pp. 3–22. LNCS, Springer (2015)
28. Wabartha, C., Liedtke, J., Huber, N., Rausch, D., Küsters, R.: Implementation of our System. (2023), `https://github.com/JulianLiedtke/ordinos-bundestag`
29. Wen, R., Buckland, R.: Minimum Disclosure Counting for the Alternative Vote. In: E-Voting and Identity, Second International Conference, VoteID 2009, Luxembourg, September 7-8, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5767, pp. 122–140. Springer (2009)