A roadmap to fully homomorphic elections: Stronger security, better verifiability

Kristian Gjøsteen and Martin Strand

Norwegian University of Science and Technology, Trondheim, Norway
{kristian.gjosteen,martin.strand}@ntnu.no

Abstract. After the trials of remote internet voting for local elections in 2011 and parliamentary elections in 2013, a number of local referendums has renewed interest in internet voting in Norway.
The voting scheme used in Norway is not quantum-safe and it has limited voter verifiability. In this case study, we consider how we can use fully homomorphic encryption to construct a quantum-safe voting scheme with better voter verifiability.
While fully homomorphic cryptosystems are not efficient enough for the the system we sketch to be implemented and run today, we expect future improvements in fully homomorphic encryption which may eventually make these techniques practical.

Keywords: fully homomorphic encryption, remote internet voting, quantum-safe

1 Introduction

Norway conducted trials of remote internet voting for the 2011 local elections and the 2013 parliamentary elections. The government discontinued the trials in 2014, but a large number of local referendums in 2016 has caused renewed interest in remote internet voting, especially for less important elections.

There are two issues with the scheme used in 2013 that should be improved. The scheme is not quantum-safe, and voter verifiability is mostly lacking today, due to an auditing protocol that can only be run by accredited organisations. This is a study to see if we can improve on both of these shortcomings concurrently. There are still some primitives lacking before this roadmap can be implemented completely.

While it is unclear if a sufficiently large and reliable quantum computer will ever be built to threaten the security of discrete logarithm-based systems, the mere possibility that the encryption protecting ballot confidentiality may be compromised in 10–30 years from now is a serious problem that needs to be adressed.

Verifiability is difficult in Norway for two reasons. Revoting is used as an anti-coercion tactic, and Norwegian ballots are sufficiently complicated to allow Italian attacks, i.e., marking a ballot with a number of insignificant yet unique changes. Also, the entire ballot is required for the count, so the election can

not be considered as a collection of independent races. Voter verifiability is in general not considered to be important by the Norwegian electorate (polls and other studies generally finds high levels of trust in Norwegian elections [20]), but even if there is no public demand for voter verifiability, better voter verifiability than in the 2013 scheme would still be an improvement.

There are many schemes in the literature that achieve better voter verifiability than the 2013 scheme, but in general, these are not quantum-safe and do not facilitate very complicated ballots. All of the mainstream fully homomorphic schemes are believed to be quantum-safe.

While the 2013 protocol [14] exploited the multiplicative structure of the ElGamal scheme, a fully homomorphic scheme can allow us to use both addition and multiplication. This enables much more flexible computations, which means that we can arrange the decryption and counting process such that it is more voter verifiable.

Alternative approaches There have been earlier attempts at completing election tallies while the ballots are still encrypted, but not at this level of complexity. Salamonsen [23] tried to apply Pailler encryption to Norwegian county elections, possibly the easiest variant, and timed the effort needed to compute ciphertexts and the necessary zero knowledge proofs, clocking in at between 2 and 5 hours of work for the voter. Peeking ahead to Section 6.4, we see that our solution is far more efficient than this.

Benaloh et al. [4] have described how one can use single-operation homomorphic encryption to tally a single transferable vote election. However, we tackle a more intricate problem in this work that cannot be solved with the same techniques. Chilotti et al. [8] have constructed a LWE based voting system in detail, but assume that their bulletin board is honest. We remove that restriction, and also get a scheme that can handle more complex (yet compact) ballots.

Contributions At a theoretical level, we are exploring a possible application of fully homomorphic encryption. The idea of FHE was first proposed in 1978 [22], but was first properly realised with Gentry's 2009 breakthrough [12]. There have been several proposed applications [1, 19], but many of those are purely theoretical due to the tremendous amount of redundant data that would be needed per user.

Next, this is a case study on how FHE could be used to make future Norwegian elections both quantum-safe and more voter verifiable. Our proposed protocol is borderline practical, at least taking into account the number of zero knowledge proofs the existing protocol must check, and it can be further optimised by implementation experts. We provide some experimental data to give a rough estimate of the computation efforts needed. We expect further progress in fully homomorphic encryption, which means that this protocol can eventually become practical in the not-too-distant future.

Organisation The next section provides an introduction to the technical nature of Norwegian elections, followed by an introduction to lattice cryptography and fully homomorphic encryption. In Section 4, we briefly recall the modelling done in previous work. Section 5 partially paves the way for our instantiation of local elections (Section 6) by discussing primitives we are going to need. Finally, we argue for the security of the protocol in Section 7. The formal security modelling and thus proofs have been omitted in this work due to space limitations.

Acknowledgements The authors wish to thank the anonymous reviewers for constructive and useful suggestions.

2 Norwegian elections

The main idea in this paper is to do most of the ballot processing as computations on encrypted data. This means that we need to give an arithmetic circuit for counting. In order for this circuit to make sense, we first need to explain the mechanics of Norwegian elections in some detail.

In all Norwegian elections, each district elects multiple members roughly as follows. Parties nominate lists of candidates for each district. The voter chooses one of these party lists as their ballot. Here we only discuss the details of the local elections. The full version of the paper will also contain a description of parliamentary elections and how to handle these with FHE.

Municipal elections To vote in a municipal election, the voter must first pick a party list. Choosing a given party list gives that party a certain number of *list votes*. The total number of list votes in a district will determine the number of members each party gets.

The voter can then give *person votes* to zero or more candidates on the list. The number of person votes each candidate gets determines which candidates are actually elected as members for that party.

The party may also *prefer* a subset of their candidates. These candidates will then automatically get an additional number of person votes equal to 25 % of the number of ballots submitted for that party.

The voter can also optionally *write in* a certain number of candidates from other party lists. These candidates will then receive person votes. However, writing in a candidate from a different party list will also transfer a list vote from the voter's party of choice to the party that the write-in candidate belongs to.

Consider the example ballot from Figure 1. If the number of members to be elected is 29, each submitted ballot will initially give 29 list votes to the indicated party, in this case the Crypto Party. But on this ballot, the voter has listed four candidates from other lists, which means that the Crypto Party only gets 25 list votes, while the Hacker Party (HP) gets two list votes, and the Analyst Party (AP) and the Eavesdropper Party (EP) gets one list vote each.

When tallying, one first counts the list votes each party gets, and decide how many representatives each party gets using a modified Saint-Laguë's method.

The original Saint-Laguë's method is to create a table with one column for each party, and with each party's number of list votes written in the first row. In the ith row, the number from the first row of the same column divided by $2i - 1$ is written. The k representatives to be elected are then distributed to the parties with the k largest numbers. The modification used in Norway is that the numbers in the first row are divided by 1.4 before distributing candidates, a modification that slightly favours larger parties.

The next step when tallying is to decide which candidates are actually elected. To do this, one counts all person votes given by voters (either to a candidate on the party list, or by writing in a candidate from another party list) and the person votes resulting from party preference. The candidates are then ranked according to the number of person votes received. In the event of a tie, the order of the candidates on the party list is used.

3 Lattices and fully homomorphic encryption

Lattices have long been important in cryptography, both as a tool to attack systems and as basis for new cryptographic systems. Two recent developments have made such lattice-based cryptography even more important, namely the development of *fully homomorphic encryption* (FHE) and the renewed interest in *quantum-safe cryptographic schemes*.

Fully homomorphic encryption is a form of encryption that allows one to do certain computations on encrypted data. While first defined in 1978 [22], the first plausible solution was Gentry's breakthrough 2009 construction [12].

Fully homomorphic encryption allows us to evaluate a function described by a *circuit* on a set of encrypted inputs, resulting in an encryption (of size independent of the number of inputs and the circuit evaluated, called *compact*) of the result we would have gotten if we instead just computed the circuit on (unencrypted) inputs.

Lattice problems such as *(ring) learning with errors* ((R)LWE) are generally considered to be hard to solve, even for a large quantum computer.

Crypto Party
Candidate list for the local elections 2019

1. ☒ **Gaius Julius Caesar**
2. ☐ **Giovan Battista Bellaso**
3. ☒ Al-Khalil ibn Ahmad al-Farahidi
4. ☒ Leone Battista Alberti

⋮

34. ☐ Charles Wheatstone

Candidates from other lists

NAME	PARTY
Alan Turing	HP
Konrad Zuse	HP
Charles Babbage	AP
Eve Mallory	EP

Fig. 1. An example ballot for a local election

Lattice cryptography has seen a tremendous development since Regev [21] found a quantum reduction from the natural lattice problems of finding the shortest vector (SVP) or finding a short basis of independent vectors (SIVP), to LWE.

Several authors have used LWE and RLWE to create fully homomorphic encryption. The main ideas remain the same as in Gentry's original construction. The plaintext is masked with *inner* and *outer* randomness, where the innermost one is denoted as noise. One can then typically perform additions and multiplications, though sometimes a NAND gate must be used. However, for each operation, the noise level grows. When it reaches the same size as the outer randomness, the ciphertext is no longer decryptable. Multiplications are usually expensive in terms of noise, causing the noise to grow quickly, while additions are cheap.

The noise problem can sometimes be solved using a technique called *bootstrapping*, during which a ciphertext is encrypted again (though this encryption could be done with no randomness), and the inner encryption is removed by running the decryption circuit in an encrypted state. One can fine-tune the parameters such that the resulting ciphertext has a lower noise level than the original one. However, the bootstrapping process is computationally expensive, so it is more common to select parameters based on the function one wants to compute, so as to achieve a designated *[multiplicative] depth* (so-called levelled fully homomorphic encryption). Many schemes have also provided solutions for limiting the noise growth, so that one can avoid bootstrapping further.

FHE has reached a level of maturity where it is practical for some applications and security levels [10]. We expect performance to increase still further. The BGV [5] cryptosystem has been implemented by Halevi and Shoup [16], and among others, Microsoft has also worked with implementations [18].

Formally, a FHE scheme consists of algorithms (Gen, Enc, Eval, Dec). The unusual member of the set is Eval, which accepts a special evaluation key evk, a circuit $\mathcal{C}$ and a number of ciphertexts $c_1, \ldots c_n$ such that

$$\mathsf{Dec}(sk, \mathsf{Eval}(evk, \mathcal{C}, c_1, \ldots, c_n)) = \mathcal{C}(\mathsf{Dec}(sk, c_1), \ldots, \mathsf{Dec}(sk, c_n)).$$

We will simplify this notation whenever it is convenient, and often just express the circuit (or function) directly on the ciphertexts, even when we really want them to be applied to the encrypted data.

The presentation of the circuits in this paper is fairly general, but we have made sure to only use features supported by the BGV scheme. We refer to the original publication [5] for the technical details, but quickly introduce some of the high-level features provided by the scheme.

Plaintext slots Following an idea of Smart and Vercauteren [25], one can pack several plaintexts into a single ciphertext and do SIMD operations (single-instruction multiple-data) on the vector of plaintexts. The advantage is that one saves space, and that one can perform operations on tuples of data in the time it would to do it on a single value. All slots must have the same

capacity. The plaintext space of the BGV scheme can thus be set to any space $\mathbb{F}_{q^\ell}^n$ for some integers q, ℓ and n, where n denotes the number of slots.

Noise management The authors use a system of modulus reduction for each multiplication, such that the noise increases slower than it would otherwise do. Hence, one can have smaller ciphertexts. The number of times one can do the modulus reduction decides the maximal multiplicative depth.

Key switching In addition to reducing the modulus, one can also efficiently transform ciphertexts from one key to another.

The term *fully homomorphic encryption* has two meanings, either that the scheme in question can process any circuits of any depth (typically by using bootstrapping) or that it can evaluate two operations, in contrast to group homomorphic schemes like ElGamal. In principle, we only need a levelled homomorphic scheme, but will use the word fully to denote the concept.

4 Modelling and security requirements

We model our system with the same players that already existed in the Norwegian e-voting project, namely the voter V with her computer and mobile phone, a ballot box B, a receipt generator R, a decryption service D and an auditor A. We quickly explain the existing motivation before proceeding. For more details, see Gjøsteen [14].

Gjøsteen acknowledged that the user may not be in control of her own equipment, for instance due to malware. One should therefore distinguish the voter's intention and what the computer actually does. When the ballot box receives an encrypted ballot from the voter's computer, it transforms and partially decrypts the ballot, and forwards it to the receipt generator. Then the transformed ballot is completely decrypted, and the correct receipt code is sent by SMS to the voter's mobile phone.

Both the ballot box and the receipt generator give the auditor information about everything they have seen, so that he can compare and make sure no one of them is ignoring information seen by the other. Any information dropped by the ballot box should ideally be detected by the voter, because of a missing receipt. (The soundness of this protocol is based on an assumption that the phone is independent of the device used to vote. While this may have been an acceptable assumption when the system was first introduced, it is less so today. Finding another solution may be necessary, but is outside the scope of this paper.)

Next consider what happens when the election closes. Then the ballot box should provide ciphertexts to the decryption service, which outputs the public result of the election. The auditor verifies that the decryption service got the right ciphertexts from the ballot box, and that the output was correct.

The security requirements can informally be summarised in the following list.

D-privacy The decryption service should not be able to correlate its input to voter identities

B-privacy The ballot box should not learn anything from the ciphertexts

R-**privacy** The receipt generator should not be able to correlate return codes to what the voter chose

A-**privacy** The auditor should not learn anything about how anyone voted

B-**integrity** The ballot box must not be able to create a convincing encrypted ballot such that its decryption is inconsistent with the related information that is sent to the receipt generator

D-**integrity** The decryption service must not be able to alter the election outcome

We conclude this section with a brief overview of some limitations that any Norwegian voting system must deal with.

Privacy is important in Norwegian elections. The ballot should obviously be confidential, but even the list of who voted is considered confidential in Norway. In particular, this means that any voter verifiable scheme that reveals the identities of the voters is unacceptable in Norway.

A second constraint is coercion resistance. It seems like the main defence against coercion must be revoting in Norway, and the revoting could possibly be paper revoting. Paper voting cannot involve any secrets or other material from previous electronic voting, and a paper ballot should also supercede any subsequent electronic ballot submission.

Related to coercion resistance, Italian attacks are easy in Norway, since adding a random set of marks to a ballot will most likely make it unique and have negligible electoral effect. This means that any public verifiability must avoid publishing complete ballots. Today, the election authorities publish lists of vote sums per party and person.

Finally, it seems like electronic voting must coexist with paper voting for the forseeable future. This means that the electronic count must somehow combine with the paper count before the final result is declared.

5 Primitives

We assume the existence of several primitives in this work. Some of them have not been described in the literature yet, and should be considered open, but feasible problems.

The main primitive we need is an efficient zero knowledge proof or argument for correct decryption. These are well known for schemes such as ElGamal [7], but for FHE, they only become efficient when applied to many ciphertexts concurrently [2, 3]. However, much of the work can be done ahead of time, and the protocol also supports distributed decryption, which will essentially guarantee the security of the complete scheme. Note that one possible instantiation of the following protocol would only require the verifiable decryption of a single ciphertext. Providing an efficient zero knowledge proof for that case can still be considered an open problem.

Next, we also need a number of subroutines. Equality checking will be used throughout the whole routine, and has been provided by Kim et al. [17]. We will

denote it as a function

$$\mathsf{Eq}(a,b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise.} \end{cases}$$

The multiplicative depth for equality checking in the BGV scheme is given as $\lceil \log(p-1) \rceil + \lceil \log \ell \rceil$ where p is the characteristic of the field and ℓ is the order of the extension. Although this will be a fairly high number, most equality checks will run in parallell, which will help keep the noise under control.

If we want to implement the whole tallying as operations on encrypted data (which we do not propose to do) we would also need sorting and division by rational numbers. Both of these primitives exist. Emmadi et al. [11] analysed a number of algorithms and concluded that Odd-Even Merge sort would work best for the Smart-Vercauteren scheme [24]. It is reasonable that some of their results will apply to the BGV scheme as well. Chung and Kim proposed that one can use a continued fraction representation of rational numbers to reduce the storage requirement for rational numbers with a given precision, and also described how to perform divisions [9]. Çetin et al. [6] have demonstrated that it is possible to compute fractions and even square roots by applying numerical methods to the encrypted data.

6 Instantiation

Recall the BGV scheme introduced in Section 3. Let m be the number of parties taking part in the election and n be the total number of candidates from all the parties. We use the BGV cryptosystem with multiple plaintext slots, and the goal is to get an encryption of the following tuple for each ballot posted by a voter.

$$(p_1, \ldots, p_m, p'_1, \ldots, p'_m, p''_1, \ldots, p''_m, c_1, \ldots, c_n) \tag{1}$$

The tuple requires $3m+n$ slots. Although they will hold data of different length, the BGV system requires them to be the same size.

Let v be the maximal number of voters from the voting district, and let k be the number of candidates. The voter may list up to $k/4$ candidates from other lists, which also places an upper bound on how many list votes that may be transferred from the chosen party to another. The first m slots will hold the number of times a list is selected, the next section holds the number of list votes given away, and the final section of m slots holds the number of list votes received. Finally, the last n items will hold person votes. The upper bound for the plaintext space is then the maximal number of list votes that can be transferred, $vk/4$, and the characteristic of the slots should in principle not be chosen smaller, although it is very unlikely that one will ever reach this bound.

Remark 1. One can possibly save some storage overhead by using several independent ciphertexts instead of larger ciphertexts with slots, or some combination of slots and separate ciphertexts, so that no slots have higher capacity than needed.

To vote, the voter must encrypt her ballot with a symmetric scheme, and attach the key encrypted under the FHE scheme [13]. Assume that the voter ciphertext encodes the vector

$$b = (p, s_1, \ldots, s_{n_p}, e_1, \ldots, e_{n'}), \tag{2}$$

where p is the index of the chosen party list, s_i is a bit indicating whether candidate i on the list receives a person vote and $e_1, \ldots, e_{n'}$ are the indices of the representatives from different lists that have been written in on this ballot.

Remark 2. It is natural to ask why the voter simply cannot encrypt a vector of the form (1). The reason is that we want to be able to validate the ballot efficiently, something which would add considerable extra work if we had to check ranges for vectors like the one above. Note that we need to check that $s_1, \ldots, s_{n_p}$ are actual bits. One way to do this is to compute the product $s(\mathsf{Enc}(1) - s)$ for each slot, and verify that it decrypts to 0. (There is an obvious weakness to this, which is that if it does not decrypt to 0, then information will leak. We can avoid this problem by normalising using the algorithm of Kim et al. [17], which depends on the Frobenius automorphism $x \mapsto x^q$. This particular exponentiation can be done for free in the BGV system. Of course, we may not care about the privacy of malformed ballots.)

The ballot box should perform a bootstrapping after transforming the ciphertext to the FHE scheme. This guarantees that the voter cannot introduce too much noise, which in turn can make the end result impossible to decrypt.

We now explain how to transform a ballot of the form in (2) to the form of (1). Recall the function Eq that returns 1 whenever the two input values are equal. Define $\mathsf{In}(a, S) = \sum_{s \in S} \mathsf{Eq}(a, s)$, which will return 1 if and only if a is a member of set the S.

Let $\{P_i\}$ denote all parties taking part in the election, and let p_i'' be the number of list votes transferred to party P_i. By abuse of notation, let P_i also denote the set of indexes for the candidates on the party list of party i. The number p_i'' of list votes transferred is then easily computed as

$$p_i'' \leftarrow (\mathsf{In}(e_1, P_i) + \mathsf{In}(e_2, P_i) + \cdots + \mathsf{In}(e_{n'}, P_i)$$

Note that each equality and membership check requires some multiplications, but since they can all be done in parallell, the overall noise remains manageable.

To compute the number p_i' of list votes given away, we need to identify the right party, and then add the sum of all p_i'' for that ballot,

$$p_i' \leftarrow \mathsf{Eq}(i, p) \cdot \sum_{j=1}^{m} p_j''.$$

The values p_i are easily decided with $\mathsf{Eq}(i, p)$.

Finally, compute person votes to candidates from other lists. Let P_j be the party that the candidate c_j belongs to, and recall that p was the party selected by the voter.

$$c_j \leftarrow (1 - \mathsf{Eq}(p, P_j))\,(\mathsf{Eq}(j, e_1) + \mathsf{Eq}(j, e_2) + \cdots + \mathsf{Eq}(j, e_{n'}))$$

The first factor ensures that the candidates really are from a different list, in order to avoid a situation where a candidate could get a person vote in two different ways.

Remark 3. Note that we do not check for this when we are counting person votes in and out. The reason is that it does not change the balance, and it should thus not create any problems for the validity of the ballot or the count in general. We aim at a forgiving system such that only destructive changes could cause a ballot to be discarded.

Use a similar technique as above to add the person votes to the candidates from the list chosen by the voter. One can also convert the whole vector into a ciphertext, and use SIMD techniques [25] to add them as one. Note, however, that one should still do basic range checking on the values as described in the remark above.

Finally, we want to verify that a ballot is valid, by computing the polynomial

$$\prod_{i \leq j} (e_i - e_j),$$

which will be zero if and only if one candidate is listed more than once. This polynomial requires many multiplications, so we propose that the end result is decrypted publicly, so that the selection bit can be public. One can normalise any non-zero value to 1.

6.1 Selecting votes to be counted

Since we allow voters to vote multiple times and even override all electronic votes by voting on paper, we need a way to identify the ballots that should be included in the final tally, while providing both voter verifiability and coercion resistance. While in theory it is possible to do everything with FHE, it would probably be more expensive than anything else in this paper, so we have opted for a more classical solution, where the ballot box selects the ballots to be counted and uses a combination of auditing and FHE to prove that its selection is correct.

1. The ballot box stores a secret record (v_i, c_i, s_i) for each ballot, where v_i is the voter's identity, c_i is the encrypted vote and s_i is a sequence number, or equivalently, a timestamp.
2. When voting closes, for each voter v_i that also submitted a paper ballot, the ballot box adds a triple (v_i, c_i, s_i) where c_i encodes a blank vote and s_i is a sequence number greater than the highest inserted in the ballot box.
3. The ballot box shuffles all identities and sends the list to the auditor.
4. The ballot box publishes a list $\{(c_i, \tilde{s}_i, \tilde{v'}_i)\}$, where the tilde indicates that the value has been encrypted using a FHE scheme and the identities $\{v_i\}$ have been replaced with pseudonyms $\{v'_i\}$. The list is ordered by identity and sequence number. The end points may still leak secret information, so the list should be treated as something circular. Concretely, we select a random item on the list, and move the records following that item to the front of the list instead.

5. The ballot box sends the randomness used to generate the new ciphertexts to the auditor for verification.
6. Define a function f by

$$f(v_i, v_j, s_i, s_j) = \begin{cases} 1 & \text{if } v_i \neq v_j \text{ or } s_i < s_j \text{; and} \\ 0 & \text{otherwise.} \end{cases}$$

   The ballot box computes $\tilde{u}_i = \mathsf{Eval}(evk, f, \tilde{v}_i, \tilde{v}_{i+1}, \tilde{s}_i, \tilde{s}_{i+1})$ for all i, and counting modulo the number of ballots, such that the last item on the list is compared to the first.
   The function f can be implemented by combining the equality checker described above with an inequality function [11].
7. The ballot box multiplies all $\tilde{u}_i$. This value should later be decrypted. Any auditing parties should accept that the list is correctly ordered if the product decrypts to 1.
8. Next define a function g such that $g(v_i, v_j)$ is 1 if and only if $v_i \neq v_j$ and 0 otherwise. The ballot box computes selection values $\tilde{z}_i = \mathsf{Eval}(evk, g, \tilde{v}_i, \tilde{v}_{i+1})$, cycling to the top of the list as above. The list of ciphertexts that should be counted is $\{c'_i = \tilde{z}_i c_i\}$.

6.2 Tally

The final tally now becomes trivial. The ballot box simply adds all the encrypted vectors $\{c'_i\}$. We suggest to pass these sums on to the decryption service for decryption and deciding which candidates are elected based on cleartext vote counts (number of list votes and number of person votes).

Of course, it is possible to do Saint-Laguë's method with encrypted data. While we do not recommend this, since the paper ballots must also be counted, we note that it is possible, and simply an engineering problem to analyse the complexity and then adjust the parameters to allow for the circuit depth. We note, however, that it may result in even bigger ciphertexts. Possibly the most interesting challenge would be to handle the precision of the rational numbers that necessarily would occur, and then handling the sorting.

6.3 Receipts

Sending a receipt to the voter becomes easy when using an FHE scheme. The ballot box homomorphically applies a voter-specific one-way transformation on the ballot, followed by a key-switch, a feature offered by the BGV cryptosystem. We do not propose such a function here, other than to state that a light-weight keyed hash function would do the job nicely. While the main purpose is to facilitate a greater multiplicative depth, the transformation key can only be built using both of the secret keys, but it leaks neither. Hence, we can give the transformation key exclusively to the ballot box (and the auditor), while the receipt generator is the only party to know the decryption key for the transformed ciphertexts.

After the receipt generator decrypts, it sends an SMS with a value derived from the transformed ballot, with which the voter can verify that the ballot has been received correctly. The auditor should be sent a copy of all ciphertexts received by the receipt code generator.

Due to the problem of phones being closely linked to the user's other equipment, the receipt system should as a whole be put under closer scrutiny.

6.4 Parameter selection

Using the above algorithms, we can estimate the parameters needed for the protocol. For a conservative estimate, we can look at the numbers for the largest municipality, Oslo. There are about 500,000 eligible voters, and the last local election saw 17 different party lists with a total of 659 candidates. The city council consists of 59 members. This means that the voter can list at most 15 names from other parties on her ballot, so the greatest number we need to handle is about 7,500,000 $\approx 2^{23}$. Then equality checks will need a depth 23 circuit. We can now compute how much depth we will need after converting from the symmetric ciphertext.

- p_i'' can be computed with many equality checks in parallell, but no other multiplications, hence depth 23. The same holds for p_i.
- p_i' is one equality check multiplied with a sum of p_i'', so we need 24 multiplications. The candidate slots are also the result of a multiplication of two equality checks.
- Computing the validity check requires $\binom{15}{2}$ multiplications. However, they can be arranged in a tree of depth 7.
- Each value $\tilde{u}_i$ requires an equality check and an inequality check. After that, all such ciphertexts must be multiplied, which can be done with depth of approximately 20.
- The selection bit $\tilde{z}_i$ takes a single comparison, and is multiplied to the rest of the ballot, adding one level to some of the previous results.

In addition comes the depth required to send the receipt, but that is dependent of the function employed to generate the return codes. Note that those computations will be in parallell to those above.

Finally, we can conclude that no part of the computation requires a depth greater than 50.

The number of slots needed in the Oslo case is $3 \cdot 17 + 659 = 710$.

We ran the bundled general test program of `HElib` [15, 16] with the above parameters on a server running Ubuntu 14.04 on Intel Xeon 2.67 GHz processors with a total of 24 cores and 256 GB of memory. The program ran the key generation on a single core, and used a maximum of 8 cores for some sample ciphertext operations. The maximum memory usage was in the order of 20 GB. The complete process took 4:52 minutes, with key generation taking about half of that time. While this order of magnitude is unreasonable for a single voter, it may be feasible for an election system, as long as the feedback to the voter is sufficiently quick. Implementing the above algorithms efficiently is an open problem.

7 Security

A formal security proof along the lines of Gjøsteen [14] is too long for this paper, and we defer such a proof for a full version of this paper. However, we briefly discuss the general security properties, and then discuss coercion resistance in some detail.

The system is not designed to be secure if two or more of the ballot box, receipt generator, decryptor or auditor are corrupt. However, when at most one of them is corrupt, encryption and careful use of key-switching ensures that the receipt generator and the decryptor cannot decrypt public ciphertexts containing sensitive information (such as ballots and voter identities), ensuring that we have both R-privacy and D-privacy. (For instance, the published ciphertexts only contain ballots and pseudonyms, so while a corrupt decryption service could decrypt them, it would learn nothing about which voters these ballots came from.) The encryption itself and the general features of the protocol ensure B-privacy and A-privacy.

Since the computation on encrypted data can be redone by any interested party using published information, the scheme is trivially almost end-to-end verifiable, and also has B-integrity. This follows from the correctness of the selection and counting circuit we have designed, and the correctness of the FHE scheme in use. An interested voter can verify that the ciphertext she submitted is listed in the public record and then redo the computation of the counting circuit, recreating the ciphertexts containing the results. The zero knowledge proofs will then ensure that the published election results are consistent with the verified encrypted results. This gives us D-integrity.

The gap in verifiability lies in the selection of votes to be counted, where a corrupt ballot box may insert fake votes that a corrupt auditor may choose to ignore. (An honest auditor should either notice that electronic votes lack a valid digital signature, or that fake paper votes have been inserted.) Even with this gap, however, our proposal is a significant improvement on previous schemes used in Norway.

- If the ballot box alters a ballot before forwarding it to the receipt generator, then the voter should get an incorrect receipt. If the ballot box alters a ballot after the receipt generator has seen it, the auditor will notice.
- The receipt generator cannot alone compromise the integrity of the election. Privacy follows from careful use of key-switching and using an appropriate function to generate receipts with a per-voter key.
- The decryption service must prove correctness of decryption, and the integrity of the result follows from this proof.
- The cryptosystem ensures that an auditor cannot read any individual votes.

We defend against coercion by letting a voter revote electronically any number of times, and decreeing that a paper ballot will override any earlier or later electronic votes. This is within the requirement of the Norwegian Election Act, which states that "[t]he purpose of this Act is to establish such conditions that

citizens shall be able to elect their representatives to the Storting, county councils and municipal councils by means of a secret ballot in free and direct elections." [26] Let us now consider how a coercer could be able to succeed.

It is clear that any coercer cooperating with a corrupt ballot box or auditor will be able to defeat revoting as an anti-coercion strategy. We therefore assume a coercer that sits next to the voter as she casts her ballot, and assume that the coercer is also able to record the precise ciphertext, and himself transform it into the FHE ciphertext c_i that will appear in the public records.

If the voter now revotes electronically, and the coercer afterwards returns and forces her to vote under surveillance again, then the public list of ciphertexts will reveal that the voter revoted. However, any paper vote will be sorted after the last electronic vote, so it cannot be discovered by the adversary. Also note that since the identities are permuted and encrypted, he cannot guarantee that a paper vote will be sandwiched between an electronic vote and the first vote of someone of whom he knows the identity, making the paper ballot truly anonymous.

References

1. Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A. Reuter, and Martin Strand. A guide to fully homomorphic encryption. Cryptology ePrint Archive, Report 2015/1192, 2015. `http://eprint.iacr.org/`.
2. Carsten Baum, Ivan Damgård, Tomas Toft, and Rasmus Winther Zakarias. Better preprocessing for secure multiparty computation. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *Applied Cryptography and Network Security - 14th International Conference, ACNS 2016*, volume 9696 of *Lecture Notes in Computer Science*, pages 327–345. Springer, 2016.
3. Carsten Baum, Ivan Damgård, Sabine Oechsner, and Chris Peikert. Efficient commitments and zero-knowledge protocols from ring-sis with applications to lattice-based threshold cryptosystems. Cryptology ePrint Archive, Report 2016/997, 2016. `http://eprint.iacr.org/2016/997`.
4. Josh Benaloh, Tal Moran, Lee Naish, Kim Ramchen, and Vanessa Teague. Shufflesum: coercion-resistant verifiable tallying for STV voting. *IEEE Trans. Information Forensics and Security*, 4(4):685–698, 2009.
5. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:111, 2011.
6. Gizem S. Cetin, Yarkin Doroz, Berk Sunar, and William J. Martin. Arithmetic using word-wise homomorphic encryption. Cryptology ePrint Archive, Report 2015/1195, 2015. `http://eprint.iacr.org/2015/1195`.
7. David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology - CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 1992.
8. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. A homomorphic LWE based e-voting scheme. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, volume 9606 of *Lecture Notes in Computer Science*, pages 245–265. Springer, 2016.

9. HeeWon Chung and Myungsun Kim. Encoding rational numbers for FHE-based applications. Cryptology ePrint Archive, Report 2016/344, 2016. `http://eprint.iacr.org/`.
10. Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. Technical report, Microsoft Research, February 2016.
11. Nitesh Emmadi, Praveen Gauravaram, Harika Narumanchi, and Habeeb Syed. Updates on sorting of fully homomorphic encrypted data. Cryptology ePrint Archive, Report 2015/995, 2015. `http://eprint.iacr.org/`.
12. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.
13. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.
14. Kristian Gjøsteen. The Norwegian internet voting protocol. Cryptology ePrint Archive, Report 2013/473, 2013. `http://eprint.iacr.org/`.
15. Shai Halevi and Victor Shoup. Algorithms in HElib. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014*, volume 8616 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2014.
16. Shai Halevi and Victor Shoup. Bootstrapping for HElib. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 641–670. Springer, 2015.
17. M. Kim, H. T. Lee, S. Ling, and H. Wang. On the efficiency of FHE-based private queries. *IEEE Transactions on Dependable and Secure Computing*, PP(99), 2016.
18. Kristin Lauter. Practical applications of homomorphic encryption, 2015.
19. Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In Christian Cachin and Thomas Ristenpart, editors, *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW*, pages 113–124. ACM, 2011.
20. OSCE Office for Democratic Institutions and Human Rights. Norway, Parliamentary Elections 9 September 2013, Final Report. Technical report, dec 2013.
21. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 84–93. ACM, 2005.
22. Ronald Rivest, Leonard Adleman, and Michael Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.
23. Kristine Salamonsen. A security analysis of the helios voting protocol and application to the norwegian county election, 2014.
24. Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography – PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.
25. Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptography*, 71(1):57–81, 2014.
26. Lov om valg til stortinget, fylkesting og kommunestyrer (valgloven). `http://lovdata.no`, sep 2002. Translation at `https://www.regjeringen.no/globalassets/upload/KRD/Kampanjer/valgportal/Regelverk/Representation_of_the_People_Act170609.pdf`.