Verifiable Homomorphic Tallying for the Schulze Vote Counting Scheme

Thomas Haines[1], Dirk Pattinson[2], and Mukesh Tiwari[2]

[1] Polyas, Denmark
[2] Research School of Computer Science, ANU, Canberra

Abstract. The encryption of ballots is crucial to maintain integrity and anonymity of electronic voting schemes. It enables, amongst other things, each voter to verify that their encrypted ballot has been recorded as cast, by checking their ballot against a bulletin board.
We present a verifiable homomorphic tallying scheme for the Schulze method that allows to verify the correctness of the count on the basis of encrypted ballots that only reveals the final tally. We achieve verifiability by using zero knowledge proofs for ballot validity and honest decryption of the final tally. Or formalisation takes places inside the Coq theorem prover and is based on an axiomatisation of cryptogtaphic primitives, and our main result is the correctness of homomorphic tallying. We then instantiate these primitives using an external library and show the feasibility of our approach by means of case studies.

1 Introduction

- general need for crypt in vote counting
- gernal need for verifiablility in vote counting: certificates!

One of the key challenges faced by both paper based and electronic elections is that results must substantiated with verifiable evidence of their correctness while retaining the secrecy of the individual ballot [**?**]. Technically, the notion of "verifiable evidence" is captured by the term *end-to-end (E2E) verifiability*, that is

- every voter can verify that their ballot was cast as intended
- every voter can verify that their ballot was collected as cast
- everyone can verify final result on the basis of the collected ballots.

While end-to-end verifiability addresses the basic assumption that no entity (software, hardware and participants) are inherently trustworthy, ballot secrecy addresses the coercion problem:

- no voter should be able to reveal their ballot to any other party

so that a possible coercer has no way of verifying whether or not a voter in fact followed their instructions.

[Thomas: is there a standard reference for E2E verifiability?]

The above desiderata can be realised using encrypted ballots, where encrypted ballots (that the voters themselves cannot decrypt) are published on a bulletin board, and the votes are then processed, and the correctness of the final tally is substantiated, using homomorphic encryption [**?**] and verifiable shuffling [1]. (Separate techniques exist to prevent ballot box stuffing and to guarantee cast-as-intended.)

[Thomas: do we have / need references for ballot box stuffing or cast-as-intended?]

This paper addresses the problem of verifiable homomorphic tallying for a preferential voting scheme, the Schulze voting scheme. We show how the Schulze Method can be implemented in a theorem prover to guarantee both provably correct and verifiable counting on the basis of encrypted ballots, relative to an axiomatisation of the cryptographic primitives. We then obtain, via program extraction, a provably correct implementation of the vote counting, that we turn into executable code by providing implementations of the primitives based on a standard cryptographic library. We conclude by presenting experimental results, and discuss trust the trust base, security and privacy as well as the applicability of our work to real-world scenarios.

Leftover Text?

In general, every election has purpose to elect candidates based on cast ballots while maintaining the integrity of election i.e. tallied correctly, maintained individual ballot privacy, and coercions free. Many of these properties we get for free in paper ballot election, but for electronic voting, not only our protocol should have these properties, but at the same time they should be evident. The inherent nature of tension (conflict) between privacy, keeping the votes private and anonymous (unidentifiable), and universal verifiablity, anyone can verify tallying is correct (consistent) according to election rules, makes electronic voting schemes very challenging. If we publish the ballots in plain text it certainly offers universal verifiability, but at the same time it opens the possibility of coercion [2], and if we don't publish then it certainly offers the more privacy, but hampers universal verifiablity.

The Schulze Method. The Schulze Method [5] is a preferential, single-winner vote counting scheme that is gaining popularity due to its relative simplicity while retaining near optimal fairness [4]. A *ballot* is a rank-ordered list of candidates where different candidates may be given the same rank. The protocol proceeds in two steps, and first computes the *margin matrix* m where, for candidates x and y,

$m(x, y) =$ ballots that rank x higher than $y-$ballots that rank y higher than x.

We note that $m(x,y) = -m(y,x)$, i.e. the margin matrix is symmetric. In a second step, a *generalised margin* g is computed as the strongest path between two candidates

$$g(x,y) = \max\{\mathsf{str}(p) \mid p \text{ path from } x \text{ to } y\}$$

where a path from x to y is simply a sequence $x = x_0, \ldots, x_n = y$ of candidates, and the strength

$$\mathsf{str}(x_0, \ldots, x_n) = \min\{m(x_i, x_{i+1}) \mid i < n\}$$

is the lowest margin encountered on a path.

A candidate w is a *winner* if $g(w,x) \geq g(x,w)$ for all other candidates x. Informally, one may think of the generalised margin $g(x,y)$ as transitive accumulated support for x over y, so that x beats y if $g(x,y) \geq g(y,x)$ and a winner is a candidate that cannot be beaten by anyone. Note that winners may not be uniquely determined (e.g. in the case where no ballots have been cast).

In previous work [3] we have demonstrated how to achieve verifiability of counting plaintext ballots by producing a verifiable *certificate* of the count. The certificate has two parts: The first part witnesses the computation of the margin function where each line of the certificate amounts to updating the margin function by a single ballot. The second part witnesses the determination of winners based on the margin function. In the first phase, i.e. the computation of the margin function, we perform the following operations for every ballot:

1. if the ballot is informal it will be discarded
2. if the ballot is formal, the margin function will be updated

The certificate then contains one line for each ballot and thus allows to independently verify the computation of the margin function. Based on the final margin function, the second part of the certificate presents verifiable evidence for the computation of winners. Specifically, if a candidate w is a winner, it includes:

1. an integer k and a path of strength k from w to any other candidate
2. evidence, in the form of a co-closed set, of the fact that there cannot be a path of strength $> k$ from any other candidate to w.

Crucially, the evidence of w winning the election *only* depends on the margin matrix. We refer to [3] for details of the second part of the certificate as this will remain unchanged in the work we are reporting here.

Related Work.

2 Verifiable Homomorphic Tallying

```
Bullet points of content for reference and later deletion

– general protocol, comparision with plaintext counting
```

- `two-phase structure: margin matrix, then winners with winners as before: encryption commutes with margin computation`
- `homomorphic computation of margin matrix, ballot representation`
- `computation of winners (as for plaintext)`

The realisation of verifiable of homomorphic tallying that we are about to describe follows the same two phases as the protocol: for each ballot, we decide whether it is formal, and if so, homomorphically update the margin function. We do this for every ballot, and record this in the certificate. We then record the decryption of the margin function in the certificate, and then publish the winner, together with evidence based on the margin matrix, as in the case of counting plaintext ballots (where evidence for winning also only depends on the margin matrix). We now describe the two phases in detail.

Format of Ballots. In preferential voting schemes, ballots are rank-ordered lists of candidates. For the Schulze Method, we require that all candidates are ranked, and two candidates may be given the same rank. That is, a ballot is most naturally represented as a function $b : C \to \mathbb{N}$ that assigns a numerical rank to each candidate, and the computation of the margin amounts to computing the sum

$$m(x,y) = \sum_{b \in B} \begin{cases} +1 & b(x) > b(y) \\ 0 & b(x) = b(y) \\ -1 & b(x) < b(y) \end{cases}$$

where B is the multi-set of ballots, and each $b \in B$ is a ranking function $b : C \to \mathbb{N}$ over a (finite) set C of candidates.

We note that this representation of ballots is not well suited for homomorphic computation of the margin matrix as practically feasible homomorphic encryption schemes do not support comparison operators and case distinctions as used in the formula above.

We instead represent ballots as matrices $b(x,y)$ where $b(x,y) = +1$ if x is preferred over y, $b(x,y) = -1$ if y is preferred over x and $b(x,y) = 0$ if x and y are equally preferred.

While the advantage of the first representation is that each ranking function is necessarily a valid ranking, the advantage of the matrix representation is that the computation of the margin matrix is simple, that is

$$m(c,d) = \sum_{b \in B} b(x,y)$$

where B is the multi-set of ballots (in matrix form), and can moreover be transferred to the encrypted setting in a straight forward way: if ballots are matrices $e(x,y)$ where $e(x,y)$ is the encryption of an integer, then

$$\mathrm{em} = \bigoplus_{\mathrm{eb} \in \mathrm{EM}} \mathrm{eb}(x,y)$$

where $\oplus$ denotes homomorphic addition, eb is an encrypted ballot in matrix form (i.e. decrypting $\mathrm{eb}(x,y)$ indicates whether x is preferred over y), and EM

is the multi-set of encrypted ballots. The disadvantage is that we need to verify that a matrix ballot is indeed valid, that is

- that the decryption of $\mathrm{eb}(x, y)$ is indeed one of $1, 0$ or -1
- that eb indeed corresponds to a ranking function.

Indeed, to achieve verifiability, we not only need *verify* that a ballot is valid, we also need to *evidence* its validity (or otherwise) in the certificate.

Validity of Ballots. By a plaintext (matrix) ballot we simply mean a function $b : C \times C \to \mathbb{Z}$, where C is the (finite) set of candidates. A plaintext ballot $b(x, y)$ is *valid* if it is induced by a ranking function, i.e. there exists a function $r : C \to \mathbb{N}$ such that $b(x, y) = 1$ if $f(x) < f(y)$, $b(x, y) = 0$ if $f(x) = f(y)$ and $b(x, y) = -1$ if $f(x) > f(y)$. A *ciphertext (matrix) ballot* is a function $\mathrm{eb} : C \times C \to \mathbb{CT}$ (where $\mathbb{CT}$ is a chosen set of ciphertexts), and it is valid if the pointwise encryption, i.e. the matrix ballot $b(x, y) = \mathsf{dec}(\mathsf{enc}(x, y))$ is valid (we use dec to denote decryption).

For a plaintext ballot, it is easy to decide whether it is valid (and should be counted) or not (and should be discarded). We use shuffles (ballot permutations) to evidence the validity of encrypted ballots. One observes that a matrix ballot is valid if and only if it is valid after permuting both rows and columns with the same permutation. That is, $b(x, y)$ is valid if and only if $b'(x, y)$ is valid, where

$$b'(x, y) = b(\pi(x), \pi(y))$$

and $\pi : C \to C$ is a permutation of candidates. (Indeed, if f is a ranking function for b, then $f \circ \pi$ is a ranking function for b'). As a consequence, we can evidence the validity of a ciphertext ballot eb by

- publishing a shuffled version eb' of eb, that is shuffled by a secret permutation, together with evidence that eb' is indeed a shuffle of eb
- publishing the decryption b' of eb' together with evidence that b' is indeed the decryption of eb'.

We use zero-knowledge proofs in the style of [1] to evidence the correctness of the shuffle, and zero-knowledge proofs of honest decryption [reference missing] to evidence correctness of decryption. This achieves ballot secrecy owing to the fact that the permutation is never revealed.

In summary, the homomorphic computation of the margin matrix starts with an encryption of the zero margin em, and for each ciphertext ballot eb

1. publishes a shuffle of eb together with a ZKP of correctness
2. publishes the decryption of a shuffle, together with a ZKP of correctness
3. publishes the updated margin function, if the decrypted ballot was valid, and
4. publishes the unchanged margin function, if the decrypted ballot is not valid.

Witnessing of Winners. Once all ballots are counted, the computed margin is decrypted, and winners (together with evidence of winning) are being computed

as for plaintext counting. We discuss this this part only briefly to be self contained as it is as for plaintext counting [3]. For each of the winners w and each candidate c we publish

- a natural number $k(w, x)$
- a path $w = x_0, \ldots, x_n = x$ of strength k
- a set $C(w, x)$ of pairs of candidates that is k-coclosed and contains (x, w)

where a set S is k-coclosed if for all $(x, z) \in C$ we have that $m(x, z) < k$ and either $m(x, y) < k$ or $(y, z) \in S$ for all candidates y. Informally, the first requirement ensures that there is no direct path (of length one) between a pair $(x, z) \in S$,and the second requirement ensures that for an element $(x, z) \in S$, there cannot be a path that connects x to an intermediate node y and then (transitively) to z that is of strength $\geq k$. We refer to *op.cit.* for the (formal) proofs of the fact that existence of co-closed sets witnesses the winning conditions.

3 Realisation in a Theorem Prover

- Axiomatisation of the cryptographic primitives
- correctness proof modulo correct crypto
- describe certificates and how to check them
- show that "certificates certify"

4 Extraction and Experiments

- extension of Unicrypt with homomorphic addition
- coupling between library and extracted code
- need for light weight wrappers
- experimental results: how long, how much memory, certicicate size

5 Analysis

- trust base: library, wrapper, bindings, extraction into unverified BigInts
- security: probabilistic aspects ignored, attacker model
- applicability to real world scenarios

References

1. Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *Proc. EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 263–280. Springer, 2012.
2. 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.

3. Dirk Pattinson and Mukesh Tiwari. Schulze voting as evidence carrying computation. In Mauricio Ayala-Rincón and César A. Muñoz, editors, *Proc. ITP 2017*, volume 10499 of *Lecture Notes in Computer Science*, pages 410–426. Springer, 2017.
4. Ronald L. Rivest and Emily Shen. An optimal single-winner preferential voting system based on game theory. In Vincent Conitzer and Jrg Rothe, editors, *Proc. COMSOC 2010*. Duesseldorf University Press, 2010.
5. Markus Schulze. A new monotonic, clone-independent, reversal symmetric, and Condorcet-consistent single-winner election method. *Social Choice and Welfare*, 36(2):267–303, 2011.
6. Xun Yi, Russell Paulet, and Elisa Bertino. *Homomorphic Encryption and Applications*. Briefs in Computer Science. Springer, 2014.

Notes and Leftovers Start Here

A Format of Ballots

Each ballot, B, is a $n\times$n matrix where n is the number of candidates participating in election. Each voter marks the entry (i,j) in ballot B as 1 if he prefers candidate i over j otherwise mark it 0. No candidate is preferred over itself, so all the diagonal entries are filled with 0. The reason for representing a ballot as a matrix because the precise

- motivate the choice of ballots and compare with plaintext version
- define encryption function that changes representation
- a ballot is valid if and only if there is evidence that proves its validity
- encrypting valid ballots leads to valid ballots, and the same for invalid ballots
- Each ballot, B, is a $n\times$n matrix where n is the number of candidates participating in election. Each voter marks the entry (i, j) as 1 and (j, i) as -1 in ballot B if he prefers candidate i over j. If the voter has no preference between candidate i and j i.e. they are ranked same then voter would mark entry (i, j) and (j, i) as 0. No candidate is preferred over itself, so all the diagonal entries are filled with 0. The reason for representing a ballot as a matrix because the precise
  nature of our algorithm involving no decryption of individual ballot to protect ballot privacy, facilitating the scrutiny for validity, and ease of computing cumulative margin over encrypted data to compute the winners.
  Each entry in ballot is encrypted by additive ElGamal i.e. encrypting the message m as $(g^r, h^r g^m)$ where g is generator of cycle group G, and $h = g^x$ is private key. After performing the encryption, a plain text ballot i.e. matrix having entries of 0, and 1 changes to a matrix having pair for
- Each entry in ballot is encrypted by additive ElGamal i.e. encrypting the message m as $(g^r, h^r g^m)$ where g is generator of cycle group G of order q, private key x and randomness r are randomly chosen from $\{1, \ldots, q - 1\}$, and public key is computed as $h = g^x$. It is important to note that generator g, private key x and public key h are generated during key generation phase of Elgamal algorithm while randomness r is generated fresh for each message.

After performing the encryption, a plain text ballot i.e. matrix having entries of -1, 0, and 1 changes to a matrix having pair for each entry representing the encryption of corresponding plain text. A representation of plain text ballot

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} & \dots & x_{nn} \end{bmatrix}$$

and data structure used to represent this ballot in Coq is

```
Definition plaintext := Z.
Definition pballot := cand -> cand -> plaintext.
```

A representation of encrypted ballot

$$\begin{bmatrix} (y_{11}, z_{11}) & (y_{12}, z_{12}) & (y_{13}, z_{13}) & \dots & (y_{1n}, z_{1n}) \\ (y_{21}, z_{21}) & (y_{22}, z_{22}) & (y_{23}, z_{23}) & \dots & (y_{2n}, z_{2n}) \\ \dots & \dots & \dots & \dots & \dots \\ (y_{n1}, z_{n1}) & (y_{n2}, z_{n2}) & (y_{n3}, z_{n3}) & \dots & (y_{nn}, z_{nn}) \end{bmatrix}$$

and the corresponding data structure in Coq is

```
Definition ciphertext := (Z * Z)%type.
Definition eballot := cand -> cand -> ciphertext.
```

The reason for using additive ElGamal is that it's composability on ciphertext under same public key. In short, (Should I explain the algorithm here that why this scheme works or explain it encryption section ?)

$$enc_k(m_1) * enc_k(m_2) = enc_k(m_1 + m_2)$$

Our choice of representing ballot as matrix comes with price. A malicious voter could try to encode all sort of ballots including cyclic ones i.e. candidate A is preferred over candidate B, candidate B is preferred over candidate C, and candidate C is preferred over candidate A, and it is clear that voter has not expressed his choices clearly as it can not be expressed in linear preference order and should not be considered for counting, or he could inflate his favourite candidate by any other number higher that 1, and It makes the ballot invalid because any value greater that 1 in a ballot would be similar to voter has casted more than one vote. We define a ballot is valid if all the entries in matrix is encryption of -1, 0 and 1, and there is no cycle in it. Formally, We have

```
Definition matrix_ballot_valid (p : pballot) :=
      (forall c d : cand, In (p c d) [-1; 0; 1]) /\
       valid cand p.
```

```
Definition valid (A : Type) (P : A -> A -> Z) :=
     exists f : A -> nat,
         forall c d : A,
         (P c d = 1 <-> (f c < f d)%nat) /\
         (P c d = 0 <-> f c = f d) /\
         (P c d = -1 <-> (f c > f d)%nat)

    (* Which one should keep ? Upper one is in Code while this one is
       more intutive after expanding the definition *)
   Definition matrix_ballot_valid (p : pballot) :=
     (forall c d : cand, In (p c d) [-1; 0; 1]) /\
     (exists f : A -> nat,
         forall c d : A,
         (p c d = 1 <-> (f c < f d)%nat) /\
         (p c d = 0 <-> f c = f d) /\
         (p c d = -1 <-> (f c > f d)%nat))
```

Intuitively, it says that a ballot p is valid if we can find a function which expresses the notion of linear preference ordering for all candidates, and we prove that we can decide for each ballot if it is valid or not

```
  Lemma matrix_ballot_valid_dec :
        forall p : pballot, {matrix_ballot_valid p} +
                            {~matrix_ballot_valid p}.
```

By now, a attentive reader would note that we have claimed that we don't decrypt the ballot to preserve the privacy, so how can we decide the validity without decrypting the original ballot. We will discuss our whole cryptographic scheme in next section, but to quench the thrust of reader for now, we have proved that if ballot p was valid before encryption, then encrypting it followed by permuting each row by secret permutation σ and permuting again the each column of resulting matrix from previous step by same permutation, σ, and decrypting it back would still lead to valid ballot. (* Write this same Lemma for $matrix_b allot_v alid$ then any permutation of it is valid and if it is invalid then any permutation of it is invalid. The proof is already lying there in other proofs so it's basically copy paste. *)

```
Lemma perm_presv_validity :
    forall (A : Type) (P : A -> A -> Z),
      (forall c d : A, {c = d} + {c <> d}) ->
      (forall c d : A, {P c d = 1} + {P c d = 0} + {P c d = -1}) ->
      forall sig : A -> A, Bijective sig -> valid A P <-> valid A (perm A P sig)
```

And similarly, for invalid ballot

```
Lemma not_perm_persv_validity :
 forall (A : Type) (P : A -> A -> Z),
      (forall c d : A, {c = d} + {c <> d}) ->
      (forall c d : A, {P c d = 1} + {P c d = 0} + {P c d = -1}) ->
      forall sig : A -> A, Bijective sig -> ~ valid A P <-> ~ valid A (perm A P sig).
```

B Homomorphic Computation of Margin

– motivate and introduce how the margin is computed homomorphcially
– prove that encrption commutes with computation of margin

C Homomorphic Counting

– prove that encruption commutes with computation of winners
– discuss the evidence: publish cleartext margin function
– describe certificates and how to check them

D Experimental Results

– how long does it take, and how much memory?
– how large are the certificates?

E Discussion and Further Work

One aspect that we have not considered here is encryption of ballots to safeguard voter privacy which can be incorporated using protocols such as shuffle-sum [2] and homomorphic encryption [6]. The key idea here is to formalise a given voting scheme based on encrypted ballots, and then to establish a homomorphic property: the decryption of the result obtained from encrypted ballots is the same as the result obtained from the decrypted ballots. We leave this to further work.