Practical Strategy-Resistant Privacy-Preserving Elections

Sébastien Canard[1], David Pointcheval[2,3], Quentin Santos[1,2,3(✉)], and Jacques Traoré[1]

[1] Orange Labs, Caen, France
quentin.santos@orange.com
[2] DIENS, École normale supérieure, CNRS, PSL University, Paris, France
[3] INRIA, Paris, France

Abstract. Recent advances in cryptography promise to let us run complex algorithms in the encrypted domain. However, these results are still mostly theoretical since the running times are still much larger than their equivalents in the plaintext domain. In this context, Majority Judgment is a recent proposal for a new voting system with several interesting practical advantages, but which implies a more involved tallying process than first-past-the-post voting. To protect voters' privacy, such a process needs to be done by only manipulating encrypted data.

In this paper, we then explore the possibility of computing the (ordered) winners in the Majority Judgment election without leaking any other information, using homomorphic encryption and multiparty computation. We particularly focus on the practicality of such a solution and, for this purpose, we optimize both the algorithms and the implementations of several cryptographic building blocks. Our result is very positive, showing that this is as of now possible to attain practical running times for such a complex privacy-protecting tallying process, even for large-scale elections.

1 Introduction

1.1 Motivation

Practical Cryptography. Homomorphic encryption allows running algorithms in a way that preserves the confidentiality of sensitive data, and thus, in a lot of practical use-cases, the privacy of some individuals. Fully homomorphic encryption permits to treat almost all applications in the encrypted domain, but is today too slow for practical use. Conversely, additively homomorphic encryption (such as in ElGamal and Paillier cryptosystems) is reasonably efficient, but, since an algorithm is, most of the time, not solely a combination of additions, practical uses typically involve a hybrid approach. In this case, part of the process is done in the encrypted domain, but some intermediate results are decrypted, and the final stages are realized in the clear. This approach can give reasonable

J. Lopez et al. (Eds.): ESORICS 2018, LNCS 11099, pp. 331–349, 2018.
https://doi.org/10.1007/978-3-319-98989-1_17

confidentiality while remaining very effective. But, for most of existing practical needs, there is today no possibility to obtain at the same time real-world performances and full confidentiality (without revealing any intermediate value).

Electronic Voting. Electronic voting concerns itself with the problem of giving voters strong guarantees regarding their own privacy, as well as the integrity of the whole election.

Referendums adapt very well to additively homomorphic encryption, since it then suffices to have voters encrypt 1 for "Yes" and 0 for "No", and then tally the ballots using homomorphic addition. It is moreover usually required that each voter provides a cryptographic proof that their ballot contains either 0 or 1 (without revealing which). First-past-the-post voting, where each voter selects a single candidate among several possible ones, and for which the candidate with the most votes wins, can be implemented in a similar fashion (but in less efficient way, since cryptographic proofs become more complicated).

Other voting systems such as Majority Judgment or Single Transferable Vote (STV) offer interesting properties, such as strategy-resistance, but are more complex, especially for the tallying phase. In both examples, one possibility is to homomorphically aggregate all the votes before decryption, and then perform the final steps in the clear. Although this approach is generally acceptable with relation with the confidentiality of the voters, it does reveal information such as the scores of all the candidates. But candidates who were eliminated may not want their exact scores to be known.

As a consequence, we explore in this paper the possibility of running such a voting system in a way that only reveals the winning candidate, or the ordered winners. As we will see, the related work in this domain (with such strong desired confidentiality property) is quite inexistent.

1.2 Related Work

Secure implementations of referendum and first-past-the-post voting have been regularly studied in cryptographic literature. In particular, [DK05] offers a solution for running referendums without revealing the exact count (only whether "Yes" or "No" won) that does not rely on MixNets or on any trusted server. As for strategy-resistant voting systems, [TRN08,BMN+09] explore the case of privacy-preserving STV using a mixing protocol. As far as we know, no such study nor implementation has been done for the Majority Judgment voting system.

1.3 Our Contributions

In this paper, we propose an implementation of the Majority Judgment voting system on a restricted set of logical gates, which we build using the Paillier cryptosystem so as to provide distributed trust, while keeping performance manageable. We then run benchmarks against our Python implementation based on the `gmpy2` wrapper library.

$$\begin{array}{c} \\ \text{Candidates} \end{array} \begin{array}{c} \begin{array}{ccccc} + & \leftarrow \text{Grades} \rightarrow & & & - \\ \text{A} & \text{B} & \text{C} & \text{D} & \text{E} \end{array} \\ \left\{ \begin{array}{l} \text{Alice} \\ \text{Bob} \\ \text{Charlie} \end{array} \right. \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} = \mathcal{B}_i$$

Fig. 1. Single ballot: this voter attributes C to Alice, B to Bob and D to Charlie

We stress that our approach provides strong cryptographic guarantees regarding confidentiality and integrity of all the voters. Additionally, we support multi-seat elections, where several winners can be determined. All this is done without revealing any intermediate value while obtaining real-world practical results since, e.g., for 5 candidates and 1000 voters, the tallying phase works in less than 10 min to give the winner.

Additionally to the real voting application, we think that our work serves to show that it is really possible to use cryptography in a very conservative setting (revealing as little information as possible) while still being very practical (actually revealing the result in reasonable time). We hope it can help bridge the gap that continues to exist between theoretical cryptography (strong guarantees) and industrial practices (high efficiency) and encourage a more widespread use of cryptography to improve users' privacy in many applications.

2 Majority Judgment

2.1 Definition

Principle. Majority Judgment was presented by Michel Balinski and Rida Laraki in [BL07,BL10]: this is a voting mechanism that claims to improve the legitimacy of the elected candidates.

In Majority Judgment, each voter $\mathcal{V}_i$ attributes a grade to each candidate $\mathcal{C}_i$. Grades need not be numbers, but do need to be ordered (with a strict ordering, from the best to the worst grade, such as, e.g., from A to E). For this, the ballot is structured as a matrix where each row represents a candidate and each column represents a grade; the voter writes a 1 in the chosen grade for each candidate and 0 in other cells (See Fig. 1).

After summing all the votes into an Aggregate Matrix A, the median grade (or "majority-grade") of each candidate is computed: the median grade corresponds to the grade for which there are as many votes for worse grades as for better grades. The candidate with the best median grade is elected, or the candidate with the worst median grade is eliminated.

In Fig. 2, Alice and Bob have the same best median grade, C, and Charlie has the worst median grade, E. Charlie is eliminated, but we cannot yet decide the winner (here, all the candidates get the same total number of grades, so values can be seen as ratios).

Solving Ties. As in our above example, it is likely that several candidates get the same median grades. In that case, for these candidates, we consider the grades lower than the median grade and the grades greater than the median grade to make a decision. We construct the Tiebreak Matrix T by aggregating grades to the left and to the right of the median grade, see Fig. 3. To make a decision, the largest of these values is considered. If it pertains to the right column (many low grades), then the corresponding candidate is eliminated. If it pertains to the left column (many high grades), then the corresponding candidate wins.

More formally, from Balinski and Laraki, the result of each candidate can be summed up into a triplet (p, α, q) where α represents the majority-grade, p represents the ratio of votes above the candidate's majority-grade α, and q the ratio of those below. For any two candidates $\mathcal{C}_A$ and $\mathcal{C}_B$ with corresponding triplets (p_A, α_A, p_B) and (p_B, α_B, q_B), then $\mathcal{C}_A$ wins against $\mathcal{C}_B$ when one of the following (mutually exclusive) conditions is met:

1. $\alpha_A > \alpha_B$ (better median grade);
2. $\alpha_A = \alpha_B \wedge p_A > q_A \wedge p_B < q_B$ ("stronger"[1] median grade);
3. $\alpha_A = \alpha_B \wedge p_A > q_A \wedge p_B > q_B \wedge p_A > p_B$ (more secure median grade);
4. $\alpha_A = \alpha_B \wedge p_A < q_A \wedge p_B < q_B \wedge q_A < q_B$ (less insecure median grade).

This defines a total ordering on the candidates with high probability. Our goal is to output the names of the candidates according to this ordering.

We now consider a set of voters $\mathcal{V}_i$, of candidates $\mathcal{C}_i$ and of authorities $\mathcal{A}_i$ (that will perform the counting). Our aim is to propose an implementation of Majority Judgment in the encrypted domain, in order to output the above ordering, but without leaking any additional information. For this purpose, we have to both find the suitable encryption scheme and provide the best possible description of such a voting system, so as to obtain the best possible privacy-preserving achievement for our problem.

Justification. As an electoral system, Majority Judgment gives the voters better incentives to simply vote for their preferred candidates rather than strategically vote for another candidate. For instance, in first-past-the-post voting, voters

$$\begin{array}{rl} & \begin{array}{ccccccc} & + & \leftarrow \text{Grades} \rightarrow & & & & - \\ & \text{A} & \text{B} & \text{C} & \text{D} & \text{E} \end{array} \\ \text{Candidates} \begin{cases} \text{Alice} \\ \text{Bob} \\ \text{Charlie} \end{cases} & \begin{pmatrix} 31 & 151 & \mathbf{529} & 254 & 35 \\ 21 & 48 & \mathbf{442} & 301 & 188 \\ 101 & 7 & 2 & 86 & \mathbf{804} \end{pmatrix} = A \end{array}$$

Fig. 2. Aggregate Matrix: each cell represents the number of voters who gave this grade to that candidate; in bold are the candidates' median grades.

[1] when $p > q$, the median grade is strong and noted α^+; when $p < q$, weak, noted α^-.

$$\begin{pmatrix} 31+151 & 254+35 \\ 21+48 & 301+188 \end{pmatrix} = \begin{pmatrix} 182 & 289 \\ 69 & \mathbf{489} \end{pmatrix} = T$$

Fig. 3. Tiebreak Matrix: candidates with equal median; the **largest value** rejects Bob.

select a single candidate on their ballots, and the candidates who was selected by the most voters wins the election; in this settings, voters are incentivized not to vote for lesser-known candidates, feeling like they are wasting their vote on a candidate with little chance of winning the election. This is partly addressed in two-round systems, where voters are given two opportunities to state their opinions: in the first round, they can vote for their favorite candidate, knowing that they will be able to express their preference between the candidates selected for the second round, who are usually among the most well-known ones.

However, even the two-round system remains imperfect, which translates in a restricted number of parties gathering most of the votes (usually two in the US, three in France). Several ranking systems propose to ameliorate this situation by allowing the voters to explicitly list their favorite candidates, ensuring that their opinion is take into account even when their first choice is eliminated. The most well-known such system is the Single Transferable Vote, also known as Instant-Runoff Voting for single-winner elections. However, this voting system is particularly complex as it requires each individual ballot to be considered potentially several times, usually by hand. Majority Judgment is a more recent proposal for a ranked voting system which allows to aggregate the ballots before counting.

By shifting the incentives of the voters away from strategic voting, such voting systems might improve the legitimacy of political and administrative elections by giving each voter the feeling that their opinion was fully taken into account, and potentially increase the turnout.

2.2 Removing Branching

At first sight, it may seem like an algorithm implementing Majority Judgment would require complex control flow with branching instructions (conditional structures and loops), depending on whether a candidate is eliminated or not. This would incur important overheads when evaluated in the encrypted domain (both branches must be computed). However, it is possible to devise a branchless algorithm without introducing such an important overhead.

Early Elimination of Candidates. The first remark is that we can avoid explicitly checking condition 1, but we can build the Tiebreak Matrix with just the best median grade. In the previous use-case, C is the best median grade (for Alice and Bob), which leads to the following Tiebreak Matrix

$$T = \begin{pmatrix} 31+151 & 254+35 \\ 21+48 & 301+188 \\ 101+7 & 86+804 \end{pmatrix} = \begin{pmatrix} 182 & 289 \\ 69 & 489 \\ 108 & \mathbf{892} \end{pmatrix}$$

Then Charlie gets eliminated, since he holds the highest number on the right column.

More generally, let W and L be candidates such that W wins against L by above condition 1 (e.g. $\alpha_W < \alpha_L$). Let us define p'_L (resp. q'_L) the ratio of votes for L that are better (resp. worse) than α_W (instead of α_L). Since $\alpha_W < \alpha_L$, $p'_L < 1/2 < q'_L$, but we also have $q_W < 1/2$, and thus $q_W < q'_L$. As a consequence, we only need to compute the p' and q' values, defined around the best median grade (rather than each candidate's median grade) and use the following (mutually exclusive) conditions to determine whether candidate $\mathcal{C}_A$ wins against candidate $\mathcal{C}_B$:

2'. $p'_A > q'_A \wedge p'_B < q'_B$;
3'. $p'_A > q'_A \wedge p'_B > q'_B \wedge p'_A > p'_B$;
4'. $p'_A < q'_A \wedge p'_B < q'_B \wedge q'_A < q'_B$.

Building the Tiebreak Matrix T. To build the Tiebreak Matrix T, we need to detect which elements are to the left (resp. right) of the best median grade. For this, we first compute the Candidate Matrix $C = (c_{i,j})$, such that $c_{i,j} = 1$ when column j represents a grade which is better than the candidate's median grade. From the Aggregate Matrix $A = (a_{i,j})$ (with the number of grades for each candidates), we want

$$c_{i,j} = \begin{cases} 1 & \text{if } 2 \times \sum_{k<j} a_{i,k} < \sum_k a_{i,k} \\ 0 & \text{otherwise.} \end{cases}$$

In our use-case, one gets the following Candidate Matrix C, where the zeroes in bold are first in their each line, and correspond to the median grade for each candidate:

$$C = \begin{pmatrix} 1 & 1 & \mathbf{0} & 0 & 0 \\ 1 & 1 & \mathbf{0} & 0 & 0 \\ 1 & 1 & 1 & 1 & \mathbf{0} \end{pmatrix}.$$

Then, we can compute the Grade Vector $G = (g_j)$, such that $g_j = 1$ when column j represents a grade which is greater than the global median grade. We can easily compute $G = (g_j)$ from $C = (c_{i,j})$ since $g_j = \wedge_i c_{i,j}$, where G and C are Boolean matrices, and 0 and 1 respectively represent False and True:

$$G = \begin{pmatrix} 1 & 1 & \mathbf{0} & 0 & 0 \end{pmatrix}.$$

Again, the first zero, in bold, corresponds to the global (the best) median grade.

Once we have this Grade Vector G, we can build the two columns of the Tiebreak Matrix $T = (t_{i,k})$, from the Aggregate Matrix $A = (a_{i,j})$, as:

$$t_{i,1} = \sum_{\substack{j \\ g_j=1}} a_{i,j} \text{ and } t_{i,2} = \sum_{\substack{j \\ g_{j-1}=0}} a_{i,j}.$$

Note that the second column uses a shifted version of G to filter votes below the median grade. With the integer representation of Boolean, that can be written as $t_{i,1} = \sum_j g_j \times a_{i,j}$ and $t_{i,2} = \sum_j (1 - g_{j-1}) \times a_{i,j}$.

Identifying the Winner. Once we have built the Tiebreak Matrix $T = (t_{i,k})$, it only remains to reveal for each candidate the result of the following explicit Boolean formula, which selects the Winner:

$$w_i = \bigwedge_{\substack{j \\ j \neq i}} \begin{pmatrix} (t_{i,1} > t_{i,2} \wedge t_{j,1} < t_{j,2}) \\ \vee (t_{i,1} > t_{i,2} \wedge t_{j,1} > t_{j,2} \wedge t_{i,1} > t_{j,1}) \\ \vee (t_{i,1} < t_{i,2} \wedge t_{j,1} < t_{j,2} \wedge t_{i,2} < t_{j,2}) \end{pmatrix}$$

Notice that $w_i = 1$ when candidate i beats all the other candidates by either condition 2', 3' or 4', hence the definition for the Winner Vector $W = (w_i)$. Once W has been computed, we search the unique component equal to 1, identifying the elected candidate.

If one wants more than one winner, one can run again the above protocol, after having removed the line of the winner in the Candidate Matrix C. The computation has to be run again to generate G, T, and W, interactively for all the winners.

2.3 Expected Features for Encrypted Scheme

If we assume that the votes are encrypted, running the Majority Judgment algorithm means that we need to use an encryption scheme that allows the following operations (without knowing the decryption key):

- *addition of two plaintexts*, to compute the Aggregate Matrix A and the Tiebreak Matrix T;
- *comparison of two plaintexts*, to compute the Candidate Matrix C and the Winning Vector W;
- *AND/OR Boolean gate between two plaintexts*, to compute the Grade Vector G and the Winning Vector W;
- *multiplication of two plaintexts*, to compute the Tiebreak Matrix T.

Eventually, a distributed decryption of the Winning Vector W would provide the final result.

Notice that the operations required for computing the Tiebreak Matrix T actually only multiply some value by 0 or 1 (i.e. one operand is restricted to $\{0, 1\}$, actually a Boolean). Indeed, multiplications are merely a conditional filter on the elements to be summed. We can thus relax our requirement from a full multiplication gate to a "*conditional gate*": $\mathsf{CondGate}(x, y) = x \times y$ for $y \in \{0, 1\}$.

If we use an additively homomorphic encryption scheme, addition gives us the logical NOT gate, since $\neg x = 1 - x$ when 0 means False and 1 means True. Then, the conditional gate gives us the AND Boolean gate, as $x \wedge y = x \times y$. Eventually, these two gates let us construct the OR and XOR gates.

In the following, we will thus use an additively homomorphic encryption scheme, together with efficient Multi-Party Computation (MPC) protocols for distributed decryption, distributed evaluation of the conditional gate, and distributed comparison. As we will explain, multi-party computation necessitates in particular the use of some zero-knowledge proofs of correctness of the computed values. In the next section, we then give all the basic cryptographic material we will need for our solution.

3 Cryptographic Tools

3.1 Paillier Encryption Scheme

Our scheme relies on the Paillier encryption scheme [Pai99], for its additively homomorphic property, and the fact that distributed decryption can be efficiently done on arbitrary ciphertexts.

Let p and q be two large safe primes (so that $p = 2p'+1$ and $q = 2q'+1$ where p' and q' are also primes), set $n = pq$, $\varphi = 4p'q'$, $g = 1+n$, and $s = n^{-1} \bmod \varphi$. Then $\mathsf{pk} = (n, g)$ and $\mathsf{sk} = (\mathsf{pk}, s)$. The encryption/decryption algorithms work as follows, for $M \in \mathbb{Z}_n$.

- $\mathsf{Encrypt}(\mathsf{pk}, M)$: pick $r \xleftarrow{\$} \mathbb{Z}_n^*$, return $C = g^M r^n \bmod n^2$;
- $\mathsf{Decrypt}(\mathsf{sk}, C)$: compute $R = C^s \bmod n$, and return $M = \frac{(CR^{-n} \bmod n^2)-1}{n}$.

Indeed, since $g = 1 + n$, $C = r^n \bmod n$, we can recover $R = C^s = r \bmod n$ and thus obtain M from $CR^{-n} = g^M = 1 + Mn \bmod n^2$.

This encryption scheme is well-known to be additively homomorphic, but it also allows efficient distributed decryption among the authorities, with a threshold: as explained in [Sho00], one can distribute s using a Shamir Secret Sharing mechanism, modulo $\varphi = 4p'q'$. Since $|n - \varphi| = 2(p' + q') + 1 < n^{1/2}$, a random element in $\mathbb{Z}_\varphi$ follows a distribution that is statistically indistinguishable from a random element in $\{0, \ldots, n-1\}$. Hence, one can choose a random polynomial P of degree $t-1$ in $\mathbb{Z}_\varphi$ so that $P(0) = s = n^{-1} \bmod \varphi$, and set $s_i = P(i)$ for $i = 1, \ldots, k$, the k authorities. If less than t of these authorities collude, no information leaks about s, while any t of them can reconstruct Δs, where $\Delta = k!$, with

$$\lambda_i^S = \left(\prod_{j \in S \setminus \{i\}} j \right) \times \left(\frac{\Delta}{\prod_{j \in S \setminus \{i\}} (j - i)} \right)$$

since for any set S of t elements, $\sum_{i \in S} \lambda_i^S \cdot s_i = \Delta \cdot P(0) = \Delta \cdot s$. One should remark that the denominator divides $i!(k-i)!$, which in turns divides $\Delta = k!$, as noted in [Sho00]. Hence, λ_i^S is an integer.

Then, each authority $\mathcal{A}_i$ just has to compute $R_i = C^{s_i} \bmod n$, and the simple combination leads to R: with $R' = \prod_{i \in S} R_i^{\lambda_i^S} = C^{\Delta s}$, one has $R'^n = C^\Delta \bmod n$.

But Δ and n are relatively prime and so there exist u and v such that $un+v\Delta = 1$: $(C^u R'^v)^n = C \bmod n$. As a consequence, from the R_i's, anybody can compute

$$R = C^u \times \left(\prod_{i \in S} R_i^{\lambda_i^S} \right)^v \bmod n.$$

Then, any subset of t authorities can compute and publish R, which leads to $M = (CR^{-n} \bmod n^2 - 1)/n$.

This encryption scheme achieves indistinguishability against chosen-plaintext attacks (IND-CPA) under the High-Residuosity assumption, which claims that the following High-Residuosity problem is hard.

High-Residuosity Problem (HR). For an RSA modulus $n = pq$, the challenger chooses a random element $r_0 \xleftarrow{\$} \mathbb{Z}^*_{n^2}$, a random element $R \xleftarrow{\$} \mathbb{Z}^*_n$ and sets $r_1 = R^n \bmod n^2$, and eventually outputs r_b for a random bit b, the adversary has to guess b.

This also holds for the distributed decryption, when the authorities are honest-but-curious. To ensure correctness of the decryption values R_i, each authority $\mathcal{A}_i$ must prove that R_i is the result of the exponentiation of C to the power s_i.

3.2 Zero-Knowledge Proofs

In the following, we will have two kinds of proofs of equality of discrete logarithms: when the order of the group is known, and when the order of the group is not known.

Chaum-Pedersen Protocol [CP93]**.** Let G be a cyclic group of known order q, and κ the security parameter. To prove knowledge (or just existence) of $x \in \mathbb{Z}_q$ such that $y_1 = g_1^x$ and $y_2 = g_2^x$ for $g_1, g_2, y_1, y_2 \in G$, the prover $\mathcal{P}$ can proceed as follows with a verifier $\mathcal{V}$:

- $\mathcal{P}$ picks $u \xleftarrow{\$} \mathbb{Z}_q$ and sends commitments $t_1 \leftarrow g_1^u$ and $t_2 \leftarrow g_2^u$
- $\mathcal{V}$ sends a challenge $h \xleftarrow{\$} \mathbb{Z}_{2^\kappa}$
- $\mathcal{P}$ returns $w \leftarrow u - hx \bmod q$
- $\mathcal{V}$ checks that $g_1^w = t_1 y_1^{-h}$ and that $g_2^w = t_2 y_2^{-h}$.

This protocol is only known to be zero-knowledge when the verifier is honest. But one can make it non-interactive, in the random oracle model using the Fiat-Shamir heuristic [FS87, PS96].

Fiat-Shamir Heuristic [FS87, PS96]**.** The challenge can be replaced by the output of hash function (modeled as a random oracle), removing the interaction needed to obtain h, and thus making the proof non-interactive and fully zero-knowledge. Then, one can set $h = H(g_1, g_2, y_1, y_2, t_1, t_2)$, and the verifier can simply check whether $h = H(g_1, g_2, y_1, y_2, g_1^w y_1^h, g_2^w y_2^h)$. The proof just consists of the pair $(h, w) \in \mathbb{Z}_{2^\kappa} \times \mathbb{Z}_q$.

Girault-Poupard-Stern Protocol [GPS06]**.** After Girault's [Gir91] work, Poupard and Stern [PS98,PS99] studied the proof of knowledge of a discrete logarithm in groups of unknown order. It leads to a similar protocol as above, for the proof of equality of discrete logarithms, but with some margins: to prove knowledge (or just existence) of $x \in \mathbb{Z}_q$, for an unknown $q < Q$ with Q public, such that $y_1 = g_1^x$ and $y_2 = g_2^x$ for $g_1, g_2, y_1, y_2 \in G$, the prover $\mathcal{P}$ can proceed as follows with a verifier $\mathcal{V}$:

- $\mathcal{P}$ picks $u \xleftarrow{\$} \mathbb{Z}_{2^{2\kappa}Q}$ and sends commitments $t_1 \leftarrow g_1^u$ and $t_2 \leftarrow g_2^u$
- $\mathcal{V}$ sends a challenge $h \xleftarrow{\$} \mathbb{Z}_{2^\kappa}$
- $\mathcal{P}$ returns $w \leftarrow u - hx$
- $\mathcal{V}$ checks that $0 < w < 2^{2\kappa}Q$, and both $g_1^w = t_1 y_1^{-h}$ and $g_2^w = t_2 y_2^{-h}$.

Again, to make this proof non-interactive, one can set $h = H(g_1, g_2, y_1, y_2, t_1, t_2)$, and the verifier can simply check whether $h = H(g_1, g_2, y_1, y_2, g_1^w y_1^h, g_2^w y_2^h)$. The proof just consists of the pair $(h, w) \in \mathbb{Z}_{2^\kappa} \times \mathbb{Z}_{2^{2\kappa}Q}$.

Since w can fall outside the correct set, but with negligible probability, the verifier can just verify it, and re-start the proof when w is wrong.

3.3 Proofs of Valid Decryption

During the counting phase, the authorities have to provide a proof that they have properly decrypted some values. We here give some details on that cryptographic tool.

In fact, in the Paillier cryptosystem, the main operation of decryption is to raise the ciphertext to the secret power s_i. Let $C \in \mathbb{Z}_{n^2}^*$ be a ciphertext, $s_i \in \mathbb{Z}_\varphi$ be a secret exponent known to the prover $\mathcal{P}$ only (one authority $\mathcal{A}_i$ in our case); $\mathcal{P}$ must provide $R_i = C^{s_i} \bmod n$, and prove it. For this, one can use the above proof of equality of discrete logarithms, with a reference value $v_i = v^{s_i} \bmod n$, where v is a generator of Q_n, the cyclic group of the quadratic residues in $\mathbb{Z}_n^*$.

Now going to the threshold version, one can thus assume that when each prover/authority receives their secret s_i, the verification value $v_i = v^{s_i} \bmod n$ is published, with a public generator v. Since we need to work in a cyclic group, the prover will prove that the same exponent s_i has been used in

$$R_i^2 = (C^2)^{s_i} \bmod n \quad \text{and in} \quad v_i = v^{s_i} \bmod n.$$

Batch Proofs. We can reduce the cost of this protocol by batching the proofs of valid decryptions, when several decryptions are performed by the same prover on several ciphertexts, as done in [APB+04, Appendix C, pp. 15–16]. For several ciphertexts $(C_j)_j$, $\mathcal{P}$ first publishes the computations $R_j = C_j^s \bmod n$, for all j; then, the verifier $\mathcal{V}$ (or a hash function if using the Fiat-Shamir heuristic), generates a sequence of random scalars $\alpha_j \xleftarrow{\$} \mathbb{Z}_{2^\kappa}$ to build the aggregations $C^* = \prod_j C_j^{\alpha_j} \bmod n$ and $R^* = \prod_j R_j^{\alpha_j} \bmod n$. They should also satisfy $(R^*)^2 = (C^{*2})^{s_i} \bmod n$, which can be proven as above: $\log_v v_i = \log_{C^{*2}} R^{*2}$.

3.4 Proof of Private Multiplication

As explained above in the Majority Judgment description, the tallying process requires multiplication gates which, in the encrypted domain, should be evaluated privately, and in a provable manner. Then, an authority $\mathcal{A}_i$, acting as a prover $\mathcal{P}$, needs to provide a proof of private multiplication, which can be done as follows.

Let $x, y \in \mathbb{Z}_n$ and C_y an encryption of y. Let $\mathcal{P}$ be a prover (an authority in our case) knowing x and C_y. It computes C_z, an encryption of $x \times y \bmod n$, which can be done using private multiplication: $C_z = C_y^x r_z^n$ for $r_z \xleftarrow{\$} \mathbb{Z}_n^*$. Then, $\mathcal{P}$ must provide C_x and C_z to verifier $\mathcal{V}$ and prove that C_x, C_y, C_z are encryptions of some x, y, and z such that $z = x \times y \bmod n$. For this:

- $\mathcal{P}$ draws $u \xleftarrow{\$} \mathbb{Z}_n$, $r_u, r_{yu} \xleftarrow{\$} \mathbb{Z}_n^*$ and sends $C_u = g^u r_u^n \bmod n^2$ and $C_{yu} = C_y^u r_{yu}^n \bmod n^2$ to $\mathcal{V}$;
- $\mathcal{V}$ draws a challenge $e \xleftarrow{\$} \mathbb{Z}_{2^\kappa}$ and sends it to $\mathcal{P}$;
- $\mathcal{P}$ sends $w = u - xe \bmod n$, $r_w = r_u r_x^{-e} \bmod n$ and $r_{yw} = r_{yu} r_z^{-e} \bmod n$ to $\mathcal{V}$;
- $\mathcal{V}$ checks that $C_u = g^w r_w^n C_x^e \bmod n^2$ and $C_{yu} = C_y^w r_{yw}^n C_z^e \bmod n^2$.

This proof can be converted into a non-interactive zero-knowledge proof as above, using the Fiat-Shamir heuristic.

When several proofs have to be conducted in parallel, with the same x and multiple y_i, the above batch proof technique can be applied again, thanks to the linear property of the multiplication: $x \times (\sum_i \alpha_i y_i) = \sum_i \alpha_i (x \times y_i) \bmod n$. Unfortunately, it cannot be applied for independent pairs (x_i, y_i). In consequence, most of the running-time of our implementation is spent computing these proofs and verifying them.

4 Gate Evaluation with Multi-party Computation

We now present how to implement the operations listed in Subsect. 2.3 using MPC protocols.

- Decryption: this is a common operation in MPC protocols, and is also used by the other two protocols below;
- Conditional gate: this can be performed by randomizing the Boolean operand, decrypting it using the previous protocol, and then performing a private multiplication;
- Comparison: this is the most complex operation, which implies to first extract the bits of the operands (using masked decryption), and then performing a bitwise addition (using conditional gates).

As we will see, they can be turned into the malicious setting by additionally providing proofs of correct execution, which will essentially be proofs of equality of discrete logarithms, as already seen for the proof of valid (partial) decryption.

Input: C_x, the encryption of $x \in \mathbb{Z}_n$ and C_y, the encryption of $y \in \{-1, 1\}$
Output: C_z, the encryption of $z = x \times y$
foreach *authority* **do**
  $o \xleftarrow{\$} \{-1, 1\}$
  $C_x \leftarrow {C_x}^o$; `/* homomorphically compute encryption of` $x \times o$ `*/`
  $C_y \leftarrow {C_y}^o$; `/* homomorphically compute encryption of` $y \times o$ `*/`
  pass C_x, C_y to the next authority
end
$y \leftarrow \mathsf{Decrypt}(C_y)$
assert $y \in \{-1, 1\}$
return $C_z = {C_x}^y$; `/* homomorphically compute encryption of` $x \times y$ `*/`

Algorithm 1: CondGate: Conditional gate

4.1 Decryption Gate

As already seen in Subsect. 3.3, one must compute $R = C^s = r \bmod n$ in order to allow full decryption. And this can be performed when s has been distributed using a secret sharing scheme "à la Shamir". We do not detail more, since this just consists on one flow from each authority, even in the malicious setting, with non-interactive zero-knowledge proofs of valid exponentiation (see above).

4.2 Conditional Gate

Schoenmakers and Tuyls introduced the conditional gate in [ST04]. Given $x \in \mathbb{Z}_n$ and $y \in \{-1, 1\}$, this gates computes $x \times y$, which is thus either x or $-x \bmod n$. Although the second parameter is restricted to $\{-1, 1\}$, it is very easy to adapt this into a gate taking $y \in \{0, 1\}$ using the homomorphic property of the Paillier cryptosystem. Thus, it can be used to implement logical AND gates, as well as multiplications by a Boolean.

Conditional Gate. The idea is simply to mask y by a random element in $\{-1, 1\}$ before decrypting it; then, it is easy to compute $x \times y \bmod n$ using private multiplication. To mask y, each authority will in turn multiply it by either -1 or 1. They will also apply the same transformation on x to keep the product unchanged. The algorithm is presented on Algorithm 1, where the decryption of C_y is performed in a distributed way as shown before.

To ensure the security of this gate against malicious adversary, each authority must provide a proof of equality of discrete logarithm o in the exponentiation of C_x and C_y, and verify the proofs of the other authorities before decrypting the final result of the election. Note that, as explained in [ST04], it is not necessary to require proofs that $o \in \{-1, 1\}$ as long as C_y indeed eventually decrypts to $y \in \{-1, 1\}$. However, such proofs may be requested when the condition does not hold, so as to prune misbehaving authorities.

Mapping to $\{0, 1\}$. As said above, the conditional gate can be easily adapted to accept its second operand from $\{0, 1\}$ using the additive property: it allows to

Input: (C_{x_i}), the bitwise encryption of $x \in \mathbb{Z}_n$ and (y_i), the bitwise notation of $y \in \mathbb{Z}_n$
Output: (C_{z_i}), the bitwise encryption of $z = x + y$
$C_c \leftarrow \mathsf{Encrypt}(0)$; /* carry */
foreach *index i, from 0 — the least significant bit*— **do**
  $C_{x_i \oplus y_i} \leftarrow C_{x_i} \times C_{y_i} \div {C_{x_i}}^{2y_i}$; /* $x_i \oplus y_i = x_i + y_i - 2x_i y_i$ */
  $C_{z_i} \leftarrow C_{x_i \oplus y_i} \times C_c \div \mathsf{CondGate}(C_{x_i \oplus y_i}, C_c)^2$; /* $(x_i \oplus y_i) \oplus c$ */
  $C_c \leftarrow (C_{x_i} \times C_{y_i} \times C_c \div C_{z_i})^{\frac{n+1}{2}}$; /* $(x_i + y_i + c - z_i)/2$ */
end
return (C_{z_i})

Algorithm 2: PrivateAddGate: Private Addition Gate

convert the ciphertext C_y of y into a ciphertext of $2y-1$. This leads to C_z being a ciphertext of $z = 2xy - x$. Using C_x, the ciphertext of x, one can additively get the ciphertext of $2xy$, which one can multiply by $2^{-1} \bmod n$ to get a ciphertext of xy.

4.3 Greater-Than Gate

Comparing two integers is done by evaluating a classical comparison circuit on their bitwise encryptions (each of their bit encrypted separately). Converting a scalar encryption of an integer into its bitwise encryption is performed by the bit-extraction gate.

Conceptually, the bit-extraction gate works as follows: one first masks the input integer, decrypts the result, encrypts the individual bits of this result, and then applies a binary addition circuit to unmask the bitwise encryption. Note that this binary addition circuit will take the masking operand into its unencrypted form, since it saves several executions of the conditional gate.

Private Addition Gate. The private addition gate is used to unmask an encrypted value. In the following algorithm (Algorithm 2), x_i is known as an encrypted value C_{x_i}, while y_i is a plaintext value. Knowing y_i, we can trivially compute encryptions C_{y_i} for homomorphic operations.

We note that [DFK+06, Section 6, pp. 13–15] offers a constant round circuit for addition (both operands encrypted). However, the constant is 37. Having one operand in the clear let us avoid interactions in line 1 of CARRIES (using private multiplications), bringing this constant down to 36. In contrast, our straightforward private addition gate implies ℓ rounds for up to $2^\ell - 1$ votes. In most practical cases, we expect $\ell < 36$, so using the constant round version is not preferable in our practical use case.

Bit-Extraction Gate. From this private addition gate, Schoenmakers and Tuyls propose a method to extract the bits of an integer encrypted with the Paillier cryptosystem [ST06, LSBs gate]. The general idea is to create a mask

Input: C_x, the encryption of $x \in \mathbb{Z}_n$
Output: (C_{x_i}), the bitwise encryption of x
/* C_y is the encryption of $y \xleftarrow{\$} \mathbb{Z}_n$, and (C_{y_i}) of the bits y_i */
/* Mask C_x with C_y, decrypt it into z, reencrypt bitwise, unmask */
$C_z \leftarrow C_x \div C_y$; /* $z = x - y$ */
$z \leftarrow \mathsf{DecryptGate}(C_z)$; /* as relative number in $[-n/2, n/2]$ */
$(C_{x_i}) \leftarrow \mathsf{PrivateAddGate}((C_{y_i}), (z_i))$; /* $(x_i) = (y_i) + (z_i)$ */
return (C_{x_i})

Algorithm 3: BitExtractGate: Bit Extraction Gate

whose scalar and bitwise encryptions are both known, apply it to the scalar input, and then remove it bitwise after reencryption.

To generate the mask, two more protocols are used to generate encrypted random integers and encrypted random bits. We skip the details of these protocols since they can be executed in the precomputation step. Details can be found in [ST06]. See Algorithm 3.

Greater-Than Gate. Using the conditional gate, we can compare two integers given as bitwise encryptions (C_{x_i}) and (C_{y_i}). For this, we can use the comparison circuit from [ST04], which evaluates $t_{i+1} \leftarrow (1-(x_i-y_i)^2)t_i + x_i(1-y_i)$, starting from $t_0 = 0$. See Algorithm 4.

Input: (C_{x_i}), (C_{y_i}) the bitwise encryptions of $x, y \in \mathbb{Z}_n$
Output: $C_{x>y}$, encryption of 1 if $x > y$, and 0 otherwise
$C_t \leftarrow \mathsf{Encrypt}(0)$; /* temporary result */
foreach *index i, from 0 — the least significant bit—* **do**
  $C_{x_i \wedge y_i} \leftarrow \mathsf{CondGate}(C_{x_i}, C_{y_i})$
  $C_a \leftarrow C_1 \div C_{x_i} \div C_{y_i} \times {C_{x_i \wedge y_i}}^2$; /* $1 - (x_i - y_i)^2$ */
  $C_b \leftarrow \mathsf{CondGate}(C_a, C_t)$; /* $(1 - (x_i - y_i)^2)t_i$ */
  $C_t \leftarrow C_b \times C_{x_i} \div C_{x_i \wedge y_i}$; /* $(1 - (x_i - y_i)^2)t_i + x_i(1 - y_i)$ */
end
return C_t

Algorithm 4: GTGate: Greater-Than Gate

We note that [DFK+06] also offers a constant round circuit for comparison. However, the constant is 19, which is not necessarily lower than the number of bits required in our use-case.

5 Implementation

5.1 Encryption of the Ballots

We remind a voter that attributes C to Alice, B to Bob and D to Charlie will cast a ballot as shown on Fig. 1, which consists of a series of 0 and 1, with

exactly one 1 per row. To participate in our secure version of an election using Majority Judgment, a voter encrypts their ballot element-wise and provides zero-knowledge proofs that each element is either 0 or 1 (OR-proof), and that there is one 1 per row (for instance, prove that the sum of each row decrypts to 1).

From this, it is easy to assemble the Aggregate Matrix A. A purely additively homomorphic implementation would decrypt A at this point and then proceed in the clear, assuming that aggregating the votes together provides adequate confidentiality. In our implementation however, we implement the approach described in Subsect. 2.2.

5.2 Avoiding Final Logical Gates

Notice that, since we use 0 and 1 to represent our Boolean values, $\bigvee_i x_i \Leftrightarrow \sum x_i \neq 0$ for any Boolean values (x_i). We can apply this remark to avoid evaluating the last part of the Boolean circuit on the negation of w_i:

$$\neg w_i = \bigvee_{j \neq i} \begin{pmatrix} \neg(t_{i,1} > t_{i,2} \wedge t_{j,1} < t_{j,2}) \\ \wedge \neg(t_{i,1} > t_{i,2} \wedge t_{j,1} > t_{j,2} \wedge t_{i,1} > t_{j,1}) \\ \wedge \neg(t_{i,1} < t_{i,2} \wedge t_{j,1} < t_{j,2} \wedge t_{i,2} < t_{j,2}) \end{pmatrix}$$

Thus, we can instead compute the Losing Vector $L = (\ell_i)$ as

$$\ell_i = \sum_{j \neq i} \begin{pmatrix} \neg(t_{i,1} > t_{i,2} \wedge t_{j,1} < t_{j,2}) \\ \wedge \neg(t_{i,1} > t_{i,2} \wedge t_{j,1} > t_{j,2} \wedge t_{i,1} > t_{j,1}) \\ \wedge \neg(t_{i,1} < t_{i,2} \wedge t_{j,1} < t_{j,2} \wedge t_{i,2} < t_{j,2}) \end{pmatrix}$$

and test whether $\ell_i = 0$ or not. For this, each authority multiplies it by a secret non-zero value before decryption in order to hide the non-zero values.

5.3 Summary

We reproduce below the full protocol for evaluating Majority Judgment in the encrypted domain (See Algorithm 5). To improve readability, we note $\mathsf{CondGate}(x_i)$ the fact of reducing (x_i) through $\mathsf{CondGate}$ (depth can be reduced by using a binary tree).

To make the protocol secure against malicious adversary, we use cryptographic proofs in the basic gates, as detailed in Subsects. 4.1 and 4.2 and when randomizing ℓ_i. Since all participants will execute the same protocol in parallel, they can reproduce all the other steps and ensure the consistency of the computation. The verifications of these proofs need not be done synchronously, but they must be finished before the final result is decrypted (which actually reveals information).

Remark 1. After candidate $\mathcal{C}_W$ is elected, it is possible to reveal the next candidate by removing the line W of A and repeating the algorithm. This allows for multi-seat elections, but does reveal the order of the elected candidates.

Input: encrypted Aggregate Matrix $(C_{a_{i,j}})$
Output: index of elected candidate
/* Candidate matrix $c_{i,j} = \sum_{k<j} a_{i,k} < \frac{1}{2} \times \sum_k a_{i,k}$ */
$C_{c_{i,j}} \leftarrow \mathsf{GTGate}(\prod_k C_{a_{i,k}}, (\prod_{1 \leq k \leq j} C_{a_{i,k}})^2)$
/* Grade vector $g_j = \wedge_i c_{i,j}$ */
$C_{g_j} \leftarrow \mathsf{CondGate}(C_{c_{i,j}})$
/* Tiebreak matrix left column $t_{i,1} = \sum_j g_j \times a_{i,j}$ */
$C_{t_{i,1}} \leftarrow \prod_j \mathsf{CondGate}(C_{a_{i,j}}, C_{g_j})$
/* Tiebreak matrix right column $t_{i,2} = \sum_j (1 - g_{j-1}) \times a_{i,j}$ */
$C_{t_{i,2}} \leftarrow \prod_j \mathsf{CondGate}(C_{a_{i,j}}, (C_1 \div C_{g_{j-1}}))$
/* Each assignment maps to an inner parenthesis of l_i */
$C_{p^1_{i,j}} \leftarrow \mathsf{CondGate}(\mathsf{GTGate}(C_{t_{i,1}}, C_{t_{i,2}}), \mathsf{GTGate}(C_{t_{j,2}}, C_{t_{j,1}}))$
$C_{p^2_{i,j}} \leftarrow \mathsf{CondGate}(\mathsf{GTGate}(C_{t_{i,1}}, C_{t_{i,2}}), \mathsf{GTGate}(C_{t_{j,1}}, C_{t_{j,2}}), \mathsf{GTGate}(C_{t_{i,1}}, C_{t_{j,1}}))$
$C_{p^3_{i,j}} \leftarrow \mathsf{CondGate}(\mathsf{GTGate}(C_{t_{i,2}}, C_{t_{i,1}}), \mathsf{GTGate}(C_{t_{j,2}}, C_{t_{j,1}}), \mathsf{GTGate}(C_{t_{j,2}}, C_{t_{i,2}}))$
/* Losing Vector $\ell_i = \sum_{j \neq i} \neg p^1_{i,j} \wedge \neg p^2_{i,j} \wedge \neg p^3_{i,j}$ */
$C_{\ell_i} \leftarrow \prod_{j \neq i} \mathsf{CondGate}(C_1 \div C_{p^1_{i,j}}, C_1 \div C_{p^2_{i,j}}, C_1 \div C_{p^3_{i,j}})$
foreach *authority* **do**
  $o \xleftarrow{\$} \mathbb{Z}_n$
  $C_{\ell_i} \leftarrow C^o_{\ell_i}$
end
$\ell_i \leftarrow \mathsf{DecryptGate}(C_{\ell_i})$
return unique i such that $\ell_i = 0$

Algorithm 5: Full Protocol

5.4 Optimizations

Batching. As noted previously when discussing the theoretical aspects of zero-knowledge proofs, it is possible to batch certain operations to reduce the total number of modular exponentiations that are performed in the protocol. To take full advantage of this, we have designed our software implementation to batch operations as much as possible. To be more specific, each gate was implemented in a "batch" version, where it receives a list of inputs to process; instead of processing these inputs sequentially, it can group them as much as possible when calling other "batch" gates. At the lowest level are the proofs of valid decryption and the proofs of private multiplication which actually take advantage of this batching.

Pipelining. The multi-party computation gates presented above target theoretical metrics such as a low number of exchanged messages or a low circuit depth. However, during implementation, more practical considerations must also be taken into account.

For instance, during the main loop of the conditional gate, only one authority is active at any given time, since they must wait for the previous authorities to provide them with the inputs. Since this operation is one of the most frequent

	3 candidates	5 candidates
Up to $2^{10} - 1$ voters	4' 26"	09' 53"
Up to $2^{20} - 1$ voters	8' 53"	19' 05"

Fig. 4. Time to tally the ballots for 3 authorities and 5 possible grades, all run on a single computer with two physical CPU cores (i5-4300U)

ones during the execution of the protocol (almost as frequent as decryption), a straightforward implementation would imply that most of the runtime would be spent waiting for input.

However, we can virtually erase idle CPU time by exploiting pipelining: as shown previously, executions the conditional gate can be batched together. Although it is not possible to batch the multiplication proofs used during this operation, we can exploit this. For this, we remark that the order in which the values circulate among the authorities is of no importance. It only matters that each authority has the opportunity to negate each value. Thus, for α authorities, we can split the batch in α sub-batches, give one sub-batch to each authority, and then have the authorities consider each sub-batch in turn. This approach reduces most of the idle time due to the sequential nature of this operation.

5.5 Benchmarks

We implemented this protocol in Python using the `gmpy2` library (GMP's `powmod` is faster than CPython's `pow`). In contrast to a real-life implementation, we have all communications go through a central point of coordination (but confidentiality and integrity do not rely on it), and we use a simpler secret sharing. Run times are shown in Fig. 4. They should be improved when using several CPU cores for each node, but we are below 20 min for electing one candidate among 5 by more than 1 million of voters! Our implementation can as of now be truly used in a real-world election.

These do not include the encryption (on each voter's computer), verification of the ballots, nor their aggregation into $(C_{a_{i,j}})$ (assumed to be done on-the-fly).

Acknowledgments. This work was supported in part by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud).

References

[APB+04] Aditya, R., Peng, K., Boyd, C., Dawson, E., Lee, B.: Batch verification for equality of discrete logarithms and threshold decryptions. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 494–508. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24852-1_36

[BL07] Balinski, M., Laraki, R.: A theory of measuring, electing, and ranking. Proc. Natl. Acad. Sci. **104**(21), 8720–8725 (2007)

[BL10] Balinski, M., Laraki, R.: Majority Judgment: Measuring Ranking and Electing. MIT Press, Cambridge (2010)

[BMN+09] Benaloh, J., Moran, T., Naish, L., Ramchen, K., Teague, V.: Shufflesum: coercion-resistant verifiable tallying for STV voting. IEEE Trans. Inf. Forensics Secur. **4**(4), 685–698 (2009)

[CP93] Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_7

[DFK+06] Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 285–304. Springer, Heidelberg (2006). https://doi.org/10.1007/11681878_15

[DK05] Desmedt, Y., Kurosawa, K.: Electronic voting: starting over? In: Zhou, J., Lopez, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 329–343. Springer, Heidelberg (2005). https://doi.org/10.1007/11556992_24

[FS87] Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12

[Gir91] Girault, M.: An identity-based identification scheme based on discrete logarithms modulo a composite number. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 481–486. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-46877-3_44

[GPS06] Girault, M., Poupard, G., Stern, J.: On the fly authentication and signature schemes based on groups of unknown order. J. Cryptol. **19**(4), 463–487 (2006)

[Pai99] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_16

[PS96] Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68339-9_33

[PS98] Poupard, G., Stern, J.: Security analysis of a practical "on the fly" authentication and signature generation. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 422–436. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054143

[PS99] Poupard, G., Stern, J.: On the fly signatures based on factoring. In: ACM CCS 1999, pp. 37–45. ACM Press, November 1999

[Sho00] Shoup, V.: Practical threshold signatures. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 207–220. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_15

[ST04] Schoenmakers, B., Tuyls, P.: Practical two-party computation based on the conditional gate. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 119–136. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30539-2_10

[ST06] Schoenmakers, B., Tuyls, P.: Efficient binary conversion for paillier encrypted values. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 522–537. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_31

[TRN08] Teague, V., Ramchen, K., Naish, L.: Coercion-resistant tallying for STV voting. In: 2008 USENIX/ACCURATE Electronic Voting Workshop, EVT 2008, 28–29 July 2008, San Jose, CA, USA, Proceedings (2008)