Solving the Discrete Logarithm Problem for Packing Candidate Preferences

James Heather, Chris Culnane, Steve Schneider, Sriramkrishnan Srinivasan, Zhe Xia

Solving the Discrete Logarithm Problem for Packing Candidate Preferences

James Heather[1], Chris Culnane[1], Steve Schneider[1], Sriramkrishnan Srinivasan and Zhe Xia[2*]

[1] Department of Computing, University of Surrey, Guildford GU2 7XH, U.K.
{j.heather,c.culnane,s.schneider}@surrey.ac.uk
[2] Department of Computing, Wuhan University of Technology, 430063, China
xiazhe@whut.edu.cn

Abstract. Ranked elections are used in many places across the world, and a number of end-to-end verifiable voting systems have been proposed to handle these elections recently. One example is the vVote system designed for the Victorian State Election, Australia. In this system, many voters will give a full ranking of up to 38 candidates. The easiest way to do this is to ask each voter to reorder ciphertexts representing the different candidates, so that the ciphertext ordering represents the candidate ranking. But this requires sending 38 ciphertexts per voter through the mixnets, which will take a long time. In this paper, we explore how to "pack" multiple candidate preferences into a single ciphertext, so that these preferences can be represented in the least number of ciphertexts possible, while maintaining efficient decryption. Both the packing and the unpacking procedure are performed publicly: we still provide 38 ciphertexts, but they are combined appropriately before they enter the mixnets, and after decryption, a meet-in-the-middle algorithm can be used to recover the full candidate preferences despite the discrete logarithm problem.

1 Introduction

Ranked elections are currently used in various elections across the world. For example, they can be found in some local government elections in the US, Scotland, Northern Ireland, New Zealand and Malta. Also, they are used to elect the lower house of parliament in some territories in Australia. Recently, several end-to-end verifiable (e2e) voting systems have been proposed to handle ranked elections. One example is the vVote system [7], [6], which is designed for the Victorian State Election, Australia. In this election, there are around 10 parties and up to 38 candidates. The ballot form consists of two parts: the Above-The-Line (ATL) part lists the parties and the Below-The-Line (BTL) part lists the candidates. The voter can cast her vote using either the ATL part or the BTL part. If a voter chooses to use the ATL part, she simply selects a single party and her vote will be interpreted according to this party's pre-published candidate ranking. If this voter does not agree on any pre-published candidate ranking, she can cast

her vote by expressing a full ranking of the candidates in the BTL part. After all votes are received, the election result will be tallied using the Single Transferable Vote (STV) method. According to the historical data, among the 430,000 voters in that election, about 95% of them will cast ATL votes and the others will cast BTL votes. To design an e2e voting solution for the vVote system, both situations need to be considered. In theory, the ATL votes can be tallied simply using the homomorphic property [9], [10]. However, since the BTL vote contains a full ranking of a very large number of candidates, how to tally these votes in an efficient manner is not so straightforward, and this has been overlooked in many existing schemes.

1.1 Design decisions in the vVote system

The vVote system [7], [6] is not designed as some theoretical concept, but it aims to be used in the real elections in the State of Victoria, Australia. Hence not only security issues but also efficiency and usability issues have been considered. Here, we review some of the design decisions in the vVote system and briefly explain why they have been made.

- **Prêt à Voter style ballot form:** the voters will cast their votes using the Prêt à Voter [8], [21], [20] style ballot form. The main reason for this design decision is that in Prêt à Voter, the vote casting and the ballot auditing are nicely separated, and the ballots can be audited without the vote choices being given. Compared with some other e2e schemes, e.g. the Benaloh scheme [4], Helios [1] and Wombat [24], in which voters need to fill in their choices before deciding whether to audit the ballots or to cast them, the Prêt à Voter approach is more appropriate in this case because the voter will only need to give a full ranking of 38 candidates once.
- **Italian attack:** if the election consists of a large number of candidates, a very large number of possible candidate rankings exist. Adversaries can force voters to cast their votes using specific orderings, and check whether ballots with these unique orderings appear among the cast ballots. This has been referred to as the Italian attack in the literature. Recently, existing techniques [23], [5] have been introduced to solve the Italian attack. However, these techniques are computationally expensive and are not practical to be used in large scale elections. Hence, some compromise needs to be made between security and efficiency: the vVote system has decided not to address the Italian attack.
- **Mixnets vs. homomorphic encryption** in e2e voting schemes, the vote tally phase is normally designed either using mixnets [22], [17], [13] or homomorphic encryption [3], [10], [2]. However, when the election is tallied using the STV method, votes may need to be transferred during the vote tally phase. Thus, each vote has to be kept separated from the other ones. To hide the voter-vote relationships, mixnets are used to shuffle the received votes.

– **ElGamal encryption vs. Paillier encryption** ElGamal [12] has been selected in the system design. The main reason for this design decision is that compared with Paillier [18], ElGamal is not only more computationally efficient but also easier to implement. For example, in order to achieve the 128-bit security level[3], 4096-bit p and 256-bit q are normally used in ElGamal, while in Paillier, the size of n is normally chosen to be 4096 bits. Therefore, in ElGamal and Paillier, the size of exponentiation is 256 bits and 4096 bits respectively, and the size of modulus is 4096 bits and 8192 bits respectively. Therefore, when using ElGamal, the re-encryption computation in the shuffle will be several dozen times quicker than using Paillier. Moreover, the implementation of distributed key generation and threshold decryption in ElGamal is much more straightforward than those in Paillier.

1.2 Our contribution

Based on the above design decisions, the BTL vote will be handled using the Prêt à Voter style ballot form. The "onions" in the ballot form are encrypted using ElGamal encryption. The received votes are shuffled using mixnets in the vote tally phase. However, if the Prêt à Voter scheme is used in ranked elections, the candidate ordering on the ballot form needs to be randomly permuted rather than just be cyclicly shifted. Otherwise, if the adversaries know who is the voter's most/least preferred candidate, they can find out the rest of this voter's ranking just by accessing her receipt. A simple method to design the randomly permuted candidate ordering is to assign a ciphertext next to each candidate. The voter's ranking will be used to reorder these ciphertexts (this is demonstrated in Figure 1). But this means that for each ballot, a 38-ciphertext tuple will be inserted into the mixnets, and after the shuffle, each of the ciphertext needs to be decrypted. Obviously, this will take a long time.

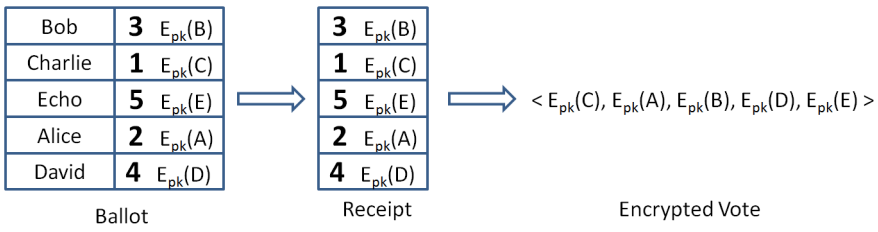

Fig. 1. The ciphertext ranking represents the candidate ordering

[3] Considering that the encrypted votes will be published on the web bulletin board. Some people argue that 128-bit security level is not enough since the votes may need to be protected far into the future. Here, the 128-bit security level is only used as an example.

If multiple candidate references can be packed into one ciphertext before the shuffle, fewer ciphertexts will be sent to the mixnets, and fewer ciphertexts need to be decrypted after the shuffle. To make this idea work, we need to use the additive homomorphic property when packing the ciphertexts. This requires us to use the exponential ElGamal encryption. However, because of the discrete logarithm problem, there does not exist an efficient algorithm to retrieve the voter's rankings after the ciphertext is decrypted.

In this paper, we investigate how to pack the sequence of 38 ciphertexts into the least number of ciphertexts possible, so that they can be mixed and decrypted more efficiently. We also introduce a meet-in-the-middle algorithm that enables the full candidate preferences to be recovered despite the discrete logarithm problem. Note that although our technique is developed using the vVote system as an example, it is applicable to many other ranked elections with a large number of candidates.

1.3 Structure of the paper

In Section 2, we briefly review the meet-in-the-middle techniques, especially the Baby-Step-Giant-Step algorithm. This is followed by describing the system parameters in Section 3. Our method to recover the candidate preferences despite the discrete logarithm problem will be introduced in Section 4. We then provide some discussions in Section 5 before concluding in Section 6.

2 Meet-in-the-middle Review

In the literature, many meet-in-the-middle methods have been introduced. A common property of these methods is that trade-off can be made between time and memory. The benefit is that the search time can be dramatically reduced at the cost of extra storage. For example, because of the meet-in-the-middle attack, double DES is not more secure than the standard DES although its keysize is doubled [11]. Triple DES with two keys is also unable to improve security over the standard DES when considering the chosen plaintext attack [16].

Another famous example of the meet-in-the-middle method is Shank's Baby-Step-Giant-Step (BSGS) algorithm, which is used to solve the discrete logarithm problem. Since it shares some similarities with our proposed technique, we briefly review this algorithm here. In BSGS, the discrete logarithm $x = log_g y$ is represented as follows:

$$x = x_\alpha \cdot \gamma + x_\beta \qquad \text{where } 0 \leq x_\alpha, x_\beta < \gamma$$

The value γ is chosen to be the size of the square root of the group order $\lceil \sqrt{|\mathbb{G}|} \rceil$. This ensures that the value x will span across the entire group. Now, if we denote $T = g^x = g^{x_\alpha \cdot \gamma + x_\beta}$, the above equation can be re-written as follows:

$$T \cdot (g^{-\gamma})^{x_\alpha} = g^{x_\beta}$$

Thus, in order to find $x \in \mathbb{G}$, we need to find a pair (x_α, x_β) that satisfies the above equation. To achieve this, we first build a lookup table that stores all possible mappings from g^{x_β} to x_β, where $0 \leq x_\beta < \gamma$. Note that the size of this table is $\sqrt{|\mathbb{G}|}$. Then, we can try all possible x_α values in the range $0 \leq x_\alpha < \gamma$, until we find a hit in the lookup table that satisfies $T \cdot (g^{-\gamma})^{x_\alpha} = g^{x_\beta}$. If we find such a pair, x can be calculated as $x = x_\alpha \cdot \gamma + x_\beta$.

When using the brute force search to solve the discrete logarithm problem, the computational cost is $|\mathbb{G}|$. By using the Baby-Step-Giant-Step algorithm, the cost can be reduced to $\sqrt{|\mathbb{G}|}$, and the extra cost is to build a lookup table of size $\sqrt{|\mathbb{G}|}$.

3 System Parameters

All these parameters are selected publicly before the election.

Crypto parameters: Let p, q be two large primes such that $q|p-1$. We denote $\mathbb{G}_q$ as the subgroup of $\mathbb{Z}_p^*$ of order q. Let g be a generator of $\mathbb{G}_q$. The public key pk is (p, q, g, y), where $y = g^x \pmod p$ and the secret key x is threshold shared among a number of parties [19], [14]. In order to achieve the 128-bit security level, p and q are suggested to be chosen as 4096 and 256 bits respectively[4].

Candidate parameters: Following the idea in [9], [2], we choose a value M which is larger than the number of candidates. Hence in our case, if there are $n = 38$ candidates, M can be chosen as $n + 1 = 39$. After the candidates are sorted into the canonical order, the first candidate will be assigned value $\mathsf{M}^0 \pmod q$, the second candidate $\mathsf{M}^1 \pmod q$, and so on. If we generalise this, the i-th candidate will be assigned the value $\mathsf{M}^{i-1} \pmod q$.

Encryption The exponential ElGamal cipher [12] will be used for encryption thanks to its additive homomorphic property, i.e. for two messages $m_1, m_2 \in \mathbb{Z}_q$, their ciphertexts can be denoted as $\mathsf{E}_{pk}(m_1) = (g^{m_1} y^{r_1}, g^{r_1}), \mathsf{E}_{pk}(m_2) = (g^{m_2} y^{r_2}, g^{r_2})$ respectively[5], and we have the property that $\mathsf{E}_{pk}(m_1) \cdot \mathsf{E}_{pk}(m_2) = \mathsf{E}_{pk}(m_1 + m_2 \pmod q)$. For the candidates in the canonical order, the ciphertext assigned to the first candidate is $C_1 = \mathsf{E}_{pk}(\mathsf{M}^0)$, the ciphertext assigned to the second candidate is $C_2 = \mathsf{E}_{pk}(\mathsf{M}^1)$, and so on. If we generalise this, the ciphertext assigned for the i-th candidate is $C_i = \mathsf{E}_{pk}(\mathsf{M}^{i-1})$.

4 Ciphertext Packing

To make the explanation clear, we describe the "ciphertext packing" technique step by step, where each step improves its previous step.

[4] For more information about the recommended key length by NIST and ECRYPT II, please refer to http://www.keylength.com/

[5] In this document, we assume all arithmetic to be modulo p where applicable, unless otherwise stated.

4.1 Packing all 38 ciphertexts into 1 ciphertext

Theoretically, it is possible to pack all the 38 ciphertexts as well as the voter's rankings into a single ciphertext. For example, we first sort the ciphertexts of a received vote based on its rankings, and the result can be represented as $\{C_{\pi(1)}, C_{\pi(2)}, \ldots, C_{\pi(38)}\}$, where $\pi(i)$ denotes the i-th candidate in the voter's rankings. Then the set of ciphertexts can be packed as

$$\widehat{C} = \prod_{i=1}^{n} {C_{\pi(i)}}^{i}$$

When the above ciphertext is decrypted, we will get the plaintext as $g^{\sum_{i=1}^{n} i \cdot \mathsf{M}^{\pi(i)-1}}$. However, because of the discrete logarithm problem, there is no efficient method to retrieve $\sum_{i=1}^{n} i \cdot \mathsf{M}^{\pi(i)-1} \pmod{q}$ from the decrypted plaintext. One method is to build a lookup table to store all the possible mappings from g^{ρ} to ρ. In our case, the 38 candidates can be ranked in any order. Thus, there will be $n! = 38! \approx 2^{148}$ possible ρ values. Obviously, it is not feasible to build such a lookup table in practice.

Example 1. *Suppose there are 5 candidates, and the voter's ranking is* $\langle 4, 5, 2, 1, 3 \rangle$ *(so that* $\pi(1) = 4$*,* $\pi(2) = 5$ *and so on). Then the packing becomes*

$$\begin{aligned} & \mathsf{E}_{pk}(\mathsf{M}^3)^1 \cdot \mathsf{E}_{pk}(\mathsf{M}^4)^2 \cdot \mathsf{E}_{pk}(\mathsf{M}^1)^3 \cdot \mathsf{E}_{pk}(\mathsf{M}^0)^4 \cdot \mathsf{E}_{pk}(\mathsf{M}^2)^5 \\ = \quad & \mathsf{E}_{pk}(1 \cdot \mathsf{M}^3 + 2 \cdot \mathsf{M}^4 + 3 \cdot \mathsf{M}^1 + 4 \cdot \mathsf{M}^0 + 5 \cdot \mathsf{M}^2) \end{aligned}$$

4.2 Packing every α ciphertexts into 1 ciphertext

Alternatively, once we have the ciphertext list $\{C_{\pi(1)}, C_{\pi(2)}, \ldots, C_{\pi(38)}\}$ sorted according to the voter's ranking, starting from the first ciphertext, we can make every α ciphertexts as a group

$$\{(C_{\pi(1)}, C_{\pi(2)}, \ldots, C_{\pi(\alpha)}), (C_{\pi(\alpha+1)}, C_{\pi(\alpha+2)}, \ldots, C_{\pi(2\alpha)}), \ldots\}$$

For each group of α ciphertexts, we treat their rankings as values from 1 to α, and we pack these α ciphertexts into one as

$$\widehat{C_j} = \prod_{i=1}^{\alpha} {C_{\pi(j\alpha+i)}}^{i}$$

where $j = 0, 1, \ldots, \lceil \frac{n}{\alpha} \rceil$ - 1. When the ciphertext $\widehat{C_j}$ is decrypted, the plaintext is $g^{\sum_{i=1}^{\alpha} i \cdot \mathsf{M}^{\pi(j\alpha+i)-1}}$. Similarly, we need to build a lookup table to retrieve $\sum_{i=1}^{\alpha} i \cdot \mathsf{M}^{\pi(j\alpha+i)-1} \pmod{q}$ from the decrypted plaintext. In this case, the size of the lookup table is $n!/(n-\alpha)!$ which is smaller than $n!$. Hence by selecting different value α, we can adjust not only the packing ratio (how many ciphertexts to be packed into one) but also the size of the lookup table: to increase the value α, both the packing ratio and the size of the lookup table increase, and

the reverse is true as well. For example, if there are 38 candidates and $\alpha = 6$, all 38 ciphertexts in a ballot can be packed into $\lceil \frac{38}{6} \rceil = 7$ ciphertexts, and the size of the lookup table will be $38!/(38-6)! \approx 2^{31}$. Since the lookup table can be generated in advance (e.g. before the election) and it only needs to be generated once, the construction of such a lookup table is within the computational capacity of modern computers.

Example 2. *Suppose there are 6 candidates and $\alpha = 3$. The voter's ranking is $\langle 4, 5, 2, 1, 3, 6 \rangle$. The packing becomes*

$$\begin{aligned} & \langle (\mathsf{E}_{pk}(\mathsf{M}^3)^1 \cdot \mathsf{E}_{pk}(\mathsf{M}^4)^2 \cdot \mathsf{E}_{pk}(\mathsf{M}^1)^3), \quad (\mathsf{E}_{pk}(\mathsf{M}^0)^1 \cdot \mathsf{E}_{pk}(\mathsf{M}^2)^2 \cdot \mathsf{E}_{pk}(\mathsf{M}^5)^3) \rangle \\ = \quad & \langle \mathsf{E}_{pk}(1 \cdot \mathsf{M}^3 + 2 \cdot \mathsf{M}^4 + 3 \cdot \mathsf{M}^1), \quad \mathsf{E}_{pk}(1 \cdot \mathsf{M}^0 + 2 \cdot \mathsf{M}^2 + 3 \cdot \mathsf{M}^5) \rangle \end{aligned}$$

4.3 Packing every $\alpha + \beta$ ciphertexts into 1 ciphertext

Now, we improve the above packing method a step further: we show how the packing ratio can be increased without increasing the size of the lookup table. Similar to the existing meet-in-the-middle methods, our search method is also a trade-off between time and memory. Suppose the ciphertext list $\{C_{\pi(1)}, C_{\pi(2)}, \ldots, C_{\pi(38)}\}$ has been sorted according to the voter's ranking. Starting from the first ciphertext, we make every $\alpha + \beta$ ciphertexts as a group

$$\{(C_{\pi(1)}, \ldots, C_{\pi(\alpha)}, C_{\pi(\alpha+1)}, \ldots, C_{\pi(\alpha+\beta)}), (C_{\pi(\alpha+\beta+1)}, \ldots, C_{\pi(2\alpha+\beta)}, C_{\pi(2\alpha+\beta+1)}, \ldots, C_{\pi(2\alpha+2\beta)}) \ldots\}$$

For each group of $\alpha + \beta$ ciphertexts, we treat their rankings as values from 1 to $\alpha + \beta$, and we can pack these $\alpha + \beta$ ciphertexts into one ciphertext as

$$\widehat{C_j} = \prod_{i=1}^{\alpha+\beta} C_{\pi(j(\alpha+\beta)+i)}{}^{i} = \prod_{s=1}^{\alpha} C_{\pi(j(\alpha+\beta)+s)}{}^{s} \cdot \prod_{t=\alpha+1}^{\alpha+\beta} C_{\pi(j(\alpha+\beta)+t)}{}^{t}$$

where $j = 0, 1, \ldots, \lceil \frac{n}{\alpha+\beta} \rceil - 1$. When the ciphertext $\widehat{C_j}$ is decrypted, the plaintext is

$$g^{\sum_{i=1}^{\alpha+\beta} i \cdot \mathsf{M}^{\pi(j(\alpha+\beta)+i)-1}} = g^{\sum_{s=1}^{\alpha} s \cdot \mathsf{M}^{\pi(j(\alpha+\beta)+s)-1}} \cdot g^{\sum_{t=\alpha+1}^{\alpha+\beta} t \cdot \mathsf{M}^{\pi(j(\alpha+\beta)+t)-1}}$$

where we have

$$\sum_{s=1}^{\alpha} s \cdot \mathsf{M}^{\pi(j(\alpha+\beta)+s)-1} = \sum_{i=1}^{\alpha+\beta} i \cdot \mathsf{M}^{\pi(j(\alpha+\beta)+i)-1} - \sum_{t=\alpha+1}^{\alpha+\beta} t \cdot \mathsf{M}^{\pi(j(\alpha+\beta)+t)-1} \pmod{q}$$

Now we build up two lookup tables: the α-table stores all the possible mappings from g^ρ to ρ, where ρ is in the form:

$$\rho = \sum_{i=1}^{\alpha} i \cdot \mathsf{M}^{\pi(i)-1} \pmod{q}$$

and the β-table stores all the possible mappings from g^δ to δ, where δ is in the form:

$$\delta = -\sum_{j=\alpha+1}^{\alpha+\beta} j \cdot \mathsf{M}^{\pi(j)-1} \pmod{q}$$

Hence to retrieve the exponent value from the decrypted plaintext

$$m = g^{\sum_{i=1}^{\alpha+\beta} i \cdot \mathsf{M}^{\pi(j(\alpha+\beta)+i)-1}}$$

we can try $m \cdot g^\delta$ for all the possible δ values in the β-table until the result is in the α-table. In this case, suppose the particular values in the α-table and β-table are ρ' and δ' respectively, we will have $m = g^{\rho'-\delta'}$. Hence, $\rho' - \delta' \pmod{q}$ is the desired exponent value of the decrypted plaintext m.

For an election with $n = 38$ candidates, if we use $\alpha = 6$ and $\beta = 4$, we can pack every 10 ciphertexts into one ciphertext. The size of the α-table is $n!/(n-\alpha)! = 38!/(38-6)! \approx 2^{31}$ and the size of the β-table is $n!/(n-\beta)! = 38!/(38-4)! \approx 2^{21}$. And both tables can be generated in advance before the election. After the shuffle, when decrypting a ciphertext and extracting its exponent, we need to try $m \cdot g^\delta$ roughly for half of the possible values in the β-table until the result is in the α-table. Hence we need to repeat the test roughly 2^{20} times.

5 Discussion

5.1 Shrink the α-table

When p and q are chosen as 4096 bit and 256 bit respectively, for both the α-table and β-table, each row consists of a 256-bit value and a 4096-bit value. Hence the data size for a row is 544 bytes, which is roughly 0.5 KB. If $\alpha = 6$ and $\beta = 4$, the α-table contains 2^{31} rows and its total size is roughly 1 TB. The β-table contains 2^{21} rows where its total size is roughly 1 GB.

Note that in the β-table, we will use the g^δ value in the calculation $m \cdot g^\delta$. Thus we have to keep this value intact. However, we can shrink the g^ρ value in the α-table by keeping its last κ bits. The only requirement is that the remaining κ bits of each value is still unique. To check whether $m \cdot g^\delta$ is in the α-table, we only need to check whether its last κ bits are in the α-table. In practice, we can shrink the g^ρ value in the α-table by keeping removing its leading bit when its remaining bits are still unique across the table. Since the size of the α-table is 2^{31}, according to the birthday paradox, if $\kappa = 62$, there is a 50% chance that every remaining value is unique. In this case, the data size for a row is 40 bytes, and the data size for the entire α-table can shrunk to 80 GB.

5.2 Shrink the β-table

Although we mentioned earlier that each value in the β-table has to be kept intact, we can shrink the β-table by reducing the number of rows rather than

reducing the size of each row. This can be achieved by applying the Steinhaus-Johnson-Trotter (SJT) algorithm [15] as follows: denote $\Delta_{i,j,l} = g^{i*\mathsf{M}^l + j*\mathsf{M}^{l+1}}$ for $i, j = -\beta, \ldots, 0, \ldots, \beta$ and $l = 0, 1, \ldots, n-1$. The mappings between (i, j, l) and $\Delta_{i,j,l}$ are stored in the β-table. By multiplying with a single element $\Delta_{i,j,l}$ of this table, we can execute an adjacent transposition in the exponent. Thus, starting from any g^{δ}, we can generate the next possible sequence using a single multiplication. Hence the running time of this approach remains the same, but the number of rows in the β-table has been reduced from $n!/(n-\beta!)$ to $(2\beta)^2 * n$. In case where there are 38 candidates and $\beta = 4$, the number of rows have been reduced from 2^{21} to 2432 which is negligible.

5.3 What if $(\alpha + \beta) \nmid n$?

Previously, we deliberately ignored the case that $(\alpha + \beta) \nmid n$. However, this is an issue we should consider in practice. There are two methods to address this issue: one with padding and one without padding.

Method with padding We can simply append the ciphertext list by repeating the list from the left side until it exactly divides $\alpha + \beta$. For example, suppose the sorted ciphertext list is $\{C_{\pi(1)}, C_{\pi(2)}, \ldots, C_{\pi(38)}\}$, where $\alpha = 6$ and $\beta = 4$. In this case, we treat every 10 ciphertexts as a group and pack them into one ciphertext. If we copy the first two ciphertexts and append them to the end of the list, all the groups will have exactly 10 ciphertexts. After decryption, the repeated candidate preferences can be removed. This method is very simple, and it always works if the number of candidates is larger than $\alpha + \beta$. Next, we introduce another method without using padding.

Method without padding Denote the number of candidates $n = k(\alpha+\beta)+r$ for some integer k. We now discuss the following various situations:

- When $r = \alpha$: after every $(\alpha + \beta)$ ciphertexts have been packed, there will be exactly α ciphertexts remaining, and they will be packed into a single ciphertext. After this packed ciphertext is decrypted, we can use the α-table to retrieve the exponent part of its plaintext.
- When $0 < r < \alpha$: after every $(\alpha+\beta)$ ciphertexts have been packed, there will be less than α ciphertexts remaining, and they will be packed into a single ciphertext. After this packed ciphertext is decrypted, neither the α-table nor the β-table can be used to retrieve the exponent part of the plaintext. To solve this problem, we need to build another lookup table, called α'-table, which stores all the possible mappings from $g^{\rho'}$ to ρ'. And there will be $\sum_{i=1}^{\alpha-1} n!/(n-i)!$ number of possible ρ' values. In case there are 38 candidates and $\alpha = 6$, the α'-table will contain roughly 2^{26} rows, and the size of the α'-table is roughly 2.5 GB. Note that the technique introduced in the previous section can be used to shrink the α'-table.

– When $\alpha < r < \alpha + \beta$: after every $(\alpha + \beta)$ ciphertexts have been packed, there will be more than α ciphertexts remaining, and they will be packed into a single ciphertext. After this packed ciphertext is decrypted, we need to build another lookup table, called β'-table, and use this table along with the α-table to retrieve the exponent part of the plaintext. The β'-table stores all the possible mappings from $g^{\delta'}$ to δ', and there will be $\sum_{i=1}^{\beta-1} n!/(n-i)!$ number of possible δ' values. To retrieve the exponent part of the plaintext m, we test $m \cdot g^{\delta'}$ for all the possible δ' values in the β'-table until the result is in the α-table. In case there are 38 candidates and $\beta = 4$, the β'-table contains roughly 2^{16} rows, and the size of the β'-table is roughly 32 MB. Note that the SJT algorithm introduced above also can be applied here to further reduce the β'-table.

5.4 Constructing the tables

It is possible to construct the tables without requiring a large number of exponentiations. Here, we only informally describe how to build the α-table. The other tables can be built similarly.

Firstly, we build a temporary table of values of the form $g^{j \cdot \mathsf{M}^i}$ where $0 \leq i < n$ and $1 \leq j \leq \alpha$. Then, building the α-table requires $\alpha - 1$ group multiplications from these values. This removes the need for many unnecessary exponentiations. By using a recursive algorithm to build the table, the computational cost can be reduced even further:

1. Set $r = 1$ (the group identity element) and $s = \alpha$ (the packing ratio), the candidate set $\mathsf{C} = \{1, 2, \ldots, n\}$, and the preference set $\mathsf{P} = \{1, 2, \ldots, \alpha\}$.
2. For each candidate $i \in \mathsf{C}$:
   (a) Remove i from C and compute $s = s - 1$.
   (b) For each preference $j \in \mathsf{P}$:
      i. Remove j from P.
      ii. Set $r \leftarrow r \cdot g^{j \cdot \mathsf{M}^{i-1}}$.
      iii. If $s > 0$, recursively run from step 2; otherwise:
         A. Output r.
         B. Restore r and s to values at previous recursive step.
         C. Add j back to P.
         D. Add i back to C.

We have written a program to build the α-table in Java using a standard laptop (Intel i7 processor with 4 cores at 2.7GHz, 8 GB memory, and 64-bit Windows 7). Our assumption is that there are 38 candidates and $\alpha = 6$. Our test shows that the time spent to build the table is just under 10 hours, and it costs slightly more than 3 hours to sort the table (this is a necessary step for binary search). The total size of the table is 100.8GB, and an average search in the table takes 49ms.

5.5 Related work

As described in Section 2, the BSGS algorithm is an important technique to solve the discrete logarithm problem. Our introduced method shares some similarities with BSGS. However, there are also some differences between them. Firstly, BSGS searches the entire group, while our method makes use of the structure of the plaintext and only searches a much smaller subgroup. Thus our method will be quicker when used in unpacking candidate preferences. Secondly, compared with BSGS, our method is more flexible since the sizes of the two lookup tables can be easily adjusted according to different cases.

Packing different votes in the homomorphic fashion was first introduced in [9], and our method follows this approach. Later, it was briefly mentioned in [10] that the meet-in-the-middle trick can be used to solve the discrete logarithm problem if votes are packed homomorphicly using the exponential ElGamal encryption. However, no technical detail was given about how this can be done. Moreover, in both these works, the ciphertext packing technique was only designed for the First-Past-The-Post (FPTP) elections, while ranked elections were not considered. Our work in this paper can be considered as some extension to these two existing works.

6 Conclusion

In this paper, we have explored the details to "pack" multiple candidate preferences into the least number of ciphertext. The benefit is that fewer ciphertexts need to be shuffled and decrypted. After decryption, the full candidate preferences can be retrieved using a meet-in-the-middle algorithm despite the discrete logarithm problem. The vVote system was used as an example, and the parameters were carefully chosen accordingly. But the method present here is generic in nature and it has the potential to be applied in many other ranked elections with a large number of candidates.

7 Acknowledgement

Dr. Sriramkrishnan Srinivasan was at the University of Surrey when this work was carried out. This work was funded by the UK Engineering and Physical Sciences Research Council (EPSRC) under grant EP/G025797/1, and we are grateful to the anonymous reviewers for their valuable comments on the paper.

References

1. Ben Adida. Helios: web-based open-audit voting. *Proceedings of the 17th conference on Security Symposium (SS'08)*, pages 335–348, 2008. Berkeley, CA.
2. Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. *Proceedings of the 20th ACM Symposium on Principles of Distributed Computing (PODC'01)*, pages 274–283, 2001. New York, NY, USA.

3. Josh Benaloh. Secret sharing homomorphisms: keeping shares of a secret secret. *Advances in CRYPTO'86*, pages 251–260, 1986. LNCS 263.
4. Josh Benaloh. Towards simple verifiable elections. *Proceedings of IAVoSS Workshop on Trustworthy Election (WOTE'06)*, pages 61–68, 2006. Cambridge, UK.
5. Josh Benaloh, Tal Moran, Lee Naish, Kim Ramchen, and Vanessa Teague. Shufflesum: coercion-resistant verifiable tallying for STV voting. *IEEE Transactions on Information Forensics and Security*, 4(4):685–698, 2009.
6. Craig Burton, Chris Culnane, James Heather, Thea Peacock, Peter Y. A. Ryan, Steve Schneider, Sriramkrishnan Srinivasan, Vanessa Teague, Roland Wen, and Zhe Xia. A supervised verifiable voting protocol for the Victorian Electoral Commission. *In the 5th International Conference on Electronic Voting (EVOTE 2012)*, 2012.
7. Craig Burton, Chris Culnane, James Heather, Thea Peacock, Peter Y. A. Ryan, Steve Schneider, Sriramkrishnan Srinivasan, Vanessa Teague, Roland Wen, and Zhe Xia. Using Prêt à Voter in the Victorian State elections. *In the 2012 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT 2012)*, 2012.
8. David Chaum, Peter Y. A. Ryan, and Steve A. Schneider. A practical voter-verifiable election scheme. *Proceedings of the 10th European Symposium on Research in Computer Science (ESORICS'05)*, pages 118–139, 2005. LNCS 3679.
9. Ronald Cramer, Matthew Franklin, Berry Schoenmakers, and Moti Yung. Multi-authority secret-ballot elections with linear work. *Advances in EUROCRYPT'96*, pages 72–82, 1996. LNCS 1070.
10. Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. *Advances in EUROCRYPT'97*, pages 103–118, 1997. LNCS 1233.
11. Whitfield Diffie and Martin Hellman. Exhaustive cryptanalysis of the nbs data encryption standard. *Journal of Computer*, 10(6):74–84, 1977.
12. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on IT*, 31(4):467–472, 1985.
13. Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. *Advances in CRYPTO'01*, pages 368–387, 2001. LNCS 2139.
14. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Advances in EUROCRYPT'99*, pages 295–310, 1999. LNCS 1592.
15. Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition.* Addison-Wesley, 1981.
16. Ralph Merkle and Martin Hellman. On the security of multiple encryption. *Communications of the ACM*, 24(7), 1981.
17. C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. *Proceedings of the 8th ACM Conference on Computer and Communications Security (CSS'01)*, pages 116–125, 2001.
18. Pascal Paillier. Public-key cryptosystems based on discrete logarithms residues. *Advances in EUROCRYPT'99*, pages 223–238, 1999. LNCS 1592.
19. Torben P. Pedersen. A threshold cryptosystem without a trusted party. *Advances in EUROCRYPT'91*, pages 522–526, 1991. LNCS 547.
20. Peter Y. A. Ryan, David Bismark, James Heather, Steve Schneider, and Zhe Xia. Prêt à Voter: a Voter-Verifiable Voting System. *In IEEE Transactions on Information Forensics and Security (Special Issue on Electronic Voting)*, 4(4):662–673, 2009.

21. Peter Y. A. Ryan and Steve A. Schneider. Prêt à Voter with re-encryption mixes. *Proceedings of the 11th European Symposium on Research in Computer Science (ESORICS'06)*, pages 313–326, 2006. LNCS 4189.
22. Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme. *Advances in EUROCRYPT'95*, pages 393–403, 1995. LNCS 921.
23. Vanessa Teague, Kim Ramchen, and Lee Naish. Coercion-resistant tallying for STV voting. *2008 USENIX/ACCURATE Electronic Voting Workshop (EVT'08)*, 2008. San Jose, CA, US.
24. Wombat. http://www.wombat-voting.com.