Threat Analysis of A Practical Voting Scheme with Receipts

Sebastien Foulle, Steve Schneider, Jacques Traoré, Zhe Xia

No Institute Given

Abstract. Kutylowski et al. have introduced a voter-verifiable electronic voting scheme "a practical voting scheme with receipts", which provides each voter with a receipt. The voter can use her receipt to check whether her vote has been properly counted in the final tally, but she cannot use the receipt to prove others how she has voted. Another interesting property of this scheme is that, thanks to the repetitive robustness mix network, the ballot tallying phase only needs to be audited if the final results fail to achieve some conditions. However, this paper will show that this scheme is vulnerable to some threats, adversaries can not only violate voter privacy, but also forge the election result.

1 Introduction

Recently, voter-verifiable e-voting schemes have attracted a lot of interest. These schemes guarantee voter privacy and meanwhile achieve end-to-end verifiability. The first prototype was introduced by Chaum in [3], followed by Neff's scheme [10], [9], the Prêt à Voter schemes [4], [12], the Scratch & Vote scheme [2], Punchscan [1], etc.

In voter-verifiable election schemes, voters need to cast their votes in a secure place, e.g. in a voting booth, where the ballot form is either printed on a paper or displayed on the DRE screen. Because the whole ballot form is available to the voter, she knows how she has voted. After that, some part of the ballot form needs to be destroyed and the voter can keep the remaining part as the receipt. The voter can use the receipt to verify that her vote has been correctly counted, but she cannot use the receipt to prove others how she has voted, therefore preventing coercion, intimidation and ballot selling. Furthermore, the accuracy of the final result in these schemes is mainly relying on some hard problems, therefore voters do not need to trust the participants who run the election or equipment used in the election.

The scheme introduced by Kutylowski et al. in [7] is one of these voter-verifiable schemes. But compared to the other schemes, it enjoys two special properties:

- Thanks to the repetitive robustness mix networks, the ballot tallying phase only needs to be audited if the final result fail to achieve some conditions, therefore, Randomised Partial Checking mechanism [5] or Neff's mix [8] can be eliminated.
- The verification process is much simpler, ordinary voters can verify that their votes have been counted without special knowledge. If all votes are properly recorded and tallied, voters will see that some information in their receipts (the identifiers) are correctly displayed on the bulletin board in the final result. Otherwise, their receipts can be used as the proof to make accusation.

In this paper, we will first briefly review the election scheme introduced in [7], then the scheme will be analysed using some threat based system perspectives. We will show that the scheme is vulnerable to a number of attacks. Because of some technical drawbacks or improper implementation, adversaries can not only violate voter privacy, but also forge the election result.

2 The voting scheme

2.1 An overview of the scheme

1. **Ballot construction:** when the voter enters the voting booth, the voting machine will generate two ballots and displays them on the screen, one row per ballot. Each ballot contains the encryptions (*Onions*) of the candidate names and an identifier in a random order. The voting machine commits to the ballots by printing hashes of the whole encrypted data on the so-called *hash ballot*, in the same order.
2. **Ballot casting and ballot checking:** the voter chooses one of the two ballots, and then she selects both the ciphertexts corresponding to her favourite candidate and the identifier from the chosen ballot. At this time, the voting machine prints these ciphertexts on the *voting ballot*. The voter can also ask the voting machine to open the second ballot, by printing a *control ballot* which contains the random data used for the encryptions on the second ballot, and the candidate names as well as the identifier, in the same order as they appear on the screen. This *control ballot* can be checked afterwards by any watch dog organisation. Thus, if a voter checks her *control ballot*, a dishonest voting machine will be caught with probability $\frac{1}{2}$.
3. **Ballot tallying:** After the end of the election day, the votes on the bulletin board are sent to a mixnet which applies several partial decryptions. This optimistic mixnet is making use of signatures embedded in the (plaintext) votes. These signatures from the voting machines prevent dishonest mixes to remove some votes or forge new ones. The result of the tally will be announced under the following two conditions:
   - There are as many tallied votes as votes on the bulletin board (in fact there will be exactly twice as much, and this is also true for the identifiers, see the following point (4));
   - All signatures for the plaintexts are valid.
4. **Voter verification:** Each voter will be convinced that her vote has not been manipulated, and it is in the final tally. Indeed, each *voting ballot* is made up with two ciphertexts of the chosen candidate and two ciphertexts of the voter identifier. After the ballot tallying phase, if no manipulation has been done, each voter can check that her identifier appears exactly twice. Moreover, the four *Onions* of the *voting ballot* cannot be distinguished by others, hence even if a mix server, for instance the first one, is able to modify the four *Onions* of a given ballot, she has only a 1-in-6 chance ($6 = \binom{4}{2}$) to successfully cheat, that is substituting the encryptions of the candidate name while leaving the encrypted identifiers untouched. In the case of a successful manipulation, the voter cannot detect it but the backtracking procedure will find the mix server cheating.
5. **Receipt freeness:** It should be noted that this scheme is receipt-free, since the identifier is not embedded in the encryptions of the candidate name.

2.2 Some technical details

Encryption and decryption Let us briefly describe the encryption scheme. Denote by G a cyclic group of prime order p with hard discrete logarithm problem, and let g be a generator of G. Suppose there are λ mix servers, and each mix server has a secret key x_j, and the corresponding public key is $y_j = g^{x_j}$. An *Onion*, which is an encryption of a message m, is given by the formula $c = (m \cdot (y_1 \cdots y_\lambda)^{k_1}, g^{k_1})$, where k_1 (modulo p) is chosen uniformly at random.

The inputs to the first mix server are couples $(\alpha, \beta) = (m \cdot (y_1 \cdots y_\lambda)^{k_1}, g^{k_1})$ and the outputs are $(\alpha_1, \beta_1) = (\alpha\beta^{-x_1}(y_2 \cdots y_\lambda)^{r_1}, \beta g^{r_1}) = (m \cdot (y_2 \cdots y_\lambda)^{k_2}, g^{k_2})$ with r_1 chosen at random and $k_2 = k_1 + r_1$.

Similarly, the inputs to the ith mix server are $(\alpha_i, \beta_i) = (m \cdot (y_i \cdots y_\lambda)^{k_i}, g^{k_i})$. And in particular, the last mix server receives $(\alpha_\lambda, \beta_\lambda) = (m \cdot {y_\lambda}^{k_\lambda}, g^{k_\lambda})$, and she can recover the original message as $m = \alpha_\lambda {\beta_\lambda}^{-x_\lambda}$.

Note that the first and the last mix server have more power than the others as the first mix server knows the relationships between voters and received votes, and the last mix server can compute all plaintext messages.

The ballot construction Suppose there are two candidates in the election, the Blue Party and the Yellow Party, denoted as B and Y respectively. Once a voter is authorised to enter the voting booth, the voting machine will generate the following *virtual ballot* (it exists only in the voting machine's memory), with couples ordered at random:

$$r \;\; r_U \;\; (B_1{}^U, B_2{}^U) \;\; (Y_1{}^U, Y_2{}^U) \;\; (I_1{}^U, I_2{}^U)$$

$$q \;\; r_L \;\; (Y_1{}^L, Y_2{}^L) \;\; (I_1{}^L, I_2{}^L) \;\; (B_1{}^L, B_2{}^L)$$

The random string r is an identifier, and I is its encrypted ciphertext. The random string q is a seed for all random exponents. The random strings r_X with $X = U, L$ allow to distinguish the votes for a given candidate.

Each voting machine has two key pairs for signature schemes. One private key K' is used for signing the plaintexts using the signature scheme sig'. For each ciphertext $Z_i{}^X$, where $Z = B, Y, I, \;\; i = 1, 2$ and $X = U, L$, after decoding, if $Z_i{}^X$ is an encryption of a vote for candidate B or Y, we will get the plaintext

$$(Z, r_X, ser_V, {sig'}_{K'}(Z, r_X, i))$$

and the decryption of $I_i{}^X$ gives us the identifier

$$(r, ser_V, {sig'}_{K'}(r, i, X))$$

The other private key K is used for creating seeds for the *Onion* construction. $Z_i{}^X$ is an ElGamal ciphertext built with random exponent k_1 as $Z_i{}^X = (m \cdot (y_1 \cdots y_\lambda)^{k_1}, g^{k_1})$, where $k_1 = sig_K(q, i, X, Z)$ and sig is a deterministic signature scheme.

The serial number sev_V of the voting machine allows to know which is the public key to use in order to verify the signature ${sig'}_{K'}$.

After that, the voting machine has to creates and prints a *hash ballot*, which is the commitment to the above *virtual ballot*. Denote σ as the signature with secret key K of

the whole data on the *hash ballot* and h represents a hash function. The voting machine will print the following *hash ballot*:

r	$h(r_U)$	$h({B_1}^U)$	$h({B_2}^U)$	$h({Y_1}^U)$	$h({Y_2}^U)$	$h({I_1}^U)$	$h({I_2}^U)$	$h(q)$
$h(r)$	$h(r_L)$	$h({Y_1}^L)$	$h({Y_2}^L)$	$h({I_1}^L)$	$h({I_2}^L)$	$h({B_1}^L)$	$h({B_2}^L)$	σ

Casting a vote In the voting booth, the voting machine will display these two ballots on the screen in the same order as in the *virtual ballot*:

$$\begin{bmatrix} Blue & Yellow & Identifier \\ Yellow & Identifier & Blue \end{bmatrix}$$

Suppose this voter chooses the upper line on the screen and the Blue Party, the voting machine will print the *voting ballot* which contains two ciphertexts for the Blue Party, two ciphertexts of the identifier r, and the signature of the voting machine.

${B_1}^U$ ${B_2}^U$	${I_1}^U$ ${I_2}^U$	$sig_K({B_1}^U, {B_2}^U, {I_1}^U, {I_2}^U)$

The voter can also asks the voting machine to generate and print a *control ballot* for the ballot in the lower line as

r	Yellow Party		Identifier		Blue Party	
r_L	${Y_1}^L$	${Y_2}^L$	${I_1}^L$	${I_2}^L$	${B_1}^L$	${B_2}^L$
	$\sigma(1, L, Y)$	$\sigma(2, L, Y)$	$\sigma(1, L, I)$	$\sigma(2, L, I)$	$\sigma(1, L, B)$	$\sigma(2, L, B)$

where $\sigma(i, L, Z) = sig_K(q, i, L, Z)$.

In the voting booth, the signature on the *voting ballot* is verified before registration of the vote. Thus no vote on the bulletin board has been forged by dishonest voters.

Finally, the voter leaves the polling place with his three ballots (*voting ballot*, *control ballot* and *hash ballot*) and she can give them to a trusted organisation for verification. The organisation should help this voter to verify the following points:

- The signatures on all these three ballots are valid.
- The *hash ballot* contains the right commitment of the *control ballot*.
- *Onions* on the *control ballot* will be verified without the private key as:

$$ {Z_i}^X = (m \cdot (y_1 \cdots y_\lambda)^{k_1}, g^{k_1}) $$

  where $k_1 = \sigma(i, X, Z)$, and $m = (Z, r_X, ser_V, {sig'}_{K'}(Z, r_X, i))$ if $Z = B, Y$, or $m = (r, ser_V, {sig'}_{K'}(r, i, X))$ if $Z = I$. In the *control ballot*, each party and the identifier should appear exactly once, and their order must be the same as in the *hash ballot*.
- The identifier value must be the same on the *hash ballot* and the opened *control ballot*.

The voter can also look at the bulletin board to be sure that his *voting ballot* has been accurately recorded.

Bulletin board, ballot tallying and backtracking After the last decoding, one has to check the following points:

- All signatures ${sig'}_{K'}({Z_i}^X)$ for the plaintexts are valid.
- The number of votes counted on the bulletin board is exactly half the number of votes $(Z, r_X, ser_V, {sig'}_{K'}(Z, r_X, i))$ and the number of identifiers $(r, ser_V, {sig'}_{K'}(r, i, X))$.
- No vote or identifier has been duplicated.
- Each identifier and each triple (Z, r_X, ser_V) appears twice.

If all these conditions are satisfied, the results are announced. If at least one vote or identifier is invalid, the last mix server must indicate where it comes from, and she has to prove that she has decoded correctly using equality of discrete logarithm proof [14]. This is repeated with the preceding mix server until one finds a mix server unable to prove her correct behaviour. This procedure together with the embedded signatures prevent any manipulation from mix servers such as removing, inserting, duplicating or modifying an *Onion*.

3 Weaknesses of the scheme

There are four minor technical errors in the description of the scheme:

1. The random string q is printed nowhere, although it is absolutely necessary to check the signature on the *control ballot*, since $\sigma(i, X, Z) = sig_K(q, i, X, Z)$. Therefore, it needs to be printed at least on the *control ballot*, it should be printed on the *hash ballot* as well.
2. The authors have not explained how to print the ciphertexts on the *voting ballot*. If they are printed as explained in the previous section, the probability of a successful vote replacement by the first mix server is not $\frac{1}{6}$, but $\frac{1}{2}$, because the first mix server can distinguish the four *Onions* into two groups, although she does not know the content for each group. The correct implementation of this point is that the voting machine has to randomly permute the ciphertexts of candidate and the ciphertexts of identifier before printing them on the *voting ballot*.
3. Even if the voting machine is honest to print the four ciphertexts in a random order on the *voting ballot*, it is still possible for the first mix server to classify the four ciphertexts into two groups if provided with the corresponding *hash ballot*, because the hashed ciphertexts in the *hash ballot* are all grouped, the first mix server can hash the ciphertexts on the *voting ballot* and compare the result with the *hash ballot*. Therefore, in this case, the first mix server still can replace the vote without changing the identifier with $\frac{1}{2}$ possibility.
4. It is not the *hash ballot* that should be marked as used, but the *control ballot* which contains opened *Onions*. We will illustrate this point in the next section.

4 Threat analysis

Some papers [6], [11] have shown that electronic voting schemes are very complex and different attacks may be applied by adversaries. Some of these attacks are because of

technical drawbacks, while others can be caused by improperly implementation of the e-voting systems. In this paper, we will classify our threats to the scheme in [7] into three categories:

- **Threats against anonymity:** these attacks only try to find out how a certain voter has casted her vote, breaking the voter-vote links.
- **Threats against correctness:** the purpose of these attacks is to violate the correctness of the final result. Sometimes, the adversaries cannot forge the result as their wish, but they can make the result random.
- **Attacks against reliability:** these attacks neither wish to learn the voter's choice nor forge the result. Their main purpose is to make the election system break down or violate user's trust to the election system.

4.1 Threats against anonymity

1. **Side channel and subliminal channel attack**
   Since voters cast their votes through a voting machine, the voting machine will know the choice of the voter. An adversaries can use a corrupted voting machine, and some side channel or subliminal channel to learn this voter's choice, e.g. a corrupted voting machine can print ciphertexts such that the space between the first and second letters is larger than the space between the second and third letters when the plaintext is a vote for the Blue Party, otherwise Yellow Party. Hence by looking at the *voting ballot*, the adversaries can learn the choices of the voters.
2. **Kleptographic channel attack**
   The voting machine can carefully choose the random values so that the ciphertext can be read by colluding adversaries without decoding. An example, a corrupted voting machine can generate ciphertexts such that the seventh most significant bit is odd exactly when the plaintext is a vote for the Blue Party, otherwise Yellow Party.
3. **Authority knowledge attack**
   Even if the voting machine is honest in the previous two attacks, it has the power to retrieve voter's choice just by reading the *voting ballot*. The voting machine can implement this attack using two methods. One is to record all relationships between ciphertexts and their plaintexts when generating ballots. The other is to remember all relationships between the identifier r and the seed generator g of this ballot. Then by reading r on the *voting ballot*, the voting machine can find out the plaintext using the signature $sig_K(q, i, X, Z)$.
4. **Duplicating onion attack**
   Suppose now that an honest voter v uses a perfectly honest voting machine to cast her vote. A collusion between a corrupted voter v', the first mix server and the last mix server allows to learn the choice of v.
   Step 1: Before the mixing procedure, the corrupted voter v' casts her encrypted vote c_{aux} (four *Onions*) which is put on the bulletin board, and she sends a valid vote m_{aux} (four opened *Onions*, e.g. taken from her control ballot[1] or from any

[1] The authors have not described any procedure to prevent reusing a control ballot

corrupted voting machine) to the last mix. Note that c_{aux} is not supposed to be an encryption of m_{aux}.
Step 2: The first mix server removes c_{aux} and duplicates (with re-encryption) four *Onions* of voter v.
Step 3: The last mix server opens all *Onions* and recognised the duplicated votes of v. Then she replaces one of the duplicated results by m_{aux}.

5. **Using ElGamal malleability – 1**
   Let $(v_i)_{1\leq i\leq n}$ be the set of all voters. The first mix multiplies the four *Onions* $(c_{i,j})_{1\leq j\leq 4}$ of v_i by i (that is, multiply the first vote by 1, the second vote by 2, and so on). The last mix server decrypts all the votes and for any fixed $k \leq n$, she divides all the obtained results $(i \cdot m_{i,j})$ by k. She will recognise exactly the two votes of v_k (beginning by B or Y), e.g. $(B, r_U, ser_V, sig'_{K'}(B, r_U, 1))$ and $(B, r_U, ser_V, sig'_{K'}(B, r_U, 2))$, and the serial number ser_V allows the last mix server to recover the two identifiers $(r, ser_V, sig'_{K'}(r, 1, U))$ and $(r, ser_V, sig'_{K'}(r, 2, U))$. Hence the last mix correctly outputs all the votes, but she knows the choice of each voter.
6. **Using ElGamal malleability – 2**
   The attack is not as practical as the above one, but it should work in theory. If the first mix server colludes with the last server, they can find out how a voter v has voted as follows: the first mix server replaces all four received *Onions* $(c_i)_{1\leq i\leq 4}$ of v by ${c_i}^2 = ({m_i}^2 \cdot (y_1 \cdots y_\lambda)^{2k_1}, g^{2k_1})$. After decoding, the last mix server will obtain four messages which are not readable. Then m_i can be retrieved by calculating the square root of ${m_i}^2$. Thus, the last mix server can read this voter's choice.
7. **Suicide attack**
   Suppose that the first mix server is the only dishonest one. By using the methods of the previous two attacks, e.g. multiplying the *Onions* and dividing them after the final decryption step, the first mix server learns the choice of each voter. Since the decrypted votes are not valid ones, she will be caught thanks to the backtracking procedure, but it is too late.
8. **Hidden camera attack**
   The authors have mentioned this attack in [7], that if the voting booth is monitored by hidden cameras, or if voters successfully bring cameras into the voting booth, voter privacy can be violated.

4.2 Threats against correctness

1. **Voting machine colludes with the first mix server**
   A corrupted voting machine can always place the ciphertexts for the identifier after the ciphertexts for candidate (or it uses a subliminal channel to indicate the position of ciphertexts for the identifier), and the first mix replaces the first two *Onions* (ciphertexts for candidate) by *Onions* forged by the voting machine.
2. **Voting machine colludes with the last mix server**
   The last mix server can replace some votes (but not identifiers) by valid votes forged by a corrupted voting machine.

3. **Voting machine colludes with any mix server**
   Since the voting machine has the ability to retrieve all identifiers just by reading the ciphertexts (before decoded by the first mix server) on the bulletin board, if it colludes with any mix server, it can generate a whole batch of forged votes (with the same identifiers). And the faulty mix server can use these forged votes to replace the original ones.
4. **Discarded receipt attack**
   When some voters are forced to surrender all their ballots, if the first mix server wish Blue Party to win, she can use the ciphertexts in the *control ballot* (for Blue Party and identifier) to replace the original *Onions*. Besides, if the voting machine colludes with the first mix server, they can forge valid votes to replace the discarded ones.
5. **Cast votes for absent voters**
   If some voters do not cast their votes, adversaries (especially some election authorities) may use their identity to cast votes.
6. **Voting machine generate faulty ballots**
   Because the ballots are generated on demand and they are not distributed to voters randomly. Therefore, if some voters are coerced to cast their votes at some particular time (e.g. between 3pm and 4pm), and these voters are not allowed to audit their *control ballots*, the voting machine can forge ballots during this period without being detected. Another possibility of this attack is similar as introduced in [6], when voters cast their votes in the voting booth, the voting machine has to print the *hash ballot* before voters make their choices. Otherwise, the *hash ballot* may not contain commitment to the *voting ballot*. If some voters do not notice the difference, they may be provided with faulty ballots.

4.3 Threats against reliability

1. **Early publishing**
   Since the voting machine has the ability to retrieve any voter's choice just by reading the bulletin board. It can reveal partial result which may affect voters before they cast their votes.
2. **Invalid signature**
   If the voting machine generate invalid signature on *control ballot* or *hash ballot* (since they will not be checked in the voting booth), it will be difficult to determine afterwards the invalid ballot is forged by the voting machine or by the faulty voter.
3. **Denial of service**
   The authors have not suggested that the private keys $(x_1, x_2, \cdots, x_\lambda)$ are threshold distributed among all mix servers. Hence the absence of at least one mix server will result in denial of service for the whole election system.

5 Conclusion and discussion

We have shown a number of attacks to the scheme in [7]. Generally speaking, the reasons for these attacks include:

– The voting machine has too much power. It generate all ballots not in a distribute fashion and it has the ability to learn voter's choice. The security of this scheme is heavily relying on that the voting machine is honest.
– The mixnet suggested in the scheme is not fully verified. Therefore, some attacks can succeed without being detected. A suggested mitigation to some attacks is that the first and the last mix server have to be verified immediately after their decoding. Note that this cannot completely solve the problem, as the first and the last mix server may collude with the mix servers next to them. To ensure the correctness of the result, the mixnet needs to be fully verified, e.g. using techniques in [13].
– The authors have not described any procedure to prevent reusing the *control ballots*. Thus if they are improperly used, the final result will be inaccurate. We suggest that all opened *control ballots* should be published onto the bulletin board as well, and one has to ensure that no unit in the final result is duplicated from the opened *control ballots*.

References

1. Punchscan. http://www.punchscan.org.
2. B. Adida and R. Rivest. Scratch & Vote: self-contained paper-based cryptographic voting. *Proceedings of the 5th ACM workshop on Privacy in Electronic Society*, pages 29–40, 2006.
3. D. Chaum. Secret ballot receipts: true voter-verifiable elections. *IEEE: Security and Privacy Magazine*, 2(1):38–47, 2004.
4. D. Chaum, P. Ryan, and S. 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.
5. M. Jakobsson, A. Juels, and R. L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. *Proceedings of the 11th USENIX Security Symposium*, pages 339–353, 2002.
6. C. Karlof, N. Sastry, and D. Wagner. Cryptographic voting protocols: a systems perspective. *Proceeding of USENIX Security Symposium*, pages 186–200, 2005. LNCS 3444.
7. M. Klonowski, M. Kutylowski, A. Lauks, and F. Zagorski. A practical voting scheme with receipts. *Proceedings of the 8th Information Security Conference (ISC 2005)*, pages 490–497, 2005. LNCS 2005.
8. C. A. 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.
9. C. A. Neff. Practical high certainly intent verification for encrypted votes. *VoteHere document*, 2004. http://www.votehere.net/vhti/documentation.
10. C. A. Neff. Verifiable mixing (shuffling) of ElGamal pairs. *VoteHere document*, 2004. http://www.votehere.net/vhti/documentation.
11. P. Ryan and T. Peacock. Prêt à Voter: a system perspective. *Technical Report of University of Newcastle*, CS-TR:929, 2005.
12. P. Ryan and S. Schneider. Prêt à Voter with re-encryption mixes. *Proceedings of the 11th European Symposium on Research in Computer Science (ESORICS'06)*, 2006. LNCS.
13. K. Sako and J. Kilian. Receipt-free mix-type voting scheme. *Advances of Eurocrypt'95*, pages 393–403, 1995. LNCS 921.
14. C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, pages 161–174, 1991.