A Practical Voting Scheme with Receipts

Marek Klonowski, Mirosław Kutyłowski, Anna Lauks, and Filip Zagórski*

Institute of Mathematics and Computer Science
Wrocław University of Technology

Abstract. David Chaum introduced Visual Voting scheme in which a voter obtains a paper receipt from a voting machine. This receipt can be used to verify that his vote was counted in the final tally, but cannot be used for vote selling. The Chaum's system requires sophisticated printers and application of randomized partial checking (RPC) method.
We propose a complete design of a voting system that preserves advantages of the Chaum's scheme, but eliminates the use of special printers and RPC.

Keywords: electronic voting, receipt voting, re–encryption, mixnet, anonymity.

1 Introduction

There is a growing interest of electronic voting systems due to high costs, unreliability of counting results and potential frauds during traditional voting procedures. For electronic systems, counting and collecting the results becomes efficient, reliable and require less personal costs. However, there are many questions regarding the goals to be achieved - see a discussion in [11]. Some nontrivial technical problems have to be solved. It must be guaranteed that the technology applied does not open the doors for manipulating the votes, either changing the results. In order to prevent vote selling, the voter should not be able to convince anybody that he voted for a particular candidate. Resilience to vote manipulations may occur at the price of anonymity of voters. Receipts obtained from the voting machines together with information published to exclude vote manipulations may betray the choice of a voter – and enable selling a vote. In turn, measures against vote selling may make it hard to verify election results. For a further discussion concerning this topic and collection of resources on major voting schemes see the Web page of Ronald Rivest [13].
Voter-verifiable Voting Schemes One can regard a voting process as submitting messages $v(x_i)$ to a kind of bulletin board by voters $x_1, ..., x_N$ in such a way that: every x_i can verify if $v(x_i)$ is delivered to the bulletin board, it is infeasible to link x_i with his vote; even if x_i is cooperating, it is infeasible to build a convincing proof that x_i voted in a particular way.

One of the key components in the electronic voting systems is a subsystem mixing the ballots in order to achieve anonymous delivery of messages $v(x_i)$. Usually, networks of mix-servers are used for this purpose. Recall that a mix-server [2] takes a batch of encrypted messages and outputs them after recoding in a random order. The recoding procedure must hide all links between inputs and outputs of the mix-server.

* Contact author – Filip.Zagorski@pwr.wroc.pl

J. Zhou et al. (Eds.): ISC 2005, LNCS 3650, pp. 490–497, 2005.

Chaum [1] presented an idea of voter-verifiable visual voting. This proposal combines visual cryptography and processing through a cascade of mixes in order to ensure anonymity of voters. In this system, a voter has a strong evidence that each vote is really counted, even if he distrusts the infrastructure devoted to voting. Moreover, a voting machine cannot cheat the voter (by showing a picture that differs from the encoded vote). The voter gets a (hard-copy) receipt designed in such a way that it is meaningless for everyone, except the voter. In order to avoid election fraud, during Randomized Partial Checking (RPC) procedure [8] every mix-server must reveal half of the links between its input and output - namely the links starting at the points determined at random by other protocol participants. So if k votes are manipulated by a mix, then it remains undetected with probability $\frac{1}{2^k}$.

Robust Mixing For voting systems it is necessary to ensure robustness of the mixes so that they cannot cheat: replace or manipulate encoded ballots, duplicate them, etc.

There have been many papers on robust mix-networks. One solution is already mentioned, RPC. The second group of solutions, presented i. a. in [12, 3], is based on zero-knowledge proofs. These solutions require a lot of interactions, high communication load and high delay of message delivery.

An interesting idea of *repetitive robustness* was presented by Jakobsson in [6, 7]. The scheme requires higher communication load because of duplication of the input batch. However, due to lack of so-called *local verifiability* it is not well suited for voter-verifiable voting schemes. Moreover, Mitomo and Kurosawa broke this protocol [10]. We use idea of *repetitive robustness*, but our solution is completely different from the solution of Jakobsson.

We achieve the same level of communication overhead as in zero-knowledge proof based protocols, but at a lower computational cost and without any delay. In comparison to RPC, our protocol requires higher communication volume between mixing stages, but the number of stages might be significantly smaller.

Problems Despite many very clever ideas presented in the former schemes, some problems may prohibit their usage.

Systems without receipts have one serious drawback. Although sophisticated protocols ensure proper mixing of the ballots, there is always a possibility that a voting machine does not encode the vote properly.

Other important problems are hardware costs (especially when concering Chaum's system) and provable unlinkability.

New Scheme We design a secure and fairly practical system of voting based on electronic voting machines, in which we combine ideas of Chaum's visual voting, printing method of van de Graaf [15], mixing via re-encryption and a cut-and-choose mechanism that is used to catch cheating parties in a mix-network without proofs on each stage. Our solution has the following important features:

low cost: the whole infrastructure requires low-cost standard devices – scanners or regular bar-code laser readers, and paper printers;

scalability: processing of votes can be parallelized with less problems for anonymity bounds than for Chaum's scheme based on the RPC method.

voter verifiable (locally verifiable) elections: every voter can verify with high probability that his vote is in the final tally and it has not been manipulated,

globally verifiable elections: everone can verify with high probability that none of the votes from the final tally has been manipulated or duplicated.

vote selling: nobody can sell votes without cooperation with a voting machine or all tallying authorities,

flexibility: the scheme works for any number of candidates and write-in elections.

trust model: we do not need to trust any server except for the voting machine regarding anonymity (but not correctness of vote encoding).

The main disadvantage of the scheme presented in this paper is that each vote shows from which voting machine it comes. In some countries it is required by law to count and publish the results by each election commission, so it is even an advantage. If it is not the case, some additional techniques can be applied.

2 Building Blocks

Onions with Recoding We describe now an encoding scheme, called *RE-onion*, which is a simplified version of URE-onions from [9]. An RE-onion shall be used to send a message m through a mix cascade of λ servers; all λ mixes have to process the RE-onion before it is finally decrypted. For $1 \leq j \leq \lambda$, let y_j be the public key of the jth mix, and let x_j be the corresponding private key, that is, $y_j = g^{x_j}$. In this formula g is a generator of a group G with hard discrete logarithm problem. The order of G must be a prime number.

In order to prepare an onion we choose a string k_1 uniformly at random. Then an onion is computed as:

$$(\alpha, \beta) := (m \cdot (y_1 \cdot \ldots \cdot y_\lambda)^{k_1}, g^{k_1}) .$$

When after some decoding and re-encryption it is delivered to mix i, it has the following form

$$(\alpha_i, \beta_i) = (m \cdot (y_i \cdot \ldots \cdot y_\lambda)^{k_i}, g^{k_i}) .$$

Afterwards the onion gets partially decrypted and re-encrypted – the following operations are executed with a randomly chosen r_i:

$$(\alpha_{i+1}, \beta_{i+1}) := (\alpha_i / \beta_i^{x_i} \cdot (y_{i+1} \cdot \ldots \cdot y_\lambda)^{r_i}, \beta_i \cdot g^{r_i}) .$$

It is easy to see that after performing these operations we get for $k_{i+1} = k_i + r_i$:

$$(\alpha_{i+1}, \beta_{i+1}) = (m \cdot (y_{i+1} \cdot \ldots \cdot y_\lambda)^{k_{i+1}}, g^{k_{i+1}}) .$$

Opening an Onion We use a trick borrowed from [1]: when we construct an RE-onion, then we need a random exponent. This exponent is generated by a strong pseudo-random number generator $\mathcal{R}$ from a seed $s(q)$, where $s(q)$ is a signature over q. The signature scheme used is deterministic (like RSA). The string q is also (pseudo)random and is stored together with the RE-onion created. Obviously, it is impossible to recover the exponent used for constructing $(\alpha, \beta) = (m \cdot (y_1 \ldots y_\lambda)^{k_1}, g^{k_1})$ given q only. Indeed, this would require finding $s(q)$ without the signing key. However, when the onion creator would like to show the contents of the onion it suffices to publish $s(q)$. Therefore everybody can reconstruct $\mathcal{R}(s(q))$, derive the exponent k_1, and finally derive $m := \alpha/(y_1 \ldots y_\lambda)^k$ and check whether $\beta = g^k$.

3 Description of the Voting Protocol

The system consists of: *voting machines*, *registration machines*, and *tallying authorities* (each under control of a different party). Additionally, *control servers* are provided by independent watch dog organizations. A voter creates his ballot at a voting machine and brings a printed ballot to a registration machine, where it is scanned. Afterwards the encoded votes are processed through a cascade of mix-servers run by tallying authorities – the goal is to decode and anonymize the ballots.

The ballots are printed for instance as a bar codes by regular printers. A voter can check the printed ballot without any risk of loosing anonymity of his vote.

For the sake of simplicity assume that the voter can choose between two parties - the Blue Party and the Yellow Party (if there are more than two parties, then the number of components in the description is larger, but the rest is essentially the same).

Initialization The public keys of the tallying authorities are published in advance and loaded to the voting machines. No other cryptographic material is read from outside to the voting machine. In the morning of an election day, each voting machine generates two key pairs for signature schemes. One private key is used only for signing the votes and identifiers with a signature scheme sig'. The second key is used for creating seeds for constructing RE-onions. The corresponding public keys are delivered to the local registration machine and to the final tallying authority.

Each voting machine is supervised by an election committee. This committee is also responsible for checking identity of each voter and registering his participation (we skip here the details).

Below we shall consider a voting machine V with a serial number ser_V. Let K and K' denote the private signing keys of V.

Voter in the Voting Booth Assume that a voter is admitted to a voting machine. Then the following steps are executed:

Step 1 In the case of write-in elections the voter may add his candidate by typing the name of the candidate.

Step 2 The voting machine creates a *virtual ballot* – it will never be printed or appear on the screen, it exists in the processor's memory. It consists in the following data: $r, q,$

$r_U,\ \ (B, B_1^U, B_2^U),\ (Y, Y_1^U, Y_2^U),\ (I, I_1^U, I_2^U)$

$r_L,\ \ (B, B_1^L, B_2^L),\ (Y, Y_1^L, Y_2^L),\ (I, I_1^L, I_2^L)$

In fact, the three last components in the second and third row must be permuted at random. Let us describe these data: r is a ballot identifier, which is a random string signed by the voting machine, q is an auxiliary string used for constructing RE-onions, RE-onions B_1^U, B_2^U, Y_1^U, Y_2^U, I_1^U, I_2^U which form so called *upper row*, RE-onions B_1^L, B_2^L, Y_1^L, Y_2^L, I_1^L, I_2^L which form the *lower row*, r_L and r_U are random strings chosen separately for each row. For constructing the RE-onions the voting machine creates signatures $\mathrm{sig}_K(q, i, X, Z)$ for $Z = B, Y, I$, and $X = U, L$, and $i = 1, 2$ (sig is a deterministic signature scheme). The signature $\mathrm{sig}_K(q, i, X, Z)$ is used as a seed by a pseudo-random generator $\mathcal{R}$ to prepare the exponents used in the construction of RE-onion Z_i^X.

For $X \in \{L, U\}$, the onions B_1^X, B_2^X encode a vote for the Blue Party, while the onions Y_1^X, Y_2^X encode a vote for the Yellow Party. The onions I_1^X, I_2^X encode the identifier r. Namely, after full decoding of the onions we get, for $i = 1, 2$ and

$X = L, U$:
- $(B, r_X, \mathrm{ser}_V, \mathrm{sig}'_{K'}(B, r_X, i))$ from B_i^X,
- $(Y, r_X, \mathrm{ser}_V, \mathrm{sig}'_{K'}(Y, r_X, i))$ from Y_i^X,
- $(r, \mathrm{ser}_V, \mathrm{sig}'_{K'}(r, i, X))$ from I_i^X.

Step 3: within this step, the voting machine creates and prints a *hash ballot*, which is its commitment to the virtual ballot. It contains r and a single hash value h_0 described below. Both values are signed by the voting machine.

For computing h_0, a Merkle tree of hashes is constructed. Its leaves are hashes of r, q, r_U, r_L, and of the RE-onions (without identifiers B, Y or I) in the order in which they appear in the virtual ballot. h_0 is the value of the root of the tree. In fact, the tree is unnecessary, if the number of parties is small. Then the hash ballot contains the hashes of the elements listed above.

Step 4: Once the hash ballot is printed a visualization of the virtual ballot appears on the screen – in each row there are pairs of icons depicting the RE-onions corresponding to the votes on particular parties and a pair denoting the RE-onion encoding the identifier r. The ordering of the pairs is the same as in the virtual ballot. Each icon clearly identifies a candidate or a party.

The voter chooses (on a touch-screen or with a mouse) a row and an icon of the party for which he votes in this row. As a result of this action the *voting ballot* is created. It contains a pair of onions corresponding to the icon chosen and a pair encoding the identifier from the same row. Additionally, it contains a signature of the voting machine.

The voting ballot is printed and released to the voter.

Step 5: A control ballot is created (the voter may skip this part, if he wants). For the verification purposes, voter may choose some number of RE-onions from the row that is not used for voting. Afterwardss the following data are printed on the control ballot:
- the RE-onions chosen for verification with their identifiers,
- the signatures necessary for opening these onions,
- the string r_U or r_L – the one from the row chosen for verification,
- hashes necessary to reconstruct the paths from the hashes of RE-onions chosen (from both rows) to the root of the Merkle tree concerned while constructing the hash ballot.

After the control ballot is printed the voter should compare the identifiers on the control ballot with the corresponding positions on the screen before he leaves the voting booth.

Registering the Voting Ballot The voter comes to registration machine and presents its voting ballot. Four RE-onions contained in the ballot are read in and stored for counting purposes, provided that the signatures of the voting machine are valid. Simultaneously, the hash ballot is marked as used.

Verification of a Ballot The voter can control honesty of the voting machine by checking the control ballot and the hash ballot through a machine equipped with a scanner and provided by any watch dog organization.

After reading all data from the control ballot, the hash ballot and the voting ballot the following steps are executed:
- Validity of signatures of the voting machine contained in the ballots are checked.

- It is checked whether the hash values provided on the control ballot reconstruct the paths from hashes of RE-onions concerned to h_0 in the Merkle tree. If yes, then these onions are indeed in the control ballot and at the places declared.

- The signatures of q are checked and using these signatures the RE-onions from the control ballot are opened. Therefore their contents can be verified, since all data concerned are known at this point. Of course, the identifier (B, Y or I) of each RE-onion is checked as well.

Mixing and Counting Procedure When all ballots are registered, counting of the votes may start. Optionally, all vote identifiers r may be published before counting of voices begins.

The first tallying authority processes the RE-onions collected by the registration machines. It partially decodes them, re-codes with random exponents, permutes at random and sends to the second tallying authority. The second authority executes the same steps and sends the result to the third authority. This process is continued until the last tallying authority finishes decoding. For the purpose of a future investigation (which is necessary, if the final output is faulty), each list of RE-onions transferred from one tallying authority to another authority is signed by both authorities and retained safely.

The last tallying authority publishes the list of the strings read from the onions from the final decoding. If every participant behaves according to the protocol, then the list contains:

- pairs encoding an identifier: $(r, \mathrm{ser}_V, \mathrm{sig}'_{K'}(r, 1, X)), (r, \mathrm{ser}_V, \mathrm{sig}'_{K'}(r, 2, X))$
- pairs encoding a single vote:
  $(B, s, \mathrm{ser}_V, \mathrm{sig}'_{K'}(B, s, 1)), (B, s, \mathrm{sig}'_{K'}(B, s, 2))$ or
  $(Y, z, \mathrm{ser}_V, \mathrm{sig}'_{K'}(Y, z, 1)), (Y, z, \mathrm{sig}'_{K'}(Y, z, 2))$ for random strings s, z.

Then each voter can check whether the identifier from his ballot is on the list. If all signatures are valid, the number of votes and the number of identifiers are equal to the number of the voters participating in the elections and there are no duplicates and single halves, the votes are counted and the election result is announced.

Investigation Procedure Let us assume that, after decoding, the last tallying authority gets a string that is neither a valid vote nor a valid identifier (i.e. a signature of a voting machine is invalid or missing). In this case the route of the faulty message m should be traced back in order to find the authority responsible for a manipulation.

First, the last tallying authority presents the ElGamal ciphertext from which it has obtained m, say (a, b). Then it proves that $a/m = b^x$, where x is the private decryption key of this authority. More precisely, a proof of equality of discrete logarithms [14] for pairs $(a/m, b)$ and (y, g) is shown, where y is the public key of the authority.

If the onion presented by the last authority is on the list of onions it has got from the previous authority, it is time for the previous authority to prove its correct behavior. The procedure is the same as in the case of the last authority, except that instead of decryption we consider partial decryption. Additionally, the authority shows which input RE-onion was re-encrypted to obtain the faulty RE-onion. For this purpose, the authority publishes the exponent used for re-encryption. Note that it is not necessary to store all exponents used for re-encryption – they might be derives from a secret key with a strong pseudorandom generator. If this authority proves that it has properly processed the onion containing the faulty m, the next authority must prove its source of m. This

procedure is continued until we come to the point that some authority cannot prove to be not guilty.

The same investigation takes place, if the final list contains duplicates. In this case we trace back each of the onions holding the duplicate message. An authority is found guilty, if it can show only one source of onions that are decoded to the same string.

Improper Behavior Let us consider different possibilities of misbehavior of the mixes:

Removing an onion: in this case the number of onions in the input and in the output of a tallying authority disagree; the fault is immediately discovered, since the number of onions (and votes!) is recorded at each stage.

Inserting a new onion: nobody except the voting machine can prepare an onion that will be correctly decoded. Indeed, the message obtained by the last tallying authority must contain a signature of a voting machine. If the signature is invalid or missing, then the last tallying authority starts an investigation described above. It shows the authority that has injected a new message.

Duplicating an onion: Thanks to re-encryption features, a duplicate can be easily hidden. However, on the final list we get two identical strings then an investigation is started and one of the authorities is found guilty. In order to succeed in cheating, one has to replace a pair of onions which encode votes but not identifiers. Having 4 onions, the probability of a successful replacement is equal to $\frac{1}{6}$ $(= \frac{2}{4} \cdot \frac{1}{3})$; in that case the probability that an onion holding an identifier has been deleted is $\frac{5}{6}$. Therefore, with high probability, some vote identifier is missing from the final list and we may start an investigation that traces the route of an onion holding this identifier. When the authority, which performs a fraud gets $4N$ onions, the probability of duplicating $2k$ onions encoding votes is less than: $1/(2N)^k$.

Manipulating an onion: Since each vote and identifier is accompanied by a signature of a voting machine, nobody can derive a new onion with a valid contents, except for creating an onion with the same contents as another onion (otherwise, we would have a procedure breaking the signature scheme used). If an onion is manipulated, then after the last decoding we get an invalid message. In this case an investigation is started to trace back the route of an onion containing this message.

A hacked or dishonest voting machine: There is a possibility that a mall-ware is running on a voting machine (remember that we use public infrastructure). In this case the ballots might be faulty. However, thanks to the commitment mechanism and the verification procedure the voter can detect irregularities in the ballots with a constant probability.

Dishonest commission: It may happen that a group of corrupted people is supervising a voting machine. They may try to use the machine for casting extra votes by fake voters. Since the registration machines are checking only the signatures of the voting machines, the manipulation is undetected at the first moment. However, finally each decoded vote reveals from which voting machine it came. If the number of votes from this machine does not match with the list of voters that participated in elections the manipulation is revealed. Therefore it is easy to recompute results of voting and repeat the voting procedure only in that commission, where the problems have occurred.

In order to cope with voting for absent voters one can design additional mechanisms. One solution is to provide the voters relatively short password numbers (generated like

PINs) that must be presented before being admitted to a voting booth or to a voting process. The local corrupted commission does not know these passwords in advance, so such a fraud can be easily detected.

Taking pictures: even if prohibited, the voters can take digital cameras into a voting booth and make a film of the voting process in order to provide a proof necessary for selling a vote. However, no digital information is present in the voting booth that can later facilitate a proof. The only problem is that the voter can film what he is doing, but this cannot be excluded by any scheme.

Acknowledgment

Many ideas, improvements and simplifications are thanks to a group of students preparing a test implementation of the scheme presented in this paper.

For further details on the project see web page: e-voting.im.pwr.wroc.pl.

References

1. Chaum, D.: Secret-Ballot Receipts and Transparent Integrity. Better and less-costly electronic voting and polling places.
2. Chaum, D.: Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. Communications of the ACM 24(2), 84-88, 1981.
3. Furukawa, J., Sako, K.: An Efficient Scheme for Proving a Shuffle. Advances in Cryptology-CRYPTO '2001, LNCS 2139, 368-387.
4. Gomułkiewicz, M., Klonowski, M., Kutyłowski, M.: Rapid Mixing and Security of Chaum's Visual Electronic Voting. Computer Security- ESORICS 2003, LNCS 2808, 132-145.
5. Golle, P., Jakobsson, M., Juels, A., Syverson, P.: Universal Re-encryption for Mixnets. CT-RSA '2004, 163-178.
6. Jakobsson, M.: A Practical Mix. Advances in Cryptology- EUROCRYPT '1998, LNCS 1403, 448-461.
7. Jakobsson, M.: Flash Mixing. ACM Symposium on Principles of Distributed Computing '1999, 83-89.
8. Jakobsson, M., Juels, A., Rivest, R.L.: Making Mix Nets Robust For Electronic Voting By Randomized Partial Checking. USENIX Security Symposium '2002, 339-353.
9. Gomułkiewicz, M., Klonowski, M., Kutyłowski, M.: Onion Routing Based On Universal Re–Encryption Immune Against Repetitive Attack. Workshop on Information Security Applications (WISA)'2004, LNCS 3225, 400-410.
10. Mitomo, M., Kurosawa, K.: Attack for Flash MIX. Advances in Cryptology- ASIACRYPT '2000, LNCS 1976, 192-204.
11. McGaley, M.: Report on DIMACS Workshop on Electronic Voting - Theory and Practice. `http://dimacs.rutgers.edu/SpecialYears/2003_CSIP/reports.html`
12. Neff, C.A.: A Verifiable Secret Shuffle and its Application to E-Voting ACM Conference on Computer and Communications Security '2001, 116-125.
13. Rivest, L.R.: Voting Resources Page. `http://theory.lcs.mit.edu/~rivest/voting/`
14. Schnorr, C.P.: Efficient Signature Generation by Smart Cards. Journal of Cryptology 4, 161-174, 1991.
15. Van de Graaf, J.: Adapting Chaum's Voter-Verifiable election scheme to the Brazilian system. `http://www.ppgia.pucpr.br/~maziero/pesquisa/wseg/2004/`