

# Fully Auditable Electronic Secret-Ballot Elections

Berry Schoenmakers

*“What could be easier than counting a bunch of votes!” is a natural thought when one first thinks of the problem of electronic voting. It does not seem very hard either to build an Internet-based system which enables people to vote conveniently from their PCs, WAP phones, or other personal devices. The problem gets much more difficult, however, if one wants to address ballot secrecy in a serious way. The challenge then is to guarantee voters’ privacy and integrity of the election result at the same time. In this paper we show how special-purpose cryptographic protocols help to achieve this goal.*

## Introduction

The apparent simplicity of electronic voting systems is probably the reason for the large number of initiatives in this area. The Internet provides an excellent environment for quick polls, which are easy to implement especially if people need not be prevented from voting more than once. Cookies are sometimes used by web servers to prevent inexperienced users from voting more than once, but this line of defense is easily circumvented: users may simply remove the cookies, or set their browser to use ‘per-session cookies’ only—which is a good idea anyway if one cares about privacy—and so on. Even inexperienced users will quickly realize that they should be able to cast multiple votes by using different computers.

A more serious election will require people to first register as a voter. The goal of the registration process is to give each eligible person a unique identity. Double voting is then prevented simply by accepting at most one vote per identity. Clearly, there must also be a mechanism to authenticate the use of such an identity. For this purpose a PIN code is sometimes used such that a vote is only accepted for a given identity if it comes with the matching PIN

code. Digital signatures can be used as a more advanced form of authentication, as we will see later on in this paper.

For Internet-based elections another simple security measure is to use a *secure* web server for collecting the votes. In that case, the communication between the voter’s client and the web server is protected by a protocol such as SSL (Secure Socket Layer). Once the communication between client and server is protected by SSL, the information necessary for casting a vote (possibly including software such as a Java applet) may be downloaded securely to the client, and the vote and authentication information may be uploaded securely to the server. The use of SSL thus prevents the votes from being read or altered when traveling over the Internet.

Recently, from March 7th to 11th 2000, the Arizona Democratic Party ran such an Internet-based election for its Presidential Preference Primary. The election involved several thousands of online voters, see [2]. The voting system has not been certified by the State of Arizona, however, since the election was internal to the Democratic Party. For such a system to be used in elections for public officials, such as the upcoming U.S. presidential elections in November, the system needs to be certified by the

states where it is used.

A major impediment to government certification of such Internet-voting systems is that these systems fail to satisfy the following two basic requirements:

**Ballot Secrecy** To protect the voters' privacy, only the voter may know which vote he/she cast. Even when the system is audited (or, monitored), all voters should have their privacy protected.

**Auditability** The election results should be verifiable to independent observers (or any interested party, for that matter). This means that it is possible to check unambiguously that the published election result corresponds to the ballots cast by legitimate voters.

We claim that achieving both of these properties at the same time is impossible when using a single server (or rather a single party) to count the votes. Only by using multiple servers we are able to solve this fundamental problem.

	vote	$S$
Alice	yes	1
Bob	no	0
Carol	yes	1
Diane	no	0
total		2

Table 1: One server  $S$  learns *all* the voters' votes.

Let us consider a very simple example to provide some intuition why using multiple servers is useful. First consider the situation described by Table 1. Each voter must choose between 'yes' and 'no', which are encoded as 1 and 0, respectively. Since the server must check whether the voter is allowed to vote, the server learns the identity of the voter. And, since the server must be able to count the votes, the server learns the individual votes as well. Hence, there is no privacy, unless we are willing to trust this single server.

	vote	$S_1$	$S_2$
Alice	yes	-1287	+1288
Bob	no	-1999	+1999
Carol	yes	-1769	+1770
Diane	no	-1334	+1334
total		-6389	+6391

Table 2: Two servers  $S_1$  and  $S_2$  learn *none* of the voters' votes, unless they collude.

Next consider the situation described in Table 2. This time to encode a 'no' vote, the voter picks a random natural number  $x$  of a sufficiently large size and sends  $-x$  to server  $S_1$  and  $x$  to server  $S_2$ . For a 'yes' vote, the voter sends  $-x$  to server  $S_1$  and  $x+1$  to server  $S_2$ . As a result, the individual views of the servers (columns  $S_1$  and  $S_2$  in Table 2) contain no information about the individual votes. However, by adding the totals for the columns we still get  $-6389 + 6391 = 2$ , which is the desired election result. (Of course, it must be prevented that for instance Alice sends  $-1287$  to server  $S_1$  and  $+1289$  to server  $S_2$ , but we are not concerned with that in this example.)

## Bulletin Board Model

The example in the introduction shows the power of using multiple servers, or rather multiple talliers, to distribute the tallying of the votes. The example also shows that if both talliers collude they are still able to find every voter's vote. Therefore, the actual system involves a larger number of talliers such that we may trust at least some of them not to collude. For instance, we could have a representative of each (major) Dutch political party, representatives of organizations such as the *Consumentenbond*, and so on. In this way we achieve a form of *distributed trust*.

The idea of distributed trust is incorporated in the *bulletin board* model, a paradigm for verifiable elections set forth by Benaloh *et al.* (e.g., see [7, 6, 4]). Another important aspect of the model is that the bulletin board is assumed to behave as a broadcast channel such that everybody is able to see what is posted in the bulletin board.

To discuss the bulletin board model we distinguish four types of roles in the election process.

**Officers** operate the bulletin board server itself.

**Voters** cast their votes by sending electronic ballots to the bulletin board.

**Talliers** assist in computing the final tally from all correctly submitted ballots.

**Scrutineers** check whether the final tally corresponds to the correctly submitted ballots.

A person may play several roles during the same election. For example, each member of a nation's parliament may play the role of voter, tallier, and scrutineer at the same time. For large-scale elections, however, there are many more voters than talliers, and the role of scrutineer is probably limited to a small set of observers. Still, any interested party other than the official observers is also able to play the role of scrutineer.

Without going into cryptographic details at this point, we may now outline the steps taken by a voting system in the bulletin board model.

Before an election is conducted, the following steps are executed.

1. The officers do all the necessary preparations for the election:
  - (a) One or several bulletin board servers are set up. In particular, this includes the generation and certification of public keys for these servers.
  - (b) The officers determine the deadlines for the various phases of the election.
  - (c) The officers determine the list of eligible voters.
  - (d) The officers determine the list of talliers.
  - (e) The above information is distributed to all relevant parties.
2. The talliers register their public keys with the bulletin board.
3. Eligible voters register their public keys with the bulletin board.
4. The officers freeze the list of talliers and the list of registered voters.

Here, 'freezing' means to digitally sign and possibly time-stamp the contents. Only voters who appear on the frozen list of registered voters are able to take part in the election.

Once the preparations are done, a typical election proceeds as follows. Clearly, several elections can be held in succession or even in parallel.

1. Registered voters cast their votes by sending in their encrypted ballots. Each voter may cast only one valid ballot. A ballot is valid when it is well-formed and signed with the public key registered with the voter's section on the bulletin

board.

2. The officers freeze the contents of the voters' sections on the bulletin board, and accumulate the valid ballots.
3. The talliers produce their sub-tallies.
4. The officers freeze the contents of the talliers' sections on the bulletin board. The final tally as computed from the sub-tallies is also included.
5. The scrutineers read the complete contents of the bulletin board, and check
  - that all and only valid ballots are counted, and
  - that the valid sub-tallies correspond to the final tally published on the bulletin board.

The tricky part is in the way the scrutineers can do their job without compromising the privacy of the individual votes. Note that, unlike in some other systems, voters are not anonymous when they cast their votes. Instead it is kept secret what they are voting throughout the entire election. Since the voters have to identify themselves as part of the voting protocol it is easy to prevent double voting. Also one knows exactly who voted and who didn't, which is useful in countries like Belgium where people get a penalty if they don't cast a vote.

## Cryptographic Protocols

To achieve privacy and auditability it does not suffice to combine some off-the-shelf digital signatures and (public key) encryption methods. We need to use some special-purpose protocols, which rely on more advanced cryptographic techniques such as verifiable secret sharing and zero-knowledge proofs of knowledge. In this paper we will consider a method based on *homomorphic encryption*, which yields a particularly efficient system. The details can be found in [9].

## Discrete Log Problem

For concreteness, we will consider the discrete log problem for a group  $G_q$  obtained as a subgroup of order  $q$  of  $Z_p^*$ , where  $p$  is a 1024-bit prime and  $q$  is a 160-bit prime satisfying  $q \mid p - 1$ . Simply put, this

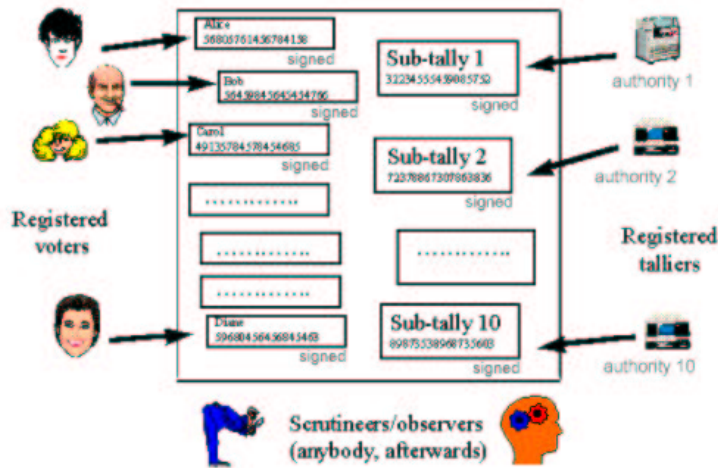


Figure 1: The Bulletin Board

means that  $G_q$  is of the form

$$G_q = \{1, g, g^2, g^3, \dots, g^{q-2}, g^{q-1}\},$$

where  $g$  is an element of order  $q$ , i.e.,  $g^q = 1$ , and multiplication is done modulo the prime  $p$ . By definition we thus have that for each  $h \in G_q$  there exists a unique  $x$ ,  $0 \leq x < q$ , such that  $h = g^x$ . We now call  $x$  the *discrete log* of  $h$  w.r.t.  $g$ , often denoted as  $x = \log_g h$ .

The discrete log problem is to compute  $\log_g h$  for random  $h$  (and fixed  $g$ ), and is assumed to be intractable for the above setting. Note, however, that exponentiation can be done efficiently, that is, given  $x$  we can efficiently compute  $g^x$  modulo  $p$ . For example, to compute  $g^{23}$  we would successively compute  $g, g^2, g^4, g^5, g^{10}, g^{11}, g^{22}, g^{23}$ , where the next value in each step is obtained by either multiplying the previous value with  $g$  or by squaring the previous value, doing all operations modulo  $p$ .

### ElGamal Cryptosystem

The ElGamal cryptosystem is a simple method for public key encryption based on the hardness of the discrete log problem [10]. We assume that  $p$ ,  $q$ , and  $g$ , as introduced in the previous section, are available as system parameters. We then have the following encryption method.

**Key Generation** Each player in the system generates a key pair by picking a random number  $x$ ,

$0 \leq x < q$ , and setting its private key equal to  $x$  and its public key  $h$  equal to

$$h = g^x.$$

Because of the hardness of the discrete log problem, nobody will be able to find  $x$  given only  $h$ .

**Encryption** To encrypt a message  $m$  for a recipient with public key  $h$ , one computes the ciphertext  $(a, b)$  with

$$(a, b) = (g^r, h^r m),$$

where  $r$  is a random number,  $0 \leq r < q$ . Hence, an encryption consists of a pair of numbers. Since the random number  $r$  must be fresh for each message sent, subsequent encryptions will be different even if the same message is sent several times.

**Decryption** To decrypt the pair  $(a, b)$  the recipient will use its private key  $x$  as follows to obtain the message  $m$  again:

$$b/a^x = m.$$

### Homomorphic ElGamal Encryption

A public key encryption algorithm  $E$  is called *homomorphic* if it satisfies the property that

$$E_{PK}(v_1) * E_{PK}(v_2) = E_{PK}(v_1 + v_2),$$

for any public key  $PK$  and any messages  $v_1$  and  $v_2$ . Hence, if we ‘multiply’ two encrypted messages (for the same public key), the result is an encryption of the ‘sum’ of the original messages. (We note that in general it is a bad idea to use a homomorphic encryption algorithm.)

The ElGamal cryptosystem can be easily made homomorphic for encrypting votes  $v \in \{0, 1\}$  by setting  $m = g^v$ . Multiplying  $(a_1, b_1) = (g^{r_1}, h^{r_1} g^{v_1})$  and  $(a_2, b_2) = (g^{r_2}, h^{r_2} g^{v_2})$  we get:

$$\begin{aligned} (a_1, b_1) * (a_2, b_2) &= (a_1 a_2, b_1 b_2) \\ &= (g^{r_1+r_2}, h^{r_1+r_2} g^{v_1+v_2}) \end{aligned}$$

which is an ElGamal encryption of  $v_1 + v_2$ .

## Overview of the Voting Scheme

The two main stages for an election are now as follows:

**Voting** Each voter  $V_i$  submits a ballot of the form  $(a_i, b_i) = (g^{r_i}, h^{r_i} g^{v_i})$ , where  $v_i$  is the desired vote.

**Tallying** The product of all submitted ballots,

$$\prod_i (a_i, b_i) = (g^{\sum_i r_i}, h^{\sum_i r_i} g^{\sum_i v_i})$$

is computed. The talliers jointly decrypt the product, which yields the sum of the votes,  $\sum_i v_i$ , as the desired tally.

We have deliberately omitted some of the (harder) details, which can be found in [9]. Namely, in the voting stage, the voter must also provide a proof of validity that shows that the encrypted vote is indeed in  $\{0, 1\}$  (and not 4 or -123 for example) without revealing any information on the actual value of  $v$ . For this we use an efficient zero-knowledge proof of knowledge. And, in the tallying stage, decryption is done jointly by the talliers in such a way that the talliers can not cheat. For this we use a threshold version of the ElGamal cryptosystem.

A scrutineer is able to verify all of the steps in the protocol.

## Practical Applications

We used a variant of this scheme as the voting engine for the *InternetStem* project, a small-scale ‘shadow election’ held during the Dutch national elections in May 1998. The Seattle-based company VoteHere.net has been using homomorphic techniques in all of their trials since October 1999, including the Alaskan Republican Straw Poll for US President, which is the first binding internet election (see [1]).

In case of *InternetStem*, the voting clients were implemented as Java applets, downloaded through SSL to a browser on a PC. Of course, voting clients running on smart cards, PDAs, mobile phones, and set-top boxes are also possible. Instead of simple yes/no ballots, the system also supports other types of ballots in which the voter must pick one or several candidate out of a list of candidates, or where the voter has to make a choice on a scale of -2 to +2 for instance. For this type of ballots the voting client must be able to do a bunch of modular exponentiations in a reasonable time (say, less than one second), which is no problem on a PC, on a smart card with crypto co-processor, and so on. For large-scale elections, the voting server must also be optimized to do the exponentiations as fast as possible, possibly using dedicated accelerator boards, which are commonly used to speed up web servers handling a lot of SSL traffic.

We are now also negotiating the contract for an EU project called *CyberVote* (IST-1999-20338), which will start in the fall of 2000. The consortium consists of

- MATRA Systèmes & Information (France),
- British Telecom Laboratories (UK),
- NOKIA Research Center (Finland),
- Eindhoven University of Technology (Netherlands),
- Katholieke Universiteit Leuven (Belgium),
- City of Mairie d’Issy-les-Moulineaux (France),
- Free Hanseatic City of Bremen (Germany),
- City of Stockholm and Kista Borough Council (Sweden).

The goal of the project is to build the system around a voting engine based on protocols as considered in this paper.

## Conclusion

As argued in this paper care must be taken in designing electronic voting systems. For paper-based elections it was at least in principle possible (with the help of human observers) to guarantee the voters' privacy *and* the integrity of the tally. In [14] it was pointed out more extensively that paper-based elections may be considered *transparent*, while there is a more or less complete lack of transparency for Internet-based elections.

Therefore, rather than trying to mimic paper-based elections in the digital world, we argue that special-purpose cryptographic protocols need to be employed which solve the fundamental problem of achieving ballot secrecy and auditability at the same time. These protocols may look a bit intimidating to the uninitiated. But as with digital signatures, where one may apply a certain formula to check the validity of a signature, a scrutineer similarly applies a formula to the contents of the bulletin board to verify its validity.

We note that since all the steps to be executed by the bulletin board itself are verifiable, it does not really matter who is actually performing the role of the bulletin board. An important consequence is that the computers and the software doing the bulk of the work need not be trusted, which makes the system much more flexible and scalable. Similarly, as a consequence of using a verifiable voting engine, it becomes much easier to address some of the other problems we have ignored in this paper. For instance, on top of the voting engine one may put different mechanisms for identifying voters, one may use different methods for storing the encrypted ballots in a redundant way (such that elections do not fail if some voting servers crash), one may distribute the computational work over different places, and so on: the particular way these subproblems are handled does not affect the secrecy of the ballots nor the integrity of the final tally.

Naturally we have skipped over many details in this paper. See the references and the references therein

for further reading, some of which are available at <http://www.win.tue.nl/berry/papers.html>.

## References

- [1] <http://VoteHere.net>.
- [2] <http://election.com>.
- [3] Masayuki Abe. Universally verifiable mix-net with verification work independent of the number of mix-servers. In *Advances in Cryptology—EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 437–447, Berlin, 1998. Springer-Verlag.
- [4] J. Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University, Department of Computer Science Department, New Haven, CT, September 1987.
- [5] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections. In *Proc. 26th Symposium on Theory of Computing (STOC '94)*, pages 544–553, New York, 1994. A.C.M.
- [6] J. Benaloh and M. Yung. Distributing the power of a government to enhance the privacy of voters. In *Proc. 5th ACM Symposium on Principles of Distributed Computing (PODC '86)*, pages 52–62, New York, 1986. A.C.M.
- [7] J. Cohen and M. Fischer. A robust and verifiable cryptographically secure election scheme. In *Proc. 26th IEEE Symposium on Foundations of Computer Science (FOCS '85)*, pages 372–382. IEEE Computer Society, 1985.
- [8] R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung. Multi-authority secret ballot elections with linear work. In *Advances in Cryptology—EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 72–83, Berlin, 1996. Springer-Verlag.
- [9] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology—EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 103–118, Berlin, 1997. Springer-Verlag.
- [10] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985.

- [11] K. Sako and J. Kilian. Secure voting using partially compatible homomorphisms. In *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 411–424, Berlin, 1994. Springer-Verlag.
- [12] K. Sako and J. Kilian. Receipt-free mix-type voting scheme—a practical solution to the implementation of a voting booth. In *Advances in Cryptology—EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403, Berlin, 1995. Springer-Verlag.
- [13] B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Advances in Cryptology—CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 148–164, Berlin, 1999. Springer-Verlag.
- [14] B. Schoenmakers. Compensating for a lack of transparency. In *Proceedings of the Tenth Conference on Computers, Freedom & Privacy (CFP2000)*, pages 231–233, New York, 2000. ACM.

## About the Author

Berry Schoenmakers is an Assistant Professor at Eindhoven University of Technology (Netherlands), where he specializes in cryptography. Currently, he is also a member of the Technical Advisory Board of VoteHere.net, a Seattle-based worldwide supplier of highly secure Internet voting solutions. Formerly, he worked with DigiCash (now eCash Technologies) and held a post-doctoral position in David Chaum's crypto group at CWI, the National Research Institute for Mathematics and Computer Science in the Netherlands. He received his MS and PhD degrees in Computing Science from Eindhoven University.

