

Secure Electronic Voting

Michael J. Korman

University of Connecticut

September 8, 2004

- 1 Introduction
 - Motivation
 - Assumptions
 - Election Schemes
- 2 Cryptographic Tools
 - ElGamal Encryption
 - Homomorphic Encryption
 - Distributed Trust
 - Zero-Knowledge Proofs
- 3 Conclusion
 - Summary
 - Further Reading

Motivation

Internet voting is fraught with perils. We need:

- Privacy
- Transparency
- Auditability
- Well-placed trust

Today, we will explore these goals in a cryptographic context.

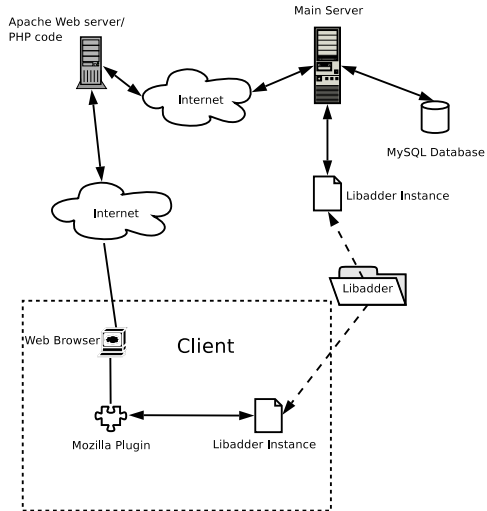
The ADDER System

The purpose of this talk is to explain the cryptography behind the ADDER system.

- Publicly available voting system in progress by Prof. Kiayias, David Walluck, and myself.
- Voters log into the system through the Web.
- They download a browser plugin, and use that to encrypt their votes.
- Votes are submitted, and stored in a public database.
- Authorities log in and submit partial decryptions of the result.
- The server combines the partial decryptions and displays the result.

<http://www.cse.uconn.edu/~adder/>

Diagram



Security Assumptions

We will assume the following about our operating environment:

- There is a set V of voters, and a set A of authorities.
- All participants (voters and authorities) have read/write access to their own private memory.
- There is a world-readable public bulletin board.
- Every user has a designated area on this board with append-only access.

These assumptions raise many security issues; we will only discuss a few in this talk.

Simple Elections

In simple elections, voters have two options, 'yes' and 'no'.

Example

Suppose we have four voters:

Voter	Choice
V_1	0 (no)
V_2	1 (yes)
V_3	1 (yes)
V_4	1 (yes)

When the votes are tabulated, the result is $0 + 1 + 1 + 1 = 3$, which gives us the number of 'yes' votes. This, subtracted from the total number of votes, gives $4 - 3 = 1$, which is the number of 'no' votes.

Multi-way Elections

Multi-way elections require a more complex scheme.

Example

Imagine a three-way election, and four voters:

Voter	Choice
V_1	010 (candidate B)
V_2	100 (candidate C)
V_3	001 (candidate A)
V_4	001 (candidate A)

Tabulation is base- M addition, where M is greater than the number of voters. The result is $010 + 100 + 001 + 001 = 112$. Each position gives us the number of votes for each candidate: 2 for A , 1 for B , and 1 for C . A is the winner.

Privacy

Privacy is a primary concern in electronic voting:

- Voters must have the guarantee that their individual votes will never be revealed.
- It must therefore be possible to tally the votes without ever revealing any of them.
- The secret decryption key of the system can never be applied to a single vote (this will be addressed later).

Preliminaries

We will use the ElGamal cryptosystem. To prepare, at the start of an election, the following events occur:

- 1 The server generates a ℓ -bit safe prime number p . Since p is safe, $q = (p - 1)/2$ is prime, as well.
- 2 The server chooses an element $g \in \mathbb{Z}_q^*$ (\mathbb{Z}_q^* is the multiplicative group of the integers modulo q). It is a fact that $\langle g \rangle$ is the subgroup of quadratic residues modulo p .
- 3 The server chooses $a \in_R \mathbb{Z}_q$ and sets $f = g^a$.

The Discrete Logarithm Problem

The ElGamal cryptosystem operates under the assumption that the following problem is hard:

The Discrete Logarithm Problem

Suppose we have a cyclic group \mathbb{Z}_q , a generator $g \in \mathbb{Z}_q^*$, and a random element $h \in \mathbb{Z}_q^*$. Find the unique integer $a < q$ such that $h = g^a$.

ElGamal Encryption

Suppose Bob wants to send a secret message m to Alice.
Standard ElGamal works as follows:

Algorithm (ElGamal)

- 1 Alice chooses $x \in_R \mathbb{Z}_q$ as her private decryption key, and stores $h = g^x \pmod p$ as her public encryption key. She publicizes h , and keeps x secret.
- 2 To encrypt, Bob picks $r \in_R \mathbb{Z}_q$ and sends

$$\langle g^r \pmod p, h^r m \pmod p \rangle.$$

- 3 To decrypt the message $\langle G, H \rangle$, Alice computes

$$m = H/G^x \pmod p.$$

ElGamal Encryption, cont.

Our encryption is slightly different, but the idea is the same. Suppose Bob wants to encrypt a vote $v \in \{1, M, M^2, \dots, M^{c-1}\}$.

Algorithm (Additive Homomorphic ElGamal)

- 1 Alice chooses $x \in_R \mathbb{Z}_q$ as her private key, and publicizes $h = g^x \pmod p$ as her public key.
- 2 To encrypt, Bob picks $r \in_R \mathbb{Z}_q$ and sends

$$\langle g^r \pmod p, h^r f^v \pmod p \rangle.$$

- 3 To decrypt the message $\langle G, H \rangle$, Alice computes

$$f^v = H/G^x \pmod p.$$

Alice does a brute-force search on v to find the answer.

Homomorphic Encryption

An encryption function that provides these properties is called a *homomorphism*.

Definition

Let \mathcal{R} be a space of random choices, \mathcal{P} the space of plaintexts, and \mathcal{C} the space of ciphertexts. Suppose we have an encryption function $\mathcal{E} : \mathcal{R} \times \mathcal{P} \rightarrow \mathcal{C}$. We say that \mathcal{E} is homomorphic if for all $r_1, r_2 \in \mathcal{R}$, and all $x_1, x_2 \in \mathcal{P}$,

$$\mathcal{E}(r_1, x_2) \odot \mathcal{E}(r_2, x_2) = \mathcal{E}(r_1 \oplus r_2, x_1 + x_2).$$

Homomorphic Encryption, cont.

Fact

“Additive homomorphic ElGamal” is homomorphic.

Proof.

To illustrate, suppose we have two ballots, an encryption of v_1 and an encryption of v_2 . Then,

$$\begin{aligned}\mathcal{E}(r_1, v_1) \odot \mathcal{E}(r_2, v_2) &= \langle g^{r_1}, h^{r_1 f^{v_1}} \rangle \odot \langle g^{r_2}, h^{r_2 f^{v_2}} \rangle \\ &= \langle g^{r_1} g^{r_2}, h^{r_1} h^{r_2 f^{v_1} f^{v_2}} \rangle \\ &= \langle g^{r_1+r_2}, h^{r_1+r_2 f^{v_1+v_2}} \rangle \\ &= \mathcal{E}(r_1 + r_2, v_1 + v_2),\end{aligned}$$

which is the encryption of $v_1 + v_2$. □

Distributed Trust

The secret key of the election must be distributed among the authorities. There are two problems:

- We do not trust any of the authorities, so we cannot hand out a complete copy of the key to each one.
- We do not want to give any authority the power to disrupt the procedure, so we cannot divide the key such that all authorities must be present to use it.

Thus, we need a way to break up the key such that only a certain threshold of shares must be present, while anything fewer than that is completely unusable.

Our Goals

For this scheme to work, we need to create:

- A private key that doesn't really exist (or else someone could access it).
- A public key that is somehow derived from this private key.

Before I explain how this magic actually works, be patient as I present the following protocol.

Key Creation Protocol

Protocol

- 1 Authority i logs in, publishes $h_i = g^{x_i} \pmod{p}$, where $x_i \in_R \mathbb{Z}_q$.
- 2 Authority i logs in, reads h_j for all authorities j , and selects coefficients $a_0^i, a_1^i, \dots, a_{t-1}^i$. It forms a polynomial from these coefficients, evaluates it at each authority's index, and encrypts it using that authority's public key. It publishes this value along with $g^{a_0^i}$.
- 3 Authority j logs in, downloads the values encrypted to it, decrypts them, adds them, and stores the result in s .
- 4 The server publishes $h = \prod_{i \in A} g^{a_0^i}$.

What We Have Accomplished

At this point, we have succeeded in:

- Implying the existence of a “master” polynomial $P(x)$ that is the sum of the polynomials that each authority created.
- Having each authority i store the secret key $P(i)$ in private memory.
- Having the server publish the value $g^{P(0)}$.

We will say that $P(0)$ is the private key of the server, and that $h = g^{P(0)}$ is the public key of the server.

Math Review: Polynomial Interpolation

Suppose there exists a polynomial $P(x)$ of degree $n - 1$. This polynomial can be expressed in two ways:

- 1 **Coefficient form:** $a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}$.
- 2 **Point-value form:** $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$.

We can convert from point-value form to coefficient form (and thus evaluate $P(x)$ at arbitrary points), provided we have n distinct points, using the following formula:

$$P(x) = \sum_{j=1}^n P_j(x), \quad P_j(x) = y_j \prod_{\substack{k=1 \\ k \neq j}}^n \frac{x - x_k}{x_j - x_k}.$$

Secret-sharing with Polynomials

- Say we have a secret S that we wish to distribute, with a threshold of n . We can create a polynomial of degree $n - 1$:

$$P(x) = S + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1},$$

and distribute the values $P(x_1), P(x_2), P(x_3), \dots$

- To reconstruct S , n shares must be combined, and the interpolation formula used to evaluate $P(0)$.
- It is *impossible* to recover the polynomial without n points, so the secret is safe without n shares.
- If a few shares are missing, the operation will still work, as we can use *any* n points.

Partial Decryption

- Once the election is complete, the result is stored in the database in the form $\langle G, H \rangle$.
- Each authority j will log in, download the result, and submit $G^{P(j)}$.
- We will call $G^{P(j)}$ authority j 's "partial decryption."

We're almost at the final result!

Result Decryption

Assume the set of authority identifiers is $\{i_n : 1 \leq n \leq t\}$.

- The server now computes the following coefficients using the Lagrange interpolation formula:

$$\lambda_j = \prod_{\substack{k=1 \\ k \neq j}}^t \frac{-i_k}{i_j - i_k}, \quad 1 \leq j \leq t.$$

- Note that $\sum_{j=1}^t \lambda_j P(j) = P(0)$.
- The server cannot compute $P(0)$ directly, but it can compute $\prod_{j=1}^t G^{\lambda_j P(j)} = G^{P(0)}$.
- Next, the server computes $H/G^{P(0)} = f^{\sum v_i}$.
- It performs a brute-force search of $\sum v_i$ to find the end result.

Ballot-stuffing

Let us return to the simple yes/no election scheme to see a hazard.

Example

Voter	Choice
V_1	0 (no)
V_2	0 (no)
V_3	1 (yes)
V_4	0 (no)
V_5	4 (???)

V_5 has an bogus ballot, but this will never be known, as it is encrypted. When tabulation occurs, there will be 5 'yes' votes, and 0 'no' votes!

Proofs of Knowledge

To solve this problem, we must require the voter to *prove* that the vote is one of the allowed choices. A *proof of knowledge* is a conversation between a prover and a verifier that convinces the prover knows some witness to a predicate without revealing the witness.

Definition

A proof of knowledge is a 4-tuple of probabilistic TMs $\langle P_1, V_1, P_2, V_2 \rangle$. The prover returns $a = P_1(\rho, x)$, where x is the witness and ρ a sequence of coin tosses. The verifier returns $c = V_1(1^\ell)$ as a challenge. The prover returns $s = P_2(a, c, x, \rho)$ as a response. Finally, the verifier accepts iff $V_2(a, c, s) = 1$.

Properties of Proofs of Knowledge

Our proofs of knowledge should satisfy the following three properties:

Completeness: Assuming both prover and verifier follow the protocol, the verifier accepts the proof with probability very close to 1.

Special Soundness: There exists an algorithm \mathcal{X} , called an “extractor,” that given two conversations $C_1 = \langle a, c, s \rangle$ and $C_2 = \langle a, c^*, s^* \rangle$, $\mathcal{X}(C_1, C_2) = x$, such that x is a valid witness.

Honest Verifier Zero-Knowledge: There is a probabilistic TM \mathcal{S} , called a “simulator,” that given c and s , produces output that is indistinguishable from a real accepting conversation.

Example: Schnorr's Identification Protocol

Suppose the prover knows a value x such that $h = g^x$ for public values h, g .

Protocol (Schnorr)

Prover	Verifier
$w \in_R [0, q)$	
$y = g^w \bmod p$	\xrightarrow{y}
	\xleftarrow{c}
	$c \in_R [0, q)$
$s = w + cx \bmod q$	\xrightarrow{s}
	$g^s \stackrel{?}{=} yh^c$

Fiat-Shamir Heuristics

- We can make the protocol non-interactive with the *Fiat-Shamir heuristic*.
- We need a publicly available hash function \mathcal{H} .
- The challenge is created as $\mathcal{H}(T||y)$, where T is the concatenation of all values that have come before, and y is the committed value.
- In Schnorr's protocol with Fiat-Shamir heuristic, the prover would submit

$$\langle y, \mathcal{H}(g, h, y), s \rangle,$$

and the verifier would also check $c \stackrel{?}{=} \mathcal{H}(g, h, g^s h^{-c})$.

Completeness of Schnorr's Protocol

Fact

Schnorr's protocol is complete.

Proof.

Assuming both prover and verifier are honest. Then, with probability 1,

$$\begin{aligned}g^s &= g^{w+cx} \\ &= g^w (g^x)^c \\ &= yh^c.\end{aligned}$$

Thus, the prover always convinces the verifier. □

Soundness of Schnorr's Protocol

Fact

Schnorr's protocol has special soundness.

Proof.

We have two conversations: $\langle y, c, S \rangle$ and $\langle y, c', S' \rangle$. Both are accepting, so $y^S = yh^c$, $y^{S'} = yh^{c'}$, and $c \neq c'$. So,

$$g^S / g^{S'} = h^c / h^{c'} \iff g^{S-S'} = h^{c-c'} \iff h = g^{\frac{S-S'}{c-c'}}.$$

Note that $(c - c')^{-1} \pmod{q}$ exists and can be found. Thus, the witness is $x = \frac{S-S'}{c-c'} \pmod{q}$. □

Zero-knowledge of Schnorr's Protocol

Fact

Schnorr's protocol has the zero-knowledge property.

Proof.

We create a simulator that chooses $c \in_R \mathbb{Z}_q$ and $S \in_R \mathbb{Z}_q$ and outputs $\langle g^S h^{-c}, c, S \rangle$. The following conversations have identical probability distributions:

$$\begin{aligned} \langle g^w, c, w + rc \pmod{q} \rangle & \quad w \in_R \mathbb{Z}_q, c \in_R \mathbb{Z}_q \\ \langle g^S h^{-c}, c, S \rangle & \quad S \in_R \mathbb{Z}_q, c \in_R \mathbb{Z}_q. \end{aligned}$$



Proving Well-Formedness of a Ballot

Suppose we have a ballot $\mathcal{E}(r, 0) = \langle G, H \rangle = \langle g^r, h^r f^0 \rangle$, and the prover wants to prove that $\log_g G = \log_h H$.

Protocol

Prover		Verifier
$t \in_R [0, q)$		
$y = g^t$	$\xrightarrow{y, z}$	
$z = h^t$		
	\xleftarrow{c}	$c \in_R [0, q)$
$s = t + cr \pmod q$	\xrightarrow{s}	$g^s \stackrel{?}{=} yG^c$
		$h^s \stackrel{?}{=} zH^c$

Proving Disjunction of Two Predicates

If we need to prove the OR of two predicates, Q and Q' , the prover will send two proofs, simulating the false one with the simulator \mathcal{S} . Suppose the prover knows a witness for Q :

Protocol

Prover		Verifier
select ρ at random		
$a = P_1(\rho, x)$	$\xrightarrow{a, a'}$	
$\langle a', d', s' \rangle \leftarrow \mathcal{S}(d', s')$		
	\xleftarrow{c}	$c \leftarrow V_1(1^\ell)$
compute d s.t. $c = d \oplus d'$	$\xrightarrow{s, s'}$	$V_2(a, d, s) \stackrel{?}{=} 1$
$s = P_2(\rho, x, a, d)$		$V_2'(a', d', s') \stackrel{?}{=} 1$
		$d \oplus d' \stackrel{?}{=} c$

Proving a Yes/No Ballot

We can now apply this to our proof of well-formedness:

Protocol

Prover		Verifier
$t_0, s_1, c_1 \in_R [0, q]$		
$y_0 = g^{t_0}, z_0 = h^{t_0}$	y_0, z_0, y_1, z_1 \longleftarrow	
$y_1 = g^{s_1} G^{-c_1}, z_1 = h^{s_1} (H/f)^{-c_1}$		
	\xleftarrow{c}	$c \in_R [0, q]$
$c_0 = c - c_1$	s_0, s_1, c_0, c_1 \longrightarrow	$g^{s_0} \stackrel{?}{=} y_0 G^{c_0}$
$s_0 = t_0 + c_0 r$		$h^{s_0} \stackrel{?}{=} z_0 H^{c_0}$
		$g^{s_1} \stackrel{?}{=} y_1 G^{c_1}$
		$h^{s_1} \stackrel{?}{=} z_1 (H/f)^{c_1}$
		$c \stackrel{?}{=} c_0 + c_1$

Non-interactive Proofs

These proofs can be made non-interactive by using the Fiat-Shamir heuristics. The prover would then send

$$\langle y_0, z_0, y_1, z_1, c, s_0, s_1, c_0, c_1 \rangle$$

as his entire proof, where

$$c = \mathcal{H}(g, h, G, H, y_0, z_0, y_1, z_1).$$

To verify, the verifier would check




$$c_0 + c_1 \stackrel{?}{=} \mathcal{H}(g, h, G, H, g^{s_0} G^{-c_0}, h^{s_0} H^{-c_0}, g^{s_1} G^{-c_1}, h^{s_1} (H/f)^{-c_1}).$$

Summary

To summarize, the ADDER system uses the following cryptographic techniques.:

- *Homomorphic encryption* allows votes to be added while encrypted.
- *Distributed trust* uses *threshold cryptography* to distribute the private decryption key among a group of authorities.
- *Zero-knowledge proofs* guarantee that voters do not submit bogus ballots that skew the election results.

For Further Reading

-  Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers
A secure and optimally efficient multi-authority election scheme.
EUROCRYPT '97, LNCS 1233, 1997. Springer Verlag.
-  Adi Shamir
How to share a secret.
Communications of the ACM, 22(11):612-613, 1979.
-  Ronald Cramer, Ivan Damgård, and Berry Schoenmakers
Proofs of partial knowledge and simplified design of witness hiding protocols.
CRYPTO '94, LNCS 1070, 1994. Springer Verlag.