6.897 Special Topics in Cryptography Instructor: Ran Canetti

Lecture 17: Introduction to Electronic Voting

April 8, 2004 *Scribe: Ben Adida*

Electronic Voting: Why?

Of all possible cryptographic applications to study, why choose electronic voting? Both the nature and timing of the electronic voting justify this topic of study.

Timeliness

The topic of electronic voting is particularly timely. 2004 is the year of the first US presidential election since the 2000 Florida recount which placed voting equipment in the spotlight. It is also the first federal election since the Help America Vote Act of 2002 (HAVA) which allocated $3 Billion over 4 years to help States purchase new voting equipment.

Voting technology tends to follow the latest technology trends, with transitions over time from one major technology to another:

$$\text{stones} \rightarrow \text{paper} \rightarrow \text{levers} \rightarrow \text{punch cards} \rightarrow \text{mark-sense} \rightarrow \text{computers}$$

The last transition to computers is underway, but has proven particularly problematic. The problems have become clear over the past couple of years, making this issue even more relevant.

Lastly, there are a number of new, fairly radical proposals for changing the way we vote. David Chaum's new "Secret Voter Receipt" scheme and VoteHere's recently published source code are two of the more interesting ones.

Controversy

The voting debate is relevant because it has recently become quite controversial, specifically on the topic of **DRE** machines (Direct Recording by Electronics), often called touch-screen machines (even though not all of them have touch-screen). The particular questions raised about DRE:

- Does it need a Voter Verifiable Paper Audit Trail (VVPAT)? In case of an audit inconsistency, what is the authoritative copy of the vote: the paper or the electronic record?
- Can we build them securely?
- How can we provide truly private and secure voting for people with disabilities? Should we have one technology for all voters or a diverse set of technologies for various voter needs?

Difficulty

A cursory look at electronic voting might lead one to think that it's a trivial security problem. On the contrary, electronic voting is quite difficult to realize. The security requirements for electronic voting are intrinsically contradictory in ways that haven't been quite resolved yet by any cryptographic means. For example, voters should be able to verify that their vote was cast as intended, but not be able to prove this to anyone else (the system must be "receipt-free").

Another complicating factor is that the main user of this application, the voter, needs to be treated as a potential adversary.

Lastly, in any practical setting for electronic voting, one must consider whether the hardware and basic software platform can be trusted. The line between user and hardware needs to be drawn, whereas most cryptographic protocol definitions tend to ignore the issue by assuming some trusted computing base.

Cryptography Should Be Helpful

All previously mentioned complications "feel" like they should be solvable using cryptography's usual "bag of tricks." Thus, cryptography *should* be helpful. Nevertheless, one must consider that an electronic system needs to be more than just technically secure: any new system must benefit from broad public acceptance by voters and election officials.

Requirements

Election Steps

An election can be decomposed into a number of precise steps:

1. Call an Election (periodic?)
2. Define election parameters: questions / races / candidates
3. Determine voter eligibility requirements
4. Register voters
5. Create/Distribute voter credentials
6. Create/Distribute ballots
7. Prepare equipment and poll workers
8. Verify voter credentials (sign-in)
   Consider provisional ballots and fallback authorization mechanisms.
9. Compose ballot
10. Confirm choices - revise?
11. Cast ballot
12. Collect ballots
13. Tally results
14. Publish results (& voter list)
15. Audit
16. Recount

Security Requirements

At all steps in this process, one must respect a number of accepted voting security requirements:

1. **Democratic**
   - only eligible voters can vote
   - each eligible voter can cast at most one vote (that counts)
2. **Private**
   - No one can tell how a voter actually voted
     (anonymity, at least within large enough cohort/precinct of voters)
   - OK (perhaps even mandatory) to publish who voted (though, obviously not actual ballot content)
3. **Uncoercible**
   - voter cannot be coerced/bribed to vote a particular way
   - voter cannot prove how he voted to another party: receipt-free.
     (Note how this requirement assumes the voter may be an adversary).
4. **Accurate**
   The final tally is the correct sum of cast votes.
   - cast ballots can't be altered, deleted, substituted.
   - all cast ballots are counted; other (invalid) ballots can't be added.
5. **Verifiable**
   Stalin: "He who votes determines nothing; he who counts the votes determines everything." Thus, we need verifiability of the vote counting process.
   - individual verifiability: each voter may verify her vote
     (Note: do you need to reveal your vote in the clear as recourse?)
   - representative verifiability: each voter may delegate to a party or other representative the task of verifying the vote (without revealing the vote in the clear, of course).
   - universal verifiability: anyone can verify total.
6. **Robust**
   Small group can't disrupt election (DOS attacks, complaint procedures, etc...)
7. **Fairness**
   No partial results are known before the election is closed.

Other Requirements

Beyond security, one should also consider further system requirements:

1. **Ease-of-Use/Convenience**:
   good User Interface, efficient voting process, and accessibility from "anywhere."
2. **Flexibility**:
   write-ins, preferential voting

Conflicts!

We note clear conflicts in the above requirements:

- individual verifiability vs. uncoercibility
- convenience vs. uncoercibility

Existing Schemes

A number of existing voting schemes are in use today. These schemes each have advantages and disadvantages. With respect to the security goals, here's how they stack up. (This chart is very rough, and its entries debatable.)

Honest Intentions	Cast as Intended	Counted as Cast	Verifiable as Counted	
Y	Y	Y	Y	Hand-Counted Paper Ballots
N?	Y	Y?	Y	Absentee / Vote-By-Mail
Y	N?	N?	N?	Lever Machines
Y	Y	Y/N?	Y	Optical Scan
Y	N?	N?	N	DRE (touch-screen)
N?	N?	N?	N	Internet / remote

The current systems raise a number of points and issues:

- a private polling place is still a strong plus as far as security requirements go
- the difference between performance of systems on "cast as intended" and "verifiable as counted" can largely be traced to the presence of a voter-verifiable paper audit trail as an official ballot.
- do we *need* to trust the voting equipment?
- there's a strong distinction between:
  - direct creation/verification of the official ballot
  - indirect creation/verification of the official ballot

  Contrary to its name, a DRE system is very much indirect, much like a blind voter who must bring a trusted friend to vote for him.

¿From these issues arose the Mercuri Proposal [**?**], where voting happens at polling stations with equipment that prints out an official paper ballot which the voter confirms and casts. The electronic equipment serves only in the production of this paper ballot.

Class Schedule

The goal of the next few classes on voting is the review of existing voting methods that involve cryptography including [**?**, **?**, **?**].

Voting via Mix-Nets

The Mix-Net Paradigm

Figure 1: A Mix-Net: Ballots B_i are Randomly Permuted

The use of mix-nets for voting is initially due to Chaum [**?**], and has since been used in many other schemes [**?**, **?**].

The process is diagrammed in figure 1 and works as follows:

1. n voters create n ballots, $B_1, B_2, \ldots B_n$
2. each voter encrypts his ballot, yielding the 0-th level ciphertexts: $C_{1,0}, C_{2,0}, \ldots C_{n,0}$.
3. We have t mixes, or mix servers, $S_1, S_2, \ldots S_t$.
4. i-th mix S_i takes in $\langle C_{1,i-1}, C_{2,i-1}, \ldots C_{n,i-1}\rangle$, secretly permutes their order and either:
   - reencrypts (reencryption mix), or
   - decrypts (decryption or Chaumian mix),

   to obtain $\langle C_{1,i}, C_{2,i}, \ldots C_{n,i}\rangle$
5. the final ciphertext sequence $\langle C_{1,t}, C_{2,t}, \ldots C_{n,t}\rangle$ may need one round of decryption (for a reencryption network).
6. all outputs of all mixnets are published on a bulletin board and universally readable, including the final tally of cleartext ballots.

Issues

Trusting Initial Encryption We define $C_{i,0} = Enc(B_i)$, and we want a Proof of Knowledge of B_i. This is likely not voter-verifiable because the encryption process is performed by a machine, and a human can't check that $C_{i,0}$, the cast vote, is indeed the encryption of B_i, the voter's intention.

We want to be able to trust the hardware without providing a voter receipt.

The solutions to this problem are much more eccentric than the typical cryptographic techniques: we will study them later in the course.

Trusting Server Operation Can we trust the servers to mix properly? One honest S_j can protect privacy, but one dishonest $S_{j'}$ can replace, alter, duplicate votes, even if all ciphertexts are published.

The servers might also collude to violate voter privacy.

We have solutions for this problem with verifiable mixnets.

Math of Mix-Nets

We have two types of mix-nets: decryption networks and reencryption networks.

Decryption Networks In a decryption network, each mix server peels off a layer of an onion of multiple encryptions of a plaintext. By the end, all layers have been peeled off and the plaintext is available.

More specifically, each server S_j has (PK_j, SK_j) for a randomized public-key encryption scheme (with semantic security). Each ciphertext is:

$$C_{i,0} = E(PK_1, E(PK_2, \ldots, E(PK_t, B_i)\ldots))$$

With such a scheme, all ciphertexts at a given layer must have the same size, otherwise they are easily traceable across that given mix server. Randomization (necessary for semantic security) means ciphertext length grows with t (the randomization is discarded after every decryption).

Reencryption Networks Most reencryption networks use a variant of ElGamal encryption and reencryption. In ElGamal encryption, we consider the usual group of integers $\mathbb{Z}_p^*$ under multiplication and g, a generator of $\mathbb{Z}_p^*$.

$x \in \mathbb{Z}_p^*$ is the secret key.

$y = g^x$ is the public key.

$E(m) = (g^r, my^r)$ is the randomized encryption function with $r \xleftarrow{R} \mathbb{Z}_{p-1}^*$.

$D(c_1, c_2) = (c_1^x)^{-1} * c_2$ is the decryption function on an ElGamal pair.
(Note how the randomized factor is divided out: as long as the pair is a correct ElGamal pair, decryption is straight-forward).

We now describe ElGamal reencryption (i.e. re-randomization):

$ReEnc(c_1, c_2) = (c_1 * g^s, c_2 * y^s) = (g^{(r+s)}, my^{(r+s)})$ for $s \xleftarrow{R} \mathbb{Z}_{p-1}^*$.

The reencryption does not affect the decryption process, nor does it require knowledge of the secret key!

Verifiable Mix-Nets

We will study two approaches to making mix-nets verifiable:

1. Randomized Partial Checking [**?**]
2. Proof-of-Subproduct [**?**]