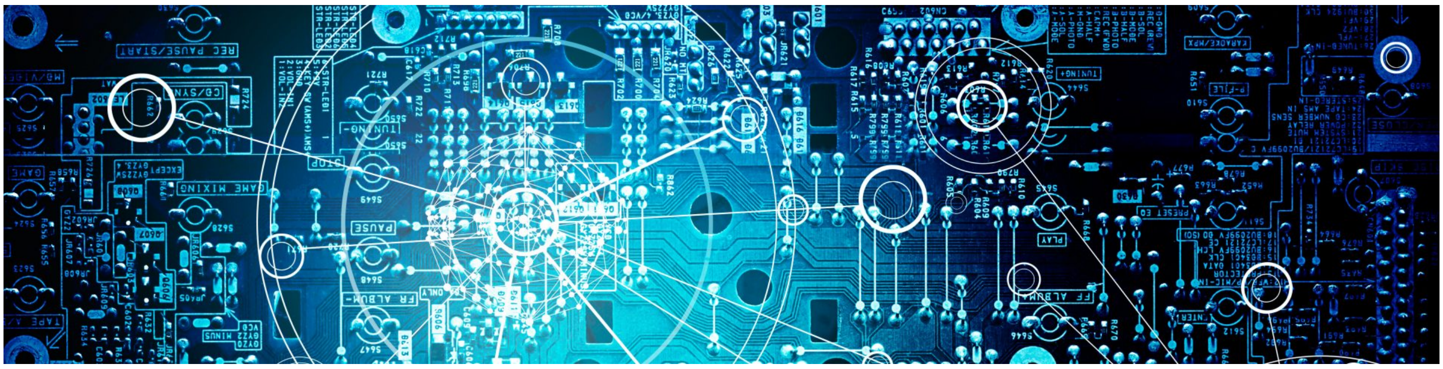

E-VOTING CRYPTO PROTOCOLS

November 5, 2018 JP Aumasson Crypto, Data privacy Leave a comment

"It's not the voting that's democracy; it's the counting."
—Tom Stoppard

For cryptography researchers, e-voting isn't about voting machine or online voting, but is a field of research in its own right. E-voting research is about designing e-voting *protocols*, the core mathematical components of end-to-end auditable voting systems, or voting systems where independent auditors and voters can *securely* verify that a vote's outcome is correct. These voting systems aren't just research papers but real technology that's been used for actual elections: the city of Takoma Park, Maryland trusted Scantegrity II's system based on paper ballots with invisible ink, while cryptographers themselves used the Helios online voting system to elect their leaders.

E-voting is a super complex topic, so in this post I'll restrict myself to the key notions: what it means to securely verify a vote, how votes can be counted without looking at each individual vote, and what stops voters from cheating. I'm not going to give you a full-blow description of an e-voting protocol and all its intricacies, but if that's what you like, papers are plenty.

Secure Verification

What do we expect from a secure voting system?

First, the most obvious: You want to verify that ballots are counted as cast, meaning that everyone should be able to verify the final tally is the correct count of the ballots cast by the voters. Verification shouldn't reveal more information than the final count. In particular, a verifier should have no way to guess who voted what. That's the equivalent of manually counting paper ballots in old-fashioned voting.

Second, you want any voter to be able to verify that *their* vote was counted, and represents their voting intention. This should be done without revealing the actual vote, and more generally a voter shouldn't be able to prove how they voted. That's in order to prevent *coercion*, and let voters freely choose without fearing for the consequence of their choice.

Finally, a voting system should defend against fraud: a voter shouldn’t be able to cast more than one ballot, and it should be impossible to modify or replicate a ballot. Furthermore, non-registered parties shouldn’t be able to vote.

To recap, we want public verifiability, voter confidence, coercion resistance, and election integrity. These are the principles put forward in the foundational research by Chaum, Neff, and others in the early 2000s.

General Principle

Most of the classic e-voting protocols work like this:

1. Voters receive a token, in the form of a ballot that they will modify according to their voting choice. Different voters get different ballots.
2. The voter encrypts their ballot (using a special type of encryption that allows for the e-voting magic) and casts it, so that the voting officials received the encrypted ballot.
3. The voting officials post encrypted ballots on a *bulletin board*, a “public broadcast channel with memory” in cryptographers’ jargon, or a kind of Pastebin, in simpler terms.
4. Voting officials combine the encrypted ballots in order to get an *encrypted tally*. Then they decrypt it (but never decrypt the actual ballots!) and publish the result.
5. Given the result and the encrypted votes, anyone can verify that the result is correct.

Secure Tallying: Homomorphic Encryption

In step 4 above, election officials combine ciphertexts in order to build another ciphertext that encrypts the sum of all individual votes. To do this, e-voting schemes use an encryption scheme **Enc**() for which you can get **Enc**(v1 + v2) given only **Enc**(v1), **Enc**(v2), and without knowing the decryption key. The encryption schemes that can do this are called *homomorphic*.

For example, and simplifying a lot, US voters may do the following on November 8th:

1. Receive a "Clinton" ballot and a "Trump" ballot from the election officials (for the sake of simplicity, I'm only considering the two main candidates).
2. Write **Enc**(1) on one ballot, and write **Enc**(0) on the other, using as a key the public key provided by election officials

The encrypted ballots are then posted on a bulletin board, along with the ID of the voter. Everyone knows who has voted, but it's impossible to find out for whom because every **Enc**(0) and every **Enc**(1) are unique, since we're using strong, *randomized* encryption. If encryption were deterministic, a voter could be coerced to reveal their vote by computing **Enc**(0) again and comparing it with the value on the bulletin board.

Finally, the election officials combine all the "Clinton" ballots and get as a result **Enc**(number of Clinton votes), and to the same with "Trump" ballots in order to get **Enc**(number of Trump votes). They then get the decryption key and decrypt those two values and announce the winner.

But how do we find a homomorphic encryption scheme? Actually it's easy, encryption schemes like RSA and ElGamal are homomorphic in their basic versions, for they satisfy

Enc(v1) × **Enc**(v2) = **Enc**(v1 × v2)

But that's not exactly what we want, these are multiplicatively homomorphic whereas we need *additive* homomorphism. There are tricks to make RSA and ElGamal additively homomorphic, but instead I'll show a lesser-known scheme that is directly additively homomorphic: Paillier's encryption, which encrypts a message v1 to

Enc(v1) = g^{v1} $r1^n$ mod n^2

Where n = pq and g are fixed parameters and r1 is a random integer in]1; n^2[. We thus have

Enc(v1) × **Enc**(v2) = (g^{v1} $r1^n$) × (g^{v2} $r2^n$) mod n^2 = g^{v1+v2} $(r1r2)^n$ mod n^2 = **Enc**(v1 + v2)

That is, we can use Paillier's scheme for tallying encrypted votes.

Preventing Fraud: Zero-Knowledge Proofs

To cheat, a voter might choose to write **Enc**(10000) instead of **Enc**(1) in their ballot to give more votes to a candidate. If they're really malicious, they may write **Enc**(some large number) in order to trigger an integer overflow and crash the voting system. So how can one ensure that the vote is a legitimate vote (0 or 1) without decrypting it?

The solution is called *NIZK*, for non-interactive zero-knowledge. An NIZK proof is a rather complex and incredibly powerful cryptographic object: in our case it allows a voter to prove that their ciphertext is either an encryption of 0 or of 1, but without leaking any information on the encrypted value. More generally, NIZK proofs allow a prover to convince a verifier that some statement is true by sending only a bunch of data with no other interactions.

Perhaps the simplest zero-knowledge proof system is *Schnorr's* protocol: say you know the solution of a discrete logarithm problem (the hard problem behind DSA and elliptic curve cryptography) and want to prove that you know the solution, but without revealing any bit of the solution. That is, you know the x such that $g^x = y \bmod p$, and the verifier only knows g, y, and p. To convince a verifier, you will then play the following game:

1. Pick a random number r and send g^r to the verifier (the *commitment*)
2. Receive a random number c from the verifier (the *challenge*)
3. Send the number $s = r + cx$ to the verifier
4. The verifier computes $g^s = g^{r + cx}$ and checks that it's equal to $g^r \times y^c = g^r \times (g^x)^c$

In this interactive protocol, the prover doesn't reveal any information on x, because everything they send is random. Yet only a prover that knows x would be able to send an s that passes the final verification.

To turn such interactive protocols into non-interactive ones (that is, in a single piece of data), NIZK proofs are built by playing with yourself and *simulating* a verifier, in such a way that the actual verifier is convinced that you couldn't have made up the NIZK proof without knowing the proven statement.

Conclusion

Key ideas viewed in this post are:

- The crux of secure e-voting system is *public verifiability*. It's achieved by posting encrypted ballots on a publicly readable forum. Election officials should also describe the mechanism that performs the actual verification.
- *Voter confidence* is achieved by authenticating each voter with a unique ID and enabling voters to verify that their ballot 1) was counted and 2) hasn't been tampered with.
- Voters can't be punished for voting for the "wrong" candidates, thanks to guaranteed *coercion resistance*, achieved in part from unpredictable, probabilistic encryption.
- *Privacy of ballots* is ensured by never decrypting the encrypted votes, but only decrypting the tally created from homomorphic encryption.
- *Fraud* is defeated by forcing voters to release a cryptographic proof that their ballot is a legitimate one, thanks to non-interactive zero-knowledge (NIZK) proof techniques.

Concepts and techniques presented here may look deep and sophisticated, but in fact I've only scratched the surface. You won't get a secure working e-voting system if you just follow the description above; I've for example omitted details on how voters can verify their ballots in practice, why the server uses NIZK proofs, and so on. The upshot is that any secure e-voting protocol is a very complicated and subtle machinery, and actual deployments add even more complexity due to human and operational factors.

To learn more about the cryptography behind e-voting, the following are good references:

- David Chaum's 2004 article in IEEE S&P: https://people.csail.mit.edu/rivest/voting/papers/Chaum-SecretBallotReceiptsTrueVoterVerifiableElections.pdf

- Report *The Future of Voting* sponsored by the US Vote Foundation: https://www.usvotefoundation.org/sites/default/files/E2EVIV_full_report.pdf
- Ben Adida's GoogleTech talk: https://youtu.be/ZDnShu5V99s
- How human and operational factors can compromise e-voting schemes: https://www.usenix.org/legacy/event/sec05/tech/full_papers/karlof/karlof.pdf
- Lecture notes on zero-knowledge proofs (describing the Chaum-Pedersen protocol used in Helios): http://www.cs.jhu.edu/~susan/600.641/scribes/lecture10.pdf

CRYPTO CRYPTOGRAPHY E-VOTING ELECTION ELECTION HACKING MIDTERM MIDTERM ELECTION MIDTERMS 2018 VOTING

« Build Your Own Hardware Implant

Ledger Donjon wins Kudelski Security Crypto Challenge: Behind the scene 1/3 »

LEAVE A REPLY

Enter your comment here...