Security and Privacy

E-voting bugs

28.05.2019

Ph. Oechslin

Intrusion tests and code publication

- As part of the certification of Post's completely verifiable e-voting protcol
  - the source code was published
  - then a public intrusion test was carried out.
- The code publication showed some important bugs
- Nobody managed to break into the system during the intrusion test

The bugs

- Trapdoor in commitments

- The (complicate) proofs of the mixnet used by Svote make use of Pederson commitments

- These commitments need two or more numbers: $H, G_1, G_2, \ldots \in G$

- These commitments are only safe if the logarithms of these numbers are not known
  - H, G_i are members of the group G, so they can be written as g^x with x being their logarithm.

- It is thus important that the prover does not know the logarithm

Commitment bug: the code

```
public CommitmentParams(final ZpSubgroup group, final int n) {
  this.group = group;
  // get random H
  this.h = GroupTools.getRandomElement(group);
  this.commitmentlength = n;
  // get list of random Gs
  this.g = GroupTools.getVectorRandomElement(group, this.commitmentlength);
}
```

```
public static ZpGroupElement getRandomElement(ZpSubgroup group) {
 Exponent randomExponent = ExponentTools.getRandomExponent(group.getQ());
 return group.getGenerator().exponentiate(randomExponent);
```

- Quiz: where is the bug?

Commitment bugs

- First error:
  - By choosing random elements of the group as $g^{\texttt{randomExponent}}$ the prover knows the logarithm (it is `randomExponent`)
  - The proof thus has no value, as the prover could use the logarithm to manipulate the commitments
    - Of course, there is no trace of any such manipulation in the program
  - "You would have to hack the CCs in order to manipulate the proofs"
    - yes, but the goal of the proofs is to demonstrate that no manipulation happened

Commitment bugs

- Second error:
  - Even if you generate H and G_i differently, how do we know that you don't just 'randomly' chose values for which you know the logarithms
    - this is the nothing up my sleeve issue.
  - To solve this we can use standardized algorithms for choosing generators (e.g. **NIST FIPS 186-4, Appendix 2.3**)
  - They use deterministic inputs (e.g. a counter and a constant string) and hash functions
    - Then you can generate, for control component 1, H with (1,"H of CC1"), and G_i with (i,"G of CC1")

NIZKP bugs

- Decryption proof error

- Remember:
  - $\mathtt{Enc}_{pk}(m, r) = (m \cdot pk^r, g^r) = (a, b)$
  - $\mathtt{Dec}_{sk}(a, b) = a/b^{sk} = m$

- After decrpytion, Svote's proves the equality of logarithm of $a/m = pk^r = g^{r^{sk}}$ in base $b = g^r$ and the public key $pk = g^{sk}$ in base g.

- It turns out that after you have calculated the challenge for the proof, you can go back and change b in certain ways and still have a valid proof.
  ➡ thus you can generate a valid proof and present an incorrect decryption

NIZKP bugs

- Decryption proof error

- It is difficult to turn a "yes" vote into a "no" vote at decryption, but you can make a vote invalid and still have correct proofs

- To do the proof correctly, you must include the value of a in the hash of the proof.
  - If you change a the verifier will see that the hash is not correct

NIZKP bugs

- Plaintext equality proof error

- The same error happens in the plaintext equality proof, which is also a proof of equal logarithm

- Remember: $\pi_{pleq} = NIZKP[(r \cdot \mathtt{VC}_{sk}^{id}) : \tilde{c}_2 = g^{r\mathtt{VC}_{sk}^{id}} \wedge \frac{\tilde{c}_1}{\prod_{l=1}^{t} \mathtt{pvc}_l^{id}} = pk^{r\mathtt{VC}_{sk}^{id}}]$

- You can calculate the proof and then modify a of the encrypted vote such that the proof is still correct.

- You can't turn a "yes" into a "no", but you can turn a "yes" into something that does not make sense

NIZKP bugs

- Plaintext equality proof error

- Possible attack:
  - The attacker sits in you browser
  - If you vote "yes", they don't intervene
  - If you vote "no", they manipulate a such that your vote makes no sense
    - you still get the correct verification codes
    - "yes" wins
  - individual verifiability is broken!
  - This is why Post has taken their current system offline.

- Simple solution
  - add a (and as many other parameters) into the hash function.

Conclusions

- Implementing crypto is hard
- Reviewing crypto is hard
  - There was a review by KPMG which certified that the code corresponded to the specification
    - the specification did not correspond to the official protocol...
    - The paper that describes the protocol and the proofs, was actually correct!
- Publishing code is a very good way of catching errors
  - it is not a guarantee
- There should be a strong interaction between people designing crypto and those implementing crypto
- Zero knowledge proofs can only detect errors in the system if they are implemented correctly

Questions

- What is individual verifiability?
  - what does it protect against?
- What are three important security objectives of e-voting systems?
- When voting is done by raising hands, which of these objectives are met?
- When a trust model of a system has more trusted parts, does it make the system more secure?
- What is an advantage of splitting keys into shares?
  - What could be a risk?
- What is an example of something that can be proven with an NIZKP in a e-voting protocol?
  - no need to know the formula, just what do we prove without revealing what?

References

- Olivier Pereira's **blog**
- His papers, with Sarah Jamie Lewis and Vanessa Teague
  - **Ceci n'est pas une preuve**
  - **How to not prove an election outcome**
  - **How to not prove an election outcome Addedum**