

# COMP 3704 Computer Security

Christian Grothoff

`christian@grothoff.org`

`http://grothoff.org/christian/`

# Zero-Knowledge Proofs

1. Peggy wants to prove to Victor that she has a particular piece of information, but without giving Victor the information.
2. Peggy transforms NP-complete problem  $P$  into equivalent NP-complete problem  $V$  using secret, random transformation  $T$
3. Given  $V$ , Victor can choose to see either solution to  $V$  or transformation  $T$ .
4. Peggy has 50% chance of cheating, so iterate  $n$  times!

## Example: Graph Isomorphism

Two graphs are isomorphic if they are identical modulo renaming of points; determining GI is NP-complete.

1. Peggy wants to provide zero-knowledge proof of her knowing the isomorphism between  $G_1$  and  $G_2$ .
2. Peggy randomly permutes  $G_1$  to produce  $H$  and now has isomorphisms  $G_1 \equiv H \equiv G_2$ .  $H$  is given to Victor.
3. Victor requests proof that either  $G_1 \equiv H$  or  $H \equiv G_2$ .
4. Peggy provides the requested isomorphism, which Victor verifies.

# Noninteractive Zero-Knowledge Proofs

- Peggy does not want to repeat the interactive proof to everyone.
- Instead of having Victor choose at random, use hash-function over information provided to Victor as PRNG.
- Use combined  $n$  transformations of the problem as input to PRNG.

# Blind Signatures

1. Alice sends Bob  $M \cdot R_A$  where  $R_A$  is called a **blinding factor**
2. Bob sends back  $S_{B_{priv}}(M \cdot R_A)$
3. Alice divides out the blinding factor, obtaining  $S_{B_{priv}}(M)$

Naturally, this only works if the signature function allows Alice to divide the blinding factor.

# Half-Blind Signatures

1. Alice sends Bob  $n$  blinded documents.
2. Bob requests the blinding factors for  $n - 1$  random documents.
3. Alice provides those blinding factors.
4. Bob unblinds and checks that the  $n - 1$  documents would have been acceptable and blindly signs the remaining document.
5. Alice can get Bob to sign anything with probability  $1 : n$ .

# Identity-based Public-Key Cryptography

- Idea: generate public-private key pairs based on the identity of the users
- Practical variant: define your identity to be your public key
- Issue: anyone can create any number of identities

# Oblivious Transfer

1. Alice generates two public-key pairs, sends the public keys to Bob.
2. Bob sends Alice  $E(K)$  using one of the public keys.
3. Alice decrypts with both of her keys, obtaining  $K$  and  $K'$ .
4. Alice sends Bob  $E_K(M_1), E_{K'}(M_2)$  or  $E_{K'}(M_1), E_K(M_2)$  (but she does not know which).
5. Bob tries to decrypt both, gets either  $M_1$  or  $M_2$ .



# Oblivious Transfer: Verification

- If Bob wants to verify that Alice did not cheat, he needs to receive Alice's private keys at some point (at that time he will learn both messages).

# Simultaneous Contract Signing (1/2)

1. Alice and Bob each generate  $2n$  symmetric keys and  $n$  pairs of messages  $L_i$  and  $R_i$  representing the left and right halves of the  $i$ -th signature.
2. Alice and Bob exchange the encrypted signature pairs.
3. Alice and Bob use oblivious transfer to share  $n$  of the  $2n$  keys (one for each pair).
4. Alice and Bob verify that the  $L_i$ 's and  $R_i$ 's that they can now decrypt are valid.

## Simultaneous Contract Signing (2/2)

5. Alice sends Bob the first bits of her  $2n$  symmetric keys.
6. Bob sends Alice the first bits of his  $2n$  symmetric keys.
7. They iterate the previous steps until all bits of all keys have been transferred.
8. They decrypt the remaining halves of the message pairs and obtain a valid signature.
9. They exchange the private keys used during oblivious transfer and verify that the other did not cheat.

# Questions



# Problem

Why is the protocol described in the textbook in Section 5.8, pages 122-123 broken?