Part II of Introduction to Cryptography

Konstantin Beznosov
COT 6421 / Spring 1998

April 7, 1998

We Will Discuss Today:

- Probabilistic encryption
  - Average Case Computational Difficulty and the Worst Case Difficulty
- Identity-Based Public-Key Cryptography
- Fair Coin Flipping Using Public-Key Cryptography
- Fair Cryptosystems (Key Escrow)
- Zero Knowledge Interactive Proof Systems

Average Case / Worst Case Difficulty: Cryptosystem

Cryptosystem is a collection of *hidden hyperplanes*:

- PRIVATE KEY – *hidden hyperplanes*
- PUBLIC KEY – a set of random points near the hyperplanes
  - Not to reveal the collection of hyperplanes
  - A random subset sum is close to a hyperplane
- A large n-dimensional cube $\vartheta \in R^n$is selected and fixed.

Average Case / Worst Case Difficulty: Encryption

Encryption is bit-by-bit:

- **0** – using public key to find a random vector $v \in \vartheta$ near one of the hyperplanes
  - the ciphertext is *v*;
- **1** – choosing a random vector *u* uniformly from ϑ
  - the ciphertext is simply *u*.

Average Case / Worst Case Difficulty: Decryption

- Determine the distance of ciphertext x to the nearest hidden hyperplane.
  - If this distance is sufficiently small, then x is decrypted as zero;
  - otherwise x is decrypted as one.

Small (but polynomial) probability of an error in decryption: **1** decrypted as **0**.

The ability to distinguish encryptions of zero from encryptions of one yields the ability to solve the hidden hyperplane problem. This implies that the only way to break the cryptosystem is to find the private key.

Average Case / Worst Case Difficulty: Equivalence Theorem

Theorem 1. *For all $c_1, c_2, c_3, c_4 > 0$ there exists a c_5 and a probabilistic algorithm B (using an oracle) so that for all sufficiently large n, condition (1) implies condition (2), where*

1. *A is a probabilistic circuit size n^{c_1} so that if $u, v_1, ..., v_m$ are picked at random as described in the protocol generating the public and private keys, then with probability of at least n^{-c_2}, A distinguishes the random variables $S_{v_1,...,v_m}$ and $E_{v_1,...,v_m}$, given $v_1, ..., v_m$ with probability at least $\frac{1}{2} + n^{-c_3}$.*

2. *B, using A as an oracle, can solve any instance of size at most n^{c_4} of the n^{D_2}-unique shortest vector problem in time n^{c_5} and with a probability at least $1 - 2^{-n}$.*

Identity-Based Public-Key Cryptography: Why?

Alice $\longrightarrow$ [secure message] $\longrightarrow$ Bob

✘ Public key from a key server

✘ Verify some trusted third party's signature on his public-key certificate

✘ Store Bob's public key on her own computer

✔ Send him a secure message

Identity-Based Public-Key Cryptography

Also called Non-Interactive Key Sharing systems:

- Bob's public key is his identity
- Makes the cryptography about as transparent as possible

Problems:

- Issuing private keys to users based on their identity
- Change your identity to get another public/private key pair if your identity is compromised

Most solutions turned out to be insecure or are very complicated to be useful in real life.

Fair Coin Flipping with PK Cryptography

The only requirement: the algorithm commute, i.e.

$$D_{K_1}(E_{K_2}(E_{K_1}(M))) = E_{K_2}(M)$$

It is not true for symmetric algorithms

It is true for some public-key algorithms: RSA with identical moduli

Fair Coin Flipping: Simple Protocol

1. Alice and Bob each generate a public-key/private-key key pair.
2. Alice generates M_T, M_H – unique random strings. Alice $\longrightarrow \frac{E_A(M_T)}{E_A(M_H)}$
3. $\frac{E_A(M_H)}{E_A(M_T)} \longrightarrow$ Bob; $E_B(E_A(M)) \longrightarrow$ Alice
4. Alice $\longrightarrow D_A(E_B(E_A(M))) = E_B(M) \longrightarrow$ Bob
5. Bob $\longrightarrow D_B(E_B(M)) = M \longrightarrow$ Alice
6. Alice verifies the arbitrary string (either for M_T or M_H) is correct.
7. Both Alice and Bob reveal their key pairs to check each other.

Protocol application: session-key generation

Fair Cryptosystems (Key Escrow): Idea

1. Alice creates her private-key/public-key key pair. She splits the private key into several public pieces and private pieces.

2. Alice sends a public piece and corresponding private piece to each of the trustees. She also sends the public key to the KDC.

3. Each trustee, independently confirms that public and private parts are correct. Each trustee stores the private piece somewhere secure and sends the public piece to the KDC.

4. The KDC performs another calculation on the public pieces and the public key. Assuming that everything is correct, it signs the public key.

If the courts order a wiretap, then each of the trustees surrends their piece to the KDC, and the KDC can reconstruct the private key.

Fair Diffie-Hellman Algorithm

Basic Diffie-Hellman algorithm: a group of users share a prime *p*, and a generator, g.

Alice's:

- private key – *s*
- public key – $t = g^s \, mod \, p$.

Fair Diffie-Hellman Algorithm: Detailed Example

1. Alice chooses integers, $s_1, s_2, s_3, s_4\, and\, s_5$, each < p-1. Alice's private key is $s = \{s_1 + s_2 + s_3 + s_4 + s_5\}\, mod\, p - 1$ and her public key is $t = g^s\, mod\, p$. Alice also computes $t_i = g^{s_i}\, mod\, p,\ for\, i = 1\, to\, 5$ Alice's public shares are t_i, and her private shares are s_i.

2. Alice sends t_i and s_i to each $trustee_i$. She sends t to KDC.

3. Each trustee verifies that $t_i = g^{s_1}\, mod\, p$, signs t_i and sends it to the KDC. The trustee stores s_i in a secure place.

4. After receiving all five public pieces, the KDC verifies that $t = \{t_1 * t_2 * t_3 * t_4 * t_5\} mod\, p$. If it does, the KDC approves the public key.

Fair Diffie-Hellman Algorithm

The KDC knows that:

- the trustees each have a valid piece
- they can reconstruct the private key if required.

Neither KDC nor any of four of the trustees working together can reconstruct Alice's private key.

Research in this area found how to make RCA fair and how to combine a threshold scheme with the fair cryptosystem, so that m out of n trustees can reconstruct the private key.

Zero-Knowledge Interactive Proof Systems

Is it possible to prove a statement without yielding anything beyond its validity?

Zero-knowledge proofs are proofs that yield no knowledge beyond the validity of the assertion.

We gain knowledge only if we receive the result of a computation which is infeasible for us.

Knowledge is related to computational difficulty, whereas information is not.

Concrete Example: 0

Alice claims she knows a "Hamiltonian cycle" on a particular graph – her proof of identity.

1. Alice scrambles the position of the cities.

2. Bob doesn't have any way of knowing which of the covered cities are which.

Concrete Example: 1

3. "Alice cuts, Bob chooses:" Bob is to make her either to show:
   - the Hamiltonian cycle for this set of covered cities and links, or
   - the cities uncovered.
4. Bob tosses a coin or chooses randomly somehow and says: "Show me the cities."
5. Alice uncovers all the cities and Bob examines the graph and sees she showed him the original graph.
6. Back to step 3.

Concrete Example: 2

After 30 rounds, Alice has either

- produced a legal Hamiltonian cycle or
- produced a graph that is the same as (isomorphic to same cities linked to same other cities) the registered graph in each and every one of the rounds.

The odds of #1 being true drop rapidly as the number of rounds are increased – 1 in 2^n

Bob believes Alice knows the solution.

Some Questions

1. Could someone else discover the Hamiltonian cycle of Alice's graph?

   The Hamiltonian cycle problem is "NP-complete.'" 50 nodes is intractable.

2. If finding a Hamiltonian cycle is intractable, how did Alice ever find one?

   She DIDN'T HAVE to find one! She knows a Hamiltonian cycle, BY CONSTRUCTION.

3. Can Bob reconstruct what the Hamiltonian cycle must be by asking for enough rounds to be done?

   Not generally.