06-06798 Distributed Systems

Lecture 15: Cryptographic algorithms

Overview

- Cryptographic algorithms
  - symmetric: TEA
  - asymmetric: RSA
- Digital signatures
  - digital signatures with public key
  - secure digest function
- Authentication
  - secret-key Needham-Schroeder
  - scenarios

Cryptographic algorithms

- Symmetric (secret key): TEA, DES
  - secret key shared between principals
  - encryption with non-destructive opns (XOR) plus transpose
  - decryption possible only if key known
  - brute force attack (check $\{M\}_K$ for all values of key) hard (exponential in no of bits in key)
- Asymmetric (public key): RSA
  - pair of keys (very large numbers), one public and one private
  - encryption with public key
  - decryption possible only if private key known
  - factorising large numbers (over 150 decimal digits) hard

Tiny Encryption Algorithm(TEA)

- Simple, symmetric (secret key) algorithm
  - written in C [Wheeler & Needham 1994]
- How it works
  - *key* 128 bits (*k[0]..k[3]*)
  - *plaintext* 64 bits (2 x 32 bits, *text[0], text[1]*)
  - in 32 rounds combines *plaintext* and *key,* swapping the two halves of *plaintext*
  - uses reversible addition of unsigned integers, XOR (^) and bitwise shift (<<, >>)
  - combines *plaintext* with constant *delta* to obscure *key*
- Decryption via inverse operations.

TEA Encryption function

```
void encrypt(unsigned long k[], unsigned long text[]) {
    unsigned long y = text[0], z = text[1];                            1
    unsigned long delta = 0x9e3779b9, sum = 0; int n;                  2
    for (n= 0; n < 32; n++) {                                          3
        sum += delta;                                                  4
        y += ((z << 4) + k[0]) ^ (z+sum) ^ ((z >> 5) + k[1]);          5
        z += ((y << 4) + k[2]) ^ (y+sum) ^ ((y >> 5) + k[3]);          6
    }
    text[0] = y;  text[1] = z;                                         7
}
```


TEA Decryption function

```
void decrypt(unsigned long k[], unsigned long text[]) {
    unsigned long y = text[0], z = text[1];
    unsigned long delta = 0x9e3779b9, sum = delta << 5;  int n;
    for (n= 0; n < 32; n++) {
        z -= ((y << 4) + k[2]) ^ (y + sum) ^ ((y >> 5) + k[3]);
        y -= ((z << 4) + k[0]) ^ (z + sum) ^ ((z >> 5) + k[1]);
        sum -= delta;
    }
    text[0] = y; text[1] = z;
}
```

Other symmetric algorithms

- TEA
  - simple & concise, yet secure and reasonably fast
- DES (The Data Encryption Standard 1977)
  - US standard for business applications till recently
  - 64 bit plaintext, 56 bit key
  - cracked in 1997 (secret challenge message decrypted)
  - triple-DES (key 112 bits) still secure, poor performance
- AES (Advanced Encryption Standard)
  - invitation for proposals 1997
  - in progress
  - key size 128, 192 and 256 bits

RSA

- Rivest, Shamir and Adelman '78
- How it works
  - relies on $N = P \times Q$ (product of two very large primes)
  - factorisation of N hard
  - choose keys e, d such that
    $e \times d = 1 \ mod \ Z$ where $Z = (P-1) \times (Q-1)$
- It turns out...
  - can encrypt M by $M^e \ mod \ N$
  - can decrypt by $C^d \ mod \ N$ (C is encrypted message)
- Thus
  - can freely make e and N public, while retaining d

RSA: past, present and future

- In 1978...
  - Rivest *et al* thought factorising numbers > 10^{200} would take more than four billion years
- Now (ca 2000)
  - faster computers, better methods
  - numbers with 155 (= 500 bits) decimal digits successfully factorised
  - 512 bit keys insecure!
- The future?
  - keys with 230 decimal digits (= 768 bits) recommended
  - 2048 bits used in some applications (e.g. defence)

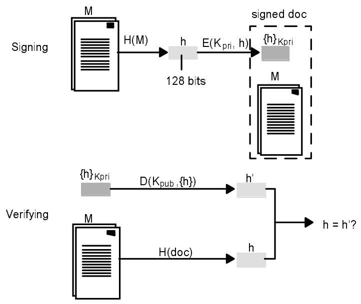
Digital signatures

- Why needed?
  - alternative to handwritten signatures
  - authentic, difficult to forge and undeniable
- How it works
  - relies on secure hash functions which compress a message into a so called *digest*
  - sender encrypts *digest* and appends to message as a signature
  - receiver verifies signature
  - generally public key cryptography used, but secret key also possible

Digital signatures with public key

- Keys
  - sender chooses key pair K_{pub} and K_{pri}; key K_{pub} made public
- Sending signed message M
  - sender uses an agreed secure hash function h to compute *digest* h(M)
  - *digest* h(M) is encrypted with private key K_{pri} to produce signature $S = \{h(M)\}_{K_{pri}}$; the pair M, S sent
- Verifying signed message M, S
  - when pair M, S received, signature S decrypted using K_{pub}, digest h(M) computed and compared to decrypted signature
- Note
  - RSA can be used, but roles of keys reversed.

Digital signatures with public key



Secure digest functions

- Based on one-way hash functions:
  - given M, easy to compute h(M)
  - given h, hard to compute M
  - given M, hard to find another M' such that h(M) = h(M')
- Note
  - operations need not be information preserving
  - function not reversible
- Example: MD5 [Rivest 1992][
  - 128 bit digest, using non-linear functions applied to segments of source text

Authentication

- Definition
  - protocol for ensuring authenticity of the sender
- Secret-key protocol [Needham & Schroeder '78]
  - based on secure key server that issues secret keys
  - see this lecture and textbook (5 steps)
  - flaw corrected '81
  - implemented in Kerberos
- Public-key protocol [Needham & Schroeder '78]
  - does not require secure key server (7 steps)
  - flaw discovered with CSP/FDR
  - SSL (Secure Sockets Layer) similar to it

Needham-Schroeder secret-key

- Principals
  - client A (initiates request), server B
  - secure server S
- Secure server S
  - maintains table with name + secret key for each principal
  - upon request by client A, issues key for secure communication between client A and server B, transmitted in encrypted form ('ticket')
- Messages
  - labelled by nonces (integer values added to message to indicate freshness)

Needham-Schroeder secret-key

Header	*Message*	*Notes*
1. A->S:	A, B, N_A	A requests S to supply a key for communication with B.
2. S->A:	$\{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_B}\}_{K_A}$	S returns a message encrypted in A's secret key, containing a newly generated key K_{AB} and a 'ticket' encrypted in B's secret key. The nonce N_A demonstrates that the message was sent in response to the preceding one. A believes that S sent the message because only S knows A's secret key.
3. A->B:	$\{K_{AB}, A\}_{K_B}$	A sends the 'ticket' to B.
4. B->A:	$\{N_B\}_{K_{AB}}$	B decrypts the ticket and uses the new key K_{AB} to encrypt another nonce N_B.
5. A->B:	$\{N_B - 1\}_{K_{AB}}$	A demonstrates to B that it was the sender of the previous message by returning an agreed transformation of N_B.

Problems!

- In step 3
  - message need not be fresh...
- So...
  - intruder with K_{AB} and $\{K_{AB}, A\}_{KB}$ (left in cache, etc) can initiate exchange with B, impersonating A
  - secret key K_{AB} compromised
- Solution
  - add nonce or timestamp to message 3, yielding $\{K_{AB}, A, t\}_{KBpub}$
  - B decrypts message and checks t recent
  - adapted in Kerberos

Summary

- Symmetric encryption
  - DES: most widely used till recently, 56-bit key insecure
  - 3DES, AES or IDEA an alternative
- Asymmetric encryption
  - RSA: 512-bit key insecure, use with 768-bit keys or above
- Authentication with secret-key
  - Kerberos, based on [Needham-Schroeder '78]
- Authentication with public-key
  - SSL (Secure Sockets Layer)
  - used in electronic commerce