

## **1. CONFIDENTIALITY**

**Magnus Jensen**

- Motivation
- Private key systemer
  - One Time Pad
    - *Sikkerhed for One Time Pad*
      - *Bruteforce*
  - *Sikkerhed i secret-key*
    - *Samme key flere gange*
    - *NONCE*
    - *Cost of attack*
  - Ciphers
    - Stream ciphers
    - Block Ciphers
  - *Pro / Con*
    - *Hastighed og Nøgler*
- Public key systemer
  - Sikkerhed
    - *Tilfældighed*
    - *Key længde*
  - RSA
    - *Nøgler*
    - *(De)Kryptering*
    - *OAEP*
      - *Public key til secrete key*

## CONFIDENTIALITY

### 1. Motivation

Confidentiality, eller fortrolighed - altså at vi holder ting gemt for andre, kan vi inden for kryptografi gøre på to måder.

#### 1.1 Privat / Public key

Enten bruger vi **private key**, hvor alle parter kender en hemmelig kode, som ingen andre må kende.

Eller så bruger vi en **public key** system, hvor man har sin egen hemmelige nøgle, og andre har en offentlige nøgle.

Kort forklaret, kan **andre sende** noget krypteret under ens offentlige nøgle, og kun en selv med den hemmelige kan læse det.

### 1.2 Ciphertext

Så for begge gælder det at vi bruger en key til at krypter noget data, til hvad vi kalder **ciphertext**:

$$c = E_k(data)$$

## 2. Private key systemer

Lad os først tage et eksempel på hvordan vi kan lave et **private-key system**.

### 2.1 One Time Pad

One Time Pad har hvad vi kalder **Ubetinget sikkerhed**, hvilket betyder simpelt at det er sikkert, og **intet kan være mere sikkert**.

Uanset mængden af **computerkræft** til rådighed, vil man ikke kunne vide indholdet af en ciphertext.

#### *One Time Pad Key*

- *Bruges kun en gang*
- *Er tilfældig*
- *Længde som besked*
- *Keys XOR's med beskeder for at få ciphertekst*

Fordi XOR er en **uniform proces**, kan man ved at **have to fra sættes**

$$\{M, K, C\}$$

Altid udregne I den sidste.

Derfor så har vi at One Time Pad virker:

$$C \oplus K = (M \oplus K) \oplus K = M \oplus (K \oplus K) = M$$

#### *2.1.1 Sikkerhed for One Time Pad*

Key er **tilfældig**, så kun bruteforce kan virke.

Men fordi keyen er lige så lang som beskeden, vil det tage tiden  $2^t$  hvor  $t$  er længden af  $m$  at bruteforce.

$$\text{bruteforce tid} = 2^t$$

Men her kommer **problemet** ved bruteforce, nemlig at vi bruger **XOR**.

Derfor vil **mange plausible løsninger** komme frem, og vi **ved ikke hvilken** der er rigtig.

**One Time Pad virker kun**, hvis både sender og modtager **kender alle keys** for hver eneste besked der vil sendes frem og tilbage.

## 2.2 Sikkerhed I secret-key: computational security

I virkelig verden deler **mange beskeder dog samme key**.

### 2.2.1 Samme key flere gange

Derfor bruger vi **computational security**. Som er sikkerhed så kraftig, at det tager **urealistisk lang tid** at bryde det.

**Problem:** hvis samme besked sendes under samme key flere gange, kan man se det.

### 2.2.2 NONCE

Det kan vi ikke lide, så derfor tager vores krypterings algoritme hvad vi kalder et **nonce**, der er unikt for hver kryptering.

Vores krypterings algoritme bliver da:

$$c = E_k(m, n)$$

Derfor vil en fremmed, aldrig kunne se at den samme besked er sendt flere gange under samme nøgle.

### 2.2.3 Cost of attack

Hvis en **fremmed får fat** i nogle beskeder og deres ciphertexter; kan han prøve at bruteforce deres nøgler.

Derfor bruger vi **lange nøgler**, med minimum 128 bit længde.

*Private key: længde: 128 bit*

Da der så vil være  $2^{128}$  **muligheder for nøglen**.

Hvilket er **for mange** til en fremmed at regne ud.

Et angreb vil altså have et for højt - **Cost of attack**.

### 2.3 Ciphers

I private key systemer, bruger vi to forskellige måde at lave ciphers på.

#### 2.3.1 Stream ciphers

Streamciphers gør det muligt at **lave keys** der er **lige så lange som vores beskeder**.

$G(k, n)$

Den tager vores *normale key* og vores *nonce*, hvorfor vi nu kan bruge XOR.

$c = m \oplus G(K, n)$

**Forskellen** ved dette og den rigtige One Time Pad, er at G **kun ser tilfældig ud, men ikke er det**.

Det **kaldes** stream cipher, da man blot vil kunne **initialisere G** og så bare **streame fra den**.

#### 2.3.2 Block Ciphers

I stedet for at kryptere hele beskeden på en gang, så sker det **over flere gange**.

Der findes forskellige måder:

- **OFB: Output Feedback**
- **CBC: Cipher Block Chaining**
- **CTR: Counter Mode**

Alle tre måder virker ved:

$$c = ENC_k(m, IV)$$

Hvor IV (initialization vector) blot er det samme som nonce.

**Output Feedback** kan bruges som en **stream** til at XOR med, og bruges som: ENCK(IV), ENCK(ENCK(IV)), ...

$$OFM = ENCK(IV), ENCK(ENCK(IV)), \dots$$

I **Cipher Block Chaining** arbejder vi altså med **blokke af data**, hvor den **sidste er pattet**. Vi lader krypteringen tage den sidste ciphertekst som input

$$C_i = ENCK(M_i \oplus C_{i-1})$$

Og lader

$$C_0 = IV$$

I **Counter Mode** gør vi tæt på det samme, men holder en counter:

$$C_i = ENK(IV + i) \oplus M_i$$

## 2.4 Pro / Con

### 2.4.1 Hastighed og Nøgler

**Pro:** Secret key systemer er hurtige, 10-100 Mbytes/sec ved software, hurtigere med dedikeret hardware

**Con:** Key skal være delt før data sendes. Specielt problem hvis flere brugere skal kommunikere og alle have hinanden nøgle.

## 3. Public key systemer

Her har hvert part som sagt en privat og en offentlig nøgle.

Hvor man krypter med en public nøgle, og den private nøgle der passer til - er så den eneste der kan dekryptere beskeden igen.

### 3.1 Sikkerhed

Der er en **speciel sammenhæng** i mellem ens private og en public nøgle - men som er meget svær at regne ud.

Så har man en public key, er det virkelig **svært at udregne den private nøgle**.

Derfor vil en fremmed der har en c og en publik key, ikke kunne se indholdet.

### *3.1.1 Tilfældighed*

Det er desuden vigtigt at hver kryptering indeholder noget tilfældigt.

For hvis man får en c, vil man blot kunne kryptere en masse indtil man får:

$$c' = c$$

Hvorfor man ved hvad c er.

### *3.1.2 Key længde*

Der findes algoritme der kan "hurtigt" udregne **sk fra pk**.

Så derfor bruger vi i public-key systemer meget større keys.

Jeg mindes vi brugte 2048 bit.

*Public key længde: 2048 bit*

## **3.2 RSA**

Er en måde at lave Public Key systemet på.

### *3.2.1 Nøgler*

Publik keyen består af to numre:

$$Pk = e \text{ og } n$$

Og Secrete key består af:

$$Sk = d \text{ og } n$$

N kaldes for modulet og er et **produkt af to primtal**. De andre numre er valgt til at tilfredstille en bestemt relation.

### 3.2.2 (De)Kryptering

Så på grund af matematik (?) så virker kryptering ved:

$$c = m^e \% n$$

Og igen på grund af matematik så virker dekryptering ved at sige:

$$m = c^d \% n$$

### 3.2.3 OAEP

En fremmed kan dog **lege med dekrypterings algoritmen**, og se hvordan den **opfører sig**, og derfor få visse information om secret keyen.

Derfor kan man bruge RSA+OAEP, der padder ens besked med noget tilfældigt data.

#### 3.2.3.1 PUBLIC KEY TIL SECRETE KEY

Man kunne f.eks bruge RSA+OAEP til at **opsætte secret key system** med, derved få nemmere og hurtigere sikkerhed.