3. KEY MANAGEMENT AND INFRASTRUCTURE

Magnus Jensen

- Motivation
- Session Keys
  - Long term key
- Key Distribution Centers
  - Eksempel
  - Problemer
- Certification Authorities
  - Registrering
  - Sikkerhed
- Certificate Chains
  - Eksempel, untrusted CA
- Tilkendegivelse
  - Kodeord
    - Sikkerhed: kvalitet og server
  - Hardware
    - Tamper Evident
      - Two Factor Authentication
    - Tamper Resident
  - Biometrics
    - Buffer til forandring

KEY MANAGEMENT AND INFRASTRUCTURE

1. Motivation

Så i kurset har vi snakket rigtig meget om nøgler, og nu skal vi så finde ud af - hvordan nøgler bliver administreret.

2. Session Keys

Et hvert sikkert system **løber konstant risikoen** om at blive eksponeret; des længere det ikke fornyer sig og stadig benyttes.

Derfor vil kommunikation primært foregå **via session keys**, hvor keys altså bliver udskiftet.

2.1 Long term key

Dettte sker typisk ved, at A og B **på forhånd har aftalt** en long term key Kab; de bruger til at sende nye session keys frem og tilbage.

Det virker ved, at hvis A vil sende M, sender A:

> *(E_Kab(Ks), Eks(m))*

Så kan B bruge Kab til at skaffe Ks, og så skaffe m.

Hver part skal altså have en long term key, for hver anden part. Antallet af nøgler **stiger eksponentielt**.

Derfor bruges der ofte andre løsninger.

3. Key Distribution Centers

Et Key Distribution Center er baseret på Secret Key teknologi.

Ideen er at hver part **deler en key** med KDC'en. Så alle kan kommunikere privat med KDC'en.

3.1 Eksempel

Lad os sige at **A vil kommunikere med B**.

Så den beder KDC'en om en session key

> *A --> KDC*

Nu sender KDC'en en session key til A:

> *KDC --> A: E_Ka(Ks)*

Og en session key til B

> *KDC --> B: E_Kb(Ks)*

Hvor vi bemærker **hver besked er krypteret** til den respektive modtager.

Nu kan A og B snakke sammen under session keyen.

3.2 Problemer

Hvis **KDC'en bryder ned**, kan ingen snakke sammen.

Man kan ikke vide om en **forbindelse er sikker**; en session key kunne være sendt flere steder hen.

Det kræver alle **stoler** på **KDC'en**, da den har den ultimative mulighed for at **afkode alle beskeder**, eftersom det er den som genere nøglerne.

4. Certification Authorities

Hvor KDC'er bruger Secret Keys, bruger Certification Authorities Public Key systemer.

CA'en starter med at * sit egen * (SKca, PKca); og alle brugere af CA'en får dens public-key.

> - *CA laver et keypair*
> - *Alle får PKca*

At alle har PKca bruges senere til at garantere, at man kan stole på modtageren.

4.1 Registrering

Det kan være **bøvlet at registrere** sig ved CA'en.

Det skal nemlig ske **ukrypteret**, da CA'en intet ved om en.

F.eks kunne det ske ved at møde op personligt. Hvor A giver CA PKa.

> *A --> CA: PKa*

Så får A et certifikat af CA'en:

> *CA --> A: (IDa, PKa, S_SKca(IDa, PKa))*

Der består af:

- Et ID
- Ens publik key
- Og en signatur over disse to, signeret af CA'en

Alle kan nu **tjekke dette certifikat**, fordi de har PKca. Og derved senere sende beskeder krypteret med PKa

4.2 Sikkerhed

Sikkerheden ved at bruge en CA, ligger i - at vi **alle stoler på den ikke uddeler certifikater på flaske grundlag**.

Derfor kan parter nu dele certifikater, tjekke hinandens - og kommunikere frit med hinandens keys som kryptering.

5. Certificate Chains

I den **virkelig verden** er der dog mere end en CA, og man kan komme ud i en situation hvor man **får et certifikat fra en CA man endnu ikke stoler på**.

5.1 Eksempel, untrusted CA

Hvis et certificat fra en CA1 betegnes:

> *CERTca1(A, PKa)*

Så kan en CA bekræftes:

1. A har et certificat fra CA1
2. B har et fra CA2.
3. A modtager CERTca2(B, PKb) hvilket han ikke kan bekræfte da han ikke kender CA2.
4. A modtager CERTca1(CA2, PKca2)
5. A kan nu bekræfte CA2 og derved bekræfte B

> - *A: CA1*
> - *B: CA2*
> - *--> A: CERTca2(B, PKb)*
> - *--> A: CERTca1(CA2, PKca2)*
> - *A: CA1, CA2*

Det kaldes en kæde ved: CERTca2(B, PKb), CERTca1(CA2, PKca2).... da det jo vil kunne fortsætte længe.

Dette giver en **observation**: Hvordan **stoler** vi på den **første** CA/KDC?

Som før nævnt, må de (ofte) **ske fysisk**.

6. Tilkendegivelse

Men det dur ikke for mennesker at gå rundt med certifikater at huske på.

Så vi bruger nogle andre løsninger til at tilkendegive os selv.

6.1 Kodeord

Anses som det **svageste af alle** tre metoder, da et kodeord netop skal huskes af et menneske; og for at et menneske kan dette; vælger det **ofte nemme koder**; eller de **skriver dem ned** - begge dele der sænker sikkerheden.

> *Kodeord skal være stærke*

En fremmed skal ikke kunne gætte det. F.eks at basere det på anden **personlig information anses som en dum ide.**

6.1.1 Sikkerhed

6.1.1.1 KVALITET

En kode skal bestå af et sæt af tegn af størrelsen C og have en længde L, så vil der være C^L forskellige kodeord.

> - *Antal lovlige tegn: C*
> - *Kodeord længde: L*
> - *Antal kodeord: C^L*

Hvilket vokser hurtigere af L end af C. Så det er bedre at have en lang kode, end mange forskellige muligt tegn.

Men godt at have en lang kode med mange forskellige tegn.

Studier viser brugere normalt kan huske 12 tegn som maks.

En bruger kan også vælge at have en **passphrase istedet**, der anses for at være nemmere at huske.

6.1.1.2 SERVER

> - *Koder skal **sendes sikkert***
> - *Koder skal ikke opbevares, men kun deres **fingerprint***
> - *Hjælp brugeren med at **vælge stærke** kodeord*
> - ***Sløv** en angriber **ned***
> - ***Opdel serveren** i dele der krypter koden og godkender koden; så en fremmed ikke får adgang til begge dele*

6.2 Hardware

En anden måde en bruger kan tilkendegive sig selv på, er ved at **bruge noget hardware**. F.eks en usb nøgle.

6.2.1 Tamper Evident

Hardware der er svær eller tidskrævende at bryde ind i, kaldes: Tamper Evident.

Her kan vores chip-dankorts f.eks nævnes; der i sig selv er små sikre computere.

6.2.1.1 TWO FACTOR AUTHENTICATION

Tamper Evident kan også bruges til TFA.

Ideen er at godkende en bruger i to stadier; først via password og så at han har den rigtige hardware.

Der er en SK i hardwaren og samme SK er i verifyeren. Verifieren sender så en nonce. Hardwaren returnere så R(sk, c) Som verifieren så tjekker

> - *Hardware og server har samme secrete-key*
> - *Server --> Hardware: NONCE*
> - *Hardware --> Server: Responce(Sk, c)*

6.2.2 Tamper Resident

Er langt sikre hardware, og bruges af CA og banker f.eks.

6.3 Biometrics

Der kan bruges

> - *Fingeraftryk*
> - *Ansigtscanning*
> - *Øjescanning*
> - *Stemme osv...*

Alle virker ved at blive omdannet til noget digitalt data, som så bliver matchet med en entry i en database.

6.3.1 Buffer til forandring

Det store problem her; er at systemet skal have en hvis buffer i forhold til at vi som levende organismer konstant er i forandring; men samtidig aldrig lade en fremmed komme ind der forsøger at udgive sig for os.