Co-funded by the Erasmus+ Programme of the European Union

InCyT træningsmodul

Informationssikkerhed for medarbejdere Enhed 2 Uge 3 Enhedens navn: Introduktion til kryptografi	
Læringsaktiviteter	❑ **Webinar** ❑ **Øvelser** ❑ **Værksted** ❑ **Præsentation** ❑ **Andet**
Læringsansvarlig	Claudio Fornaro
Læringsaktivitetsmål	1. Kryptografi 2. Kryptoanalyse 3. Engangs-pad-algoritme 4. Symmetriske algoritmer 5. Asymmetriske algoritmer 6. Hash funktioner 7. Signaturer og offentlige nøgleinfrastrukturer 8. Kryptografiske tokens 9. Udfordring-respons mekanisme
Færdigheder, der skal opnås	• Grundlæggende viden om kryptografiske algoritmer og infrastrukturer • Teknisk færdighed 2 - TS2
Varighed af læringsaktivitet	En uge
Læring krav	***Hvad skal der til?***
	• Der kræves ingen forudgående specifik viden

Co-funded by the
Erasmus+ Programme
of the European Union

Kort information om modulet

Kryptografi vedrører metoder til at skjule information fra videregivelse, både under en kommunikation og til arkiveringsformål. Dens mål opnås med design og implementering af protokoller, der forhindrer en uautoriseret tredjepart i at hente oplysningerne. I en sikker kommunikation kan modstanderens tredjepart ikke få adgang til de transmitterede oplysninger, i tilfælde af arkivering kan tredjemand ikke spore indholdet af det arkiverede.

I kryptografi er tredjeparten altid en ondsindet enhed, der har til formål at hente oplysninger, som den ikke må have adgang til, hvad enten det er af hensyn til privatlivets fred, forretningshemmeligheder eller national sikkerhed. Datafortroligheden og -integriteten, autentificeringen af de involverede parter og ikke-afvisningen af, hvad datakilden har transmitteret eller lagret, er de grundlæggende principper for moderne kryptografi.

Efter en indledende del, hvor de grundlæggende begreber er beskrevet, præsenterer vi en progression af grundlæggende kryptografiske teknikker, startende fra *One Time Pad Algorithm* og derefter opbygning for at overveje de to kryptografiske familier af algoritmer: Symmetriske algoritmer og Asymmetrisk (eller Public Key) Algoritmer med en sammenligning af dem. Hash-funktionernes rolle diskuteres også for at introducere digitale signaturer. En beskrivelse af nøglekonceptet for Public Key Infrastructures er præsenteret i dybden med nogle detaljer om standardmekanismerne og -formaterne. Som konklusion er der givet nogle oplysninger om kryptografiske tokens og udfordrings-svar-mekanismen.

Lærematerialets indhold

Co-funded by the
Erasmus+ Programme
of the European Union

Introduktion

Kryptografi er den mekanisme, hvorved indholdet af en meddelelse er skjult for andre enheder. Med andre ord omhandler kryptering kryptering af beskeder, dvs. at tage en klartekst, ved hjælp af en algoritme og generelt en nøgle, hvilket gør indholdet af beskeden uforståelig.

Krypteringsanalyse undersøger, hvordan man dekrypterer beskeden uden at kende nøglen. I et angreb består kryptoanalyse i at forsøge at spore indholdet af beskeden uden at have nøglen. Kryptografer og kryptoanalytikere er faktisk de samme mennesker: hvem der skriver de kryptografiske algoritmer, analyserer dem også for at finde deres svagheder eller validere deres robusthed. Som altid, når man beskæftiger sig med sikkerhed, er ideen om at kunne håndtere beskyttelsesproblemer uden at vide, hvordan angrebsteknikker fungerer, ikke særlig effektiv.

Information i meddelelser er generelt overflødige, i formel forstand bruger meddelelser generelt et meget højere antal bits end det, der reelt er nødvendigt for at bære informationen. **Informationsredundansen letter krypteringsanalysen** : redundans i informationen betyder, at meddelelserne kan vise en struktur, der gør det nemmere at spore indholdet af meddelelsen, da ikke alle meddelelser er lige sandsynlige. Hvis vi tænker på en tekst repræsenteret med normale ASCII-tegn i byte, ved vi, at kombinationerne af bit, der kan repræsentere meningsfulde meddelelser, er relativt få sammenlignet med alle mulige kombinationer. De sande beskeder, der kan repræsenteres med et vist antal bytes, er meget få. Dette gør det muligt at identificere gyldige meddelelser ved at kassere dem, der ikke indeholder sand information. Alt dette hjælper i høj grad på aktiviteten af kryptoanalytikeren, så formålet med krypteringsalgoritmen er også at skjule denne redundans for bedre at skjule informationen.

Eksempel: Indkodning af *en sekvens* af de fire farver BLÅ, GRØN, RØD og GUL.

Disse kan kodes:

- med 2 bit (00, 01, 10, 11)
- som 4 forskellige store bogstaver
- kryptering af farvenavnene

I det første tilfælde bruges minimumsantallet af entiteter (bits), ved at erstatte hvert farvenavn med de tilsvarende 2 bits vil den originale sekvens skjules, men ikke rækkefølgen, da der er en overensstemmelse mellem FARVE og de 2 bits. I det andet tilfælde, da hvert bogstav bruger 8 bits i ASCII-koden, er 6 bits redundante for hver farve → 75 % redundans og vi opnår samme grad af skjul som de 2 bits. Vi kan stadig ikke forbinde farver og værdier.

Overvej følgende meget simple ciffer, den såkaldte *Cæsar-ciffer* , som ikke gør andet end at rulle bogstaverne i alfabetet med et vist antal positioner. For eksempel, hvis vi glider dem med 3, bliver

alle 'A'erne 'D'er, alle 'B'er' bliver til 'E'er, og så videre. Dette er en meget simpel chiffer; faktisk tilskrevet Julius Cæsar, men hvis farvenavnene er krypteret med denne metode, er det gentagne mønster af deres kodede bogstaver let at identificere (ASCII-koden for R efterfulgt af ASCII-koden for E efterfulgt af ASCII-koden for D er let at anerkende som en enhed). Så rækkefølgen kan identificeres, og hvis ordene også er kendt i dette tilfælde, er det nok at tælle antallet af bogstaver til at identificere de oprindelige ord.

Dette simple eksempel giver os mulighed for at forstå, at kryptoanalyse ikke er en analyse, der er uafhængig af indholdet af beskeden: en fuldstændig randomiseret besked ville være meget sværere at identificere og analysere end en besked, der har en genkendelig struktur.

Nogle typer kryptografiske angreb og algoritmer er afhængige af at kende en del af teksten. I nogle tilfælde gør det at kende en del af teksten det muligt at spore nøglen og derefter få adgang til resten af teksten. Dette er meget væsentligt, når det kommer til netværkskommunikation, for i netværkskommunikation har protokollerne på forskellige niveauer normalt en vis mængde information, der under alle omstændigheder er genkendelig, fx i en IP-pakke ved vi, at headeren indeholder afsenderadressen og modtageradresse.

En krypteringsalgoritme skjuler også redundant information, så at anvende en komprimeringsalgoritme på en krypteret meddelelse giver ikke den store fordel, fordi det ville reducere komprimeringshastigheden. Hvis vi vil bruge komprimering og kryptering, skal vi først bruge komprimering og derefter kryptering.

One Time Pad -algoritmen

One Time Pad (OTP) er en algoritme, så at sige, "perfekt". Det er en meget simpel krypteringsalgoritme, meget enkel at implementere og har en egenskab, der gør den særlig interessant: Det er bevist, at uden at kende nøglen er det umuligt at gå tilbage til den oprindelige tekst. Nu ser vi på det, og så diskuterer vi, hvorfor vi, da vi har så god en algoritme, skal finde andre.

EXCLUSIVE OR $\oplus$ (XOR) er en logisk funktion, hvor resultatet er Sandt, når de to operander er forskellige, det opererer på logiske værdier, men oversat til binære cifre, normalt True=1 og False=0, dens *sandhedstabel* er:

$0 \oplus 0 = 0$

$0 \oplus 1 = 1$

$1 \oplus 0 = 1$

$1 \oplus 1 = 0$

Fra denne tabel er det tydeligt, at hvis X ⊕ Y → Z, så Z ⊕ Y → X og Z ⊕ X → Y. EXCLUSIVE OR er en meget enkel funktion og også nem at implementere som et elektronisk kredsløb.

OTP-mekanismen bruges til at overføre en besked fra to parter. I den videnskabelige litteratur er det almindelig praksis at identificere skuespillerne med personnavne, de kaldes normalt Alice (A) og Bob (B). Så vi vil beskrive, hvordan Alice sikkert kan sende en *n* -bytes besked (*PlainText*) til Bob.

One Time Pad-algoritmen kræver en opsætningsfase:

- En tilfældig sekvens på nøjagtigt *n* bytes genereres, dette er *One Time Pad* (OTP) eller blot *nøglen* ;
- OTP'en er sikkert delt mellem de to dele på en eller anden måde (givet på forhånd, sendt i en allerede sikret kanal osv.).

Når begge parter har nøglen, kan de bruge den til at sende beskeder. Hvis Alice vil sende en besked til Bob:

- Alice beregner *lidt efter lidt* EKSKLUSIVT ELLER af *almindelig tekst* med *nøglen* og producerer en *CypherText* :
  Kryptering: PlainText ⊕ Nøgle → CypherText
- Den resulterende *CypherText* overføres af Alice til Bob over en muligvis usikker kanal
- Bob beregner *bit for bit* EKSKLUSIVT ELLER af *CypherText* og *Key* , så den originale *PlainText gendannes* :
  Dekryptering: CypherText ⊕ Nøgle → PlainText

Eksempel

Lad os antage, at både Alice og Bob allerede er blevet enige om en tilfældig OTP (*nøgle*).

Alice ønsker at sende "HEJ" til Bob (i ASCII-koden er mellemrum introduceret her for at lette læsningen):

	HEJ
Almindelig tekst :	01001000 01000101 01001100 01001100 01001111
	⊕
Nøgle :	10101001 01110010 10110010 10000110 11000110
	=
CypherText :	11100001 00110111 11111110 11001010 10001001

Bob modtager *CypherText* og anvender *nøglen* :

CypherText : 11100001 00110111 11111110 11001010 10001001

⊕

Nøgle : 10101001 01110010 10110010 10000110 11000110

=

Almindelig tekst : 01001000 01000101 01001100 01001100 01001111

HEJ

PlainText *er* blevet gendannet.

Da *nøglen* er en tilfældig sekvens, er der ingen måde at sige, om en bestemt bit af sekvensen er '0' eller '1', da de er lige sandsynlige. Uden *nøglen* , givet en *CypherText*- bit, har vi ingen chance for at forsøge at gætte, hvad den originale bit kan være. Dette gælder for enkelte bits, for hele beskeden, for ethvert fragment af vores besked.

Hvis en del af *nøglen* er kendt, er det muligt at gendanne den tilsvarende del i *almindelig tekst* , men uden resten af *nøglen* er der ingen mangel på information om de andre bits af *almindelig tekst* . Derfor er den førnævnte kryptoanalyse ikke effektiv. I dette tilfælde er den resterende del af *nøglen* nødvendig for at finde den resterende del af den originale besked . Al den information, der formidles af *nøglen* , er nødvendig for at gendanne al informationen i teksten.

One Time Pad problemer

Styrken af OTP-mekanismen er afhængig af, at *nøglens længde* er nøjagtigt lig med længden af teksten, og at den kun bruges én gang. Den største ulempe er, at transmissionen af OTP'en har de samme problemer som meddelelsen.

I nogle tilfælde er dette problem let at løse, for eksempel kan en diplomatisk dokumentmappe med et bånd indeholdende en meget lang nøgle leveres til en ambassade, og efterfølgende kan de beskeder, der vil blive udvekslet med den ambassade, krypteres ved gradvist at forbruge nøglens bits . Disse er meget ejendommelige anvendelser, men ret effektive. Det er klart, at hvis vi overvejer netværkskommunikation, er den reelle anvendelse af One Time Pad meget begrænset. Der er brug for noget mere fleksibelt, noget der tillader udveksling af en nøgle, der kan bruges i meget lang tid.

Lad os overveje hvert af problemerne.

Offline transmission på forhånd

Dette er et generelt problem med nøgler: nøgler skal aftales på en eller anden måde af parterne. Dette behov for at blive enige om nøgler er det vigtigste problem ved brug af kryptografi. Problemet med kryptografi er ikke så meget algoritmer og kryptoanalyse, såvel som nøglehåndtering: det er at være sikker på, at alle og kun de rigtige personer har adgang til nøglerne, så udveksle dem på forhånd og være sikker på, at de udvekslede nøgler er rigtige.

Engangsbrug

One Time Pad bruges kun én gang, som navnet antyder, dette er et vigtigt aspekt og begrænsning. Hvis vi forsøgte at bruge den samme nøgle på to beskeder, ville vi svække metodens sikkerhed.

Det er meget vanskeligt at generere tilfældige tal i et deterministisk system

Computere til generelle formål bruger generering af *pseudotilfældige* tal, hvor hvert tal er beregnet fra det foregående med en vis beregning, dette producerer en sekvens af tal, der gentages efter en meget lang periode. Disse er ikke sande tilfældige tal, da en computer er en deterministisk maskine. I kryptografi er denne funktion imidlertid skadelig. Hvis det er muligt at udlede det næste tilfældige tal fra et tidligere, bliver beskyttelsen fra kryptografiske algoritmer eller protokoller meget mindre sikre. At skrive kode, der implementerer kryptografiske algoritmer, er ikke trivielt: der er mange detaljer at holde øje med, alle de fejl, der er blevet identificeret gennem årene af kodeudviklere, som kun kodeudviklerne i dette specifikke område har lært at håndtere. Det betyder, at det altid er en dårlig idé at udvikle og/eller implementere in-house kryptografiske algoritmer i stedet for at bruge velafprøvede, meget brugte, omfattende validerede biblioteker.

Andre algoritmer end OTP

De andre algoritmer bruger nøgler med fast længde. Hvis vi bruger en *n* bit nøgle, har vi 2^n forskellige nøgler, hvis værdien af *n* er stor nok, og du bare anvender et brute force angreb (det vil sige at prøve alle de mulige nøgler), ville det tage så mange år at finde højre, så "sandsynligvis" indholdet af beskeden ikke længere er af interesse. For eksempel, med en af de hurtigste supercomputere til rådighed, vil kontrol af en 256 bit nøgle kræve $3{,}67\times10^{55}$ år. Det er klart, at brug af nøgler af tilstrækkelig længde har nogle ekstra beregningsomkostninger, selv for dem, der lovligt krypterer og dekrypterer, men virkningen er begrænset.

Der er symmetriske og asymmetriske algoritmer.

Symmetriske algoritmer

Den mest typiske og gamle familie af kryptografiske algoritmer, dem vi typisk tænker på, når vi tænker på kryptering af meddelelser, er de ***symmetriske algoritmer*** , kaldet efter symmetrien af krypterings- og dekrypteringsoperationerne. Disse algoritmer bruger en enkelt nøgle, der skal være kendt af både afsender og modtager; den bruges til at kryptere en besked og derefter til at dekryptere den.

Et simpelt symmetrisk skema kunne være det samme som OTP, da OTP er en symmetrisk algoritme, med den store forskel, at *nøglen* har en fast længde (f.eks. 256 bit), og den gentages på en eller anden måde, muligvis efter en kompleks ændring hver gang :

Kryptering: *encryption_function* (PlainText, Key) → CypherText

Dekryptering: *decryption_function* (CypherText, Key) → PlainText

Mere generelt, selvom vi bruger den samme *nøgle* , er den funktion, der bruges til at dekryptere, ikke nødvendigvis den samme, der bruges til at kryptere. For eksempel, hvis vi betragter Cæsars chiffer, er det klart, at funktionen, der dechifrerer beskeden, glider alfabetet til venstre og ikke til højre, men dekryptering bruger den samme nøgle: nummer 3.

<u>Nogle vigtige symmetriske nøglealgoritmer</u>
DES (Data Encryption Standard)

Denne historiske algoritme, nu forældet, bruger en 56-bit nøgle, der i slutningen af 1970'erne (da algoritmen blev udviklet) var absolut tilstrækkelig til mange anvendelser, fordi et brute force angreb på en 56-bit nøgle for at få beskeden med computeren ressourcer, der var tilgængelige på det tidspunkt og i de sammenhænge, hvor denne algoritme blev brugt, var upraktisk. 40 år senere ville DES-algoritmen ikke vare et par minutter af en hjemmecomputer. Dette er et meget vigtigt aspekt: Nøglens længde vælges ud fra de *faktiske* tilgængelige computeregenskaber. En nøgle, der anses for robust nok nu, er måske ikke længere så stærk om 20 år. Generelt har vi en tendens til at bugne af nøglens længde for at sikre, at der er en vis margin, både fordi computerkapaciteten øges, men også fordi kryptoanalyse som disciplin gør fremskridt og måske kan finde svagheder i algoritmen. Dette er et ganske interessant aspekt. I modsætning til hvad man skulle tro, fejler en kryptografisk algoritme normalt ikke pludseligt, dvs. den dag en fejl opdages i en algoritme, bliver den øjeblikkeligt fra sikker til fuldstændig usikker, og enhver kan finde klarteksten på et øjeblik. Det, der sker, er, at forskere analyserer algoritmen og måske har fundet små svagheder, der kan lette at angribe den i visse tilfælde. Så er der muligvis nogen, der har fundet en algoritmeoptimering for at løse problemet. Som tiden går, bliver algoritmen gradvist svækket på grund af uløselige problemer, indtil den ikke længere er tilstrækkelig til bestemte anvendelser eller overhovedet. Der er en progressiv udhuling af forsvaret af kryptografiske algoritmer, og denne progressive svækkelse er et

vanskeligere problem at håndtere, fordi vi naturligvis ikke kan vide, hvilke fremskridt der vil være i fremtiden ud fra et kryptoanalyses synspunkt.

AES (Advanced Encryption Standard, også kendt under sit oprindelige navn Rijndael)

AES blev valgt ud fra ikke kun robusthed, men også på effektiviteten af implementering i typiske arkitekturer. Der blev ikke søgt nogen hemmelig algoritme, fordi en hemmelig algoritme ikke anses for at være mere robust end en ikke-hemmelig algoritme: Tværtimod var algoritmen beregnet til at blive analyseret af så mange værdifulde kryptoanalytikere som muligt for at sikre, at den var iboende robust. En algoritme, der er sikker, men for langsom, for dyr i forhold til computerressourcer, ville ikke være brugbar. Denne begrænsning blev sat til at være den sværeste for konkurrenter at overvinde sammenlignet med robusthed.

Asymmetriske (eller offentlig nøgle) algoritmer

Asymmetriske algoritmer bruger ikke kun én nøgle, men to: én nøgle bruges til at kryptere og den anden til at dekryptere. Ved udveksling af beskeder bruger afsender og modtager derfor ikke den samme nøgle. Disse nøgler genereres og administreres i par : hvad en nøgle (enhver af parret) krypterer, kan kun dekrypteres af den anden nøgle i parret.

Grunden til at bruge asymmetriske algoritmer stammer fra problemerne med symmetrisk kryptografi:

1. Fortrolig kommunikation mellem 2 enheder kræver, at en unik nøgle kun deles mellem dem, hvis enhederne er n , kræves $n \cdot (n-1)/2$ forskellige nøgler for at tillade fortrolig kommunikation mellem hvert par af parterne
2. Både afsender og modtager kender deres delte nøgle, så *ikke-afvisning* (afvisning er at benægte at være forfatter til en bestemt besked) er ikke mulig

Det første problem handler ikke om antallet af nøgler, som hvert fag skal administrere, problemet er *at udveksle* dem. Hvert emne skal aftale en nøgle med hvert af de andre n-1 emner, hvilket faktisk er en kompleks operation; dette gør symmetrisk kryptografi upraktisk til mange anvendelser.

Det andet problem, *ikke-afvisningen* , ligger i det faktum, at vi ofte ønsker at sikre os, at en bestemt besked faktisk blev sendt af et bestemt emne. Det er klart, at hvis et emne modtager en besked fra et andet emne, og de to emner er de eneste, der kender nøglen, der bruges til at kryptere den, er hvert emne sikker på, at det andet emne er forfatteren til meddelelsen. Omvendt, da de kun kender nøglen, ved hvert emne, at kun det andet emne kan læse beskeden. Problemet opstår, når der er behov for at bevise dette over for en tredjepart, for en eller anden form for retssager, da denne mekanisme ikke virker: hver part er i stand til at oprette en besked, kryptere den og derefter sige,

at den anden part er forfatter. Ikke -*afvisningsproblemet* kan ikke løses ved at bruge symmetrisk kryptografi.

Det asymmetriske skema ligner det symmetriske, men de anvendte nøgler er de to, der genereres i par:

Kryptering: *encryption_function* (*PlainText* , ***Key1***) → *CypherText*

Dekryptering: *decryption_function* (*CypherText* , ***Key2***) → *Simpel tekst*

Navnene ***Key1*** og ***Key2*** er blevet brugt i stedet for *EncryptionKey* og *DecryptionKey* , fordi de to nøgler er matematisk udskiftelige. Emnet, der genererer nøglerne, holder generelt den ene privat/hemmelig (kaldet den *private nøgle*) og gør den anden offentlig (kaldet den *offentlige nøgle*), deraf navnet på algoritmeklassen, der også kaldes den *offentlige nøglealgoritme* . Mere om udtrykket *offentlighed* vil blive diskuteret senere.

Den funktion, der bruges til at kryptere, er ikke nødvendigvis den samme som den, der bruges til at dekryptere. Det vigtige er, at de tilsammen repræsenterer den kryptografiske algoritme, den ene del er afhængig af den ene nøgle og den anden del er afhængig af den anden nøgle.

De to nøgler har så matematiske egenskaber, at opdagelse af den ene nøgle fra den anden er ret kompleks og kræver enormt lang tid. Især selv at kende den offentlige nøgle, *PlainText* og *CypherText* , ville rekonstruering af den private nøgle kræve urimelig lang tid.

<u>Anvendelser af asymmetrisk algoritme</u>

Antag (detaljer vil blive givet senere), at den *offentlige nøgle* er offentligt tilgængelig, og dens ejer er den eneste, der kender den tilsvarende private nøgle. Da den er den offentlige nøgle, der er tilgængelig for enhver, kan enhver bruge den til at kryptere en besked, men kun ejeren af den tilsvarende private nøgle vil være i stand til at dekryptere beskeden, dette giver **fortrolighed** af beskedtransmissionen.

Så vi har bevæget os fra en situation, hvor et emne havde brug for *n* forskellige nøgler for fortroligt at kommunikere med *n* andre emner til en situation, hvor der kun kræves én nøgle (modtagerens offentlige nøgle), for at alle kan sende en fortrolig besked til et emne.

For at tillade *n* emner at kommunikere med hinanden, skal hvert emne have et par nøgler, så det samlede antal nøgler er $2 \cdot n$. Det interessante er ikke så meget, at nøglerne er faldet i antal; men fordi de er offentlige nøgler, kan disse kommunikeres på en meget mere enkel måde. Hvis en anden lærer disse offentlige nøgler at kende, er det overhovedet ikke et problem. Derfor kan disse nøgler virkelig publiceres, gøres tilgængelige for alle og alle kan bruge dem uden at svække beskyttelsen af beskederne.

Hvis emnet er den eneste ejer af en privat (hemmelig) nøgle, hvis offentlige modstykke er offentligt tilgængelig, kan enhver bekræfte (dekryptere) meddelelser krypteret med emnets private nøgle. Så hvis det er muligt at bruge den offentlige nøgle for et emne til at dekryptere en meddelelse, betyder det, at kun emnet , der har den tilsvarende private nøgle, kunne have krypteret meddelelsen, dette giver mulighed for **ikke at afvise** meddelelsen.

Et problem skal stadig løses: at sikre, at en offentlig nøgle virkelig er forbundet med det legitime emne. Her skal der leveres en *super partes* certificeringsmetode, denne leveres af en *Public Key Infrastucture* (PKI), diskuteret senere.

Som en sidebemærkning kan asymmetriske algoritmer også bruges som en *udfordring-respons-algoritme* . For at subjekt A skal *autentificere* (bevise, at et subjekt reelt er den, der hævder at være) subjekt B, sender A (i det klare) et tilfældigt tal til B; så krypterer B den med sin private nøgle og sender den tilbage til A. Hvis A er i stand til at dekryptere beskeden med den offentlige nøgle af B, er A sikker på, at den anden part er B, da det er den eneste, der kan kryptere noget den offentlige nøgle af B kan dekryptere.

Fordele og ulemper

Det samlede antal nøgler er væsentligt reduceret, hvilket mindsker kompleksiteten af at garantere den korrekte håndtering af alle nøglerne (og kun ejeren kender den hemmelige nøgle). Da offentlige nøgler gøres offentligt tilgængelige, er der ikke behov for specifikke aftaler mellem forskellige emner. Denne brug af offentlige nøgler er for eksempel meget vigtig i e-handel, fordi e-handel ikke kan kræve, at der er en forudgående aftale af sælger med kunder.

For at opnå denne forbindelse mellem de to nøgler og samtidig garantere, at den ene ikke kan beregnes ud fra den anden, og at det, der krypteres med den ene, dekrypteres med den anden, gør asymmetriske algoritmer meget mere komplekse med hensyn til matematiske problemer end symmetriske algoritmer. Disse matematiske problemer kræver i det væsentlige langsommere algoritmer end symmetriske algoritmer og brugen af længere nøgler, i det mindste med de aktuelt anvendte algoritmer. Derfor, hvis der på den ene side er en større fleksibilitet, er de på den anden side mere besværlige med hensyn til computerressourcer.

Fremtrædende algoritme: RSA *(fra initialerne af Ronald* ***R*** *ivest, Adi* ***S*** *hamir og Leonard* ***A*** *dleman)*
Denne algoritme er baseret på kompleksiteten af primtalsfaktorisering: det er nemt at gange to primtal og opnå resultatet, men det er komplekst/uoverskueligt at finde de to primtal ud fra resultatet. Primfaktorisering er et vanskeligt problem. Vi ved også, at det ikke er let at finde meget høje primtal, da der ikke er nogen regel, der tillader at finde ud af primtal ud over at forsøge at faktorisere dem og finde ud af, om de er primtal. Det betyder, at arbejdet med meget store tal, at finde ud af, hvilke to primtal, som et givet tal er afledt af, er et problem, der kræver betydelige

computerressourcer. På den anden side kan legitime operationer udføres med begrænsede computerressourcer.

Fremtrædende algoritme: Elliptisk kryptografi *(ECC - Elliptic Curve Cryptography)*

Det "vanskelige problem" i dette tilfælde er beregningen af den diskrete logaritme (i et begrænset felt). Et "finite field" betyder, at heltalværdier er i et begrænset sæt (meget grov definition).

ECC-problemet er mere komplekst end primfaktorisering, så du får den samme sikkerhed med kortere nøgler. ECC-algoritmer er hurtigere end RSA.

Sammenligning af symmetriske og asymmetriske algoritmer

Asymmetriske algoritmer udnytter andre matematiske problemer end symmetrisk kryptografi. Det betyder, at nøglelængderne ikke er sammenlignelige med symmetrisk kryptografi. Problemet med symmetrisk kryptering er bare, at nøglen er lang nok til at forhindre forsøg på alle mulige tal, der kunne matche en nøgle, så med en 128-bit nøgle er der 2^{128} (det vil sige $3{,}4 \cdot 10^{38}$) mulige nøgler. Nutidens symmetriske algoritmer bruger typisk 128-, 256- og nogle gange 512-bit nøgler.

Hvis vi for eksempel betragter RSA, som udnytter primtal, er det klart, at ikke alle nøgler er mulige, fordi de er relateret til tilstedeværelsen af primtal. Ikke alle tal er primtal, så ikke alle tal er mulige nøgler. Meget forenklet, for at have den samme algoritme robusthed (derfor den samme vanskelighed fra et beregningsmæssigt synspunkt) som en 128-bit symmetrisk algoritme, har RSA algoritmen brug for en nøgle i størrelsesordenen 1024 bit. Nutidens asymmetriske algoritmer bruger typisk 1024, 2048 og 4096 bit nøgler.

Hash funktioner

Hash-funktioner beregner en kort oversigt med fast længde af en tekst af vilkårlig længde; dette resumé kaldes en *digest* . En hash-funktion skal være enkel og hurtig at beregne.

Sikkerhed ejendomme

Hash-funktioner er "envejsfunktioner" i den forstand, at det er meget komplekst og langsomt fra fordøjelsen at gendanne en *mulig* tekst, der genererer den, men denne tekst er næppe brugbar. Dette er en konsekvens af det faktum, at den originale tekstdigest er (og bør være) større end digest, så praktisk talt uendelige sekvenser af bytes kan resultere i en specifik digest. Selvom en tekst viser sig at have den givne sammenfatning, er det yderst usandsynligt, at det er den originale tekst eller endda giver mening. Det er ekstremt vanskeligt at bygge to meningsfulde beskeder med

den samme digest eller opbygge en meningsfuld besked ud fra en given digest, enhver ændring af en besked, selv en enkelt bit, vil give en helt anden digest (ca. 50% af dens bits er forskellige).

Integritetstjek

Givet en besked M, beregnes dens digest D ved hjælp af en hashfunktion H: D = H(M)

En sikker transmission af sammendrag D gør det muligt at sikre, at meddelelse M er korrekt: modtageren af meddelelse M beregner sammendraget på den og sammenligner den med den (D) modtagne; hvis de er identiske, er meddelelsen ikke blevet ændret. Det er vigtigt, at fordøjelsen overføres på en sikker måde, ellers er der ingen beskyttelse overhovedet.

Denne mekanisme bruges også som en verifikation af en downloadet software i stedet for andre mindre kraftfulde (men hurtigere) algoritmer kendt som CRC'er (Cyclic Redundancy Checks), der er almindelige i netværksprotokoller.

Nogle vigtige Hash-algoritmer

Følgende er to af de mest kendte hash-algoritmer.

MD5 (Message Digest #5, 1991)

MD5 kræver en lav beregningskraft (sammenlignet med størstedelen af de andre hash-algoritmer) og producerer en 128-bit hash. Med tiden er den gradvist blevet svækket til at være fuldstændig upålidelig af sikkerhedsmæssige årsager (på nuværende tidspunkt er det muligt at finde to adskilte beskeder med den samme MD5-hash – som kaldes en *kollision* – på sekunder), alligevel er det stadig nyttigt i ikke-kryptering applikationer, for eksempel til at opdage transmissionsfejl i stedet for den mere almindelige (og svage, men hurtigere) CRC-32.

SHA-3 (Secure Hash Algorithm #3, 2015)

SHA er faktisk en familie af sikre hash-algoritmer, SHA-3 er den sidste version og resultatet af en konkurrence blandt kryptografer. Algoritmen kan tunes balancerer sikkerhed med hastighed; det producerer hash-værdier på 224, 256, 334 og 512 bit. Det er 2-3 gange langsommere end MD5, men der er indtil videre ikke fundet nogen svaghed.

Signaturer (digitale signaturer)

Digitale signaturer, kaldet elektronisk signatur i nogle landes lovgivning, tager en besked, beregner en hash og krypterer den (kun hashen) med den hemmelige nøgle til en asymmetrisk algoritme. Denne hash signeret med den hemmelige nøgle kaldes en *signatur* . En digital signatur krypterer så ikke den originale besked, tillad blot at garantere, at den tilsvarende tekst ikke er blevet ændret, for ellers ville sammendraget ændre sig. Det garanterer også, at oprindelsen kan verificeres (*ikke afvisning*), da den faktisk blev krypteret med den private nøgle af en asymmetrisk algoritme. Det

betyder, at enhver kan verificere, at teksten er intakt og faktisk er blevet genereret af et bestemt emne.

Svarende til digitale signaturer er **Message Authentication Codes** (MAC'er). De har også til formål at sikre integriteten og ægtheden af et budskab. Sammenfatningen er beregnet, der hashderer teksten, der skal beskyttes , og en *symmetrisk algoritmenøgle* (der er ingen garanti for ikke-afvisning, så den er kun egnet, når denne egenskab ikke er nødvendig). En af de mest kendte algoritmer er HMAC: Keyed-hash MAC.

Offentlige nøgleinfrastrukturer

Fortroligheds-, integritets- og autenticitetskravene til asymmetrisk kryptering og digital signaturoperationer kræver etablering af et tillidsforhold mellem de involverede parter. Da parterne ofte ikke havde nogen kontakt før, skal en betroet (af dem) tredjepart give de nødvendige garantier, med andre ord skal denne tredjepart sikre ægtheden af ejeren af en offentlig nøgle.

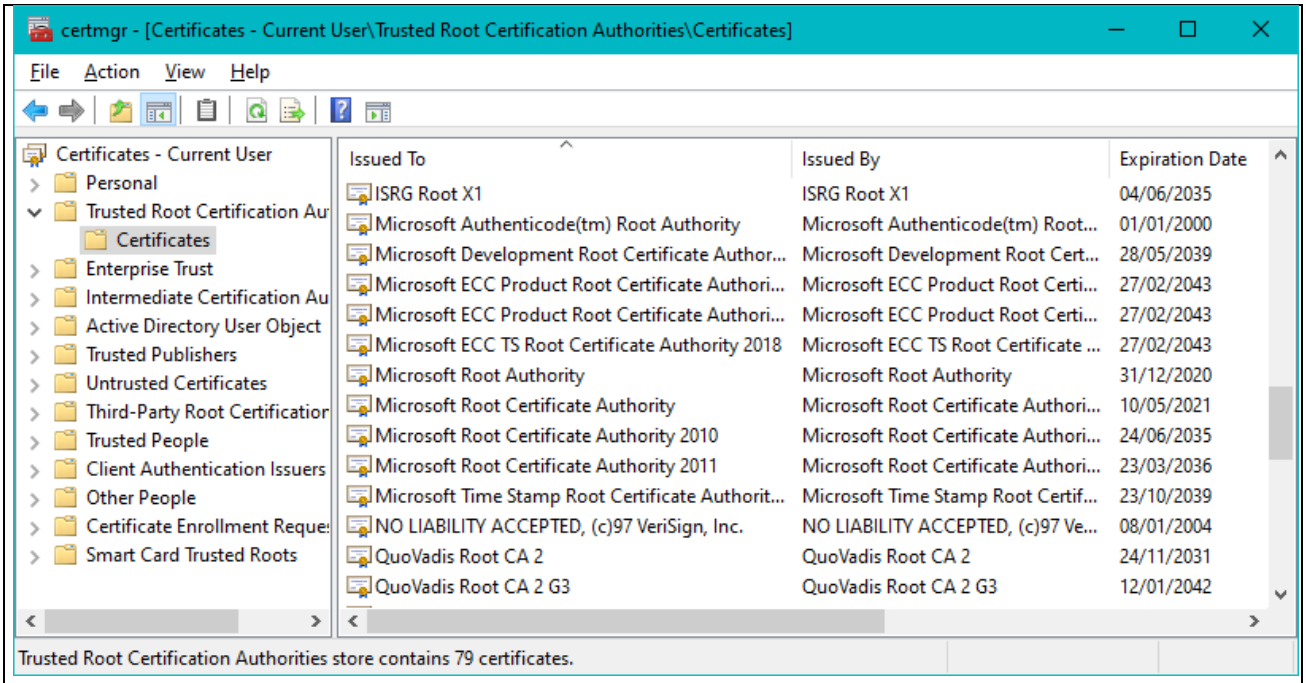
En *Certificate Authority* (CA) er en del af en hierarkisk tillidsmodel, der garanterer for brugerens identitet. Det skal anerkendes som "betroet" af alle enheder, der er involveret i kommunikationen.

For at sikre ægtheden af et emne, underskriver en CA et *digitalt certifikat* , der forbinder emnet med dets offentlige nøgle (det ligner konceptuelt et identitetsdokument for en person). Signaturen anvendt af CA er en digital signatur, så den beregnes ved at bruge CA's private nøgle. Det digitale certifikat skal naturligvis valideres ved at bruge CA's offentlige nøgle.

En *Public Key Infrastructure* (PKI) er en hierarkisk struktur af CA'er, hvor hver CA har sin offentlige nøgle signeret af det tidligere niveau CA (som bruger sin private nøgle til dette formål), processen går op til *Root Certificate Authority* , der selv- underskrive et certifikat med dets offentlige nøgle ved at bruge dets private nøgle.

En hierarkisk model gør det muligt at opbygge hierarkier af enhver dybde. For eksempel kunne vi inden for et land have de kommunale, …, regionale certificeringsmyndigheder, op til en national rod-CA.

Rot-CA-certifikatet skal offentliggøres på mange steder og måder for at gøre det umuligt at erstatte det med et forfalsket. Som et eksempel er selve operativsystemerne leveret med et sæt rod- og betroede mellemliggende CA'er. Figur 1 viser et uddrag af listen over rodcertifikater fra et MS Windows 10-operativsystem.



Figur 1. Windows 10 Certificate Manager

Komponenter af en PKI

Hovedaktøren er **Certification Authority (CA)** , den administrerer PKI's kryptografiske nøgler og certifikater. Det er den komponent, der har de rigtige nøgler, og som derefter udfører de kryptografiske operationer. I denne forstand er det naturligvis også det mest kritiske element i certificeringsmyndigheden.

Registration Authority (RA) administrerer certifikatanmodninger og leverer udstedte certifikater. Det beskæftiger sig med distribution af kryptografiske tokens og software (det er front-end af CA). Det er for eksempel som en skranke, som en borger kan gå til for at anmode om et digitalt certifikat, og hvorfra det kan fås. Det kan være en fysisk tæller eller en hjemmeside, typen er forbundet med forskellige sikkerhedsniveauer. Derudover kan en RA generelt også hjælpe brugeren med at generere nøgleparret og administrere disse certifikater sikkert, for eksempel ved at levere ad hoc software eller hardwareværktøjer, der understøtter ansøgeren i at generere signaturer, kryptere dokumenter, verificere signaturer og dekryptere information.

En **certifikatserver** er et bibliotek, hvor brugercertifikater udgives og på en eller anden måde gøres søgbare. Biblioteket følger X.500-standarden og forespørges ved hjælp af LDAP-protokollen.

Det digitale certifikat

Den digitale certifikatstruktur og -format er beskrevet af ITU-T X.509-standarden (det er en standard fra International Telecommunication Union nummereret X.509).

Et X.509 digitalt certifikat skal mindst indeholde følgende felter:

DN (adskillelsesnavn) bruger: Gaius Julius Cæsar

CN (almindeligt navn): Julius Cæsar

O (organisation): Regeringen

OU (organisationsenhed): Senatet

C (land): Romerriget

Dette er det klassiske mønster af et certifikat, der opfylder X.500-standarden, hvoraf X.509 er medlem af familien. Et certifikat kan have andre attributter (f.eks. e-mail). Der kan være en IP-adresse eller en URL, der svarer til en X.500 Directory-sti (f.eks. LDAP – Lightweight Directory Access Protocol):

IT / Romerriget / Regeringen / Julius Cæsar / Gaius Julius Cæsar /

Et X.509 digitalt certifikat skal også mindst indeholde følgende felter:

DN-udsteder : myndighedens navn i X.500-format (certificeringsmyndigheden har også sit eget fornemme navn, som fremgår af certifikatet. Det er klart, at det format, det repræsenteres med, er det samme som det fornemme navn på det emne, som certifikat gælder.

Serienummer : serienummeret på certifikatet, som myndigheden har (hvert certifikat er entydigt identificeret med det serienummer, som myndigheden har tildelt det. På den anden side identificerer parret (*serienummer* og *udsteders navn*) entydigt et certifikat i hele overflod af certifikater, der kan udstedes over hele verden som et unikt par).

NotBefore : aktiveringsdato

NotAfter : udløbsdato

Disse er to oplysninger, der er strengt relateret til hinanden. Et certifikat har derfor en begyndelsesdato og en udløbsdato, hvorefter det ikke længere er gyldigt. Årsagen stammer fra ønsket om at beskytte nøglerne mod overdreven brug.

Offentlig nøgle : den offentlige nøgle for det certificerede emne (certifikatet indeholder personlige data og den offentlige nøgle)

Offentlig nøglealgoritme: den algoritme, som den offentlige nøgle er egnet til

Signaturalgoritme : den algoritme, der bruges af myndigheden til at underskrive certifikatet

Signatur : signaturen påført certifikatet af certifikatmyndigheden (derfor det element, der garanterer ægtheden af selve certifikatet)

KeyUsage: de formål nøglen kan bruges til

Generering og tilbagekaldelse af certifikater

RFC 2314-standarden (fra PKCS # 10) definerer formatet for *certifikatanmodningsmeddelelsen* og indeholder ansøgerens offentlige nøgle og data (begge givet til en RA). Anmodningsmeddelelsen kræver det såkaldte *bevis for besiddelse* af den private nøgle: certifikatanmodningsmeddelelsen skal være underskrevet med ansøgerens private nøgle. Myndigheden verificerer ved at verificere denne signatur i det væsentlige, at ansøgeren faktisk besidder denne nøgle (ellers ville ansøgeren ikke have været i stand til at fremvise denne signatur).

På den dato, der svarer til "notAfter"-attributten, betragtes certifikatet som "udløbet". Udløbet er en mekanisme, der også tjener til at sikre, at de samme asymmetriske nøgler, der er knyttet til det digitale certifikat, ikke kan bruges i en for lang periode, hvilket kan udsætte dem for en række kryptoanalytiske angreb.

Udløbet kan henvise til:

- **certifikatet** (foreningens emnenøgle)
- **de tilknyttede nøgler** i forhold til den brug, de er beregnet til (en nøgle kan kun bruges til bestemte mål og ikke til andre)

Et udløbet certifikat kan ikke bruges af dets ejer, da det er udløbet, er det i praksis forbudt. I virkeligheden forhindrer intet indehaveren af certifikatet i at bruge det til andre formål, men alle dem, der kommunikerer med ham, vil på en eller anden måde opdage, at dette certifikat er udløbet og vil ikke acceptere det. Certifikatet er naturligvis tilgængeligt til at verificere og dekryptere meddelelser udstedt inden udløb. I nogle tilfælde er det muligt at "forny" et certifikat, hvilket normalt ikke er automatisk.

Nogle gange kan et certifikat gøres ugyldigt før udløb gennem *tilbagekaldelse* (dette sker før "notAfter" attributværdien. Årsagen kan for eksempel være, at nøglen, der er knyttet til certifikatet, er blevet kompromitteret, emnet er ikke længere knyttet til virksomheden eller ændret enhed, den CA, der har udstedt certifikatet (eller en af dets højere niveau CA'er), er blevet kompromitteret eller andre årsager. Et tilbagekaldt certifikat kan ikke længere bruges, undtagen når det bare holdes "på hold". Dette er en bestemt form. af tilbagekaldelse, hvor certifikatet midlertidigt er indsat i CRL'et på grund af specifikke årsager (såsom "mulig nøglekompromittering", "anmodning om tilbagekaldelse fra en anden person end ejeren", "midlertidig manglende brug af certifikatet" f.eks. til ferie)", osv. og dens gyldighed kunne genoprettes senere. I tilfælde af "genaktivering" fra

suspension, indsættes en ny post i CRL med en med en særlig årsagskode: *removeFromCRL* , da ingen post nogensinde kan slettes fra CRL. .

For at informere PKI-brugerne om, at et certifikat har mistet gyldigheden, er dets serienummer angivet i en *Certificate Revocation List* (CRL). Denne liste udsendes med jævne mellemrum af den udstedende certificeringsmyndighed og udbredes (aktivt eller passivt) til alle brugere, således at alle de personer, der har et sådant certifikat, ikke længere kan bruge det. CRL'en overholder ITU-T X.509-standarden - rfc3280 - rfc4325, hvilket betyder, at dens format er standardiseret. Den underskrives med jævne mellemrum af CA og inkluderer et felt kaldet "nextUpdate", som indeholder datoen og klokkeslættet, hvor den nye Certificate Revocation List vil være tilgængelig. Hvis denne dato hører fortiden til, betyder det, at vores CRL er "udløbet", hvis datoen hører til fremtiden, ved vi, hvornår vi skal opdatere vores liste over tilbagekaldte certifikater. Hvert tilbagekaldt certifikat indeholder årsagen til tilbagekaldelsen. CRL'en leveres som en service via LDAP-protokollen.

Da CRL'er kan blive ret store, især hvis certificeringsmyndigheden har hundredtusindvis eller endda millioner af brugere, kan de opdeles, og CA'en leverer bare dele af Certificate Revocation List, for eksempel med henvisning til en bestemt tidsperiode. Derfor, når det er nødvendigt at verificere et certifikat, downloader borgeren det eneste stykke CRL af interesse. I stedet for at downloade en CRL eller en del af den, kan en tjeneste, der implementerer *Online Certificate Status Protocol* (OCSP - RFC 2560) bruges til at finde ud af, om et bestemt certifikatnummer er i CRL'en. Svarene er underskrevet af CA for at være tillid

Tillidskæden

For at verificere et certifikat skal et emne verificere hele kæden af de mellemliggende CA'er startende fra en betroet, muligvis rod-CA. Først verificeres certifikatet, der skal kontrolleres, mod udsteder-CA, derefter verificeres CA-certifikatet mod dets moder-CA, og så videre op til en mellemliggende betroet (af brugeren) CA eller selve rod-CA.

Nøglebrug

Faktisk kan nøgler bruges til forskellige formål, og et certifikat angiver de tilladte anvendelser, blandt dem har vi:

- **nonrepudiation** : nøglen kan bruges til at producere en signatur med juridisk værdi, som derefter kan modstå tredjeparter som en håndskrevet signatur
- **dataEncipherment:** Nøglen kan bruges til at kryptere data
- **nøgleaftale:** nøglen kan bruges til sikker udveksling af nøgler

TimeStamp ing

En PKI kan også bruges til at tidsstemple et dokument i henhold til Time Stamp Protocol (RFC 3161) for at bevise dets eksistens på et tidspunkt i fortiden. En CA vil underskrive dokumentet og

inkluderer et tidsmæssigt mærke. Underskriften sikrer også, at dokumentet ikke er blevet ændret efter påsætning af tidsstemplet. Et tidsstempel kan forlænge gyldigheden af en elektronisk signatur ud over levetiden af det digitale certifikat, der er knyttet til de anvendte nøgler, for eksempel ved at anmode om påføring af et nyt tidsstempel på kontrakten hvert år, der går. Anbringelsen vil ske hvert år ved hjælp af kryptografiske teknikker med en styrke, der er passende for den periode, hvor selve signaturen anvendes .

Standardmekanismer og -formater

Følgende er de to almindelige standardmekanismer og -formater.

PKCS #7 - Offentlig nøgle kryptografistandard #7

PKCS #7 beskriver formatet af meddelelser/filer, der er signeret, krypteret (med symmetrisk og asymmetrisk kryptering) og verificerbar af MAC (fordøjede data)

S/MIME - Secure Multipurpose Mail Extension

Secure Multipurpose Mail Extension (RFC 3851 - SMIMEv3) er en udvidelse af MIME-standarden (RFC 2045, 2046 og 2047 som en udvidelse af RFC 822) og beskriver kryptering af mailmeddelelser. PKCS #7-enheder er i det væsentlige oversat til nye MIME-enheder og beskriver derfor nye containere, hvori vi kan finde dele af signerede eller krypterede e-mail-meddelelser eller signerede eller krypterede vedhæftede filer.

Udvekslings- og lagringsformater

Personal **Information Exchange Syntax Standard #12)** definerer et format for udveksling og sikker lagring (på fil) af kryptografiske nøgler og digitale certifikater (pålidelig certifikatkæde). Opbevaring er sikker, fordi den er baseret på kryptering af både nøglerne og certifikaterne og af hele kæden af certifikater, som et emne vælger at stole på. Det er det format, der bruges af browsere, e-mail-klienter og operativsystemer, når en bruger eksporterer en signatur- og krypteringsnøgle og et certifikat, for eksempel til en sikkerhedskopi eller for at kopiere det til en anden enhed.

En PKCS #12 krypterer dataene med en symmetrisk algoritme, hvis nøgle er afledt af en adgangskode leveret af brugeren. Det garanterer integriteten af indholdet gennem HMAC eller gennem en digital signatur .

Fordelene ved at bruge PKCS #12 er:

- der er **ingen implementeringsomkostninger,** da det er et format tilgængeligt i mange biblioteker til mange enheder (fra computere til mobiltelefoner)
- de **understøttes allerede af de fleste operativsystemer** og software, hvilket betyder, at den er tilgængelig med det samme

- den **bruger "stærke" symmetriske algoritmer** (den kryptering, der anvendes, er en stærk, meget modstandsdygtig kryptering, så de er nøgler, der modstår f.eks. brute force-angreb
- den er **bærbar på forskellige enheder** , det er muligt at flytte en fil til andre enheder eller kopiere den til alle de enheder, vi bruger for at have nøglerne altid tilgængelige

Der er også nogle ulemper:

- **det kan slettes, kopieres, overføres** , dette er et vigtigt problem, fordi det kan stjæles, slettes, kopieres for at tage det væk eller bare gøres utilgængeligt
- **den private nøgle skal bruges direkte af signerings-/dekrypteringssoftwaren** , det betyder, at softwaren bruger dekrypterer denne fil og får direkte adgang til nøglen, så der er et øjeblik, hvor nøglen er i frirummet på systemet, ubeskyttet, selvom det måske er blot den flygtige hukommelse, som er det mindst let tilgængelige område; men det er tilgængeligt i klar tekst på systemet på en eller anden måde, og dette er en meget stærk begrænsning af dette format
- **det er muligt at udføre en brute force-angreb** , for eksempel i tilfælde af at nogen stjæler PKCS #12-filen, kan de prøve alle mulige adgangskoder i et separat computersystem

Kryptografiske tokens

Kryptografiske tokens er hardwareenheder, der indbygget implementerer krypteringsalgoritmer. De tillader ikke direkte adgang til kryptografisk materiale udefra, de er de eneste, der kan få adgang til og bruge nøglen. Desuden kan disse kommandoer generelt kun udføres, efter at brugeren og systemet på vegne af brugeren har autentificeret sig til tokenet, for eksempel ved at indtaste en PIN-kode. Tokens kan have et lille operativsystem, hvorigennem brugeren anmoder om nøglegenerering, kryptering og datasignaturoperationer. Den sikre ødelæggelse af tokenet betyder ofte ikke kun annulleringen, men også manglende evne til at få adgang til de hukommelsesområder, hvor nøglen blev skrevet, for at undgå enhver form for angreb, som vi ikke kunne forestille os i dag. Operativsystemet og token-hardwaren kan certificeres i henhold til specifikke sikkerhedsstandarder.

Udfordring-respons-mekanismen

En udfordrings-/svarmekanisme giver to parter mulighed for at autentificere uden at afsløre nyttig information på kommunikationskanalen. Tanken er, at den ene part A "udfordrer" den anden part B til at operere på et tilfældigt tal (kaldet *nonce* , fordi det er et tal, der kun bruges én gang) med sin egen tast, for eksempel med en envejsfunktion H. Derefter B. returnerer H(nonce) til A. Hvis B returnerer, hvad A forventer, er A sikker på identiteten af B. I det simpleste tilfælde krypterer B nonce'en med en hemmelig nøgle, A. A kender, og som har samme nøgle, i turn anvender

funktionen på nonce ved hjælp af den samme tast og verificerer, at tallet givet til det af B, matcher det, det beregnede.

Dette er en godkendelse baseret på en adgangskode, der ikke er sendt over kanalen, så det er ikke muligt at spore nøglen. Det, der gør mekanismen effektiv, er det faktum, at det tilfældige tal er forskelligt for hver session. Da informationen, der læses på kanalen, er forskellig for hver session, er den ikke nyttig til efterfølgende godkendelse. En fordel er, at der ikke er behov for synkronisering (til forskel fra One-Time-Password-mekanismen, hvor det er vigtigt at kende udgangspunktet for den Pad, der skal bruges). Da det er ubehageligt at blive brugt af mennesker, bruges det typisk til maskine-til- maskine-godkendelse. En udfordring-svar-mekanisme findes i netværksgodkendelsesmekanismerne for ADSL-routere til udbyderens godkendelsesserver.

Co-funded by the
Erasmus+ Programme
of the European Union

Spørgsmål :

I separat fil.

Vedhæftede filer :

Powerpoint i en separat fil.

Referencer:

Hvert emne har sin egen side på Wikipedia, nogle gange allerede for grundig til formålet med dette kursus; bøger er for specialiserede og er kun nyttige efter en meget dyb træning om det specifikke emne.