FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Beveiliging van netwerken en computers

Bert De Saffel

Master in de Industriële Wetenschappen: Informatica Academiejaar 2018–2019

Gecompileerd op 24 december 2018

UNIVERSITEIT
GENT

Inhoudsopgave

Hoofdstuk 1

Basisconcepten

Begrip entiteit = systeem, persoon, organisatie, ...

1.1 Doel van beveiliging

- **Confidentialiteit**

  = Informatie kan enkel gelezen worden door entiteiten die deze informatie mogen lezen.

  Berichten die niet geëncrypteerd zijn bieden automatisch geen confidentialiteit.

  Men kan hier nog verder op ingaan, namelijk: **traffic-flow confidentialiteit**. Dit wil zeggen dat het ook een geheim is wie de zender en ontvanger zijn. Dit is een pak moeilijker te realiseren dan gewone confidentialiteit.

- **Authenticatie**

  = De identiteit van een entiteit garanderen op basis van:

  - *Entiteitauthenticatie*: De identiteit wordt bepaald aan de hand van een *verzameling* attributen. Elk entiteit heeft een unieke identiteit.
  - *Attribuutauthenticatie*: De identiteit wordt bepaald aan de hand van een enkelvoudig attribuut.
  - *Data-origin authenticatie*: De identiteit wordt bepaald op basis van de oorsprong van de brong.

  In de realiteit worden deze drie methoden door elkaar gebruikt. Een globale entiteitauthenticatie om een entiteit te authenticeren tegenover het systeem, gevolgd door een attribuutauthenticatie om de entiteit meer functionaliteit te geven in het systeem.

  Indien de identiteit niet gegarandeert is, kan het zijn dat een entiteit zich voordoet als een ander entiteit. Men lost dit op door een bepaalde "handtekening"toe te voegen aan informatie, die uniek is voor elke identiteit.

- **Authorisatie**

  = Voor elke entiteit bepalen welke resources ze mogen gebruiken.

  Hiervoor moet de entiteit eerst geauthenticeerd zijn. Op basis van attributen (bv user, admin, superadmin) wordt er bepaald welke systeemresources de entiteit mag gebruiken.

- **Data-integriteit**

  = Garanderen dat de verzonden en ontvangen informatie exact dezelfde is.

  Als er zelfs maar één bit gewijzigd, toegevoegd of veranderd wordt, is er geen data-integriteit.

- **Onweerlegbaarheid**

  = De zender kan niet ontkennen dat hij een bericht heeft verzonden.

  = De ontvanger kan niet ontkennen dat hij een bericht heeft ontvangen.

  Ten eerste komt dit neer op het feit dat men niet kan liegen over bepaalde transacties, maar ook dat een aanvaller dit bericht niet *kan* verzonden hebben. Onweerlegbaarheid is moeilijker te bereiken bij de ontvanger, omdat de ontvanger dan een antwoord moet geven aan de verzender.

- **Beschikbaarheid**

  = Het systeem is 'altijd' beschikbaar

  Met altijd beschikbaar bedoelen we, beschikbaar in de vooropgelegde tijden dat het systeem beschikbaar moet zijn. Als het systeem op een bepaald tijdstip moet draaien, maar door een bepaalde aanval ligt deze stil, dan is er geen beschikbaarheid.

1.2 Bedreigingen

- **Passieve aanvallen**: Deze soort aanvallen zijn moeilijk te detecteren aangezien er geen extra activiteit naar het systeem is. Voorbeelden zijn: verkeeranalyse en afluisteren.
- **Actieve aanvallen**: Deze soort aanvallen zullen zelf berichten wijzigen om het systeem binnen te dringen of plat te leggen. Voorbeelden zijn: Berichtmodificatie, impersonatie, replay aanvallen, Denial-of-Service, Hijacking.

Mogelijke aanvallen:

- **Brute force**: Elke mogelijke combinatie uitproberen totdat de juiste gevonden wordt.
- **Cryptoanalyse**: Deze soort aanvallen maken gebruik van het verkeer dat tussen ontvanger en verzender verstuurd worden. Met behulp van kennis over cryptografische algoritmen proberen ze berichten te ontcijferen of zelf valse berichten te maken.
- **Side-channel aanval**: Maakt gebruik van fysieke attributen zoals electromagnetische radiatie, processortijd of zelf het energieverbruik.

De verschillende categorieën van aanvallen:

- **Ciphertext**: Enkel de ciphertext is bekend voor de aanvaller.
- **Known plaintext**: Eén of meerdere paren $\langle$ plaintekst, ciphertekst $\rangle$, gevonden met één sleutel, zijn bekend voor de aanvaller.
- **Chosen plaintext**: Eén of meerdere paren $\langle$ plaintekst, ciphertekst $\rangle$, gevonden met één sleutel zijn bekend voor de aanvaller. De aanvaller heeft zelf een plaintekst mogen kiezen. M.a.w.: de aanvaller kan aan een ciphertext geraken voor willekeurige plaintexts.
- **Chosen ciphertext**: Eén of meerdere paren $\langle$ plaintekst, ciphertekst $\rangle$, gevonden met één sleutel zijn bekend voor de aanvaller. De aanvaller heeft zelf een ciphertext mogen kiezen. M.a.w.: de aanvaller kan aan een plaintext geraken voor willekeurige ciphertexts.
- **Chosen text**: Combinatie van chosen plaintext en chosen ciphertext.

Bij het best beveiligde systeem is het onmogelijk om beveiligingstransformaties te inverteren. Geen enkel beveiligingsmechanisme kan dit garanderen.

1.3 Beveiligingsmechanismen

- **Encryptie**

  = Een bericht zo aanpassen dat deze onleesbaar wordt.

  Symmetrische encryptie maakt gebruik van dezelfde sleutel voor zowel de zender als ontvanger. Dit garandeert:

  - Confidentialiteit: Enkel de geëncrypteerde bestanden worden opgeslagen. De sleutel is het enige dat beveiligd moet worden
  - Authenticatie: Enkel indien beide partijen dezelfde sleutel hebben (en dus elkaar vertrouwen), kunnen ze communiceren.

  Assymetrische encryptie zorgt ervoor dat elke entiteit een sleutelpaar heeft bestaande uit een private sleutel en een publieke sleutel. Dit garandeert:

  - Confidentialiteit: Enkel de ontvanger kan berichten decrypteren die geëncrypteerd zijn met de ontvanger zijn publieke sleutel.
  - Authenticatie: Enkel de verzender kan een bericht versturen met zijn specifieke private sleutel. Dit bericht is dan ook enkel decrypteerbaar met de publieke sleutel van dezelfde verzender.

- **Hash functies**

  = Een bericht van lengte m transformeren naar een hash

  Een hashfunctie **H(m)** genereert een hash **h** voor een gegeven input **m**. Een hashfunctie dat gebruikt wordt voor beveiligingsdoeleinden moet aan een aantal vereisten voldoen:

  - Het moet werken voor elke lengte van m.
  - Ze moeten berekening snel uitvoeren. Hier kan een uitzondering op gemaakt worden indien de hashfunctie deel uitmaakt van heel gevoelige informatie zoals paswoorden. Een tragere hashfunctie zal aanvallers ook vertragen.
  - Het moet een **one-way** functie zijn. Dit wil zeggen dat, indien **h** gegeven is, **m** onmogelijk te achterhalen moet zijn.
  - Er moet **weak collision resistance** zijn. Het moet onmogelijk zijn om een bericht **n** te genereren zodat **H(n) = H(m)**. Is dit niet het geval, kan het mogelijk zijn dat er totaal andere informatie verstuurd kan worden door een aanvaller, maar die informatie heeft dezelfde hash als de geldige informatie. De ontvanger zal niet doorhebben dat hij met foutieve informatie werkt. Integriteit en data-origin authentication worden hier overtreden.
  - Er moet ook **strong collision resistance** zijn. Indien **n** een bericht is dat verschillend is van **m**, mag de hashfunctie nooit dezelfde **h** uitkomen voor deze twee inputs. Indien dit wel mogelijk is, kan een aanvaller meerdere versies van een document genereren die dezelfde hashwaarde hebben. Niet-verlorenbaarheid wordt hier overtreden.

  Strong collision resistance is veel moeilijker om te implementeren dan weak collision resistance. De kans dat twee berichten uit een groep van **k** berichten dezelfde hashwaarde opleveren met een hashfunctie die **N** waarden kan genereren is:

$$P(N,k) = \frac{1 - N!}{((N-k)!N^k)} = 1 - e^{\frac{-k^2}{2N}}$$

  hieruit volgt

$$k \approx -\sqrt{N}\ln(1 - P(N,k))$$

  Stel nu **P(N, k)) = 0.99**, dan is **k** $\approx$ **4,6** en $\sqrt{N} \approx 4,6 \cdot 2^{n/2}$.

- **Message Authentication Code**

  Een **Message Authentication Code (MAC)** kan net zoals een hash gebruikt worden om informatie te encrypteren. De input van deze functie is echter een combinatie van plaintext en een private sleutel. Het biedt ook bijkomende functionaliteit:

  – Controle of een bericht aangepast wordt.
  – Controle of dat de verzender correct is
  – Indien een sequentienummer bijgehouden wordt, dat de berichtvolgorde gerespecteerd wordt.

  MAC kent een gelijkenis met **symmetrische encryptie**. In beide gevallen moet de zender en ontvanger een private sleutel hebben zodat ze hetzelfde authenticatiemechanisme hebben. Het nadeel is natuurlijk dat de ontvanger ook deze sleutel moet hebben. MAC biedt wel geen confidentialiteit. Dit laat echter toe om beide functies op te splitsen. De authenticatie kan met een MAC gebeuren op een server, terwijl confidentialiteit de taak is van een werkstation, gebruik makend van encryptie. Gegeven de notatie $MAC = C_k(M)$ met k de private sleutel en M het bericht. Een MAC moet aan volgende vereisten voldoen:

  – Indien M en $C_k(M)$ gekend zijn, moet het onmogelijk zijn om een bericht M' te construeren zodat
  $$C_k(M') = C_k(M)$$
  Dit is nodig voor data-integriteit
  – De waarden die $C_k(M)$ kan aannemen met gelijk verspreidt worden over alle mogelijke waarden van de MAC. De kans dat twee berichten M en M' dezelfde MAC krijgen moet $\frac{1}{2^n}$ zijn.

  De ideale MAC is enkel kwetsbaar voor brute force aanvallen.

1.4 Testvragen

1. Leg het verschil uit tussen confidentialiteit, authenticatie, authorisatie, data-integriteit, onweerlegbaarheid en beschikbaarheid.

   Zie deel 1.1

2. Waarom worden sequentienummers toegevoegd aan berichten? Is het een goed idee om een timestamp te gebruiken voor die reden?

   Een sequentienummer zorgt ervoor dat een bericht uniek is voor een bepaalde sessie. Een ontvanger verwacht een bepaald sequentienummer van de verzender. Op die manier worden replay aanvallen voorkomen, omdat eens dat een sequentienummer door de ontvanger opgebruikt wordt, deze niet meer opnieuw kan gebruikt worden. Een timestamp kan een goed idee zijn, zodat ook de tijd in rekening gebracht wordt.

3. Welke maatregelen kunnen genomen worden tegen DoS en DDoS aanvallen?

4. Geef vijf voorbeelden van actieve aanvallen die een netwerkprotocol kunnen comprimeren.

   (a) Berichtmodificatie
   (b) Replay-aanvallen
   (c) Impersonatie
   (d) DoS
   (e) Hijacking

5. Wat zijn de strong en weak collision requirements? Geef voorbeelden waarom deze belangrijk zijn.

   Weak collision requirement = Gegeven een hashwaarde $H(m) = h$, moet het onmogelijk zijn een bericht n te vinden die dezelfde waarde $H(n) = h$ oplevert. Indien dit wel mogelijk zou zijn, dan kan een aanvaller andere berichten versturen, die toch dezelfde hash hebben als het legitieme bericht. De ontvanger kan dit onderscheidt dan ook niet maken zodat data-integriteit hier niet voldaan wordt. Data-origin wordt hier ook niet voldaan, omdat de ontvanger niet kan weten dat het van een andere ontvanger zou komen.

   Strong collision requirement = Gegeven twee berichten m en n, die verschillend zijn van elkaar, dan mogen $H(m)$ en $H(n)$ niet gelijk zijn aan elkaar. Is dit wel het geval, dan kan een aanvaller meerdere berichten n_i zoeken die dezelfde hashwaarde heeft als m, zodat onweerlegbaarheid niet voldaan is. waarom?

6. Wat is het hoofdzakelijk verschil tussen een digitale handtekening en een MAC.

   Een MAC werkt via een symmetrisch sleutelmechanisme, zodat enkel zenders en ontvangers moet dezelfde sleutel, kunnen verifiëren of dat het bericht ongewijzigd (integriteit) is of dat het bericht van de juiste zender komt (authenticatie). Een digitale handtekening werkt volgens een asymmetrisch sleutelmechanisme.

Hoofdstuk 2

Netwerk en communicatiebeveiliging

Voorlopig hebben we enkel de basisconcepten gezien zoals: symmetrische encryptie, asymmetrische encryptie, hashfuncties en message authentication codes. Dit hoofdstuk zal deze concepten toepassen om beveiligingsprotocollen te ontwikkelen voor netwerken.

2.1 SSH

SSH (Secure Shell) is een **applicatielaagprotocol**. Ondanks deze naamgeving bevat SSH ook een **transportlaagprotocol**, met als bedoeling een veilige connectie te maken tussen andere OSI applicatielaagprotocollen (bv HTTP) en de werkelijke OSI transportlaag. SSH can draaien op zowel workstations als routers en switches. Routers en switches volgen de OSI laag namelijk niet strict, zodat zij ook een applicatielaag hebben. Vroeger werd het programma **telnet** gebruikt om remote configuratie toe te passen. Deze manier van werken is **onveilig** aangezien verkeer tussen de local host en remote host niet geëncrypteerd wordt, zodat deze door eender wie kunnen bekeken worden. SSH verhelpt dit probleem door de informatie te encrypteren. Vrijwel alle UNIX en Linux distributies komen met een versie van SSH geinstalleerd. SSH laat onder andere toe om:

- een veilige remote verbinding op te zetten vanuit eender welke local host naar eender welke remote host. Het protocol maakt gebruik van erkende algoritme voor zowel encryptie (sleutels van ten minste 128 bits), data-integriteit, sleuteluitwisselingen en public key management. Tijdens het leggen van een connectie worden algoritmen tussen de local host en remote host afgesproken, op basis van welke ze al dan niet ondersteunen,
- TCP te tunnelen via een SSH connectie,
- bestanden te verplaatsen, gebruik makend van bijhorende protocols zoals SCP (Secure Copy) of SFTP (SSH File Transfer Protocol),
- zowel X-sessions als poorten te forwarden.

2.1.1 Architectuur

SSH kan opgesplitst worden in drie elementen: Transport Layer Protocol, User Authentication Protocol en het Connection Protocol.

Transport Layer Protocol

Dit protocol heeft onder andere de verantwoordelijkheid om: authenticeren van servers, sleuteluitwisselingen en perfect forward privacy implementeren. Perfect forward privacy wil zeggen dat, indien een sleutel gecomprimeerd wordt tijdens een sessie, deze geen invloed kan hebben op de beveiliging van vorige sessies. Dit protocol loopt uiteraard op de transportlaag, en in de meeste gevallen zal dit TCP zijn. Vooraleer de cliënt kan connecteren moet hij de **public host key** van de server bezitten. Hiervoor kunnen er twee modellen gebruikt worden (cfr. RFC 4521) waarbij:

1. de cliënt een lokale databank bevat die de mapping beschrijft van elke hostnaam naar de correspondeerde public host key. Op deze manier is er geen centrale entiteit nodig, maar het beheer van deze databank kan, indien deze groot genoeg wordt, lastig zijn om te onderhouden.
2. de mapping bevestigd wordt door een **CA (Certification Authority)**. De cliënt kent in dit geval enkel de CA root key waarmee de geldigheid van elke host key kan nagaan die getekend zijn door deze CA. In dit geval moet de cliënt slechts één of enkele CA root keys bevatten.

Wanneer er een connectie kan gelegd worden van een cliënt tussen een server, is het eerste proces altijd het **onderhandelen van de algoritmen**. Deze stap zal algoritmen selecteren die compatibel zijn met zowel de cliënt als de server. Wanneer de cliënt een TCP connectie heeft met de server worden volgende pakketten verstuurd:

- **Identification string exchange.** Dit is een string dat zowel het protocolversie als de softwareversie van SSH bevat. Deze string wordt ten eerste van de cliënt naar de server verstuurd, waarop de server dan antwoord met zijn protocol en softwareversie. Indien hieruit blijkt dat de machines niet compatibel zijn met elkaar, wordt de connectie onderbroken (SSH2 is bv niet compatibel met SSH) en worden volgend stappen bijgevolg niet meer uitgevoerd.
- **Algorithm Negotiation.** In deze fase sturen zowel de server als de cliënt een `SSH_MSG_KEXINIT` bericht, die een lijst van alle ondersteunde algoritmen bevat, in volgorde van voorkeur. Elk type van algoritme zoals sleuteluitwisseling, encryptie, MAC algoritme en compressiealgoritme heeft zo zijn eigen lijst. Elk type moet dan ook een algoritme toegekend krijgen en wordt bepaald door het eerste algoritme dat de cliënt goed vind dat ook beschikbaar is op de server.
- **Key Exchange.** Indien de vorige fase goed gelukt is, start nu de sleuteluitwisseling. Voor de sleuteluitwisseling worden er momenteel slechts twee versie van **Diffie-Helman** ondersteund (cfr. RFC 2409). Het einde van de sleuteluitwisseling wordt gesignaleerd door een `SSH_MSG_NEWKEYS` pakket, met als gevolg dat zowel de cliënt als de server de gegenereerde sleutels mag gebruiken.
- **Service Request.** De laatste stap, dat eigenlijk het begin is van een volgend proces, wordt gesignaleerd door een `SSH_MSH_SERVICE_REQUEST` pakket. Dit pakket vraagt ofwel de start van het User Authentication Protocol of van het Connection Protocol. Alle verkeer tussen server en cliënt wordt op dit moment getransporteerd als de payload van een SSH Transport Layer pakket, beveiligd met encryptie en een MAC.

Een SSH Transport Layer pakket heeft de volgende vorm, waarbij elementen met een δ-symbool geëncrypteerd en geauthenticeerd zijn en elementen met een Δ-symbool ook optioneel gecompresseerd kunnen zijn:

δ **Pakketlengte (4 bytes).** De lengte van het pakket in bytes, zonder de lengte van de pakketlengte zelf en het MAC veld in beschouwing te nemen.

δ **Paddinglengte (1 byte).** Dit is de lengte van het random padding veld.

Δ **Payload**. De eigenlijke informatie van het pakket.

δ **Random padding.** Dit veld dient om cryptoanalyse moeilijk te maken. Kleine pakketten zijn op deze manier minder veel minder voorspelbaar en algoritmen, die vaak een kenmerkende vaste lengte hebben, worden ook moeilijker te achterhalen.

/ **MAC veld.** Dit veld wordt enkel gegenereerd indien dit zo in de onderhandeling besproken werd. Dit veld wordt berekent over het hele pakket en krijgt ook nog een bijhorend sequentienummer. Het sequentienummer start op 0, en wordt telkens met 1 geïncrementeerd voor elk pakket. Een aanvaller kan dit sequentienummer niet achterhalen aangezien een MAC een onomkeerbaar proces is.

Het **sleuteluitwisselingsproces** vraagt wat enige uitleg, er is echter nog niet nagegaan of deze sleutels op een **veilige** manier uitgewisseld worden. Op het moment van sleuteluitwisseling is er helemaal nog geen SSH connectie. Het uitwisselingsproces verloopt in twee fasen: eerst wordt er een gedeelde sleutel gegenereerd met Diffie-Helman, daarna wordt deze gedeelde sleutel gesigneerd met de publieke sleutel van de cliënt voor authenticatie. Sleutels kunnen ook heruitwisseld worden, hierbij gelden volgende regels:

- Mogelijk op elk moment, behalve tijdens het sleuteluitwisselingsproces.
- Kan aangevraagd worden door beide partijen.
- Sessie identificaties blijven ongewijzigd.
- Cryptografische algoritmen kunnen gewijzigd worden.
- Sessiesleutels worden vervangen.

Meestal worden sleutels vervangen na het behalen van een bepaalde quota zoals een tijdslimiet of het aantal totaal verstuurde bytes.

User Authentication Protocol

Dit protocol specifieerd **hoe** een cliënt zich moet authenticeren aan een server. Meerdere methoden zijn mogelijk waaronder de drie belangrijkste ervan: Public Key Authentication, Password Authentication en Host Based Authentication:

- **Public Key Authentication.** De implementatie van deze methode is afhankelijk van het gebruikte public-key algoritme. Een cliënt verstuurt berichten, dat gesigneerd is door de cliënt zijn private sleutel, naar de server, die de publieke sleutel van de cliënt bevat. De server gaat na of deze publieke sleutel nog geldig is en zoja, of dat de signatuur correct is.
- **Password Based Authentication.** De cliënt verstuurt zijn paswoord, dat geëncrypteerd wordt door het SSH Transport Layer Protocol naar de server, die nagaat of het paswoord correct is.
- **Host Based Authentication.** De authenticatie is gebaseerd op de host in plaats van een individuele gebruiker (cliënt) zelf. De cliënt verstuurt een handtekening, gemaakt met de private key van de host. De server verifieerd enkel de host, en dus niet de individuele gebruikers.

Connection Protocol

Het connection protocol veronderstelt dat een beveiligde, geauthenticeerde verbinding tot stand gebracht is. Deze verbinding, **tunnel** genoemd, wordt gebruikt door het connection protocol om een aantal logische kanalen te multiplexen doorheen deze tunnel. Het connection protocol specifieert vier kanalen:

1. **Session.** Dit kanaal refereert naar het remote uitvoeren van een programma zoals een shell, een applicatie zoals e-mail of een systeemcommando.
2. **X11.** Het X Window System laat toe om applicaties te runnen op een server, maar dat ze eigenlijk op een desktop getoond worden.
3. **Direct-tcpip.** Dit kanaal dient voor local port forwarding.
4. **forwarded-tcpip.** Dit kanaal dient voor remote port forwarding.

De lifecycle van een kanaal verloopt in drie stages: een kanaal openen, uitwisselen van informatie en een kanaal sluiten. Wanneer de server of de cliënt (hier uitgewerkt voor de cliënt) een kanaal wil openen, wordt er een lokaal nummer bijgehouden voor dat kanaal en wordt er een bericht verstuurt naar de server met volgende informatie:

- Eén van de vier kanaaltypes.
- Het lokale kanaalnummer van de cliënt.
- De initiële window size. Dit is het aantal bytes dat kan verstuurt worden naar de verzender zonder dat het venster moet aangepast worden.
- De maximum pakketgrootte specifieert hoeveel bytes een individueel pakket mag bevatten.

Als de server het kanaal kan openen, zal het een `SSH_MSG_CHANNEL_OPEN_CONFIRMATION` bericht versturen. Dit bericht bevat het kanaalnummer van de cliënt, het kanaalnummer van de server, en de waarde voor de window-en pakketgrootten voor inkomend verkeer. Als de server het kanaal niet kan openen, zal het een `SSH_MSG_CHANNEL_OPEN_FAILURE` versturen, met een foutmelding. Bij een open kanaal worden er voortdurend `SSG_MSG_CHANNEL_DATA` berichten verstuurd, die het kanaalnummer van de ontvanger en een datablok bevat. Deze berichten kunnen in twee richtingen voorkomen. Tot slot kan er nog een `SSH_MSG_CHANNEL_CLOSE` bericht verstuurd worden, die het kanaalnumer bevat dat gesloten moet worden.

2.1.2 Port forwarding

Port forwarding of port mapping is een toepassing van **NAT (Network Address Translation)** dat een een communicatieaanvraag van een specifiek adres en poort naar een ander zal omleiden. Op deze manier kan elke onveilige TCP verbindingen in een veilige SSH verbinding gemaakt worden. Een klein woordje uitleg over een poort. Een poort is een identificatie van een TCP-gebruiker. Elke applicatie dat bovenop TCP draait, heeft een poortnummer. Het **SMTP (Simple Mail Transfer Protocol)** luistert algemeen naar poort 25. Inkomende SMTP berichten zullen dus deze poortnummer bevatten. TCP Herkent dit poortnummer en zal het verkeer routen naar de SMTP server applicatie. SSH ondersteunt twee types port forwarding: local forwarding en remote forwarding:

- Local forwarding.
- Remote forwarding.

2.1.3 Tekortkomingen van SSH

! Niet geschikt voor trage connecties.

! Niet geschikt voor onstabiele netwerken.

2.1.4 Testvragen

1. Juist of fout: het SSH transportlaag protocol encrypteerd TCP packetten.
2. Leg de functie van het SSH transportlaag protocol, SSH user authentication protocol en SSH connectieprotocol uit.

   Δ Het SSH transportlaag protocol zal proberen om een SSH connectie te leggen tussen de SSH cliënt en de SSH server. De cliënt en de server spreken onder andere af welke cryptografische algoritmen ze gaan gebruiken voor verschillende categoriën zoals: encryptie, compressie, mac en sleuteluitwisseling. Dit protocol is voltooid indien de cliënt één van de volgende protocollen oproept met het `SSH_MSG_SERVICE_REQUEST` bericht. Op dit moment verloopt het verkeer tussen de cliënt en de server via SSH transportlaagpakketten verstuurd, die beschermd zijn met encryptie en een MAC.

   Δ Het SSH user authentication protocol zal, nadat het transportlaag protocol voltooid is, de cliënt trachten te authenticeren. Er zijn drie mogelijke manieren:

   – Host-based authentication: Enkel de host wordt geauthenticeerd, zodat er verondersteld moet worden dat de host zelf al de individuele gebruikers heeft geauthenticeerd. De gebruiker maakt gebruik van de host zijn private host key om een handtekening te maken. De server verifieerd deze handtekening, gebruik makend van de public host key van de host.
   – Password-based authentication: Bij elke nieuwe SSH connectie moet de gebruiker zijn paswoord meegeven. Dit paswoord is natuurlijk geëncrypteerd door het SSH transportlaag protocol.
   – Public key authentication: Een gebruiker moet een public-private sleutelpaar hebben. Bij een SSH connectie voor een gebruiker wordt er een speciale handtekening, gegenereerd met behulp van zijn private sleutel. De server controleert of de bijhorende publieke sleutel correct is, en authenticeerd al dan niet de gebruiker.

3. In welke gevallen is het aan te raden om een SSH sleutel*her*uitwisseling te initialiseren? Waarom?
4. Wat is het verschil tussen een SSH sessie en een SSH kanaal. Welke kanaaltypes worden ondersteund?
5. Naar welk poortnummer luistert SSH in normale omstandigheden?

   Poort 22.
6. Leg het verschil uit tussen local en remote port forwarding.

2.2 Sleuteluitwisselingen

Gegeven twee partijen A en B. Een sleuteluitwisseling kan op verschillende manieren gebeuren:

1. A kan het fysiek afleveren aan B.
2. Een betrouwbare partij geeft de sleutel fysiek aan A en B.

3. Als A en B vroeger al gecommuniceerd hebben, kunnen ze de vorige sleutel gebruiken om de nieuwe te encrypteren.
4. Als A en B een beveiligde verbinding hebben met een derde partij C, dan kan C de sleutel doorverwijzen naar A en B.

2.2.1 Out-of-band sleuteluitwisseling

Sleutels niet-digitaal uitwisselen wordt out-of-band sleuteluitwisseling genoemd. Dit is toelaatbaar voor kleine organisaties maar voor grotere organisaties is dit niet voldoende.

2.2.2 Diffie-Hellman

Dit protocol was de eerste praktische methode om een gedeelde sleutel over een onbeveiligde verbinding te versturen, gebaseerd op discrete logaritmen. Het maakt gebruik van de eigenschap

$$(g^b \ mod \ p)^a \ mod \ p = (g^a \ mod \ p)^b \ mod \ p$$

indien p een priemgetal en g de primitieve wortel modulo p is. Een voorbeeld is $p = 7$ en $g = 3$.

$$\begin{aligned} 3^1 &= 3 = 3 \bmod 7 \\ 3^2 &= 9 = 2 \bmod 7 \\ 3^3 &= 27 = 6 \bmod 7 \\ 3^4 &= 81 = 4 \bmod 7 \\ 3^5 &= 243 = 5 \bmod 7 \\ 3^6 &= 729 = 1 \bmod 7 \end{aligned}$$

Het algoritme verloopt in volgende stappen:

1. Alice en Bob zoeken een priemgetal p, en een daarbijhorende primitieve wortel g. Deze p en g mogen publiek gekend zijn.
2. Alice kiest een geheim getal $a < p$. Haar publieke sleutel is $y_A = g^a \bmod p$.
3. Bob kiest een geheim getal $b < p$. Zijn publieke sleutel is $y_B = g^b \bmod p$.
4. Alice en Bob bereken respectievelijk $y_B^a \bmod p$ en $y_A^b \bmod p$. Deze zullen beiden dezelfde uitkomst bekomen, en wordt dan hun sleutel.

Aangezien g^x elk mogelijk getal zal genereren modulo p voor alle $x < p$, is het moeilijk voor een aanvaller, die de publieke sleutels toch kent, de private sleutel te achterhalen. Ze moeten hierbij het discrete logaritme probleem oplossen.

Stel $p = 23$ en $g = 5$. Alice neemt $a = 6$ en Bob neemt $b = 15$, dan wordt $y_A = 5^6 \bmod 23 = 8$ en $y_B = 5^{15} \bmod 23 = 19$. Alice berekent $19^6 \bmod 23 = 2$ en Bob berekent $8^{15} \bmod 23 = 2$.

Normaal wordt p een priemgetal van ongeveer 300 cijfers genomen. De getallen a en b zijn dan weer minstens 100 cijfers lang. De private sleutel bepalen, indien g, p, y_A en y_B gekend zijn duurt quasi oneindig lang.

Nadelen *normale* Diffie-Hellman

- ! Dure operaties die kunnen gebruikt worden om een server plat te leggen.
- ! Man-in-the-middle aanval: Alice kan haar publieke sleutel naar Bob versturen, maar Carol ondermijnt dit bericht. Carol verstuurd nu de publieke sleutel van Alice naar Bob, waarop Bob antwoord met zijn publieke sleutel. Carol kan nu Bob zijn publieke sleutel naar Alice versturen. Op die manier hebben Alice en Carol, en Carol en Bob een gedeelde sleutel. Alice en Bob denken echter dat ze tegen elkaar praten, omdat Diffie-Helman geen authenticatie voorziet.

Varianten

- Δ **ECDH**: Elliptic curve Diffie-Helman.
- Δ **Fixed Diffie-Hellman**: De publieke sleutels worden getekend door een CA.
- Δ **Anonieme Diffie-Hellman**: Normale Diffie-Hellman.
- Δ **Ephemeral Diffie-Hellman**: Publieke sleutels zijn nu tijdelijk, zodat ze enkel maar gelden voor één enkele sessie. Op die manier is er perfect forward privacy.

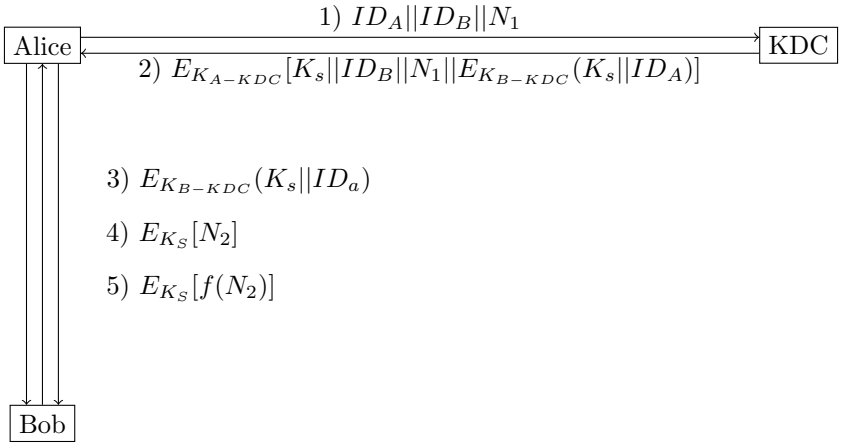
2.2.3 Sleuteldistributiecentrum

Het sleuteldistributiecentrum vermijdt dat gebruikers zelf de sleutel moeten overhandigen. Elke gebruiker heeft een private sleutel die kan gebruikt worden om met het **KDC (Key Distribution Centre)** te communiceren. Er zijn twee soorten sleutels: de sessiesleutels zijn een tijdelijke sleutel die gebruikt worden voor één enkele sessie. Deze sessiesleutel wordt gebruikt om informatie te encrypteren tussen verschillende gebruikers. De ander sleutel is de mastersleutel. Deze wordt gebruikt om sessiesleutels te encrypteren en wordt gedeeld tussen een gebruiker en de KDC. De basiswerking van de KDC (figuur 2.1) is als volgt:

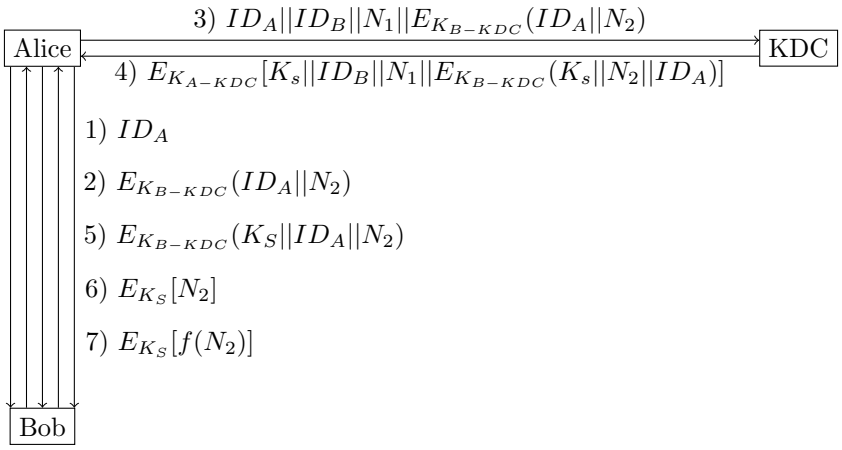
1. Alice vraagt aan de KDC om een verbinding op te zetten met Bob. Alice stuurt ook een "nonce" N_1. Dit is een willekeurig getal dat slechts éénmaal gebruikt wordt om replay aanvallen te voorkomen.
2. De KDC geeft volgende informatie terug, dat geëncrypteerd wordt door Alice haar sleutel K_{A-KDC}: De sessiesleutel, de identiteit van Bob, dezelfde nonce dat Alice stuurde naar de KDC, en geëncrypteerde informatie (met Bob zijn sleutel K_{B-KDC}).
3. Alice verstuurt de geëncrypteerde informatie naar Bob.
4. Bob antwoordt met een tweede nonce N_2, geëncrypteerd met de sessiesleutel, zodat Bob geauthenticeerd is naar Alice toe.
5. Alice antwoordt terug naar Bob met een bewerkte nonce (een bepaalde functie, bv 1 toevoegen), geëncrypteerd met de sessiesleutel, zodat Alice geauthenticeerd is naar Bob toe.

Een aanvaller kan, indien hij K_s bemachtigd, stap 3 herhalen, zodat de aanvaller nu Bob zijn response kan beantwoorden. Bob is dan aan het praten met de aanvaller, terwijl hij denkt dat het Alice is. Een beter algoritme:

1. Alice informeert Bob dat ze wil communiceren.
2. Bob antwoordt met een geëncrypteerde nonce N_2.



Figuur 2.1: Basiswerking KDC.



Figuur 2.2: Verbeterde werking KDC.

3. Alice vraagt aan de KDC om communicatie te starten met Bob. Alice verstuurt een andere nonce N_1, samen met de geëncrypteerde nonce N_2.
4. De KDC beantwoordt dit met een geëncrypteerd bericht, gebruikmakend van Alice haar sleutel $E_{K_{A-KDC}}$ (De KDC authenticeerd zich op die manier naar Alice toe, en bereikt confidentialiteit). De inhoud van dit is: de sessiesleutel, Bob zijn identiteit, N_1, en een geëncrypteerd bericht met de sessiesleutel, N_2 en Alice haar identiteit.
5. Alice verstuurt de sessiesleutel, N_2 en haar eigen identiteit geëncrypteerd met de sleutel van Bob naar Bob toe.
6. Bob antwoordt met een tweede nonce N_2, geëncrypteerd met de sessiesleutel, zodat Bob geauthenticeerd is naar Alice toe.
7. Alice antwoordt terug naar Bob met een bewerkte nonce (een bepaalde functie, bv 1 toevoegen), geëncrypteerd met de sessiesleutel, zodat Alice geauthenticeerd is naar Bob toe.

2.2.4 Public key infrastructure

Vier manieren om publieke sleutels uit te wisselen:

1. **Public Announcement**: Publieke sleutels worden gebroadcast naar een heel domein, of verstuurd naar specifieke gebruikers. Het nadeel hiervan is dat aanvallers zich kunnen voordoen als iemand anders.
2. **Public directory**: Alle publieke sleutels bevinden zich in een database in de vorm van ⟨ naam, publieke sleutel ⟩ paren. Gebruikers die hun publieke sleutel op deze directory willen zetten, moeten zich eerst authenticeren. De beheerder van de directory moet een vertrouwd entiteit zijn.
3. **Public key authority**: Het sleutelbeheer wordt overgelaten aan een betrouwbare externe organisatie. Dit is gelijkaardig aan het sleuteluitwisselingscentrum.
4. **Public key infrastructure**: (dit hoofdstuk)

- PKI = sleutels op een betrouwbare manier uitwisselen, gebruik makend van certificaten.
- Een certificaat verbindt een identiteit met een publieke sleutel, waarvan de inhoud getekend wordt door een Certificate Authority (CA).
- Alice verstuurd haar certificaat naar Bob. Bob kan dit certificaat controleren door gebruik te maken van de publieke sleutel van de CA die dit getekend heeft.

Drie rollen bij PKI:

1. Certificate Authority: Tekent en geeft certificaten op basis van de publieke sleutel en de identiteit van de aanvrager. Er zijn verschillende providers zoals: VeriSign, Equifax, PGP (web of trust), ...
2. Registration Authority: Het doel is om minder werk te geven aan de CA. De RA moet de entiteiten die een certificaataanvraag doen authenticeren.
3. Validation Authority: De VA kan op basis van een entiteit, de attributen er uit halen die hem uniek identificeren. Op deze manier is de workload van de CA nog minder.

Er blijft nog de vraag: Wie kunnen we vertrouwen? 5 implementaties:

- **CA Hiërarchy**: De publieke sleutels van Alice en Bob kunnen door een CA getekend worden, waarvan de publieke sleutel door een hoger gelegen CA getekend werd, ...
- **Web model**: Elke browser en operatingsysteem komt met een lijst van erkende root-certificaten. Het is dan ook hun taak om een nieuw certificaat te controleren, op basis van de CA hiërarchy.
- **User-centrisch** (PGP): Elke gebruiker is zijn eigen root CA. Stel dat Alice en Bob elkaar vertrouwen, en dat nu Charlie met Alice wilt praten. Alice kent Charlie niet, maar Bob wel. Charlie vraagt aan Bob om zijn publieke sleutel te tekenen met Bob zijn private sleutel. Alice kan dit nu controleren, en door transitieve eigenschap, kent Alice nu ook Charlie. Op die manier wordt er een netwerk opgebouwd, maar is dus enkel handig wanneer veel mensen collaboreren.
- **Cross-certification**: _ToDo: dunno, lijkt me niet zo belangrijk ook

Soms moet een certificaat teruggetrokken worden:

- Certificate Revocation List (CRL) mechanisme: Er wordt een lijst bijgehouden met certificaat-identifiers die teruggetrokken zijn. Bij het verifiëren van een certificaat wordt deze lijst overlopen en indien de identifier er in zit, wordt het certificaat niet goedgekeurd. Twee nadelen zijn:

- ! De lijst kan heel lang worden, en de hele lijst moet altijd opgehaald worden.
- ! Er bestaat geen mechanisme dat een notificate stuurt wanneer een certificaat teruggetrokken wordt. Resultaten die gecached werden kunnen soms dus gebruikt worden, ook al zijn ze in werkelijkheid al teruggetrokken.

Δ Online Certificate Status Protocol: Deze uitbreiding op CRL laat toe om enkel de status van één enkel certificaat op te vragen, zodat real-time validatie mogelijk is. Ook hier zijn er vier nadelen:

- ! De CA moet een massa van real-time queries kunnen behandelen.
- ! De CA moet ervoor zorgen dat de service altijd bereikbaar is.
- ! Deze query is synchroon, zodat de cliënt moet wachten op een antwoord.
- ! Real-time requests kan de privacy van een gebruiker negeren, omdat da CA nu weet welke website bezocht wordt.

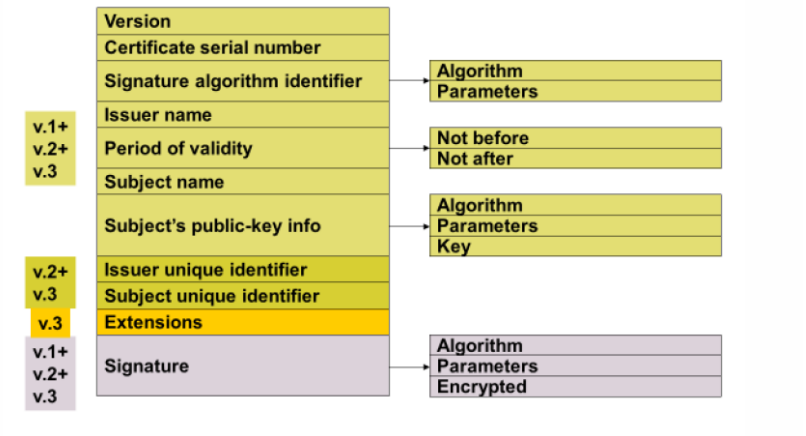
2.2.5 X.509 authentication

= een gestandariseerd formaat voor certificaten.

Δ Wordt gebruikt door veel protocollen: TLS, PGP, IPsec, HTTPS, ...

Δ Moet aangekocht worden (bestaat ook gratis alternatief: Lets Encrypt!) van een CA. Je kan ook zelf een eigen certificaatservice opstellen, en deze laten tekenen door een erkende CA.

Δ Formaat van een X.509 certificaat op figuur 2.3.



Figuur 2.3: De structuur van een X.509 certificaat.

V3 extensions

Drie categorieën:

Δ **Informatie over de sleutel en beveiligingspolicy**: Deze extensie bevat meer informatie over waar het certificaat gebruikt mag worden en de levensduur van een private sleutel

Δ **Attributen van de subject en issuer**: Er kunnen extra attributen zoals het mailadres van de subject of IPsec gerelateerde attributen bijgehouden worden.

Δ **Padconstraints**: Voorkomen dat een gebruiker zich zelf als een CA kan voordoen, de lengte van de het certificaatpad kan beperkt worden en men kan voorkomen dat een CA buiten zijn domein ook certificaten kan tekenen.

Beperkingen X.509 certificaten

! X.509 is heel vaag in veel gebieden. Dit heeft als resultaat dat niemand nog te wachten staat op verbeteringen van deze standaard.

! Het terugtrekken van certificaten gebeurd niet altijd zoals het moet; Er zit vaak een grote delay op het effectief terugtrekken van een certificaat, en het publiek maken van deze terugtrekking.

! Authenticatie-en confidentialiteitscertificaten worden op dezelfde manier behandeld.

! Constant toevoegen van attributen zorgt ervoor dat er teveel overhead onstaat.

! Veel software is incompatibel met X.509 door bugs, foute extensies, verschillende interpretaties, ...

! Certificaten zijn gebaseerd op identiteiten, terwijl dit vaak onbelangrijk is. Een identiteit kan namelijk veranderen (email, paswoord, woonplaats, ...).

! Te veel attributen opnemen in een certificaat zorgt voor een toeneming van het heraanvragen van certificaten. Een enkel attribuut veranderen, slects 1 bit, leidt tot een heraanvraag van het certificaat.

2.2.6 Testvragen

2.3 Beveiligen van netwerkprotocollen

2.3.1 Beveiligen van de transportlaag

Het belangrijkste protocol is **TLS (Transport Layer Security)**. Dit is de opvolger van SSL, zodat TLS dus ook alle features heeft van de laatste SSL versie. Hedendaags wordt enkel nog TLS gebruikt.

In eerste instantie kan SSL/TLS vergeleken worden met SSH, aangezien SSH ook de transportlaag beveiligd. De verschillen tussen SSH en SSL/TLS zijn:

- SSL/TLS is ontworpen om generiek verkeer over de transportlaag te beveiligen.
- SSH bevat ook nog multiplexing, user authenticatie en terminal management.
- SSL gebruikt X.509 certificaten, SSH gebruikt een eigen formaat.
- Verschillende optimalisaties: SSH voor shell applicaties en TLS voor betere performantie bij https.

Verder wordt enkel nog TLS behandelt, SSL kan gegenegeerd worden.

De verschillen tussen SSH en TLS zijn:

- **TLS server authenticatie is optioneel.**
  - Anonieme operaties zijn toegelaten.

  - Hierdoor is een man-in-the-middle attack een zeer eenvoudige aanval.
  - Server authenticatie in SSH is verplicht.
- **TLS heeft geen user authenticatie.**
  - TLS moet enkel de twee interfaces authenticeren (SSH kan dit ook).
  - User authentication wordt behandeld door een bovenliggende laag.
- **TLS gebruikt X.509 public-key certificaten om te authenticeren.**
  - SSH heeft een eigen certificaatformaat.
  - Vereist een werkend PKI systeem.
  - Een voordeel van een PKI systeem is dat het schaalbaar is op vlak van sleutelbeheer, iets wat SSH nog niet kan.
- **TLS kent enkel public-key authenticatie**
  - SSH heeft ook nog host-based, password, ...
- **TLS heeft niet de extra features die het SSH connection protocol wel heeft**

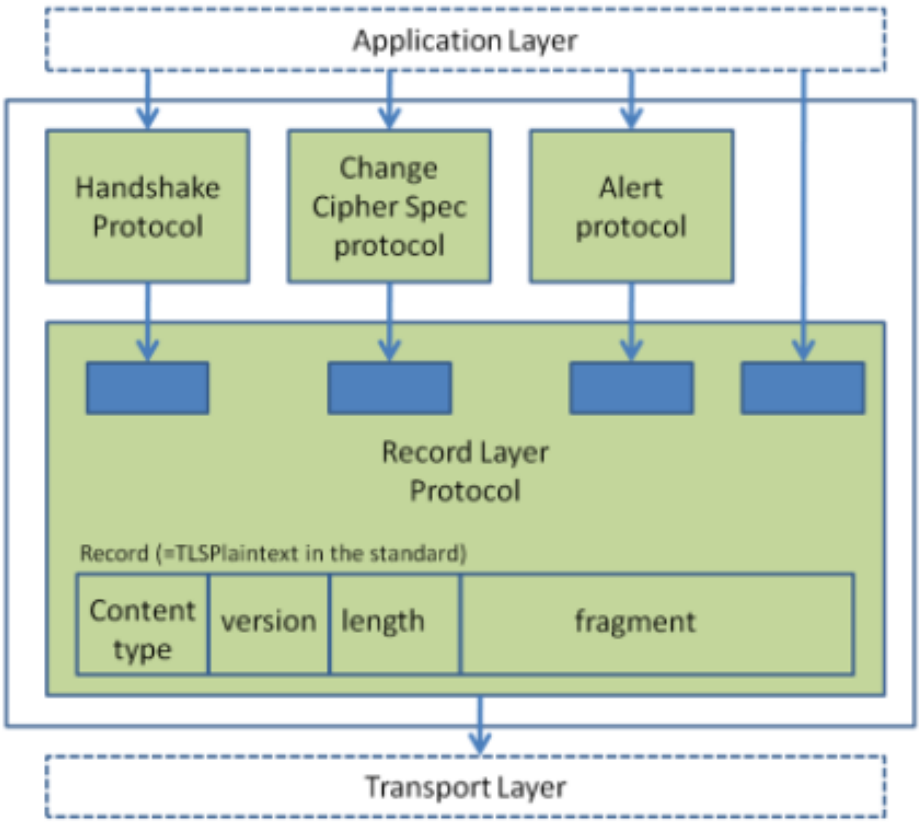
Architectuur

(elk belangrijk item staat in het vet en is onderlijnd)

- Een **TLS connectie** is een communicatiekanaal tussen een cliënt en een server. Deze connecties zijn vaak van korte levensduur en servers zullen een connectie zelf vernietigen na een bepaalde tijdsduur indien de connectie te lang op idle modus staat.
- Een **TLS sessie** wordt gebruikt om de server een bepaalde staat te geven. De connectie kan gesloten worden, maar kan een sessie behouden. Deze sessie kan dan verdergezet worden met een nieuwe connectie. Een sessie wordt op zowel de client als de server bewaart. Er kan ook een nieuwe sessie aangemaakt worden tijdens een connectie. Een sessie wordt gedefinieerd aan de hand van een aantal parameters:
  - De sessieidentifier, een random byte sequentie, gegenereerd door de server.
  - De optionele X.509 certificaten.
  - De optionele compressiemethode.
  - De *master secret*, een gedeelde sleutel van 48 bytes die gekend is door de cliënt en de server.
  - Een bit *is resumable*, die aangeeft of dat de sessie kan gebruikt worden om nieuwe connecties aan te maken.

  Figuur 2.4 toont aan dat TLS gebruik maakt van een tweelagen architectuur met

  1. het TLS record layer protocol en,
  2. het TLS-specifieke protocollen: TLS Handshake protocol, TLS Change Cipher Protocol en TLS Alert Protocol.
- Het **Handshake protocol** laat toe om de cliënt en server te authenticeren aan elkaar. Een HP bericht heeft de volgende structuur:
  - 1 byte voor de berichtsoort (10 typen gedefinieerd).
  - 3 bytes voor de berichtgrootte.



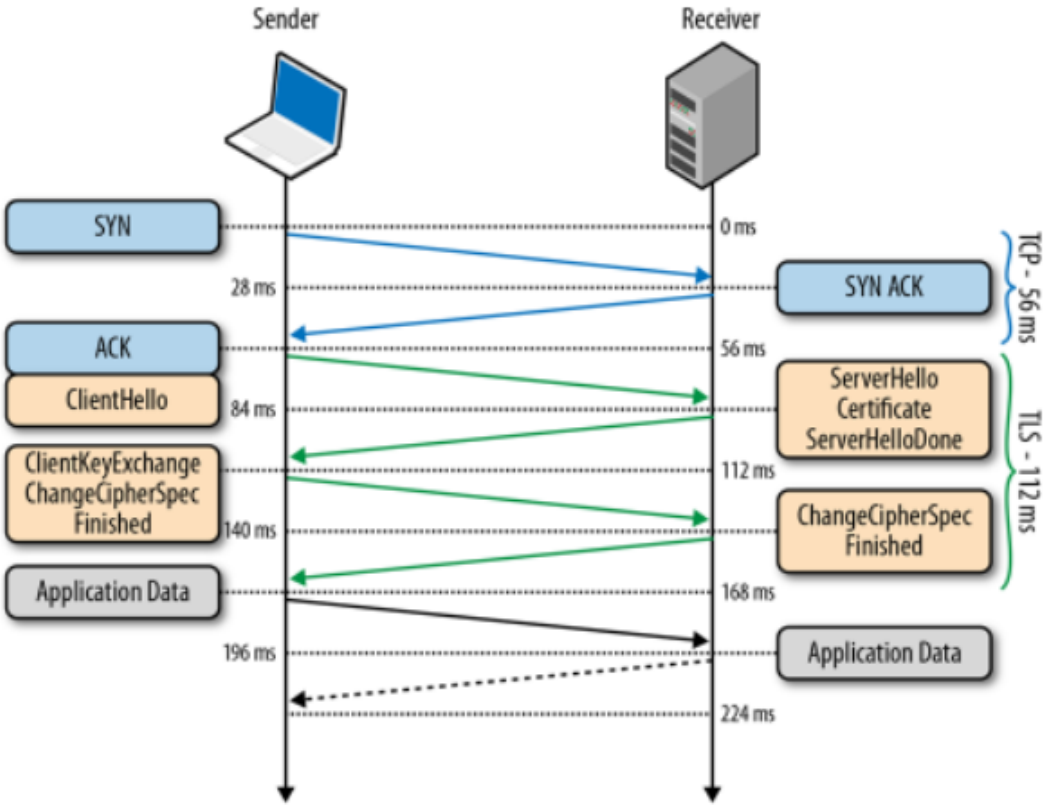
Figuur 2.4: Tweelagen architectuur van TLS

– variabel aantal bytes voor de eigenlijke informatie, soms zelfs leeg.

Het handshake protocol verloopt in 4 fasen (ook gedemonstreerd op figuur 2.5):

1. Eerst moet een TCP sessie aangemaakt worden via het normale TCP process: SYN → SYN-ACK → ACK.
2. Daarna verstuurt de cliënt een **client_hello** bericht naar de server met volgende parameters:
   – **Version**: De hoogste TLS versie waarover de cliënt beschikt.
   – **Random**: Een timestamp (32 bits) en 28 bytes gegenereerd door een veilige randomgenerator, dat gebruikt wordt als nonce.
   – **SessionID**: Een identifier dat gebruikt wordt om de sessie te identificeren. Wordt onder andere gebruikt om een sessie verder te zetten met een andere connectie.
   – **CipherSuite**: Een lijst van combinaties van cryptografische algoritmen die de cliënt ondersteunt. Een combinatie bestaat uit een sleuteluitwisselingsmethode, de encryptiespecificatie, de MAC functie en de pseudorandomfunctie dat gebruikt wordt.
   – **CompressionMethod**: Een lijst van compressiemthoden waarover de cliënt beschikt.

   De server reageert met een **server_hello** bericht die dezelfde structuur bevat als de client_hello variant. De server neemt uiteraard de versie en cryptografische algoritmen dat hijzelf ook ondersteund.



Figuur 2.5: TLS Handshake Protocol

3. Indien authenticatie vereist is, stuurt de server een X.509 certificaat naar de cliënt. **Optioneel** zal de server ook een certificaat vragen aan de cliënt, indien authenticatie vereist is. De server eindigt met een **server_done** bericht. De cliënt antwoordt dan indien nodig met zijn certificaat. De cliënt stuurt ook een **client_key_exchange** bericht, met de nodige informatie om een sessiesleutel te genereren.
4. De cliënt verstuurt nu een **change_cipher_spec** bericht (met het Change Cipher Protocol), die de juiste combinatie van cryptografische algoritmen bevat. Tot slot stuurt de cliënt een **finished** bericht. De server verstuurt dezelfde berichten naar de cliënt. Nu zijn de cliënt en server in staat om te communiceren met elkaar via het SSL Record protocol.

- Het **Change Cipher protocol** kan slechts één bericht verstaan: 1 byte met 1 waarde. Het zorgt ervoor dat de tijdelijke keuze van cryptografische algoritmen de werkelijke keuze wordt. Het kan ook gebruikt worden om de encryptiealgoritmen te veranderen gedurende een connectie.

- Het **Alert protocol** bevat berichten met foutmeldingen en waarchuwingen. Een bericht bestaat enerzijds uit één byte, die "fatalöf "warningäangeeft, en een andere byte die meer uitleg geeft over de fout.

  Mogelijke waarschuwingen:

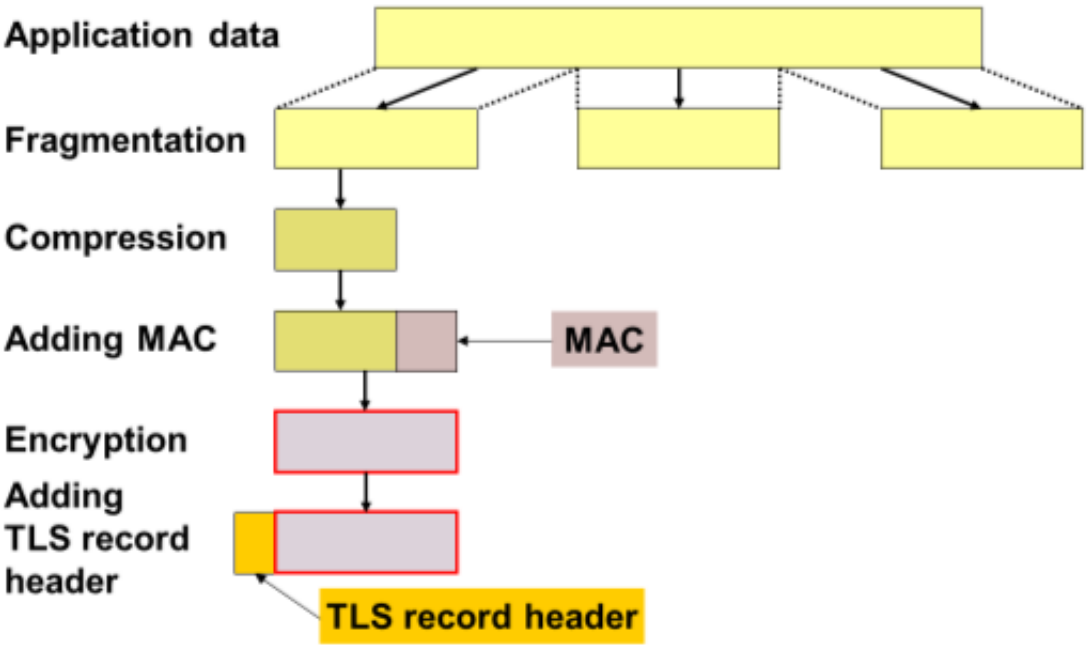
  – Problemen met certificaten.

  – De connectie wordt verbroken door één van de partijen.

  Mogelijke fouten:

- Problemen met het decompresseren.
- Een fout tijdens het handshake protocol (geen geldige beveiligingsparameters gevonden).
- Fout ingevulde parameters tijdens het handshake protocol.

- Het **Record protocol** is een gelaagd protocol dat informatie zo zal bewerken dat: het bruikbare blokken worden van ten hoogste 2^{14} bytes, optioneel de informatie zal compresseren, een MAC zal toevoegen voor integriteit, de informatie zal encrypteren en het resultaat doorsturen. Ontvangen informatie wordt dan in omgekeerde volgorde bewerkt. De typische workflow gaat als volgt (figuur 2.6):



Figuur 2.6: Werking van het TLS record protocol.

1. De informatie wordt verdeeld in fragmenten van hoogstens 2^{14} bytes ($\equiv$ 16 KB).
2. De informatie wordt eventueel gecompresseerd. De compressie is optioneel, moet terugkeerbaar zijn, en mag de lengte van de informatie niet meer dan 1024 bytes vergroten.
3. De MAC wordt als volgt berekend: MAC($K_{MAC}||Seq\#||Type||Length||Data$).
4. De informatie wordt geëncrypteerd met een symmetrisch encryptiealgoritme.
5. De TLS record header wordt toegevoegd.

Het verschil met SSH:

- SSH zal eerst encrypteren, en dan pas de MAC toevoegen. Geen van beide methoden is beter dan de andere, en komt met zijn eigen problemen.

De TLS record header heeft volgende structuur (figuur 2.7):

- 1 byte voor het contenttype. Dit is het applicatielaagprotocol dat gebruikt wordt.
- 2 bytes voor de versie (voor TLS 1.2 heeft dit de waarden 3 en 3).
- Lengte van de informatie.

Figuur 2.7: De TLS record header.

- De **master secret**, die nodig is bij een TLS sessie, wordt ofwel gegenereerd door de cliënt met RSA encryptie en verzonden naar de server, ofwel uitgewisseld met Diffie-Helman. De grootte van de sleutel bedraagt 384 bits. Sleutels kunnen niet hergebruikt worden aangezien ze afhankelijk zijn van de nonces die gebruikt werden tijdens het handshake protocol.
- Het **Heartbeat protocol** draait bovenop het record protocol en heeft voornamelijk als doel om huidige connecties te testen, en om ze in leven te houden zonder dat er opnieuw moet onderhandeld worden tussen de cliënt en de server. Het protocol ondersteund twee berichten:
  - **HeartbeatRequest**: De cliënt of server verstuurt naar de ander een bericht met daarin een payload (typisch een string) en de lengte van de payload.
  - **HeartbeatResponse**: Een ontvanger van een HeartbeatRequest moet antwoorden met dezelfde payload als een HeartbeatResponse.

  Ontstaan van Heartbleed bug. De server controleerde niet of de lengte van de payload werkelijk de lengte was. Zo kon een aanvaller een payload met werkelijke lengte 5 hebben, maar de payload lengte zelf instellen op 500, zodat er 495 extra bytes teruggestuurd worden vanuit het geheugen van de server, vaak met gevoelige informatie.

Performantie

TLS wisselt minder berichten uit dan SSH en authenticeerd het entiteit volledig over de transportlaag.

TLS geeft wel meer overhead op vlak van latency. Het aantal uitwisselingen tussen de cliënt en server is vrij groot. Initieël moet de TCP connectie opgesteld worden (3 uitwisselingen), gevolgd door 4 uitwisselingen via het handshake protocol.

Hoe TLS performanter maken?

- **Sessie voortzetten**: Als de cliënt al eerder met de server heeft gecommuniceerd, kunnen de vorige parameterwaarden gebruikt worden, zodat het aantal uitwisselingen met 2 daalt (de changecipherspec voor zowel server als cliënt).
- **Valse start**: De cliënt of server zullen al informatie versturen vooraleer het handshake protocol voltooid is, meer specifiek: nadat de ChangeCipherSpec en Finished berichten verstuurt zijn, maar zonder te wachten op de andere kant op dezelfde berichten. Vanaf dat de cliënt de algoritmen kent, kan hij zijn informatie al encrypteren. Deze methode vergt minimale aanpassing aan het TLS algoritme, namelijk wanneer de berichten mogen verstuurd worden.
- **Proxies**: In het geval dat de geografische afstand tussen cliënt en server te groot is, kunnen er proxyservers geïnstalleerd worden, die dichter bij de cliënt zitten.
- **Pakketgrootte**: Alle informatie via TLS wordt verpakt via het record protocol. In het begin van een TLS verbinding is het beter om de fragmentgrootte kleiner te maken dan de default 16 kb, zodat de verbindingen sneller kan gelegt worden.
- **Certificaatketting**: Stuur niet enkel eigen certificaat maar ook het certificaat van diegene dat het getekend heeft. Dit bespaart een DNS lookup.

2.3.2 Beveiligen van de netwerklaag

= gebruik van IPSec

Mogelijke bedreigingen op de netwerklaag:

- IP-adres kan gewijzigt worden (source spoofing).
- Vervalste routeinformatie kan verzonden worden.

Dit resulteert dat er geen data-integriteit of confidentialiteit is.

IPSec is een applicatieonafhankelijk protocol dat veilige IP verbindingen legt.

Voordelen van IPSec:

- Aangezien dat IPSec applicatieonafhankelijk is, moeten applicaties zelf geen beveiliging op netwerklaag voorzien.
- Beveiligingsmechanismen moeten maar gelden op beperkt aantal systemen (firewall of router). Het interne verkeer wordt hierdoor niet beïnvloedt.
- Eindgebruikers moeten zich hier niet van bewust zijn.

Nadelen van IPSec:

- Er is geen beveiliging nadat het bericht de gateway heeft gepasseerd.
- Systeemresources zijn nodig om cryptographische functies te berekenen.
- De implementatie is vrij complex en het gebeurt vaak dat verschillende versies van IPSec incompatibel zijn met elkaar.

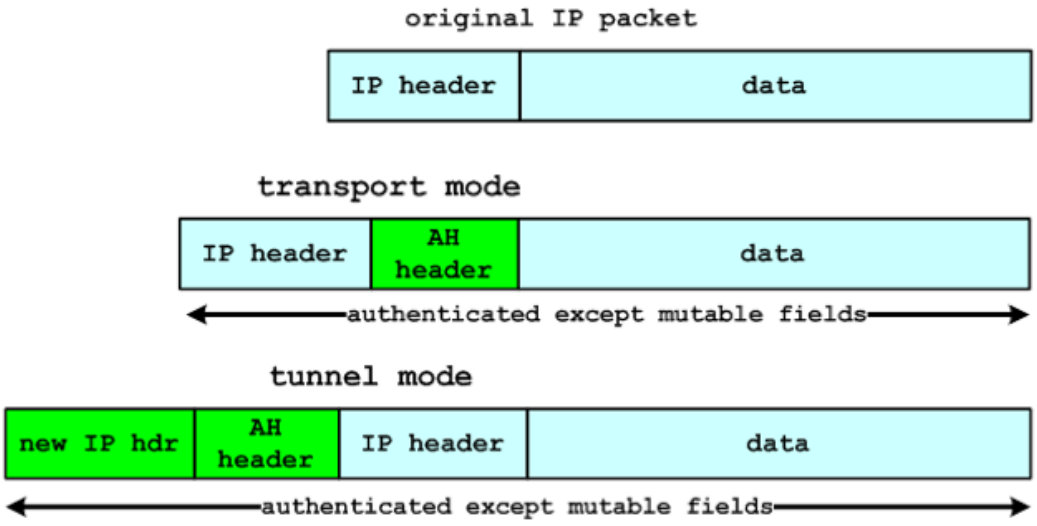
IPSec kent twee modi:

- **Layer 2 tunnel mode** (default): Dit beschermt de interne routinggegevens door heel het pakket te encrypteren en een nieuw IP header vooraan toe te voegen. Dit is geschikt voor *network-to-netwerk* (tussen 2 verschillende routers op verschillende locaties), *host-to-network* (remote user access) en *host-to-host* (private chat) communicatie.
- **Transport mode**: Enkel de payload wordt geëncrypteerd zodat de IP header ongewijzigt blijft. Dit is geschikt voor *end-to-end* communicatie tussen toestellen met publieke IP adressen.

Protocollen

IPsec maakt gebruik van twee protocollen: Authentication Header en Encapsulating Security Payload.

- **Authentication Header** (figuur 2.8): Dit biedt authenticatie, data-integriteit en voorkomt IP spoofing en replay mechanismen.
  1. De IP header en data payload worden gehasht.
  2. De returnwaarde van deze hashfunctie wordt gebruikt om een authentication header te maken, die toegevoegd wordt aan het originele pakket.
  3. Het pakket wordt doorgestuurd naar de IPSec router van de begunstigde.



Figuur 2.8: Authentication Header.

4. Deze router zal de IP header en data opnieuw hashen, en vergelijkt deze met de authentication header. Indien deze verschillend zijn, wil dit zeggen dat het pakket is aangepast.

- **Encapsulating Security Payload** (figuur 2.9): Dit biedt confidentialiteit, data-origin authenticatie en data-integriteit. Voorkomt ook replay mechanismen en biedt beperkte traffic-flow confidentialiteit.

  1. ESP maakt gebruik van een confidentialiteitsfunctie. De informatie (en optioneel de IP header) wordt geëncrypteerd met een traditioneel algoritme (AES-128-CBC, AES-GCM, ...)
  2. Optioneel wordt er ook geauthenticeerd, gebruik maken van dezelfde methoden als bij de authentication header.

  In transport mode biedt ESP encryptie en authenticatie, maar geen traffic flow confidentialiteit.

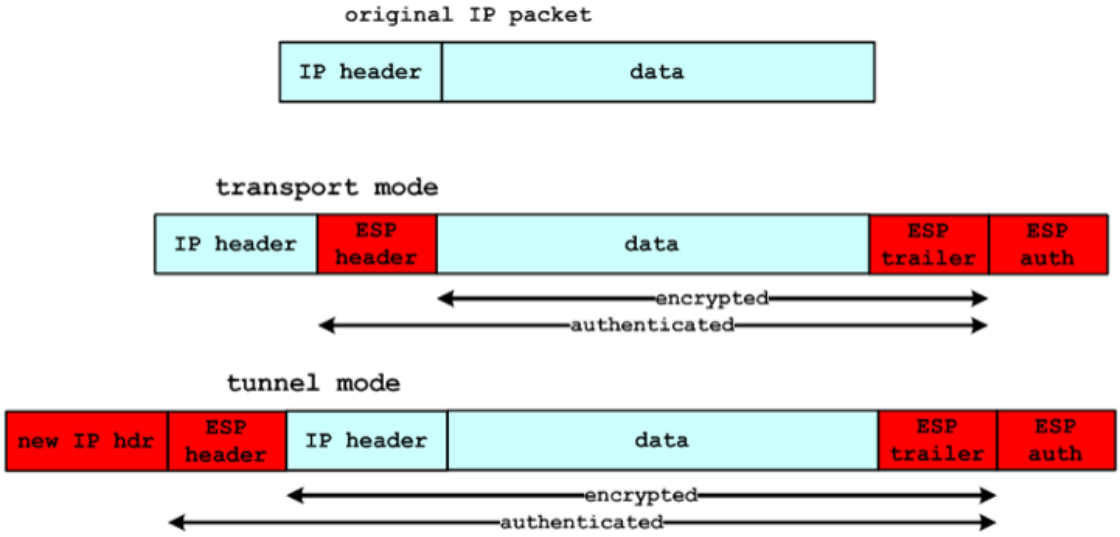
  In tunnel mode biedt het wel traffic flow confidentialiteit. Een aanvaller kan enkel zien naar welk lokaal netwerk een pakket werd verstuurd, maar weet niet naar welk specifiek toestal van dit netwerk het verstuurd werd.

Beide protocollen kunnen ook gecombineerd worden in beide modi (figuur 2.10).

2.3.3 Beveiligen van de datalinklaag

Mogelijke aanvallen op de datalinklaag

- **Content Address Memory aanval**: Het blijven versturen van valse MAC adressen zodat de CAM tabel vol geraakt. Op die manier kunnen geen nieuwe MAC adressen meer toegevoegt worden.
- **Address Resolution Protocol spoofing**: ARP is het mappen van een IP adres naar het bijhorende macadres. Er kunnen echter vervalste ARP berichten verstuurt worden doorheen



Figuur 2.9: Encapsulating Security Payload.

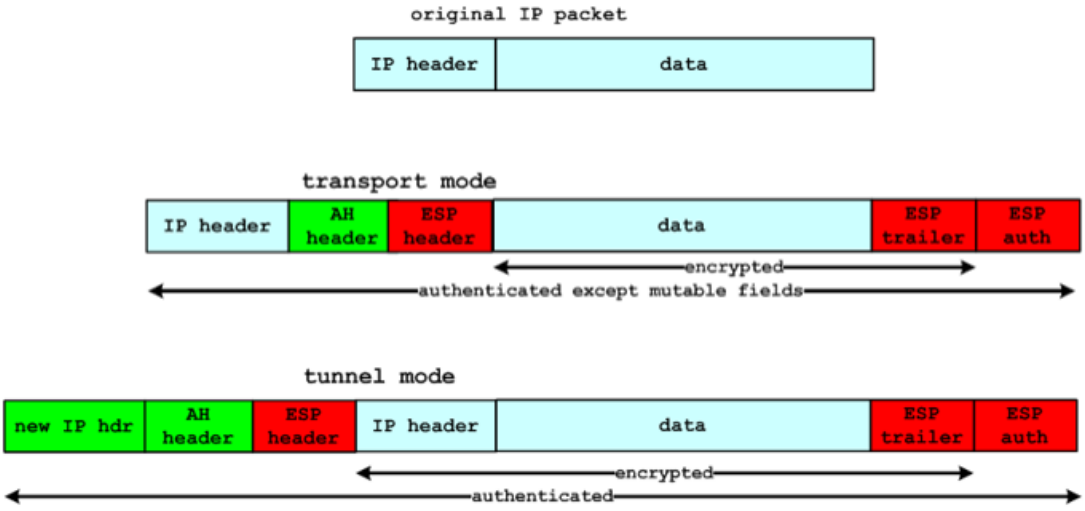
het netwerk, zodat het IP adres van een willekeurige host gemapt wordt op het MAC adres van de aanvaller. Op die manier krijgt de aanvaller alle berichten die eigenlijk bestemd waren voor de oorspronkelijke host.

- **DHCP starvation**: Het blijven versturen van DHCP requests.

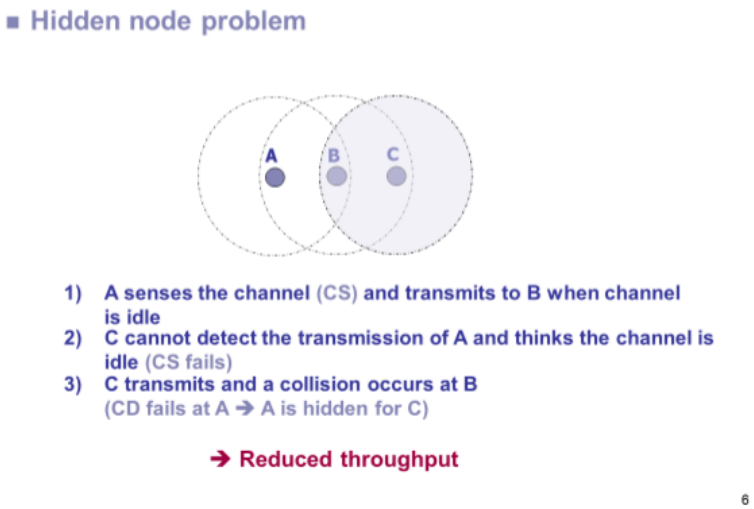
Deze lijst kan nog uitgebreidt worden indien men spreekt over draadloze netwerken:

- **Pakketten zijn afluisterbaar**
- **Deauthenticatie aanval**: Om van een draadloos netwerk te gaan, moet er een deauthenticatierequest verstuurd worden naar het access point waarop er geauthenticeerd werd. Een aanvaller kan in de naam van iemand anders zo een deauthenticatierequest versturen.
- **Hidden node probleem** (figuur 2.11):

ToDo: saai



Figuur 2.10: AH + ESP.



In a wireless network many hosts or nodes are sharing a common medium. If nodes A and C are both wireless laptop computers communicating in an office environment their physical separation may require that they communicate through a wireless access point B. Consider the scenario with three wireless devices as shown in the figure. The transmission range of A reaches B, but not C (the detection range does not reach C either). The transmission range of C reaches B, but not A. Finally, the transmission range of B reaches A and C, i.e., A cannot detect C and vice versa. A starts sending to B, C does not receive this transmission. C also wants to send something to B and senses the medium. The medium appears to be free, the carrier sense fails. C also starts sending causing a collision at B. But A cannot detect this collision at B and continues with its transmission. A is hidden for C and vice versa. Since only one device can transmit at a time in order to avoid packet collisions, packet loss occurs.

Figuur 2.11: Hidden node problem.

Hoofdstuk 3

Encryptiealgoritmen

3.1 Geschiedenis

Encryptiemethoden worden al eeuwenlang gebruikt om informatie onleesbaar te maken voor partijen die deze informatie niet mogen achterhalen. Deze inleiding bespreekt de basismethoden om encryptie toe te passen.

Zogenaamde **substitution ciphers** zijn de meest eenvoudigste vorm van encryptie. De originele versie, ook wel Caesar cipher genoemd, vervangt elke letter van het alfabet met een voorgedefinieerde shift in positie. Een shift van 3 zal het originele alfabet (in kleine letters) afbeelden op het verschoven alfabet (in grote letters): Het duidelijke nadeel is dat er slechts 25 verschillende encryp-

a b c d e f g h i j k l m n o p q r s t u v w x y z
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

tiemogelijkheden zijn. Een verbeterde versie mapt elke letter met een willekeurige andere (nog niet gebruikte) letter: waardoor er nu 26! $\approx 4 \cdot 10^{26}$ mogelijkheden zijn. Deze manier is daarom niet meer

a b c d e f g h i j k l m n o p q r s t u v w x y z
D K V Q F I B J W P E S C X H T M Y A U O L R G Z N

secuur, aangezien de onderliggende frequentie van letters nog steeds dezelfde is. Een eenvoudige dencryptiemethode is om de relatieve frequenties van de ciphertekst te tellen, waardoor er statistisch kan achterhaald worden met welke gedecrypteerde letter elke geëncrypteerde letter overeenkomt.

Een encryptiealgoritme wil ook de relatieve frequenties verbergen. Een methode dat dit implementeert is de Vigenère Cipher. Deze bevat een sleutel $K = k_1 k_2 ... k_d$ waarbij k_i het i-de alfabet specifieert dat gebruikt moet worden. Na d letters start men terug vanaf k_1.

Een laatste voorbeeld van een substitutiecipher is de Playfair cipher. Hier wordt er een $5x5$ matrix opgesteld, waarbij eerst het sleutelwoord ingevuld wordt, gevolgd door de niet gebruikte letters in alfabetische volgorde. Dit wordt uitgewerkt met de sleutel `MONARCHY` in Tabel 3.1. Een bericht wordt in paren van letters geëncrypteerd. Stel dat we het woord ballon willen encrypteren, de paren letters worden: ba lx lo nx. Er wordt een x toegevoegd indien er twee dezelfde letters achter elkaar komen, en als het eindpaar uit slechts één letter bestaat. Er kunnen zich drie gevallen voordoen bij elk paar:

1. Als beide letters in dezelfde rij voorkomen, wordt elke letter vervangen door de letter die er rechts van ligt: (on → na).

M	O	N	A	R
C	H	Y	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

Tabel 3.1: Playfair cipher.

2. Als beide letters in dezelfde kolom voorkomen, wordt elke letter vervangen door de letter die er onder van ligt: (ba → ib).
3. Anders wordt elke letter vervangen door de letter van de kolom van de andere letter: (lx → su).

Er zijn $26 * 26 = 676$ mogelijke diagrammen die gemaakt kunnen worden.

Na de **substitution ciphers** zijn er ook **transposition ciphers**. Zulke ciphers gaan letters niet vervangen, maar gaan ze echter verplaatsen. Dit heeft natuurlijk als nadeel dat de relatieve frequentie van de letters behouden wordt. Twee eenvoudige voorbeelden zijn:

1. Keer elke letter om:
   `A SIMPLE EXAMPLE` → `ELPMAXE ELPMIS A`
2. Keer de woordvolgorde om, en elk woord keert ook de lettervolgorde om:
   `A SIMPLE EXAMPLE` → `A ELPMIS ELPMAXE`

Een iets beter voorbeeld is de Rail Fence cipher. Deze heeft als private sleutel het aantal rijen dat gebruikt wordt. Elke letter zal diagonaal geschreven worden, uitgewerkt in Tabel 3.2 op de zin `DEFEND THE EAST WALL`: Geëncrypteerd is dit dus `DNETLEEDHESWLXFTAAX`.

D				N				E				T				L		
	E		E		D		H		E		S		W		L		X	
		F				T				A				A				X

Tabel 3.2: Rail Fence cipher.

De laatste methode die besproken wordt is de columnar transposition cipher. Een bericht wordt in rijen geschreven over een bepaald aantal kolommen. Hierna worden de kolommen gesorteerd op basis van de private sleutel $K = k_i k_j ... k_n$. Stel $K = k_3 k_4 k_2 k_1 k_5 k_6 k_7$ en plaintext `ATTACK POSTPONED UNTIL TWO AM`: Op Tabel 3.3 wordt deze plaintext uitgeschreven in rijen en kolommen. De ciphertext wordt dan `TTNAAPTMTSUOAODWCOIXKNLYPETZ`.

A	T	T	A	C	K	P
O	S	T	P	O	N	E
D	U	N	T	I	L	T
W	O	A	M	X	X	X

Tabel 3.3: Rail Fence cipher.

Uiteindelijk kan men ook de combinatie maken van **substitution** en **transposition** ciphers, wat de basis vormt van moderne cryptografie.

3.2 Symmetrische algoritmen

Vooraleer er in detail kan ingegaan worden op symmetrische algoritmen, moet eerst de term **block cipher** besproken worden. Een block cipher wil zeggen dat informatie in blokken zullen verstuurd worden en kan best vergeleken worden met een substitutiecipher, maar dan toegepast op blokken. Een typische blokgrootte varieert van 8 tot 128 bytes en wordt door de meeste algoritmen gebruikt. De **Feistel Cipher encryptie** vormt de basis van moderne blockciphers en maakt gebruik van twee primitieve encryptieoperaties:

- Substitutie of Diffusion (de S-box)
- Permutatie of Confusion (de P-box)

Het Feistel encryptieschema maakt gebruik van n ronden, een functie F en de sleutel voor ronde i, K_i. De data wordt opgesplitst in twee delen. Op het linkerdeel wordt altijd een substitutie uitgevoerd, gebaseerd op F. Bij de start van een volgende ronde worden beide delen van plaats verwisseld, zodat nu het rechterdeel de functie heeft als linkerdeel. Het decryptieschema werkt op exact dezelfde manier, maar beginnend van de geëncrypteerde data. De sleutels K_i worden dan wel in omgekeerde volgorde behandeld.

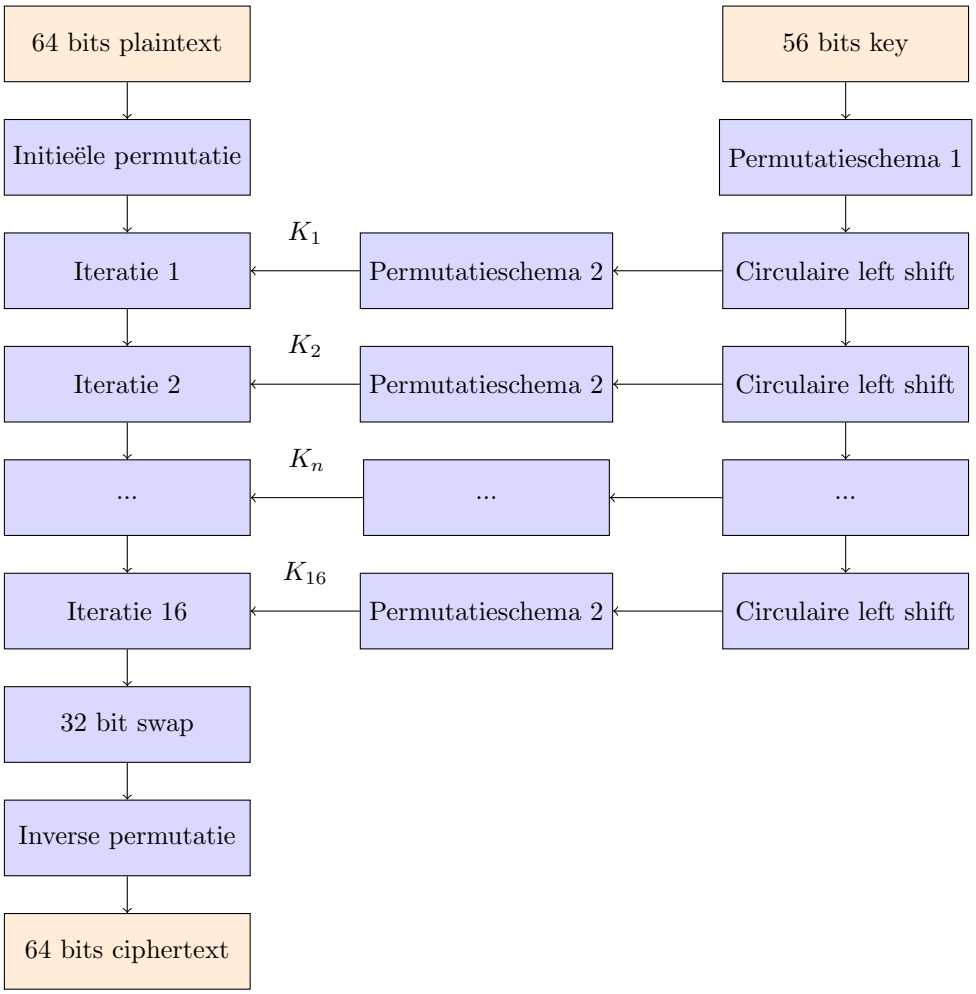
3.2.1 DES & 3-DES

Het **DES (Data Encryption Standard)** algoritme maakt gebruik van het reeds vermelde Feistel schema waarbij de **blokgrootte 64 bits** en **sleutelgrootte 56 bits** bedraagt. Deze methode is redelijk traag, ondanks de kleine blokgrootte, en wordt door de kleine sleutelgrootte ook als onveilig beschouwd. Een 56 bit sleutel kan namelijk binnen de 10 uur gekraakt worden indien het aantal operatiers per **micro**seconde 10^6 bedraagt.

Het algoritme maakt gebruik van 16 ronden in het Feistel schema. De 56 bit sleutel wordt gebruikt om iteratiesleutels $K_1, K_2, ...K_{16}$ te bepalen, die elk 48 bits groot zijn. Voor de eerste ronde en na de laatste ronde wordt er een extra permutatie uitgevoerd, waarbij de laatste permutatie het inverse is van de eerste permutatie. Het resultaat van dit algoritme is één van de 2^{64} uitvoermogelijkheden van de 64 bits.

De 16 vereiste iteratiesleutels worden in het begin van het algoritme aangemaakt. Eerst en vooral wordt de 56 bit sleutel gepermuteerd, afhankelijk van het gekozen permutatieschema, op Figuur 3.1 permutatieschema 1 genoemd. Een permutatieschema mapt een bit van de oorspronkelijke sleutel op een bit van de gepermuteerde sleutel, zodat er 2^{64} mogelijke schemas bestaan. De gepermuteerde sleutel wordt hierna beschouwd als twee resultaatsleutels C_n en D_n, met n het huidig rondenummer, van elk 28 bits groot. Beide resultaatsleutels worden onafhankelijk van elkaar verwerkt door een circulaire left shift van 1 of 2 bits, afhankelijk van de vorige ronde, dat eenvoudig door de conditionele operator kan beschreven worden: $(n-1)\ \%\ 2 == 0\ ?\ 2\ :\ 1$. Indien C_2 en D_2 de resultaatsleutels zijn van ronde 2, dan zijn C_3 en D_3 de resultaatsleutels van ronde 3 door respectievelijk C_2 en D_2 2 bits naar links te permuteren. Voor elke iteratiesleutel K_i, wordt er een tweede permutatieschema, op Figuur 3.1 permutatieschema 2 genoemd, toegepast op C_nD_n, waarbij de 8 meest significante bits van C_n genegeert worden. Het eindresultaat zijn 16 verschillende iteratiesleutels, die zullen toegepast worden bij elke DES ronde.

Vooraleer het iteratieproces plaatsvindt is er eerst een initiële permutatie van het hele 64-bit blok. Deze permutatie wordt uitgevoerd met behulp van een permutatieschema, zoals bij de sleutelgeneratie. Het gepermuteerde blok wordt hierna opgedeeld in de blokken L_1 en R_1 waarbij de 32 meest significante bits in L_1 komen, en de 32 minst significante bits in R_1, en kan ronde 1 van start gaan. Bij elke iteratie wordt de functie F gebruikt, die twee blokken manipuleert: R_{n-1} en de sleutel van



Figuur 3.1: DES schema.

48-bit groot, die op voorhand al gegenereerd werd. Omdat de functie F bij elke iteratie wordt gebruikt wordt het ook wel de iteratiefunctie genoemd. De uiteindelijke uitvoer van de iteratiefunctie is een blok 32 bits. L en R kan voor een volgende iteratie berekent worden als:

$$\begin{aligned} L_{n+1} &= R_n \\ R_{n+1} &= L_n \oplus F\big(R_n, K_n\big) \end{aligned} \tag{3.1}$$

Hoe werkt de iteratiefunctie? Eerst moet het blok R_n uitgebreidt worden via een selectietabel. Het gebruik van deze selectietabel wordt hier voorgesteld als de functie $E(R_n)$, die als input een blok van 32-bit heeft en zal als output een blok van 48-bit genereren waarbij de bits gegroepeerd per 6 zitten. Sommige bits van R_n worden zullen dus meerdere keren gemapt moeten worden. Na deze uitbreiding wordt het resultaat van $E(R_n)$ en de sleutel K_n met een XOR operatie bewerkt (XOR geeft 0 indien beide bits gelijk zijn, en 1 als ze verschillend zijn). Elke groep van 6 bits stelt een adres voor in een welbepaalde substitutietabel S_i, voor de i-de groep van 6 bits: er zijn dus 8 substitutietabellen. Elk adres wijst naar een 4-bit getal, dat de originele 6 bits zal vervangen, zodat het eindresultaat terug een blok van 32 bits is. Hoe worden deze 4 bits bepaald? De substitutietabel neemt een blok van 6 bits $b_5b_4b_3b_2b_1b_0$ van B als input en voert volgende procedure uit:

1. b_5 en b_0 representeren tesamen een getal van 0 tot en met 3. Stel dit getal gelijk aan i.

2. De tussenliggende bits representeren tesamen een getal van 0 tot en met 15. Stel dit getal gelijk aan j.
3. Zoek in de tabel het getal op de i-de rij en j-de kolom. Dit getal heeft een waarde tussen 0 en 15, en bevat dus slechts 4 bits. Deze bitrepresentatie van dat getal vervangt de 6 bits.

Tot slot moet er nog een permutatie P uitgevoerd worden op de 32 bits verkregen van de substitutiestap, zodat enkel nog de XOR operatie moet gedaan worden met L_n om zo R_{n+1} te bepalen. De iteratiefunctie kort genoteerd:

$$F(R_n, K_n) = P\Big(S\big(E(R_n) \oplus K_n\big)\Big)$$

Block cipher modi

Er zijn verschillende implementaties van het DES-algoritme, en deze hangen dan weer af van de gebruikte block cipher modus. Verschillende mogelijkheden zijn:

- **ECB (Electronic Code Book).** Elk 64-bit blok van de oorspronkelijke informatie wordt individueel geëncrypteerd
- **CBC (Cipher Block Chaning).** Elk cipherblok is nu afhankelijk van elkaar, door een initiële XOR-operatie uit te voeren met het vorige geëncrypteerde blok, en het te encrypteren blok. Maakt ook gebruik van een initialisatievector en moet gekend zijn voor de zender en ontvanger.
- **CFB (Cipher Feedback).** Gelijkaardig aan CBC, maar _ToDo: wat?
- **OFB (Output Feedback)** _ToDo: ik vind dit maar raar uitgelegd
- **CTR (Counter)**

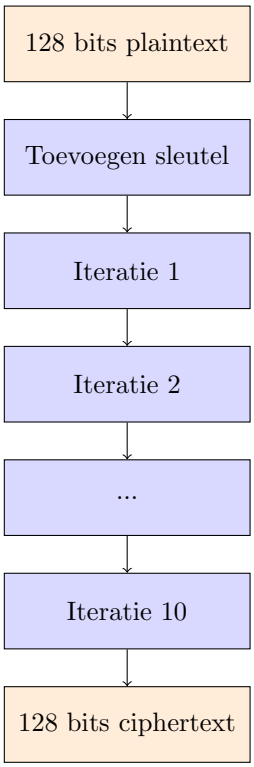
3-DES

Omdat de opgelegde sleutelgrootte van **56 bits** voor DES onvoldoende is, gaan we DES drie maal na elkaar uit te voeren. De sleutelgrootte bedraagt nu **168 bits**. Dit is natuurlijk nog trager dan DES, wat al behoorlijk traag is. De sleutel van 168 bits wordt nu opgedeeld in 3 sleutels van elk 56 bits, K_1, K_2 en K_3. De plaintext wordt eerst in geëncrypteerd met DES en sleutel K_1, de ciphertext hiervan wordt gedecrypteerd met DES en sleutel K_2, de plaintext hiervan wordt terug geëncrypteerd maar met sleutel K_3. 3DES is nog altijd nutteloos, en wordt aangezien als een zwak encryptiealgoritme.

3.2.2 AES

_ToDo: no clue what w[x, y] wil zeggen + DECRYPTIE (SLIDE 96)

Het **AES (Advanced Encryption Standard)** algoritme is ook een symmetrisch encryptiealgoritme waarbij de **blokgrootte 128 bits** en **sleutelgrootte 128, 192 of 256 bits** bedraagt. Het werd speciaal ontworpen om cryptoanalyse moeilijk te maken en is zelfs sneller dan 3DES op 32-bit toestellen. Op Figuur 3.2 wordt het schema getoond voor een blokgrootte van 128 en een sleutelgrootte van 128 bits. Het schema is gelijkaardig voor een sleutelgrootte van 192 of 256 bits, maar hebben dan respectievelijk 12 en 14 iteraties (voor 128 bits zijn er 10 iteraties). Het algoritme is ongeveer 2,5 maal sneller dan het DES algoritme (200 Mbit/s versus 80 Mbit/s) en werkt uitsluitend voor CBC of CTR block cipher modi. Bovendien is het heel goed **paralleliseerbaar**.



Figuur 3.2: AES schema.

3.2.3 Andere symmetrische algoritmen

- Blowfish / twofish.
- RC5
- CAST5
- ...

3.3 Asymmetrische algoritmen

3.3.1 RSA

Rivest-Shamir-Adleman (RSA) is een asymmetrisch encryptiealgoritme waarbij de sleutelgrootte typisch **512, 1024, 2048 of 4096** bits bedraagt. Het encryptie- en decryptieproces voeren dezelfde wiskundige operaties uit. De veiligheid van het algoritme ligt aan het feit dat het ontbinden in factoren van het product van 2 priemgetallen p en q moeilijk is. De sleutel genereren begint van twee priemgetallen p en q, waarvan het product $n = p \cdot q$ wordt berekend. De priemgetallen p en q zijn **privaat**, het getal n is **publiek**. Op n wordt de Eulers totiënt functie losgelaten. Aangezien dat p en q beiden priemgetallen zijn is dit eenvoudig:

$$\phi(n) = (p-1) \cdot (q-1)$$

($\phi(n)$ geeft het aantal getallen terug in het bereik $[1, n]$ die relatief priem (grootst gemene deler = 1) zijn ten opzichte van n. Bij een priemgetal zal elk positief getal, kleiner dan het priemgetal, de grootst gemene deler altijd 1 zijn)

De berekende waarde $\phi(n)$ is **privaat**. Verder wordt een getal e, dat ook **publiek** is, gekozen zodat

$$gcd(\phi(n), e) = 1 \text{ en } 1 < e < \phi(n)$$

Tot slot wordt een getal d, dat **privaat** is berekend als:

$$d = e^{-1} \mod \phi(n)$$

De publieke sleutel bestaat dan uit e en n: $KU = \langle e, n \rangle$

De private sleutel bestaat uit d en n: $KR = \langle d, n \rangle$

Voorbeeld

Stel $p = 13$ en $q = 19$. Het product wordt dan $p \cdot q = 247 = n$.

$\phi(n) = (13 - 1) \cdot (19 - 1) = 12 \cdot 18 = 216$

Stel nu $e = 5$ want $gcd(\phi(n), e) = gcd(216, 5) = 1$ en $1 \leq e \leq \phi(n)$

$d = 5^{-1} \mod 216 = 173$

$KU = \langle 5, 247 \rangle$

$KR = \langle 173, 247 \rangle$

Gegeven een bericht $M = 71$

Encryptie: $C = 71^5 \mod 247 = 67$

Decryptie: $M = 67^{173} \mod 247 = 71$

Er kan ook een handtekening gemaakt worden met de private sleutel: $S = M^d \mod n$. De handtekening verifiëren gebeurd dan met de publieke sleutel: $M = S^e \mod n$.

3.3.2 Andere asymmetrische algoritmen

ToDo: niet belangrijk

3.4 Hashalgoritmen

Een hashfunctie H bewerkt een bepaald bericht M zodat $H(M) = h$. Hierbij wordt er op voorhand vastgelegd hoeveel bits h zal hebben. Zo zal $MD5$ een hashwaarde van 128 bits genereren, en $SHA1$ een hashwaarde van 160 bits. Een hash wordt gebruikt om wijzigingen te detecteren in een bericht, aangezien dat een wijziging in het bericht ook een andere hashwaarde zal opleveren.

Het verschil met encryptiealgoritmen

- Een hashalgoritme is performanter, omdat ze een andere functie hebben. Ze dienen niet om een bericht te encrypteren, want een hashfunctie kan niet gedecrypteerd worden.
- De sleutelgrootten en datablokken bij hashfuncties zijn veel groter dan die van encryptiealgoritmen. Hashfuncties kunnen ook van sleutel verwisselen per datablok.
- Hashalgoritmen zijn resistent tegen sleutel-gebaseerde aanvallen.

3.4.1 MD5

Message Digest (MD5) genereert hashwaarde met grootte **128 bits**. Dit aantal bits is eigenlijk niet genoeg om **strong collision resistance** te hebben, en wordt daarom niet als een goed hashalgoritme beschouwd. De uitwerking van dit hashalgoritme is dan ook niet interessant.

3.4.2 SHA1

Dit hashalgoritme genereert een hashwaarde die **160 bits** bevat. De **strong collision resistance** requirement is hier al veel beter, maar cryptoanalyse heeft did hashalgoritme ook in meerwaarde doen dalen en is ook trager dan $MD5$. Het hashalgoritme werkt op blokken van **512 bits**.

3.4.3 SHA2

Deze uitbreiding op $SHA1$ omvat eigenlijk 4 versies, die best in een tabel kunnen beschreven worden:

Versie	grootte datablok	outputgrootte
SHA224	224 bits	512 bits
SHA256	256 bits	512 bits
SHA384	384 bits	1024 bits
SHA512	512 bits	1024 bits

3.5 MAC-algoritmen

Een MAC kan niet gebruikt worden als een encryptiealgoritme, omdat het net zoals een hash een onomkeerbaar proces is.

3.5.1 CBC-MAC

Een symmetrische encryptie van het hele bericht met behulp van de CBC-mode. Het laatste blok wordt gebruikt als de MAC. Deze implementatie kan dus gebruik maken van een symmetrisch encryptiealgoritme zoals DES en AES.

3.5.2 HMAC

ToDo: onbelangrijk

Hoofdstuk 4

Software- en systeembeveiliging

4.1 Veilige applicaties

4.1.1 Email

PGP

Staat voor Pretty Good Privacy en biedt beveiligingtools voor dataopslag en emailverkeer. Het maakt gebruik van betrouwbare algoritmen zoals AES, RSA en SHA1.

Een sleutelpaar aanmaken gebeurd op basis van de identiteit, de geldigheidsperiode en een paszin. Er zijn twee mogelijkheden:

- Een sleutelpaar voor de digitale handtekening. Dit wordt ook wel de *master key* genoemd.
- Eén of meer sleutelparen voor encryptie. Deze worden *subkeys* genoemd.

Om authenticate te garanderen, moet de **verzender** volgende acties ondernemen:

- Genereer een hashwaarde van het bericht.
- Encrypteer deze hashwaarde met een asymmetrisch encryptiealgoritme met de private sleutel van de verzender.
- Optioneel kan dit bericht nog gecompresseerd worden en omgevormd worden tot een radix-64 formaat.

De verzender stuurt nu het plaintext bericht, samen met de geëncrypteerde sleutel naar ontvanger. De **ontvanger** van dit bericht onderneemt dan volgende stappen:

- De optionele conversie van radix-64 en de compressie uitvoeren.
- De hashwaarde moet gedecrypteerd worden met de publieke sleutel van de verzender.
- De verzender berekend nu ook de hashwaarde van het bericht, en vergelijk dit met de gedecrypteerde hash.

Als de hashes gelijk zijn, dan is het bericht geauthenticeerd. Zijn de hashes niet gelijk, dan werd het bericht niet geëncrypteerd door de verzender die verwacgt wordt.

Om confidentialiteit te garanderen, moet de **verzender** volgende acties ondernemen:

- Genereer een sessiesleutel, dat enkel geldig is voor het huidig bericht.
- Optioneel wordt het bericht gecompresseerd.
- Het bericht wordt geëncrypteerd met een symmetrisch encryptiealgoritme met de sessiesleutel in CFB mode.
- De sessiesleutel wordt ook geëncrypteerd met de ontvanger zijn publieke sleutel, gebruik makend van een asymmetrisch encryptiealgoritme.
- Optioneel wordt het bericht geconverteerd naar radix-64 formaat.

Het geëncrypteerde bericht kan nu verzonden worden naar de **ontvanger**, die nu volgende stappen overloopt:

- Eventueel de radix-64 transformatie ongedaan maken.
- De sessiesleutel wordt gedecrypteerd met de private sleutel van de ontvanger.
- Het bericht kan nu gedecrypteerd worden met deze sessiesleutel.
- Optioneel wordt het bericht gedecompresseerd.

Hoe kunnen authenticatie en confidentialiteit gecombineerd worden?

- Eerst wordt authenticatie toegevoegd. Dit bericht wordt dan geëncrypteerd.

Omdat een gebruiker meerdere encryptiesleutelparen kan hebben, moet er een eenvoudige manier bestaan om de juiste op te halen. Daarvoor wordt een sleutel geïdentificeerd door 64 bits, zodat er 2^{64} mogelijke sleutels zijn, met als gevolg dat de kans op collisie klein is. Het is dan ook eenvoudig om de juiste sleutel op te halen, op basis van deze 64 bits. PGP bewaard deze sleutels in **twee sleutelhangers**. De publieke sleutelhanger en de private sleutelhanger.

- **Private sleutelhanger**: Deze houdt volgende informatie bij per sleutel:
  - De 64 bit identificatie.
  - De correspondeerde publieke sleutel.
  - De private sleutel, geëncrypteerd met een symmetrisch encryptiealgoritme.
  - Een identificatie van de gebruiker.

  Bij het aanmaken van een private sleutel wordt er gebruik gemaakt van een paszin. Op basis van deze paszin wordt er een tijdelijke sleutel aangemaakt die gebruikt zal worden van deze encryptie. Om zo een private sleutel te decrypteren, moet deze paszin opnieuw ingegeven worden, zodat dezelfde tijdelijke sleutel kan gegenereerd worden.
- **Publieke sleutelhanger**: Deze houdt volgende informatie bij per sleutel:
  - De 64 bit identificatie.
  - De publieke sleutel
  - Een identificatie van de gebruiker.
  - Informatie over de betrouwbaarheid van de sleutel.
  - Digitale handtekeningen van andere gebruikers van deze sleutel.

  - Informatie over de betrouwbaarheid van deze digitale handtekeningen.

  De publieke sleutelhanger van een gebruiker, bevat dus publieke sleutels van andere gebruikers. De publieke sleutelhanger is echter niet publiek voor andere gebruikers, maar enkel voor de eigenaar.

PGP doet heel veel beroep op vertrouwen. Hier kunnen drie aspecten betrokken worden:

- △ **Vertrouwen in de eigenaar van de publieke sleutel**: Zo zijn er verschillende niveaus zoals: *ultimate trust*, *always to be trusted to sign other keys*, *usually to be trusted to sign other keys*, *usually not to be trusted to sign other keys*. Ultimate trust en 'always to be trusted' lijken op het eerste zicht op elkaar, maar ultimate trust dient enkel voor sleutels die ook in de private sleutelhanger zitten.
- △ **Vertrouwen in de digitale handtekening van een publieke sleutel**: PGP zal de betrouwbaarheid van de eigenaar die de handtekening gezet heeft opzoeken.
- △ **Vertrouwen in de link tussen de eigenaar en een publieke sleutel**: PGP zal periodiek een betrouwbaarheidsniveau aangeven op basis van de digitale handtekeningen op deze publieke sleutel.

Een sleutel x is compleet geldig als:

- △ er een pad bestaat van gesigneerde sleutels die terug naar x verwijst in 5 stappen of minder.
- △ x getekend werd door voldoende andere sleutels.

S/MIME

Multipurpose Internet Mail Extensions (MIME) beschrijft een formaat voor de inhoud van een email die de limitaties van **RFC 822** en **SMTP** oplost, maar toch compatibel blijft met **RFC 822**. **Securei Multipurpose Internet Mail Extension (S/MIME)** voegt extra beveiligingselementen toe.

Een standaard **RFC 822** bericht bestaat uit een aantal header lines zoals **from**, **to** en **subject**, gevolgd door een lege lijn en de body van het bericht. Deze standaard in combinatie met **SMTP** heeft volgende problemen:

- ! Ondersteund slechts 7-bit ASCII karakters.
- ! Het versturen van executables is moeilijk.
- ! Er bestaat geen standaardimplementatie van SMTP.
- ! Sommige servers converteren de berichten (tabs naar spaties, CR LF naar LF, whitespace verwijderen, ...)

MIME definieert vijf nieuwe headers:

- △ `MIME-VERSION`
- △ `CONTENT-TYPE`: beschrijving van de structuur van de inhoud, zodat de ontvanger de juiste conversiemethode kan kiezen. Een aantal voorbeelden zijn:
  - text/plain

  - image/jpeg, image/gif
  - video/mpeg
  - audio/basic
  - text/enriched
  - multipart/mixed
  - application/octet-stream

- `CONTENT-TRANSFER-ENCODING`: laat toe om het bericht om te vormen naar een formaat dat accepteerbaar is voor emailverkeer. Voorbeelden zijn:
  - 7bit
  - 8bit
  - binair
  - base64
  - x-token
- `CONTENT-ID`
- `CONTENT-DESCRIPTION`: beschrijving van de inhoud van het bericht. Wordt meestal gebruikt wanneer de inhoud nietleesbaar is, zoals audio.

S/MIME-functies:

- Enveloped data: de inhoud en sleutels worden geëncrypteerd voor 1 of meerdere gebruikers.
- Signed data: het bericht krijgt een digitale handtekening. Zowel de inhoud als deze handtekening worden geëncodeerd in base64.
- Clear-signed data: zelfde als signed data, maar enkel de handtekening wordt geëncodeerd in base64. Dit heeft als gevolg dat een ontvanger zonder S/MIME het bericht wel kan lezen, maar niet kan verifiëren.
- Signed and enveloped data: combinatie van encryptie en digitale handtekening.

Door deze functies kan S/MIME nieuwe content-types definieren:

- multipart/signed: clear signed data
- application/pkcs7-mime: signed data, enveloped data of een entiteit met enkel publieke sleutels (degenerate signed data), afhankelijk van de parameter.
- application/pkcs7-signature: enveloped data
- application/pkcs10-mime: certificaatregistratie nodig

De S/MIME procedure:

1. MIME bericht wordt aangemaakt.
2. Bericht wordt omgevormd tot een PKCS object, die onder andere de gebruikte algoritmen en certificaten bevat.
3. PKCS object wordt gezien als een message body, en wordt verpakt in MIME.
4. Het bericht moet omgevormd worden naar canonieke vorm.

4.1.2 Webapplicaties

Mogelijke zwaktes in een webapplicatie:

- Δ **Slechte configuratie**: Gebruik maken van oude software, paswoorden die makkelijk te raden zijn, source code die gedownload kan worden.
- Δ **Client-side**: De client-side vertrouwen is een slechte zaak. Het is eenvoudig om met javascript manipulaties uit te voeren. Controleer elke actie op de server-side.
- Δ **Direct object reference**: De applicatie bevat resources die enkel mogen gezien worden voor een geautoriseerde gebruiker, maar iedereen kan eigenlijk gewoon de URL intypen.
- Δ **Authenticatiefouten**: Het gebruik van zwakke paswoorden, die gekraakt kunnen worden met een brute-force methode. Ook als een gebruiker een fout paswoord, maar wel een juist username intypt, moet de applicatie toch tonen dat het ofwel een fout paswoord of fout usernaam is.
- Δ **Cross-site scripting (XSS)**:

4.2 Veilige systemen

4.2.1 Authenticatiemethoden

...

4.2.2 Vertrouwd besturingssysteem

Veilig	$\neq$	Vertrouwd
Iets is ofwel veilig of niet veilig		Een vertrouwd systeem kent gradaties en is vaak relatief

Een besturingssysteem is vertrouwd indien het consistent:

- geheugen kan afschermen,
- toegangscontrole ondersteund,
- gebruikers kan authenticeren.

Een policy moet gedefinieerd worden vooraleer er kan nagegaan worden ofdat een besturingssysteem vertrouwd is of niet. Een voorbeeld hiervan is een militair beveiligingsmodel (unclassified, restricted, confidential, secret, top secret). Het wil uiteraard niet zeggen dat als iemand toegang heeft tot 'secret', dat die dan alle informatie van de 'secret', 'confidential', 'restricted' en 'unclassified' tak kan bekijken. Er komt nog een bijkomende toegangscontrole. Elk stukje informatie kan gelinkt worden aan een compartiment (informatie die bij elkaar hoort, bv informatie over een bepaalde bom).

Voorbeeld: De informatie <secret, België> kan gelezen worden door iemand met <top secret, België>, <secret, België>, maar niet door iemand met <top secret, Nederland>. Een stukje informatie kan in meerdere compartimenten zitten.

Er bestaand drie toegangscontrolemethoden:

- **DAC (Discretionary Access Control)**: De gebruiker die de informatie aanmaakt is ook de eigenaar. Deze gebruiker bepaalt dan ook wie de informatie mag zien.
- **MAC (Mandatory Access Control)**: Enkel de administrator bepaalt de rechten van de informatie. Dit brengt veel overhead mee voor de administrator zelf.
- **RBAC (Role-Based Access Control)**: Elke gebruiker bevindt zich in een een groep of heeft een bepaalde rol. Deze rollen/groepen krijgen authorisaties toegekend.

ToDo: os kernel

4.2.3 Schijfencryptie

= het encrypteren van de harde schijf. Dit zit ofwel standard in het besturingssysteem, of kan via een softwareapplicatie gerealiseerd worden.

Er zijn drie manieren om een schijf te encrypteren:

- **Manueel**: Laat de gebruiker zelf toe om individuele bestanden of mappen te encrypteren. Een voorbeeld hiervan is *PGP*.
- **Filesystem-level**: Het bestandssysteem zal zelf bestanden encrypteren en decrypteren. Een toepassing hiervoor is het gebruik van shares (onedrive, google drive, lokale share ergens, ...). De bestanden of mappen die geëncrypteerd zijn, zullen ook op die shares in geëncrypteerde vorm staan. Deze methode zal metadata (eigenaar, laatst gewijzigd, grootte) niet encrypteren.
- **Volledige schijfencryptie**: Deze methode encrypteert heel de schijf, inclusief metadata en tijdelijke bestanden. Het staat wel open voor een aantal aanvallen zoals een cold boot aanval: het geheugen uitlezen van het RAM door het systeem te resetten, want de databits in het geheugen blijven voor een beperkte tijd bestaan ook al is het toestel uitgezet. Een andere aanvalsvector zijn keyloggers.

Mogelijke features van schijfencryptie zijn:

- **Plausible deniability**: Encryptie zorgt ervoor dat iemand niet kan bevestigen dat de plaintext informatie bestaat.
- **Hidden containers**: op basis van een wachtwoord slechts een deel van het bestandssysteem beschikbaar maken.
- **Pre-boot authentication**: Vooraleer de bootprocedure begint al authenticeren. Op deze manier is er geen toegang meer tot de BIOS.
- **Hardware acceleration**: Een extra chip inschakelen die de encryptie op zich neemt zodat het proces versnelt worden.
- **Two-factor authentication**

4.3 Veilige software

4.3.1 Malware

= **Mal**icious soft**ware**. Een systeem kan nog zo goed beveiligd zijn, slechts één malware applicatie is voldoende om een systeem onveilig te maken.

Een aantal voorbeelden van malware:

- **Logische bom**: Een logische bom wordt pas geactiveerd nadat er een bepaald predikaat voldaan is. Deze soort malware wordt vaak intern ontworpen en geplaatst als wraak.
- **Backdoor**: Een backdoor is een manier om een bepaalde toegangscontroleprocedure te omzeilen. Zulke backdoors worden vaak gebruikt om de software te testen, maar wordt daarna niet meer verwijderd uit vergeetachtigheid of luiheid. Het kan ook geïnstalleerd woorden door malware. Een voorbeeld hiervan is een linux kernel update in 2003. Een functie moest het paswoord vergelijken, maar gebruikte de toekenningsoperator (=) in plaats van de vergelijkingsoperator (==). Een toekenning is altijd true.

  Er bestaan verschillende soorten backdoors:

  - *Software backdoor*: Deze worden geplaatst in de code zelf.
  - *Machinecode backdoors*: Deze worden geplaatst in de machinecode omdat ze dan moeilijker te detecteren zijn.
  - *Assymetrische backdoors*: Een assymetrische backdoor laat enkel toe dat de eigenaar van de backdoor toegang krijgt (kleptografie). Een voorbeeld hiervan is de NSA backdoor in de `Dual_EC_DRBG` standaard. Deze elliptische krommen gebruik je dus best niet.
  - *Compiler backdoors*
- **Trojaans paard**: Software kan doen alsof het nuttig is en goede intenties heeft, maar dat dit eigenlijk niet is. Dit soort malware hoopt dat gebruikers de software zullen uitvoeren als administrator. Een trojaans paard wordt vaak gebruikt om andere soorten malware te injecteren.
- **Spyware**: Deze software is passief en zal pogingen ondernemen om gevoelige informatie te bemachtigen. Niet alle virusscanners verwijderen spyware om juridische redenen. Spyware werd vroeger niet gezien als malware, en reclamebureaus hadden hier baat bij. Meer recent laten virusscanners wel toe om spyware te verwijderen, maar met verminderde functionaliteit (of extra betalingen).
- **Adware**: Het vervangen en manipueren van advertenties zodat de aanvaller meer opbrengst krijgt. Als je bijvoorbeeld 'Colruyt' opzoekt, dat je reclame krijgt over Carrefour.
- **Ransomware**: Een deel of het volledige bestandssyteem encrypteren. In ruil voor geld zal ransomware de geëncrypteerde bestanden terug decrypteren. Sommige ransomware, dat niet meer ondersteund is, wordt jammer genoeg nog steeds verspreidt. Gebruikers die dus betalen zullen hun bestanden niet kunnen decrypteren, en zijn hun geld kwijt. Als je toestel door ransomware aangevallen is, kan je beter naar de politie gaan aangezien zij voor de populaire ransomware applicaties over decryptiesleutels beschikken.
- **Scareware**: Dit heeft de bedoeling om de gebruiker af te schrikken met boodschappen zoals 'Het toestel bevat illegale content' of 'Windows is niet meer geactiveerd, betaal nu om te heractiveren'.
- **Virus**: Een virus heeft als eigenschap dat het zichzelf kan verspreiden. Een virus blijft een lange periode stil, om dan na een bepaalde activiation trigger de virus actief te zetten.
- **Worm**: Een worm is een subset van een virus, in de mate dat het geen menselijke interactie nodig heeft om te verspreiden. Een worm zal automatisch pogingen ondernemen om zich te verspreiden over het netwerk. Traditioneel is het gebruik van e-mail applicaties, standaardpaswoorden en bepaalde transportprotocollen. Een worm wordt vaak gebruikt om een botnet op te zetten, die dan gebruikt kan worden om DDoS aanvallen uit te voeren.

Al deze vormen van malware, kunnen via één of meerdere aanvalsvectoren binnendringen:

- **Social Engineering**: De malware doet zich voor als een betrouwbaar programma, vaak door de extentie te verbergen (I_LOVE_YOU.txt.vbs)
- **Besturingssysteem**: Aanvallers doen geen moeite om obscure besturingssystemen te infecteren. Vaak zijn het populaire besturingssystemen (Windows, linux, macOS).
- **Software exploits**: Door fouten in softwareapplicatie zoals e-mail programmas.
- **Buffer overflow**: Op het einde van een fuctie worden de inputwaarden in een buffer gestoken. Indien deze buffer te groot is, wordt de stack en het returnadres overschreven. De buffer bevat dan een nieuw returnadres naar het start van de malware code (zie slide 27 C05c). Om dit tegen te gaan wordt het geheugen opgesplitst in het executable en niet-executable deel. Verder wordt ook de adresruimte gerandomiseerd.
- **Macro virus**: In bijvoorbeeld de Microsoft Office suite kunnen macros geschreven worden die automatisch kunnen uitgevoerd worden wanneer het document geopened wordt. Microsoft heeft dit aangepast zodat macros default niet aan staan.

Malware doet er alles aan om niet gedetecteerd te worden, onder andere door gebruik te maken van volgende technieken:

- Ervoor zorgen dat de 'last modified date' niet aangepast wordt (in windows is het mogelijk om het last modified date attribuut van een bestand programmatisch aan te passen).
- Ervoor zorgen dat de grootte van een bestand niet gewijzigd wordt.
- Wanneer een virusscanner de malware wil scannen, probeert de malware dit tegen te houden of een 'mooie' versie van de malware te geven.
- De code van de malware kan geëncrypteerd zijn.

4.3.2 Software cracking

= het aanpassen van software zodat toegangscontroles worden verwijderd.

Eenvoudige cracks kunnen uitgevoerd worden met behulp van een aantal tools: W32Dasm (een disassembler), Hex Workshop (een hex editor) en OllyDBG (een debugger). Je kan eenvoudig een breakpoint zetten in OllyDBG bij een bepaalde opcode, en de lijn zoeken die bijvoorbeeld de popup toont om een serial nummer in te geven. Vaak staat dit bij een *cmp* statement: indien de gebruiker nog niet geauthenticeerd is, vraag naar de authenticatiecode. De *jne* die daarop volgt kan gewoonweg verwijderd worden en zal het systeem doen alsof de gebruiker wel geauthenticeerd is. In een hex-editor kan bijvoorbeeld het *jne* statement vervangen worden door een *nop* statement, zodat elke authenticatiecode nu geldig is.

Een key generator kan legale sleutels aanmaken. De basis voor illegale keygenerators ligt aan de legale keygenerators, die vaak gebruikt worden in bedrijfscontext, zodat het bedrijf zelf sleutels kan maken en niet elke maal aan bijvoorbeeld Microsoft moet vragen voor nieuwe Office sleutels.

Een authenticerende server kan gesimuleerd worden door een eigen server.

De code kan ook geïnjecteerd worden. Men gaat op zoek naar zogenaamde 'code caves', dit zijn ongebruikte geheugenlocaties in de executable. Deze lege geheugenlocaties kan dan eender welke instructies bevatten.

Hoofdstuk 5

Intrusiedetectie

= moeilijk te definieren.

- Iemand die SSH gebruikt om in te loggen op een systeem? Wat met het root account?
- Iemand die poorten scant?
- Iemand die iets probeert te downloaden van de server?

Om aan intrusiedetectie te doen, moet er een **IDS (Intrusiedetectiesysteem)** zijn. Een goed IDS kan interne en externe hackpogingen monitoren, maar het dient niet om aanvallen tegen te houden.

Attack detection	vs	Intrustion detection
De IDS hoort *buiten* de beveiligingsomgeving.		De IDS hoort *binnen* de beveiligingsomgeving.
Genereert veel overbodige alerts		Detecteert interne hacking.

Het doel van intrusiedetectie is:

- Een aanval detecteren *tijdens* of *na* de aanval (bijvoorbeeld, vanaf dat iemand poorten begint te scannen).
- Kan geen aanval detecteren *voor* het gaat gebeuren.

Intrusiedetectie gaat ervan uit dat aanvallers een ander gedrag hebben dan legitieme gebruikers. Er bestaan wel handelingen die zowel aanvallers als normale gebruikers kunnen uitvoeren, zodat er hierdoor een kleine overlap bestaat. Dit heeft als bijkomend effect dat er false negatives (een aanval doorlaten) of false positives (denken dat een legitieme gebruiker een aanvaller is) zijn.

5.1 Audit records

= bevat logs over de handelingen die gebruikers uitvoeren op het interne netwerk. Elke log bevat een user ID om de juiste gebruiker te traceren.

Er kan gebruikt gemaakt worden van de naïeve, door het besturingssysteem ondersteunde logging-systeem. Dit heeft als voordeel dat er geen extra componenten moeten geïnstalleerd worden, maar het grote nadeel is dat de structuur van de logberichten meestal niet aan de gewenste vorm voldoen. Men gebruikt een speciaal, op maat gemaakt systeem, die enkel nuttige informatie (probe points) bijhoudt voor het IDS en ook nog systeemonafhankelijk is.

Voorbeelden van nuttige informatie in een audit record is:

- het onderwerp, die de actie uitvoert,
- het object, waarop de actie uitgevoerd is,
- de uitgevoerde actie,
- de timestamp.

Met deze informatie kunnen al heel wat 'rare' gebeurtenissen opgespoord worden:

- Gebruikers die inloggen op niet-contionele tijdstippen.
- Accounts waarop veel mislukte inlogpogingen zijn uitgevoerd.
- Onverklaarde reboots/tijdswijzigingen
- ...

Er zijn twee manieren om audit informatie te verzamelen: host based en network based.

- **Host Based**: De informatie kan op het besturingssysteem zelf (via softwarelogs, systeemlogs) gegenereerd worden. De informatie is vrij compact, maar systeemspecifiek.

  Voordelen: Hoge kwaliteit van de informatie.

  Nadelen: Systeemspecifiek, en extra overhead op de toestellen zelf. Ook als een toestel gehackt wordt, kunnen de logs die gegenereerd worden vanuit dat toestel niet meer vertrouwd zijn.

- **Network Based**: In deze versie wordt de informatie verzameld van een hub of switch. Het heeft als doel om netwerkanalyse te gebruiken om aanvallen te detecteren.

  Voordelen: Geen performantieimpact, geen overhead op de toestellen zelf, besturingssysteemonafhankelijk. Deze vorm kan informatie bieden dat Host Based niet kan, namelijk gefragmenteerde pakketten en poort scanning.

  Nadelen: Pakketten kunnen verloren geraken op het netwerk of ze kunnen niet terug samengesteld worden. Besturingssysteemafhankelijke protocollen kunnen mogelijks niet geanalyseerd worden. Geëncrypteerde informatie is evenmin leesbaar.