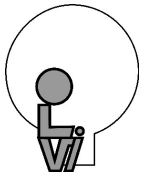
OLVI
Boom
Bovenbouw
Seminarie
Wiskunde



Cryptografie

Michiel STAESSEN
Klas 6.7
Nummer 20
Academiejaar 2006 – 2007
Vakleerkracht: Dhr. De Petter

Gecompileerd met LaTeX op 8 maart 2008

Woord vooraf

Zoals vele leerlingen van een zesde jaar, moest ik dit jaar ook een eindwerk maken.

Gedurende anderhalf trimester heb ik mij toegelegd op de cryptografie. Dit is natuurlijk een zeer uitgebreide wetenschap, maar ik heb geprobeerd om de voornaamste elementen toe te lichten. Ik hoop dat ik erin geslaagd ben de materie duidelijk over te brengen.

Ook wil ik graag enkele mensen bedanken voor de steun die zij verleend hebben gedurende het maken van dit eindwerk.

- In eerste instantie Dhr. De Petter, voor de begeleiding.
- Mijn ouders voor de morele steun.
- ir. Freek van de Griendt voor het aanraden van LaTeX, het uitlenen van zijn cursus, het uitleggen van een bewijs en het meermaals nalezen.
- Mijn zus Sarah, voor het nalezen.
- En iedereen die ik nog vergeten ben, want het is onmogelijk iedereen te vermelden. Bedankt allemaal!

Na het lezen van *Het Juvenalis Dilemma* van *Dan Brown*, is mijn interesse voor cryptografie enorm toegenomen. In combinatie met mijn interesse voor computer werd het enthousiasme eens zo groot. Ik heb dit werk dan ook met het grootste plezier gemaakt.

Mijn voorwoord sluit ik af met de geliefde woorden van Dhr. De Petter, weliswaar gecodeerd...

Irry cyrmvre!!!
Staessen Michiel

Inhoudsopgave

Hoofdstuk 1

Inleiding

1.1 Wat is cryptografie?

Wanneer we het woord ontleden, vinden we de Griekse woorden '$\kappa\rho\upsilon\pi\tau\sigma\sigma$' (geheim) en '$\gamma\rho\alpha\phi\epsilon\iota\nu$' (schrijven) terug, wat samen *geheimschrijven* vormt. De cryptografie is dus een wetenschap die zich bezighoudt met het versleutelen en ontcijferen van informatie (berichten, programma's enz.)

Cryptoanalyse bestaat ook. Dit is de wetenschap die de versleutelde informatie tracht te ontcijferen.

1.2 Geschiedenis van de cryptografie

Cryptografie bestaat al lang. Een afgesproken teken kan een primitieve vorm van cryptografie zijn. Buitenstaanders weten niet noodzakelijk wat het betekent. De cryptografie is zich erg gaan ontwikkelen in oorlogstijd: geen enkel militair leider wil dat zijn plannen bekend raken bij de vijand. De eerste bekende militair is Caesar, die zijn berichten aan Rome versleutelde met het naar hem genoemde Caesaralgoritme (zie verder). Wanneer de boodschapper iets overkwam, zou er toch niets bekend geraken.

De volgende grote ontwikkeling vinden we tijdens de Tweede Wereldoorlog. De codeermachine Enigma (zie verder) was een vernuftig systeem van het Duitse leger. De Engelsen bouwden als reactie een supercomputer die de boodschappen kraakte. Vanaf toen heeft de ontwikkeling niet meer stilgestaan en hebben veel geavanceerdere algoritmes hun ingang gevonden in de wereld.

1.3 Terminologie

Zoals elk vakgebied, beschikt de cryptografie ook over een specifieke terminologie. We overlopen even het hele proces. Alice (Persoon A) wil een bericht verzenden naar Bob (Persoon B) over een onveilig kanaal. Ze wil het bericht dus vercijferen of *encrypten* (Engels: 'to encrypt'). Na de encryptie krijgt ze een nieuw bericht, de *ciphertext*. Deze kan ze versturen over elk kanaal, de boodschap lijkt toch willekeurige onzin. Wanneer Bob het bericht ontvangt gaat

hij het ontcijferen of *decrypten* (Engels: 'to decrypt') om terug het oorspronkelijk bericht te krijgen of de *plaintext*.

Om te encrypten gebruiken we algoritmes die we als een functie E zouden kunnen voorstellen. Zo zouden we kunnen schrijven:

$$E(P) = C \tag{1.1}$$

waarbij de plaintext P versleuteld wordt door de functie E en als resultaat de ciphertext C geeft. De meeste algoritmes maken gebruik van *sleutels*. Hoe deze gebruikt worden, hangt af van de gebruikte methode. Binnen de cryptografie onderscheidt men twee soorten: de *symmetrische (conventionele cryptografie)* of de *asymetrische (public-key cryptografie)*. Bij de conventionele cryptografie gebruikt men dezelfde sleutel om te encrypten en te decrypten. Bij public-key cryptografie daarentegen, heeft men verschillende sleutels nodig. Een publieke sleutel om te encrypten en een private sleutel om te decrypten. Zo kan men schrijven:

$$\begin{gathered} E_{k_1}(P) = C \quad \text{en} \quad D_{k_2}(C) = P \\ \Downarrow \\ P = D_{k_2}(E_{k_1}(P)) \end{gathered} \tag{1.2}$$

Om deze algoritmes veilig te doen verlopen is het van essentieel belang dat de private sleutel niet verkregen kan worden uit de publieke sleutel. Hoe men dit kan verkrijgen, wordt later besproken.

Hoofdstuk 2

Symmetrische cryptografie

2.1 Substitutie

Substitutie betekent vervangen. De substitutiemethoden gaan dus de tekens van de plaintext vervangen door andere tekens en zo de ciphertext vormen.

2.1.1 Caesarvercijfering

De caesarvercijfering is het bekendste, maar vooral het oudste algoritme in de cryptografie. Een letter in het bericht wordt vervangen door een letter die k plaatsen verder staat in het alfabet. Veel keuze is er niet (er zijn 26 sleutels) waardoor deze codering zeer gemakkelijk te kraken is. Je hoeft maar even de mogelijkheden af te gaan tot je een leesbaar bericht verkrijgt.

De caesarvercijfering heeft één eigenaardigheid: wanneer men de letters 13 plaatsen opschuift en men codeert de tekst twee maal, dan bekomt men terug de plaintext. Dit staat gekend als *ROT13*.

2.1.2 Enkelvoudige substitutie

Bij deze methode wordt een letter in het bericht vervangen door een andere, willekeurige (nog beschikbare) letter. Zo verkrijgt men 26! (afgerond $4 \cdot 10^{26}$) mogelijke sleutels. Een nadeel is vast en zeker dat de sleutel minder eenvoudig vast te leggen is.

2.2 Enigma

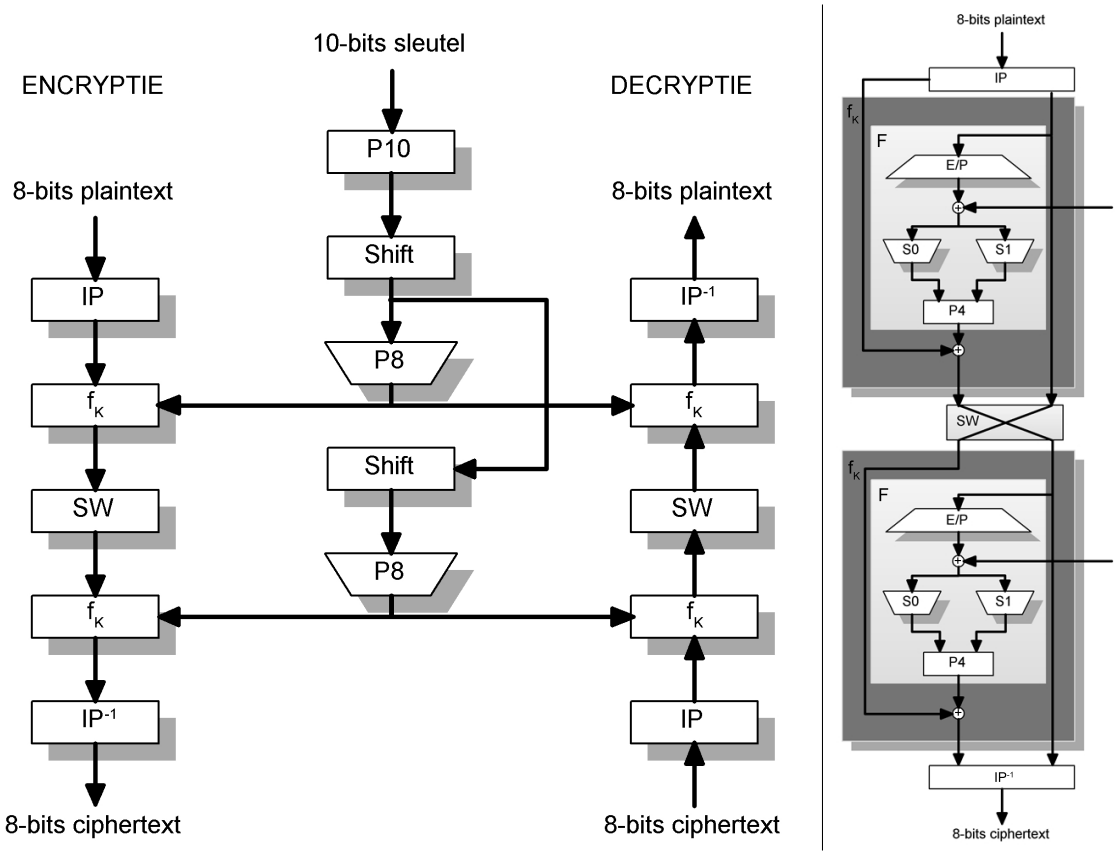
Enigma is een elektromechanische machine van Duitse afkomst die vooral gebruikt werd tijdens de Tweede Wereldoorlog. Het bestaat uit een toetsenbord, een lichtbord (waar de letters van de ciphertext oplichten) en drie rotors. Omdat het kraken van de code makkelijker werd, heeft men achteraf nog een extra rotor toegevoegd. Binnenin elke rotor wordt een substitutie uitgevoerd (door de bedrading binnenin geleidt de stroom naar een andere letter) en elke keer dat een toets ingedrukt wordt, draait de laatste rotor een stapje verder. Wanneer de laatste rotor een volledige omwenteling gemaakt heeft (na 26 stappen), draait de tweede rotor een stapje vooruit en zo gaat het door. Ook beschikt Enigma over een een schakelbord, waarmee men door het verbinden van contacten nogmaals een substitutie kan doorvoeren.

Dit was een zeer doordacht systeem omdat de sleutel voor substitutie bij elke toetsaanslag veranderde. Wat van belang was voor de vercijfering was de beginpositie van de rotors en de verbindingen op het schakelbord. Deze combinaties veranderden dagelijks en werden bijgehouden in boeken.

2.3 Data Encryption Standard

2.3.1 S-DES

S-DES staat voor 'Simplified DES' en is een educatieve variant van DES. S-DES laat namelijk te wensen over wat betreft veiligheid en is dus makkelijk te kraken. Waarom? S-DES werkt met een 10-bitssleutel en dus zijn er maar $2^{10} = 1024$ mogelijke sleutels. Met een brute-krachtaanval (alle mogelijkheden afgaan) is de klus al snel geklaard. Wel is S-DES handig om de structuur van DES te vatten. De structuur ziet eruit als afgebeeld in figuur 2.1.



Figuur 2.1: Links: Simplified DES, rechts: Het encryptiealgotitme uitvergroot

We lichten even de gebruikte afkortingen toe. We beginnen met de plaintext die een beginpermutatie ondergaat ('initial permutation') wat wil zeggen dat de bits van plaats worden veranderd. De functie f_K is een ingewikkelde functie die zowel permutaties als substituties uitvoert. Daarna volgt de functie SWITCH, de eerste vier en laatste vier bits worden verwisseld. Als we figuur 2.1 goed bekijken, merken we op dat het encryptie- en decryptiealgoritme

symmetrisch zijn opgebouwd ten opzichte van elkaar. We bekijken S-DES nu van dichterbij.

Sleutelgeneratie

S-DES gebruikt blokken van 8 bits lang, maar wel een sleutel van 10 bits. Dit lijkt vreemd, maar de 10-bits sleutel is niet de uiteindelijke sleutel. S-DES maakt gebruik van een algoritme dat twee deelsleutels (K_1 en K_2) van 8 bits genereert. De functies die gebruikt worden, zijn vast bepaald. Je kan de werkwijze van de functies wijzigen, maar dan moet je hierin ook consequent zijn, m.a.w. zender en ontvanger moeten de werking van de functies kennen. We bespreken de sleutelgeneratie aan de hand van een voorbeeld.

1. Door een permutatie veranderen de bits van plaats.

$$\text{P10}\,(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10}) = (k_3, k_5, k_2, k_7, k_4, k_{10}, k_1, k_9, k_8, k_6) \tag{2.1}$$

   Zo wordt een sleutel (1010000010) gepermuteerd tot (1000001100).

2. Nu worden de eerste 5 en de laatste 5 bits uiteen getrokken en wordt een verschuiving naar links uitgevoerd (de meest linkse bit van een groepje van 5 bits komt aan de rechterkant van dat groepje). Nu heeft men als sleutel (00001 11000).

3. De volgende stap is de P8-functie. Deze functie bewaart niet alle bits, maar bewaart er 8 van de 10 en gooit ze door elkaar. Het resultaat van deze permutatie is de eerste deelsleutel K_1.

$$\text{P8}\,(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10}) = (k_6, k_3, k_7, k_4, k_8, k_5, k_{10}, k_9) \tag{2.2}$$

   $K_1 = (10100100)$.

4. We gaan verder van waar we nog een 10-bitscode hadden (Stap 2). Nu voeren we een verschuiving naar links uit van 2 plaatsen. (00001 11000) geeft dan (00100 00011).

5. Om K_2 te bekomen, wordt diezelfde P8 functie uitgevoerd. Het resultaat hiervan is $K_2 = (01000011)$

Encryptie

Laten we nu de eigenlijke encryptie behandelen. We behandelen eerst alle stappen, daarna behandelen we een voorbeeld.

1. We hebben reeds gesteld dat de plaintext eerst een beginpermutatie ondergaat.

$$\text{IP}\,(b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8) = (b_2, b_6, b_3, b_1, b_4, b_8, b_5, b_7) \tag{2.3}$$

2. Nu zijn we aanbeland bij de meest ingewikkelde component van S-DES en dus ook van DES, de functie f_K. Deze is een combinatie van permutatie- en substitutiefuncties. Als we L als de 4 meest linkse bits beschouwen en R als de 4 meest rechtse bits, dan kunnen we f_K als volgt definiëren:

$$f_K\,(L,\, R) = (L \oplus F\,(R,\, SK)\,,\, R) \tag{2.4}$$

Laat je niet afschrikken door deze notatie, je bestudeert best even figuur 2.1. In deze notatie is F een afbeelding van 4-bits strings in 4-bits strings is en SK een deelsleutel. Het teken $\oplus$ is de *exclusive OR-functie* (XOR). Deze geeft 1 als uitvoer als slechts één van de twee invoerbits 1 is (wanneer je dus tweemaal 1 of tweemaal 0 hebt, is de uitvoer 0).

Laat ons nu eens kijken naar de afbeelding F. We beginnen met een invoer van 4 bits (b_1, b_2, b_3, b_4). E/P in de figuur 2.1 staat voor Expansie/Permutatie. Van een 4-bits string wordt dus een grotere string gemaakt, in dit geval een 8-bits string. De regel hiervoor is:

$$\text{E/P}\,(b_1, b_2, b_3, b_4) = (b_4, b_1, b_2, b_3, b_2, b_3, b_4, b_1) \tag{2.5}$$

We gaan de gegevens herschijven opdat het geheel een beetje duidelijk blijft. Nu gaan we de deelsleutel $K_1 = (k_{11}, k_{12}, k_{13}, k_{14}, k_{15}, k_{16}, k_{17}, k_{18})$ gebruiken. Deze wordt met de XOR-functie opgeteld bij de string die we tot hier toe hebben.

$$\begin{array}{c|cc|c} b_4 & b_1 & b_2 & b_3 \\ b_2 & b_3 & b_4 & b_1 \end{array} \quad \Rightarrow \quad \begin{array}{c|cc|c} b_4 + k_{11} & b_1 + k_{12} & b_2 + k_{13} & b_3 + k_{14} \\ b_2 + k_{15} & b_3 + k_{16} & b_4 + k_{17} & b_1 + k_{18} \end{array} \tag{2.6}$$

3. We hebben nu een 2x4-matrix. De eerste rij zal de S0-box aansturen, de tweede rij de S1-box. De eerste en de vierde bit van elke rij vormen samen een nieuw binair getal en duiden de rij aan. Het 2-bits getal heeft een bereik van 0 tot en met 3 en we hebben vier rijen. Als we dus een 10_{BIN} verkrijgen, zitten we op de derde rij in de matrix. De tweede en derde bit duiden de kolom aan. Het systeem werkt analoog aan het aanduiden van rijen. De S-boxen zien er zo uit:

$$\text{S0} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \left[\begin{array}{cccc} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{array}\right] \end{array} \qquad \text{S1} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \left[\begin{array}{cccc} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{array}\right] \end{array} \tag{2.7}$$

De elementen in de matrices staan hier uitgedrukt in decimale termen, maar moeten terug als binaire getallen.

4. Als laatste (binnen de functie f_K) voeren we nog een P4-functie uit.

$$\text{P4}\,(b_1, b_2, b_3, b_4) = (b_2, b_4, b_3, b_1) \tag{2.8}$$

5. Nu gaat de invoer door de Switch-functie die de eerste 4 bits wisselt met de laatste 4 en we herhalen de functie f_K maar deze keer met deelsleutel K_2.

6. Als laatste wordt de eerste permutatie ongedaan gemaakt met IP^{-1}, al zijn de bits wel veranderd ondertussen. IP^{-1} is als volgt gedefinieerd:

$$\text{IP}^{-1}\,(b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8) = (b_4, b_1, b_3, b_5, b_7, b_2, b_8, b_6) \tag{2.9}$$

Als je alles perfect uitgevoerd hebt, is de boodschap nu gecodeerd.

Een voorbeeld

Laten we een willekeurige letter nemen. We weten dat een letter 1 byte is en dat een byte 8 bits omvat (zie Bijlage A, pagina 21). Perfect dus voor dit voorbeeld. In dit voorbeeld nemen we de letter M (01001101). We gebruiken de sleutels uit het voorbeeld bij sleutelgeneratie.

1. $IP\,(01001101) = (11000110)$
2. $L = (1100)$, $R = (0110)$
3. $E/P(R) = E/P\,(0110) = (00111100)$
4. $\begin{array}{c|cc|c} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{array}$
5. $\begin{array}{cccccccc} & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ \oplus & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ \hline = & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{array}$
6. 1001 voor de S0-box, 1000 voor de S1-box
   - rij 11 (3), kolom 00 (0) voor S0 $\Rightarrow$ 3 (11)
   - rij 10 (2), kolom 00 (0) voor S1 $\Rightarrow$ 3 (11)
7. resultaat 1111 wordt gepermuteerd tot 1111
8. $\begin{array}{cccc} & 1 & 1 & 1 & 1 \\ \oplus & 1 & 1 & 0 & 0 \\ \hline = & 0 & 0 & 1 & 1 \end{array}$
9. Switch: 1010 0110 $\Rightarrow$ 0110 1010
10. $E/P\,(0011) = (10010110)$
11. $\begin{array}{c|cc|c} 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{array}$
12. $\begin{array}{cccccccc} & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ \oplus & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline = & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}$
13. 1101 voor de S0-box, 0101 voor de S1-box
    - rij 11 (3), kolom 10 (2) voor S0 $\Rightarrow$ 3 (11)
    - rij 01 (1), kolom 10 (2) voor S1 $\Rightarrow$ 1 (01)
14. Resultaat 1101 wordt gepermuteerd tot 1101
15. $\begin{array}{cccc} & 1 & 1 & 0 & 1 \\ \oplus & 0 & 1 & 1 & 0 \\ \hline = & 1 & 0 & 1 & 1 \end{array}$
16. $IP^{-1}\,(10110011) = (11101010)$

Terug naar ASCII tekens omgezet levert dit Ø, wat wel verschillend is van M...

2.3.2 DES

Data Encryption Standard (DES) is een van de meest gebruikte encryptiesystemen. De vercijfering verloopt min of meer gelijk aan S-DES, vandaar dat we dat eerder besproken hebben. DES werkt wel met andere blokgrootten. De plaintext wordt in 64-bits blokken opgedeeld, de sleutel is 56 bits groot (waardoor er dus $2^{56} = 7,2 \cdot 10^{16}$ mogelijke sleutels zijn en een brute-krachtaanval ondoenbaar wordt). Normaal verwacht men – net als bij S-DES – een 64-bits sleutel, maar elke 8ste bit dient ter controle (zie codeertheorie). Ook werkt DES met 16 ronden, d.w.z. dat de functie f_K 16 keer wordt uitgevoerd (16 deelsleutels dus ook), telkens met een functie SWITCH ertussen. De S-boxen zijn ook groter (4x16-matrices).

Hoofdstuk 3

Wiskundige principes

Tot nu toe hebben we alleen met permutaties en substituties gewerkt, maar wat nu als we de wiskunde kunnen inschakelen? Onze berichten worden door substitutie omgezet in getallen (a=11, b=12, enz. of we gebruiken bijvoorbeeld de ASCII-tabel (zie Bijlage A, pagina 21), zolang de ontvanger en de zender maar op de hoogte zijn van de gebruikte substitutie), en worden vervolgens versleuteld volgens een wiskundige bewerking.

3.1 Priem en relatief priem

Een getal groter dan 1 is een priemgetal als het als delers alleen 1 en zichzelf heeft.

Twee getallen zijn relatief priem met elkaar als $\text{ggd}(a, b) = 1$.

De priemgetallen spelen een cruciale rol in de getaltheorie, aangezien elk geheel getal groter dan 1 geschreven kan worden als het product van machten van priemgetallen (priemfactoren).

3.2 Modulaire rekenkunde

3.2.1 Definitie

Een belangrijk principe uit de getaltheorie is de modulaire rekenkunde. Men noemt dit ook wel eens 'klokrekenen' omdat een klok de beste toepassing is van de modulaire rekenkunde. We werken namelijk in een eindige verzameling. Na het grootste element in de verzameling, pikken we terug aan met het eerste element in de verzameling. *We gaan dus eigenlijk op zoek naar de rest bij deling door de modulo.* We zeggen dat een getal congruent modulo is met een ander getal als ze dezelfde rest of restklasse hebben.

Een deeltal $A(x)$ kan omschreven worden als

$$A(x) = Q(x) \cdot D(x) + R(x) \tag{3.1}$$

Waar we naar op zoek gaan bij modulorekenen is de rest na deling door een bepaald getal n zodat $R(x) < n$. We kunnen dit veralgemenen

$\forall a, b, r \in \mathbb{Z}$ en $k \in \mathbb{N}$:

$$a = k \cdot b + r \tag{3.2}$$

Natuurlijk is deze notatie omslachtig, als de formules groter worden. Daarom heeft men de operator $\bmod n$ in het leven geroepen, met n een willekeurig natuurlijk getal. Nu is

$$a \equiv c \bmod n \quad \text{Lees: } a \text{ en } b \text{ zijn congruent modulo } n \tag{3.3}$$

Een getal waarop de operator modulo werd toegepast, zal dus nooit groter zijn dan de modulo. Twee getallen zijn congruent modulo als ze dezelfde rest hebben na deling door n.

Het interessante is dat de modulaire rekenkunde niet onmiddellijk omkeerbaar is. Er zijn oneindig veel getallen a waarvoor vergelijking 3.3 klopt.

3.2.2 Eigenschappen

Wanneer we de operator mod n inschakelen, wordt de verzameling van de gehele getallen omgezet in de verzameling met gehele getallen $\{0, 1, \ldots, (n-1)\}$. De verzameling is dus niet meer oneindig. We definiëren deze verzameling als Z_n waarbij de elementen positieve gehele getallen zijn, kleiner dan n. We zeggen dat dit de verzameling is van de *residuen modulo n*. Bijgevolg moeten we ook de rekenkundige bewerkingen controleren. In deze verzameling Z_n geldt:

commutativiteit

$$(w + x) \bmod n = (x + w) \bmod n$$
$$(w \times x) \bmod n = (x \times w) \bmod n$$

associativiteit

$$[(w + x) + y] \bmod n = [w + (x + y)] \bmod n$$
$$[(w \times x) \times y] \bmod n = [w \times (x \times y)] \bmod n$$

distributiviteit

$$[w \times (x + y)] \bmod n = [w \times (x + y)] \bmod n$$

identiteit

$$(0 + w) \bmod n = w \ mod \, n$$
$$(1 \times w) \bmod n = w \ mod \, n$$

tegengestelde of additieve inverse $-w$

Voor elke $w \in Z_n$bestaat er een z, zodanig dat $w + z \equiv 0 \bmod n$

bijzondere eigenschappen De volgende eigenschappen onderscheidt de modulaire rekenkunde van de normale.

$$\textbf{Als } (a+b) \equiv (a+c) \bmod n \quad \textbf{dan} \quad b \equiv c \bmod n \tag{3.4}$$

Deze bewering bewijst het bestaan van de additieve inverse. Wanneer we in beide leden de additieve inverse van a optellen, vinden we:

$$((-a)+a+b) \equiv ((-a)+a+c) \bmod n$$
$$b \equiv c \bmod n$$

De volgende bewering echter is alleen geldig als a relatief priem is met n

$$\textbf{Als } (a \times b) \equiv (a \times c) \bmod n \textbf{ dan } b \equiv c \bmod n \tag{3.5}$$

Waarom is dat? Wanneer we bewerkingen uitvoeren op de getallen in Z, modulo n bevat de verzameling niet alle elementen. De elementen in Z_n zullen slechts de gemeenschappelijke factoren zijn van a en n. Om toch een volledige verzameling te bekomen, eisen we dat a relatief priem is met n.

multiplicatieve inverse w^{-1} Beschouw een priemgetal p. Dit getal p is dus relatief priem met alle $z < p$. Hieruit kunnen we nog een eigenschap afleiden, de multiplicatieve inverse.

$$\text{Voor elke } w \in Z_p \text{ bestaat er een } z, \text{ zodanig dat } w \times z \equiv 1 \bmod p \tag{3.6}$$

Wanneer we de elementen van Z_p vermenigvuldigen met w, krijgen we opnieuw de verzameling Z_p, weliswaar gepermuteerd. Er is dus minstens één getal in Z_p dat na vermenigvuldiging met w, residu 1 geeft.

OPMERKING: Er bestaan gehele getallen waarvoor men een multiplicatieve inverse kan vinden als p geen priemgetal is. In dat geval zijn a en n relatief priem en kan men een b vinden zodat $a \times b = 1 \bmod n$.

3.3 Het algoritme van Euclides

Om de grootste gemene deler te berekenen, kunnen we gebruik maken van het algoritme van Euclides.

$\forall a, b \in \mathbb{N}$:

$$\text{ggd}\,(a,\ b) = \text{ggd}\,(b,\ a \bmod b) \tag{3.7}$$

Omdat men elke keer en nieuwe b en een nieuwe a verkrijgt, kan men blijven doorgaan totdat $a = 0$. In dat geval is $\text{ggd}\,(b,\ 0 \bmod b) = b$

3.4 Stelling van Fermat

Als p is priem en a is een positief geheel getal, niet deelbaar door p, dan:

$$a^{p-1} \equiv 1 \bmod p \tag{3.8}$$

3.5 Euler

3.5.1 Eulers Phi-functie $\phi(n)$

De phi-functie of totiënt-functie $\phi(n)$ geeft het aantal positieve gehele getallen, kleiner dan een gegeven getal n, en relatief priem met het getal n. De waarde $\phi(1)$ is zonder betekenis, maar heeft per definitie de waarde 1.

Aangezien priemgetallen geen delers hebben, kleiner dan zichzelf en groter dan 1, volgt dat voor een priemgetal p

$$\phi(p) = p - 1 \tag{3.9}$$

Voorbeelden

$$\phi(6) = 2$$
$$\phi(7) = 6$$

Stel dat je twee verschillende priemgetallen p en q hebt, dan

$$\phi(pq) = (p-1) \times (q-1) \tag{3.10}$$

Bewijs. Om dit in te zien, moeten we bedenken dat de verzameling met residuen in Z_n bestaat uit $\{0, 1, \ldots, (pq-1)\}$. De residuen die *niet* relatief priem zijn met n, vormen drie verzamelingen.

- $\{p, 2p, \ldots, (q-1)\,p\}$ $(q-1)$ elementen
- $\{q, 2q, \ldots, (p-1)\,q\}$ $(p-1)$ elementen
- $\{0\}$ 1 element

De ϕ-functie geeft het *aantal* getallen die relatief priem zijn met n, m.a.w. het aantal getallen in Z_n min het aantal getallen in de drie deelverzamelingen hierboven beschreven. Zodus kan men stellen

$$\begin{aligned}
\phi(n) &= pq - (q-1) - (p-1) - 1 \\
&= pq - (p+q) + 1 \\
&= (p-1) \times (q-1) \\
&= \phi(p) \times \phi(q)
\end{aligned}$$

□

3.5.2 De stelling van Euler

De stelling van Euler is eigenlijk een veralgemening van de kleine stelling van Fermat, ze zegt dat als a en n positief zijn:

$$a^{\phi(n)} \equiv 1 \bmod n \tag{3.11}$$

Laat ons deze stelling even bewijzen, want deze zal belangrijk zijn voor RSA.

Bewijs. Stel dat n een priemgetal is, dan verkrijgen we de stelling van Fermat, aangezien $\phi(p) = p - 1$ voor p een priemgetal. We kunnen deze stelling echter uitbreiden voor elke $n \in \mathbb{Z}^+$. We definiëren een verzameling R met alle elementen die relatief priem zijn met n. Deze heeft dus $\phi(n)$ elementen, omwille van de definitie van $\phi(n)$.

$$R = \{x_1, x_2, \ldots, x_{\phi(n)}\}$$

We vermenigvuldigen elk element in deze verzameling met $a \bmod n$ en bekomen de verzameling

$$S = \{ax_1 \bmod n, ax_2 \bmod n, \ldots, ax_{\phi(n)} \bmod n\}$$

De verzameling S is dus eigenlijk een permutatie van R omdat:

1. a is relatief priem met n en x_i is relatief priem met n. ax_i is bijgevolg ook relatief priem met n. Bovendien zijn al deze elementen kleiner dan $n \pmod n$.
2. In S bestaan geen duplicaten (zie vergelijking 3.5) Als $ax_i \bmod n = ax_j \bmod n$ dan $x_i = x_j$ en dit is onmogelijk omdat er in R geen duplicaten zitten.

Hieruit volgt:

$$\begin{aligned}
ax_1 \bmod n \cdot ax_2 \bmod n \cdot \ldots \cdot ax_{\phi(n)} \bmod n &\equiv x_1 \cdot x_2 \cdot \ldots \cdot x_{\phi(n)} \bmod n \\
&\Updownarrow \\
\prod_{i=1}^{\phi(n)} (ax_i) &\equiv \prod_{i=1}^{\phi(n)} x_i \pmod n \\
&\Updownarrow \\
a^{\phi(n)} \cdot \prod_{i=1}^{\phi(n)} (x_i) &\equiv \prod_{i=1}^{\phi(n)} x_i \pmod n \\
&\Updownarrow \quad \text{(Zie vergelijking 3.5)} \\
a^{\phi(n)} &\equiv 1 \bmod n
\end{aligned}$$

□

De stelling komt ook in een andere vorm voor. We vermenigvuldigen beide leden met a. Aangezien $a < n$ kunnen we stellen dat:

$$a^{\phi(n)+1} \equiv a \bmod n \tag{3.12}$$

We ontwikkelen een uitvloeisel van deze stelling om later de geldigheid van RSA aan te tonen. Gegeven zijn 2 priemgetallen p en q, en gehele getallen $n = pq$ en m, met $0 < m < n$. Dan geldt de volgende relatie:

$$m^{\phi(n)+1} = m^{(p-1)(q-1)+1} \equiv m \bmod n \tag{3.13}$$

Als m en n relatief priem zijn, geldt de relatie (zie Euler, vergelijking 3.12). Stel nu dat $\text{ggd}\,(m, n) \neq 1$. Dan wil dit zeggen dat ofwel p, ofwel q een deler is van m.

We bekijken een geval waar m een veelvoud is van p, zodat $m = cp$ voor een $c \in \mathbb{Z}^+$. In elk geval moet $\text{ggd}\,(m, q) = 1$, anders is m zowel veelvoud van p als q en toch $m < pq$. Als $\text{ggd}\,(m, q) = 1$ dan geldt de stelling van Euler en is

$$m^{\phi(q)} \equiv 1 \bmod q$$

Volgens de regels van de modulaire rekenkunde geldt:

$$\left[m^{\phi(q)}\right]^{\phi(p)} \equiv 1^{\phi(p)} \bmod q$$
$$m^{\phi(n)} \equiv 1 \bmod q$$

Er bestaat een waarde k, zodat

$$m^{\phi(n)} = 1 + kq$$

Als we beide leden vermenigvuldigen met $m = cp$, dan

$$m^{\phi(n)+1} = m + kcpq = m + kcn$$
$$m^{\phi(n)+1} \equiv m \bmod n$$

Analoog kan men deze stelling bewijzen voor m een veelvoud van q. Een alternatieve vorm van deze stelling zal gebruikt worden om de geldigheid van RSA te staven.

$$\begin{aligned} \left[m^{\phi(n)}\right]^k &\equiv 1^k \bmod n \\ m^{k\phi(n)} &\equiv 1 \bmod n \\ m^{k\phi(n)+1} = m^{k(p-1)(q-1)+1} &\equiv m \bmod n \end{aligned} \tag{3.14}$$

Hoofdstuk 4

Asymmetrische cryptografie

4.1 Diffie-Hellman

In 1976 werd door Diffie en Hellman een baanbrekende verhandeling gepubliceerd. Whitfield Diffie en Martin Hellman introduceerden hierin de public-key cryptografie. De nieuwe vorm van cryptografie had enkele eisen:

- Er zijn twee sleutels, een voor encryptie, en een voor decryptie
- Het is ondoenlijk (zelfs met de krachtigste machine) om de decryptiesleutel te achterhalen als alleen het algoritme en de publieke sleutel gekend zijn.

Deze techniek blies een nieuw leven in de cryptografie want hiermee ontstonden nieuwe mogelijkheden:

- Encryptie en decryptie op basis van twee verschillende sleutels
- *Authenticatie* met behulp van een *digitale handtekening* (zie sectie 4.2)
- Sleuteluitwisseling

Een van de elementen in de verhandeling is het mogelijk maken van sleuteluitwisseling over een onveilig kanaal. Stel dat Alice en Bob een sleutel overeen willen komen. Het protocol gaat als volgt:

1. Alice en Bob komen een priemgetal p en een getal g (priemfactor mod p) overeen.
2. Alice kiest een willekeurig geheim getal a en berekent $g^a \bmod p$ en stuurt het resultaat naar Bob
3. Ook Bob kiest een willekeurig geheim getal b en berekent $g^b \bmod p$ en stuurt het resultaat naar Alice.
4. Alice berekent $\left(g^b \bmod p\right)^a \bmod p$
5. Bob berekent $\left(g^a \bmod p\right)^b \bmod p$

Omdat $g^{ab} = g^{ba}$ hebben zowel Alice en Bob na stappen 4 en 5 dezelfde sleutel. Deze hebben zij verkregen zonder dat ze het geheim getal van de andere kennen. De getallen a en b trachten te achterhalen is ondoenbaar gezien de definitie van de modulaire rekenkunde.

4.2 Hash-functies en Authenticatie

Hash-functies worden gebruikt bij het digitaal ondertekenen van documenten. Dit algoritme vervormt het bericht zodanig, dat de oorspronkelijke boodschap wordt herleid tot een korte reeks tekens, compleet onbegrijpbaar voor een mens. Ook een hash-functie voldoet aan enkele eisen:

- De grootte van de input maakt niet uit.
- De output heeft een vaste lengte.
- De output is relatief makkelijk en snel te berekenen.
- De hash-functie werkt maar in één richting, men kan ze niet decoderen.
- De hash-functie is collisievrij, d.w.z. dat verschillende berichten nooit dezelfde output kunnen krijgen.

De output van de hash-functie wordt een *Message Digest* (MD) genoemd, letterlijk een samenvatting van het bericht. Deze digest kan men versleutelen met zijn eigen private sleutel. De ontvanger ontvangt dus een archiefbestand met de al dan niet versleutelde boodschap, de hash-functie en de versleutelde message digest. De ontvanger kan nu het bericht controleren op echtheid.

1. Voer de hash-functie uit op het bericht.
2. De versleutelde message digest wordt ontcijferd met de public key van de zender.
3. De ontvanger vergelijkt de output van 1 en 2.

Als de twee message digests niet overeenstemmen is er iets misgegaan en wordt de handtekening niet als officieel erkend.

- Het bericht werd gewijzigd nadat de zender het ondertekend heeft.
- Iemand geeft zich uit voor de zender, en zijn private key klopt dus niet.
- Er trad een fout op gedurende de transmissie van de gegevens. De codeertheorie moet dit probleem uitsluiten, maar in dit werk gaan we daar niet verder op in.

Een veelgebruikt algoritme is MD5 en wordt vaak in PHP scripts gebruikt om wachtwoorden op te slaan in databases.

4.3 RSA

De verhandeling van Diffie en Hellman was een soort oproep tot een nieuw en veiliger encryptiealgoritme. Het daagde alle cryptologen uit om een algoritme te vinden dat voldeed aan alle eisen van de public-key cryptografie. In 1977 verscheen RSA, een systeem ontwikkeld door Rivest, Shamir en Adleman, dat aan de eisen van public-key cryptografie voldeed. Het kan zowel voor encryptie/decryptie gebruikt worden als voor digitale handtekeningen (authenticatie).

4.3.1 Werking RSA

Het RSA-systeem maakt gebruik van expressies met exponenten. Om te versleutelen, deelt men de plaintext op in blokken. Elk blok heeft een bepaalde grootte n. De binaire waarde van een blok is dus steeds kleiner dan n. In de praktijk zal de blokgrootte n dus altijd als 2^k te schrijven zijn.

Voorbeeld:

$$11111111_{\text{BIN}} = 255_{\text{DEC}} < 256 = 2^8 = n$$

Een plaintext-blok M wordt dus omgezet naar C op de volgende manier:

$$C = M^e \bmod n \tag{4.1}$$

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n \tag{4.2}$$

Uit de definitie van de public key cryptografie volgt dat er twee sleutels nodig zijn: een publieke $\text{KU} = \{e, n\}$ en een private $\text{KR} = \{e, d\}$. Beide partijen moeten dus de bloklengte kennen. De bloklengte n bekomt men door het product te berekenen van twee verschillende grote (zo'n 100 decimale cijfers) priemgetallen p en q. Zodus bestaat n uit ongeveer 200 cijfers. De moeilijkheidsgraad bij het kraken ligt hem in de grootte van n. Als men p en q kan achterhalen, kan men de volledige vercijfering kraken. De factorisatie (het splitsen in priemfactoren) is echter een tijdrovende bezigheid.

Om aan de eisen van public key cryptografie te voldoen, moeten de volgende voorwaarden gerealiseerd worden

- Het is mogelijk om e, d, n te vinden, zodat $M^{ed} \equiv M \bmod n$ voor elke $M < n$.
- Het is relatief makkelijk M^e en C^d te berekenen voor alle waarden van $M < n$.
- Het is ondoenlijk om d te bepalen als alleen e en n gegeven zijn.

We bekijken eerst wat we moeten doen om te mogen stellen dat $M^{ed} = M \bmod n$. Herinner u stelling 3.14:

$$m^{k\phi(n)+1} = m^{k(p-1)(q-1)+1} \equiv m \bmod n \tag{4.3}$$

Als we de gewenste relatie (4.2) willen krijgen, moet

$$ed = k\phi(n) + 1$$

We kunnen dit anders formuleren:

$$ed \equiv 1 \bmod \phi(n)$$
$$d \equiv e^{-1} \bmod \phi(n)$$

Dit wil zeggen dat e en d multiplicatieve inversen $\bmod n$ zijn. Volgens de regels van de modulaire rekenkunde kan dit alleen maar als d (en daardoor ook e) relatief priem is met $\phi(n)$. Dus $\text{ggd}(\phi(n), d) = 1$. Nu hebben we alle noodzakelijke elementen voor RSA verklaard.

- p en q, twee priemgetallen (privaat, gekozen)
- $n = pq$ (publiek, berekend)
- e, waarbij ggd $(\phi(n), e) = 1$ en $1 < e < \phi(n)$ (publiek, gekozen)
- $d \equiv e^{-1} \bmod n$ (privaat, berekend)

Wanneer Alice nu een gecodeerd bericht wil sturen naar Bob, moet ze eerst zijn publieke sleutel KU $= \{e, n\}$ opzoeken. Vervolgens codeert zij haar bericht in blokken M, kleiner dan n door M^e, modulo n te berekenen. Deze blokken C stuurt ze door naar Bob. Bob berekent $C^d \bmod n = M^{ed} \bmod n \equiv M \bmod n$, waarmee hij dus terug de oorspronkelijke tekst heeft.

In de praktijk vraagt men voor een veilige encryptie en decryptie voor n een lengte van ongeveer 200 decimale digits. Dit wordt omschreven in (4).

Een praktisch voorbeeld is hier niet uitgewerkt, maar je kan altijd de praktijk toetsen met het script voor RSA encryptie en decryptie (Zie 5.2.3 op pagina 20).

Hoofdstuk 5

Toepassingen

5.1 Software

5.1.1 Enigma

Enigma is niet uitgebreid beschreven in dit werk. Je kan wel het programmaatje Enigma Simulator 6.3 (freeware) downloaden (`http://users.telenet.be/d.rijmenants/EnigmaSim.zip`). Hiermee kan je zelf aan de slag en je ziet hoe bij elke toetsaanslag de rotors een plaatsje verder draaien.

5.2 Eigen brouwsels

Ter illustratie zijn bij dit eindwerk enkele webapplicaties gemaakt in PHP. Deze zijn via te bereiken via
`http://staessen.1111mb.com/cryptografie`.

Sta mij toe even de scripts bij te lichten...

5.2.1 ROT 13

We zagen reeds de caesarvercijfering[1]. Deze verschuift de letters een bepaald aantal plaatsen. Een bijzondere variant is ROT13, die de letters 13 plaatsen verschuift, waardoor men na tweemaal coderen terug de plaintext verkrijgt.

PHP biedt voor dit voorbeeld een interessante functie str_rot13(). De broncode stelt dus niet veel voor, maar het is een handig script voor wie snel een boodschap wil coderen.

5.2.2 S-DES

We zagen de vereenvoudiging van DES, genaamd S-DES. Ook van dit algoritme heb ik een script gemaakt. Echter, dit is niet zo dynamisch als de andere scripts.

[1] Zie 2.1.1 op pagina 3

Als input moet je een 8-bits string ingeven. Vanaf dan kan je volgen wat er allemaal gebeurt tijdens het algoritme. Omdat S-DES met twee cycli werkt, wordt het script herhaald na de functie Switch (even op de link klikken).

5.2.3 RSA

Het meest interessante algoritme is het RSA-algoritme. Voor alle aspecten van het algoritme is er een script voorzien.

Eerst hebben we de Φ-functie. Hiermee kan je zowel nagaan of een getal priem is (de output van de Euler Φ-functie is dan één minder dan het opgegeven getal.) of je kan $\Phi(n)$ berekenen.

Een volgende stap is dan het bepalen van de sleutels. n heb je al bepaald aan de hand van je priemgetallen p en q. Nu moet je nog een passende e en d vinden zodat het algoritme deftig kan werken. Het programma geeft als output de mogelijke combinaties van e en d, beide kleiner dan $\Phi(n)$.

Dan hebben we natuurlijk nog de eigenlijke encryptie en decryptie. Om de PHP-servers niet te zwaar te belasten (en omdat er limieten gesteld zijn bij gratis hosting) codeert het script maar een letter per blok. Dit is allesbehalve veilig. Herinner u dat de RSA-makers een waarde voor n aanraadden, die ongeveer 200 decimale digits lang is.

5.2.4 MD5

Last, but not least, de hash-functie MD5. Ook hier stelt de broncode weinig voor, gezien PHP hier de functie md5() standaard heeft ingebouwd.

Bijlage A

ASCII-Tabel

ASCII is een afkorting van American Standard Code for Information Interchange en is een standaard om een aantal letters, cijfers, leestekens en andere symbolen te representeren en aan ieder teken in die reeks een geheel getal te koppelen, waarmee dat teken kan worden aangeduid. De standaard-tabel bestaat uit 128 tekens, de uitgebreide wordt aangevuld naargelang de gebruikte taal. In dit eindwerk wordt de Latin-1 standaard gebruikt, voor de West-Europese talen, vastgelegd door het ISO (International Organization for Standardization) in de ISO-8859-1 standaard.

DEC	HEX	BIN	CODE	DEC	HEX	BIN	CODE
0	0	00000000	NUL	128	80	10000000	€
1	1	00000001	SOH	129	81	10000001	□
2	2	00000010	STX	130	82	10000010	‚
3	3	00000011	ETX	131	83	10000011	f
4	4	00000100	EOT	132	84	10000100	„
5	5	00000101	ENQ	133	85	10000101	…
6	6	00000110	ACK	134	86	10000110	†
7	7	00000111	BEL	135	87	10000111	‡
8	8	00001000	BS	136	88	10001000	ˆ
9	9	00001001	HT	137	89	10001001	‰
10	A	00001010	LF	138	8A	10001010	Š
11	B	00001011	VT	139	8B	10001011	‹
12	C	00001100	FF	140	8C	10001100	Œ
13	D	00001101	CR	141	8D	10001101	Unused
14	E	00001110	SO	142	8E	10001110	Ž
15	F	00001111	SI	143	8F	10001111	Unused
16	10	00010000	DLE	144	90	10010000	Unused
17	11	00010001	DC1	145	91	10010001	‘

DEC	HEX	BIN	CODE	DEC	HEX	BIN	CODE
18	12	00010010	DC2	146	92	10010010	´
19	13	00010011	DC3	147	93	10010011	‟
20	14	00010100	DC4	148	94	10010100	″
21	15	00010101	NAK	149	95	10010101	•
22	16	00010110	SYN	150	96	10010110	–
23	17	00010111	ETB	151	97	10010111	—
24	18	00011000	CAN	152	98	10011000	˜
25	19	00011001	EM	153	99	10011001	™
26	1A	00011010	SUB	154	9A	10011010	š
27	1B	00011011	ESC	155	9B	10011011	›
28	1C	00011100	FS	156	9C	10011100	œ
29	1D	00011101	GS	157	9D	10011101	Unused
30	1E	00011110	RS	158	9E	10011110	ž
31	1F	00011111	US	159	9F	10011111	Ÿ
32	20	00100000	Space	160	A0	10100000	Non-Breaking Space
33	21	00100001	!	161	A1	10100001	¡
34	22	00100010	"	162	A2	10100010	¢
35	23	00100011	#	163	A3	10100011	£
36	24	00100100	$	164	A4	10100100	¤
37	25	00100101	%	165	A5	10100101	¥
38	26	00100110	&	166	A6	10100110	¦
39	27	00100111	’	167	A7	10100111	§
40	28	00101000	(	168	A8	10101000	¨
41	29	00101001	)	169	A9	10101001	©
42	2A	00101010	*	170	AA	10101010	ª
43	2B	00101011	+	171	AB	10101011	«
44	2C	00101100	,	172	AC	10101100	¬
45	2D	00101101	-	173	AD	10101101	
46	2E	00101110	.	174	AE	10101110	®
47	2F	00101111	/	175	AF	10101111	¯
48	30	00110000	0	176	B0	10110000	°
49	31	00110001	1	177	B1	10110001	±
50	32	00110010	2	178	B2	10110010	²
51	33	00110011	3	179	B3	10110011	³
52	34	00110100	4	180	B4	10110100	´
53	35	00110101	5	181	B5	10110101	µ
54	36	00110110	6	182	B6	10110110	¶

DEC	HEX	BIN	CODE	DEC	HEX	BIN	CODE
55	37	00110111	7	183	B7	10110111	·
56	38	00111000	8	184	B8	10111000	¸
57	39	00111001	9	185	B9	10111001	¹
58	3A	00111010	:	186	BA	10111010	º
59	3B	00111011	;	187	BB	10111011	»
60	3C	00111100	<	188	BC	10111100	¼
61	3D	00111101	=	189	BD	10111101	½
62	3E	00111110	>	190	BE	10111110	¾
63	3F	00111111	?	191	BF	10111111	¿
64	40	01000000	@	192	C0	11000000	À
65	41	01000001	A	193	C1	11000001	Á
66	42	01000010	B	194	C2	11000010	Â
67	43	01000011	C	195	C3	11000011	Ã
68	44	01000100	D	196	C4	11000100	Ä
69	45	01000101	E	197	C5	11000101	Å
70	46	01000110	F	198	C6	11000110	Æ
71	47	01000111	G	199	C7	11000111	Ç
72	48	01001000	H	200	C8	11001000	È
73	49	01001001	I	201	C9	11001001	É
74	4A	01001010	J	202	CA	11001010	Ê
75	4B	01001011	K	203	CB	11001011	Ë
76	4C	01001100	L	204	CC	11001100	Ì
77	4D	01001101	M	205	CD	11001101	Í
78	4E	01001110	N	206	CE	11001110	Î
79	4F	01001111	O	207	CF	11001111	Ï
80	50	01010000	P	208	D0	11010000	Ð
81	51	01010001	Q	209	D1	11010001	Ñ
82	52	01010010	R	210	D2	11010010	Ò
83	53	01010011	S	211	D3	11010011	Ó
84	54	01010100	T	212	D4	11010100	Ô
85	55	01010101	U	213	D5	11010101	Õ
86	56	01010110	V	214	D6	11010110	Ö
87	57	01010111	W	215	D7	11010111	×
88	58	01011000	X	216	D8	11011000	Ø
89	59	01011001	Y	217	D9	11011001	Ù
90	5A	01011010	Z	218	DA	11011010	Ú
91	5B	01011011	[	219	DB	11011011	Û

DEC	HEX	BIN	CODE	DEC	HEX	BIN	CODE
92	5C	01011100	\	220	DC	11011100	Ü
93	5D	01011101	]	221	DD	11011101	Ý
94	5E	01011110	^	222	DE	11011110	Þ
95	5F	01011111	_	223	DF	11011111	ß
96	60	01100000	‘	224	E0	11100000	à
97	61	01100001	a	225	E1	11100001	á
98	62	01100010	b	226	E2	11100010	â
99	63	01100011	c	227	E3	11100011	ã
100	64	01100100	d	228	E4	11100100	ä
101	65	01100101	e	229	E5	11100101	å
102	66	01100110	f	230	E6	11100110	æ
103	67	01100111	g	231	E7	11100111	ç
104	68	01101000	h	232	E8	11101000	è
105	69	01101001	i	233	E9	11101001	é
106	6A	01101010	j	234	EA	11101010	ê
107	6B	01101011	k	235	EB	11101011	ë
108	6C	01101100	l	236	EC	11101100	ì
109	6D	01101101	m	237	ED	11101101	í
110	6E	01101110	n	238	EE	11101110	î
111	6F	01101111	o	239	EF	11101111	ï
112	70	01110000	p	240	F0	11110000	ð
113	71	01110001	q	241	F1	11110001	ñ
114	72	01110010	r	242	F2	11110010	ò
115	73	01110011	s	243	F3	11110011	ó
116	74	01110100	t	244	F4	11110100	ô
117	75	01110101	u	245	F5	11110101	õ
118	76	01110110	v	246	F6	11110110	ö
119	77	01110111	w	247	F7	11110111	÷
120	78	01111000	x	248	F8	11111000	ø
121	79	01111001	y	249	F9	11111001	ù
122	7A	01111010	z	250	FA	11111010	ú
123	7B	01111011	{	251	FB	11111011	û
124	7C	01111100	|	252	FC	11111100	ü
125	7D	01111101	}	253	FD	11111101	ý
126	7E	01111110	~	254	FE	11111110	þ
127	7F	01111111	DEL	255	FF	11111111	ÿ

Tabel A.1: Extended ASCII table

Besluit

Tijd om even terug te blikken op mijn eindwerk. Een voorstel voor een onderwerp had ik al klaar voor het schooljaar begon, want ik wist dat dat eraan kwam. Zoals gewoonlijk ben ik onmiddellijk nadat het onderwerp goedgekeurd was veel dingen gaan opzoeken. Daarna is het lang blijven liggen, en toen het min of meer dringend tijd werd dat er wat aan gedaan werd, ben ik toch maar begonnen.

Na er een tijdje stevig aan gewerkt te hebben, is het moeilijk te stoppen. Ik heb veel dingen kunnen integreren die mijn interesseveld dekken. Niet alleen cryptografie zelf, maar ook het programmeren in PHP en het maken van dit eindwerk in LaTeX waren enorm boeiend. Ik heb deze opdracht dan ook als aangenaam ervaren.

Gedurende het hele proces heb ik vele dingen bijgeleerd. Ik hoop dan ook ten zeerste dat met dit eindwerk mijn doelstelling bereikt is: cryptografie min of meer duidelijk maken.

Het feit dat mijn scripts in PHP werken, is ook een hele bevestiging voor mij. Ik heb af en toe met de handen in het haar gezeten omdat er iets maar niet wou werken. Enkele geniale invallen op de meest onverwachte momenten hebben gelukkig de problemen kunnen oplossen.

Bibliografie

[1] Diffie, W. & Hellman, M. E., *New Directions in Cryptography*, in *IEEE Transactions on Information Theory*, vol. IT-22 (6), 1976, pag. 644–654

[2] Haegemans, A., *Toepassingen van de algebra in de informatica*, Cursusdienst VTK vzw, Heverlee, 2001

[3] RSA Laboratories, *Frequently Asked Questions about Today's Cryptography*, 4de druk, `www.rsasecurity.com`, 2000

[4] Rivest, R. L., Shamir, A. & Adleman, L. M., *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, in *Communications of the ACM*, vol. 21 (2), 1978, pag. 120–126

[5] Stallings, W., *Netwerkbeveiliging en cryptografie Beginselen en praktijk*, Academic Service, Schoonhoven, 2000

[6] Tieleman, O. & Vernooij, J., *Cryptografie*, 2002