1 Cryptografie - de wetenschap van geheimen
Benne de Weger

1.1 Cryptografie als Informatiebeveiliging

Cryptografie zie ik als de (technische) wetenschap van het omgaan met geheimen[1]. Daar zit natuurlijk de kant aan van versleuteling: hoe hou je informatie geheim als je het toch uit handen moet geven, bijvoorbeeld aan de postbode. Maar dat is niet het enige: soms is de informatie niet geheim maar wil je wel zeker weten van wie het afkomstig is; dat kan bijvoorbeeld met een handtekening: daar zit ook iets geheims in.

Je kunt ook zeggen: cryptografie gaat over hoe om te gaan met bepaalde *informatierisico's*. Een gebruikelijke indeling van informatierisico's is aan de hand van beveiligingseisen, als volgt:

Authenticiteit: informatie is authentiek, de oorsprong is echt;
Integriteit: informatie is niet gewijzigd, per ongeluk noch expres;
Vertrouwelijkheid: informatie is alleen bekend bij geautoriseerde partijen;
D...: hmm, tja, verzin zelf een mooie term[2].

Gegevensdiefstal is het belangrijkste voorbeeld van gebroken vertrouwelijkheid; identiteitsdiefstal is een mooi voorbeeld van gebroken authenticiteit, en geknoei met de cijfers betekent dat de cijfers niet meer integer zijn.

Authenticatie moet onderscheiden worden van *identificatie* en *autorisatie*: identificatie is: je vertelt iemand je naam; authenticatie is: je laat zien dat die naam klopt, bijvoorbeeld door een paspoort te laten zien; autorisatie is: je hebt bepaalde rechten, bijvoorbeeld het recht om een ander land binnen te reizen.

Privacy ligt wel tegen vertrouwelijkheid aan, maar is een veel breder be-

[1] Er zijn ook niet-technische wetenschappen die zich met geheimen bezighouden, bijv. de psychologie, de sociologie, religiewetenschappen, enz. Een interessant boek over de psychosociale kant is "Het Geheimen Protocol" door Andreas Wismeijer, 2016.

[2] Die D staat er alleen maar bij voor de afkorting. De Engelse grap is beter: Confidentiality, Integrity, Authenticity.

grip, dat gaat over bescherming van de persoonlijke levenssfeer. Vertrouwelijkheid van persoonsgegevens is daar een belangrijk onderdeel van.

Een term als *veiligheid* is een paraplubegrip. Het kan van alles betekenen. Ik leer mijn studenten het af om te zeggen "dit of dat is veilig". Wees specifieker: "dit is vertrouwelijk", "dat is integer", "deze moet authentiek zijn". Dan pas weet je hoe en wat voor cryptografie je nodig hebt.

En wat kan cryptografie dan doen aan deze beveiligingseisen? Vertrouwelijkheid van gegevens die over een openbaar communicatiekanaal moet worden verstuurd (denk aan het Internet) kunnen we beschermen met behulp van *versleuteling*: dat is traditioneel de belangrijkste taak van de cryptografie. Versleuteling heet ook wel *encryptie*[3] of codering[4].

Authenticiteit van papieren documenten wordt vaak geprobeerd te garanderen door een handgeschreven handtekening. Dat geeft namelijk een biometrische soort veiligheid: een vervalste handtekening is door een deskundig grafoloog wel te ontmaskeren. Maar er is nog wel een probleempje: nadat je de handtekening hebt gezet kun je nog wel het document wijzigen, en dat is natuurlijk niet de bedoeling. Het is niet voor niets dat originele aktes bij notarissen en bij de burgerlijke stand in een kluis bewaard worden. Beetje lastig op het internet na te doen. Maar het zal duidelijk zijn dat er behoefte is aan een digitaal mechanisme dat een identiteit van een persoon onlosmakelijk koppelt aan een digitaal document, een soort van digitale handtekening dus. Welnu, dat bestaat, dat heet een *digitale handtekening*, en maakt gebruik van cryptografie. Die heeft bovendien de prettige extra eigenschap dat je aan de handtekening zelf kunt zien of het digitale document gewijzigd is of niet. Dat is mooi. Anders gezegd: digitale handtekeningen geven met authenticiteit ook gratis integriteit.

Vaak wordt gezegd dat integriteit gegeven wordt door een "hash": ook wel een *cryptografisch controlegetal* (*checksum*) genaamd. Dat is maar ten dele waar: zo'n hash beschermt hooguit tegen onopzettelijke wijzigingen (ruis op de lijn), niet tegen opzettelijke. Toch zijn hashes in de cryptografie erg belangrijk als deel-mechanisme van digitale handtekeningen, en ze zijn bijvoorbeeld essentieel voor de werking van Bitcoin. Daarover in latere hoofdstukken meer.

[3] Maar dan krijg je gruwelwoorden als "geëncrypteerd". Praat Nederlands met me.

[4] Ook dat kan mijn goedkeuring niet wegdragen: een code hoeft niets geheims te hebben, denk aan de morsecode. Er is een mooi wiskunde-vak dat *coderingstheorie* heet. Heel nuttig, maar heeft niet per se met geheimen of met crypto te maken (er zijn op codes gebaseerde cryptosystemen, maar daar hebben we het nu niet over). Als iemand over coderen praat en versleutelen bedoelt, dan kun je er donder op zeggen dat je niet met een academische cryptograaf van doen hebt.

Er is nog een niet genoemde beveiligingseis: *beschikbaarheid*: dat informatie er moet zijn als het nodig is. Daar kan cryptografie helaas nou net niks aan bijdragen. Daarom laten we het nu verder buiten beschouwing[5].

Maar nu genoeg geleuterd, we gaan aan het werk. Want we willen weten hoe die wonderbaarlijke cryptografische mechanismen er uitzien, en wat voor leuke wiskunde erachter zit.

1.2 Symmetrische versleuteling

1.2.1 Alice en Bob

Alice en Bob willen vertrouwelijke berichten uitwisselen. Ze kunnen dat alleen doen over een communicatiekanaal waar ze zelf geen controle over hebben, zoals het Internet (denk aan email). Ze moeten ervan uitgaan dat er een afluisteraar is, die we Eva noemen[6]. Voor dit moment is Eva passief: ze kan alles wat over het onveilige kanaal langskomt lezen, maar ze kan er niet op ingrijpen.

Een leesbaar bericht heet *klare tekst*. We noemen dat **m** (van "message"). Alice gebruikt een versleutelingstechniek, met een sleutel **k**, om **m** onleesbaar te maken voor iedereen die niet de sleutel **k** heeft. Die versleutelde tekst heet het *geheimschrift* of *cryptogram*, en geven we aan met **c**. Hier introduceer ik een kleurcode: rood is geheim, groen is openbaar[7].

Alice verstuurt nu het geheimschrift **c** naar Bob, over het communicatiekanaal waar Eva op zit af te luisteren. Geen nood, want Eva krijgt alleen iets onleesbaars te zien. Maar wat heeft Bob nu wat Eva niet heeft? De sleutel **k** natuurlijk. Daarmee kan hij het geheimschrift **c** weer ontsleutelen, leesbaar maken, dus Bob krijgt dan de oorspronkelijke **m** te zien.

Duidelijk is dat Alice en Bob dezelfde sleutel **k** nodig hebben, en dat Eva die sleutel niet in handen mag krijgen. Daar moeten Alice en Bob dus iets op verzinnen. In de digitale wereld is de sleutel **k** natuurlijk gewoon een rijtje bits, zoals alles. Alice moet de sleutel dus niet over hetzelfde kanaal naar Bob sturen, want dan krijgt Eva hem allicht in handen!

[5] Maar het is wel van cruciaal belang natuurlijk. De afkorting CIA betekent eigenlijk Confidentiality, Integrity, Availability, waarbij authenticiteit dan als onderdeel van integriteit wordt gezien. Maar als cryptograaf zie ik de A liever als Authenticity.

[6] Omdat de Engelse vakterm "eavesdropper" is, en niet vanwege de bijbelse connotatie.

[7] Een geheimschrift is groen, dus openbaar. Is dat niet gek? Nee, juist niet!

Alice en Bob moeten dus een veilig communicatiekanaal hebben om de sleutel uit te wisselen. Maar raken we nu niet in een Baron von Münchhausen-situatie: om een veilig communicatiekanaal door versleuteling te maken, moet je eerst een veilig communicatiekanaal hebben om de sleutel uit te wisselen? In de praktijk is dit probleem vaak wel op de één of andere manier op te lossen, vooral omdat een sleutel vaak maar een heel kort rijtje bits is, terwijl berichten vaak heel lang zijn, en omdat je dezelfde sleutel misschien wel voor veel berichten kunt gebruiken. Voor de sleutel loont het dan de moeite om een duur, veilig kanaal te gebruiken (elkaar fysiek ontmoeten, een wachtwoord per brief opsturen, of zoiets), om dan lange tijd een goedkoop onveilig kanaal te gebruiken om geheimschriften over uit te wisselen. Men zegt wel eens:

Versleutelen is: een groot geheim vervangen door een klein geheim.

Toch is dit sleuteluitwisselingsprobleem een echt groot probleem, bijvoorbeeld op het Internet, waar je je communicatiepartner vaak niet in persoon kent, en een apart sleuteluitwisselingskanaal gewoon veel te onhandig of te duur is.

Dit soort versleutelingsmethoden, waarbij dezelfde sleutel wordt gebruikt voor versleutelen en ontsleutelen, heet *symmetrische versleuteling*. Een leuke metafoor is een briefje opsluiten in een geldkistje. Alle historische cryptografische methoden waren symmetrisch. Het grote voordeel van moderne symmetrische versleutelingstechnieken is dat ze ongelofelijk snel zijn (nu al gigabytes per seconde op speciale hardware).

Figuur 1.1: Een metafoor voor symmetrische versleuteling.

1.2.2 Historische voorbeelden

De Caesar-methode, bedacht door de bekende wiskundige Julius Caesar (100 v.Chr. – 44 v.Chr.), werkt op een alfabet met 26 letters: `A`, `B`, ..., `Z`. Een bericht wordt in aparte letters opgeknipt. De sleutel is een getal $\mathbf{k} \in \{1, 2, \ldots, 25\}$ (Caesar zelf koos altijd $\mathbf{k} = 3$). Versleutelen is: vervang elke letter door de letter $\mathbf{k}$ plaatsen verder in het alfabet, waarbij achter de `Z` weer de `A` wordt genomen. Ontsleutelen is dan: vervang elke letter door de letter $\mathbf{k}$ plaatsen terug in het alfabet, waarbij vóór de `A` weer de `Z`

wordt genomen.

De klare tekst $\mathbf{m}$ = VAKANTIECURSUS met $\mathbf{k} = 10$ geeft nu het geheimschrift $\mathbf{c}$ = FKUKXDSOMEBCEC.

De methode van Caesar is een eenvoudig voorbeeld van de *substitutiemethode*, waarbij iedere letter vervangen wordt door een vaste andere letter (dus een vaste permutatie op het alfabet). De sleutel is dan een substitutietabel, zoals

A $\rightarrow$ F, B $\rightarrow$ Q, C $\rightarrow$ B, ..., Z $\rightarrow$ H.

Het aantal mogelijke sleutels is nu veel groter geworden.

Zoek op Internet eens de Vigenère-versleutelingsmethode op, een systeem dat vanaf de 16e eeuw veel gebruikt is. In de 20e eeuw kwamen de versleutelmachines, mechanisch of elektromechanisch, op, zoals de Duitse Enigma (naar een Nederlands ontwerp) en de Hagelin-machines (zie de voorkant van dit boekje).

Het principe van een substitutiesysteem is *confusie*: je wordt in verwarring gebracht omdat een letter iets anders betekent dan je denkt. Maar de tekst wordt nog steeds wel letter voor letter versleuteld, de volgorde blijft hetzelfde. Een ander principe dat kan worden toegepast is *diffusie*: je verandert de posities van letters op een ingewikkelde manier, maar laat de letters zelf wel onveranderd. De simpelste manier is een transpositiemethode: schrijf bijvoorbeeld de klare tekst in een rechthoekig schema van links naar rechts en van boven naar onder, en lees het af van boven naar onder en van links naar rechts. Voorbeeld: schrijf DIFFUSIEWERKTOOK als

```
DIFF
USIE
WERK
TOOK
```

en lees het af als DUWTISEOFIROFEKK.

1.2.3 AES / Rijndael

Een moderne symmetrische methode is AES, ook bekend onder de naam Rijndael. We geven een overzicht van de werking, en zien daar de principes van confusie en diffusie weer in terugkomen. Confusie zorgt ervoor dat iedere bitpositie evenveel kans heeft om in een 0 of een 1 te worden omgezet. Diffusie zorgt ervoor dat het wijzigen van één bit in de klare tekst effect heeft op alle bits in de versleutelde tekst.

AES is een voorbeeld van een blok-versleuteling: altijd wordt een vast aantal bits versleuteld, en het cryptogram heeft evenveel bits als het blok klare tekst. AES werkt met blokken van 128 bits, opgedeeld in 16 bytes.

AES is ontworpen door de Vlaamse cryptografen Vincent Rijmen en Joan Daemen, en was de winnaar van een competitie uitgeschreven door de Amerikaanse overheid. Het is de belangrijkste wereldwijde standaard voor symmetrische versleuteling sinds 2002.

AES werkt met sleutels van 128, 192 of 256 bits. In het geheugen wordt een toestandsmatrix van 4 bij 4 bytes (da's 128 bits) onderhouden. Allereerst wordt het blok klare tekst van 128 bits in de toestandsmatrix geladen. Dan wordt de sleutel "uitgevouwen" tot een aantal "rondesleutels"; hier gaan we verder niet in detail op in. Dan wordt in een aantal rondes telkens een paar relatief simpele stappen herhaald, waarin de toestandsmatrix wordt ververst:

- `AddRoundKey` – de eerste rondesleutel wordt gemengd in de toestandsmatrix;
- 9, 11 of 13 rondes, elk bestaande uit 4 stappen:
  - `SubBytes` – een substitutie met een vaste tabel wordt toegepast op de bytes van de toestandsmatrix (confusie!)
  - `ShiftRows` – de rijen van de toestandsmatrix worden intern verdraaid volgens een vast patroon (diffusie!)
  - `MixColumns` – de kolommen van de toestandsmatrix worden intern gemengd volgens een vast patroon (vooral diffusie!)
  - `AddRoundKey` – de rondesleutel van deze ronde wordt gemengd in de toestandsmatrix;

  (het aantal rondes hangt af van de sleutellengte: 9, 11 of 13 rondes voor sleutels van resp. 128, 192, 256 bits);
- de slotronde, bestaande uit `SubBytes`, `ShiftRows` en `AddRoundKey`.

De resulterende toestandsmatrix bevat nu het cryptogram.

De 4 bij 4 toestandsmatrix heeft 4 rijen en 4 kolommen, en op elk van de 16 posities staat een byte (8 bits). De leesvolgorde is: rij voor rij, en per rij van links naar rechts. Elke byte geven we weer met de zogenaamde *hexadecimale notatie*: twee symbolen uit `0`, `1`, ..., `9`, `A`, `B`, ..., `F`. Hierbij staan `A`, `B`, ..., `F` voor de getallen `10`, `11`, ..., `15`, en elk van deze 16 getallen wordt dan geïnterpreteerd in het binaire stelsel om per getal 4 bits te krijgen. Dus `BA` staat voor $11 \times 16 + 10 = 186$, in bits: $\underbrace{1011}_{\texttt{B}=11}\underbrace{1010}_{\texttt{A}=10}$.

`SubBytes` vervangt elke byte door een vaste andere byte, volgens de vol-

gende tabel.

	·0	·1	·2	·3	·4	·5	·6	·7	·8	·9	·A	·B	·C	·D	·E	·F
0·	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1·	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2·	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3·	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4·	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5·	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6·	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7·	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8·	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9·	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A·	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B·	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C·	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D·	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E·	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F·	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Bijvoorbeeld, `A9` wordt vervangen door `D3` (blauw in de tabel).

`ShiftRows` roteert de rijen van de toestandsmatrix naar links: de tweede rij over één byte, de derde rij over twee bytes, de vierde rij over drie bytes.

`MixColumns` werkt per kolom, door een "matrixvermenigvuldiging" met een vaste matrix:

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 2 \otimes a \oplus 3 \otimes b \oplus & c \oplus & d \\ a \oplus 2 \otimes b \oplus 3 \otimes c \oplus & & d \\ a \oplus & b \oplus 2 \otimes c \oplus 3 \otimes & d \\ 3 \otimes a \oplus & b \oplus & c \oplus 2 \otimes d \end{pmatrix}$$

Maar het is wel een bijzonder soort matrixvermenigvuldiging. Optellen betekent hier de bitsgewijze XOR-operatie $\oplus$, gegeven door $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, en $1 \oplus 1 = 0$. En vermenigvuldigen $\otimes$ gebeurt volgens de regels van een bepaald zogenaamd eindig lichaam. Omdat er alleen met 2 en 3 vermenigvuldigd wordt kunnen we dat makkelijk zo opschrijven:

- $2 \otimes (b_7, b_6, \ldots, b_0) = (b_6, b_5, b_4, b_3 \oplus b_7, b_2 \oplus b_7, b_1, b_0 \oplus b_7, b_7)$,
- $3 \otimes b = (2 \otimes b) \oplus b$

En tenslotte `AddRoundKey`: de rondesleutel (ook altijd 128 bits) wordt eenvoudig bit voor bit bij de toestandsmatrix opgeteld (volgens $\oplus$).

Voor het "uitvouwen" van de sleutel naar de rondesleutels worden varianten van deze technieken gebruikt. Het gevolg is dat in iedere ronde een totaal nieuwe, ogenschijnlijk ongerelateerde, sleutel wordt gebruikt.

Ik hoop dat je met deze uitleg enig idee hebt hoe een modern symmetrisch crypto-algoritme er uitziet. Naast een mix van diffusie en confusie is het ontwerp ook gericht op snelheid, zowel in software als in hardware. Het

aantal rondes is zorgvuldig met een flinke marge gekozen, op basis van allerlei analyses over mogelijke aanvallen, waarbij bijvoorbeeld uitgebreid wordt gekeken naar de voortplanting van één bitverschil, in de sleutel of in de klare tekst.

Ontsleutelen bij AES is simpelweg het achterstevoren doorlopen van het boven beschreven proces.

1.2.4 Kraken

Je hebt vast al wel bedacht hoe substitutie-methoden te kraken zijn, als je weet dat de klare tekst bijvoorbeeld in het Nederlands is geschreven: met *frequentieanalyse*: de meest voorkomende letter van het cryptogram zal wel de versleuteling van de `E` zijn, en zo doorgaand kun je een heleboel raden als je de frequentietabel van het Nederlands weet, en je geheimschrift lang genoeg is.

Omdat bij elke versleutelingsmethode de sleutel maar eindig lang is, kun je in theorie een geheimschrift altijd ontcijferen door met alle mogelijke sleutels te proberen, tot er iets leesbaars uitkomt. Dit heet de *brute-kracht-methode.*

1	2	4	8	16	32	64	128	256	512	
1K	2K	4K	8K	16K	32K	64K	128K	256K	512K	Kilo
1M	2M	4M	8M	16M	32M	64M	128M	256M	512M	Mega
1G	2G	4G	8G	16G	32G	64G	128G	256G	512G	Giga
1T	2T	4T	8T	16T	32T	64T	128T	256T	512T	Tera
1P	2P	4P	8P	16P	32P	64P	128P	256P	512P	Peta
1E	2E	4E	8E	16E	32E	64E	128E	256E	512E	Exa
1Z	2Z	4Z	8Z	16Z	32Z	64Z	128Z	256Z	512Z	Zetta
1Y	2Y	4Y	8Y	16Y	32Y	64Y	128Y	256Y	512Y	Yotta
2^{90}	2^{91}	2^{92}	2^{93}	2^{94}	2^{95}	2^{96}	2^{97}	2^{98}	2^{99}	
2^{100}	2^{101}	2^{102}	2^{103}	2^{104}	2^{105}	2^{106}	2^{107}	2^{108}	2^{109}	
2^{110}	2^{111}	2^{112}	2^{113}	2^{114}	2^{115}	2^{116}	2^{117}	2^{118}	2^{119}	
2^{120}	2^{121}	2^{122}	2^{123}	2^{124}	2^{125}	2^{126}	2^{127}	2^{128}		

Figuur 1.2: De kracht van machten.

Maar dat is theorie. Veel symmetrische methoden met een sleutel van, zeg, n bits, staan alle mogelijke combinaties van die n bits als sleutel toe. Dat betekent dat er 2^n verschillende sleutels zijn om uit te proberen. Dat loopt al gauw verschrikkelijk uit de hand. Zie figuur 1.2, waarin ik dit inzichtelijk probeer te maken.

Een moderne processor kan ongeveer 2^{32} (4 miljard) instructies per seconde uitvoeren. Er zijn al computers verkrijgbaar[8] met $2^6 = 64$ processoren. Een jaar heeft ongeveer 2^{25} seconden. Dus met zo'n beest van een computer doe je er twee jaar over om 2^{64} instructies uit te voeren. En dan zit je nog maar halverwege de verdubbelingen van Figuur 1.2. Er zijn

[8] Gewoon in de winkel, als je de juiste winkel weet te vinden.

processoren die een volledige AES-blok-versleuteling in 2^6 instructies kunnen doen. Met 64 van zulke computers kun je dan binnen één jaar één geheimschrift kraken, mits AES maar een sleutellengte van 64 gehad zou hebben. Met iedere bit extra verdubbelt te benodigde inspanning. Als je hierop doorrekent dan kom je uit op het kunnen kraken van één met een 128-bits AES versleuteld geheimschrift, als je de beschikking hebt over alle 2 miljard computers in de wereld, in slechts 600 miljoen millennia[9].

Dit alles veronderstelt natuurlijk wel dat er geen manier is om AES te kraken, die sneller is dan de brute kracht-methode. Zo'n betere manier is inderdaad niet gevonden. AES is uiterst zorgvuldig ontworpen om de confusie en diffusie optimaal tot hun recht te laten komen. Over het algemeen is het bij symmetrische systemen mogelijk (maar niet makkelijk![10]) om dit niveau van beveiliging te bereiken. En dan is een sleutellengte van 128 bits met een ruime marge voldoende voor de komende decennia. Natuurlijk worden computers nog steeds sneller, en kan er dus ook sneller gekraakt worden. De Wet van Moore, die al sinds de jaren 60 onveranderd van kracht blijkt te zijn, zegt dat de rekenkracht van processoren (bijvoorbeeld gemeten per uitgegeven dollar) ongeveer elke anderhalf jaar verdubbelt. Dat betekent dat je dan een 1 bit langere sleutel zou moeten gaan nemen om de hoeveelheid werk voor de aanvaller op hetzelfde onhaalbare niveau te houden. Voor de snelheid van versleutelen en ontsleutelen betekent 1 bit extra vrijwel niets. Men zegt wel: dat computers steeds sneller worden werkt alleen maar in het voordeel van de goeden, juist niet van de krakers.

1.3 Asymmetrische versleuteling

1.3.1 Publieke sleutels (?)

Er zijn andere soorten sloten dan die van het geldkistje. Neem een hangslot: in iedere bouwmarkt kun je ze kopen, met de bijzondere eigenschap dat je het sleuteltje niet nodig hebt om het hangslot dicht te doen. Het sleuteltje is alléén nodig voor het opendoen.

Figuur 1.3: Een metafoor voor asymmetrische versleuteling.

[9] Proeft u de fijne ironie?

[10] Probeer dit vooral niet zelf thuis. Dit is werk voor de hogere nerds.

Anders gezegd: voor *versleutelen* heb je alleen het open hangslot nodig; voor *ontsleutelen* heb je het sleuteltje nodig.

Dit kunnen we gebruiken in combinatie met een onveilig communicatiekanaal, als volgt. Als Alice een geheim bericht wil sturen aan Bob, dan moet Bob een hangslot met bijbehorend sleuteltje hebben. Het sleuteltje geeft hij aan niemand af, maar het hangslot (open natuurlijk) kan hij rustig aan iedereen geven (in tegenstelling tot de fysieke wereld waarin je iets zelf niet meer hebt als je het weggeeft, kun je in de digitale wereld overal zoveel gratis kopieën van maken als je maar wilt, dus Bob kan hetzelfde open hangslot aan iedereen weggeven). Want met een open hangslot kun je alleen maar kistjes dichtdoen, niet openmaken. Ook slechte Eva, de afluisteraar, mag Bob's open hangslot hebben. Dus: Bob stuurt zijn open hangslot over een onbeveiligd kanaal naar Alice; Alice versleuteld het bericht ermee en stuurt dat naar Bob, en Bob kan het ontsleutelen. Eva onderschept zowel het open hangslot als het geheimschrift, maar kan er niets mee.

Vanaf nu noemen we het hangslot de *publieke sleutel* (je versleutelt er immers mee), en het sleuteltje de *privé-steutel*. Ze vormen een sleutelpaar, ze horen wel bij elkaar. Anders gezegd: de publieke sleutel bevat wel degelijk informatie over de privé-sleutel, maar die is, als het goed is, er niet uit te halen. Dit soort cryptografie noemen we *asymmetrische cryptografie*: asymmetrisch omdat er voor ontsleutelen een andere sleutel nodig is van voor versleutelen.

De vraag is nu: hoe krijg je zoiets voor elkaar met digitale sleutels? Het antwoord: met leuke wiskunde. Vaak speelt modulorekenen daar een belangrijke rol bij, en dat moeten we dus eerst leren. Voor wie een grondiger maar leesbare behandeling wil in het Nederlands, zie mijn boek[11]. En als je zelf wilt oefenen maar het met de hand rekenen je teveel wordt (en dat wordt het al gauw, want met heel kleine getallen werken geeft vaak te weinig inzicht), kun je de MCR-software gebruiken die gratis voor je klaar staat op mijn website[12].

1.3.2 Snelcursus modulorekenen

Modulorekenen is klokrekenen, maar dan op een klok met m uren, waarbij m ieder getal > 1 mag zijn (maar wel vastligt). Deze m heet de *modulus*.

[11] Benne de Weger, Elementaire Getaltheorie en Asymmetrische Cryptografie, Epsilon Uitgaven, 3e druk, 2012. Schaamteloze zelfpromotie, ik geef het ruiterlijk toe.
[12] `https://www.win.tue.nl/~bdeweger/MCR/`.

Op een 24-uurs-klok nemen we $m = 24$. Het betekent dat we twee gehele getallen die een veelvoud van 24 verschillen als hetzelfde zien. We zeggen dat a en b *congruent modulo* m zijn als $a - b$ een veelvoud is van m. Dan is er dus een geheel getal k (op zich een oninteressant getal) zódanig dat $b = a + k \cdot m$. Een gebruikelijke notatie hiervoor is $b \equiv a \pmod{m}$. Elk geheel getal is congruent $\pmod{m}$ met één getal uit $\{0, 1, 2, \ldots, m-1\}$. Dat getal berekenen kun je doen door *delen met rest*: je deelt door de modulus, en je behoudt alléén de rest; het quotient is niet interessant meer. Dit proces heet *modulaire reductie*.

Figuur 1.4: De klok in de Dom van Florence, met "Italiaanse tijd".

Optellen, aftrekken en vermenigvuldigen gedragen zich netjes modulo m. Daarmee bedoelen we het volgende: als $x \equiv a \pmod{m}$ en $y \equiv b \pmod{m}$, dan geldt altijd ook:

- $x + y \equiv a + b \pmod{m}$,
- $x - y \equiv a - b \pmod{m}$,
- $x \times y \equiv a \times b \pmod{m}$.

Bijvoorbeeld, $3 \times 7 = 21 \equiv 9 \pmod{12}$, en omdat $51 \equiv 3 \pmod{12}$ en $-113 \equiv 7 \pmod{12}$ is het ook zo dat $51 \times (-113) \equiv 9 \pmod{12}$. Inderdaad, $51 = 4 \times 12 + 3$, $-113 = (-10) \times 12 + 7$, dus $51 \times (-113) = -5763 = (-481) \times 12 + 9 \equiv 9 \pmod{12}$.Bovenstaande rekenregels zijn niet moeilijk om te bewijzen.

Machtsverheffen is niets anders dan herhaald vermenigvuldigen, en er is dus ook een regel daarvoor: als $a \equiv b \pmod{m}$ en $n \geq 0$ geheel, dan geldt altijd ook:

- $a^n \equiv b^n \pmod{m}$.

Maar pas op: bij delen kunnen er problemen ontstaan. Bijvoorbeeld: het is wel waar dat $21 \equiv 6 \pmod{15}$, maar het is niet waar dat je nu links en rechts door 3 mag delen: $7 \not\equiv 2 \pmod{15}$.

Er is nog een waarschuwing op zijn plaats: bij machtsverheffen mag je de modulus alléén op de grondtallen toepassen, en niet op de exponenten. Bijvoorbeeld: $10 \equiv 3 \pmod{7}$, en dus wel $10^2 \equiv 3^2 \pmod{7}$ ($10^2 = 100 \equiv 2 \pmod{7}$ en ook $3^2 = 9 \equiv 2 \pmod{7}$), maar $2^{10} = 1024 \equiv 2 \pmod{7}$, terwijl $2^3 = 8 \equiv 1 \pmod{7}$, dus $2^{10} \not\equiv 2^3 \pmod{7}$.

Laat $p \geq 3$ een priemgetal zijn. Dit priemgetal gaat als modulus optreden, m.a.w. we kijken naar de verzameling $\mathbb{F}_p = \{0, 1, 2, \ldots, p-2, p-1\}$ van alle mogelijke getallen modulo p. Binnen deze verzameling kunnen we altijd optellen en aftrekken, en de gebruikelijke regels voor die operaties gelden, als we altijd maar (mod p) terugreduceren naar deze verzameling. We zeggen ook wel: $\mathbb{F}_p$ is een *optelgroep* (*additieve* groep).

We kunnen er ook in vermenigvuldigen, en dus in machtsverheffen. De 0 heeft dan een uitzonderingspositie omdat die bij vermenigvuldigen alles doodslaat. We spreken daarom nu liever over $\mathbb{F}_p^* = \{1, 2, \ldots, p-2, p-1\}$.

Een belangrijke eigenschap, die sterk samenhangt met het priemgetal zijn van p, is de "Stelling van Fermat", dat voor iedere $a \in \mathbb{F}_p^*$ geldt dat $a^{p-1} \equiv 1 \pmod{p}$. Niet lastig te bewijzen, maar we verwijzen daarvoor toch maar naar de literatuur. Een interessant gevolg hiervan is dat we nu probleemloos kunnen gaan delen: voor iedere a is er een b zodat $ab \equiv 1$, namelijk $b \equiv a^{p-2} \pmod{p}$. We schrijven liever niet $1/a \pmod{p}$ omdat we iedere gedachte aan breuken willen vermijden: de hele getallenwereld waarin we werken bestaat immers uit niet meer dan $1, 2, \ldots, p-1$. Hoe dan ook, in $\mathbb{F}_p^*$ bestaat een deling, en dat maakt $\mathbb{F}_p^*$ ook tot een groep t.a.v. de vermenigvuldiging: een *multiplicatieve groep*. Als de modulus geen priemgetal is, gaat die vlieger helaas niet meer op.

We zien nu ook dat we best negatieve exponenten aankunnen: omdat $a^{p-1} \equiv 1 \pmod{p}$, is ook $a^{p-2} \equiv a^{-1} \pmod{p}$, enzovoorts. We zien nu ook wat de regel is voor omgaan met exponenten: bij dezelfde grondtallen en verschillende exponenten geldt: als $m \equiv n \pmod{p-1}$, dán geldt $a^m \equiv a^n \pmod{p}$. Voorbeeld: $2^{10} \equiv 2^4 \pmod{7}$, want $10 \equiv 4 \pmod{7-1}$. Vreemd, maar waar. Gewoon even wennen.

We gaan nog een stap verder: $\mathbb{F}_p^*$ is niet alleen maar een multiplicatieve groep, maar zelfs een *cyclische* groep. Daarmee bedoelen we dat je alle elementen ervan kunt bereiken door herhaald te vermenigvuldigen met één vast getal, een *voortbrenger*. Er is dus altijd een $g \in \mathbb{F}_p^*$ (voor iedere p mogelijk weer een andere), waarvoor geldt dat $1 = g^0, g = g^1, g^2, g^3, \ldots$ precies alle getallen $\pmod{p}$ gaan opleveren. En omdat we met g^{p-1} weer op 1 uitkomen, zijn alle machten $g^0, g^1, g^2, \ldots, g^{p-2}$ dus verschillend $\pmod{p}$. Bijvoorbeeld, met $p = 19$ kunnen we $g = 2$ nemen:

$$\begin{array}{llllllllllll} 2^0 \equiv 1, & 2^1 \equiv 2, & 2^2 \equiv 4, & 2^3 \equiv 8, & 2^4 \equiv 16, & 2^5 \equiv 13, & 2^6 \equiv 7, \\ 2^7 \equiv 14, & 2^8 \equiv 9, & 2^9 \equiv 18, & 2^{10} \equiv 17, & 2^{11} \equiv 15, & 2^{12} \equiv 11, & 2^{13} \equiv 3, \\ 2^{14} \equiv 6, & 2^{15} \equiv 12, & 2^{16} \equiv 5, & 2^{17} \equiv 10, & 2^{18} \equiv 1 & \pmod{19}. \end{array}$$

Niet iedere g doet het, bijvoorbeeld $g = 11$ niet: 11^3 wordt al $1 \pmod{19}$,

dus 11 brengt lang niet alles voort in $\mathbb{F}_{19}^*$. We zeggen wel dat 11 een cyclische ondergroep $\{1, 7, 11\}$ van orde 3 voortbrengt. Het begrip *orde* van $g \pmod p$ is: de kleinste $e > 0$ waarvoor $g^e \equiv 1 \pmod p$ is. Een voortbrenger van heel $\mathbb{F}_p^*$ heeft orde $p-1$. Alleen delers van $p-1$ kunnen optreden als orde.

1.3.3 Discrete logaritmen

Je zag hierboven dat machtsverheffen zo'n verzameling $\mathbb{F}_p^*$ flink door elkaar gooit. Maar aan de andere kant blijft de vermenigvuldigingsstructuur helemaal behouden. Dat maakt het uiterst geschikt voor cryptografie: de structuur maakt het mogelijk om een publieke sleutel en een privé-sleutel bij elkaar te laten horen zodat ontsleutelen precies het versleutelen ongedaan gaat maken, terwijl het chaotische element informatie over de privésleutel weet te verbergen in de publieke sleutel.

In het bijzonder geldt: voor gegeven modulus p en voortbrenger $g \pmod p$ en een gegeven $x \in \{0, 1, 2, \ldots, p-2\}$ is het makkelijk om $g^x \pmod p$ te berekenen, maar andersom, als je één of andere $h \in \mathbb{F}_p^*$ voorgeschoteld krijgt, dan is het niet direct duidelijk hoe de exponent x te vinden waarvoor geldt dat $y \equiv g^x \pmod p$, ook al ken je verder zowel y als g en p. Dit is een erkend moeilijk probleem, het heet het *Discrete Logaritme-Probleem.*

Waarom is machtsverheffen makkelijk? In de praktijk werken we met bijzonder grote getallen, van honderden cijfers. In de eerste plaats zorgt het modulorekenen ervoor dat de getallen altijd binnen de modulus blijven: a^b kan gigantisch groot zijn, maar $a^b \pmod m$ wordt nooit groter dan m. In de tweede plaats doe je machtsverheffen niet door herhaald te vermenigvuldigen, maar slimmer, door de kwadrateer-en-vermenigvuldig-methode. We laten het zien aan de hand van een voorbeeld. Om a^{23} uit te rekenen hoeven we niet 22 maal te vermenigvuldigen, maar veel minder: merk op dat de tweetallige schrijfwijze van 23 gegeven wordt door 10111, dat betekent dat $23 = 2^4+2^2+2^1+2^0$. Dat betekent dat $a^{23} = a^{16} \cdot a^4 \cdot a^2 \cdot a$, en het rijtje $a \to a^2 \to a^4 \to a^8 \to a^{16}$ is makkelijk te berekenen door telkens te kwadrateren. Dus met 4 kwadrateringen en 3 vermenigvuldigingen hebben we a^{23}. Voor een getal x tussen 2^{n-1} en 2^n, dus van n bits, hebben we nooit meer dan $n-1$ kwadrateringen en $n-1$ vermenigvuldigingen nodig om a^x te berekenen. Dus hooguit $2 \log_2 x$ berekeningen in plaats van x.

We laten nu zien hoe je hiermee cryptografie kunt gaan bedrijven. In Hoofdstuk 5 van Greg Alpár komt dit thema van Discrete Logaritmen weer uitgebreid terug.

1.3.4 Diffie-Hellman – een geheime sleutel afspreken

Het oudste publieke-sleutel-systeem is dat van Whitfield Diffie en Martin Hellman uit 1977. Het is niet direct een versleutelingssysteem, maar het zorgt ervoor dat Alice en Bob een geheime symmetrische sleutel kunnen afspreken terwijl al hun communicatie mag worden afgeluisterd. Het is gebaseerd op het moeilijk zijn van het Discrete Logaritme-Probleem, en werkt als volgt.

Systeemparameters Alice en Bob spreken een priemgetal $\mathbf{p}$ af[13] en een voortbrenger $\mathbf{g}$ van de multiplicatieve groep $\mathbb{F}_{\mathbf{p}}^*$.

Sleutelparen Alice maakt een sleutelpaar $\mathbf{x}_A, \mathbf{y}_A$ aan, als volgt: ze kiest een willekeurige $\mathbf{x}_A \in \{2, 3, \ldots, \mathbf{p}-3\}$ als privé-sleutel, en berekent de bijbehorende publieke sleutel als $\mathbf{y}_A \equiv \mathbf{g}^{\mathbf{x}_A} \pmod{\mathbf{p}}$.
Bob doet hetzelfde, maar volledig onafhankelijk van Alice: hij kiest zijn eigen willekeurige $\mathbf{x}_B \in \{2, 3, \ldots, \mathbf{p}-3\}$ als privé-sleutel, en berekent de bijbehorende publieke sleutel als $\mathbf{y}_B \equiv \mathbf{g}^{\mathbf{x}_B} \pmod{\mathbf{p}}$.

Uitwisselen publieke sleutels Alice stuurt haar publieke sleutel $\mathbf{y}_A$ naar Bob, en Bob stuurt zijn publieke sleutel $\mathbf{y}_B$ naar Alice. Dat sturen mag over een onbeveiligd communicatiekanaal; afluisteraar Eva krijgt deze twee publieke sleutels dus ook in bezit.

Berekenen gedeelde geheim Alice berekent nu het getal $\mathbf{s}_A \equiv \mathbf{y}_B^{\mathbf{x}_A} \pmod{\mathbf{p}}$ met Bob's publieke sleutel en haar eigen privé-sleutel.
En Bob berekent evenzo het getal $\mathbf{s}_B \equiv \mathbf{y}_A^{\mathbf{x}_B} \pmod{\mathbf{p}}$ met Alice's publieke sleutel en zijn eigen privé-sleutel.

Hiermee eindigt het protocol. Eva kan de getallen $\mathbf{s}_A, \mathbf{s}_B$ niet berekenen, want zij heeft geen privé-sleutels en kan die ook niet uitrekenen, want dat komt neer op het oplossen van een Discrete Logaritme-Probleem. De crux is nu dat Alice en Bob hetzelfde getal hebben uitgerekend, want $\mathbf{s}_A = \mathbf{s}_B$. Maar dat is pure magie! Nee hoor, het is wiskunde:
$\mathbf{s}_A \equiv \mathbf{y}_B^{\mathbf{x}_A} \equiv (\mathbf{g}^{\mathbf{x}_B})^{\mathbf{x}_A} = \mathbf{g}^{\mathbf{x}_B \mathbf{x}_A} \pmod{\mathbf{p}}$, en ook
$\mathbf{s}_B \equiv \mathbf{y}_A^{\mathbf{x}_B} \equiv (\mathbf{g}^{\mathbf{x}_A})^{\mathbf{x}_B} = \mathbf{g}^{\mathbf{x}_A \mathbf{x}_B} \pmod{\mathbf{p}}$. Ja hoor, hetzelfde.

Een voorbeeldje met speelgoedgetallen[14]. We nemen $\mathbf{p} = 97$ en $\mathbf{g} = 5$. Inderdaad is 5 een voortbrenger (dat is wel wat werk om aan te tonen). Alice kiest $\mathbf{x}_A = 84$ en berekent $\mathbf{y}_A = 5^{84} \equiv 47 \pmod{97}$. Bob kiest

[13] Hier introduceren we een nieuwe kleur: blauw; dat is evenals groen ook publieke informatie, maar deze hoort niet bij een bepaalde persoon, zoals een sleutel, maar is voor alle gebruikers van het systeem dezelfde.
[14] Met de MCR-software kun je een meer realistisch voorbeeld zelf doen.

$\mathbf{x}_B = 23$ en berekent $\mathbf{y}_B = 5^{23} \equiv 82 \pmod{97}$. Ze sturen elkaar hun publieke sleutel toe. Alice berekent dan $\mathbf{s}_A = 82^{84} \equiv 64 \pmod{97}$, en Bob berekent $\mathbf{s}_B = 47^{23} \equiv 64 \pmod{97}$. Alle machtsverheffingen gebeuren met de kwadrateer-en-vermenigvuldig-methode.

Eigenlijk is er meer aan de hand dan alleen maar het Discrete Logaritme-Probleem. Want Eva hoeft misschien niet een Discrete Logaritme-Probleem op te lossen: ze hoeft alleen maar uit $\mathbf{g}^{\mathbf{x}_A}$ en $\mathbf{g}^{\mathbf{x}_B}$ te kunnen uitvinden wat $\mathbf{g}^{\mathbf{x}_A\mathbf{x}_B}$ is. Het is niet bewezen dat ze dat alleen maar kan als ze het Discrete Logaritme-Probleem kan oplossen, maar waarschijnlijk is dat wel.

Als Alice en Bob het Diffie-Hellman-protocol hebben afgespeeld, dan kunnen ze het gedeelde geheime getal vervolgens gebruiken, bijvoorbeeld als sleutel voor een symmetrisch systeem. Zo is het sleuteluitwisselingsprobleem van de symmetrische cryptografie elegant opgelost met behulp van asymmetrische cryptografie.

1.3.5 ElGamal – versleutelen

Het idee van ElGamal is een heel eenvoudig mechanisme om het Diffie-Hellman-protocol om te zetten in een asymmetrisch versleutelingssysteem. Alice en Bob spreken eerst de systeemparameters $\mathbf{p}$ en $\mathbf{g}$ af (die mogen ook rustig jarenlang gebruikt worden, vaak zitten ze gewoon in software ingebakken). Stel Alice wil de klare tekst $\mathbf{m}$ versleuteld naar Bob versturen. We gaan ervan uit dat deze klare tekst gecodeerd is in een getal dat kleiner dan $\mathbf{p}$ is. Bob stuurt zijn publieke DH-sleutel $\mathbf{y}_B$ naar Alice (Bob mag zijn sleutelpaar best vaker, zelfs jarenlang, gebruiken). Nu maakt Alice een eenmalig DH-sleutelpaar $\mathbf{x}_1, \mathbf{y}_1$ aan, voor ieder bericht weer een verse, zodat nooit twee berichten met dezelfde sleutel versleuteld worden (dat zien we namelijk al als een zwakheid; daarmee zou al informatie uitlekken, al is het alleen maar dat er kennelijk tweemaal hetzelfde bericht versleuteld werd). Alice berekent het eenmalige gedeelde geheim $\mathbf{s}_1 \equiv \mathbf{y}_B^{\mathbf{x}_1} \pmod{\mathbf{p}}$, en de "symmetrische" versleuteling hoeft nu op zich niet heel sterk te zijn: simpelweg vermenigvuldigen (mod $\mathbf{p}$) volstaat al: Alice berekent $\mathbf{c} \equiv \mathbf{m} \cdot \mathbf{s}_1 \pmod{\mathbf{p}}$. Als geheimschrift stuurt Alice nu $(\mathbf{y}_1, \mathbf{c})$ naar Bob: haar eenmalige publieke sleutel en het geheimschrift in engere zin. Bob kan nu met zijn privé-sleutel ook het eenmalige gedeelde geheim berekenen als $\mathbf{s}_1 \equiv \mathbf{y}_1^{\mathbf{x}_B} \pmod{\mathbf{p}}$, en vindt dan de klare tekst terug met een deling: $\mathbf{m} \equiv \mathbf{c}/\mathbf{s}_1 \pmod{\mathbf{p}}$.

1.3.6 RSA – versleutelen

Er is een ander asymmetrisch systeem, dat wel direct versleutelt en ontsleutelt, en dus geen sleuteluitwisseling doet. Dit werd in 1978, dus kort na Diffie en Hellman, gepubliceerd door Ron Rivest, Adi Shamir en Len Adleman. Het is ook gebaseerd op modulorekenen en machtsverheffen, maar niet op het Discrete Logaritme-probleem: de veiligheid van RSA is gebaseerd op het moeilijk zijn van het factorisatieprobleem. In de praktijk is RSA in de afgelopen decennia het meest gebruikte systeem geweest. We leggen het kort uit.

Alice wil weer een klare tekst $\mathbf{m}$ versleuteld aan Bob sturen. We gaan ervan uit dat deze klare tekst weer gecodeerd is in een klein genoeg getal.

Sleutelpaar Bob maakt twee priemgetallen $\mathbf{p}, \mathbf{q}$, van dezelfde bitlengte. Hij vermenigvuldigt ze met elkaar tot $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$. Aan de kleurcode zie je al dat Bob de priemgetallen geheim houdt, maar hun product mag publiek zijn. Het product bevat informatie over de geheime priemgetallen, maar het is ondoenlijk die informatie eruit te halen. De $\mathbf{n}$ moet groter zijn dan de klare tekst $\mathbf{m}$.
Bob berekent ook $\phi = (\mathbf{p} - 1)(\mathbf{q} - 1)$. Ook deze moet Bob geheim houden: als ϕ zou lekken dan zullen uit kennis van uitsluitend $\mathbf{n}$ en ϕ ook $\mathbf{p}$ en $\mathbf{q}$ lekken.
Bob neemt een willekeurig getal $\mathbf{e}$, tussen 3 en $\phi - 2$. Een technische eis is dat $\mathbf{e}$ geen delers gemeen mag hebben met ϕ. Dit getal $\mathbf{e}$ is publiek en hoeft ook niet per se groot te zijn; in de praktijk wordt heel vaak $\mathbf{e} = 65537$ genomen.
Bob berekent nu $\mathbf{d}$ zodanig dat $\mathbf{e} \cdot \mathbf{d} \equiv 1 \pmod{\phi}$. Dat is in essentie een deling. We geven geen details over waarom en hoe deze deling uitgevoerd kan worden; Euclides wist in de 3e eeuw v.Chr. al hoe dit efficiënt kan. Dit getal $\mathbf{d}$ is weer strikt geheim.
De publieke sleutel van Bob bestaat nu uit $\mathbf{n}$ en $\mathbf{e}$. De privé-sleutel van Bob is in feite $\mathbf{d}$, maar omdat voor ontsleutelen ook de publieke $\mathbf{n}$ nodig is, wordt meestal gezegd dat de privé-sleutel bestaat uit $\mathbf{d}$ en $\mathbf{n}$. De getallen $\mathbf{p}, \mathbf{q}, \phi$ zijn niet meer nodig en Bob kan ze weggooien.

Versleutelen Alice gebruikt Bob's publieke sleutel $\mathbf{n}, \mathbf{e}$, ze voert een machtsverheffing uit om het geheimschrift te berekenen: $\mathbf{c} \equiv \mathbf{m}^{\mathbf{e}} \pmod{\mathbf{n}}$.

Ontsleutelen Bob ontvangt het geheimschrift, en voert ook een machtsverheffing uit, maar dan met zijn privé-sleutel: $\mathbf{m} = \mathbf{c}^{\mathbf{d}} \pmod{\mathbf{n}}$.

Dat is alles. RSA heeft geen systeemparameters, zoals Diffie-Hellman.

Waarom werkt dit? Waarom vindt Bob altijd de klare tekst terug? Eigenlijk omdat rekenen modulo een samengesteld getal als $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$ neerkomt op de combinatie van rekenen modulo de priemfactoren. Als twee getallen modulo $\mathbf{n}$ hetzelfde zijn, dan zijn ze dat ook modulo iedere deler van $\mathbf{n}$. Andersom is het ook waar: als twee getallen modulo $\mathbf{p}$ en modulo $\mathbf{q}$ hetzelfde zijn, dan zijn ze modulo $\mathbf{n}$ hetzelfde. Laten we dan eens kijken wat er modulo $\mathbf{p}$ gebeurt:

- Alice berekent $\mathbf{c} \equiv \mathbf{m}^{\mathbf{e}} \pmod{\mathbf{n}}$, dat betekent dan ook dat $\mathbf{c} \equiv \mathbf{m}^{\mathbf{e}} \pmod{\mathbf{p}}$ (hoewel Alice helemaal niet weet wat $\mathbf{p}$ is),
- Bob berekent $\mathbf{c}^{\mathbf{d}} \pmod{\mathbf{n}}$, maar we kijken daar modulo $\mathbf{p}$ naar: $\mathbf{c}^{\mathbf{d}} \equiv (\mathbf{m}^{\mathbf{e}})^{\mathbf{d}} \equiv \mathbf{m}^{\mathbf{ed}} \pmod{\mathbf{p}}$, en nu gebruiken we het verband tussen $\mathbf{e}$ en $\mathbf{d}$, namelijk dat $\mathbf{e} \cdot \mathbf{d} \equiv 1 \pmod{\phi}$ is; die ϕ is een veelvoud van $\mathbf{p} - 1$, en dus moet wel $\mathbf{e} \cdot \mathbf{d} \equiv 1 \pmod{\mathbf{p} - 1}$ kloppen. Nu komt de Stelling van Fermat ons te hulp, want die zei dat je bij machtsverheffen modulo een priemgetal $\mathbf{p}$ de exponenten modulo $\mathbf{p} - 1$ kunt nemen. Dus volgt $\mathbf{m}^{\mathbf{ed}} \equiv \mathbf{m}^1 \pmod{\mathbf{p}}$. En wat Bob berekend heeft modulo $\mathbf{p}$ is dus precies de klare tekst.

Eenzelfde redenering geldt uiteraard voor het priemgetal $\mathbf{q}$. Bob heeft precies $\mathbf{m}$ modulo $\mathbf{p}$ en modulo $\mathbf{q}$ gevonden, dus ook modulo $\mathbf{n}$.

Een speelgoedvoorbeeld[15]. Bob kiest als priemgetallen $\mathbf{p} = 41, \mathbf{q} = 47$, en berekent $\mathbf{n} = 41 \times 47 = 1927$ en $\phi = 46 \times 40 = 1840$. Hij kiest $\mathbf{e} = 1223$ en berekent $\mathbf{d} = 1223^{-1} \equiv 167 \pmod{\phi}$. Bob's publieke sleutel is $1927, 1223$ en zijn privé-sleutel is $1927, 167$. Alice wil het bericht $\mathbf{m} = 1234$ versleutelen zó dat alleen Bob het kan lezen. Zij berekent dan het geheimschrift als $1234^{1223} \equiv 761 \pmod{1927}$ en stuurt dat naar Bob. Bob vindt de klare tekst weer terug als $761^{167} \equiv 1234 \pmod{1927}$.

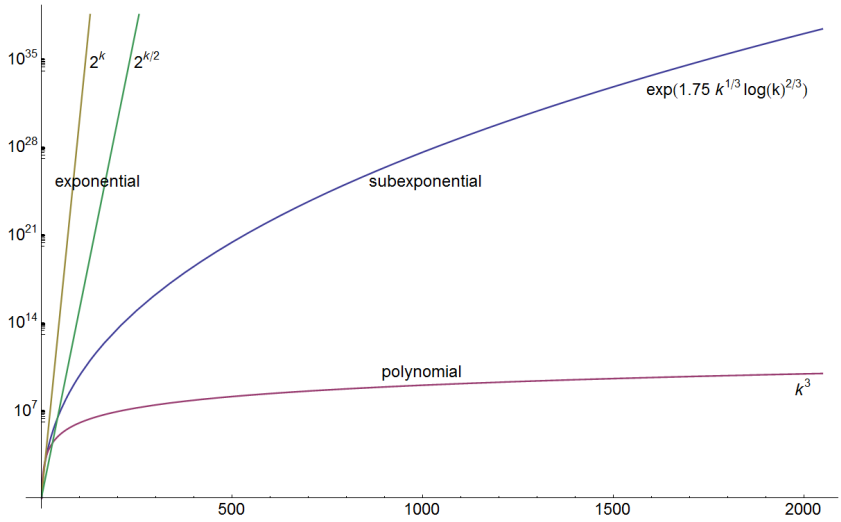
We moeten iets zeggen over snelheid. Dat is namelijk best een probleem. Een modulaire machtsverheffing, zelfs met een efficiënte kwadrateer-en-vermenigvuldig-methode, kost veel rekenkracht in vergelijking met de operaties van AES. Het kan zomaar een milliseconde duren om één machtsverheffing te doen. Dat lijkt weinig, maar voor een wat langere tekst die je moet opdelen in veel stukjes die elk apart versleuteld moeten worden, wordt dat een dingetje. En voor een website die duizenden pagina's per seconde moet kunnen wegsturen, wordt het echt een groot probleem. We komen er in Hoofdstuk 4 op terug.

[15] Met de MCR-software kun je een meer realistisch voorbeeld zelf doen.

1.3.7 Kraken, en sleutellengtes

Diffie-Hellman baseert de veiligheid op het Discrete Logaritme-Probleem in $\mathbb{F}_p^*$, en RSA op het Factorisatieprobleem van de natuurlijke getallen. Natuurlijk kun je brute kracht gaan proberen: probeer bij Diffie-Hellman alle mogelijke privé-sleutels $\mathbf{x}$, en probeer bij RSA alle mogelijke priemgetallen tot aan $\sqrt{\mathbf{n}}$ uit[16]. Als sleutellengte kun je het aantal bits van de modulus nemen ($\mathbf{p}$ bij Diffie-Hellman, en $\mathbf{n}$ bij RSA, maar bij RSA ligt het dus meer voor de hand om de halve bitlengte van $\mathbf{n}$ te nemen).

Maar deze problemen hebben een heleboel wiskundige structuur op de achtergrond: allerlei fraaie getaltheorie. Kunnen we dat niet uitbuiten? Uiteraard is daar ongelofelijk veel onderzoek naar gedaan, en met het nodige succes. Er zijn inderdaad oplossingen van beide moeilijke problemen die aanzienlijk sneller zijn dan je met brute kracht kunt bereiken. Die methoden gaan het niveau van deze cursus echter flink te boven. In beide gevallen werkt de techniek van de zogeheten Getallenlichamenzeef (Number Field Sieve) goed. Voor bitlengte k is de complexiteit van die methode niet meer exponentieel (2^k of $2^{k/2}$) maar *subexponentieel*. Voor het factorisatieprobleem geldt de formule $\exp\left(1.75k^{1/3}(\log k)^{2/3}\right)$, waarbij exp staat voor e-macht. Dat is veel beter dan exponentieel, maar nog altijd veel slechter dan polynomiaal (polynomiaal is: een vaste macht van k, bijv. k^3;



Figuur 1.5: Complexiteit: exponentieel, subexponentieel, polynomiaal. N.B.: logaritmische schaal!

[16] Waarom is dat genoeg?

polynomiaal wordt over het algemeen gezien als “volledig onveilig”). Zie Figuur 1.5.

Voor het Discrete Logaritme-probleem in $\mathbb{F}_p^*$ geldt iets soortgelijks.

Dit betekent overigens niet dat Diffie-Hellman en RSA nu onveilig zijn. Het betekent wel dat je je sleutels voor dit soort systemen veel groter moet gaan kiezen dan 128 bits. Over het algemeen wordt nu aangeraden om bij dit soort systemen de grootte van de modulus minstens 2048 bits te kiezen. Het record voor factorisatie dateert uit 2010, dat ging om een RSA-modulus **n** van 768 bits. Hoewel dat record nog steeds staat, wordt er nu van uitgegaan dat het veelgebruikte 1024 bits niet meer veilig is: misschien nog wel voor je buurjongen, maar niet meer voor een formidabele maar realistische tegenstander als de geheime dienst van een vijandige (of officieel bevriende) natie.

Ook zijn er voor allerlei speciale gevallen methoden die dan soms veel beter werken. Je zou kunnen zeggen dat er een aantal categorieën zwakke sleutels zijn, die je moet vermijden. Voor de oude rotten onder u: in 2010 sprak ik daarover op de Vakantiecursus[17].

Deze subexponentiële complexiteit geldt overigens weer niet voor alle asymmetrische systemen. je moet dus altijd goed uitkijken bij het kiezen van je sleutellengte.

[17] Benne de Weger, Hoe je het cryptosysteem RSA soms kunt kraken, in: Wiskunde: de uitdaging – Vakantiecursus 2010, CWI Syllabus 60, Amsterdam, 2010.