2 De cryptografie achter Bitcoin
Benne de Weger

2.1 Digitale handtekeningen

2.1.1 Het principe

Hoofdstuk 1 ging eigenlijk alleen maar over versleutelen: de cryptografische techniek die voor vertrouwelijkheid zorgt. Maar dat is niet het enige waar cryptografie iets kan betekenen voor een goede informatiebeveiliging. We willen ook kunnen authenticeren: er zeker van kunnen zijn dat die website waar je je bankzaken op gaat doen, echt van de bank is, en niet door de maffia is nagemaakt. Misschien is het niet leuk dat Eva weet dat jij 100 Euro aan je broer overmaakt, maar is dat nog tot daar aan toe; het wordt erger als Eva zich tegenover de bank kan voordoen als jou, en zonder jouw medeweten overboekingen van jouw rekening kan gaan doen. Eva is nu dus niet alleen maar een passieve afluisteraar, ze is nu aktief aan het ingrijpen geslagen op de communicatie tussen Alice en Bob.

Het RSA-systeem bijvoorbeeld is in feite ontworpen als een techniek voor zowel versleuteling als digitale handtekeningen. Sterker nog, met alleen symmetrische cryptografie is het wel erg moeilijk om een echt betrouwbare digitale handtekening te krijgen: de sleutel waarmee de handtekening gezet is moet niet in handen van iemand anders komen, omdat die dan ook daarmee handtekeningen kan gaan zetten.

Het zetten van een handtekening moet alleen mogelijk zijn door één persoon. Dat lijkt dus bij uitstek iets wat je met een privé-sleutel wilt doen. Nagaan dat een handtekening echt is zou voor meer mensen mogelijk moeten zijn. Kan dat dan met een publieke sleutel? We leggen het uit aan de hand van RSA.

2.1.2 RSA – digitale handtekeningen

Het aardige van het RSA-systeem is dat er, hoewel behorend bij de asymmetrische cryptografie, toch een symmetrie in zit: de relatie tussen de publieke en privé-exponenten $\mathbf{e}$ en $\mathbf{d}$ is symmetrisch: $\mathbf{e} \cdot \mathbf{d} \equiv 1 \pmod{\phi}$, en ook de operaties die je ermee uitvoert zijn, mutatis mutandis, dezelfde, namelijk een machtsverheffing: $\mathbf{c} \equiv \mathbf{m}^{\mathbf{e}} \pmod{\mathbf{n}}$ en $\mathbf{m} \equiv \mathbf{c}^{\mathbf{d}} \pmod{\mathbf{n}}$.

In de praktijk gebeurt het volgende on RSA voor digitale handtekeningen te gaan gebruiken. Stel Alice wil een bericht $\mathbf{m}$ naar Bob sturen, dat niet vertrouwelijk is (het is groen), maar waarvan Bob wel zeker wil weten dat het van Alice afkomt. Nu is het Alice die een RSA-sleutelpaar heeft, bestaande uit een modulus $\mathbf{n}$, een publieke exponent $\mathbf{e}$ en een privé-exponent $\mathbf{d}$, net als in Hoofdstuk 1. Alice wil het bericht nu met haar privé-sleutel gaan behandelen, maar zoals uitgelegd in paragraaf 1.3.6 is dat onhandig, want langzaam.

Onder andere voor dit doel zijn *hashfuncties* uitgevonden. We komen er verderop uitgebreid op terug, en volstaan nu met te zeggen dat je van een bericht $\mathbf{m}$ een soort van vingerafdruk kunt maken: een korte code die het bericht uniek identificeert. Dat gaat met een hashfunctie h, dus Alice berekent eerst $\mathbf{h} = h(\mathbf{m})$. Zo'n hashfunctie heeft geen sleutel nodig. De hashwaarde $\mathbf{h}$ heeft een vaste lengte, tegenwoordig doorgaans 256 bits. Als je in je bericht ook maar één bit wijzigt, zal de hashwaarde er totaal anders gaan uitzien. Een hashfunctieis niet omkeerbaar: uit alléén $\mathbf{h}$ kun je nooit het bijbehorende bericht $\mathbf{m}$ terugvinden. N.B.: de hash blijft groen: er is niets geheims aan.

Alice past dan RSA met de privé-sleutel toe[1] op die hash: ze berekent $\mathbf{s} \equiv \mathbf{h}^{\mathbf{d}} \pmod{\mathbf{n}}$. En dit getal $\mathbf{s}$ is dan de handtekening van Alice bij het bericht $\mathbf{m}$. Alice stuurt nu niet alleen het bericht $\mathbf{m}$ naar Bob, maar ook haar publieke sleutel $\mathbf{n}, \mathbf{e}$ en de handtekening $\mathbf{s}$.

Bob kan de handtekening nu verifiëren: hij berekent eerst ook zelf $\mathbf{h} = h(\mathbf{m})$, en past dan RSA met Alice's publieke sleutel toe op de handtekening: Bob berekent $\mathbf{s}^{\mathbf{e}} \pmod{\mathbf{n}}$. Dat is, als alles goed is, $\left(\mathbf{h}^{\mathbf{d}}\right)^{\mathbf{e}} \equiv \mathbf{h}^{\mathbf{de}} \equiv \mathbf{h} \pmod{\mathbf{n}}$. En als dat inderdaad precies het getal $\mathbf{h}$ oplevert dat Bob uit het bericht berekend heeft, dan weet Bob dat het goed is: de enige die bij dít bericht $\mathbf{m}$ déze handtekening $\mathbf{s}$ heeft kunnen maken, is de bezitter van de privé-exponent die hoort bij de publieke sleutel $\mathbf{n}, \mathbf{e}$, dus Alice.

[1] Op het Internet zie je veel dat mensen het hebben over "versleutelen met de privé-sleutel". Die mensen begrijpen het niet. Ik verbied mijn studenten om dit soort verwarrende taal te bezigen.

Eva heeft $\mathbf{d}$ niet, en had $\mathbf{s}$ nooit kunnen maken. En bovendien weet Bob dat Eva niet onderweg het bericht $\mathbf{m}$ kan hebben aangepast, want dan zou Bob een andere waarde voor $\mathbf{h}$ hebben gevonden en was zijn verificatieberekening anders uitgekomen. Dus Bob krijgt authenticatie van Alice als oorsprong van het bericht, én hij weet dat de inhoud van het bericht onderweg niet gemanipuleerd kan zijn: gratis integriteit bij authenticatie.

2.1.3 DSA – digitale handtekeningen

Er bestaan ook handtekeningsystemen gebaseerd op het Discrete Logaritmeprobleem. We behandelen DSA – het Digital Signature Algorithm, omdat een variant daarop in Bitcoin voorkomt. Op dezelfde manier als bij RSA wordt een hash $\mathbf{h} = h(\mathbf{m})$ van het te ondertekenen bericht gebruikt.

Parameterkeuze Kies een priemgetal $\mathbf{q}$, groot genoeg zodat de hash $\mathbf{h}$ er altijd onder valt. Kies een priemgetal $\mathbf{p}$, groot genoeg om het Discrete Logaritme-Probleem in $\mathbb{F}_{\mathbf{p}}^*$ echt moeilijk te laten zijn, en ook zó dat $\mathbf{p} \equiv 1 \pmod{\mathbf{q}}$. Deze laatste eis zorgt ervoor dat er elementen in $\mathbb{F}_{\mathbf{p}}^*$ zijn van orde $\mathbf{q}$. Kies zo'n element $\mathbf{g}$, dan werken we in de cyclische ondergroep $\langle \mathbf{g} \rangle$ van $\mathbb{F}_{\mathbf{p}}^*$ met $\mathbf{q}$ elementen.

Sleutelpaar Alice kiest een willekeurige $\mathbf{x}$ met $1 < \mathbf{x} < \mathbf{q} - 1$ als privésleutel. Ze berekent haar publieke sleutel als $\mathbf{y} \equiv \mathbf{g}^{\mathbf{x}} \pmod{\mathbf{p}}$. Dat is dus bijna net zo als bij Diffie-Hellman, alleen wordt de publieke sleutel nu een element uit een veel kleinere ondergroep $\langle \mathbf{g} \rangle$ van $\mathbb{F}_{\mathbf{p}}^*$. Toch maakt dat het Discrete Logaritme-Probleem niet makkelijker. Dit sleutelpaar van Alice kan ze voor veel ondertekeningen gaan gebruiken, ook jarenlang.

Ondertekenen Alice kiest een random éénmalig getal $\mathbf{k}$ met $1 < \mathbf{k} < \mathbf{q} - 1$. Ze berekent $\mathbf{r} \equiv \left(\mathbf{g}^{\mathbf{k}} \pmod{\mathbf{p}}\right) \pmod{\mathbf{q}}$ en $\mathbf{s} \equiv (\mathbf{h} + \mathbf{xr})\mathbf{k}^{-1} \pmod{\mathbf{q}}$. De handtekening is dan $(\mathbf{r}, \mathbf{s})$.

Verificatie van de handtekening Bob herberekent zelf de hash $\mathbf{h}$, en berekent dan $\mathbf{u}_1 \equiv \mathbf{hs}^{-1} \pmod{\mathbf{q}}$ en $\mathbf{u}_2 \equiv \mathbf{rs}^{-1} \pmod{\mathbf{q}}$, en daarmee $\mathbf{v} \equiv (\mathbf{g}^{\mathbf{u}_1}\mathbf{y}^{\mathbf{u}_2} \pmod{\mathbf{p}}) \pmod{\mathbf{q}}$. De verificatie is dan dat voldaan moet zijn aan de vergelijking $\mathbf{v} = \mathbf{r}$.

De $\mathbf{k}$ is een soort eenmalige privé-sleutel die garandeert dat de handtekening vers is. De $\mathbf{r}$ is de bijbehorende publieke sleutel, zodat Bob kan nagaan dat die handtekening vers was. De handtekening verbindt de privé-sleutel $\mathbf{x}$ van Alice aan het bericht via de hash $\mathbf{h}$. Merk op dat $\mathbf{g}^{\mathbf{u}_1}\mathbf{y}^{\mathbf{u}_2} \equiv \mathbf{g}^{\mathbf{u}_1 + \mathbf{u}_2\mathbf{x}} \pmod{\mathbf{p}}$, en $\mathbf{u}_1 + \mathbf{u}_2\mathbf{x} \equiv (\mathbf{h} + \mathbf{xr})\mathbf{s}^{-1} \equiv \mathbf{k} \pmod{\mathbf{q}}$, en daaruit volgt in-

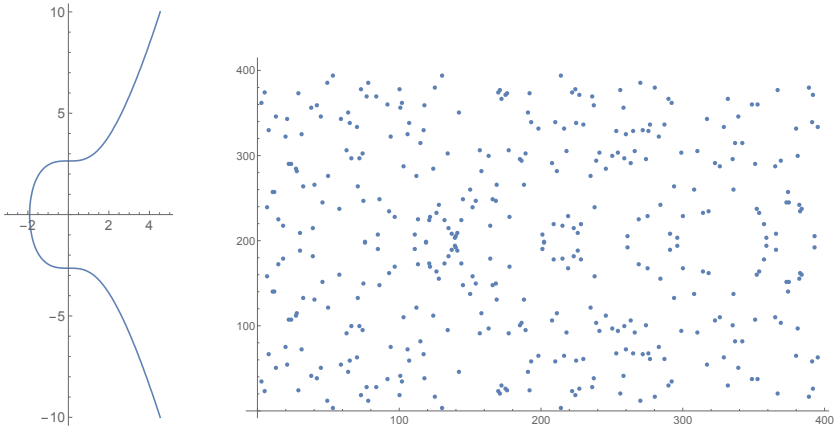
derdaad dat $\mathbf{r} = \mathbf{v}$.

Een voordeel van dit systeem is dat de handtekening nu klein is: tweemaal de grootte van de kleine priem $\mathbf{q}$ (meestal $2 \times 256 = 512$ bits) in plaats van de grootte van de grote priem $\mathbf{p}$ (vaak 2048 bits). Een tweede voordeel is (zo vond de Amerikaanse overheid) dat dit handtekening-systeem niet net zoals RSA kan worden "mis"bruikt voor versleuteling[2].

2.2 Elliptische krommen

2.2.1 Rekenen op de Bitcoin-kromme

Bitcoin maakt gebruik van een variant op het Discrete Logaritme-Probleem. Daarbij wordt er niet in een groep $\mathbb{F}_p^*$ gewerkt, maar in een wat ingewikkelder groep: een zogenaamde Elliptische Kromme. De reden is dat er in dat soort groepen geen subexponentieel algoritme bekend is om het Discrete Logaritme-Probleem op te lossen, alleen maar exponentiële methoden, en dan kunnen je parameters en sleutels veel kleiner blijven, en wordt het rekenwerk ook sneller.



Figuur 2.1: De Bitcoin-kromme $y^2 = x^3 + 7$ over $\mathbb{R}$ (links) en over $\mathbb{F}_p$ met $p = 397$ (rechts).

We gaan eerst maar eens kijken wat dat is, zo'n Elliptische Kromme. Het is geen ellips, maar heel iets anders. Voor het gemak behandelen we de theorie aan de hand van wat in Bitcoin wordt gebruikt: de Bitcoin-kromme.

[2]Versleutelen wordt namelijk door sommige overheden als gevaarlijk gezien. Handtekeningen zijn volstrekt onschuldig.

De Bitcoin-kromme wordt gegeven door de vergelijking $\boxed{y^2 = x^3 + 7}$. Je kunt ook zeggen: ze bestaat uit de grafieken van de functies $y = \sqrt{x^3 + 7}$ en $y = -\sqrt{x^3 + 7}$ aan elkaar geplakt. Als je in eerste instantie x, y in de reële getallen neemt, krijg je het plaatje van Figuur 2.1 links. Maar dit is misleidend om twee redenen: de echte Bitcoin-kromme is niet gedefinieerd over de reële getallen maar over de gehele getallen modulo een zeker priemgetal p, dus over $\mathbb{F}_p$, waarbij de coördinaten x, y dus uit $\{0, 1, 2, \ldots, p-1\}$ komen en modulo p genomen moeten worden bij het rekenen. En in de tweede plaats omdat er nog één extra punt op de kromme ligt dat niet in het plaatje te zien is: het zogeheten *punt op oneindig*, dat we hier aangeven[3] met ∞. Merk op dat dit punt op oneindig géén twee coördinaten heeft. Je kunt het je indenken als een punt dat in verticale richting oneindig ver weg ligt (naar boven of beneden maakt niet uit, dan kom je in hetzelfde punt ∞ uit).

Een voorbeeld: met $p = 11$ laat een klein beetje rekenwerk zien dat je precies de volgende punten krijgt op de Bitcoin-kromme $y^2 = x^3 + 7$:

$$\infty, (2, 2), (2, 9), (3, 1), (3, 10), (4, 4), (4, 7), (5, 0), (6, 5), (6, 6), (7, 3), (7, 8).$$

Voor elke p zijn er maar eindig veel punten. De grafiek ziet er dan ook heel anders uit dan in het geval van reële coördinaten. In Figuur 2.1 rechts geven we de grafiek van de Bitcoin-kromme met $p = 397$. Deze ziet er lekker chaotisch uit, en da's maar goed ook: daarom is dit soort krommen zo prettig voor cryptografie. Maar merk wel op dat de kromme de symmetrie in een horizontale as (hier op hoogte $\frac{1}{2}p$) wel behouden heeft.

Nu gaan we bekijken hoe je op zo'n kromme kunt rekenen met punten. Om historische redenen wordt de rekenoperatie optellen genoemd, met andere woorden, we gaan van de elliptische kromme een *additieve* groep maken[4].

Er is een *optelwet* die twee punten P en Q op de kromme neemt, en er een derde punt van maakt, dat ook altijd weer op de kromme ligt, en dat we $P + Q$ gaan noemen. Het idee is van meetkundige oorsprong, en we leggen het uit voor de kromme over de reële getallen, omdat je je daar iets bij kunt voorstellen. De bijbehorende formules blijken vervolgens ook heel goed toe te passen te zijn voor de kromme over $\mathbb{F}_p$, ook al ben je daar je meetkundige intuïtie kwijt.

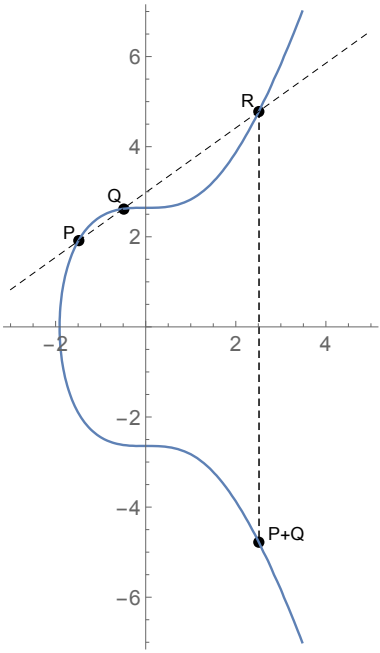
De optelwet is als volgt: voor de twee punten P, Q is er een rechte lijn die door beide punten gaat. Deze rechte snijdt de kromme altijd in een derde

[3] In de literatuur wordt het vaak met $\mathcal{O}$ aangegeven.

[4] Maar dat is alleen maar terminologie: je zou net zo goed multiplicatieve terminologie kunnen gebruiken.

punt, omdat een derdegraads vergelijking nu eenmaal altijd 3 oplossingen heeft. Dat derde punt noemen we even R. Nu spiegelen we R in de x-as, en het punt dat we dan vinden, dát is nu $P+Q$. Zie Figuur 2.2.

Er zijn twee speciale gevallen. Het kan gebeuren dat de lijn door P en Q perfect verticaal is. Dan vind je het punt R niet meer op de grafiek, en dan moet je constateren dat $P+Q=\infty$. We zien het speciale punt ∞ nu als het nul-element van de optelwet, met andere woorden, in dit geval is $Q=-P$. De rechte lijn door een punt P en ∞ zien we als de verticale lijn door P, die snijdt de kromme in een derde punt dat natuurlijk $-P$ is, en dus is $P=-(-P)$ de uitkomst van de optelling $P+\infty$. De ∞ gedraagt zich inderdaad als nul.



Figuur 2.2: Optellen op een elliptische kromme.

Het tweede speciale geval is: wat als je P bij zichzelf wilt optellen, m.a.w. als je P wilt verdubbelen? In dat geval ligt het voor de hand om als rechte lijn de *raaklijn* aan de kromme door P te nemen, en de constructie daarmee uit te voeren. Dit gaat prima werken.

Het is niet moeilijk om de formules uit te werken. We schrijven $P=(x_\mathrm{P},y_\mathrm{P}), Q=(x_\mathrm{Q},y_\mathrm{Q}), R=(x_\mathrm{R},y_\mathrm{R})$ voor de drie snijpunten van de rechte lijn en de kromme, en dan is $P+Q=-R=(x_\mathrm{R},-y_\mathrm{R})$. De richtingscoëfficiënt van de lijn (mits $x_\mathrm{P}\neq x_\mathrm{Q}$) is $\lambda=\dfrac{y_\mathrm{Q}-y_\mathrm{P}}{x_\mathrm{Q}-x_\mathrm{P}}$, en als $P=Q$, dan differentieer je om de juiste λ te vinden. De lijn heeft nu als vergelijking $y-y_\mathrm{P}=\lambda(x-x_\mathrm{P})$.

Dan neem je de vergelijking van de lijn en die van de kromme, en samen geven die een derdegraads vergelijking voor x, waarvan je weet dat de oplossingen $x_\mathrm{P},x_\mathrm{Q},x_\mathrm{R}$ zijn. Met wat algebraïsche manipulatie vind je dan de coördinaten van $P+Q$ uitgedrukt in de coördinaten van P en Q. Hier zijn de formules, links voor optellen: $P+Q=(x_\mathrm{R},-y_\mathrm{R})$, mits $x_\mathrm{Q}\neq x_\mathrm{P}$, en rechts voor verdubbelen: $2P=(x_\mathrm{R},-y_\mathrm{R})$, mits $y_\mathrm{P}\neq 0$:

$$\begin{aligned} \lambda &= \frac{y_\mathrm{Q} - y_\mathrm{P}}{x_\mathrm{Q} - x_\mathrm{P}}, \\ x_\mathrm{R} &= \lambda^2 - x_\mathrm{P} - x_\mathrm{Q}, \\ -y_\mathrm{R} &= -y_\mathrm{P} + \lambda(x_\mathrm{P} - x_\mathrm{S}). \end{aligned}$$

$$\begin{aligned} \lambda &= \frac{3x_\mathrm{P}^2}{2y_\mathrm{P}}, \\ x_\mathrm{R} &= \lambda^2 - 2x_\mathrm{P}, \\ -y_\mathrm{R} &= -y_\mathrm{P} + \lambda(x_\mathrm{P} - x_\mathrm{S}). \end{aligned}$$

Vanwaar heeft de optelwet deze vreemde vorm? Omdat ze de belangrijkste eigenschappen van een optelwet hebben, namelijk commutativiteit (je mag de volgorde verwisselen: $P+Q = Q+P$, dat volgt direct uit de definitie) en associativiteit (je mag de haakjes anders zetten: $(P+Q)+T = P+(Q+T)$, dit is waar, maar bepaald niet eenvoudig te bewijzen). Dit maakt de elliptische kromme, inclusief het punt ∞, tot een echte *additieve groep*.

De gegeven formules gelden in eerste instantie voor reële getallen. Maar je kunt precies dezelfde formules ook hanteren over de gehele getallen (mod p). En dat doen we dan maar voor de cryptografisch interessante krommen.

Voorbeelden met de Bitcoin-kromme met $p = 11$: neem $P = (4, 4)$, dan berekenen we $2P$: $\lambda = 3 \cdot 4^2 \cdot (2 \cdot 4)^{-1} \equiv 6 \pmod{11}$, dus $x_\mathrm{R} \equiv 6^2 - 2 \cdot 4 \equiv 6 \pmod{11}$ en $-y_\mathrm{R} \equiv -4 + 6 \cdot (4 - 6) \equiv 6 \pmod{11}$. We vinden $2P = 2(4, 4) = (6, 6)$, en dat ligt inderdaad op de kromme.

We gaan verder met $3P$ als $2P + P = (6, 6) + (4, 4)$: dan berekenen we $\lambda = (4-6) \cdot (4-6)^{-1} \equiv 1 \pmod{11}$, dus $x_\mathrm{R} \equiv 1^2 - 6 - 4 \equiv 2 \pmod{11}$ en $-y_\mathrm{R} \equiv -6+1 \cdot (6-2) \equiv 9 \pmod{11}$. We vinden $3P = (6, 6)+(4, 4) = (2, 9)$, en dat ligt inderdaad weer op de kromme.

Als we zo doorgaan krijgen we de volgende tabel.

n	nP
1	$(4, 4)$
2	$(6, 6)$
3	$(2, 9)$

n	nP
4	$(3, 10)$
5	$(7, 3)$
6	$(5, 0)$

n	nP
7	$(7, 8)$
8	$(3, 1)$
9	$(2, 2)$

n	nP
10	$(6, 5)$
11	$(4, 7)$
12	∞

Dus alle punten op deze kromme zijn "veelvouden" van $P = (4, 4)$. Dus is $P = (4, 4)$ een voortbrenger van de hele kromme; de groep is cyclisch.

Het is nuttig op te merken dat er géén vermenigvuldiging van punten is: iets als PQ bestaat niet. De enige vermenigvuldiging die wel zinvol is is *scalaire* vermenigvuldiging: herhaald optellen geeft je een geheel getal maal een punt: nP voor een $n \in \mathbb{Z}$ en een punt op de kromme P kan wel. Maar let op: er is niet zoiets als scalair delen: $\frac{1}{2}P$ is (vaak) niet

gedefinieerd. Bijvoorbeeld, voor gegeven punten Q, R kan een vergelijking als $nQ = R$ meer dan één oplossing hebben: op de Bitcoin-kromme met $p = 11$ is $2(4, 4) = (6, 6)$ maar ook $2(7, 8) = (6, 6)$.

Voor een elliptische kromme over $\mathbb{F}_p$ heeft ieder punt een veelvoud dat gelijk is aan ∞. De kleinste positieve n waarvoor $nP = \infty$ geldt noemen we weer de *orde* van P. De orde van een punt is altijd weer een deler van de orde (aantal elementen inclusief ∞) van de groep.

En zo hebben we gevonden dat we op een zinvolle manier kunnen optellen op de Bitcoin-kromme over $\mathbb{F}_p$ voor (bijna) alle priemgetallen p. De "echte" Bitcoin-kromme is de zogeheten *secp256k1*-kromme, gegeven door de vergelijking $y^2 = x^3 + 7$ over $\mathbb{F}_p$ met $p = 2^{256} - 2^{32} - 977$, in hexadecimale notatie:

$p =$ `FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F`,

en met voortbrenger $G = (x_G, y_G)$ met als coördinaten

$x_G =$ `79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798`,
$y_G =$ `483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8`.

De orde van G is

$n =$ `FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141`,

dat is zelf weer een priemgetal, en is gelijk aan het aantal punten op de kromme. Merk op dat n dichtbij p ligt, dit is geen toeval.

Er is een versie van de MCR-software, MCRE genaamd, m et een Bitcoin-kromme-calculator aan boord. De Bitcoin-kromme met de eigenlijke Bitcoin-parameters is voorgeprogrammeerd.

2.2.2 Discrete logaritmen op elliptische krommen

In feite hebben we het meeste werk nu gedaan. De Bitcoin-kromme heeft weliswaar een gekke optelling, maar dat is een kwestie van wennen. Deze optelling maakt er een groep van die in vrijwel alles lijkt op de groepen $\mathbb{F}_p^*$, als we tenminste de vertaalslag van multiplicatief naar additief kunnen maken. Die vertaalslag betekent dat we in plaats van de machten $g^n \pmod p$ in $\mathbb{F}_p^*$ nu kijken naar de scalaire producten nP van punten op de kromme. Voor het efficiënt berekenen van zo'n scalair product hebben we uiteraard het analogon van de kwadrateer-en-vermenigvuldig-methode, die heet nu de *verdubbel-en-optel*-methode. En het Discrete Logaritme-Probleem om voor gegeven p, g en y het getal x te vinden zodat $y \equiv g^x$

(mod p), wordt nu het zogeheten *Elliptische Kromme-Discrete Logaritme-Probleem*[5]: voor gegeven p en voortbrenger G van de Bitcoin-kromme over $\mathbb{F}_p$, en gegeven punt Q op die kromme, vind het getal n waarvoor $Q = nG$. Dat is ook een erkend moeilijk probleem. En voor Elliptische Krommen hebben we niet zoiets moois als de Getallenlichamenzeef die dit probleem in subexponentiële tijd kan oplossen: de best bekende methode doet het wel beter dan met brute kracht, maar blijft wel exponentieel. Brute kracht heeft complexiteit 2^k voor een kromme met $p \approx 2^k$, de best bekende methode doet het in tijd $2^{k/2}$, dat is wel veel beter, maar nog steeds exponentieel. Dat is mooi, want het betekent dat we om een zelfde complexiteit te krijgen als 128 bits brute kracht geeft, we slechts 256-bits hoeven te hebben voor p. Vandaar ook de gekozen waarde voor de Bitcoin-parameters.

2.2.3 ECDH: Diffie-Hellman op elliptische krommen

Nu is het simpel geworden om de hiervoor al behandelde, op Discrete Logaritmen gebaseerde, systemen over te zetten naar de wereld van de Bitcoin-kromme. Hier is het Bitcoin-Diffie-Hellman-protocol (dat overigens niet echt gebruikt wordt in Bitcoin zelf):

Systeemparameters De Bitcoin-kromme met het gekozen priemgetal $\mathbf{p}$, en de voortbrenger $\mathbf{G}$; ook het aantal punten $\mathbf{n}$ is nodig.

Sleutelparen Alice maakt een sleutelpaar $\mathbf{z}_A, \mathbf{Q}_A$ aan, als volgt: ze kiest een willekeurige $\mathbf{z}_A \in \{2, 3, \ldots, \mathbf{n} - 2\}$ als privé-sleutel, en berekent de bijbehorende publieke sleutel $\mathbf{Q}_A = \mathbf{z}_A\mathbf{G}$.
Bob doet hetzelfde, maar volledig onafhankelijk van Alice: hij kiest zijn eigen willekeurige $\mathbf{z}_B \in \{2, 3, \ldots, \mathbf{n} - 2\}$ als privé-sleutel, en berekent de bijbehorende publieke sleutel $\mathbf{Q}_B = \mathbf{z}_B\mathbf{G}$.

Uitwisselen publieke sleutels Alice en Bob sturen elkaar hun publieke sleutel, over een niet per se veilig communicatiekanaal.

Berekenen gedeelde geheim Alice berekent nu het punt $\mathbf{S}_A = \mathbf{z}_A\mathbf{Q}_B$, en Bob berekent het punt $\mathbf{S}_B = \mathbf{z}_B\mathbf{Q}_A$.

Alice en Bob hebben nu hetzelfde punt berekend: $\mathbf{S}_A = \mathbf{z}_A\mathbf{Q}_B = \mathbf{z}_A(\mathbf{z}_B\mathbf{G}) = (\mathbf{z}_A\mathbf{z}_B)\mathbf{G}$, en $\mathbf{S}_B = \mathbf{z}_B\mathbf{Q}_A = \mathbf{z}_B(\mathbf{z}_A\mathbf{G}) = (\mathbf{z}_B\mathbf{z}_A)\mathbf{G}$. Ze kunnen bijvoorbeeld de x-coördinaat van dit punt gebruiken als symmetrische sleutel.

[5] Ook al zijn we van multiplicatieve taal overgestapt op additieve taal, het woord "logaritme" blijft hier traditioneel toch gehandhaafd.

2.2.4 ECDSA – digitale handtekeningen op elliptische krommen

Ook het handtekeningen-systeem DSA is vrijwel direct over te zetten naar Elliptische Krommen.

Parameterkeuze De Bitcoin-kromme met het gekozen priemgetal $\mathbf{p}$, en de voortbrenger $\mathbf{G}$; ook het aantal punten $\mathbf{n}$ is nodig.

Sleutelpaar Alice kiest een willekeurige $\mathbf{z}$ met $1 < \mathbf{z} < \mathbf{n} - 1$ als privésleutel. Ze berekent haar publieke sleutel als $\mathbf{Q} = \mathbf{zG}$. In dit geval wordt er niet in een kleinere ondergroep gewerkt, omdat de parameters bij Elliptische Krommen toch al klein genoeg zijn.

Ondertekenen Alice kiest een random éénmalig getal $\mathbf{k}$ met $1 < \mathbf{k} < \mathbf{n}-1$. Ze berekent $\mathbf{R} = \mathbf{kG}$ en neemt diens x-coördinaat $\mathbf{r}$, en ze berekent $\mathbf{s} \equiv (\mathbf{h} + \mathbf{xr})\mathbf{k}^{-1} \pmod{\mathbf{n}}$. De handtekening is dan $(\mathbf{r}, \mathbf{s})$.

Verificatie van de handtekening Bob herberekent zelf de hash $\mathbf{h}$, en berekent dan $\mathbf{u}_1 \equiv \mathbf{hs}^{-1} \pmod{\mathbf{n}}$ en $\mathbf{u}_2 \equiv \mathbf{rs}^{-1} \pmod{\mathbf{n}}$, en daarmee $\mathbf{R} = \mathbf{u}_1\mathbf{G} + \mathbf{u}_2\mathbf{Q}$. Hij neemt de x-coördinaat van dit punt, en verifieert of dat precies $\mathbf{r}$ is.

Deze methode wordt in het Bitcoin-systeem gebruikt, zie Hoofdstuk 3.

2.3 Hashfuncties

2.3.1 Digitale vingerafdrukken, originelen en botsingen

We hebben al kort gezien waar hashfuncties voor gebruikt worden. Het idee is dat een input van willekeurige lengte door een hashfunctie wordt omgezet in een output van vaste lengte, waarbij de output de input min of meer uniek kan identificeren. Het is een poging om vingerafdrukken van mensen na te bootsen.

Wat zijn de eisen op een vingerafdruk? Makkelijk te maken, op te slaan en te verwerken, maar moeilijk na te maken, en uniek voor de persoon.

Een cryptografische hashfunctie h van lengte n is een functie h die op efficiënt berekenbare wijze willekeurig lange bitrijtjes omzet in bitrijtjes van lengte n, en voldoet aan de volgende veiligheidseisen:

botsing-bestendig het is ondoenbaar om een *botsing* te vinden: twee ver-

schillende berichten $\mathbf{m}_1, \mathbf{m}_2$ met dezelfde hash: $h(\mathbf{m}_1) = h(\mathbf{m}_2)$,

origineel-bestendig het is ondoenbaar om voor een gegeven bitrijtje h_0 een *origineel* te vinden: een bericht $\mathbf{m}$ waarvoor $h(\mathbf{m}) = h_0$,

2e-origineel-bestendig het is ondoenbaar om voor een gegeven bericht $\mathbf{m}_1$ een *tweede origineel* te vinden: een ánder bericht $\mathbf{m}_2 \neq \mathbf{m}_1$ waarvoor $h(\mathbf{m}_2) = h(\mathbf{m}_1)$.

"Ondoenbaar" betekent hier: met een flinke marge is er in de wereld onvoldoende rekenkracht beschikbaar om dit binnen flink ruime tijd te doen. Het punt is niet dat er geen botsingen en (tweede) originelen bestaan, die bestaan (vrijwel) zeker, alleen al omdat de ruimte van mogelijke originelen oneindig groot is, en de ruimte van mogelijke hashes eindig. Het punt is dat je, zelfs als je exceptioneel paranoïde bent, toch het risico wel kunt nemen, omdat de kans op het vinden van een botsing of (tweede) origineel meer dan astronomisch klein is.

Een afgeleide eigenschap is dat een hashfunctie een *random functie* is: de hashwaarde van elke bericht is statistisch niet te onderscheiden van een random rijtje bits van dezelfde lengte.

Bekende hashfuncties zijn MD5, SHA1, RIPEMD160, SHA256 en Keccak. Bitcoin gebruikt SHA256 en RIPEMD160. Technieken voor het ontwerpen van hashfuncties lijken erg op technieken voor het ontwerpen van symmetrische versleuteling: je wilt ook dat elke invoerbit invloed heeft op elke uitvoerbit. Het berekenen van een hash wordt soms[6] wel "versleutelen met een hashfunctie" genoemd.

MD5 kom je nog veel tegen, hoewel in 2004 de botsing-bestendigheid ervan werd gebroken. Ook SHA1 is niet meer botsing-bestendig, sinds 2012. In hoofdstuk 4 komen we hier op terug met een voorbeeld van waarom dit erg is. SHA256 is nu de wereldwijde standaard. Keccak is de beoogde opvolger van SHA256, met een totaal andere ontwerpfilosofie dan MD5, SHA1 en SHA256 hadden.

Er is een subtiel maar belangrijk verschil tussen aan de ene kant botsing-bestendigheid, en aan de andere kant (tweede) origineel-bestendigheid. Omdat hashfuncties een eindige hash-lengte hebben, is er in theorie altijd een aanval met brute kracht mogelijk: probeer maar net zo lang willekeurige berichten[7] te hashen, je zult dan op de lange duur altijd wel een botsing of een (tweede) origineel vinden. De vraag is nu alleen: hoe lang is

[6] Op het Internet, door mensen die het niet begrijpen.

[7] Of minder willekeurige: je kunt net zo goed systematisch alle mogelijke bitrijtjes gaan aflopen, dat werkt even goed.

die duur? En daar zit het belangrijke en subtiele verschil. Kansrekenaars hebben het dan over de *verjaardagsparadox*. Die zegt: je hebt een groep van 253 mensen nodig om een kans van 50% te bereiken dat in die groep iemand op dezelfde dag jarig is als jij, maar je hebt slechts een groep van 23 mensen nodig om een kans van 50% te bereiken dat in die groep twee mensen op dezelfde dag jarig zijn. Doe je dit niet voor verjaardagen maar voor een functie $f(x)$ met n mogelijke uitkomsten, dan heb je een constante maal n willekeurige x nodig om een vooraf gegeven uitkomst $f(x) = f_0$ te krijgen, maar slechts een constante maal $\sqrt{n}$ paren x_1, x_2 om een botsing te krijgen: $f(x_1) = f(x_2)$.

Voor de veiligheid van hashfuncties in termen van de lengte betekent dit een enorm verschil. Voor een hashfunctie met hashlengte n bits is de complexiteit van een brute kracht-aanval op het (tweede) origineel-probleem ongeveer 2^n, maar voor een brute kracht-aanval op het botsing-probleem is dat $2^{n/2}$. Wil je dezelfde veiligheid bereiken die je krijgt met een symmetrische versleuteling met een sleutellengte van 128 bits, dan zul je bij een hashfunctie dus een lengte van 256 bits moeten hebben. Dat is tegenwoordig dan ook de standaard.

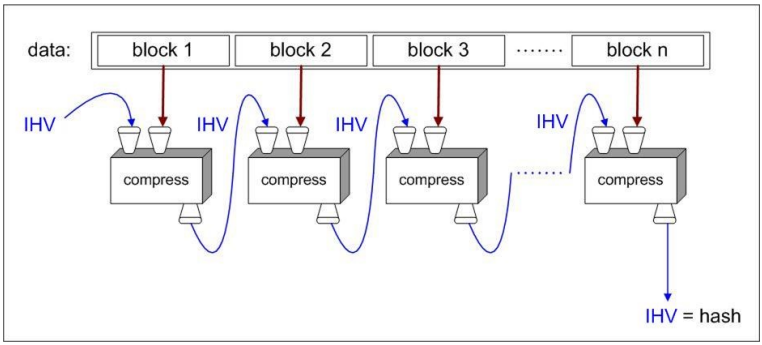
Het moge duidelijk zijn dat voor veilige digitale handtekeningen een (tweede) origineel-bestendige botsing-bestendige hashfunctie van groot belang is: anders zou je bij een bestaande geldige handtekening een nieuw bericht kunnen maken met precies diezelfde handtekening. Maar zelfs botsing-bestendigheid is nodig: anders zou één handtekening voor twee heel verschillende berichten kunnen gelden. In hoofdstuk 4 geven we daar een voorbeeld van.

2.3.2 SHA-256

We geven een idee van hoe SHA-256 in elkaar zit. Het is een hashfunctie die is ontworpen volgens de zogenaamde Merkle-Damgård-constructie, met de techniek van de *herhaalde compressie*. Dat betekent het volgende:

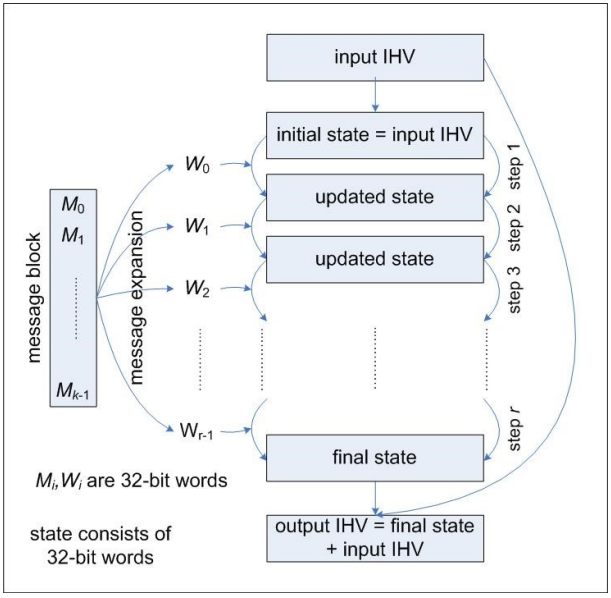
- het bericht is opgedeeld in blokken van een gegeven lengte (nadat *padding* is toegepast: het toevoegen van loze bits, om een veelvoud van de bloklengte te krijgen),
- er is een *compressiefunctie* die als invoer een blok data heeft, én een *tussen-hash* (IHV genaamd, voor Intermediate Hash Value), en die als uitvoer een nieuwe tussen-hash geeft:
  $\text{IHV}_i = \text{compress}(IHV_{i-1}, \mathbf{m}_i)$,

- dan heb je een begin-hash IHV_0 nodig: dat is een vast rijtje bits, en de eind-hash (de laatste tussen-hash) is dan de uitkomst van de hashfunctie.



Figuur 2.3: De Merkle-Damgård-constructie.

Het voordeel is dat je nu alleen nog maar een compressiefunctie hoeft te ontwerpen, met vaste invoer- en uitvoerlengtes. Overigens bestaat de padding niet alleen uit loze bits: ook de lengte van het oorspronkelijke bericht wordt erin gecodeerd, om zo bepaalde aanvallen gebaseerd op bits aanplakken te voorkomen. De compressiefunctie ziet er als volgt uit:



Figuur 2.4: De compressiefunctie.

- het bericht-blok wordt uitgevouwen tot r woorden $W_0, W_1, \ldots, W_{r-1}$. De functie kent r stappen, één voor ieder woord,
- er is een "toestand" ("state") met de gewenste bitlengte; die toestand wordt in elke stap ververst,
- aan het begin wordt de IHV in de toestand geladen,
- aan het eind wordt de uitvoer-IHV berekend als de XOR van de toestand en de invoer-IHV.

Een van de belangrijkste eisen waarin een hashfunctie fundamenteel verschilt van een versleutelingsfunctie is dat een versleutelingsfunctie omkeerbaar moet zijn, maar een hashfunctie niet omkeerbaar mag zijn. Dat is dan ook de reden voor het optellen van de invoer-IHV aan het eind.

SHA-256 heeft een bloklengte van 512 bits. Padding werkt zo:

- voeg een 1-bit toe (om te weten waar het bericht ophoudt en de padding begint),
- voeg zoveel 0-bits toe zodat de lengte 64 minder is dan een veelvoud van 512,
- codeer de bitlengte van het oorspronkelijke bericht in 64 bits, en voeg die toe.

Een gevolg is dat SHA-256 geen berichten aankan met lengte $\geq 2^{64}$. Maar dat is niet echt een probleem: er zullen voorlopig geen berichten voorkomen van langer dan 2 miljoen terabyte.

De begin-IHV bestaat uit 8 woorden van elk 32 bits; hiervoor zijn van de wortels van de eerste acht priemgetallen de 32 bits direct na de binaire komma gekozen (dit zijn zogenaamde NUMS-getallen: Nothing Up My Sleeve, die geven een bepaalde garantie dat er geen achterdeurtjes in verstopt zouden zijn[8]).

SHA-256 gebruikt zes hulpfuncties, die werken op 32-bits woorden:

$$\begin{aligned} Ch(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z) \\ Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \\ \Sigma_0(x) &= S^2(x) \oplus S^{13}(x) \oplus S^{22}(x) \\ \Sigma_1(x) &= S^6(x) \oplus S^{11}(x) \oplus S^{25}(x) \\ \sigma_0(x) &= S^7(x) \oplus S^{18}(x) \oplus R^3(x) \\ \sigma_1(x) &= S^{17}(x) \oplus S^{19}(x) \oplus R^{10}(x) \end{aligned}$$

[8] Maar daar zijn de meningen in te cryptogemeenschap over verdeeld.

De gebruikte symbolen:

$\wedge$: bitsgewijze EN,
$\oplus$: bitsgewijze XOR,
$\neg$: bitsgewijze NIET,
R^n : schuiven naar rechts over n bits en links opvullen met nullen,
S^n : roteren naar rechts over n bits.

De namen Ch en Maj komen van *choose* en *majority*.

SHA256 heeft 64 constanten $K_0, K_1, \ldots, K_{63}$, ook weer NUMS-getallen: de eerste 32 bits na de binaire komma van de derdemachts-wortels van de eerste 64 priemgetallen.

Het berichtblok van 512 bits is opgedeeld in 16 woorden $W_0, W_1, \ldots, W_{15}$. Deze worden verder uitgevouwen tot in totaal 64 woorden, als volgt (hier betekent + de optelling modulo 2^{32}):

$$W_j \leftarrow \sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W_{j-15}) + W_{j-16}.$$

De toestand wordt gezien als een rijtje van 8 woorden van elk 32 bits: (a, b, c, d, e, f, g, h). In het begin worden die geladen met de 8 woorden van de eerste IHV:

$$(a, b, c, d, e, f, g, h) \leftarrow (H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8).$$

Dan worden 64 stappen uitgevoerd, genummerd van 0 tot 63, waarin de toestand telkens wordt ververst. Stap j is:

$$\begin{aligned}
T_1 &\leftarrow \Sigma_1(e) + Ch(e, f, g) + K_j + W_j, \\
T_2 &\leftarrow \Sigma_0(a) + Maj(a, b, c), \\
(a, b, c, d, e, f, g, h) &\leftarrow (a, b, c, d, e, f, g, h) + (0, 0, 0, T_1, 0, 0, 0, T_1 + T_2), \\
(a, b, c, d, e, f, g, h) &\leftarrow (h, a, b, c, d, e, f, g).
\end{aligned}$$

Na deze 64 stappen wordt bij de toestand nog de oorspronkelijke IHV opgeteld om de nieuwe IHV te krijgen:

$$\begin{aligned}
&(H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8) \leftarrow \\
&\quad (a, b, c, d, e, f, g, h) + (H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8).
\end{aligned}$$

Ik zou zeggen: leer dit allemaal vooral niet uit je hoofd. Maar hopelijk zie je in dat het goed te programmeren is in software en in hardware, en wees ervan overtuigd dat over iedere stap goed is nagedacht om weer zo goed mogelijk diffusie en confusie te bewerkstelligen.

2.3.3 Hash-puzzels

Hashes worden niet alleen gebruikt bij digitale handtekeningen, als een korte unieke identificatie van een bericht, maar ook op andere manieren. Een toepassing waar in Hoofdstuk 5 verder op wordt ingegaan is die van een *commitment*. Bij een commitment leg je je vast op een bepaald bericht, zonder dat je dat bericht zelf hoeft prijs te geven. Stel je voor dat ik wil aantonen dat ik heel goed de uitslag van de wedstrijd PSV-Ajax van zondag 23 september 2018 kan voorspellen. Ik denk dat het 10-0 gaat worden. Maar dat ga ik niet van de daken roepen, om twee redenen: ik wil niet afgaan als het toch anders uitpakt, en ik wil de wedstrijd niet beïnvloeden. Dan bereken ik een hash van het volgende bericht:

"Voorspelling van Benne: PSV wint op 23 september 2018 met 10-0 van Ajax."

De SHA256 van dit bericht is

```
88F1E664 4E8A255D D33874B4 A917091C C1DA0703 A36FB0C4 D2E97F43 0B5EECD2.
```

Nu publiceer ik op mijn veelgelezen twitterpagina alléén deze hash, ruim voor de wedstrijd begint. Twitter is zo vriendelijk om er precies bij te zetten wanneer ik dat bericht geplaatst heb, dus daar kan geen twijfel over komen[9]. Op 23 september 2018 blijkt uiteraard dat PSV inderdaad met 10-0 van Ajax gewonnen heeft, en dan ben ik blij: niet omdat PSV gewonnen heeft want voetbal interesseert mij geen barst, maar omdat de hele wereld kan zien hoe goed ik kan voorspellen[10].

Een heel andere toepassing van hashfuncties wordt in Bitcoin gebruikt: de zogenaamde hashpuzzels. Die kunnen namelijk zorgen voor een *bewijs van werk* (de Engelse term is *proof of work*), een essentieel onderdeel van Bitcoin en veel andere blockchains.

Hashfuncties horen zich te gedragen als random-functies. Daarop gebaseerd is het idee van een hashpuzzel. Als de hele hash willekeurig is, geldt dat ook voor een klein deel, bijvoorbeeld de eerste n bits, voor een kleiner getal n dan de volledige bitlengte. Het betekent dat je verwacht dat in 1 op de 2^n gevallen een gegeven patroon van n bits gaat voorkomen in de hash. En dat betekent weer dat als iemand een bericht produceert waarvan de hashwaarde een bepaald patroon van n bits erin heeft (bijvoorbeeld de eerste n bits moeten allemaal 0 zijn), dan mag je ervan uitgaan dat die persoon daar ongeveer 2^n hashberekeningen voor heeft moeten doen. Dat is dan in feite een brute kracht-aanval tegen de origineel-bestendigheid van

[9] Vroeger zou ik een advertentie in de krant hebben gezet.
[10] En als de uitslag anders is, verwijder ik stilletjes het bericht van Twitter.

de tot n bits ingekorte hashfunctie.

Een voorbeeld: ik was op zoek naar een rijtje hoofdletters met een SHA256-hash die begint met 20 nul-bits. Hier is er eentje:

```
m = FRFLLCRHBXCYUOOXOTIETULOPMQFOEBTHUTYYJPPRSOJOCNHQQ
```

heeft als SHA256-hash:

```
00000CC1 C71511D8 3641ED08 401A7603 D9FEB947 843F7DCB 96B4752C A269081B.
```

Ik heb hiervoor 453422 SHA256-berekeningen moeten uitvoeren, wat in de buurt komt van de verwachte 2^{20}. De publicatie van zo'n tekst `m` kan dan gezien worden als een bewijs dat ik ongeveer zoveel werk heb moeten doen. De verificatie van het bewijs is bijzonder eenvoudig: bereken gewoon de hash, en kijk of aan de gestelde eis is voldaan. Dit principe wordt in Bitcoin gebruikt om de delvers een beloning te geven: een delver moet met een bericht komen waarvan de SHA256-hash voldoende nulbits vooraan heeft; dat heeft de delver alleen kunnen vinden een grote hoeveelheid hashes te hebben uitgeprobeerd (en daarmee flink veel energie verbruikt). Cryptografie is tegenwoordig niet altijd meer zo heel erg milieuvriendelijk...