Joan Daemen
Digital Security Groep
Radboud Universiteit Nijmegen
joan@cs.ru.nl

Symmetrische cryptografie 2.0

Joan Daemen is hoogleraar in de Digital Security-groep van de Radboud Universiteit, Nijmegen en verder werkzaam voor STMicroelectronics. Daemen is gespecialiseerd in symmetrische cryptografie, mede-uitvinder van versleutelingsstandaard AES en hashstandaard SHA-3, en andere cryptografische schema's. In dit artikel geeft Daemen een toegankelijke uiteenzetting over symmetrische cryptografie en de nieuwe richting die het vakgebied is ingeslagen.

Doelstelling

Alice en Bob willen communiceren in omstandigheden waar er personen of organisaties zijn die er belang bij hebben om de inhoud van hun communicatie te kennen of te manipuleren. Alice en Bob willen de communicatie dus beveiligen in de zin dat nieuwsgierige neuzen zoals Facebook, Google of de NSA niet kunnen meeluisteren en dat Alice en Bob kunnen zien als er met hun communicatie wordt geknoeid.

Voorbeelden van communicatie tussen personen zijn e-mail, chatsessies, telefoongesprekken of Skypesessies en dergelijke. Maar het is ook belangrijk communicatie te beveiligen tussen computerprocessen zoals tussen een betaalterminal en een centraal banksysteem of een virtueel private netwerk-verbinding. Andere voorbeelden zijn bescherming van gegevens in langetermijngeheugen zoals harde schijven van laptops, smartphones of tablets of gegevens waarvan de opslag wordt uitbesteed aan dienstverleners zoals Dropbox.

De veiligheid die we willen garanderen omvat doorgaans twee aspecten. Het eerste aspect is geheimhouding, waarmee we bedoelen dat alleen de rechtmatige personen toegang hebben tot de inhoud. Het tweede aspect is integriteit, ook wel *waarmerken* genoemd, waarmee we bedoelen dat de rechtmatige personen kunnen zien als er met de inhoud van de communicatie of opslag is geknoeid.

Er zijn nog andere aspecten van communicatie en opslag die kunnen worden beveiligd. Zo kan men proberen ervoor te zorgen dat opslag of communicatie gegarandeerd is. Een ander aspect is het proberen te verbergen van het bestaan van communicatie tussen twee partijen, of het feit dat er op een langetermijngeheugenmedium gegevens zijn opgeslagen. Dit zijn maar een paar voorbeelden.

In dit artikel beperken we ons echter tot het beschermen van de geheimhouding en het waarmerken van communicatie en opslag.

Asymmetrische cryptografie

De techniek om geheimhouding te bewerkstelligen is *versleuteling*. Voor de gegevens worden verzonden of opgeslagen, worden ze versleuteld en na ontvangst of ophalen worden ze ontsleuteld. Voor ontsleuteling is een geheime sleutel nodig, die alleen in het bezit is van de rechtmatige ontvanger. Zo'n sleutel is zoals een geheim wachtwoord, maar niet noodzakelijk gemakkelijk te onthouden. Voor versleuteling is er in principe geen geheime sleutel nodig, maar dit vereist wel een openbare sleutel die verbonden is met de geheime sleutel van de ontvanger.

De techniek om berichten of dossiers te waarmerken wordt (elektronische) handtekening genoemd. Hierbij wordt aan het bericht of dossier een bitrij toegevoegd, handtekening genoemd. Het aanmaken van zo een handtekening vereist een geheime sleutel, die alleen in het bezit is van degene die de handtekening zet. De verificatie van een handtekening kan met een openbare sleutel die gelinkt is aan de geheime sleutel waarmee het dossier getekend werd.

De zojuist besproken technieken zijn voorbeelden van asymmetrische cryptografie, zo genoemd omdat de twee bewerkingen verschillende sleutels vereisen. Asymmetrische cryptografie heeft het voordeel dat de eigenaar van een geheime sleutel, meestal private sleutel genoemd, deze met niemand moet delen en dat de openbare sleutel publiek mag worden gemaakt. Een nadeel is dat de private sleutel in theorie uit de openbare sleutel kan berekend worden. De stille hoop is dat dit echter zoveel rekenwerk zou kosten dat het onbetaalbaar is. Dit laatste kunnen we jammer genoeg nooit zeker weten en boven asymmetrische cryptografische schema's hangt dan ook altijd een zwaard van Damocles: ze kunnen elk ogenblik gebroken worden. Om toch enige betrouwbaarheid te geven, wordt er geavanceerde wiskunde ingeschakeld zoals getaltheorie en elliptische krommen. De redenering is dan dat je het schema alleen kan breken als er een historisch moeilijk wiskundig probleem is opgelost. Een voorbeeld van zo'n probleem is het ontbinden van zeer grote getallen in priemfactoren. Op die manier staat asymmetrische cryptografie op de schouders van wiskundige reuzen zoals Fermat, Euler, Gauss en Galois, wat ze een grote geloofwaardigheid verleent. Een nadeel van deze sterk wiskundige fundamenten is dat asymmetrische cryptografie zeer reken-

intensief en volumineus is. Alsof dat niet genoeg is, maakt de laatste tijd het spook van kwantumrekenen opgang. Kwantumrekenen is een hypothetische vorm van rekenen die heel veel bewerkingen tegelijk, *in superpositie*, kan doen en in staat zou zijn historisch moeilijke problemen efficiënt op te lossen. Kwantumrekenen is momenteel nog science fiction en even hypothetisch als economisch rendabele kernfusie, maar creëert toch een hoop onrust. Een positief bijverschijnsel van de kwantumcomputerdreiging is dat er veel fondsen worden vrijmaakt voor cryptografisch onderzoek, met name naar zogenaamd kwantumveilige asymmetrische cryptografie.

Symmetrische cryptografie

In een andere tak van de cryptografie gebruikt men dezelfde sleutel voor versleuteling en ontsleuteling. Ook het zetten van een handtekening en het verifiëren ervan vereisen dezelfde geheime sleutel. Daarom wordt deze tak *symmetrische cryptografie* genoemd. Symmetrische cryptografie heeft niet de besproken nadelen van asymmetrische cryptografie en is grootte-orden minder rekenintensief voor dezelfde beveiligingsgraad. Het heeft echter het nadeel dat bij verzending van gegevens de zender een geheime sleutel moet delen met de ontvanger. Hiervoor zijn veel oplossingen. Zo kunnen twee personen met het oog op toekomstige communicatie een geheime sleutel afspreken als ze elkaar ergens ontmoeten. Als dit niet praktisch is kunnen ze bijvoorbeeld asymmetrische cryptografie gebruiken voor het opzetten van symmetrische sleutels. Dit laatste is zinvol want zo beperken ze het gebruik van de trage cryptografie voor het opzetten van een korte sleutel en schakelen ze de snelle cryptografie in voor het beveiligen van de eigenlijke data. Dit artikel gaat over technieken in symmetrische cryptografie en we gaan ervan uit dat er een geheime sleutel beschikbaar is. Het genereren, verdelen en uit gebruik nemen van sleutels is een discipline op zich, die sleutelbeheer wordt genoemd. Dit is een interessant domein en in praktijk van veel groter belang voor (on)-veiligheid dan cryptografie, maar dat is voor een andere keer.

Symmetrische cryptografie beslaat het ontwerpen en ontleden van schema's die willekeurige hoeveelheden gegevens aankunnen. We spreken van schema's voor versleuteling en digitale handtekening. Die laatsten worden gewoonlijk MACs genoemd, wat staat voor *message authentication codes*. De laatste tijd zijn daar ook hybride schema's bijgekomen die versleuteling en waarmerken met elkaar combineren. Deze schema's vormen, samen met het (cryptografisch) *prakken* (hashing in Engels), de hoofdtaken van symmetrische cryptografie. Een prakfunctie zet een bericht van willekeurige lengte om tot een reeks tekens die er volledig willekeurig uitzien. Cryptografisch prakken heeft vele toepassingen waaronder identificatie en asymmetrische handtekeningen, maar dit beschrijven zou ons te ver leiden.

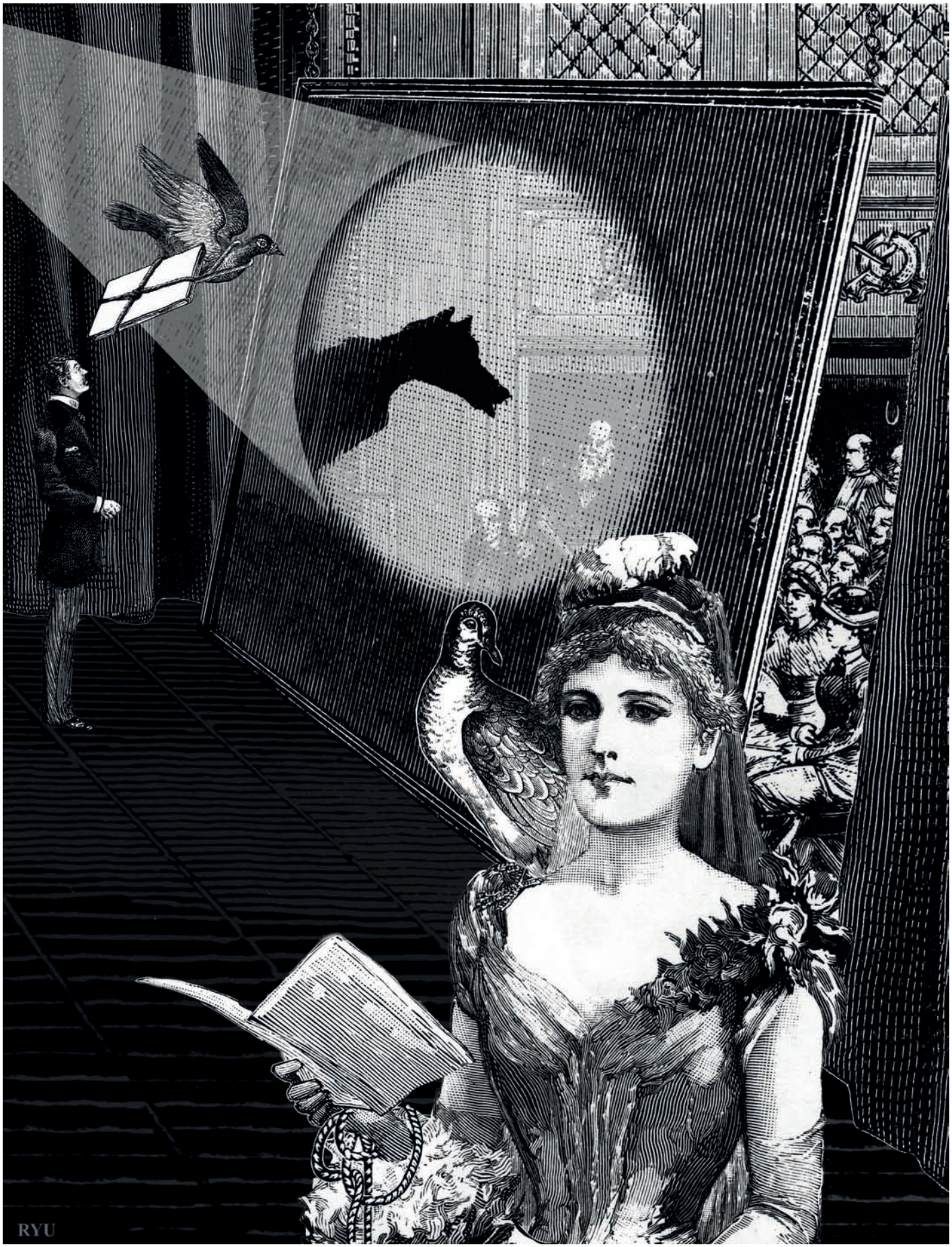
Symmetrische cryptografie, oude stijl

Een moeilijkheid van cryptografisch ontwerp is schema's te bouwen die willekeurige hoeveelheden gegevens aankunnen. In de jaren zeventig is de aanpak gelanceerd die tot op heden symmetrische cryptografie domineert. Deze aanpak bestaat erin om een schema te bouwen dat berichten van een bepaalde lengte kan versleutelen tot cryptogrammen met dezelfde lengte. Zo'n schema noemen we een *blokcijfer* en het blokcijfer waar het allemaal mee begon was de zogenaamde Data Encryption Standard (DES) [10]. DES kan berichten van 64-bit (de zogenaamde bloklengte) versleutelen en het resultaat terug ontsleutelen, allebei onder een geheime sleutel van 56 bits. Mathematisch zou je kunnen zeggen dat DES bestaat uit een verzameling van 2^{56} 64-bit permutaties en dat de sleutel er één van selecteert. De veiligheid die gewoonlijk van een blokcijfer verwacht wordt is het volgende: een aanvaller die de sleutel niet kent kan het blokcijfer niet onderscheiden van een willekeurig permutatie. Dit heet pseudorandom permutation (PRP) veiligheid.

Door de tijd is het inzicht gegroeid dat het onmogelijk is om een blokcijfer te bouwen waarvoor we wiskundig kunnen bewijzen dat het PRP-veilig is. Hoe kunnen we dan vertrouwen hebben in een blokcijfer? Vertrouwen is gebaseerd op openbaar kritisch onderzoek. DES was niet zo'n slecht ontwerp maar het kon beter. Na de publicatie van DES is het openbaar ontwerp en onderzoek van blokcijfers dan ook een academische bezigheid geworden. Het schrijven van artikelen over breken en verbeteren van blokcijfers was een manier om een academische loopbaan op te bouwen. Deze goedaardige wapenwedloop leidde tot steeds betere blokcijfers, met als hoogtepunt de verkiezing van Rijndael [8] als Advanced Encryption Standard (AES) in 2000 [11]. AES heeft een bloklengte van 128 bits en ondersteunt sleutels van 128, 192 en 256 bits. We kunnen nog steeds niet bewijzen dat AES PRP-veilig is, maar het feit dat tot op heden niemand erin is geslaagd om AES te breken ondanks verwoede pogingen van vele experts maakt het zeer onwaarschijnlijk dat AES nog gebroken zal worden.

We kunnen nu berichten van 64 bits versleutelen met DES en berichten van 128 bits met AES, maar wat doen we met berichten die een andere lengte hebben? Hiervoor is in de loop der jaren een arsenaal aan technieken voorgesteld om versleutelingsschema's te bouwen met een blokcijfer als bouwblok. Deze worden werkingsmodes voor versleuteling genoemd. Ook een MAC-berekening kan men doen op basis van blokcijfers door middel van werkingsmodes voor waarmerken. Meer recent maken werkingsmodes opgang die zowel versleutelen als waarmerken. In tegenstelling tot blokcijfers kan men voor deze werkingsmodes wel bewijzen dat ze aan bepaalde veiligheidsvereisten voldoen, op voorwaarde dat het gebruikte blokcijfer PRP-veilig is. Zelfs cryptografisch prakken wordt gedaan met blokcijfers, alhoewel daar PRP-veiligheid niet voldoende is. Zo komt het dat bijna alle standaarden in symmetrische cryptografie tot voor kort gebaseerd waren op blokcijfers en hun werkingsmodes.

Voor de leek mag deze aanpak dan al naïef lijken, vele cryptografische experts zweren erbij en zijn ervan overtuigd dat dit de juiste weg is. Het blokcijfermodel is recentelijk zelfs nog *verfijnd* door toevoeging van een bijkomende parameter naast de sleutel: een zogenaamde *aanpasser*. Feit is dat blokcijfers zijn ontworpen om berichten van een bepaalde lengte efficiënt te versleutelen en ontsleutelen en dat ontsleuteling in de meerderheid van de werkingsmodes helemaal niet gebruikt wordt. Het is zelfs zo dat in de meeste werkingsmodes de omkeerbaarheid van blokcijfers als een beperking werkt voor de veiligheid: de zogenaamde verjaardagslimiet (zie ook het artikel van Bart Mennink in dit verband). Het lijkt er dan ook op dat er ruimte is voor een meer doordachte manier om aan symmetrische cryptografie te doen.



Illustratie: Ryu Tajiri

Die is er wel degelijk en ik noem het graag symmetrische crypto 2.0.

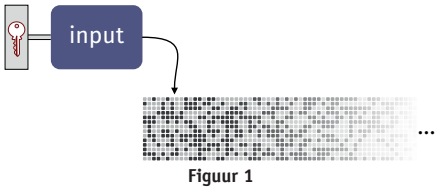
Symmetrische cryptografie 2.0

In symmetrische cryptografie 2.0 worden de twee rollen tot nu toe vervuld door blokcijfers verdeeld over twee andere functies. De eerste zijn zogenaamde onvoorspelbare functies, die zullen dienen om cryptografische functies mee te definiëren door er werkingsmodes op toe te passen. De tweede zijn cryptografische permutaties, die we zullen bouwen en dan gebruiken in constructies om er de eerder vermelde onvoorspelbare functies mee te bouwen.

Onvoorspelbare functies

In plaats van blokcijfers, stellen we hier een ander type functie centraal, die we *onvoorspelbare functie* (OVF) noemen. Dit is een functie die op basis van een korte sleutel en een argument van willekeurige lengte, een resultaat teruggeeft van onbeperkte lengte. In Figuur 1 illustreren we onze OVF op schematische wijze.

De veiligheidsdoelstelling voor een OVF heet *pseudo-willeurige-functie-veiligheid* (PRF) en dit is het volgende. Voor een tegenstander die de sleutel niet kent, ziet het resultaat van een OVF er volledig willekeurig uit, met als enige beperking dat de OVF met dezelfde argumenten hetzelfde resultaat teruggeeft. Voor verschillende argumenten daarentegen geeft een OVF totaal ongerelateerde resultaten terug, ook al verschillen die argumenten maar heel weinig. We zullen later bespreken hoe we zo'n OVF kunnen bouwen, maar eerst gaan we eens kijken wat we ermee kunnen doen, en verfijnen daarbij het concept.



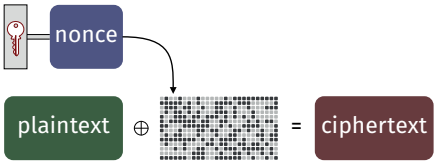
Figuur 1

Versleuteling

Er is een methode voor versleuteling die zeer veilig is: het zogenaamde *one-time pad*. Hierbij gebruiken zender en ontvanger een sleutel die even lang is als het bericht. We noemen dit een *sleutelstroom*. De zender telt deze sleutelstroom bit-per-bit op bij het bericht en de ontvanger trekt deze sleutel er weer van af. Voor een tegenstander die geen informatie heeft over de sleutel, met andere woorden, voor wie de sleutel er volledig willekeurig uitziet, geeft het cryptogram geen informatie over de inhoud van het versleutelde bericht. Dit kan men eenvoudig bewijzen, in de veronderstelling dat er geen bits van de sleutelstroom gebruikt worden voor de versleuteling van meerdere berichten.

Het grote nadeel van de one-time pad is dat zender en ontvanger een sleutel moeten delen die even lang is als de totale te verwachten communicatie en dit is onpraktisch voor de meeste toepassingen. We wensen de communicatie te beveiligen met een korte sleutel, en dit is waar de OVF om de hoek komt piepen. Zender en ontvanger gebruiken een OVF om een sleutelstroom te genereren met een gedeelde sleutel en argument en gebruiken deze voor one-time pad-versleuteling van het bericht. We beelden de versleuteling af in Figuur 2.

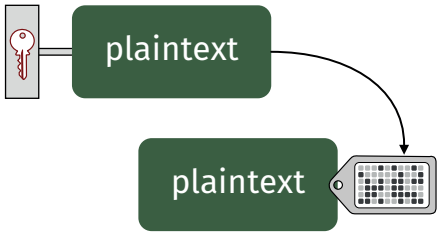
Deze methode is veilig als de OVF veilig is en als het argument voor elk bericht uniek is. Met andere woorden, het argument moet een zogenaamde *nonce* zijn. In de meeste toepassingen is er een nonce voorhanden. Bij e-mail kan men bijvoorbeeld de datum en tijd van verzending gebruiken. Voor dossiers op een computer kan men de naam en de plaats in de folder gebruiken.



Figuur 2

MAC-berekening

Een MAC-functie berekent een korte tag op basis van een sleutel en bericht van willekeurige lengte. De veiligheid van een MAC-functie bestaat erin dat het voor een tegenstander moeilijk moet zijn om *vervalsingen* te doen. Een vervalsing bestaat erin een ontvanger een bericht met tag te laten aanvaarden waarbij een tegenstander de tag heeft berekend in plaats van de legitieme afzender. Meer concreet, als de tag een lengte heeft van n bits, mag de kans op succes bij elke poging tot vervalsing maar 2^{-n} zijn. Dit is het geval als de tags voor verschillende berichten er volledig willekeurig uitzien. Een OVF leent zich dus uitstekend voor het gebruik als MAC-functie. Voor het berekenen van een n-bit tag gebruiken we de OVF met als argument het bericht en kappen we het resultaat af na n bits, zoals afgebeeld in Figuur 3.

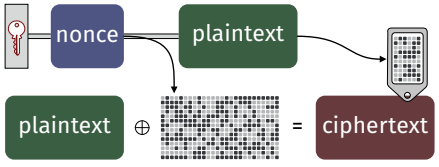


Figuur 3

Combinatie van versleuteling en MAC

In de meeste gevallen willen we graag berichten beschermen tegen zowel afluisteren als vervalsing. Dit kunnen we eenvoudig doen door het bericht te versleutelen en een tag te berekenen over het cryptogram. Beide bewerkingen kunnen we weer doen met onze OVF. Eerst berekent de zender een sleutelstroom op basis van een nonce en telt die op bij het bericht, resulterend in de cijfertekst. Dan berekent hij de tag op basis van deze nonce en de cijfertekst. Dit beelden we af in Figuur 4.

Dus het cryptogram bestaat uit een nonce, cijfertekst en tag. Bij ontvangst berekent de ontvanger de verwachte tag opnieuw op basis van de nonce en cijfertekst. Als de ontvangen en berekende tag niet gelijk zijn, verwerpt hij het cryptogram. Als ze wel gelijk zijn, berekent hij de sleutelstroom op basis van de ontvangen nonce en ontsleutelt de cijfertekst door de sleutelstroom erbij op te tellen.



Figuur 4

OVF met de oplopende eigenschap

We zien dus dat zender en ontvanger beiden twee OVF-berekeningen doen: één keer met als argument de nonce en één keer de nonce en het bericht. Het zou interessant zijn als de bewerking op de nonce gedeeld moest kunnen worden tussen de twee OVF-berekeningen. Hier kunnen we rekening mee houden als we onze OVF gaan bouwen. Meer concreet, we willen dat het argument van onze OVF bestaat uit een serie bitrijen in plaats van één enkele. We schrijven series van bitrijen met het symbool $\circ$ dat staat voor *volgend op*. Dus $B \circ A$ betekent: eerst bitrij A en dan bitrij B.

Twee series bitrijen zijn gelijk enkel als ze evenveel bitrijen bevatten en als de bitrijen één voor één aan elkaar gelijk zijn. De OVF toegepast op twee verschillende bitrijen $B \circ A$ en $D \circ C$ geeft dus verschillende resultaten, zelfs al is $A \,|\, B = C \,|\, D$, waarbij $|$ staat voor concatenatie.

In twee OVF-berekeningen met dezelfde sleutel en waarbij de eerste bitrijen gelijk zijn, kunnen deze geabsorbeerd worden in een enkele berekening. Een OVF met deze eigenschap noemen we *oplopend*. Vanaf hier gaan we ervan uit dat OVFs oplopend zijn. Een toepassing waar de oplopende eigenschap interessant is, is bij de beveiliging van een communicatiekanaal.

Beveiliging van een communicatiekanaal

Dikwijls willen we niet gewoon afzonderlijke berichten beschermen tegen afluisteren en vervalsing, maar een stroom berichten, of zelfs geluid- of beeldcommunicatie. We geven hier aan hoe we dat kunnen doen met een oplopende OVF. Om het eenvoudig te houden gaan we in ons voorbeeld uit van communicatie in één richting, maar het kan gemakkelijk veralgemeend worden.

Het protocol beveiligt een stroom berichten waarbij elk bericht bestaat uit klaartekst en randgegevens. We willen beiden waarmerken maar alleen de klaartekst beschermen tegen afluisteren. We spreken van een sessie. De zender zet elk bericht om in een cryptogram, bestaande uit de cijfertekst (versleutelde klaartekst), de (onversleutelde) randgegevens en een tag. De tag wordt niet alleen berekend op het cryptogram en randgegevens van het huidige bericht, maar ook op alle voorgaande berichten.

Dit gebeurt als volgt. Tijdens de sessie houden zender en ontvanger een *geschiedenis* bij die bestaat uit de serie cijfertekst en randgegevens bitrijen. Als de zender een nieuw bericht wil versleutelen, genereert hij de sleutelstroom door de OVF toe te passen op de geschiedenis op dat moment. Hij voegt dan de cijfertekst en de randgegevens toe aan de geschiedenis en berekent de tag door de OVF toe te passen op de geschiedenis.

Voor de veiligheid is het essentieel dat voor elke OVF-berekening het argument een andere waarde heeft. Uit de beschrijving hierboven zie je dat de tag van een cryptogram en de sleutelstroom van het volgende cryptogram worden berekend op basis van dezelfde geschiedenis. Om die reden neemt het protocol de tag als de eerste n bits van het OVF-resultaat en de sleutelstroom vertrekkende van bit n. Voor elke tagberekening verlengt de geschiedenis en dus heeft in elke sessie elke OVF-berekening een ander argument. Tussen verschillende sessies met dezelfde sleutel kan de eenmaligheid van het OVF-argument bewerkstelligd worden met een nonce. Zender en ontvanger starten de geschiedenis door er een nonce in te schrijven.

Gedurende de sessie wordt de geschiedenis steeds langer, maar in elke OVF-berekening is de geschiedenis gelijk aan de vorige berekening met twee bitrijen daaraan toegevoegd. Dankzij de oplopende eigenschap hoeft de OVF alleen maar deze laatste twee bitrijen te absorberen.

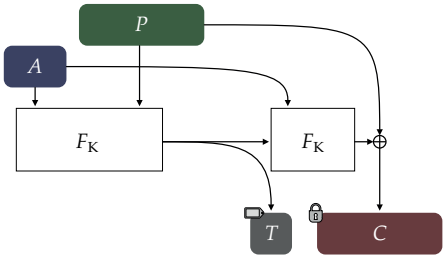
Voor de berekening van de tag moet de geschiedenis op eenduidige manier de serie cijfertekst en randgegevens coderen. Met andere woorden, uit de geschiedenis moet deze serie kunnen worden gereconstrueerd. Dit is geen probleem als elk bericht een klaartekst en randgegevens heeft, want de geschiedenis is simpelweg de opeenvolging van cijferteksten afgewisseld met randgegevens. Wanneer we ook berichten ondersteunen die alleen uit klaartekst of alleen uit randgegevens bestaan, is dit niet meer het geval. Dit kan echter gemakkelijk door cijfertekst en randgegevens te markeren met een bijkomende bit aan het einde zodat de resulterende bitrijen zich in verschillende domeinen bevinden. Vooraleer ze de geschiedenis vervoegen, markeren we dus elke cijfertekst met een bit 0 en elke randgegevens bitrij met een bit 1.

Versleuteling met ingebouwde nonce

Sommige ontwikkelaars hebben de grootste problemen in de wereld om voor een nonce te zorgen. Nonces gebaseerd op datum en tijd vereisen een correct werkende klok en meestal hebben ze er geen die ze kunnen of willen vertrouwen. Nonces gebaseerd op serienummers vereisen dat de zender over midden- of langetermijngeheugen beschikt en daar hebben ze ook helemaal geen vertrouwen in. Een andere oplossing, het gebruik van willekeurige bitrijen als nonces, heeft dan weer het nadeel dat we door de verjaardagsparadox relatief lange nonces moeten genereren en versturen. Bovendien is het genereren van willekeurige bitrijen iets waar deze mensen maar niet in slagen. Voor zulke mensen zou het goed zijn als we een bericht veilig kunnen versleutelen en waarmerken zonder nood aan een nonce.

We gaan weer berichten beveiligen bestaande uit klaartekst en randgegevens, en deze keer zonder te rekenen op een nonce. Dit impliceert een beperking: als twee berichten hetzelfde zijn, geeft versleuteling met dezelfde sleutel twee dezelfde cryptogrammen. Maar zo snel als twee berichten verschillen, willen we dat de cryptogrammen ook verschillen. Met name, we willen dat de sleutelstromen voor de versleuteling van de klaarteksten verschillen.

Een werkingsmode die dit realiseert heet kunstmatig-beginwaarde, oftewel synthetic initial value (SIV) [14]. Onze OVF leent zich hier perfect voor en dit leidt tot een heel simpele berekening. Eerst berekenen we de tag door de OVF te berekenen over randgegevens (A) gevolgd door de klaartekst (P). Dan berekenen we de sleutelstroom door de OVF te berekenen over randgegevens gevolgd door de tag (T). Dit geeft geen 100 procent garantie dat het OVF-argument verschilt voor elke sleutelstroom. Met name, het gaat fout als twee berichten dezelfde randgegevens hebben, én dezelfde tag. Maar we kunnen de kans dat dit gebeurt zo klein maken als we zelf willen door de tag lang genoeg te kiezen. De SIV-mode van een OVF ziet uit zoals in Figuur 5.



Figuur 5

Permutaties

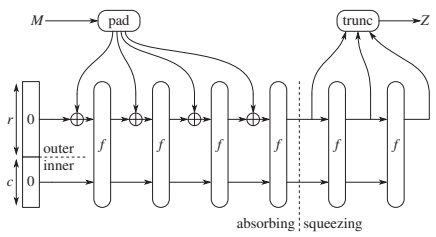
Één pijler van symmetrische cryptografie 2.0 is de vervanging van het blokcijfer door een OVF. De tweede pijler is het bouwblok waar we onze OVFs op gaan baseren: de permutatie. Een cryptografische permutatie beeldt een argument van b bits af op een resultaat van b bits, en dit op een omkeerbare wijze. Je zou het kunnen zien als een blokcijfer, maar er zijn wel belangrijke verschillen. Ten eerste heeft een permutatie geen sleutelargument. Ten tweede hebben we OVF-constructies voor ogen waar geen

nood is aan de berekening van de inverse permutatie, dus de inverse hoeft niet noodzakelijk efficiënt te zijn.

Voor het ontwerpen van permutaties kunnen we de ervaring gebruiken die we hebben opgedaan met blokcijfers. We herhalen gewoon een eenvoudige rondefunctie een aantal keer, dit afgewisseld met het optellen van rondeconstanten. De rondefunctie kunnen we samenstellen als de opeenvolging van een niet-lineaire laag, een menglaag en een dispersielaag, waarbij we die lagen kiezen en samenstellen volgens één of andere ontwerpstrategie. Een populaire ontwerpstrategie is de *strategie van het brede spoor*. Voor het kiezen van het aantal ronden moeten we kijken naar de OVF-constructie waarin de permutatie gebruikt wordt. In principe moeten we het aantal ronden zo kiezen dat de OVF een goede veiligheidsmarge geeft. Met andere woorden, we kijken naar het grootste aantal ronden dat in de context van een OVF gebroken kan worden, en tellen er een aantal ronden bij op. Natuurlijk hangt het vertrouwen dat men kan hebben in een permutatie af van het begrip van zijn potentiële zwakheden. Dit begrip hangt af van hoe overtuigend de auteurs het ontwerp kunnen verantwoorden en hun analyse, maar ook van de cryptanalyse door derden. Hoe meer helderheid en cryptanalyse, hoe groter het vertrouwen. We zullen nu twee voorbeelden geven van OVF-constructies die gebaseerd zijn op permutaties. Voor meer achtergrond over het ontwerp van permutaties verwijzen we naar het laatste hoofdstuk van [3] en [4].

De spons

De spons is een constructie voor het bouwen van cryptografische prakfuncties op basis van een permutatie [1,2]. Sinds haar introductie in 2007 is zij bezig aan een steile opgang, niet in het minst om het feit dat zij aan de basis ligt van de SHA-3 standaard [12]. Een andere troef is haar elegante structuur, zie Figuur 6.



Figuur 6

De spons verdeelt de b bits van de toestand in een *binnengedeelte*, bestaande uit *c* bits, en een *buitengedeelte*, bestaande uit *r* bits. De haalbare beveiligingsgraad hangt af van c: hoe kleiner, hoe zwakker. Dit bepaalt de efficiëntie, want $r = b - c$ en r is het aantal bits die de spons kan verwerken per uitvoering van de permutatie f.

Terwijl prakfuncties vroeger altijd een resultaat hadden van vaste lengte, kan de spons een resultaat van willekeurig lengte aan. Dit komt tot uiting in de SHA-3 standaard die een nieuwe term introduceert voor zo'n type functie: een eXtendable Output Function (XOF), en ook twee dergelijke functies definieert: SHAKE128 en SHAKE256.

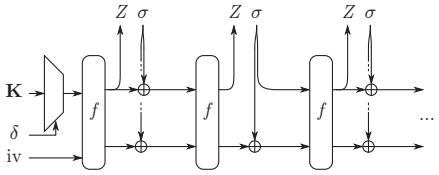
OVF op basis van een SHAKE-functie

De SHAKE-functies hebben slechts één argument en zijn dus niet onmiddellijk bruikbaar als OVF. Maar het volstaat om een injectieve codering van een sleutel en een serie bitrijen naar één enkele bitrij te bouwen om er een OVF van te maken. Met injectief bedoelen we dat het mogelijk is om de sleutel en de serie bitrijen terug op te bouwen alleen maar op basis van het resultaat. Voorbeelden van zulke codering kan men vinden in [13]. Bovendien is de spons oplopend. Een enkele oogopslag volstaat om te zien dat dit het geval is: de spons comprimeert alles wat ze absorbeert in een b-bit toestand.

De veiligheidsconcepten voor een XOF en een OVF liggen dicht bij elkaar. Met name, een zwakheid in de OVF impliceert een zwakheid van de onderliggende XOF. Dus vertrouwen in de XOF gaat over op de OVF. Maar de vereisten voor een XOF zijn eigenlijk veel strenger dan voor een OVF: een XOF moet weerstand bieden aan tegenstanders zonder dat er een geheime sleutel is, terwijl voor een OVF er wel zo'n sleutel is. Dit leidt ertoe dat een OVF op basis van een XOF suboptimaal is.

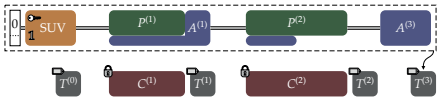
Gesleutelde duplex

Na een turbulente periode waarin varianten van de spons werden voorgesteld en bestudeerd, is er een constructie naar boven gekomen die zeer goed geschikt is om als OVF te gebruiken: de gesleutelde duplex [9]. Net als de spons werkt die op een b-bit toestand. In het begin wordt een sleutel en een beginwaarde (IV) in de toestand geschreven. Dan kunnen er zogenaamde *duplex-iteraties* worden gedaan, waarbij telkens een bitrij σ van maximaal b bits wordt geladen (in plaats van r bits zoals in de spons) en er een resultaat Z van maximaal r bits terugkomt, zie Figuur 7.



Figuur 7

Men kan hierop een OVF bouwen door middel van een eenvoudige codering. Door bijvoorbeeld elke bitrij σ te beperken tot maximaal $b-1$ bits en hieraan één bit 1 op het einde toe te voegen, hebben we een OVF, maar met de beperking dat het argument bestaat uit een serie bitrijen van elk maximaal $b-1$. Men kan er ook rechtstreeks modes op bouwen zoals bijvoorbeeld de communicatiekanaalbeveiligingsmode Motorist [6], die we schematisch illustreren in Figuur 8.



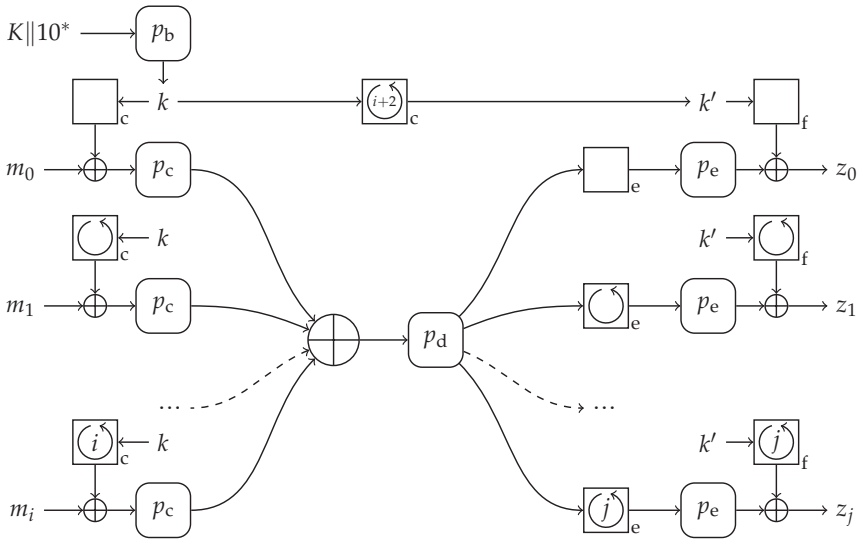
Figuur 8

Men kan zich afvragen: kan de gesleutelde duplex-constructie wel veilig zijn, en zo ja, wat is dan de veiligheidsgraad die je ermee kan krijgen? Een antwoord op deze vraag wordt gegeven door zogenaamde onderscheidingsbewijzen. Zulke bewijzen zeggen iets over de veiligheid van een constructie in de veronderstelling dat die gebruikmaakt van een willekeurige permutatie. In zo'n bewijs heeft een hypothetische tegenstander toegang tot een systeem dat ofwel de constructie is met een geheime sleutel, ofwel een ideale functie. A priori weet ze niet met welk van de twee systemen ze spreekt en ze mag dit proberen te achterhalen door aan het systeem het resultaat te vragen voor argumenten van haar keuze. Op het einde van het experiment moet ze gokken welke van de twee het is.

In het bewijs berekenen we dan een bovengrens voor de kans op succes dat ze de juiste keuze maakt. Deze bovengrens is een uitdrukking waarin typisch het totaal aantal bits in de argumenten en resultaten voorkomen en het aantal berekeningen uitgevoerd door de tegenstander en parameters van de constructie zoals de capaciteit c. Het is deze uitdrukking die aangeeft hoe hoog de potentiële veiligheidsgraad van de constructie is. Als de kans op succes pas afwijkt van

$1/2$ (kans op succes bij een pure gok) voor een aantal berekeningen tot 2^s en voor een aantal bevraagde bits tot 2^s, dan spreken we van een veiligheidsgraad van s bits.

Als we de constructie gebruiken met een concrete permutatie, is de onderscheidingslimiet niet langer van toepassing. Hij zegt namelijk alleen maar iets over de gezondheid van de constructie en maakt abstractie van de onderliggende permutatie. We gaan er stilzwijgend van uit dat een concrete permutatie in combinatie met de constructie bijvoorbeeld een veilige OVF oplevert. Of dit zo is, moet worden bevestigd door openbare cryptanalyse. Dit is vergelijkbaar met de PRP-veiligheid van blokcijfers. Als we bijvoorbeeld AES gebruiken in een of andere mode die PRP-veiligheid vereist, vertrouwen we stilzwijgend dat AES PRP-veilig is. Dit vertrouwen is niet gebaseerd op een of ander bewijs, maar op openbare cryptanalyse. Als we een permutatie, bijvoorbeeld Keccak-f, gebruiken in een of andere mode die vereist dat keyed duplex PRF-secure is, dan vertrouwen we dat Keccak-f veilig is in deze context. Bij het kiezen van het aantal ronden van een permutatie kunnen we verschillende keuzes maken. Als de doelstelling is dat de veiligheidsgraad van de functie niet lager is dan de veiligheidsgraad van de constructie met een willekeurige permutatie, spreken we van *hermetische* ontwerpstrategie. Dit is bijvoorbeeld het geval bij ons vercijferingsschema Keyak [6]. Maar we kunnen ook tolereren dat de veiligheidsgraad zakt en dit compenseren door hogere waarden voor de parameters in de constructie te kiezen. Zo heeft ons vercijferingsschema Ketje



Figuur 9

[5] een hogere capaciteit dan strikt nodig voor wat het behoort te doen.

Farfalle

De gesleutelde duplex levert een efficiënte oplopende OVF op die zeer weinig geheugen vereist maar één nadeel heeft: ze is strikt serieel en kan niet optimaal gebruikmaken van beschikbare rekenkracht in moderne processoren. Om die reden zou het interessant zijn om een meer parallelle OVF te bouwen. We zijn hieraan begonnen en het resultaat is Farfalle [7], zoals hier geïllustreerd in Figuur 9.

Farfalle heeft een compressiefase (links) en een expansiefase (rechts) en maakt gebruik van verschillende permutaties P_F, P_f en P_A. Het idee is dat deze permutaties dezelfde rondefunctie hebben en enkel verschillen in het aantal ronden. Verder maakt het gebruik van zogenaamde rolfuncties voor het laten variëren van maskers en de interne toestand. Dit zijn zeer lichtgewicht functies vergelijkbaar met lineaire terugkoppelingsregisters. Door de seriële schakeling kan het aantal ronden sterk gereduceerd worden in vergelijking met permutaties gebruikt in duplex voor dezelfde veiligheidsgraad. Bovendien zijn zowel de compressiefase en expansiefase zeer parallelliseerbaar en is ze tegelijk oplopend.

Besluit

Symmetrische cryptografie 2.0 vormt een interessant alternatief voor blokcijfers en zijn modes.

Dankwoord

Ik dank Gilles Van Assche voor het gebruik van zijn illustraties.

Referenties

1 G. Bertoni, J. Daemen, M. Peeters en G. Van Assche, Sponge functions, *Ecrypt Hash Workshop 2007*, May 2007, http://sponge.noekeon.org.

2 G. Bertoni, J. Daemen, M. Peeters en G. Van Assche, On the indifferentiability of the sponge construction, in: N. P. Smart, ed., *Advances in Cryptology–Eurocrypt 2008*, Lecture Notes in Computer Science, Vol. 4965, Springer, 2008, pp. 181–197.

3 G. Bertoni, J. Daemen, M. Peeters en G. Van Assche, Cryptographic sponge functions, January 2011, http://sponge.noekeon.org/.

4 G. Bertoni, J. Daemen, M. Peeters en G. Van Assche, The KECCAK reference, January 2011, http://keccak.noekeon.org.

5 G. Bertoni, J. Daemen, M. Peeters, G. Van Assche en R. Van Keer, CAESAR submission: KETJE v2, September 2016, http://ketje.noekeon.org.

6 G. Bertoni, J. Daemen, M. Peeters, G. Van Assche en R. Van Keer, CAESAR submission: KEYAK v2, document version 2.2, September 2016, http://keyak.noekeon.org.

7 G. Bertoni, J. Daemen, M. Peeters, G. Van Assche en R. Van Keer, Farfalle: parallel permutation-based cryptography, IACR Cryptology ePrint Archive, Report 2016/1188, http://eprint.iacr.org/2016/1188.

8 J. Daemen en V. Rijmen, *The Design of Rijndael–AES, the Advanced Encryption Standard*, Springer, 2002.

9 J. Daemen, B. Mennink en G. Van Assche, Full-state keyed duplex with built-in multi-user support, IACR Cryptology ePrint Archive, Report 2017/498, http://eprint.iacr.org/2017/498.

10 NIST, Federal information processing standard 46, Data Encryption Standard (DES), October 1999.

11 NIST, Federal information processing standard 197, Advanced Encryption Standard (AES), November 2001.

12 NIST, Federal information processing standard 202, SHA-3 standard: Permutation-based hash and extendable-output functions, August 2015, http://dx.doi.org/10.6028/NIST.FIPS.202.

13 NIST, NIST special publication 800-185, SHA-3 derived functions: cSHAKE, KMAC, TupleHash and ParallelHash (draft), August 2016, http://csrc.nist.gov/publications/drafts/800-185/sp800_185_draft.pdf.

14 P. Rogaway en T. Shrimpton, A provable-security treatment of the keywrap problem, in: S. Vaudenay, ed., *Eurocrypt*, Lecture Notes in Computer Science, Vol. 4004, Springer, 2006, pp. 373–390.