Faculteit
Ingenieurswetenschappen

Departement
Elektrotechniek – ESAT

KATHOLIEKE
UNIVERSITEIT
LEUVEN

Cryptanalyse van Hashfuncties

Eindwerk voorgedragen tot het behalen van het diploma van Burgerlijk Elektrotechnisch Ingenieur, Optie ICT-Multimedia en Signaalverwerking (Master in de Ingenieurswetenschappen: Elektrotechniek (ICT))

Nicky Mouha

Promotor:
Prof. Dr. Ir. Bart Preneel
Dagelijkse begeleiding:
Dr. Ir. Christophe De Cannière
Ir. Sebastiaan Indesteege
M.Sc. Özgül Küçük Öztarhan

2007 – 2008

Voorwoord

Op het einde van deze masterproef heb ik de eer en het genoegen om iedereen te bedanken die mij onderweg geholpen heeft.

Op de eerste plaats had ik graag mijn promotor Prof. Dr. Ir. Bart Preneel bedankt, die mij eveneens introduceerde tot het domein van de cryptografie. Voor het vakkundig beoordelen van de eindtekst bedank ik mijn assessoren, Prof. Dr. Ir. Vincent Rijmen en Prof. Dr. Ir. Luc Van Eycken.

Voor de talrijke opmerkingen, suggesties en inspirerende discussies had ik graag mijn dagelijkse begeleiders, Dr. Ir. Christophe De Cannière, Ir. Sebastiaan Indesteege en M.Sc. Özgül Küçük bedankt. Hen bedank ik ook voor mijn eerste kennismaking met wetenschappelijk onderzoek in de praktijk.

Verder bedank ik mijn familie en vrienden voor hun interesse en morele steun. Daarnaast wil ik ook de Katholieke Universiteit Leuven bedanken, inclusief alle professoren, assistenten en medestudenten waardoor ik de voorbije vijf jaar omringd werd. Deze masterproef is immers het resultaat van een vijfjarig vormingsproces, waarbij hun steun en toeverlaat onontbeerlijk is geweest.

Samenvatting

Hashfuncties vormen een belangrijke bouwblok uit de cryptografie. Zoals beschreven in [35], zijn dit functies h die een invoer m van een willekeurige lengte op een deterministische manier omzetten in een uitvoer $h(m)$ met een vaste lengte van n bits. Hierbij is het belangrijk dat aan een aantal veiligheidsvoorwaarden voldaan zijn, onder andere bestendigheid tegen botsingen. Een botsing bestaat uit twee invoerwaarden m, m' met $m \neq m'$ waarvoor $h(m) = h(m')$.

Recente aanvallen van X. Wang op onder andere de hashfuncties MD4 [42], MD5 [44], RIPEMD [42] en SHA-1 [43] tonen aan dat het voor een aantal hedendaagse hashfuncties echter wel haalbaar is om botsingen te vinden.

De hashfunctie HAS-V [34] heeft een structuur die gelijkaardig is aan deze hashfuncties. Er is slechts één paper bekend die zich toelegt op de cryptanalyse van HAS-V [28]. Resultaten die gebruik maken van de recente aanvallen op hashfuncties, werden nog niet eerder gepubliceerd. HAS-V bestaat uit twee parallelle stromen. Er wordt een variant opgesteld met één stroom, waarop de verdere cryptanalyse zich toelegt.

Een aantal bestaande technieken voor hashfuncties worden dan aangepast aan deze vereenvoudigde HAS-V en verder uitgebreid. Hierbij wordt gezocht naar een 'differentieel pad', een berichtverschil dat doorheen de interne toestanden van de hashfunctie gevolgd kan worden om een botsing te vinden.

Zo worden onder andere lokale botsingen [5] en L-karakteristieken onderzocht [37]. Bij deze techniek bleek de variabele rotatie en de structuur van de berichtexpansie ervoor te zorgen dat geen goede differentiële paden konden gevonden worden voor meer dan twee ronden.

Een techniek van M. Stevens die succesvol gebruikt werd voor MD5 [40], wordt aangepast voor de vereenvoudigde HAS-V. Pogingen om deze techniek te gebruiken, lukten niet omwille van verschillen tussen MD5 en HAS-V die deze techniek bemoeilijken. Een differentieel pad voor 45 stappen werd wel bekomen via handmatige constructie, zie Tabel 7.2. Hierbij werd de berichtexpansie evenwel licht gewijzigd. Een aantal problemen die de techniek van M. Stevens vertonen bij HAS-V, worden echter verholpen door de methode van C. De Cannière en C. Rechberger [12, 11].

Gebruik makende van deze techniek, wordt een differentieel pad (zie Tabel 8.10) voor 45 stappen gegeven. Een groot aantal berichtwoorden m_i hebben daar geen invloed op de botsing en zijn dus vrij te kiezen. Om tot een botsing te komen met dit differentiële pad, zijn naar schatting $2^{75,13}$ stapfunctieberekeningen nodig. Door de technieken uit [12] verder uit te breiden, wordt een beter differentieel pad (zie Tabel 8.11) opgesteld dat dit aantal vermindert tot $2^{49,21}$ stapfunctieberekeningen, zodat het praktisch haalbaar wordt om een botsing te vinden.

Inhoudsopgave

Lijst van symbolen

$x \parallel y$	de concatenatie van de binaire strings x en y
$x \wedge y$	de bitsgewijze EN van x en y
$x \vee y$	de bitsgewijze OF van x en y
$x \oplus y$	de bitsgewijze exclusieve OF van x en y
$\neg x$	de bitsgewijze NIET van x
$x \lll s$	de rotatie van de binaire string x naar links over s posities
$x \ggg s$	de rotatie van de binaire string x naar rechts over s posities
$x + y$	optelling van x en y modulo 2^{32} (notatie in de tekst)
$x \boxplus y$	optelling van x en y modulo 2^{32} (notatie in figuren)
$x[i]$	bitselectie: $(x \wedge 2^i) \neq 0$ geeft 1, $(x \wedge 2^i) \equiv 0$ geeft 0
$x^{[t]}$	de lengte in bits van x wordt expliciet aangegeven als t

Lijst van figuren

Lijst van tabellen

Hoofdstuk 1

Inleiding

1.1 Situering en toepassingsdomein

In de laatste decennia zijn computers en computernetwerken een essentieel deel geworden van het dagelijkse leven. Een groot aantal toepassingen zoals communicatie, handel, gezondheidszorg en financiën zijn ondenkbaar geworden zonder deze systemen. Deze trend zal zich ongetwijfeld verder doorzetten: het aantal computers en netwerken zal toenemen, computers zullen kleiner worden en zullen meer en meer geïntegreerd worden in het dagelijkse leven.

Historisch gezien was er voor militaire toepassingen vooral aandacht voor confidentialiteit van gegevens. Door de informatisering van de financiële sector werd later ook integriteit van data belangrijk. Recent is er ook een toenemende vraag naar de privacy van individuen. Om onder andere aan deze technologische uitdagingen tegemoet te komen, worden in het domein van de cryptografie technieken ontwikkeld en onderzocht.

Een uitgebreid overzicht van technieken in de cryptografie kan in 'Handbook of Applied Cryptography' [29] teruggevonden worden. Van sommige technieken wordt de veiligheid aangetoond via informatietheorie of complexiteitstheorie, bij anderen is dit niet mogelijk. In dat geval wordt cryptanalyse gebruikt om de zwakheden van algoritmes aan te tonen.

Hashfuncties zijn één van de bouwblokken uit de cryptografie. Zoals beschreven in [35], zijn dit functies h die een invoer m van een willekeurige lengte op deterministische manier omzetten in een uitvoer $h(m) \in \mathcal{H}(m)$ met een vaste lengte van n bits. Hierbij is het belangrijk dat aan een aantal veiligheidsvoorwaarden voldaan zijn.

Zo moet het onhaalbaar zijn om een (tweede) invers beeld te vinden of een botsing. Bij een invers beeld word gezocht naar een invoerwaarde m gegeven $Y \in \mathcal{H}(m)$, waarbij $Y = h(m)$. Voor een tweede invers beeld, wordt naar een m' gezocht gegeven m en $h(m)$, waarvoor $m \neq m'$ en $h(m) = h(m')$. Een botsing bestaat uit twee invoerwaarden m, m' met $m \neq m'$ waarvoor $h(m) = h(m')$.

Deze hashfuncties worden onder andere gebruikt bij beveiligde websites [18] en elektronische handtekeningen [38]. Ze vormen op die manier een cruciaal bouwblok om de veiligheid van onder andere e-commerce te waarborgen.

1.2 Recente aanvallen

In tegenstelling tot bij blokcijfers (zoals AES [8]), is er niet veel bekend over de veilige constructie van hashfuncties [23]. Recente aanvallen van X. Wang op onder andere de hashfuncties MD4 [42], MD5 [44], RIPEMD [42] en SHA-1 [43] hebben een impuls gegeven aan onderzoek naar hashfuncties. Voor MD5 volstaat enkele seconden rekenwerk op één pc om botsingen te vinden [40]. Ook voor SHA-1 wordt het zoeken naar een botsing praktisch haalbaar, zij het met veel meer rekenkracht [12].

1.3 Doelstellingen

De hashfunctie HAS-V [34] heeft een structuur die gelijkaardig is aan de meest gebruikte hashfuncties, zoals onder andere MD5, SHA-1 en RIPEMD-160. Er is slechts één paper bekend die zich toelegt op de cryptanalyse van deze hashfunctie [28]. In die paper worden er echter geen resultaten gegeven die gebruik maken van de recente aanvallen op hashfuncties.

Bij de aanvallen op onder andere MD5 en SHA-1 wordt gezocht naar twee verschillende berichten m, m' die slechts in een beperkt aantal bits verschillen. Deze verschillen worden beschreven in een 'differentieel pad' dat via verschillende technieken kan bekomen worden.

Het doel van deze masterproef is de toepassing en de uitbreiding van recente technieken ontwikkeld voor onder andere MD5, SHA-0 en SHA-1 voor de cryptanalyse van HAS-V. Hierbij wordt de klemtoon gelegd op het automatiseren van deze technieken, met het oog op uitbreidbaarheid naar andere hashfuncties. Zo kan een beter inzicht verworven worden in de veiligheid van algemene constructies in hashfuncties. Deze kennis is belangrijk voor de open competitie van NIST voor een nieuwe hashfunctiestandaard [33].

1.4 Structuur van de tekst

Een algemene inleiding wordt gegeven in Hoofdstuk 1. Hashfuncties worden gesitueerd in het domein van de cryptografie en een kort overzicht wordt gegeven van recente aanvallen op hashfuncties. Het verder onderzoeken van deze technieken is het doel van deze tekst.

Hoofdstuk 2 verduidelijkt het begrip van hashfuncties. Algemene principes en constructiemethoden worden aangegeven, alsook een aantal veiligheidsvoorwaarden waaraan een hashfunctie verwacht wordt te voldoen. Hier worden eveneens praktische aanvallen beschreven die gebruik maken van botsingen voor hashfuncties. In Hoofdstuk 3 wordt de cryptografische hashfunctie HAS-V beschreven. Van deze hashfunctie wordt een vereenvoudigde variant voorgesteld, alsook een cyclische formulering die nuttig is voor verdere cryptanalyse.

Een beschrijving op hoog niveau van de technieken van X. Wang wordt in Hoofdstuk 4 gegeven. Er wordt verduidelijkt wat lokale botsingen zijn in Hoofdstuk 5. De technieken van F. Chabaud en A. Joux worden dan verder uitgebreid en toegepast op de vereenvoudigde HAS-V. L-karakteristieken worden verduidelijkt in Hoofdstuk 6. Er wordt een overzicht gegeven van de L-karakteristieken met het laagst gevonden gewicht voor de vereenvoudigde HAS-V in Tabel 6.1. In Hoofdstukken 7 en 8 worden NL-karakteristieken gebruikt. Hierbij worden zowel methoden

van M. Stevens als van C. De Cannière en C. Rechberger toegepast om te zoeken naar deze NL-karakteristieken. De methoden zelf worden verduidelijkt, verder uitgebreid en toegepast.

De resultaten van deze masterproef worden samengevat in Hoofdstuk 9. Hier worden ook suggesties gegeven voor verder onderzoek.

Hoofdstuk 2

Cryptografische hashfuncties

2.1 Inleiding

In dit hoofdstuk wordt uitgelegd wat cryptografische hashfuncties zijn, en aan welke veiligheidsvoorwaarden ze moeten voldoen. Verder wordt de Merkle-Damgård constructie voor hashfuncties toegelicht. Een meer uitvoerige beschrijving kan in [35] gevonden worden. De recente aanvallen zoeken naar een botsing. Er worden enkele praktische situaties geschetst waarbij de veiligheid van systemen in het gedrang wordt gebracht indien deze botsingen gevonden kunnen worden.

2.2 Definitie van een cryptografische hashfunctie

Hashfuncties zijn functies h die een invoer m van een willekeurige lengte op deterministische manier omzetten in een uitvoer $h(m) \in \mathcal{H}(m)$ met een vaste lengte van n bits. Hierbij moet er aan een aantal veiligheidsvoorwaarden voldaan zijn, die verder verduidelijkt worden. In deze tekst wordt met 'hashfunctie' steeds 'cryptografische hashfunctie' bedoeld, dus hashfuncties die door ontwerp verondersteld worden aan deze voorwaarden te voldoen. Indien dit niet het geval is, wordt de de cryptografische hashfunctie 'gebroken' genoemd.

Een eerste vereiste bij hashfuncties is niet-inverteerbaarheid. Zo moet het onhaalbaar zijn, gegeven $Y \in \mathcal{H}(m)$ om een invers beeld te vinden, dit is een bericht m waarvoor $Y = h(m)$. Voor een tweede invers beeld, wordt naar een m' gezocht, gegeven m en $h(m)$, waarvoor $m \neq m'$ en $h(m) = h(m')$. Gezien de lengte van de uitvoer n bits is, wordt verwacht dat na $\mathcal{O}(2^n)$ hashfunctie-evaluaties op willekeurige berichten R, een bericht $r \in R$ gevonden wordt waarvoor $h(r) = Y$.

Er zal altijd een generische aanval met $\mathcal{O}(2^n)$ hashfunctie-evaluaties bestaan om een invers beeld of een tweede invers beeld te vinden. Door n voldoende groot te kiezen, wordt deze aanval gezien de toepassing onhaalbaar. Een aanvaller mag echter niet significant minder dan $\mathcal{O}(2^n)$ hashfunctie-evaluaties uitvoeren om een invers beeld of een tweede invers beeld te vinden.

Een tweede vereiste is dat het botsingsbestendige functies zijn, dit zijn functies waarvoor het

onhaalbaar is om m, m' te vinden met $m \neq m'$ waarvoor $h(m) = h(m')$. Gezien de uitvoer een lengte van n bits heeft, wordt verwacht dat na $\mathcal{O}(2^{n/2})$ hashfunctie-evaluaties op willekeurige berichten R, twee berichten $r, r' \in R$ met $r \neq r'$ gevonden worden waarvoor $h(r) = h(r')$. Deze aanval, ook wel de verjaardagsaanval [35] genoemd, steunt op de verjaardagsparadox. Deze paradox stelt dat in een groep van 23 personen, er meer dan 50% kans is dat minstens twee personen op dezelfde dag jarig zijn. Hierbij wordt verondersteld dat er 365 mogelijke verjaardagen zijn, die elk met een gelijke kans voorkomen.

De berekening verloopt als volgt [16, p. 33]. De complementaire kans $\bar{P}(n)$ dat alle n personen op een verschillende dag jarig zijn, is 0 indien $n > 365$ door het duiventilprincipe. Indien $n \leq 365$ wordt deze kans gegeven door:

$$\bar{P}(n) = \frac{364}{365} \cdot \frac{363}{365} \cdot \ldots \cdot \frac{365 - n + 1}{365} = \frac{365!}{365^n (365 - n)!} \tag{2.1}$$

Indien de verjaardag van één persoon vast ligt, is de kans immers 364/365 dat een tweede persoon niet op dezelfde dag jarig is. Een derde persoon verjaart met kans 363/365 niet op dezelfde dag als de twee eersten, enzovoort. De kans dat minstens twee personen uit een groep van n personen op dezelfde dag jarig zijn, wordt dan gegeven door $P(n) = 1 - \bar{P}(n)$. Voor $n \geq 23$ is $P(n)$ groter dan 50%. De waarde van $P(23)$ is $50,7\%$.

Er zal altijd een generische aanval met $\mathcal{O}(2^{n/2})$ hashfunctie-evaluaties bestaan om een botsing te vinden. Door n voldoende groot te kiezen, wordt deze aanval gezien de toepassing onhaalbaar. Een aanvaller mag echter niet significant minder dan $\mathcal{O}(2^{n/2})$ hashfunctie-evaluaties uitvoeren om een botsing te vinden.

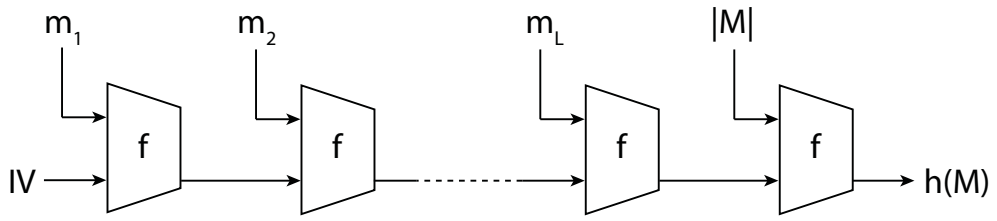
2.3 Merkle-Damgård constructie

Gegeven een compressiefunctie met een vaste invoerlengte $f : \{0,1\}^{b+n} \to \{0,1\}^n$, wordt een hashfunctie met een willekeurige invoerlengte $h : \{0,1\}^* \to \{0,1\}^n$ bekomen worden via de Merkle-Damgård constructie. Deze werd onafhankelijk beschreven door R. Merkle [30] en I. Damgård [9].

Hierbij wordt het bericht $M \in \{0,1\}^*$ aangevuld tot een veelvoud van de bloklengte b in bits, door het bericht te laten volgen door de bit '1', en eventueel een aantal keren de bit '0'. Dus: $m_1 \parallel \ldots \parallel m_L \leftarrow M \parallel 10 \ldots 0$, met $|m_i| = b$. Voor ieder bericht wordt een vaste initiële waarde IV gebruikt, eigen aan de hashfunctie. Dan wordt $h(M)$ als volgt bekomen (zie ook Figuur 2.1):

$$\begin{aligned} H_0 &\leftarrow IV \\ H_i &\leftarrow f(m_i \parallel H_{i-1}), \qquad i = 1, 2, \ldots, L \\ h(X) &\leftarrow f(|M| \parallel h_L) \end{aligned} \tag{2.2}$$

Hashfuncties die volgens deze constructie opgebouwd zijn, worden geïtereerde hashfuncties genoemd. Het toevoegen van de berichtlengte in deze constructie wordt 'MD-strengthening' genoemd. Door het toevoegen van deze berichtlengte, kan voor de Merkle-Damgård constructie



Figuur 2.1: Merkle-Damgård constructie

bewezen worden, dat indien een botsing gevonden wordt voor de hashfunctie h, hieruit een botsing kan afgeleid worden voor de compressiefunctie f. De botsingsbestendigheid van h volgt dus bewijsbaar uit de botsingsbestendigheid van f [9].

2.4 Recente aanvallen zoeken naar een botsing

De recente aanvallen zoeken naar botsingen voor de hashfunctie. Bij deze recente aanvallen wordt er geen uitspraak gedaan over de haalbaarheid van het berekenen van een (tweede) invers beeld, met uitzondering van [45].

Een gevolg van de Merkle-Damgård constructie uit Paragraaf 2.3 is dat iedere botsing die zich voordoet vooraleer het blok met de berichtlengte wordt verwerkt, gebruikt kan worden om nieuwe botsingen te bekomen. Dit gebeurt door het toevoegen van zelfgekozen, maar identieke blokken aan het bericht. Indien een botsing voor een willekeurige IV kan gerealiseerd worden, is het ook mogelijk om de blokken in het begin van het bericht te kiezen, vóór de botsing optreedt. Bij MD5 is men er zelfs in geslaagd om botsingen te realiseren voor berichten met een verschillende IV [40]. Zo is het mogelijk om aan twee zelfgekozen berichten van gelijke lengte, één of meer extra blokken toe te voegen om een botsing te bekomen.

Dat de aanvaller beide berichtwoorden die leiden tot een botsing zelf kan bepalen, lijkt misschien onrealistisch in de praktijk. Dit is des te meer het geval gezien er bij de recente aanvallen vaak weinig controle is over de inhoud van beide berichtwoorden. Praktische omstandigheden kunnen echter wel gevonden worden.

2.4.1 Betekenishebbende botsingen

Voor bepaalde bestandsformaten, is het door de Merkle-Damgård constructie mogelijk om twee botsende bestanden te genereren met een zinvolle betekenis voor de eindgebruiker. Dit gaat onder andere voor PostScript, PDF, TIFF, MS Word 97 en uitvoerbare bestanden [17]. Hierbij wordt een botsing vaak voorafgegaan en/of gevolgd door andere, vaste blokken. Indien het bestandsformaat een controle-instructie bevat voor conditionele uitvoering (if-then-else), zoals bij PostScript, is een triviale aanval altijd mogelijk. Voor andere formaten zonder programmeerinstructies is een creatievere aanpak nodig, zoals het gebruik van verschillende kleurenruimten bij PDF en verschillende offsets bij TIFF. Een eindgebruiker die deze bestanden opent met de programma's die hier normaal voor bestemd zijn, zal geen verschil merken. Door de bytes van deze bestanden te bestuderen, kan een expert deze bestanden echter wel vrij snel

ontmaskeren. Indien een gebruiker overtuigd kan worden om een cryptografische handtekening aan te maken voor één van deze bestanden, wordt automatisch een handtekening bekomen voor het andere bestand, waarvan de inhoud volledig door de aanvaller gekozen werd.

2.4.2 De aanval voor APOP

Indien de aanvaller de laatste bytes van de botsing zelf kan bepalen, is het mogelijk om bij APOP het paswoord van de gebruiker iteratief karakter per karakter te achterhalen [24]. APOP is een protocol dat gebruikt kan worden voor entiteitsauthenticatie bij het 'Post Office Protocol (POP)', dat wereldwijd gebruikt wordt voor het ophalen van e-mail. Bij APOP wordt een 'uitdaging' (challenge) naar de gebruiker gestuurd, die daaraan zijn eigen paswoord toevoegt en de hashwaarde berekent, om als antwoord (response) terug te sturen.

De aanvaller genereert een botsingspaar zodanig, dat een gok voor de eerste karakters van een paswoord horen tot één berichtblok, en de rest tot het volgende. De gebruiker krijgt het begin van beide berichten van het botsingspaar als uitdagingen, voegt zijn eigen paswoord telkens toe en stuurt de hashwaarden terug. Het al dan niet voordoen van een botsing geeft dan aan of de aanvaller een goede gok gemaakt had. Zo kan de aanvaller karakter per karakter achterhalen.

Door enkel gebruik te maken van de differentiële paden van X. Wang [44] en M. Stevens [40] voor MD5, kunnen slechts de eerste drie karakters van het paswoord achterhaald worden. De volgende karakters kunnen offline bekomen worden door exhaustief te zoeken. Het is niet mogelijk om met deze differentiële paden een botsingspaar te construeren waarbij enkel ASCII-karakters worden gebruikt. Hoewel dit volgens de standaard verplicht is voor de uitdagingen, wordt dit door veel e-mailprogramma's niet gecontroleerd [24].

2.4.3 Botsende X.509 certificaten met zelfgekozen velden voor de entiteitsinformatie

Indien het mogelijk is voor de aanvaller om aan twee willekeurige, niet-identieke berichten één of meer extra blokken toe te voegen om een botsing te bekomen, is het mogelijk om certificaten te vervalsen [39]. Een voorbeeld wordt gegeven voor X.509 certificaten, die onder andere worden gebruikt bij beveiligde websites [18]. Stel dat een aanvaller twee certificaten wil opstellen, één voor zijn eigen bedrijf en één voor een grote multinational. Aan de verschillende entiteitsinformatie in de certificaten, worden extra blokken toegevoegd om een botsing te bekomen. De willekeurig uitziende data die nodig is voor deze botsing, maakt deel uit van geldige publieke sleutels. De aanvaller kan dan vragen om het certificaat voor zijn eigen bedrijf cryptografisch te ondertekenen, waarbij men in feite alleen de hashwaarde ondertekent, om eveneens een geldig certificaat te bekomen voor de grote multinational.

Deze aanval kan gebruikt worden bij 'phishing' [13], waarbij er hier echter geen enkel teken is aan de browser dat iets niet in orde is.

Hoofdstuk 3

HAS-V

3.1 Inleiding

Nadat HAS-V, een Zuid-Koreaanse hashfunctie wordt beschreven, wordt een vereenvoudigde variant van HAS-V voorgesteld, waarop de cryptanalyse zich in volgende hoofdstukken toelegt. De HAS-V hashfunctie werd als standaard voorgesteld door Park et al. in SAC 2000 [34]. Eveneens wordt een cyclische formulering van de vereenvoudigde HAS-V voorgesteld, die nuttig is voor verdere cryptanalyse.

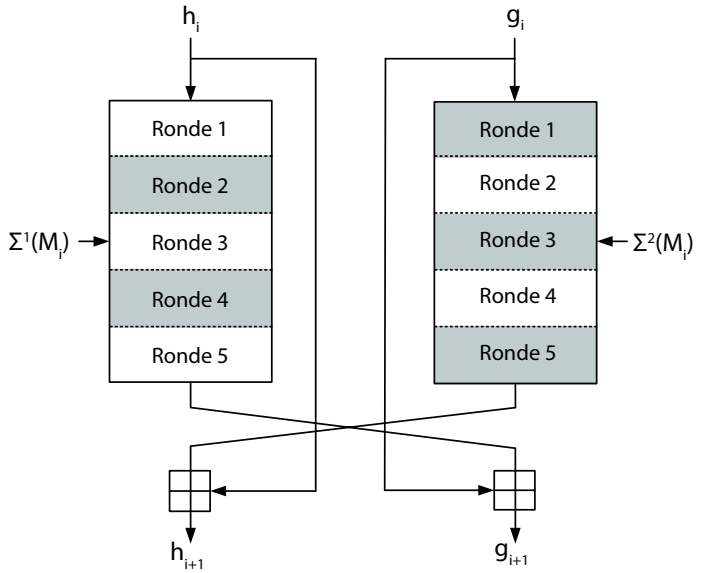
3.2 Beschrijving van HAS-V

HAS-V is een geïtereerde hashfunctie die berichten in blokken van 1024 bits omzet naar een uitvoer van een variabele lengte van 128, 160, 192, 224, 256, 288 of 320 bits. Deze hashfunctie is ontworpen om in KCDSA (Korea Certificate-based Digital Signature Algorithm [25]) gebruikt te worden, die een hashfunctie van variabele lengte ondersteunt. HAS-V werd voorgesteld als opvolger voor HAS-160, een gelijkaardige hashfunctie die echter enkel een uitvoerlengte van 160 bits ondersteunt.

Het bericht dat als invoer wordt gebruikt voor de hashfunctie, wordt eerst opgevuld tot de lengte in bits congruent is aan 952 mod 1024. Dit gebeurt door een enkele '1'-bit toe te voegen, en het nodige aantal '0'-bits. De resterende 72 bits om een veelvoud van 1024 te bekomen, bestaan uit de lengte van de hashwaarde in bytes en de lengte van de invoer in bits, modulo 2^{64}.

De hashfunctie bestaat uit twee parallelle stromen met elk vijf ronden. De berichtexpansie voor elke ronde is een permutatie van 20 uitgebreide berichtwoorden w_i. Eerst wordt een 1024-bit berichtblok M_i opgesplitst in twee 512-bit delen $\underline{M_i}$ en $\overline{M_i}$. Voor de oneven ronden in de linkerstroom en de even ronden in de rechterstroom wordt $\underline{M_i}$ gebruikt, voor de andere ronden wordt $\overline{M_i}$ gebruikt, zoals aangegeven in Figuur 3.1. De 20 uitgebreide berichtwoorden w_i bestaan uit de 16 berichtwoorden m_i aangevuld met 4 woorden afgeleid van de berichtwoorden (XOR-woorden), zoals aangegeven in Tabel 3.1.

Hoe de geëxpandeerde berichtwoorden W_t bekomen worden uit de uitgebreide berichtwoorden



Figuur 3.1: De volledige HAS-V compressiefunctie

	w_{16}	w_{17}	w_{18}	w_{19}
Ronde 1	$w_0 \oplus w_1 \oplus w_2 \oplus w_3$	$w_4 \oplus w_5 \oplus w_6 \oplus w_7$	$w_8 \oplus w_9 \oplus w_{10} \oplus w_{11}$	$w_{12} \oplus w_{13} \oplus w_{14} \oplus w_{15}$
Ronde 2	$w_3 \oplus w_6 \oplus w_9 \oplus w_{12}$	$w_{15} \oplus w_2 \oplus w_5 \oplus w_8$	$w_{11} \oplus w_{14} \oplus w_1 \oplus w_4$	$w_7 \oplus w_{10} \oplus w_{13} \oplus w_0$
Ronde 3	$w_{12} \oplus w_5 \oplus w_{14} \oplus w_7$	$w_0 \oplus w_9 \oplus w_2 \oplus w_{11}$	$w_4 \oplus w_{13} \oplus w_6 \oplus w_{15}$	$w_8 \oplus w_1 \oplus w_{10} \oplus w_3$
Ronde 4	$w_7 \oplus w_2 \oplus w_{13} \oplus w_8$	$w_3 \oplus w_{14} \oplus w_9 \oplus w_4$	$w_{15} \oplus w_{10} \oplus w_5 \oplus w_0$	$w_{11} \oplus w_6 \oplus w_1 \oplus w_{12}$
Ronde 5	$w_{15} \oplus w_9 \oplus w_5 \oplus w_3$	$w_{12} \oplus w_8 \oplus w_6 \oplus w_2$	$w_{13} \oplus w_{11} \oplus w_7 \oplus w_1$	$w_{14} \oplus w_{10} \oplus w_4 \oplus w_0$

Tabel 3.1: Berekening van de XOR-woorden voor HAS-V

stap t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Ronde 1	18	0	1	2	3	19	4	5	6	7	16	8	9	10	11	17	12	13	14	15
Ronde 2	18	3	6	9	12	19	15	2	5	8	16	11	14	1	4	17	7	10	13	0
Ronde 3	18	12	5	14	7	19	0	9	2	11	16	4	13	6	15	17	8	1	10	3
Ronde 4	18	7	2	13	8	19	3	14	9	4	16	15	10	5	0	17	11	6	1	12
Ronde 5	18	15	9	5	3	19	12	8	6	2	16	13	11	7	1	17	14	10	4	0

Tabel 3.2: De berichtexpansie voor HAS-V

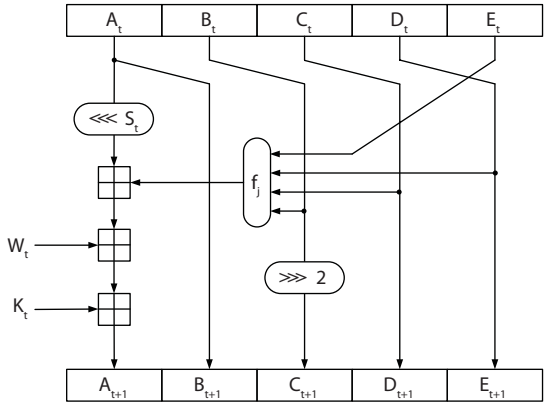
w_i, wordt in Tabel 3.2 aangegeven.

De berekening van de nieuwe interne toestand, ook stapfunctie genoemd, start met een vaste initiële waarde IV van vijf 32-bit registers per stroom, die in vijf ronden van ieder 20 stappen wordt bijgewerkt. De IV-waarden worden in Tabel 3.3 gegeven.

Figuur 3.2 geeft onderstaande stapfunctie schematisch weer:

	IV links	IV rechts
A	0x67452301	0x8796A5B4
B	0xEFCDAB89	0x4B5A6978
C	0x98BADCFE	0x0F1E2D3C
D	0x10325476	0xA0B1C2D3
E	0xC3D2E1F0	0x68794E5F

Tabel 3.3: De IV-waarden voor HAS-V



Figuur 3.2: De HAS-V stapfunctie

$$
\begin{aligned}
A_{t+1} &\leftarrow (A_t \lll S_t) + f_j(B_t, C_t, D_t, E_t) + W_t + K_t \\
B_{t+1} &\leftarrow A_t \\
C_{t+1} &\leftarrow B_t \ggg 2 \\
D_{t+1} &\leftarrow C_t \\
E_{t+1} &\leftarrow D_t
\end{aligned}
\tag{3.1}
$$

Hierbij is f_j een booleaanse functie, verschillend voor iedere ronde. Deze f_j dient niet verward te worden met de compressiefunctie uit paragraaf 2.3. f_j wordt gebruikt voor de j-de lichtgekleurde ronde en de $(6-j)$-de donkergekleurde ronde in Figuur 3.1.

$$
\begin{aligned}
f_1(B,C,D,E) &\triangleq (B \wedge C) \oplus (\neg B \wedge D) \oplus (C \wedge E) \oplus (D \wedge E) \\
f_2(B,C,D,E) &\triangleq (B \wedge D) \oplus C \oplus D \\
f_3(B,C,D,E) &\triangleq (B \wedge C) \oplus (\neg B \wedge E) \oplus D \\
f_4(B,C,D,E) &\triangleq B \oplus (C \wedge D) \oplus E \\
f_5(B,C,D,E) &\triangleq (\neg B \wedge C) \oplus (B \wedge D) \oplus (C \wedge E) \oplus (D \wedge E)
\end{aligned}
\tag{3.2}
$$

	K_t links	K_t rechts
Ronde 1	0x00000000	0xA953FD4E
Ronde 2	0x8F1BBCDC	0x5A827999
Ronde 3	0x6ED9EBA1	0x00000000
Ronde 4	0x5A827999	0x8F1BBCDC
Ronde 5	0xA953FD4E	0x6ED9EBA1

Tabel 3.4: Constante K_t voor HAS-V

stap t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
S_t	5	11	7	13	15	6	13	9	5	11	7	12	8	15	13	8	15	6	7	14

Tabel 3.5: Rotatiewaarde S_t voor zowel HAS-V als de vereenvoudigde HAS-V

Bij iedere stap wordt een constante K_t toegevoegd, die verschillend is voor iedere ronde, gegeven in Tabel 3.4.

De rotatiewaarde S_t is verschillend voor iedere stap van een ronde, en wordt gegeven in Tabel 3.5.

Stel dat de inhouden van de interne toestanden vóór de eerste stapfunctie gegeven worden door de variabelen A tot E voor de linkerstroom en F tot J voor de rechterstroom. De inhouden van de interne toestanden na de laatste stapfunctie worden AA tot EE voor de linkerstroom, en FF tot JJ voor de rechterstroom genoemd. De volgende feedforward-bewerking wordt dan toegepast:

$$A \leftarrow A + FF, \quad B \leftarrow B + GG, \quad C \leftarrow C + HH, \quad D \leftarrow D + II, \quad H \leftarrow H + JJ, \tag{3.3}$$

$$F \leftarrow F + AA, \quad G \leftarrow G + BB, \quad H \leftarrow H + CC, \quad I \leftarrow I + DD, \quad J \leftarrow J + EE \tag{3.4}$$

In het geval van een 320-bit hashwaarde, wordt de uitvoer gegeven als de concatenatie van de tien interne toestanden, dus $A \parallel B \parallel C \parallel D \parallel E \parallel F \parallel G \parallel H \parallel I \parallel J$. Indien een hashwaarde kleiner dan 320 bits gewenst is, wordt de uiteindelijke hashwaarde afgeleid uit deze tien interne toestanden via een korte berekening die "output tailoring" genoemd wordt. Deze kortere hashwaarde wordt genoteerd als $O_0 \parallel O_1 \parallel \ldots \parallel O_t$, met $t \in \{3, 4, 5, 6, 7, 8\}$. Bij de bespreking van deze gevallen, wordt de notatie $X^{[t]}$ gebruikt om expliciet aan te geven dat X een lengte heeft van t bits.

- 128-bit hashwaarde: De 32-bit interne toestanden E en J worden als volgt gesplitst:

$$E_1^{[16]} \parallel E_0^{[16]} \leftarrow E, \quad J_1^{[16]} \parallel J_0^{[16]} \leftarrow J \tag{3.5}$$

  O_i worden als volgt berekend:

$$O_0 \leftarrow A + F + E_1^{[16]}, \quad O_1 \leftarrow B + G + E_0^{[16]}, \tag{3.6}$$

$$O_2 \leftarrow C + H + J_1^{[16]}, \quad O_3 \leftarrow D + I + J_0^{[16]} \tag{3.7}$$

- 160-bit hashwaarde: O_i worden als volgt berekend:

$$O_0 \leftarrow A + F, \quad O_1 \leftarrow B + G, \quad O_2 \leftarrow C + H, \quad O_3 \leftarrow D + I, \quad O_4 \leftarrow E + J \tag{3.8}$$

- 192-bit hashwaarde: De 32-bit interne toestanden D, E, I en J worden als volgt gesplitst:

$$D_2^{[11]} \parallel D_1^{[11]} \parallel D_0^{[10]} \leftarrow D, \quad E_2^{[11]} \parallel E_1^{[11]} \parallel E_0^{[10]} \leftarrow E, \tag{3.9}$$

$$I_2^{[11]} \parallel I_1^{[11]} \parallel I_0^{[10]} \leftarrow I, \quad J_2^{[11]} \parallel J_1^{[11]} \parallel J_0^{[10]} \leftarrow J \tag{3.10}$$

  O_i worden als volgt berekend:

$$O_0 \leftarrow A + \left(J_2^{[11]} \parallel I_1^{[11]}\right), \quad O_1 \leftarrow B + \left(J_1^{[11]} \parallel I_0^{[10]}\right), \tag{3.11}$$

$$O_2 \leftarrow C + \left(J_0^{[10]} \parallel I_2^{[11]}\right), \quad O_3 \leftarrow F + \left(E_2^{[11]} \parallel D_1^{[11]}\right), \tag{3.12}$$

$$O_4 \leftarrow G + \left(E_1^{[11]} \parallel D_0^{[10]}\right), \quad O_5 \leftarrow H + \left(E_1^{[11]} \parallel D_2^{[11]}\right) \tag{3.13}$$

- 224-bit hashwaarde: De 32-bit interne toestanden E, I en J worden als volgt gesplitst:

$$E_2^{[11]} \parallel E_1^{[11]} \parallel E_0^{[10]} \leftarrow E, \quad I_3^{[8]} \parallel I_2^{[8]} \parallel I_1^{[8]} \parallel I_0^{[8]} \leftarrow I, \quad J_3^{[8]} \parallel J_2^{[8]} \parallel J_1^{[8]} \parallel J_0^{[8]} \leftarrow J \tag{3.14}$$

  O_i worden als volgt berekend:

$$O_0 \leftarrow A + \left(J_3^{[8]} \parallel I_2^{[8]}\right), \quad O_1 \leftarrow B + \left(J_2^{[8]} \parallel I_1^{[8]}\right), \tag{3.15}$$

$$O_2 \leftarrow C + \left(J_1^{[8]} \parallel I_0^{[8]}\right), \quad O_3 \leftarrow D + \left(J_0^{[8]} \parallel I_3^{[8]}\right), \tag{3.16}$$

$$O_4 \leftarrow F + E_2^{[11]}, \quad O_5 \leftarrow G + E_1^{[11]}, \quad O_6 \leftarrow H + E_0^{[10]} \tag{3.17}$$

- 256-bit hashwaarde: De 32-bit interne toestanden E en J worden als volgt gesplitst:

$$E_3^{[8]} \parallel E_2^{[8]} \parallel E_1^{[8]} \parallel E_0^{[8]} \leftarrow E, \quad J_3^{[8]} \parallel J_2^{[8]} \parallel J_1^{[8]} \parallel J_0^{[8]} \leftarrow J \tag{3.18}$$

  O_i worden als volgt berekend:

$$O_0 \leftarrow A + J_3^{[8]}, \quad O_1 \leftarrow B + J_2^{[8]}, \tag{3.19}$$

$$O_2 \leftarrow C + J_1^{[8]}, \quad O_3 \leftarrow D + J_0^{[8]}, \tag{3.20}$$

$$O_4 \leftarrow F + E_3^{[8]}, \quad O_5 \leftarrow G + E_2^{[8]}, \tag{3.21}$$

$$O_6 \leftarrow H + E_1^{[8]}, \quad O_7 \leftarrow I + E_0^{[8]} \tag{3.22}$$

- 288-bit hashwaarde: De 32-bit interne toestand J wordt als volgt gesplitst:

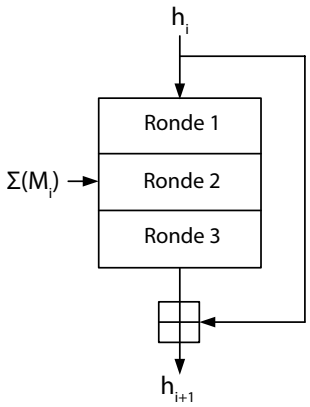
$$J_4^{[7]} \parallel J_3^{[7]} \parallel J_2^{[6]} \parallel J_1^{[6]} \parallel J_0^{[6]} \leftarrow J \tag{3.23}$$

  O_i worden als volgt berekend:

$$O_0 \leftarrow A + J_4^{[7]}, \quad O_1 \leftarrow B + J_3^{[7]}, \tag{3.24}$$

$$O_2 \leftarrow C + J_2^{[6]}, \quad O_3 \leftarrow D + J_1^{[6]}, \quad O_4 \leftarrow E + J_0^{[6]}, \tag{3.25}$$

$$O_5 \leftarrow F, \quad O_6 \leftarrow G, \quad O_7 \leftarrow H, \quad O_8 \leftarrow I \tag{3.26}$$



Figuur 3.3: De vereenvoudigde HAS-V compressiefunctie

	w_{16}	w_{17}	w_{18}	w_{19}
Ronde 1	$w_0 \oplus w_1 \oplus w_2 \oplus w_3$	$w_4 \oplus w_5 \oplus w_6 \oplus w_7$	$w_8 \oplus w_9 \oplus w_{10} \oplus w_{11}$	$w_{12} \oplus w_{13} \oplus w_{14} \oplus w_{15}$
Ronde 2	$w_{12} \oplus w_5 \oplus w_{14} \oplus w_7$	$w_0 \oplus w_9 \oplus w_2 \oplus w_{11}$	$w_4 \oplus w_{13} \oplus w_6 \oplus w_{15}$	$w_8 \oplus w_1 \oplus w_{10} \oplus w_3$
Ronde 3	$w_{15} \oplus w_9 \oplus w_5 \oplus w_3$	$w_{12} \oplus w_8 \oplus w_6 \oplus w_2$	$w_{13} \oplus w_{11} \oplus w_7 \oplus w_1$	$w_{14} \oplus w_{10} \oplus w_4 \oplus w_0$

Tabel 3.6: Berekening van de XOR-woorden voor de vereenvoudigde HAS-V

3.3 Beschrijving van een vereenvoudigde HAS-V

Een vereenvoudigde versie van HAS-V gebruikt 512-bit blokken in plaats van 1024-bit blokken. De rechterstroom en de donkergekleurde ronden in de linkerstroom in Figuur 3.1 worden weggelaten, om zo Figuur 3.3 te bekomen. Om misverstanden te voorkomen over de exacte manier waarop HAS-V aangepast wordt, volgt hier een korte beschrijving van de vereenvoudigde versie.

Het bericht dat als invoer wordt gebruikt voor de hashfunctie, wordt eerst opgevuld tot de lengte in bits congruent is aan 448 mod 512. Dit gebeurt door een enkele '1'-bit toe te voegen, en het nodige aantal '0'-bits. De resterende 72 bits om een veelvoud van 512 te bekomen, bestaan uit de lengte van de invoer in bits, modulo 2^{64}.

De berichtexpansie voor elke ronde is een permutatie van 20 uitgebreide berichtwoorden w_i. De 20 uitgebreide berichtwoorden w_i bestaan uit de 16 berichtwoorden m_i van het 512-bit berichtblok, aangevuld met 4 woorden afgeleid van de berichtwoorden (XOR-woorden), zoals aangegeven in Tabel 3.6.

Hoe de geëxpandeerde berichtwoorden W_t bekomen worden uit de uitgebreide berichtwoorden w_i, wordt in Tabel 3.7 aangegeven.

De berekening van de nieuwe interne toestand, ook stapfunctie genoemd, start met een vaste initiële waarde IV van vijf 32-bit registers, die in vijf ronden van ieder 20 stappen worden bijgewerkt. De IV-waarden worden in Tabel 3.8 gegeven.

De stapfunctie is dezelfde als voor HAS-V, zie Figuur 3.2. Hierbij is f_j een booleaanse functie,

stap t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Ronde 1	18	0	1	2	3	19	4	5	6	7	16	8	9	10	11	17	12	13	14	15
Ronde 2	18	12	5	14	7	19	0	9	2	11	16	4	13	6	15	17	8	1	10	3
Ronde 3	18	15	9	5	3	19	12	8	6	2	16	13	11	7	1	17	14	10	4	0

Tabel 3.7: De berichtexpansie voor de vereenvoudigde HAS-V

	IV
A	0x67452301
B	0xEFCDAB89
C	0x98BADCFE
D	0x10325476
E	0xC3D2E1F0

Tabel 3.8: De IV-waarden voor de vereenvoudigde HAS-V

	K_t
Ronde 1	0x00000000
Ronde 2	0x6ED9EBA1
Ronde 3	0xA953FD4E

Tabel 3.9: Constante K_t voor de vereenvoudigde HAS-V

verschillend voor iedere ronde. f_j wordt gebruikt voor de j-de ronde in Figuur 3.3.

$$\begin{aligned} f_1(B,C,D,E) &\triangleq (B \wedge C) \oplus (\neg B \wedge D) \oplus (C \wedge E) \oplus (D \wedge E) \\ f_2(B,C,D,E) &\triangleq (B \wedge C) \oplus (\neg B \wedge E) \oplus D \\ f_3(B,C,D,E) &\triangleq (\neg B \wedge C) \oplus (B \wedge D) \oplus (C \wedge E) \oplus (D \wedge E) \end{aligned} \tag{3.27}$$

Bij iedere stap wordt een constante K_t toegevoegd, die verschillend is voor iedere ronde, gegeven in Tabel 3.9.

De rotatiewaarde S_t is verschillend voor iedere stap van een ronde, en wordt gegeven in Tabel 3.5. Deze is dezelfde als voor HAS-V.

Stel dat de inhouden van de interne toestanden vóór de eerste stapfunctie gegeven worden door de variabelen A tot E. De inhouden van de interne toestanden na de laatste stapfunctie worden AA tot EE genoemd. De volgende feedforward-bewerking wordt dan toegepast:

$$A \leftarrow A + AA, \quad B \leftarrow B + BB, \quad C \leftarrow C + CC, \quad D \leftarrow D + DD, \quad H \leftarrow H + HH \tag{3.28}$$

De uitvoer is een 160-bit hashwaarde, gegeven door de concatenatie van de vijf interne toestanden: $A \parallel B \parallel C \parallel D \parallel E$. Een variabele uitvoerlengte wordt niet ondersteund.

3.4 Cyclische formulering van de vereenvoudigde HAS-V

Bij de stapfunctie van HAS-V worden vijf interne variabelen A_t, B_t, C_t, D_i en E_t bekomen uit de vijf vorige interne variabelen, A_{t-1}, B_{t-1}, C_{t-1}, D_{t-1} en E_{t-1}. Gezien $B_t \equiv A_{t-1}$, $C_t \equiv A_{t-2} \ggg 2$, $D_t \equiv A_{t-3} \ggg 2$ en $E_t \equiv A_{t-4} \ggg 2$, volstaat het om enkel A_t bij te houden bij het berekenen van de stapfunctie. Deze A_t, uitgebreid met waarden van de IV wordt Q_t genoemd. Dit leidt tot een alternatieve formulering van de vereenvoudigde HAS-V compressiefunctie voor $t = 0, \ldots, 59$:

$$Q_{t+1} \leftarrow (Q_t \lll S_i) + f_j(Q_{t-1}, Q_{t-2} \ggg 2, Q_{t-3} \ggg 2, Q_{t-4} \ggg 2) + W_t + K_t \tag{3.29}$$

Q_t met $t = -4, \ldots 0$ worden uit de IV afgeleid:

$$(Q_{-4}, Q_{-3}, Q_{-2}, Q_{-1}, Q_0) \leftarrow (E \lll 2, D \lll 2, C \lll 2, B, A) \tag{3.30}$$

De uitvoer van de compressiefunctie komt dan overeen met:

$$\begin{aligned} &Q_0 + Q_{60} \parallel Q_{-1} + Q_{59} \parallel (Q_{-2} \ggg 2) + (Q_{58} \ggg 2) \parallel (Q_{-3} \ggg 2) + (Q_{57} \ggg 2) \\ &\quad \parallel (Q_{-4} \ggg 2) + (Q_{56} \ggg 2) \end{aligned} \tag{3.31}$$

Deze cyclische formulering van de hashfunctie zal handig van pas komen bij verdere cryptanalyse. Een gelijkaardige formulering waarbij de interne variabelen Q_t worden genoemd, werd voorgesteld in [19] voor MD5.

Opdat de besproken technieken eenvoudig uit te breiden zijn naar andere hashfuncties, werd ervoor gekozen om deze variabelen niet A_t te noemen zoals in [12]. Bij onder andere MD5 en RIPEMD-160 komen immers niet A_t maar B_t overeen met met de variabelen Q_t uit de cyclische formulering.

Hoofdstuk 4

Algemene technieken van Wang

4.1 Inleiding

De algemene principes van de recente aanvallen van X. Wang worden verduidelijkt. Deze dienen als leidraad voor de lezer bij de volgende hoofdstukken. Hoewel de principes zelf niet constructief zijn, worden in de volgende hoofdstukken methoden aangeboden om de cryptanalyse van hashfuncties te automatiseren. Deze zullen telkens concreet toegepast worden op de vereenvoudigde HAS-V, een hashfunctie die sterk gelijkt op varianten uit de MD4-familie. Hierdoor wordt de impact duidelijker van ontwerpbeslissingen op de veiligheid van hashfuncties, in het kader van de recent ontwikkelde aanvallen. Een uitgebreidere behandeling van deze technieken kan in [10] gevonden worden.

4.2 Een berichtverschil

Om een botsing te bekomen, moeten bepaalde bits in de berichtwoorden m, m' verschillend zijn. In een eerste stap moet bepaald worden waar deze berichtverschillen zich voordoen. Het aantal verschillen kan zeer klein zijn. Bij de botsingen voor MD5 zijn bij de twee 512-bit berichtblokken die verschillen bevatten, slechts drie bits per berichtblok verschillend [44]. De overblijvende bits kunnen niet volledig vrij gekozen worden. Zij moeten voldoen aan bepaalde voorwaarden, die ervoor zorgen dat de interne toestanden (Q_{t+1}, Q'_{t+1}) voor $0 \leq t < T$ het verder beschreven differentiële pad volgen. Indien T kleiner is dan het volledige aantal stappen, wordt gesproken van een variant van de hashfunctie met een gereduceerd aantal stappen. Het is de haalbaarheid van een aanval met dit differentiële pad dat zal bepalen of de berichtverschillen in deze stap goed gekozen werden.

Merk op dat er gebruik kan gemaakt worden van meer dan één berichtblok om een botsing voor de hashfunctie te construeren. Dit wordt onder andere gedaan in [44] voor MD5 en in [12] voor SHA-1. Voor deze hashfuncties worden daar twee blokken worden gebruikt om een botsingspaar te genereren. Een botsing kan ook voorafgegaan worden door een berichtblok, identiek voor beide berichten, dat zodanig gekozen is, dat een betere IV bekomen wordt om de botsing te realiseren, zoals bijvoorbeeld in [27].

4.3 Een differentieel pad

Gegeven een berichtverschil, zullen er ook verschillen optreden in de interne toestanden (Q_{t+1}, Q'_{t+1}). Deze verschillen kunnen beschreven worden als exclusieve OF-verschillen, zoals in Hoofdstukken 5 en 6 zal gedaan worden. In Hoofdstuk 7 worden additieve verschillen modulo 2^{32} gehanteerd, door gebruik te maken van de condities die in Tabel 7.1 weergegeven worden. In Hoofdstuk 8 zullen deze condities verder uitgebreid worden in Tabel 8.1.

Bij de aanvallen van X. Wang wordt er gebruik gemaakt van additieve verschillen modulo 2^{32}. Zoals verduidelijkt wordt in [10], is de methode van X. Wang geïnspireerd door de techniek van H. Dobbertin om een botsing voor de compressiefunctie van MD5 te bekomen [14]. Bij deze laatste methode werden eveneens additieve verschillen modulo 2^{32} gebruikt.

Een differentieel pad geeft aan hoe deze verschillen zich in de interne toestanden (Q_{t+1}, Q'_{t+1}) gevolgd kunnen worden, opdat een botsing voor de hashfunctie bekomen wordt. Een differentieel pad dient vooral weinig verschillen op te leggen aan de interne toestanden (Q_{t+1}, Q'_{t+1}). Indien de invoerbits bij de stapfunctie gelijk zijn voor beide berichten, zijn de uitvoerbits immers altijd gelijk. Een verschil aan de invoer doorgeven of opslorpen gebeurt meestal slechts met een bepaalde kans. Bij de aanval van X. Wang zal eerst een differentieel pad gezocht worden voor de interne toestanden (Q_{t+1}, Q'_{t+1}), en daarna pas een stel van berichtwoorden m, m'.

Het zoeken van een differentieel pad gebeurt gelijkaardig aan de aanval van F. Chabaud en A. Joux [5], in de zin dat er gezocht wordt naar een volledig bepaald differentieel pad. Dit staat in contrast met de methode van H. Dobbertin voor MD5 [14, 10], die minder restrictief is in het bepalen van de verschillen. Hoewel er ook gebruik gemaakt wordt van combinaties van lokale botsingen zoals Hoofdstukken 5 en 6, gebruikt de aanval van X. Wang geen benadering van de stapfunctie, maar de echte stapfunctie zoals in Hoofdstukken 7 en 8.

Bij de constructie van de differentiële paden van X. Wang worden drie belangrijke vrijheidsgraden gebruikt:

- Afhankelijk van de invoerverschillen van de booleaanse functies f_j, is er vaak meer dan één mogelijkheid voor de verschillen aan de uitvoer. Indien er geen invoerverschil is, zal er nooit een uitvoerverschil zijn. Is er echter wel een invoerverschil, dan is het mogelijk dat dit invoerverschil doorgegeven wordt, of opgeslorpt door de booleaanse functie f_j. Indien gerekend wordt met additieve verschillen modulo 2^{32}, kunnen er eveneens verschillende mogelijkheden zijn voor het teken van het verschil aan de uitvoer.
- Bij onder andere MD5 en RIPEMD-160 kunnen voor de rotatie maximaal vier verschillende mogelijkheden gekozen worden. Er zal echter vaak één verschil zijn dat een veel grotere kans heeft dan de anderen, zodat hier meestal geen rekening mee wordt gehouden. Bij onder andere SHA-0, SHA-1 en HAS-V introduceren de rotaties geen bijkomende vrijheid voor het differentiële pad.
- Gegeven een additief verschil modulo 2^{32}, kan hieruit een verschil afgeleid worden voor de individuele bits. Hierbij kan een additief verschil modulo 2^{32} in slechts één bit, als het ware 'uitgesmeerd' worden over een groot aantal bits bij het bepalen van een bitverschil. Hoe verder een bit wordt uitgesmeerd, hoe kleiner de kans dat deze situatie zich voordoet. In Hoofdstuk 7 zal een notatie geïntroduceerd voor deze verschillen, zodat een concreet voorbeeld van dit principe gegeven kan worden.

Het doel van deze differentiële paden is tweevoudig. Enerzijds wordt getracht om de zoekruimte te verkleinen, terwijl de kans op botsingen vergroot wordt. De zoekruimte wordt kleiner indien enkel botsingen met niet alleen een bepaald berichtverschil, maar ook met een bepaald verschil in de interne toestanden worden beschouwd. Indien de kans groter is dat het volgen van een differentieel pad tot een botsing leidt, wordt een botsing sneller gevonden. Hoe groter deze kans, hoe beter het differentiële pad. Anderzijds wordt het differentiële pad gebruikt om het zoekproces efficiënter te laten verlopen, gezien voorwaarden in het differentiële pad voor de eerste ronde minder impact hebben op de complexiteit van de aanval voor volgende ronden. Dit wordt verder verduidelijkt.

4.4 Een efficiënt zoekproces

De aanvaller heeft grotendeels controle over de interne toestanden in de eerste ronde, door het kiezen van gepaste berichtwoorden om deze interne toestanden te bekomen. Deze techniek wordt enkelvoudige berichtmodificatie ('single-message modification[1]') genoemd [44]. Met deze eenvoudige techniek zal in alle hoofdstukken die volgen rekening gehouden worden om de complexiteit van de aanval aan te geven.

Vanaf de tweede ronde worden eerdere berichtwoorden hergebruikt. Indien een berichtwoord hier wordt aangepast, zal dit voor een goede hashfunctie een grote impact hebben op alle interne toestanden na het eerste gebruik van dit berichtwoord. Verschillende gerelateerde technieken trachten dit probleem deels te verhelpen, waaronder meervoudige berichtmodificatie ('multi-message modification') [44], tunnels [22], neutrale bits [2] en boemerangaanvallen [21]. Deze technieken zijn allen verschillende uitdrukkingsvormen van het zelfde onderliggende basisidee, namelijk dat bepaalde bits in de interne toestanden (Q_{t+1}, Q'_{t+1}) aangepast kunnen worden, zonder een invloed te hebben op het differentiële pad.

- Bij meervoudige berichtmodificatie [44], worden niet één, maar een aantal berichtwoorden (m_i, m'_i) aangepast om een verkeerde bit in de interne toestand te corrigeren. De constructie van deze aanpassingsregels verloopt echter handmatig en lijkt veel inzicht te vergen, wat zich niet leent tot een automatische implementatie. Om deze reden zal deze techniek niet gebruikt worden bij verdere cryptanalyse.
- In [22] wordt het punt waarop de interne toestanden (Q_{t+1}, Q'_{t+1}) niet meer kunnen gekozen worden, maar slechts met een bepaalde kans voldoen aan de verdere voorwaarden, het verificatiepunt genoemd. Waar meervoudige berichtmodificatie gebruikt wordt tot aan dit verificatiepunt, is dat de plaats waar tunnels beginnen. Door een verzameling van mogelijkheden op te stellen op deze plaats, kan er als het ware een 'tunnel' opgesteld worden om tot het volgende verificatiepunt te raken.
- Een bit wordt neutraal genoemd in [2], indien het inverteren van deze bit niet verhindert dat het bericht voldoet aan het differentiële pad tot een bepaalde stap t. Een paar bits wordt neutraal genoemd, indien deze eigenschap geldt voor ieder van deze bits en voor het inverteren van beide bits tegelijkertijd. Een verzameling van bits wordt neutraal genoemd, indien deze eigenschap geldt voor iedere deelverzameling van bits. Indien ieder paar bits neutraal is, wordt de verzameling van bits 2-neutraal genoemd.

[1] In [44] wordt gesproken van 'single-message modification', maar in [42] van 'single-step modification' voor dezelfde techniek.

- De boemerangaanval die voorgesteld werd voor de cryptanalyse van blokcijfers door D. Wagner in [41], kan eveneens gebruikt worden voor hashfuncties. In [21] wordt beschreven dat deze techniek zeer gelijkaardig is aan neutrale bits. Een verschil is echter dat hier eveneens differentiële hulppaden ('auxiliary differential paths') worden gebruikt.

4.5 Besluit

In dit hoofdstuk wordt een algemeen raamwerk uitgewerkt, dat uitlegt welke stappen ondernomen worden bij de recente aanvallen van X. Wang. Uitgaande van een berichtverschil wordt een differentieel pad geconstrueerd. Dit differentiële pad maakt gebruik van de vrijheden in de booleaanse functies, de rotatie en de overdrachten van de optelling modulo 2^{32} om tot een botsing te komen met een hoge kans indien dit differentiële pad gevolgd wordt.

In een volgend stadium worden berichten gekozen die voldoen aan dit differentieel pad voor een bepaald aantal stappen. Via de techniek van enkelvoudige berichtmodificatie [44] kan gemakkelijk voldaan worden aan de voorwaarden in de eerste ronde. Hierbij wordt rekening gehouden bij de constructie van het differentiële pad. Via meervoudige berichtmodificatie [44] is er in beperktere mate ook controle over verdere stappen. Er wordt een overzicht gegeven van de technieken van meervoudige berichtmodificatie [44], tunnels [22], neutrale bits [2] en boemerangaanvallen [21]. Deze zijn vrij gelijkaardig, en steunen allen op hetzelfde basisprincipe dat bepaalde bits van de interne toestanden (Q_{t+1}, Q'_{t+1}) vrij gekozen kunnen worden zonder invloed te hebben op het differentiële pad.

Hoofdstuk 5

Lokale botsingen

5.1 Inleiding

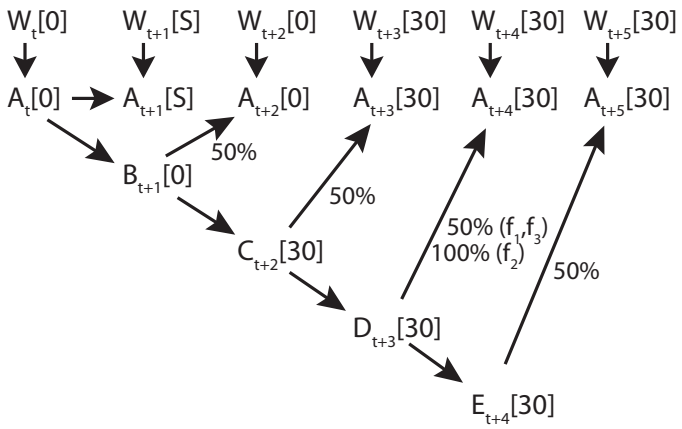
Een lokale botsing bestaat uit een perturbatie van een bepaalde bit in een geëxpandeerd berichtwoord $W_t[i]$, gevolgd door een of meerdere correcties. Deze correcties in volgende $W_{t'}[i]$ met $t' = \{t+1, t+2, \ldots\}$ doen de perturbatie teniet, zodat een botsing ontstaat. Lokale botsingen werden door F. Chabaud en A. Joux gebruikt bij de cryptanalyse van de oorspronkelijke versie van SHA (Secure Hash Algorithm), die SHA-0 genoemd wordt [5]. Dit is een hashfunctie die door het Amerikaanse NSA (National Security Agency) werd ontworpen. Dit hoofdstuk verduidelijkt hoe deze lokale botsingen eruit zien voor de vereenvoudigde HAS-V.

Meestal is er een combinatie van lokale botsingen nodig, om te voldoen aan de berichtexpansie. Voor de vereenvoudigde HAS-V worden nieuwe technieken geïntroduceerd om combinaties van lokale botsingen te zoeken. In plaats van slechts één vorm van een lokale botsing te gebruiken, worden hier alle mogelijke vormen van lokale botsingen tegelijkertijd in rekening gebracht. Er wordt een combinatie van 144 lokale botsingen gevonden in Tabel 5.3, die leidt tot een botsing voor de volledige compressiefunctie.

5.2 Lokale botsingen voor de vereenvoudigde HAS-V

In Figuur 5.1 wordt getoond hoe lokale botsingen er voor de vereenvoudigde HAS-V uitzien, naar analogie met de lokale botsingen voor SHA-0 van F. Chabaud en A. Joux [5]. Een lokale botsing voor HAS-V bestaat uit een perturbatie van één bit, gevolgd door een correctie van één tot vijf bits. Stel immers dat een verschil in een bepaalde bit van een geëxpandeerd berichtwoord $W_t[i]$ wordt geïntroduceerd. Dit verschil beweegt zich dan voort doorheen de interne toestanden van de compressiefunctie. Telkens hierdoor een bit van A zou veranderen, wordt een extra verschil aangelegd aan $W_{t'}[i']$ in deze ronden, om dit verschil teniet te doen. Zo ontstaat uiteindelijk een lokale botsing.

Het 'Strict Avalanche Criterion' voor booleaanse functies stelt dat bij het complementeren van invoerbit, de uitvoerbit in de helft van de gevallen ook verandert [35]. De booleaanse functies f_1 en f_3 van HAS-V voldoen aan deze eigenschap door ontwerp [34]. Bij f_2 geldt het 'Strict



Figuur 5.1: Lokale botsingen voor de vereenvoudigde HAS-V

Avalanche Criterion' voor alle invoerbits behalve voor de derde, die altijd een verschil aan de uitvoer geeft. Deze eigenschappen worden weergegeven door de percentages in Figuur 5.1. Op die manier zijn er voor de eerste en de derde ronde 2^4 lokale botsingen, en voor de tweede ronde 2^3.

5.3 C32SAT

In de rest van dit hoofdstuk wordt een wiskundig probleem geformuleerd als een uitdrukking in de programmeertaal C met variabelen met een onbekende waarde. Daarna wordt gebruik gemaakt van C32SAT [4], een programma dat vorig jaar ontwikkeld werd om (onder andere) een toekenning te geven aan deze vrije variabelen. Deze C-uitdrukking wordt via enkele tussenstappen omgezet naar een Boolean Satisfiability Problem (SAT), waarna een geldige toekenning wordt voor de vrije variabelen gezocht via een SAT-solver.

Door het gebruik van C32SAT is het voldoende om het wiskundig probleem te formuleren in een algemeen gebruikte programmeertaal als C. Het is niet nodig om zelf een zoekalgoritme te schrijven om een geldige toekenning te zoeken voor dit probleem, dat vaak minder performant is dan een geavanceerde SAT-solver. Het zoeken naar een geldige toekenning van een SAT-probleem hoort echter tot de 'NP hard'-complexiteitsklasse, wat wil zeggen dat er geen algoritme bekend is dat dit probleem in veeltermtijd oplost. Of zo'n algoritme al dan niet bestaat, is op dit moment een open probleem in de wiskunde. Het zal dus nodig zijn om de oplossingsruimte te beperken, zodat het zoeken naar een oplossing haalbaar blijft binnen een aanvaardbare tijd.

5.4 Combinaties van lokale botsingen die voldoen aan de berichtexpansie

De eerste stappen van de vereenvoudigde HAS-V worden beschouwd. Stel dat een perturbatie P_{18} wordt toegevoegd in de eerste stap. Indices i van perturbaties P_i komen overeen met deze

van de uitgebreide berichtwoorden w_i. Deze zal in de volgende stap aanleiding geven tot een correctie $P_{18} \lll 11$. In deze stap kan echter een nieuwe perturbatie P_0 worden toegevoegd, zodat het exclusieve OF-verschil in het uitgebreide berichtwoord in de tweede stap, $\nabla^{\oplus} W_0$, gelijk is aan $P_0 \oplus (P_{18} \lll 11)$.

In de derde stap kan weer een nieuwe perturbatie P_1 worden toegevoegd, en verschijnt door de perturbatie uit de vorige stap een correctie van $P_0 \lll 7$. De perturbatie die drie stappen geleden werd geïntroduceerd, kan nu verschijnen als P_{18} indien geen enkel verschil werd opgeslorpt door de booleaanse functie, of als 0 indien alle verschillen verder opgeslorpt. Om dit weer te geven in code, wordt een nieuwe dummyvariabele D_0 gebruikt, zodat alle mogelijke correctietermen geschreven kunnen worden als $D_0 \wedge P_{18}$. Deze dummyvariabele zal de bit 0 bevatten op plaatsen waar het verschil opgeslorpt moet worden door de booleaanse functie, en 1 op plaatsen waar het verschil aan de invoer van de booleaanse functie ook een verschil aan de uitvoer moet geven. Op die manier wordt $\nabla^{\oplus} W_1 = P_1 \oplus (P_0 \lll 7) \oplus (D_0 \wedge P_{18})$.

De woordverschillen $\nabla^{\oplus} W_0$ tot en met $\nabla^{\oplus} W_{15}$ zijn dezelfde in iedere ronde. De XOR-woorden verschillen voor iedere ronde, maar moeten wel elk gelijk zijn aan de exclusieve OF van vier berichtwoorden m_i in deze ronde, zoals aangegeven bij de beschrijving van de vereenvoudigde HAS-V compressiefunctie in Tabel 3.6. Het is niet mogelijk om een perturbatie in de allerlaatste stap aan te brengen. Dit komt omdat er altijd minstens één correctiewoord nodig is voor een lokale botsing.

De perturbatiewoorden P_i, hun afgeleide correctiewoorden, de exclusieve OF-verschillen in de berichtwoorden $\nabla^{\oplus} W_i$ en de dummyvariabelen D_i zijn de onbekende variabelen in een C-expressie die aan C32SAT wordt doorgegeven. Hierbij wordt geëist dat iedere perturbatie aanleiding geeft tot correcties, en dat in iedere ronde wordt voldaan aan de berichtexpansie.

5.5 Resultaten

De som van het aantal bits in alle perturbatiewoorden wordt opgeteld, dit zijn het aantal lokale botsingen dat nodig is om te voldoen aan de berichtexpansie. Hieraan wordt een maximum gegeven, dat telkens weer verlaagd wordt tot er geen oplossing meer bestaat zonder dat het aantal lokale botsingen dit maximum overschrijdt. Hierbij wordt verondersteld dat er minstens één lokale botsing is, anders zijn beide berichtwoorden m, m' gelijk aan elkaar en is er geen sprake van een botsing. Om een oplossing te vinden binnen aanvaardbare tijd, dient de oplossingsruimte beperkt te worden zoals verder aangegeven.

5.5.1 Perturbaties in iedere bit van een woord

Indien perturbaties in elke bit van een woord worden opgelegd, wordt een resultaat bekomen met 192 lokale botsingen, zoals aangegeven in Tabel 5.1. Alleen woorden waarvan alle 32 bits verschillend zijn voor beide berichten worden weergegeven.

Zoals beschreven in Paragraaf 4.4, is er bij het zoeken naar botsingen vooral controle over de verschillen $\nabla^{\oplus} W_t$ van de eerste ronde. Daarom wordt een variant gegeven in Tabel 5.2, die evenveel lokale botsingen bevat, maar waarvan zoveel mogelijk lokale botsingen zich in de eerste ronde bevinden.

stap i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Ronde 1																	12		14	
Ronde 2		12		14																
Ronde 3						19	12									17	14			

Tabel 5.1: Botsing voor de compressiefunctie door 192 lokale botsingen, voor weergegeven woorden zijn alle bits verschillend

stap i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Ronde 1	18			2		19			6		16				11	17		13		
Ronde 2									2	11			13	6						
Ronde 3									6	2		13	11							

Tabel 5.2: Botsing voor de compressiefunctie door 192 lokale botsingen, zoveel mogelijk in de eerste ronde, voor weergegeven woorden zijn alle bits verschillend

stap i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Ronde 1	18			2							16	8								
Ronde 2						19			2							17	8			
Ronde 3								8		2										

Tabel 5.3: Botsing voor de compressiefunctie door 144 lokale botsingen, weergegeven woorden betekenen een XOR-verschil van 0xAAAAAAAA

De woorden 0x00000000 en 0xFFFFFFFF zijn de twee enige woorden die onveranderd blijven indien een willekeurige rotatie wordt toegepast. Door het beperken van de zoekruimte voor de perturbatiewoorden en de correctiewoorden tot deze twee waarden, heeft de variabele rotatie naar links in iedere ronde met S_t geen effect meer. Hetzelfde geldt voor de vaste rotatie naar rechts met 2. Zonder deze rotaties, is er geen invloed tussen bits op verschillende bitposities in de woorden.

In plaats van één zoekprobleem naar de 32-bit perturbatiewoorden voor iedere ronde ontstaan zo 32 identieke problemen waarbij slechts één bit van de perturbatiewoorden gezocht wordt. Het volstaat om slechts één van die problemen, bijvoorbeeld voor de minst beduidende bit, op te lossen.

5.5.2 Perturbaties in iedere bit van een woord, of in de even of oneven bits

Indien ook perturbaties worden toegelaten in alleen de even of alleen de oneven bits van een woord, kan het aantal lokale botsingen tot 144 verminderd worden. In Tabel 5.3 betekenen weergegeven woorden een exclusieve OF-verschil van 0xAAAAAAAA, of 1010 . . . 10 in het binair.

Er bestaat geen variant die hetzelfde aantal lokale botsingen heeft, maar waarvan de meeste zich in de eerste ronde bevinden.

De woorden 0x00000000, 0x55555555, 0xAAAAAAAA en 0xFFFFFFFF vormen een verzameling

waarin alle even en oneven rotaties inwendig zijn. Door het beperken van de zoekruimte voor de perturbatiewoorden en de correctiewoorden tot deze vier waarden, heeft de variabele rotatie naar links in iedere ronde met S_t ofwel geen effect, ofwel hetzelfde effect als een rotatie over één bit naar links.

In plaats van één zoekprobleem naar de 32-bit perturbatiewoorden voor iedere ronde ontstaan zo 16 identieke problemen waarbij slechts twee bits van de perturbatiewoorden gezocht worden. Het volstaat om slechts één van die problemen, bijvoorbeeld voor de twee minst beduidende bits, op te lossen.

5.6 Besluit

Door de variabele rotatie S_t in de stapfunctie en de verspreiding van de uitgebreide berichtwoorden in iedere ronde, is het met deze methode niet mogelijk om een klein aantal botsingen te vinden dat voldoet aan de berichtexpansie zoals bij SHA-0.

In plaats van slechts één vorm van een lokale botsing te gebruiken, worden hier alle mogelijke vormen van lokale botsingen tegelijkertijd in rekening gebracht. Er wordt een combinatie van 144 lokale botsingen gevonden in Tabel 5.3, die leidt tot een botsing voor de volledige compressiefunctie. De bekomen resultaten nemen aan dat perturbaties beperkt worden tot 0x00000000, 0x55555555, 0xAAAAAAAA of 0xFFFFFFFF. Hierdoor wordt het grote aantal mogelijkheden voor de variabele rotatie S_t beperkt tot een klasse van even, en een klasse van oneven rotaties.

Hoewel deze aanval niet praktisch haalbaar is, valt het op dat het Strict Avalanche Criterion verschillende mogelijkheden toelaat voor een lokale botsing, en dat deze lokale botsingen eveneens zeer kort kunnen zijn (tot slechts twee stappen). Deze eigenschap zal verder nog van pas komen.

Hoofdstuk 6

L-karakteristieken

6.1 Inleiding

In Hoofdstuk 5 werd gezocht naar het minimale aantal lokale botsingen dat nog steeds aan de berichtexpansie voldoet. Een betere doelfunctie zou echter zijn om het aantal bits dat verschilt in de interne toestanden te minimaliseren.

Om dit probleem op te lossen, wordt de hashfunctie eerst gelineariseerd. In een lineaire voorstelling is een botsing volledig bepaald door de exclusieve OF van de berichten m, m', en niet door m of m' zelf. Dit verschil wordt een L-karakteristiek genoemd.

Goede, voor een aanvaller interessante L-karakteristieken hebben de bijkomende eigenschap dat ze met grote kans ook gevolgd worden door de niet-lineaire hashfunctie. Deze kans kan benaderd worden door het aantal verschillende bits in de interne toestand te tellen.

L-karakteristieken worden bij de cryptanalyse van SHA-1 gebruikt door V. Rijmen en E. Oswald in [37]. Met de resultaten uit dit hoofdstuk is een betere aanval mogelijk op de vereenvoudigde HAS-V dan in Hoofdstuk 5. Het lijkt echter niet haalbaar om deze techniek te gebruiken om een botsing voor meer dan twee ronden te bekomen, zoals aangegeven in Tabel 6.1.

6.2 Lineaire benadering

Alle niet-lineaire elementen van de hashfunctie moeten vervangen worden door lineaire benaderingen bij het zoeken naar L-karakteristieken. Een lineaire functie λ is een goede benadering voor een niet-lineaire functie γ indien de aan de relatie

$$\gamma(x \oplus \delta) \oplus \gamma(x) = \lambda(x \oplus \delta) \oplus \lambda(x) = \lambda(\delta) \tag{6.1}$$

voldaan is voor relatief veel waarden van x en δ. Bij HAS-V is de berichtexpansie volledig lineair. In de stapfunctie wordt de (niet-lineaire) optelling modulo 2^{32} vervangen door de exclusieve OF. Het is eenvoudig om in te zien dat een optelling niet-lineair is. In de optelling van A en B, hangt het resultaat van $C[i]$ niet alleen af het verschil tussen $A[i]$ en $B[i]$, maar ook van de

waarde van de overdracht uit de berekening van $C[i-1]$. De booleaanse functies f_j worden vervangen door de lineaire benaderingen die zoveel mogelijk invoerverschillen opslorpen. Indien in Figuur 5.1 voor 50% van de mogelijke invoerverschillen ook een uitvoerverschil is, wordt er altijd voor gekozen om dit verschil op te slorpen. Zo worden de volgende lineaire benaderingen bekomen:

$$\begin{aligned} f_1(B,C,D,E) &\triangleq 0 \\ f_2(B,C,D,E) &\triangleq D \\ f_3(B,C,D,E) &\triangleq 0 \end{aligned} \tag{6.2}$$

Uit experimenten zal blijken dat van alle lineaire benaderingen, deze leiden tot de beste L-karakteristieken. Merk op dat in tegenstelling tot in Hoofdstuk 5, er slechts één lineaire benadering wordt gebruikt voor elke booleaanse functie.

6.3 De compressiefunctie als matrixvermenigvuldiging

Vanaf nu tot het einde van deze masterproef, wordt de cyclische formulering uit Paragraaf 3.4 gebruikt voor de vereenvoudigde HAS-V.

Door het lineariseren van de volledige compressiefunctie, is het mogelijk deze voor te stellen door een matrixvermenigvuldiging met een matrix A, waarvan de coëfficiënten in $GF(2)$ liggen:

$$y_{1\times(32\cdot T)} = m_{1\times(32\cdot 16)} \cdot A_{(32\cdot 16)\times(32\cdot T)} \tag{6.3}$$

Indien het aantal stappen T van de compressiefunctie kleiner dan 60 gekozen wordt, spreekt men van een variant met een gereduceerd aantal stappen.

A is een vaste matrix, eigen aan de gelineariseerde hashfunctie. De matrix A komt overeen het toepassen van zowel de berichtexpansie, als de berekening van de interne toestanden Q_{t+1} voor $0 \leq t < T$ van de compressiefunctie. In [37] wordt dit door twee aparte matrices beschreven, en wordt een pariteitscontrolematrix berekend vanuit de coderingstheorie. Hier begint de berichtexpansie echter niet met een identiteitsmatrix, wat een gelijkaardige beschrijving bemoeilijkt. De matrix A wordt daarom in deze tekst niet geschreven als het product van verschillende matrices.

Vector m bestaat uit de 16 berichtwoorden, hier voorgesteld als een vector met alle bits uit die berichtwoorden. De vector y geeft alle interne toestanden Q_{t+1} van met $0 \leq t < T$ van de lineaire benadering van de compressiefunctie. Deze wordt een L-karakteristiek genoemd. De laatste 160 bits van y geven de uitvoer van de compressiefunctie.

De inhoud van de i-de rij van de matrix A komt overeen met de interne toestanden Q_{t+1} met $0 \leq t < T$ van de lineaire benadering van de compressiefunctie indien de i-de bit van de berichtvector m de waarde 1 krijgt, en de overige 511 gelijk gesteld worden aan 0,

Stel dat er twee berichten m, m' gevonden kunnen worden met slechts een klein aantal bits verschillend in de interne toestanden (Q_{t+1}, Q'_{t+1}), met de laatste 160 bits (de uitvoer van de

compressiefunctie) gelijk. Met een bepaalde kans, zullen deze berichten ook een botsing geven voor de echte, niet-lineaire compressiefunctie. Indien deze kans kleiner is dan 2^{-80}, is deze aanval een efficiëntere manier om een botsing te vinden dan algemene technieken steunend op de verjaardagsparadox.

De rijruimte van A stelt alle mogelijke berichtwoorden voor. Door een beperkt aantal waarden van y vast te kiezen, kan uit het stelsel $y = m \cdot A$ met coëfficiënten in $GF(2)$ een nieuwe, kleinere oplossingsruimte voor y bekomen worden via Gausseliminatie. Deze techniek om de oplossingsruimte te verkleinen zal in de volgende paragraaf gebruikt worden om het zoekprobleem haalbaar te maken.

6.4 Codewoorden met een laag Hamminggewicht

Interpreteer y als codewoorden van een lineaire blokcode. Uit de coderingstheorie volgt dat voor lineaire codes het verschil tussen twee codewoorden y_1, y_2 zelf ook een codewoord is. Het volstaat dus om te zoeken naar een codewoord y met een laag (Hamming-)gewicht, dus een klein aantal bits verschillend van 0. Om uit dit codewoord het bericht m af te kunnen afleiden, wordt een matrix B opgesteld, waarin de matrix A wordt voorafgegaan door een identiteitsmatrix. De eerste $32 \cdot 16$ bits van x geven dan de berichtwoorden m_i weer:

$$x_{1\times(32\cdot16+32\cdot T)} = m_{1\times(32\cdot16)} \cdot B_{(32\cdot16)\times(32\cdot16+32\cdot T)} \tag{6.4}$$

met

$$B_{(32\cdot16)\times(32\cdot16+32\cdot T)} \leftarrow \begin{bmatrix} I_{32\cdot16} & A_{(32\cdot16)\times(32\cdot T)} \end{bmatrix} \tag{6.5}$$

Bij de berekening van het gewicht van het codewoord x worden de eerste $32 \cdot 16$ bits niet in rekening genomen. Het gewicht van de interne toestanden Q_{t+1} uit de eerste ronde wordt evenmin in rekening gebracht, waarbij verondersteld wordt de woorden zodanig gekozen kunnen worden om aan het verschil in Q_{t+1} in deze ronde te voldoen, zoals beschreven in Paragraaf 4.4.

6.5 Zoeken naar codewoorden met een laag Hamminggewicht

Uit de coderingstheorie volgt echter ook, dat het zoeken naar het codewoord met het laagste gewicht, een NP-hard probleem is. Daarom wordt getracht om in redelijke tijd een codewoord te zoeken met een laag gewicht, door het gebruik van een heuristiek. Dit is niet noodzakelijk ook het codewoord met het allerlaagste gewicht, maar kan voldoende laag zijn om de aanval te doen slagen.

Als er een codewoord met een laag gewicht bestaat, zal deze slechts weinig bits bevatten die verschillen van 0. Men kan dus stellen dat er een grote kans is, dat een willekeurige bit van het codewoord met het laagste gewicht, gelijk is aan 0. Stel dat de zoekruimte beperkt wordt door bit i van y gelijk te stellen aan 0.

Dan worden niet alle lineaire combinaties van rijen van B doorzocht, maar enkel lineaire combinaties waarbij de i-de bit in alle rijen 0 is. Stel dat in de j-de rij van A, er zich in kolom i een 1 bevat. Is deze veronderstelling voor geen enkele rij j juist, dan zal het i-de element van vector x altijd 0 zijn, zodat de stappen die nu volgen overbodig worden.

Op alle rijen behalve de j-de, die een 1 bevatten in de i-de kolom, wordt een exclusieve OF toegepast met rij j. Deze j-de rij wordt dan verwijderd uit de matrix B. Alle lineaire combinaties van de overblijvende rijen zullen dan steeds een 0 bevatten in de i-de kolom. Dit algoritme komt overeen met Gausseliminatie toegepast op een matrix met elementen in $GF(2)$.

De laatste vijf interne toestanden Q_{t+1}, waarbij dus $T - 5 \leq t < T$, worden gelijk gesteld aan 0 om een botsing te eisen. Verder worden eveneens willekeurige bits van de interne toestanden Q_{t+1} in de vector y gelijk gesteld aan 0, tot het mogelijk wordt om de overblijvende zoekruimte exhaustief te doorzoeken. Dit exhaustief zoeken gebeurt best door een Gray-code op te stellen die evenveel bits heeft als het aantal rijen van de overblijvende matrix

Van alle lineaire combinaties van de rijen waarvan de overeenkomstige bits in de Gray-code 1 zijn, wordt het Hamminggewicht berekend. Gezien door de Gray-code geldt dat er slechts één rij zal toegevoegd of weggelaten worden uit deze lineaire combinatie bij de volgende berekening, moet slechts één exclusieve OF met deze rij toegepast worden op het vorige codewoord om een nieuw codewoord te bekomen.

Door de meest beduidende bits van deze Gray-code vast te kiezen voor iedere proces dat dit algoritme uitvoert, wordt de berekening eenvoudig geparallelliseerd. Voor het berekenen van het Hamminggewicht van één 32-bit woord, wordt een zijdelingse optelling gebruikt [1].

6.6 Resultaten

Een zeer groot aantal experimenten werden uitgevoerd, met verschillen in onder andere het aantal stappen, de lineaire benadering van de booleaanse functies f_j in iedere ronde, de heuristiek om de zoekruimte te beperken en het type botsing. Zo werden ook bijna-botsingen gezocht, dit zijn L-karakteristieken met een laag gewicht, waarbij de vereiste niet wordt opgelegd dat de uitvoerwaarden identiek zijn.

Indien het toegestaan is dat de invoer van de compressiefunctie verschillen bevat, wordt gesproken van een pseudo-botsing.

Bij een pseudo-bijna-botsing wordt een L-karakteristiek gezocht, die niet noodzakelijk gelijke in- of uitvoerwaarden heeft voor de compressiefunctie. Zoals al beschreven in [28], is het mogelijk om voor HAS-V pseudo-bijna-botsingen te vinden zonder berichtverschil. Dit kan door een verschil in het 32-bit woord E van de IV aan te brengen. Dit verschil kan onmiddellijk opgeslorpt worden door de booleaanse functie f_j, zodat er een botsing ontstaat nog voor de berichtwoorden gebruikt worden. Deze pseudo-bijna-botsingen zijn echter niet bruikbaar in een verdere aanval, gezien $m \neq m'$ vereist is voor een botsing.

Uit deze experimenten blijkt dat andere benaderingen voor de booleaanse functies geen codewoorden met een lager Hamminggewicht opleveren. Tot en met twee ronden kunnen codewoorden met een laag Hamminggewicht gevonden worden, maar zodra de derde ronde in rekening wordt gebracht, stijgt dit Hamminggewicht zeer drastisch. Zoals eerder vermeld in Paragraaf

laagst gevonden gewicht	40 stappen	45 stappen
botsing	30	75
bijna-botsing	26	68
pseudo-botsing	27	65
pseudo-bijna-botsing	0	0

Tabel 6.1: Laagst gevonden Hamminggewichten voor codewoorden y, het gewicht in de eerste ronde niet meegerekend

6.4, worden in Tabel 6.1 de Hamminggewichten van de codewoorden in de eerste ronden niet meegerekend.

In tegenstelling tot bij SHA-0 en SHA-1, vermindert het minimale gewicht van de codewoorden tijdens het zoekproces slechts geleidelijk, in plaats een snelle daling indien een codewoord met een zeer laag gewicht wordt gevonden. Een ander verschil met SHA-0 en SHA-1 is dat de codewoorden geen duidelijke structuur hebben zoals in [5] en [37], maar er willekeurig uitzien.

6.7 Besluit

De kans dat de gelineariseerde hashfunctie overeenkomt met de echte hashfunctie, hangt af van het aantal bits dat verschillend is in de interne toestanden (Q_{t+1}, Q'_{t+1}). Ieder verschil in een (Q_{t+1}, Q'_{t+1}) van de lineaire benadering moet immers hetzelfde gedrag tonen voor zowel de booleaanse functie als de optelling modulo 2^{32}. Ook gaat de veronderstelling niet volledig op dat alle woorden in de eerste ronden vrij gekozen kunnen worden. Het laatste woord voorkomt bij de berekening van een XOR-woord moet immers gelijk zijn aan de exclusieve OF van vier eerdere woorden.

Bij SHA-0 en SHA-1 worden woorden in de berichtexpansie recursief berekend uit vier vorige woorden uit die expansie. Bij de vereenvoudigde HAS-V worden de berichtwoorden in iedere ronde op andere posities verspreid, zoals aangegeven in Tabel 3.7. Dit zorgt er mede voor dat het niet lukte om goede L-karakteristieken te vinden voor meer dan twee ronden, zoals aangegeven in Tabel 6.1.

Hoofdstuk 7

NL-karakteristieken, methode van Stevens

7.1 Inleiding

Tot nu toe werden alle optellingen modulo 2^{32} benaderd door een exclusieve OF. Opdat deze benadering juist is, mag er voor geen enkele positie bij de optelling een verschil ontstaan in de overdrachtsbits. De technieken die vanaf nu behandeld worden, houden wel rekening met deze mogelijkheid. Zoals reeds verduidelijkt in Paragraaf 6.2, is de optelling een niet-lineaire bewerking. De resulterende karakteristieken worden daarom NL-karakteristieken genoemd. Uiteindelijk nemen deze NL-karakteristieken de vorm aan van een differentieel pad, zodat ze gebruikt kunnen worden bij de methoden uit Hoofdstuk 4. Dit hoofdstuk baseert zich op technieken van M. Stevens uit [40].

Bij de cryptanalyse van HAS-160 [46, 6, 28], een hashfunctie die veel gelijkenissen vertoont met HAS-V, worden eveneens differentiële paden gebruikt waarbij de niet-lineaire eigenschappen van de optelling worden gebruikt. Deze differentiële paden werden echter niet bekomen via geautomatiseerde methoden.

Het begrip Binary Signed Digit voorstelling (BSDR) wordt verduidelijkt, om een BSDR te bekomen die Non-Adjacent Form of NAF heet. Deze NAF is niet alleen uniek, maar heeft ook van alle BSDR's het kleinste aantal niet-nul coëfficiënten. In [36] wordt deze NAF geïntroduceerd, met bewijs van de eigenschappen. Een algoritme wordt gegeven om een BSDR met een gegeven aantal niet-nul coëfficiënten te bekomen.

Verder worden bitcondities verduidelijkt, en wordt een methode gegeven om een differentieel pad te construeren. Hoewel de methode van M. Stevens niet toepasbaar lijkt op de vereenvoudigde HAS-V, wordt wel een differentieel pad gegeven dat met de hand werd geconstrueerd. Een meer systematische behandeling van de relatie tussen verschillen bij de exclusieve OF en bij optelling modulo 2^{32} kan in [10] gevonden worden.

7.2 Binary Signed Digit voorstelling

Een Binary Signed Digit voorstelling of BSDR van een woord X is een sequentie $Y \leftarrow (k_i)_{i=0}^{31}$ met $k_i \in \{1, 0, -1\}$ waarbij

$$X \equiv \sum_{i=0}^{31} k_i \cdot 2^i \bmod 2^{32} \tag{7.1}$$

Deze BSDR zal als $[a_i]$ genoteerd worden, met

$$a_i = \begin{cases} i & \text{als } k_i = 1 \\ \emptyset & \text{als } k_i = 0 \\ -i & \text{als } k_i = -1 \end{cases} \tag{7.2}$$

Hierbij geeft $\emptyset$ aan dat het element niet vermeld wordt bij de opsomming. Deze notatie is een variant van de notatie van Wang [44], met als verschil dat daar de nummering van de bits daar vanaf 1 start. Er zijn vaak verschillende manieren om dezelfde X voor te geven als een BSDR. Indien $X = 1$, dan kan X geschreven worden als 2^0, $2^1 - 2^0$, $2^2 - 2^1 - 2^0$, $2^3 - 2^2 - 2^1 - 2^0$, ... In de gebruikte notatie wordt dit: $[0]$, $[1, -0]$, $[2, -1, -0]$, $[3, -2, -1, -0]$, ...

Iedere BSDR heeft ook een (Hamming-)gewicht, dit zijn het aantal k_i die van 0 verschillen. Merk uit het voorbeeld op hoe een woord met een klein gewicht in de binaire voorstelling toch een groot gewicht in de BSDR kan hebben, door deze bit als het ware 'uit te smeren'.

Van alle BSDR-voorstellingen, heeft de Non-Adjacent Form (NAF) [36] het laagst mogelijke gewicht. Bij de NAF zijn geen twee opeenvolgende k_i verschillend van 0. Er kunnen echter ook andere BSDR's zijn met hetzelfde gewicht. Zo kan 3 geschreven worden als $[2, -0]$ in NAF, maar ook als $[1, 0]$ met hetzelfde gewicht. Indien $k_{31} = 0$, dan is de NAF uniek. Indien een BSDR bestaat met $k_{31} = 1$, zal er ook een BSDR met $k_{31} \equiv -1$ bestaan door de modulobewerking. Om een woord X om te zetten in NAF kan het algoritme van Reitwiesner gebruikt worden [36].

Gegeven een woord X en een maximaal gewicht voor de BSDR's, dat minstens gelijk is aan het gewicht van de NAF, kunnen alle BSDR's die hieraan voldoen als volgt efficiënt opgesomd worden. Stel $d \leftarrow X$. Itereer dan voor iedere bit $i = 0 \ldots 31$. Als d op positie i de bit 0 bevat, geldt dit ook voor de BSDR. Indien $d - 2^i = 0$, is er een BSDR met op positie i de term $+2^i$. Dit is tevens ook de laatste BSDR die op positie i de term $+2^i$ bevat. Indien $d - 2^i \neq 0$ is er enkel een BSDR met op positie i de term $+2^i$ indien $|d - 2^i| > 2^i$. Vanaf bit $i + 1$ kunnen immers enkel verschillen d van minstens 2^{i+1} gecorrigeerd worden. Voor deze BSDR wordt dan $d \leftarrow d - 2^i$, en wordt de volgende iteratiestap uitgevoerd.

Een analoge berekening gebeurt om na te gaan of er een BSDR is met op positie i de term -2^i. Het algoritme voert geen volgende iteratiestap uit voor BSDR's die een verschil $d = 0$ of $d \leq 2^i$ hebben, of die een gewicht hebben dat gelijk is aan het maximaal toegelaten gewicht.

$x[i]$	conditie voor $(X[i], X'[i])$	k_i
-	$X[i] = X'[i]$	0
n	$X[i] = 0, \quad X'[i] = 1$	+1
u	$X[i] = 1, \quad X'[i] = 0$	−1

Tabel 7.1: Condities voor $(X[i], X'[i])$

7.3 Bitcondities

Om aan te tonen hoe deze BSDR's gebruikt worden bij HAS-V, wordt (3.29) voor de stapfunctie opnieuw vermeld:

$$Q_{t+1} \leftarrow (Q_t \lll S_t) + f_j(Q_{t-1}, Q_{t-2} \ggg 2, Q_{t-3} \ggg 2, Q_{t-4} \ggg 2) + W_t + K_t \tag{3.29}$$

Indien er een verschil is in het woordpaar (W_t, W'_t), zal dit zich manifesteren als een additief verschil modulo 2^{32} in (Q_{t+1}, Q'_{t+1}). Om de verschillen voor de vijf opeenvolgende stappen te volgen, is het belangrijk om te weten voor welke bits i het bitpaar $(Q_{t+1}[i], Q'_{t+1}[i])$ verschillend is. Deze worden vastgelegd door hiervoor een BSDR te gebruiken. Voor bits i waar $k_i \neq 0$, zijn dan eveneens de bits $(Q_t[i], Q'_t[i])$ bekend. Een voorwaarde die opgelegd wordt aan twee bits in de interne toestanden (Q_t, Q'_t) of het woordpaar (W_t, W'_t), wordt een conditie genoemd. Voorlopig kunnen drie verschillende condities zich voordoen, die een verkorte notatie krijgen in Tabel 7.1.

7.4 Constructie van differentiële paden

Bij het construeren van een voorwaarts pad, is het best om gegeven een additief verschil modulo 2^{32} in (Q_{t+1}, Q'_{t+1}), een BSDR te kiezen die een laag gewicht heeft. Hiervoor kan gebruik gemaakt worden van de NAF, of een BSDR een iets hoger gewicht, zoals beschreven in Paragraaf 7.2.

Op een gelijkaardige manier wordt dan een achterwaarts pad geconstrueerd. Deze paden ontmoeten elkaar op het einde van de eerste ronde, zodat er nog vrijheid is om (W_t, W'_t) te kiezen. Over een kort aantal stappen doet men een poging om deze twee paden met elkaar te verbinden. Indien dit niet lukt, moeten andere voorwaartse en/of achterwaartse paden gekozen worden.

7.5 Resultaten

De hashfunctie HAS-160 is vrij gelijkaardig aan HAS-V. Voor HAS-160 werden via niet-geautomatiseerde methoden botsingen gevonden voor 45 stappen [46] en 53 stappen [6]. Indien de differentiële paden voor HAS-V er gelijkaardig uit zouden zien, wordt al snel duidelijk dat de techniek van M. Stevens hier moeilijk toepasbaar is. Zo is in het differentiële pad voor 53 stappen van HAS-160, het Hamminggewicht van de BSDR's voor (Q_{t+1}, Q'_{t+1}) met $4 \leq t \leq 17$

t	∇Q_{t+1}	∇W_t
15	---------unnnnnnnnnnnnnnnnnnnnnn	------------------------------u
16	----------------u-------------u-	------------------------------u
17	--------------------------------	--------------------------------
18	nu--------------------------u--	--------------------------------
19	nn--------------------------u--	--------------------------------
20	--------------------------------	------------------------------u
21	--------------------------------	------------------------------u
22	--------------------------------	--------------------------------
23	--------------------------------	--------------------------------
24	--------------------------------	--------------------------------

Tabel 7.2: Differentieel pad voor 45 stappen met licht gewijzigde berichtexpansie, handmatig bekomen

altijd groter dan 6, met uitzondering van $t = 7$ en $t = 16$. Bij $t = 15$ heeft de gebruikte BSDR zelfs een gewicht van 12.

Stek dat we voor een bepaalde stap t alle BSDR's beschouwen voor (Q_{t+1}, Q'_{t+1}), die een Hamminggewicht van hoogstens zes hebben. Dit geeft $\sum_{k=0}^{6} C_{32}^{m} \cdot 2^m = 65\,057\,409$ mogelijke BSDR's. Hierbij geeft C_{32}^{m} alle mogelijkheden aan om m niet-nul elementen te kiezen in de 32-bit BSDR. De factor 2^m bepaalt dat op ieder van deze posities in de BSDR zowel $+1$ als -1 kunnen voorkomen. Indien er 64 bits worden gebruikt voor de geheugenvoorstelling van één 32-bit BSDR, is er zo 496 MB nodig om alle BSDR's op te slaan, en dit enkel voor deze ene stap t. Alle BSDR's met een gewicht tot en met 12 voorstellen op deze manier vraagt zelfs 9,3 TB. Gezien de booleaanse functies f_j vaak ieder verschil aan de ingang zowel kunnen opslorpen als doorgeven, zullen een groot aantal van deze BSDR's bij het zoekproces mogelijk zijn.

Er kan voor gekozen worden om de mogelijke BSDR's voor (Q_{t+1}, Q'_{t+1}) niet allen op te slaan voor iedere stap t, door bijvoorbeeld diepte-eerst te zoeken in plaats van breedte-eerst. Hoewel er nu geen geheugenprobleem is, kan het zoekproces nu zeer veel tijd innemen, gezien identieke deelproblemen waarbij er een gelijke BSDR voor $(Q_{t'+1}, Q'_{t'+1})$ is voor een bepaalde stap t', dan telkens opnieuw uitgerekend worden.

Zelfs indien een partieel voorwaarts en achterwaarts pad gevonden wordt, ontstaan er problemen bij de toepassing van Algoritme 6.1 uit [40, p. 31] om deze paden te verbinden. Waar bij MD5 de bitcondities voor $(Q_{t+1}[i], Q'_{t+1}[i])$ berekend kunnen worden voor opeenvolgende $i = 0 \ldots 31$, wordt dit bij HAS-V bemoeilijkt door de manier waarop de twee rotaties gebruikt worden.

Om deze redenen werden er geen resultaten bekomen door het uitbreiden naar HAS-V van de technieken van M.. Stevens uit [40]. Tijdens het debuggen van de code bleek het echter mogelijk om eenvoudige differentiële paden handmatig te construeren. In het differentiële pad van Tabel 7.2 voor 45 stappen worden de XOR-woorden 17 en 19 in de eerste ronde van plaats verwisseld, en de XOR-woorden 16 en 19 in de tweede ronde. In berichtwoord m_{12} wordt een verschil van -2^0 geïntroduceerd. Voor deze alternatieve berichtexpansie zijn alle (W_t, W'_t) gelijk, behalve $\nabla W_{15} = \nabla W_{16} = \nabla W_{20} = \nabla W_{21} = -2^0$.

De constructie is vrij eenvoudig. Een eerste verschil wordt in ∇W_{15} geïntroduceerd, en dan in de interne toestand ∇Q_{t+1} 'uitgesmeerd' door een BSDR te kiezen met een groot Hamminggewicht.

Het Strict Avalanche Criterion waaraan f_1 voldoet, wordt dan gebruikt om invoerverschillen op te slorpen, of juist door te geven aan de uitvoer van f_1 om zo een later geïntroduceerd woordverschil op te slorpen.

7.6 Besluit

Hoewel de pogingen om de methode van M. Stevens te gebruiken om differentiële paden te construeren niet slaagden, kon wel een differentieel pad voor met de hand gevonden worden voor 45 stappen (zie Tabel 7.2). Hierbij werd de berichtexpansie van de vereenvoudigde HAS-V licht aangepast. De vorm van een mogelijk differentieel pad is nu bekend. Deze kennis zal verder gebruikt worden om deze paden op een automatische manier te genereren.

Hoofdstuk 8

NL-karakteristieken, methode van De Cannière en Rechberger

8.1 Inleiding

Bij de techniek ontwikkeld door C. De Cannière en C. Rechberger in [12], wordt een NL-karakteristiek berekend om de verbinding te realiseren van een goede L-karakteristiek met de in- en uitvoer van de compressiefunctie. Hoewel voor de vereenvoudigde HAS-V zo'n goede L-karakteristiek niet gevonden wordt, blijkt het toch mogelijk om gebruik te maken van methoden die voor SHA-1 ontwikkeld zijn. Dit kan door op zoek te gaan naar korte botsingen van een specifieke structuur, geïnspireerd door het resultaat uit Tabel 7.2, dat met de hand werd bekomen in Hoofdstuk 7. Verder zal blijken dat het niet nodig is om deze specifieke structuur op te leggen, en dat nog betere resultaten bekomen kunnen worden, indien enkel de voorwaarde wordt opgelegd dat de botsingen kort zijn.

De techniek van C. De Cannière en C. Rechberger wordt in zijn geheel verduidelijkt, verder uitgebreid en geïmplementeerd. Hiervoor worden de condities uit Tabel 7.1 veralgemeend in Tabel 8.1. Dubbelcondities [20] en een veralgemeende methode om rekening te houden met de XOR-woorden vormen verdere uitbreidingen die niet in de oorspronkelijke methode gebruikt werden.

Er wordt uitgelegd hoe de interne toestanden (Q_t, Q'_t) worden voorgesteld bij deze techniek, en hoe met deze interne voorstelling een propagatie van opgelegde condities uitgevoerd kan worden, voor zowel de stapfunctie als de berichtexpansie.

Bij ieder differentieel pad kan het verwachte aantal knopen berekend worden dat in de zoekboom moet doorlopen worden om een botsing te vinden. De berekening van deze werkfactor N_w aan de hand van de ongecontroleerde kans $P_u(t)$ en de gecontroleerde kans $P_c(t)$ wordt verduidelijkt.

Een globaal zoekalgoritme wordt gegeven om differentiële paden te berekenen met een lage werkfactor N_w. Hier wordt eveneens toegelicht hoe een botsing gevonden kan worden.

Een differentieel pad (Tabel 8.7) met $N_w = 2^{7,70}$ en een botsing voor 26 stappen (Tabel 8.8) worden gegeven. Voor 45 stappen wordt een differentieel pad met $N_w = 2^{49,21}$ gegeven, bekomen door de technieken uit [12] te implementeren en verder uit te breiden. In Tabel 8.12 wordt een

	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
?	✓	✓	✓	✓
-	✓	-	-	✓
x	-	✓	✓	-
0	✓	-	-	-
u	-	✓	-	-
n	-	-	✓	-
1	-	-	-	✓
#	-	-	-	-

	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
3	✓	✓	-	-
5	✓	-	✓	-
7	✓	✓	✓	-
A	-	✓	-	✓
B	✓	✓	-	✓
C	-	-	✓	✓
D	✓	-	✓	✓
E	-	✓	✓	✓

Tabel 8.1: Alle mogelijke condities voor $(X[i], X'[i])$

differentieel pad voor 60 stappen gegeven met $N_w = 2^{108,22}$.

8.2 Veralgemeende condities

Net als in [12], worden de mogelijke condities voor twee bits uitgebreid ten opzichte van Tabel 7.1 uit Hoofdstuk 7. Een volledig overzicht van alle mogelijke condities voor $(X[i], X'[i])$ en hun verkorte notatie wordt gegeven in Tabel 8.1. De condities zelf worden als $\nabla X[i]$ genoteerd. Een NL-karakteristiek bestaat uit een verzameling van condities $\nabla Q_{-4}, \ldots \nabla Q_T$ en $\nabla W_0 \ldots \nabla W_{T-1}$.

8.3 Voorstelling van de interne toestanden

8.3.1 Voorstelling van condities voor één bit $\nabla Q_{t+1}[i]$ in de interne toestand

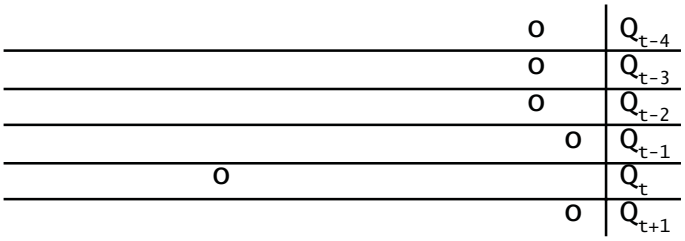
Gelijkaardig aan de voorstelling van condities voor de interne toestanden ∇Q_{t+1} in [12], verduidelijkt in [11], worden de interne toestanden van de vereenvoudigde HAS-V beschreven.

De stapfunctie wordt als volgt geschreven, voor $0 \leq t < T$ en $0 \leq i < 32$. Indices i worden modulo modulo 32 berekend. De binnenkomende overdracht van de optelling wordt genoteerd als $C_{t,j}$, de buitengaande overdracht als $C_{t,j+1}$.

$$\begin{aligned} C_{t,i+1} \parallel Q_{t+1}[i] \leftarrow & Q_t[i - S_t] \\ & + f_j(Q_{t-1}[i], Q_{t-2}[i+2], Q_{t-3}[i+2], Q_{t-4}[i+2]) \\ & + W_t[i] + K_t[i] + C_{t,i} \end{aligned} \tag{8.1}$$

Om $Q_{t+1}[i]$ te berekenen, worden de bitposities i_k van de vorige $Q_{t-k}[i_k]$ voor $0 \leq k < 5$ schematisch voorgesteld in Figuur 8.1. Deze grafische voorstelling van de condities uit (8.1) zal gebruikt worden bij de introductie van dubbelcondities in Paragraaf 8.5.

Merk op dat de echte booleaanse functies f_j gebruikt worden in plaats van een lineaire benadering. Zo kan er optimaal gebruik gemaakt worden van de eigenschap dat invoerverschillen vaak zowel opgeslorpt als doorgeven kunnen worden. De optelling modulo 2^{32} wordt evenmin benaderd.



Figuur 8.1: Berekening van $Q_{t+1}[0]$ uit $Q_t[32 - S_t]$, $Q_{t-1}[0]$, $Q_{t-1}[2]$, $Q_{t-1}[2]$ en $Q_{t-1}[2]$

Bij de berekening van ieder bitpaar $(Q_{t+1}[i], Q_{t+1}[i])$ worden daartoe de overdrachten $C_{t,i}$ uit de berekening van $Q_t[i]$ bijgeteld, alsook een nieuw overdrachtspaar $(C_{t,i+1}, C'_{t,i+1})$ berekend. $(C_{t,0}, C'_{t,0}) \triangleq (0, 0)$ gezien er voor de eerste bit nog geen overdracht is. Bij HAS-V kan $C_{t,i}$ hoogstens 3 worden, en dus met 2 bits voorgesteld worden.

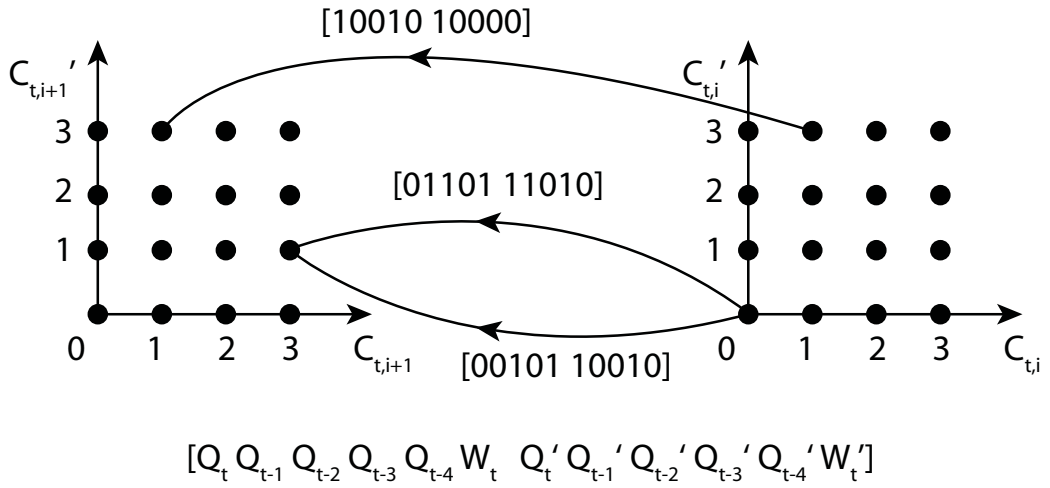
Voor de berekening van iedere $\nabla Q_{t+1}[i]$, wordt een lijst bijgehouden van alle overblijvende $(C_{t,i}, C'_{t,i})$ (2×2 bits), (Q_{t-k}, Q'_{t-k}) met $0 \leq k < 5$ (2×5 bits) en (W_t, W'_t) (2×1 bit). Ieder van deze 2^{16} combinaties wordt voorgesteld als één bit geheugen, die aangeeft of de combinatie al dan niet kan voorkomen. Het vernauwen van condities $\nabla Q_{t+1}[i]$ en $\nabla W_t[i]$ tijdens het zoekproces geeft aanleiding tot het schrappen van sommige van deze combinaties.

Ieder van deze 2^{16} invoerbits bepaalt de zes uitvoerbits $(Q_{t+1}[i], Q'_{t+1}[i])$ (2×1 bit) en $(C_{t,i+1}, C'_{t,i+1})$ (2×2 bits) volledig. Hiervoor kan een 'wijzer' (pointer) naar een opzoekingstabel gebruikt worden, die verschillend is voor iedere ronde naar gelang de booleaanse functie f_j die gebruikt wordt, en verschillend naar gelang de waarde van $K_t[i]$. In totaal vereist de voorstelling van één bit voor alle 2^{16} invoercombinaties, voor alle 32 bits van één woord ∇Q_{t+1} en voor alle stappen van de volledige compressiefunctie $t = 0 \ldots 59$ een geheugengebruik van 15 MB.

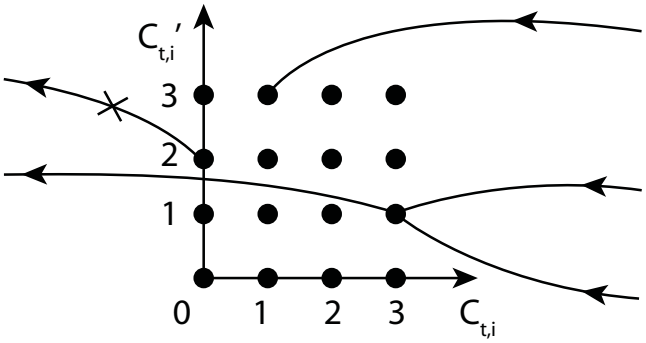
In een praktische implementatie kan voor de berekening van $\nabla Q_{t+1}[i]$ naast deze 2^{16} invoerbits een lijst van wijzers naar de condities $\nabla Q_t[i - S_t]$, $\nabla Q_{t-1}[i]$, $\nabla Q_{t-2}[i + 2]$, $\nabla Q_{t-3}[i + 2]$ en $\nabla Q_{t-4}[i + 2]$ bijgehouden worden. Eveneens wordt de conditie ∇W_t bijgehouden, alsook de drie dubbelcondities die in Paragraaf 8.5 geïntroduceerd zullen worden. Zo wordt een voorstelling bekomen die, eens de opzoekingstabel en de wijzers naar de vorige condities geïnitialiseerd zijn, onafhankelijk is van de gebruikte hashfunctie. Hierdoor is het eenvoudiger om het programma aan te passen indien de hashfunctie verandert. Het staat ook toe dat inzicht verworven wordt in de algemene gebruikte principes van MD4-achtige hashfuncties, aangezien elementen die specifiek zijn aan de implementatie van één specifieke hashfunctie afgeschermd worden.

8.3.2 Berekening van overdrachten tussen condities voor bits $\nabla Q_{t+1}[i]$, $i = 0 \ldots 31$

Zoals aangegeven in Figuur 8.2, is de enige informatie bij de berekening van $\nabla Q_{t+1}[i]$ die bij $\nabla Q_{t+1}[i + 1]$ gebruikt wordt, de mogelijkheden voor de overdrachten die zich kunnen voordoen. De overdrachten $(C_{t,i}, C'_{t,i})$, die worden gebruikt als invoer bij de berekening van $\nabla Q_{t+1}[i + 1]$, moeten immers een geldige uitvoer zijn bij de berekening van $\nabla Q_{t+1}[i]$. Dit wordt nagegaan door bits i voorwaarts $(0, 1, \ldots, 31)$ te doorlopen, waarbij overdrachten die als invoer gebruikt worden, geschrapt worden indien ze geen geldige uitvoer waren bij de berekening van de vorige



Figuur 8.2: Betekenis van de verbindingen in de grafe



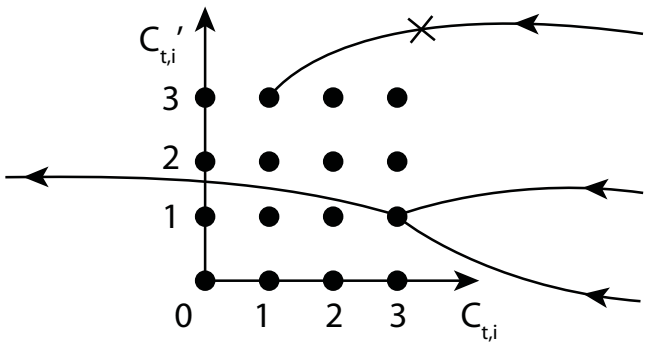
Figuur 8.3: Verbindingen schrappen bij voorwaarts doorlopen

bit. Zie hiervoor Figuur 8.3.

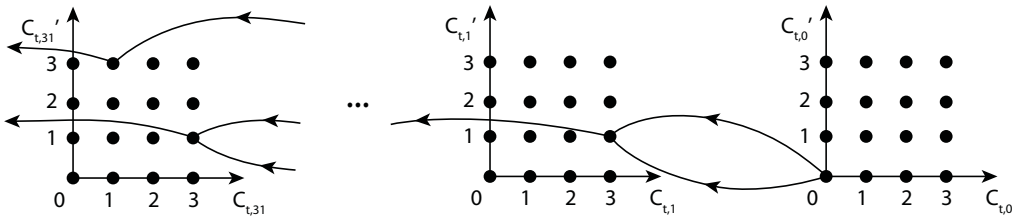
Hier moet echter de aanname gemaakt worden, dat alle overdrachten die als uitvoer gebruikt worden, geldig zijn. Gezien deze veronderstelling niet altijd juist is, moeten bits i ook achterwaarts $(31, 30, \ldots, 0)$ doorlopen worden. Analoog wordt nu berekend welke overdrachten als invoer kunnen bij $\nabla Q_{t+1}[i+1]$, om deze vereisten op te leggen aan de overdrachten als uitvoer bij $\nabla Q_{t+1}[i]$. Dit wordt geïllustreerd in Figuur 8.4. Bij dit achterwaarts doorlopen, worden de invoercondities $\nabla Q_t[i - S_t]$, $\nabla Q_{t-1}[i]$, $\nabla Q_{t-2}[i+2]$, $\nabla Q_{t-3}[i+2]$,$\nabla Q_{t-4}[i+2]$ en $\nabla W_t[i]$, de uitvoerconditie $\nabla Q_{t+1}[i]$ en de drie dubbelcondities bijgewerkt indien nodig. Aangezien ieder woord altijd zowel voorwaarts als achterwaarts doorlopen wordt, is het overbodig om deze aanpassing al bij het voorwaarts doorlopen uit te voeren.

8.3.3 Voorstelling van condities voor één woord ∇Q_{t+1} in de interne toestand

Gegeven de voorstelling van condities voor één bit $\nabla Q_{t+1}[i]$, worden alle mogelijkheden voor condities van één woord ∇Q_{t+1} bepaald door alle paden in de grafevoorstelling uit Figuur 8.5.



Figuur 8.4: Verbindingen schrappen bij achterwaarts doorlopen



Figuur 8.5: Geldige overblijvende paden

Ieder van deze paden moet starten in $(C_{t,0}, C'_{t,0}) = (0, 0)$, gezien er nog geen overdracht is voor de eerste bit. Ze moeten ook allen eindigen één van de 16 mogelijkheden voor $(C_{t,31}, C'_{t,31})$. Gezien deze laatste overdracht niet gebruikt wordt in een berekening, is er geen beperking op de toegelaten waarden. Het beschreven algoritme schrapt alle verbindingen tussen twee knopen, indien ze geen deel uitmaken van een geldig pad.

Dit kan zeer efficiënt gebeuren via hierboven beschreven algoritme, gezien het niet nodig is om alle mogelijkheden voor condities van het woord ∇Q_{t+1}, expliciet te berekenen zoals in Hoofdstuk 7. Door dit algoritme, dat in de computerwetenschappen onder de noemer dynamisch programmeren valt, wordt niet alleen het vereiste geheugen beperkt, maar worden ook een groot aantal overlappende deelproblemen tegelijkertijd opgelost.

Merk op dat er veel parallelle verbindingen kunnen zijn tussen twee knopen in de grafe. Bij de berekening van ∇Q_{t+1} kunnen er verschillende paden zijn met dezelfde (Q_{t+1}, Q'_{t+1}), maar met verschillende invoerwaarden. Deze dienen slechts één keer geteld te worden. Op een analoge manier kan uit alle paden ∇W_t berekend worden, waarbij paden met dezelfde (W_{t+1}, W'_{t+1}), maar verschillende (Q_{t-k}, Q'_{t-k}) met $-1 \leq k < 5$, slechts één keer in rekening gebracht moeten worden.

8.3.4 Voordelen ten opzichte van de methode van Stevens

Het is niet nodig om alle ∇Q_{t+1} en ∇W_t uit alle paden expliciet te berekenen en op te slaan. Dit zou immers te veel geheugen vereisen.

Bij het zoeken naar een goede karakteristiek, is het gewenst om zo weinig mogelijk verschillen te

introduceren in de interne toestanden. Gelijke bits in de invoer van de stapfunctie leiden immers altijd tot gelijke bits aan de uitvoer, terwijl invoerverschillen meestal slechts met een bepaalde kans een uitvoerverschil geven. Door systematisch tijdens het zoekproces vrije bits te kiezen, en van deze te eisen dat ze gelijk zijn voor beide berichten, kan de NL-karakteristiek verbeterd worden. Deze techniek dient toegepast te worden zolang het aantal verwachte berekeningen bij het zoeken naar een botsing, de werkfactor N_w uit [12], verminderd wordt. De berekening van de werkfactor wordt verduidelijkt in Paragraaf 8.7.

Deze techniek leidt mogelijk tot betere differentiële paden, gezien het aantal verschillen in ∇Q_{t+1} niet wordt beperkt bij opbouwen van de grafevoorstelling. Dit gebeurt pas in een latere stap wanneer er extra condities opgelegd worden op deze bits. Op die manier wordt een algoritme bekomen dat niet 'gulzig' (greedy) is, maar kan toelaten om grotere verschillen in ∇Q_{t+1} toe te laten, om zo latere verschillen uit te sparen.

8.4 Propagatie van condities doorheen alle stappen

De vorige beschrijving geeft aan hoe de condities voor één woord ∇Q_{t+1} en ∇W_t berekend kunnen worden. Deze berekening moet uitgevoerd worden voor $0 \leq t < T$. Voor de volledige compressiefunctie is $T = 60$. Telkens hierbij verbindingen in één van de grafen geschrapt worden, moet dit algoritme opnieuw uitgevoerd worden. Aangezien het aantal verbindingen in de grafen niet kan toenemen, en het algoritme enkel wordt uitgevoerd indien er verbindingen geschrapt werden, stopt deze berekening na een eindig aantal stappen.

Zonder optimalisatie zal deze berekening echter zeer traag verlopen. Een eerste optimalisatie is, om het algoritme afwisselend voor stijgende en voor dalende waarden van t uit te voeren. Zo kan informatie in beide richtingen propageren, zonder één van deze richtingen te bevoordelen.

Een eerste propagatie doorheen alle stappen wordt enkel bij initialisatie uitgevoerd. Het is daarom vooral belangrijk om volgende propagaties te optimaliseren. Hierbij worden meestal slechts een beperkt aantal condities $\nabla Q_{t+1}[i]$ en $\nabla W_t[i]$ aangepast door het schrappen van verbindingen bij een vorige propagatie of door het toevoegen van extra condities door het globale zoekalgoritme. Daarom wordt bij de berekening van iedere $\nabla Q_{t+1}[i]$ een tabel bijgehouden met de vorige in- en uitvoerwaarden in deze stap.

Zonder verschil aan de invoerwaarden, zal er evenmin een verschil aan de uitvoerwaarden zijn. Het kan echter voorkomen dat er een verschil is in de interne voorstelling, doordat verbindingen door propagatie in de andere richting (voorwaarts of achterwaarts) geschrapt werden. Deze verandering in de interne voorstelling wordt ook aangegeven. Indien er verbindingen geschrapt werden bij het voorwaarts doorlopen van $\nabla Q_{t+1}[i]$, zal deze $\nabla Q_{t+1}[i]$ altijd ook opnieuw achterwaarts worden doorlopen. Hetzelfde geldt indien er verbindingen geschrapt werden bij het achterwaarts doorlopen van $\nabla Q_{t+1}[i]$. Dan wordt er geen gebruik gemaakt van de opgeslagen waarden bij het voorwaarts doorlopen van $\nabla Q_{t+1}[i]$.

Een verdere uitbreiding zou niet alle alle stappen $0 \leq t < T$ overlopen, maar slechts deze stappen waarop aanpassingen uit de vorige iteratie invloed kunnen hebben. Deze stappen kunnen bijgehouden worden in een lijst, die in de literatuur ook wel een 'dirty list' wordt genoemd. Deze techniek wordt eveneens gebruikt in Paragraaf 8.6, waar een concrete toepassing van een 'dirty list' wordt beschreven.

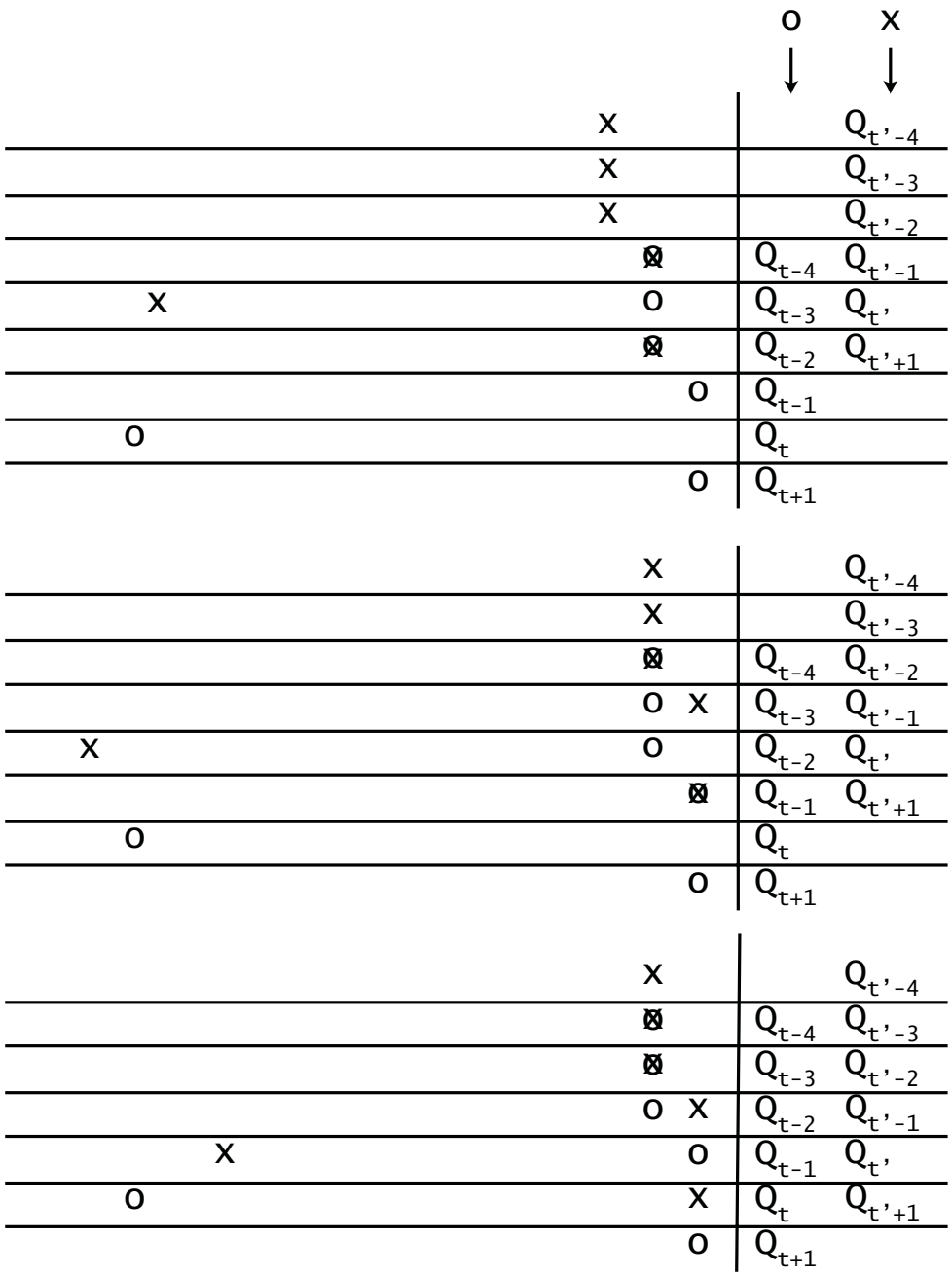
8.5 Dubbelcondities

Condities die niet gelden voor één paar bits, maar voor twee paar bits, worden verder dubbelcondities genoemd. De mogelijkheid om deze bij HAS-V te gebruiken werd gesuggereerd in [20]. Er zijn 2^{16} mogelijke dubbelcondities. Merk de analogie op met Tabel 8.1, met als verschil dat hier condities voor vier bits worden opgelegd. Deze dubbelcondities maken eveneens deel uit van de NL-karakteristiek. Bij de cryptanalyse van MD5 worden ook enkele eenvoudige condities gebruikt voor twee paar bits, onder andere in [22] en [40]. De dubbelcondities die hier worden voorgesteld, zijn hier een veralgemening van. In tegenstelling tot in [40], zijn er geen 'voorwaartse' of 'achterwaartse' dubbelcondities. Iedere dubbelconditie kan in beide richtingen gebruikt worden.

Voor de berekening van één bit van ∇Q_{t+1} kan er op drie plaatsen een dubbelconditie gebruikt worden. Deze worden weergegeven in Figuur 8.6. Hier stellen '`o`' de bits voor die betrokken zijn bij de berekening van $\nabla Q_{t+1}[0]$ en '`x`' de bits uit de berekening van $\nabla Q_{t'+1}[2]$ in de eerste figuur en van $\nabla Q_{t'+1}[0]$ voor de twee onderste figuren. Ieder van deze berekeningen wordt voorgesteld zoals in Figuur 8.1. De figuren stellen de enige mogelijkheden voor waarmee het patroon uit Figuur 8.1 met zichzelf kan overlappen, zodat (minstens) twee keer een '`o`' en een '`x`' elkaar overlappen.

Het gebruik van de eerste dubbelconditie wordt verduidelijkt, voor de anderen verloopt dit analoog. Bij de berekening van $\nabla Q_{t'+1}[2]$ wordt een dubbelconditie opgesteld die het samen voorkomen van $\nabla Q_{t'+1}[2]$ en $\nabla Q_{t'-1}[2]$ weergeeft. Hier kan het bijvoorbeeld het geval zijn, dat beiden de conditie '`x`' opgelegd krijgen, maar dat deze condities ook verschillend moeten zijn van elkaar. Dan zijn enkel (`u`,`n`) en (`n`,`u`) geldige dubbelcondities, maar (`u`,`u`) en (`n`,`n`) niet. De dubbelconditie bevat dus meer informatie dan de twee condities apart. Indien dan $\nabla Q_{t+1}[0]$ berekend wordt, dan zal identiek dezelfde dubbelconditie terugkomen, maar dan opgelegd op de bits $\nabla Q_{t-2}[2]$ en $\nabla Q_{t-4}[2]$. Als hier eveneens de conditie '`x`' opgelegd is voor beide bits, maar de extra voorwaarde dat beide condities gelijk moeten zijn, dan kunnen enkel (`u`,`u`) en (`n`,`n`), wat in strijd is met de dubbelconditie. Zo komt men er op een eenvoudige manier achter dat de karakteristiek inconsistent is.

De extra rekentijd voor deze dubbelcondities, is in de gebruikte implementatie te verwaarlozen. In Tabellen 8.2 en 8.3 wordt aangegeven hoe het gebruik van dubbelcondities ervoor kan zorgen dat condities verder gepropageerd worden. De verschillen tussen deze twee tabellen worden onderlijnd. Deze tabellen tonen een willekeurige situatie waarbij het toevoegen van dubbelcondities ervoor zorgen dat ook gewone condities aangepast worden. Het is eveneens mogelijk dat deze dubbelcondities geen gewone condities aanpassen, maar wel verbindingen in de interne voorstelling schrappen. Dan zullen de kansen uit Paragraaf 8.7 nauwkeuriger berekend kunnen worden. Er bestaan ook voorbeelden waarin het toevoegen van dubbelcondities geen verdere invloed heeft.



Figuur 8.6: Dubbelcondities voor de HAS-V stapfunctie, bekomen door de enige mogelijke overlappingen van minstens twee bits van Figuur 8.1 met een verplaatste versie van dit patroon

t	∇Q_{t+1}	∇W_t
-5	00001111010010111000011111000011	
-4	01000000110010010101000111011000	
-3	01100010111010110111001111111010	
-2	11101111110011011010101110001001	
-1	01100111010001010010001100000001	
0	--------------------------------	--------------------------------
1	-------??????????????----------	----????????????????----------
2	-----????----xx-xx-?????--------	---??x????-???-?????u??--------
3	----???-------?????????????----	---?????----?????x???x??????----
4	--------------------------------	-----?----??x?-?-x-?-?-?--------
5	---------??--????????????------	-------??-?-?-??-x-?-?--??------
6	------x--?????-????------------	----------?-?-x-?-?-?----------
7	--------------------------------	--------------------------------
8	--------------------------------	--------------------------------
9	--------------------------------	--------------------------------

Tabel 8.2: NL-karakteristiek voor 10 stappen, zonder gebruik van dubbelcondities

t	∇Q_{t+1}	∇W_t
-5	00001111010010111000011111000011	
-4	01000000110010010101000111011000	
-3	01100010111010110111001111111010	
-2	11101111110011011010101110001001	
-1	01100111010001010010001100000001	
0	--------------------------------	--------------------------------
1	-------????????????_----------	----????????????????----------
2	-----????----xx-xx-?????--------	---??x????-???-x?????u??--------
3	----???-------?????????????x----	---x????----?????x???x??????----
4	--------------------------------	-----?----??x?-?-x-?-?-?--------
5	---------??--????????????------	-------??-?-?-??-x-?-?--??------
6	------x-------?????------------	----------?-?-x-?-?-?----------
7	--------------------------------	--------------------------------
8	--------------------------------	--------------------------------
9	--------------------------------	--------------------------------

Tabel 8.3: NL-karakteristiek voor 10 stappen, met gebruik van dubbelcondities

8.6 Consistentie van de berichtexpansie

8.6.1 Vereisten van de berichtexpansie

Indien meer dan één ronde van de vereenvoudigde HAS-V wordt uitgevoerd, zal eenzelfde berichtwoord m_i voorkomen op verschillende plaatsen in de berichtexpansie W_t. Eveneens maakt ieder berichtwoord m_i in iedere ronde deel uit van één XOR-woord. Dit XOR-woord moet gelijk zijn aan de exclusieve OF van vier berichtwoorden m_i, zoals aangegeven bij de beschrijving van de vereenvoudigde HAS-V.

8.6.2 Een algoritme om de berichtexpansie consistent te maken

Om ervoor te zorgen dat aan al deze vereisten voldaan is, wordt een aparte functie opgeroepen die indien één conditie $\nabla W_t[i]$ wordt aangepast, ook andere $\nabla W_{t'}[i]$ met $t \neq t'$ op dezelfde bitpositie aanpast indien nodig. Dit wordt gedaan door een lijst op te stellen met condities die mogelijk aangepast moeten worden als een 'lijst met $\nabla W_t[i]$ die mogelijk aangepast moeten worden' (dirty list). Bij het aanpassen van één conditie $\nabla W_t[i]$, zal deze conditie en de conditie voor bit i in het geassocieerde XOR-woord initieel toegevoegd worden aan de lijst. Deze lijst zal alle condities van alle woorden ∇W_t uit de eerste ronde alsook alle XOR-woorden in volgende ronden bevatten, indien dit algoritme voor de eerste keer wordt opgeroepen.

Het algoritme dat nu volgt, wordt uitgevoerd voor ieder conditie ∇W_t dat zich in de lijst bevindt. Deze conditie wordt verwijderd uit de lijst, en nieuwe $\nabla W_{t'}$ met $t \neq t'$ worden enkel toegevoegd indien hun geldige condities verder beperkt werden. Hieruit volgt de eindigheid van het algoritme.

Als de conditie $\nabla W_t[i]$ uit een gewoon berichtwoord komt, dan wordt aan alle condities $\nabla W_{t'}$ met $t \neq t'$ die met dit gewoon berichtwoord overeenkomen, eenzelfde conditie opgelegd. Deze conditie komt overeen met de doorsnede van $\nabla W_t[i]$ en alle $\nabla W_{t'}[i]$. Indien hierdoor één van deze condities aangepast wordt, wordt de conditie op dezelfde bitpositie uit het overeenkomstige XOR-woord toegevoegd aan de lijst.

Komt de conditie ∇W_t van een XOR-woord, dan worden deze conditie, alsook de vier condities uit de hiermee verbonden gewone berichtwoorden mogelijk aangepast, gebruik makende van de voorwaarde dat de exclusieve OF van deze vier condities gelijk is aan de conditie van het XOR-woord. Wat hier precies mee wordt bedoeld en hoe dit geïmplementeerd wordt, zal verder verduidelijkt worden. Indien hierbij een conditie uit een gewoon berichtwoord aangepast wordt, zal deze toegevoegd worden aan de lijst.

8.6.3 Een algoritme om de XOR-woorden consistent te maken

De vereiste dat de exclusieve OF van vier condities uit gewone berichtwoorden gelijk moet zijn aan de conditie op dezelfde bitpositie in het XOR-woord, komt overeen met de eis dat de exclusieve OF van deze vijf condities gelijk moet zijn aan $(0, 0)$. Dit wordt geïllustreerd met enkele voorbeelden.

Stel dat deze vijf condities gelijk zijn aan '`--?--`'. Men kan dan exhaustief alle combinaties geven die aan deze condities voldoen. Dit zijn alle combinaties met $(0,0)$ of $(1,1)$ voor iedere '`-`', en $(0,0)$, $(1,0)$, $(0,1)$ of $(1,1)$ voor '`?`'. Er is dan een totaal van $2 \times 2 \times 4 \times 2 \times 2 = 64$ geldige toekenningen voor deze vijf condities. Bij ieder van deze combinaties wordt nagegaan of de exclusieve OF gelijk is aan $(0,0)$. Dit zal slechts in de helft van de combinaties het geval zijn. Zo kunnen combinaties waarbij de derde conditie gelijk is aan $(1,0)$ of $(0,1)$ immers nooit voldoen. Indien deze geschrapt worden, worden de vijf condities opnieuw bepaald. Hierdoor wordt '`-----`' bekomen, waarbij de derde conditie werd aangepast. Wat dit precies betekent, is dat er nu voor iedere geldige toekenning van één van deze condities, ook een geldige toekenning bestaat voor de vier andere condities waarbij de exclusieve OF gelijk is aan $(0,0)$.

Hetzelfde kan gedaan worden voor '`--?-x`'. De berekening is dezelfde, maar nu wordt '`--x-x`' bekomen. Merk het verschil op met het vorige voorbeeld, het is hier de laatste conditie die de waarde van de derde conditie bepaalt. Het zal dus nooit volstaan om alle condities eenmalig voorwaarts te doorlopen.

Stel het aantal mogelijkheden voor iedere conditie, in dit geval 4, gelijk aan m. Indien n het totaal aantal condities in de berekening is, in dit geval 5, dan is de complexiteit van het exhaustief zoeken gelijk aan $\mathcal{O}(m^n)$ indien er veel mogelijkheden zijn voor de condities.

Dit kan echter verbeterd worden tot $\mathcal{O}(m^2 \cdot n)$ door het gebruik van dynamisch programmeren. Dit wordt geïllustreerd in Figuur 8.7. Iedere verbinding stelt een mogelijke waarde voor een conditie voor. De knopen stellen dan de exclusieve OF voor van alle voorgaande paden. Voor de eerste knoop kan dit enkel $(0,0)$ zijn, aangezien nog geen enkele conditie verwerkt is. Op het einde wordt de eis gesteld dat de exclusieve OF van alle condities gelijk is aan $(0,0)$. Deze grafe wordt eerst voorwaarts van rechts naar links doorlopen, waarbij alle verbindingen worden getekend van paden die starten in $(0,0)$. Bij het achterwaarts van links naar rechts doorlopen, worden de verbindingen die niet in $(0,0)$ eindigen, geschrapt. Deze worden in stippellijn weergegeven in Figuur 8.8. In Figuur 8.9 worden de verbindingen in stippellijn weggelaten en wordt de derde conditie veranderd van '`?`' naar '`-`'.

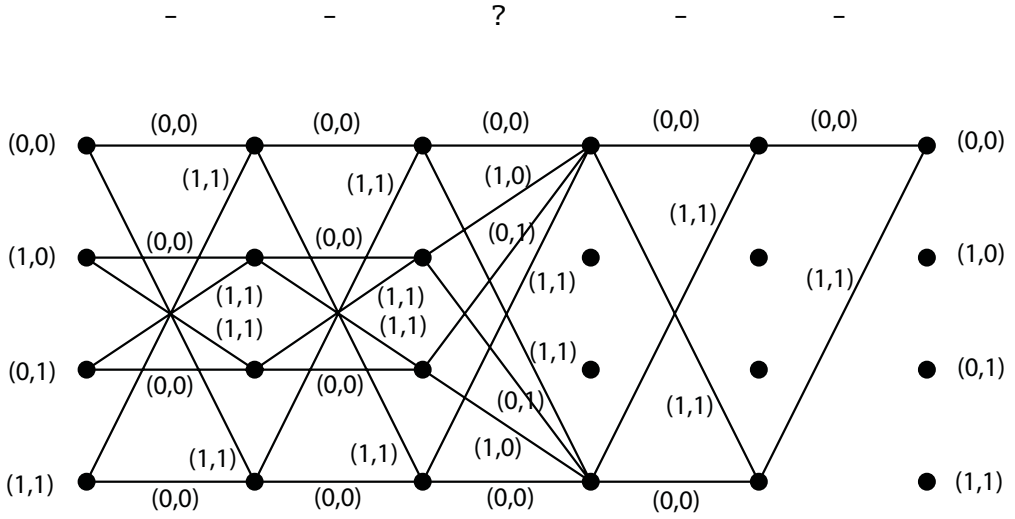
Merk op dat deze berekening analoog is aan deze uit Paragraaf 8.4, zij het daar met veel grotere dimensies en met mogelijk meer dan één verbinding tussen twee knopen. Daarom was het niet mogelijk om daar een volledig uitgewerkt voorbeeld te geven.

8.7 Berekening van de werkfactor van een differentieel pad

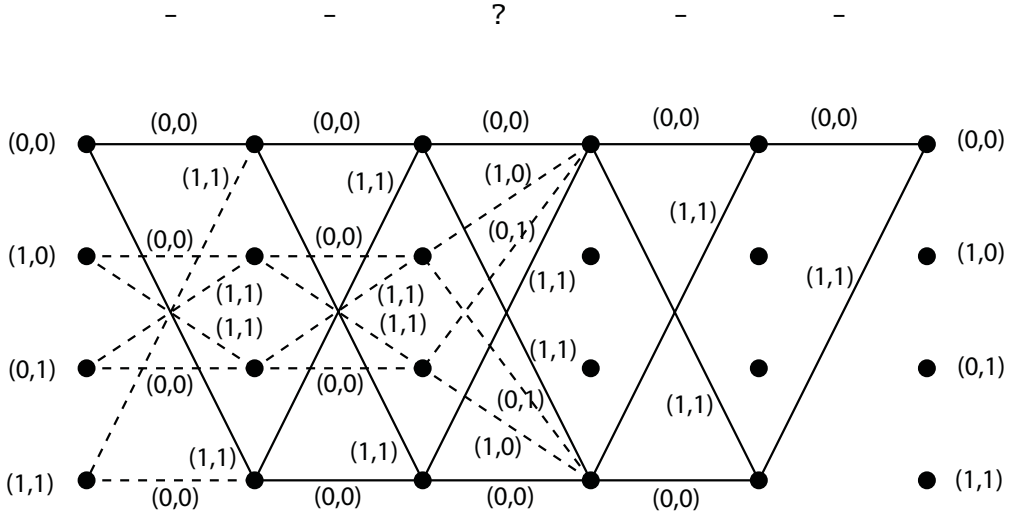
Het bepalen van de werkfactor van een differentieel pad, gebeurt zoals beschreven in [12] en verduidelijkt in [11].

8.7.1 De woordvrijheid $F_W(t)$

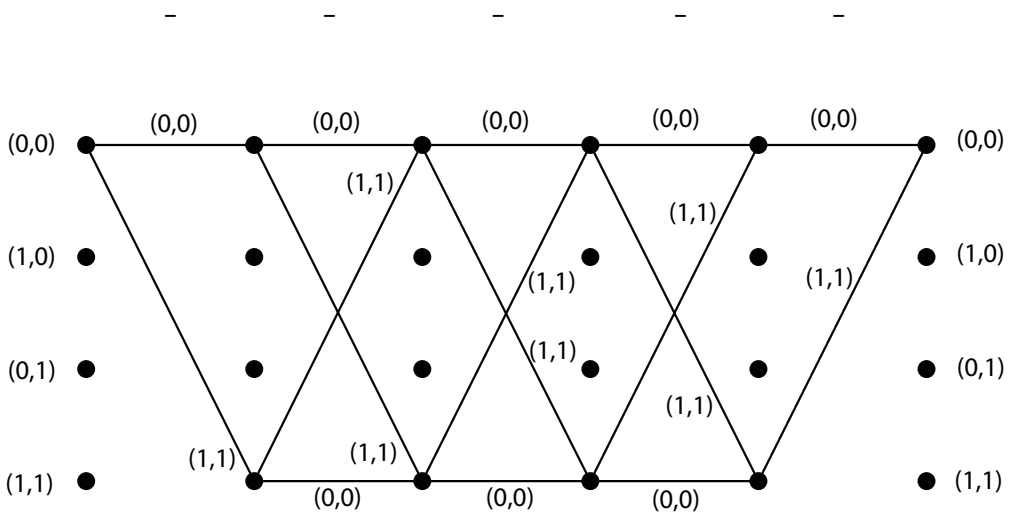
Bij het zoekproces kan er gebruik gemaakt worden van de in Paragraaf 4.4 beschreven enkelvoudige berichtmodificatie. Door de beperkingen die de XOR-woorden opleggen, gaat dit niet voor alle 20 woordparen (W_t, W'_t) in de eerste ronde. Van de vijf woordparen die betrokken zijn bij ieder XOR-woord, kunnen de vier eerste (W_t, W'_t) gekozen worden. De laatste $(W_{t'}, W'_{t'})$ is niet vrij te kiezen, maar moet gelijk zijn aan de exclusieve OF van de vier andere (W_t, W'_t).



Figuur 8.7: Voorwaarts doorlopen van de grafe bij de XOR-woorden



Figuur 8.8: Achterwaarts doorlopen van de grafe bij de XOR-woorden



Figuur 8.9: Aanpassing van de derde conditie bij de XOR-woorden

De woordvrijheid $F_W(t)$ stelt het aantal mogelijkheden voor om ieder woordpaar woordpaar (W_t, W'_t) te kiezen, zonder een (lineaire) voorwaarde te schenden die door de berichtexpansie wordt opgelegd, gegeven vaste waarden van (W_j, W'_j) voor $0 \leq j < t$. Uit de beschrijving van de vereenvoudigde HAS-V volgt dat $F_W(t)$ altijd 1 is voor $t = 10, 14, 15, 19$ en $t \geq 20$. Voor de andere t is $F_W(t)$ het product van het aantal mogelijkheden voor de condities $\nabla W_t[i]$ voor $i = 0 \ldots 31$. Dit aantal mogelijkheden is het aantal vinkjes (✓) voor de overeenkomstige conditie in Tabel 8.1.

8.7.2 De ongecontroleerde kans $P_u(t)$

De ongecontroleerde kans $P_u(t)$ van een karakteristiek op stap t is de kans dat een uitvoerpaar (Q_{t+1}, Q'_{i+1}) de karakteristiek volgt, gegeven dat alle invoerparen (Q_{t-k}, Q'_{t-k}) met $0 \leq k < 5$ en de woorden (W_t, W'_t) deze karakteristiek ook volgen:

$$\begin{aligned} P_u(t) = & P((Q_{t+1}, Q'_{t+1}) \in \nabla Q_{t+1} \\ & \mid (Q_{t-k}, Q'_{t-k}) \in \nabla Q_{t-k} \text{ voor } 0 \leq k < 5, \text{ en } (W_t, W'_t) \in \nabla W_t) \end{aligned} \tag{8.2}$$

Deze kans komt overeen met alle mogelijke overblijvende paden uit Figuur 8.5, gedeeld door alle mogelijke paden waarbij alleen voor de invoerparen (Q_{t-k}, Q'_{t-k}) met $0 \leq k < 5$ en de woorden (W_t, W'_t) de karakteristiek moeten volgen, maar niet het uitvoerpaar (Q_{t+1}, Q'_{t+1}).

De kans dat de karakteristiek gevolgd wordt, kan preciezer bepaald worden, indien de karakteristiek ook de dubbelcondities uit Paragraaf 8.5 bevat.

8.7.3 De gecontroleerde kans $P_c(t)$

De gecontroleerde kans $P_c(t)$ van een karakteristiek op stap t is de kans dat er minstens één paar berichtwoorden (W_t, W'_t) bestaat dat de karakteristiek volgt, zodat het uitvoerpaar (Q_{t+1}, Q'_{t+1}) de karakteristiek ook volgt, gegeven dat alle invoerparen (Q_{t-k}, Q'_{t-k}) met $0 \leq k < 5$ de karakteristiek ook volgen:

$$\begin{aligned} P_c(t) = & P(\exists (W_t, W'_t) \in \nabla W_t : \ (Q_{t+1}, Q'_{t+1}) \in \nabla Q_{t+1} \\ & \mid (Q_{t-k}, Q'_{t-k}) \in \nabla Q_{t-k} \text{ voor } 0 \leq k < 5) \end{aligned} \tag{8.3}$$

Om deze kans te bepalen, wordt eveneens een grafe opgesteld voor iedere bit i van de berekening van $(Q_{t+1}[i], Q'_{t+1}[i])$. Deze bevat echter per segment niet $4 \times 4 = 16$ toestanden voor alle combinaties van overdrachten $(C_{t,i}, C'_{t,i})$. In plaats daarvan, wordt hier een overdrachtsmasker gebruikt, dat 2^{16} mogelijke waarden kan aannemen. Ieder van deze waarden geeft aan welke combinatie van de 16 mogelijke overdrachten $(C_{t,i}, C'_{t,i})$ kan voorkomen. Merk de analogie op met de veralgemeende condities uit Tabel 8.1, die aangeeft welke combinaties van (X_t, X'_t) kunnen voorkomen.

Laat p het aantal mogelijkheden van $(Q_{t-k}[i], Q'_{t-k}[i]) \in \nabla Q_{t-k}[i]$ voor $0 \leq k < 5$ voorstellen. Voor ieder van deze mogelijkheden, worden alle overdrachten $(C_{t,i}, C'_{t,i})$ en alle woordparen $(W_t[i], W'_t[i])$ doorlopen. Deze worden gebruikt om een 16×16 matrix in te vullen die aangeeft welke transitiemogelijkheden van $(C_{t,i}, C'_{t,i})$ naar $(C_{t,i+1}, C'_{t,i+1})$ mogelijk zijn.

Gegeven deze 16×16 transitiematrix, kunnen de 2^{16} mogelijke overdrachtsmaskers doorlopen worden. Initieel heeft enkel één overdrachtsmasker, dat aangeeft dat de overdracht dan enkel $(0, 0)$ kan zijn, een kans van 1, en alle andere een kans van 0. Uit de transitiematrix kan afgeleid worden, welk het overdrachtsmasker voor bit $i + 1$ zal zijn, gegeven een overdrachtsmasker voor bit i. Ieder van deze verbindingen krijgt een kans van $1/p$. Parallelle verbindingen komen hier niet voor.

Door deze berekening voor ieder van de 32 bits van een woord uit te voeren, weten de kans van ieder overdrachtsmasker indien het volledige woord doorlopen wordt. Eén van deze overdrachtsmaskers geeft aan dat geen enkele verzameling van overdrachten $(C_{t,31}, C'_{t,31})$ geldig is. De kansen van alle andere overdrachtsmaskers worden gesommeerd, en geven $P_c(t)$.

De kans dat er minstens één paar berichtwoorden (W_t, W'_t) bestaat dat de karakteristiek volgt, kan preciezer bepaald worden, indien de karakteristiek ook de dubbelcondities uit Paragraaf 8.5 bevat.

8.7.4 De totale werkfactor N_w

Het gemiddeld aantal uitgaande verbindingen van een knoop tijdens het zoekproces wordt dan gegeven door $F_W(t) \cdot P_u(t)$. Slechts een fractie $P_c(t)$ van alle knopen heeft kinderen. Zodra de laatste stap $T - 1$ van de compressiefunctie wordt bereikt, stopt het algoritme. Hieruit volgt de volgende recursieve relatie voor het aantal knopen $N_s(t)$ dat bij iedere stap van de compressiefunctie wordt gevolgd:

$$N_s(t) = \begin{cases} 1 & \text{voor } t = T-1 \\ \max\left(N_s(t+1) \cdot F_W^{-1}(t) \cdot P_u^{-1}(t), P_c^{-1}(t)\right) & \text{voor } 0 \leq t < T-1 \end{cases} \tag{8.4}$$

De totale werkfactor wordt gegeven door

$$N_w = \sum_{t=0}^{T-1} N_s(t) \tag{8.5}$$

Deze totale werkfactor geeft een schatting van kost van het zoekproces naar berichten m, m' die het differentiële pad volgen. Bij de berekeningen en voor de weergave in tabellen wordt gewerkt met de logaritmen met basis 2 van voor $F_W(t)$, $P_u(t)$, $P_c(t)$, $N_s(t)$ en $N_w(t)$.

Dezelfde techniek die in Paragraaf 8.4 gebruikt wordt om waarden uit een vorige berekening te hergebruiken, kan ook hier overgenomen worden. Indien er immers geen verschil is in de verbindingen in de interne toestand, gevormd door de mogelijkheden voor $(Q_{t-k}[i], Q'_{t-k}[i]) \in \nabla Q_{t-k}[i]$ voor $0 \leq k < 5$ en voor het woordpaar $(W_t[i], W'_t[i])$, zal er evenmin een verschil zijn in het deel van de berekening van $P_u(t)$ of $P_c(t)$ dat voor bit i in stap t wordt uitgevoerd.

8.8 Een globaal zoekalgoritme voor differentiële paden

Om een NL-karakteristiek te bekomen via deze methode, wordt in [12] het algoritme opgestart met een spaarse L-karakteristiek. Aangezien goede L-karakteristieken niet gevonden werden voor de vereenvoudigde HAS-V (zie Tabel 6.1), wordt hier gebruik gemaakt van zeer korte botsingen.

Indien een aanzet voor een differentieel pad wordt gegeven, kan de werkfactor van dit differentiële pad verbeterd worden door een willekeurige bitpositie te kiezen die nog geen beperkingen opgelegd kreeg ('`?`'), en deze te vervangen door de conditie '`-`'. Dit is gelijkaardig aan de techniek die bij het zoeken naar L-karakteristieken gebruikt werd, met als verschil dat hier de echte, niet-lineaire compressiefunctie gebruikt wordt. Dan wordt nagegaan hoe deze conditie zich propageert.

Dit wordt herhaald tot alle condities zonder beperkingen geëlimineerd worden, of tot een inconsistentie gevonden wordt. Dan wordt er backtracking toegepast over deze beslissingen. Een uitbreiding van dit algoritme kan ook '`x`' condities omzetten in '`u`' of '`n`' zodra deze verschijnen, en ook backtracken over deze beslissingen. Dit proces heeft enkel zin zolang de werkfactor van het differentiële pad verminderd kan worden.

Er zijn verschillende verbeteringen mogelijk bij dit zoekalgoritme. Zo kan het zijn dat niet alle toegelaten mogelijkheden voor één conditie zich ook echt kunnen voordoen, maar dat dit pas blijkt nadat deze conditie één van deze mogelijkheden krijgt, gevolgd door een propagatie. Dit kan nagegaan worden, door alle condities die nog verschillende mogelijkheden toelaten, beurt om beurt alle mogelijke waarden te geven en te propageren. Indien uit deze propagatie een inconsistentie volgt, wordt deze mogelijkheid weggelaten uit de conditie.

t	∇Q_{t+1}	∇W_t
-5	00001111010010111000011111000011	
-4	01000000110010010101000111011000	
-3	01100010111010110111001111111010	
-2	11101111110011011010101110001001	
-1	01100111010001010010001100000001	
0	------------------------------u	------------------------------u
1	-------B???????x????????????????n	------------------------------x
2	?x?????????????????????????x-------	--------------------------------
3	--------------------------------	--------------------------------
4	--------------------------------	--------------------------------

Tabel 8.4: NL-karakteristiek voor 5 stappen, zonder consistente waarde voor de onderlijnde conditie

Deze techniek is vooral nuttig om na te gaan of het differentiële pad nog steeds een geldige toekenning kan hebben. In Tabel 8.4 kunnen geen inconsistenties gevonden worden bij propagatie. Er bestaat echter geen geldige toekenning voor de onderlijnde conditie. Indien deze conditie de waarde '`-`' of '`x`' krijgt, ontstaat na propagatie immers altijd een inconsistentie. Deze inconsistentie kan met de zonet beschreven methode echter wel gevonden worden.

Een karakteristiek is inconsistent als de propagatie van een net aangepaste conditie tot een inconsistentie leidt, of indien er geen geldige toekenning is voor iedere conditie afzonderlijk. Het heeft geen zin om verder te propageren zodra een inconsistentie wordt gevonden, zodat het algoritme op deze plaats stopt en de positie van deze conditie doorgeeft. Uit experimenten blijkt, het vaak bij dezelfde conditie fout loopt indien een inconsistentie gevonden wordt. Er kan gebruik gemaakt worden van deze informatie, door deze condities eerst toe te kennen in het zoekproces, en over deze beslissingen te backtracken indien nodig.

8.9 Zoeken naar een botsing

Om twee berichten m, m' te vinden die het differentiële pad volgen, kan een zoekproces gebruikt worden zoals beschreven in Paragraaf 4.4. Een mogelijke uitbreiding hier is, om niet enkel condities voor de individuele bits, maar ook voor de dubbelcondities op te leggen. Deze maken in feite ook deel uit van de NL-karakteristiek. Indien het zoekproces vaak voor dezelfde conditie faalt, is het mogelijk om deze conditie niet willekeurig te kiezen, een vaste waarde te geven in het begin en hierover te backtracken.

8.10 Resultaten

In een eerste differentieel pad in Tabel 8.5, worden verschillen geïntroduceerd in $(m_0[0], m_0[0]')$ en $(m_2[0], m_2[0]')$. Enkel 26 stappen worden uitgevoerd, zodat alleen (W_1, W_1') en (W_3, W_3') verschillen bevatten. Het bitverschil in $\nabla W_1[0]$ wordt uitgesmeerd van $\nabla Q_{t+1}[0]$ tot $\nabla Q_{t+1}[31]$.

t	∇Q_{t+1}	∇W_t	F_W	$P_u(t)$	$P_c(t)$	$N_s(t)$
-5	00001111010010111000011111000011					
-4	01000000110010010101000111011000					
-3	01100010111010110111001111111010					
-2	11101111110011011010101110001001					
-1	01100111010001010010001100000001					
0	0110--111011000------01--0000101	0110--10110100-------1---1011011	12,00	-2,00	0,00	0,00
1	nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn	----------1---11111010011--1100u	15,00	-15,00	0,00	0,00
2	----------------------------0000u	----------------------------00001	27,00	0,00	0,00	0,00
3	-------?------uu----?-?---?-----	--------------1----------------u	30,00	-7,00	-3,66	3,66
4	?-??----?---?--D-?--------?-----	--------------------------------	32,00	-19,42	-15,85	15,85
5	-?-----------?---?-?---?--?-?-??	--------------------------------	32,00	-23,00	-18,14	18,14
6	??-----??-----?---?----------???	--------------------------------	32,00	-21,47	-18,72	18,72
7	--------------------------------	--------------------------------	32,00	-16,06	-16,06	16,06
8	--------------------------------	--------------------------------	32,00	-11,19	-11,19	11,19
9	--------------------------------	--------------------------------	32,00	-9,31	-9,31	9,31
10	--------------------------------	--------------------------------	0,00	-6,74	-6,74	10,47
11	--------------------------------	--------------------------------	32,00	-3,74	-3,74	3,74
12	--------------------------------	--------------------------------	32,00	0,00	0,00	0,00
13	--------------------------------	--------------------------------	32,00	0,00	0,00	0,00
14	--------------------------------	--------------------------------	0,00	0,00	0,00	0,00
15	--------------------------------	--------------------------------	0,00	0,00	0,00	0,00
16	--------------------------------	--------------------------------	32,00	0,00	0,00	0,00
17	--------------------------------	--------------------------------	32,00	0,00	0,00	0,00
18	--------------------------------	--------------------------------	32,00	0,00	0,00	0,00
19	--------------------------------	--------------------------------	0,00	0,00	0,00	0,00
20	--------------------------------	--------------------------------	0,00	0,00	0,00	0,00
21	--------------------------------	--------------------------------	0,00	0,00	0,00	0,00
22	--------------------------------	--------------------------------	0,00	0,00	0,00	0,00
23	--------------------------------	--------------------------------	0,00	0,00	0,00	0,00
24	--------------------------------	--------------------------------	0,00	0,00	0,00	0,00
25	--------------------------------	--------------------------------	0,00	0,00	0,00	0,00

Tabel 8.5: Differentieel pad voor 26 stappen, automatisch berekend, $N_w = 2^{19,70}$

Voor voor ∇W_2 tot ∇W_6 waren alle condities initieel '?'. Dit werd gevolgd door alle condities '-' voor ∇W_7 tot ∇W_{11} om een botsing te eisen.

Dan werden condities '?' door '-' vervangen, waarna telkens een propagatiestap werd uitgevoerd. Uiteindelijk wordt het differentiële pad uit Tabel 8.5 bekomen, met als werkfactor $N_w = 2^{19,70}$.

Gebruik makende van dit differentiële pad, wordt een botsing bekomen, die in Tabel 8.6 gegeven is. Merk op dat een groot aantal berichtwoorden (m_i, m_i') pas gebruikt worden nadat de botsing opgetreden is. Deze kunnen dus vrij gekozen worden.

Het blijkt echter niet nodig te zijn om de voorwaarde op te leggen dat deze eerste bit breed uitgesmeerd wordt. Indien deze voorwaarde weggelaten wordt, kan een beter differentieel pad

t	∇Q_{t+1}	∇W_t
-5	00001111010010111000011111000011	
-4	01000000110010010101000111011000	
-3	01100010111010110111001111111010	
-2	11101111110011011010101110001001	
-1	01100111010001010010001100000001	
0	01100011101100001011101110000101	01100010110100011111111011011011
1	nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn	01111110001001111110100111111100u
2	10110111010111101011011101100000u	01011101011011010110111010100001
3	0111100n111011uu110101n111u00100	01001001001000110111010000011100u
4	n1un1011u00000110111000100n11110	01110000001100010011010001100111
5	1u11001111010u11011n100u0110u1u1	11110010111101110111100011010101
6	u011000n110101n110u011110110010n	10110101010011111011001100010010
7	00100100011111111000100110000001	11011110111001001100101011010001
8	00010000000011100001000100110111	11000011001110000000001010111000
9	00111001100000100101001001101010	01011100100000101100111000010001
10	00100011101000011101111100110011	00011010010110001100011100100110
11	10010010111111000111010101101110	00101000111111011000000011110011
12	10010010111111000111010101101110	00000111000111110010101000101010
13	10010001101101100010110011011010	10111101010111010010001000001110
14	10011100101100101110000010000001	11110000011011100111011000001100
15	00010001001111010000100101101110	11110100000100011011010101101010
16	11010110000010001010111010010001	01101011010001101011111101100110
17	11100111111111010100010110001100	01110001100110010111101111011100
18	01110110001110000111011100011000	01000011101111100000000100111101
19	01100100100001111011100101101111	10101011100101101011110101010010
20	11001111111010011011001001011010	01111111011110100001010100011010
21	00011000000000011100101101011001	01101011010001101011111101100110
22	11011011100100000111001010000100	11011110111001001100101011010001
23	11100010101001111101110001101100	01000011101111100000000100111101
24	10100000000000010000100101111011	01011100100000101100111000010001
25	10000110100101111001101101011011	01000000111111110000001011001011

Tabel 8.6: Botsing voor differentieel pad van 26 stappen, automatisch berekend

t	∇Q_{t+1}	∇W_t	F_W	$P_u(t)$	$P_c(t)$	$N_s(t)$
-5	00001111010010111000011111000011					
-4	01000000110010010101000111011000					
-3	01100010111010110111001111111010					
-2	11101111110011011010101110001001					
-1	01100111010001010010001100000001					
0	----------0------------------1--	--------------------------------	32	-2,00	0,00	0,00
1	--------------------------------u	--------------------------------u	31	0,00	0,00	0,00
2	-1--------------------u-------	--------------------------------	32	-2,00	0,00	0,00
3	----------u------------------u-	--------------------------------u	31	-2,00	0,00	0,00
4	--------------u------------u---	--------------------------------	32	-2,00	0,00	0,00
5	---------uu---------un----------	--------------------------------	32	-7,00	-3,00	3,00
6	--------u--------------------n-	--------------------------------	32	-6,00	-4,00	4,00
7	--------------------------------	--------------------------------	32	-5,00	-5,00	5,00
8	--------------------------------	--------------------------------	32	-6,00	-6,00	6,00
9	--------------------------------	--------------------------------	32	-6,00	-6,00	6,00
10	--------------------------------	--------------------------------	0	-2,00	-2,00	2,00
11	--------------------------------	--------------------------------	32	0,00	0,00	0,00
12	--------------------------------	--------------------------------	32	0,00	0,00	0,00
13	--------------------------------	--------------------------------	32	0,00	0,00	0,00
14	--------------------------------	--------------------------------	0	0,00	0,00	0,00
15	--------------------------------	--------------------------------	0	0,00	0,00	0,00
16	--------------------------------	--------------------------------	32	0,00	0,00	0,00
17	--------------------------------	--------------------------------	32	0,00	0,00	0,00
18	--------------------------------	--------------------------------	32	0,00	0,00	0,00
19	--------------------------------	--------------------------------	0	0,00	0,00	0,00
20	--------------------------------	--------------------------------	0	0,00	0,00	0,00
21	--------------------------------	--------------------------------	0	0,00	0,00	0,00
22	--------------------------------	--------------------------------	0	0,00	0,00	0,00
23	--------------------------------	--------------------------------	0	0,00	0,00	0,00
24	--------------------------------	--------------------------------	0	0,00	0,00	0,00
25	--------------------------------	--------------------------------	0	0,00	0,00	0,00

Tabel 8.7: Beter differentieel pad voor 26 stappen, automatisch berekend, $N_w = 2^{7,70}$

bekomen worden met als werkfactor $N_w = 2^{7,70}$. Deze wordt gegeven in Tabel 8.7, met een botsing in Tabel 8.8. Gezien er geen brede uitsmering nodig is van de bitverschillen in de interne toestanden ∇Q_{t+1}, zijn er een groot aantal mogelijkheden om alle verschillen in ∇Q_{t+1} over dezelfde afstand te roteren, waarbij telkens nieuwe geldige differentiële paden kunnen bekomen worden.

Om een differentieel pad van 45 stappen te bekomen, wordt een ander woordverschil gebruikt. Nu worden enkel verschillen geïntroduceerd in $(m_{12}[0], m_{12}[0]')$ en $(m_{14}[0], m_{14}[0]')$. Verschillen in dezelfde woorden werden eveneens geïntroduceerd in Tabel 5.1, maar nu slechts met één bit verschillend bij ieder van deze berichtwoorden. Net als in 'Stadium 2' uit [12], wordt het differentiële pad bekomen door condities '`?`' door '`-`' te vervangen, en condities '`x`' door '`u`' of

t	∇Q_{t+1}	∇W_t
-5	00001111010010111000011111000011	
-4	01000000110010010101000111011000	
-3	01100010111010110111001111111010	
-2	11101111110011011010101110001001	
-1	01100111010001010010001100000001	
0	01111111010010011000011111010111	01111110011010101100101100101101
1	01111000100100011001001010000u	00110000010101101110101111011110u
2	11110011011101010010010u0010000	01010000110110100110100100010010
3	10100000011u01010000101110111 0u0	10100000010011111000110010001 01u
4	00011110010001 1u100110101001u000	00111000111100111111010001011 1001
5	0010111010uu100011110un001010100	01100000010011010000000110101001
6	110000111u011110101011101010 01n0	01101000101101101110011000101001
7	01100111000111110011011100011001	11111111101010000000011011100100
8	10100011000100111101001100111001	10101111100110101100000101101001
9	01100011101000011100001100010100	10011001011000010100100001010011
10	11100001110110100011111001010011	11111000001100001110011010011101
11	10011000111011111101011100010010	10101100010001001011001100100111
12	10010001101110001101110011001101	01001001000110010101010101100110
13	01000111001110011000011001011110	11011111111100000110100111101101
14	01100101110010011011010011111100	01000100110001110100010010000001
15	10010000000110010101101101100010	10100001111001010110100111110111
16	10100010111101101110100111110110	01001111011011100101010110101 11
17	01100101110000000001001010100010	10010110000010110010011101100011
18	10000110011011100001000100001111	01111110001000100101101001111111
19	11110100000010110111001001011010	11000111000100110101011001100010
20	10011110101110011010100111001111	10100101100100110111011000101011
21	11111101011011110010000000111100	01001111011011100101010110101 11
22	10100010111011010001100111101000	11111111101010000000011011100100
23	11101000111101011101001101100010	01111110001000100101101001111111
24	11001010110100100100111011011011	10011001011000010100100001010011
25	11111010001110010100111000000010	11000101111010110000100011110011

Tabel 8.8: Botsing voor beter differentieel pad van 26 stappen, automatisch berekend

t	∇Q_{t+1}	∇W_t	F_W	$P_u(t)$	$P_c(t)$	$N_s(t)$
-5	00001111010010111000011111000011					
-4	01000000110010010101000111011000					
-3	01100010111010110111001111111010					
-2	11101111110011011010101110001001					
-1	01100111010001010010001100000001					
0	----------	----------	32	0,00	0,00	0,00
1	----------	----------	32	0,00	0,00	0,00
2	----------	----------	32	0,00	0,00	0,00
3	----------	----------	32	0,00	0,00	0,00
4	----------	----------	32	0,00	0,00	0,00
5	----------	----------	32	0,00	0,00	0,00
6	----------	----------	32	0,00	0,00	0,00
7	----------	----------10	30	0,00	0,00	0,00
8	----------	----------	32	0,00	0,00	0,00
9	----------	----------	32	0,00	0,00	0,00
10	----------	----------	0	0,00	0,00	0,00
11	----------	----------	32	0,00	0,00	0,00
12	----------	----------	32	0,00	0,00	0,00
13	----------	----------	32	0,00	0,00	13,61
14	----------0-0--1101100-	----------	0	-9,00	0,00	45,61
15	----------11---1--	----------	0	-3,00	0,00	36,61
16	---0----------unnnnnnnnnnnn	----------u	31	-14,00	-1,00	33,61
17	0--1----------0-0-u111010	----------	32	-11,00	0,00	50,61
18	0--1------1---------uu-uu0001110	----------u	31	-22,34	-9,34	71,61
19	01-unn--nnu----------1---0nn10	----------	0	-17,00	-3,00	80,27
20	n-u0uu001-000----------0-00-10	----------1--00	0	-17,61	-4,61	63,27
21	u-u-0n11----0---------11-01---10	----------u	0	-20,83	-8,83	45,66
22	--1-110---011----------n----11	----------10	0	-10,42	-2,42	24,83
23	--0-0-10----------n-0---0	----------u	0	-7,42	-1,42	14,42
24	--1-1--1----------1----	----------	0	-4,00	0,00	7,00
25	----------0----	----------	0	-2,00	-1,00	3,00
26	----------1----	----------	0	-1,00	0,00	1,00
27	----------	----------	0	0,00	0,00	0,00
28	----------	----------	0	0,00	0,00	0,00
	...	...	...	...		...

Tabel 8.9: Differentieel pad van 45 stappen, versie 1, automatisch berekend, $N_w = 2^{80,27}$

'n' zodra deze voorkomen. Dit wordt zowel voor ∇Q_{t+1} als ∇W_t gedaan. Het beste differentiële pad dat zo gevonden werd, wordt in Tabel 8.9 gegeven, met een werkfactor $N_w = 2^{80,27}$. Merk op dat 'Stadium 1' uit [12], het bepalen van een goede L-karakteristiek, hier niet toegepast kan worden, zoals bleek uit Tabel 6.1.

Analoog aan 'Stadium 3' uit [12], worden lokaal aanpassingen gedaan, door voor elke $\nabla Q_{t+1}[i]$ en $\nabla W_t[i]$ tijdelijk iedere geldige conditie toe te kennen, en dan na te gaan of de werkfactor

t	∇Q_{t+1}	∇W_t	F_W	$P_u(t)$	$P_c(t)$	$N_s(t)$
-5	00001111010010111000011111000011					
-4	01000000110010010101000111011000					
-3	01100010111010110111001111111010					
-2	11101111110011011010101110001001					
-1	01100111010001010010001100000001					
0	----------	----------	32	0,00	0,00	0,00
1	----------	----------	32	0,00	0,00	0,00
2	----------	----------	32	0,00	0,00	0,00
3	----------	----------	32	0,00	0,00	0,00
4	----------	----------	32	0,00	0,00	0,00
5	----------	----------	32	0,00	0,00	0,00
6	----------	----------	32	0,00	0,00	0,00
7	----------	----------10	30	0,00	0,00	0,00
8	----------	----------	32	0,00	0,00	0,00
9	----------	----------	32	0,00	0,00	0,00
10	----------	----------	0	0,00	0,00	0,00
11	----------	----------	32	0,00	0,00	0,00
12	----------	----------	32	0,00	0,00	0,00
13	----------0---0-----	----------	32	-2,00	0,00	18,32
14	----------010001101100-	----------	0	-12,00	0,00	48,32
15	----------001011---1-0	----------	0	-8,00	0,00	36,32
16	---0----------unnnnnnnnnnnn	----------u	31	-14,00	-1,00	28,32
17	0--1----------0100u111010	----------	32	-13,00	0,00	45,32
18	0--1------1---------uu-uu0001110	----------00------u	29	-18,19	-7,19	64,32
19	01-unn--nnu----------11000nn10	----------	0	-18,00	-1,00	75,13
20	n-u0uu001-000----------0-0000-10	----------1-101--00	0	-14,30	-2,30	57,13
21	u-u-0n11----0---------11-010--10	----------u	0	-19,42	-6,42	42,83
22	--1-110---011----------0-n----11	----------10	0	-10,00	-1,00	23,42
23	--0-0-10----------n-0---0	----------00------u	0	-7,42	-1,42	13,42
24	--1-1--1----------1----	----------	0	-4,00	0,00	6,00
25	----------0----	----------	0	-1,00	0,00	2,00
26	----------1----	----------	0	-1,00	0,00	1,00
27	----------	----------	0	0,00	0,00	0,00
28	----------	----------	0	0,00	0,00	0,00
	...	...	...	...	...	...

Tabel 8.10: Differentieel pad van 45 stappen, versie 2, automatisch berekend, $N_w = 2^{75,13}$

verbetert. Is dit het geval, dan wordt deze conditie definitief toegevoegd. Met deze techniek wordt de werkfactor verlaagd tot $N_w = 2^{75,13}$. Het resultaat wordt in Tabel 8.10 gegeven.

Bij het toevoegen van één enkele nieuwe conditie in Tabel 8.10 op eenderwelke plaats, blijft de werkfactor ofwel ongewijzigd, ofwel stijgt de werkfactor. Indien nieuwe condities worden toegevoegd die de werkfactor ongewijzigd laten, of zelfs licht laten stijgen, blijkt het zeer moeilijk om de werkfactor verder te laten dalen. Het is immers juist het verlagen van de werkfactor dat

het zoekproces leidde. Door toe te staan dat de werkfactor ook gelijk mag blijven of stijgen, valt deze doelfunctie weg en wordt het zoekproces willekeurig.

Het is echter wel mogelijk om de werkfactor verder te verlagen. Hiertoe worden condities toegevoegd in de buurt van de botsing, bij voorkeur op plaatsen waar er woordvrijheid is ($F_w > 2^0$), waarbij de werkfactor niet mag stijgen. Verder worden sommige condities terug vrij gemaakt door '`1`' of '`0`' te vervangen door '`-`', indien hierdoor de werkfactor verlaagt. Dit gebeurt bij voorkeur op plaatsen waar er geen woordvrijheid is ($F_w = 2^0$).

Door condities ook terug vrij te maken, kan niet alleen de werkfactor verbeterd worden, maar wordt ook vermeden dat onnodig veel extra condities worden toegevoegd, wat het zoekproces belemmert. Dit blijkt efficiënter te zijn dan te backtracken over de condities die toegevoegd worden. Dit kan via een voorbeeld geïllustreerd worden. Stel dat, indien er zeven condities worden toegevoegd, de werkfactor verlaagt. Een combinatie van drie van deze condities verlagen bijvoorbeeld de werkfactor, terwijl de vier overige condities geen invloed hebben. Door alle zeven condities tegelijkertijd toe te voegen, en dan één voor één te verwijderen, kan snel achterhaald worden welke condities geen invloed hadden op de werkfactor. Dit verloopt sneller dan alle combinaties van één, twee en drie condities toe te voegen: dan moeten in totaal van $C_7^1 + C_7^2 + C_7^3 = 7 + 21 + 35 = 63$ deelverzamelingen van condities toegevoegd worden.

Merk de analogie op met het wiskundige bewijs dat een probleem 'NP-easy' is [31, p. 261-265], zoals bijvoorbeeld het geval is bij het deelverzameling-som probleem. Een probleem wordt 'NP-easy' genoemd als er een Turingreductie bestaat naar een NP-probleem in veeltermtijd. Een NP-probleem is een probleem dat met een niet-deterministisch algoritme (gebruik makend van een orakel) in veeltermtijd kan opgelost worden. Bij het deelverzameling-som probleem wordt, gegeven een verzameling van gehele getallen, gezocht wordt naar een deelverzameling waarvan de som gelijk is aan 0. Indien er een orakel is dat aangeeft of een deelverzameling-som probleem al dan niet een oplossing heeft, kan de deelverzameling zelf in veeltermtijd gevonden worden.

Dit gebeurt door beginnende van de volledige verzameling, systematisch elementen weg te laten en na te gaan of het nieuwe deelverzameling-som probleem nog een oplossing heeft. Indien een element wordt weggelaten dat deel uitmaakte van de oplossingsverzameling van het deelverzameling-som probleem, zal het nieuwe deelverzameling-som dat zo ontstaat immers geen oplossing meer hebben. De analogie bestaat erin dat het algoritme hetzelfde is, indien het orakel wordt vervangen door de berekening van de werkfactor. Deze werkfactor is echter een deterministische berekening, zodat een verdere gelijkenis niet opgaat.

Het toevoegen en verwijderen van condities wordt enkele stappen herhaald. De werkfactor verbetert zo tot $N_w = 2^{49,21}$ in Tabel 8.11. Merk op dat er zowel voor $t = 14$ en $t = 19$ een knelpunt ontstaat met hetzelfde aantal geschatte bladeren dat doorzocht moet worden, $N_s(t) = 2^{48,20}$.

In Tabellen 8.12 en 8.13 wordt een differentieel pad de volledige vereenvoudigde HAS-V gegeven, dat echter meer stappen vereist om tot een botsing te komen dan de verjaardagsaanval uit Paragraaf 2.2 ($N_w = 2^{108,22}$). Dit aantal stapfunctiebewerkingen is ongeveer equivalent aan 2^{103} hashfunctiebewerkingen, meer dus dan $\mathcal{O}(2^{80})$, vereist voor een verjaardagsaanval. Indien er echter bijkomende condities kunnen gevonden worden om de twee botsingsregio's met elkaar te verbinden, is het mogelijk dat een differentieel pad gevonden wordt dat leidt tot een aanval die beter is dan de verjaardagsaanval. Het is echter niet duidelijk hoe deze botsingsregio's op

t	∇Q_{t+1}	∇W_t	F_W	$P_u(t)$	$P_c(t)$	$N_s(t)$
-5	00001111010010111000011111000011					
-4	01000000110010010101000111011000					
-3	01100010111010110111001111111010					
-2	11101111110011011010101110001001					
-1	01100111010001010010001100000001					
0	----------	----------	32	0,00	0,00	0,00
1	----------	----------	32	0,00	0,00	0,00
2	----------	----------	32	0,00	0,00	0,00
3	----------	----------	32	0,00	0,00	0,00
4	----------	----------	32	0,00	0,00	0,00
5	----------	----------	32	0,00	0,00	0,00
6	----------	----------	32	0,00	0,00	0,00
7	----------	000000----------10	24	0,00	0,00	0,00
8	----------	----------	32	0,00	0,00	0,00
9	----------	----------	32	0,00	0,00	0,00
10	----------	----------	0	0,00	0,00	0,00
11	----------0----------	----------	32	-1,00	0,00	0,00
12	----------000--------	----------	32	-3,00	0,00	0,00
13	----------1-00100--0--	----------	32	-7,00	0,00	23,20
14	----------0100011011001	----------	0	-13,00	0,00	48,20
15	---1--0----------0001011011110	----------	0	-15,00	0,00	35,20
16	-1-0--110----------unnnnnnnnnnnn	10----------1-000--0u	24	-17,00	-5,24	20,20
17	01-1--1----------110100u111010	----------0-------	31	-16,00	0,00	27,20
18	00-1-0110011111-----uu1uu0001110	----------01000--1--u	25	-19,00	-1,00	42,20
19	01-unn--nnu1111------1011000nn10	000-00000000000--------000000000	0	-11,00	-7,01	48,20
20	n-u0uu0010000---------0000000-10	000001011----------0101011000	0	-8,24	-4,14	37,20
21	u-u-0n11-1110---------11-010--10	10----------1-000--0u	0	-11,03	-1,00	28,96
22	0-11110---011----------0-n----11	000000----------10	0	-6,46	-0,12	17,93
23	--0-0-10----------n-0---0	----------01000--1--u	0	-5,48	0,00	11,48
24	--1-1--1----------1----	----------	0	-4,00	0,00	6,00
25	----------0----	----------	0	-1,00	0,00	2,00
26	----------1----	----------	0	-1,00	0,00	1,00
27	----------	----------	0	0,00	0,00	0,00
28	----------	----------	0	0,00	0,00	0,00
	...	...	...	...	...	...

Tabel 8.11: Differentieel pad van 45 stappen versie 3, deels automatisch berekend, $N_w = 2^{49,21}$

t	∇Q_{t+1}	∇W_t	F_W	$P_u(t)$	$P_c(t)$	$N_s(t)$
-5	00001111010010111000011111000011					
-4	01000000110010010101000111011000					
-3	01100010111010110111001111111010					
-2	11101111110011011010101110001001					
-1	01100111010001010010001100000001					
0	----------	----------	32	0,00	0,00	0,00
1	----------	----------	32	0,00	0,00	0,00
2	----------	----------	32	0,00	0,00	0,00
3	----------	----------	32	0,00	0,00	0,00
4	----------	----------	32	0,00	0,00	0,00
5	----------	----------	32	0,00	0,00	0,00
6	----------	----------	32	0,00	0,00	0,00
7	----------	000000----------10	24	0,00	0,00	0,00
8	----------	----------	32	0,00	0,00	0,00
9	----------	----------	32	0,00	0,00	0,00
10	----------	----------	0	0,00	0,00	19,21
11	----------0----------	----------	32	-1,00	0,00	19,21
12	----------000--------	----------	32	-3,00	0,00	50,21
13	----------1-0---0--0--	----------	32	-4,00	0,00	79,21
14	----------0100011011001	----------	0	-13,00	0,00	107,21
15	---1--0----------0001011011110	----------	0	-15,00	0,00	94,21
16	-1-0--110----------unnnnnnnnnnnn	10----------1-000--0u	24	-17,01	-5,25	79,21
17	01-1--1----------110100u111010	----------0-------	31	-16,00	0,00	86,20
18	00-1-0110011111-----uu1uu0001110	----------01000--1--u	25	-19,00	-1,00	101,20
19	01-unn--nnu1111------1011000nn10	000-00000000000--------000000000	0	-11,00	-7,01	107,20
20	n-u0uu0010000---------0000000-10	000001011----------0101011000	0	-8,24	-4,14	96,20
21	u-u-0n11-1110---------11-010--10	10----------1-000--0u	0	-11,03	-1,00	87,96
22	0-11110---011----------0-n----11	000000----------10	0	-6,46	-0,12	76,93
23	--0-0-10----------n-0---0	----------01000--1--u	0	-5,48	0,00	70,48
24	--1-1--1----------1----	----------	0	-4,00	0,00	65,00
25	----------0----	----------	0	-1,00	0,00	61,00
26	----------1----	----------	0	-1,00	0,00	60,00
	...	...	...	...	...	...

Tabel 8.12: Differentieel pad van 60 stappen, $N_w = 2^{108,22}$ (wordt vervolgd)

een goede manier met elkaar verbonden kunnen worden. Indien condities aan de randen van beide regio's worden toegevoegd, lijkt de werkfactor immers enkel te verslechteren.

8.11 Besluit

Voor dit hoofdstuk worden de technieken die door C. De Cannière en C. Rechberger in [12, 11] ontwikkeld werden, geïmplementeerd. Hierbij wordt vooral de nadruk gelegd op leesbaarheid en uitbreidbaarheid. Gezien de specifieke eigenschappen van de hashfunctie zich enkel in

t	∇Q_{t+1}	∇W_t	F_W	$P_u(t)$	$P_c(t)$	$N_s(t)$
	...	...	...	...	...	...
27	----------	----------	0	0,00	0,00	59,00
28	----------	----------	0	0,00	0,00	59,00
29	----------	----------	0	0,00	0,00	59,00
30	----------	----------	0	0,00	0,00	59,00
31	----------	----------	0	0,00	0,00	59,00
32	----------	----------0-------	0	0,00	0,00	59,00
33	----------	----------	0	0,00	0,00	59,00
34	----------	000-00000000000--------000000000	0	0,00	0,00	59,00
35	----------	----------	0	0,00	0,00	59,00
36	----------	----------	0	0,00	0,00	59,00
37	----------	----------	0	0,00	0,00	59,00
38	----------	----------	0	0,00	0,00	59,00
39	----------	----------	0	0,00	0,00	59,00
40	----------	----------	0	0,00	0,00	59,00
41	----------	000-00000000000--------000000000	0	0,00	0,00	59,00
43	----------	----------	0	0,00	0,00	59,00
44	----------	000000----------10	0	0,00	0,00	59,00
45	----------	----------	0	0,00	0,00	59,00
46	----------un	----------u	0	-2,00	-1,00	59,00
47	----------un----------u	10----------1-000--0u	0	-3,00	-1,00	57,00
48	---------u----------u---------	----------	0	-2,00	0,00	54,00
49	----u----------un----------	----------	0	-6,00	-3,00	52,00
50	-n---u----------u------	----------	0	-6,00	-3,00	46,00
51	----------u-------n--un-	----------	0	-9,00	-5,00	40,00
52	----------n----------	----------0-------	0	-7,00	-6,00	31,00
53	----------n--	----------	0	-8,00	-7,00	24,00
54	----------	----------	0	-8,00	-8,00	16,00
55	----------	----------	0	-5,00	-5,00	8,00
56	----------	----------u	0	-2,00	-2,00	3,00
57	----------	----------01000--1--u	0	-1,00	-1,00	1,00
58	----------	----------	0	0,00	0,00	0,00
59	----------	----------	0	0,00	0,00	0,00

Tabel 8.13: Differentieel pad van 60 stappen, $N_w = 2^{108,22}$ (vervolg)

initialisatiefuncties bevinden, is het eenvoudig om een andere hashfunctie uit de MD4-familie te gebruiken.

Deze technieken worden verder uitgebreid door onder andere de toevoeging van de dubbelcondities uit Paragraaf 8.5. Het programma wordt opgestart zonder L-karakteristiek door korte botsingen te eisen. Voor de woorden die betrokken zijn bij een exclusieve OF, wordt een algemene techniek ontwikkeld die met alle mogelijke condities uit Tabel 8.1 overweg kan. Indien het algoritme vaak vaalt voor eenzelfde bitpositie, kan de zoekboom herschikt worden om deze bitpositie eerst toe te kennen.

Een differentieel pad (Tabel 8.5) met als werkfactor $N_w = 2^{19,70}$ en een botsing voor 26 stappen (Tabel 8.6) worden gegeven. Deze konden niet met eerdere automatische technieken bekomen worden. Een groot aantal berichtwoorden m_i hebben geen invloed op de botsing en zijn dus vrij te kiezen. Een beter differentieel pad voor 26 stappen en $N_w = 2^{7,70}$. en een bijhorende botsing worden gegeven in Tabellen 8.7 en 8.8.

Een differentieel pad voor 45 stappen wordt gegeven in 8.9 met als werkfactor $N_w = 2^{80,27}$, naar analogie met 'Stadium 2' uit [12]. 'Stadium 1' komt overeen met het zoeken naar een goede L-karakteristiek, wat hier niet toegepast kan worden, zoals blijkt uit Tabel 6.1. Na toepassing van 'Stadium 3' wordt Tabel 8.10 bekomen, met een werkfactor van $N_w = 2^{75,13}$. Door het gebruik van nieuwe technieken, wordt dit differentiële pad verder verbeterd in Tabel 8.11, waarbij de werkfactor tot $N_w = 2^{49,21}$ verlaagd wordt.

In Tabellen 8.12 en 8.13 wordt een differentieel pad voor 60 stappen gegeven, dat gezien de werkfactor $N_w = 2^{108,22}$ geen aanval oplevert die beter is dan de generische verjaardagsaanval uit Paragraaf 2.2. Het is echter wel mogelijk dat toekomstige technieken erin slagen om deze werkfactor verder te verlagen om zo toch een bruikbare aanval te bekomen. Bij de methode van H. Dobbertin bij MD5 [14, 10] worden de botsingsregio's met elkaar verbonden door gebruik te maken van de woordvrijheid die overblijft gezien botsingen zeer kort zijn. Dit wordt echter bemoeilijkt bij de vereenvoudigde HAS-V door de XOR-woorden: iedere bit van een berichtwoord m_i heeft invloed zo op zes bits in geëxpandeerde berichtwoorden W_t, die verspreid liggen over de volledige berichtexpansie.

Hoofdstuk 9

Besluit

9.1 Oplossingsmethoden en resultaten

Zoals beschreven in Hoofdstuk 3, bestaat de hashfunctie HAS-V uit twee parallelle stromen. Er wordt een variant opgesteld met één stroom, waarop alle verdere cryptanalyse zich baseert.

Recente technieken van X. Wang, beschreven in Hoofdstuk 4, worden dan gebruikt om te zoeken naar een botsing voor deze vereenvoudigde HAS-V. Hierbij wordt een berichtverschil en een differentieel pad geconstrueerd. Via enkelvoudige berichtmodificatie of geavanceerdere technieken kan dan een botsing bekomen worden.

In Hoofdstuk 5 wordt voor deze vereenvoudigde HAS-V eerst gezocht naar lokale botsingen, om zo de techniek van F. Chabaud en A. Joux voor SHA-0 [5] toe te passen. Deze methode wordt verder uitgebreid, door niet één, maar alle mogelijke lineaire benaderingen voor de booleaanse functies f_j tegelijkertijd in rekening te brengen bij het zoeken naar een verzameling van lokale botsingen die voldoen aan de berichtexpansie. Bij deze techniek bleek de variabele rotatie S_t en de structuur van de berichtexpansie ervoor te zorgen dat geen goede differentiële paden konden gevonden worden voor meer dan twee ronden.

Het aantal verschillende bits in de interne toestanden is echter een betere maat voor het differentiële pad dan het aantal lokale botsingen. Om deze differentiële paden te zoeken, wordt een techniek gebruikt in Hoofdstuk 6, die gelijkaardig is aan deze voor de cryptanalyse van SHA-1 door V. Rijmen en E. Oswald in [37]. Daar worden deze differentiële paden L-karakteristieken genoemd. Er wordt gebruik gemaakt van één lineaire benadering voor iedere booleaanse functie f_j, de benadering die zoveel mogelijk invoerverschillen opslorpt. Dit is een verschil met de uitbreiding van de methode van F. Chabaud en A. Joux, waar alle lineaire benaderingen tegelijkertijd in rekening gebracht worden.

Hoewel er betere differentiële paden gevonden worden dan met de vorige techniek, is het ook hier niet mogelijk om goede differentiële paden te vinden voor meer dan twee ronden. De variabele rotatie S_t en de structuur van de berichtexpansie blijken hier eveneens voor problemen te zorgen.

Bij deze technieken werd de hashfunctie telkens gelineariseerd. De optelling modulo 2^{32} werd telkens als een exclusieve OF benaderd. Er wordt dus verondersteld dat er zich geen verschillen

in de overdrachtsbits voordoen. Bij de differentiële paden die gebruikt worden voor MD5 [44] en SHA-1 [12] is dit echter wel het geval. Door het gebruik van niet-lineariteiten, worden deze differentiële paden NL-karakteristieken genoemd. Een techniek van M. Stevens die succesvol werd gebruikt voor MD5 [40], wordt beschreven en aangepast voor de vereenvoudigde HAS-V in Hoofdstuk 7. Het lukte echter niet om deze techniek te gebruiken om differentiële paden voor deze vereenvoudigde HAS-V te construeren.

Een differentieel pad voor 45 stappen werd wel bekomen via handmatige constructie, zie Tabel 7.2. Hierbij werd de berichtexpansie evenwel licht gewijzigd.

Een potentieel geheugenprobleem bij een aanval gebaseerd op methoden van M. Stevens, kan echter verholpen worden door de mogelijkheden voor (Q_{t+1}, Q'_{t+1}) niet expliciet op te slaan. Hiervoor kan een grafe gebruikt worden, zoals bij de methode van C. De Cannière en C. Rechberger in [12, 11]. De mogelijke (Q_{t+1}, Q'_{t+1}) worden daar gegeven door alle mogelijke paden doorheen de grafe. Deze methode wordt beschreven, verder uitgebreid en toegepast op de vereenvoudigde HAS-V in Hoofdstuk 8.

Bij de implementatie van deze techniek, wordt de nadruk gelegd op uitbreidbaarheid. Zo is het eenvoudig om de hashfunctie te vervangen door een andere hashfunctie uit de MD4-familie. Via onder andere de dubbelcondities uit Paragraaf 8.5 wordt de techniek van C. De Cannière en C. Rechberger uitgebreid. Een L-karakteristiek is niet nodig om het programma te starten, indien geëist wordt dat de botsingen zeer kort zijn.

Een differentieel pad voor 26 stappen en een bijhorende botsing worden gegeven in Tabellen 8.7 en 8.8 met $N_w = 2^{7,70}$. Deze konden niet bekomen worden met eerdere automatische technieken. Een groot aantal berichtwoorden m_i hebben geen invloed op de botsing en zijn dus vrij te kiezen.

Een differentieel pad voor 45 stappen wordt gegeven Tabel 8.11, met een werkfactor $N_w = 2^{49,21}$. Deze werd bekomen door de technieken uit [12, 11] toe te passen en verder uit te breiden. In Tabellen 8.12 en 8.13 wordt een differentieel pad voor 60 stappen gegeven, dat gezien de werkfactor $N_w = 2^{108,22}$ geen aanval oplevert die beter is dan de generische verjaardagsaanval uit Paragraaf 2.2. Het is echter wel mogelijk dat verdere technieken erin slagen om deze werkfactor verder te verlagen om zo toch een bruikbare aanval te bekomen.

9.2 Suggesties voor verder onderzoek

De vereenvoudigde HAS-V telt in totaal 60 stappen. Indien een goede karakteristiek hiervoor gevonden kan worden, leidt dit tot botsingen voor de volledige vereenvoudigde HAS-V. Een aanzet hiervoor wordt gegeven in Tabel 8.12. Een verdere uitbreiding kan ook rekening houden met beide stromen van HAS-V, om zo een botsing te bekomen voor de echte hashfunctie in plaats van voor een vereenvoudigde variant.

Hoewel in deze tekst enkel botsingen besproken worden, is het mogelijk dat nieuwe technieken er ook in slagen om een (tweede) invers beeld te zoeken. Hierdoor wordt een totaal andere klasse van aanvallen mogelijk. Een aanval om een tweede invers beeld te vinden werd door X. Wang evenwel beschreven in [45] voor MD4.

Resultaten die hier werden voorgesteld, kunnen eveneens gebruikt worden bij het ontwerpen

van een nieuwe hashfunctie. Bij de vereenvoudigde HAS-V wordt immers verduidelijkt welke elementen zorgen voor het al dan niet slagen van bepaalde aanvallen. Doordat de klemtoon hier werd gelegd op geautomatiseerde technieken, is het eenvoudig om deze nieuwe hashfunctie te testen tegen de beschreven aanvallen. Dit kan zeker van pas komen bij de open competitie van NIST voor de nieuwe hashfunctiestandaard SHA-3 [33].

Bij de cryptanalyse van HAS-V wordt er gebruik gemaakt van technieken voor SHA-1 [12], die echter niet via een L-karakteristiek worden opgestart. Dit kan mogelijk helpen bij de cryptanalyse van RIPEMD-160 [15], waarvoor er evenmin goede L-karakteristieken gevonden werden [26].

Bibliografie

[1] S. E. Anderson, "Counting bits set, in parallel," http://www.cs.utk.edu/~vose/c-stuff/bithacks.html.

[2] E. Biham and R. Chen, "Near-Collisions of SHA-0," in *CRYPTO*, ser. Lecture Notes in Computer Science, M. K. Franklin, Ed., vol. 3152. Springer, 2004, pp. 290–305.

[3] G. Brassard, Ed., *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, ser. Lecture Notes in Computer Science, vol. 435. Springer, 1990.

[4] R. Brummayer and A. Biere, "C32SAT: Checking C Expressions," in *CAV*, ser. Lecture Notes in Computer Science, W. Damm and H. Hermanns, Eds., vol. 4590. Springer, 2007, pp. 294–297.

[5] F. Chabaud and A. Joux, "Differential Collisions in SHA-0," in *CRYPTO*, ser. Lecture Notes in Computer Science, H. Krawczyk, Ed., vol. 1462. Springer, 1998, pp. 56–71.

[6] H.-S. Cho, S. Park, S. H. Sung, and A. Yun, "Collision Search Attack for 53-Step HAS-160," in *ICISC*, ser. Lecture Notes in Computer Science, M. S. Rhee and B. Lee, Eds., vol. 4296. Springer, 2006, pp. 286–295.

[7] R. Cramer, Ed., *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, ser. Lecture Notes in Computer Science, vol. 3494. Springer, 2005.

[8] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[9] I. Damgård, "A Design Principle for Hash Functions," in *CRYPTO*, ser. Lecture Notes in Computer Science, G. Brassard, Ed., vol. 435. Springer, 1989, pp. 416–427.

[10] M. Daum, "Cryptanalysis of Hash Functions of the MD4-family," Ph.D. dissertation, Ruhr-Universität Bochum, 2005.

[11] C. De Cannière, Persoonlijke communicatie, December 2007.

[12] C. De Cannière and C. Rechberger, "Finding SHA-1 Characteristics: General Results and Applications," in *ASIACRYPT*, ser. Lecture Notes in Computer Science, X. Lai and K. Chen, Eds., vol. 4284. Springer, 2006, pp. 1–20.

[13] R. Dhamija, J. D. Tygar, and M. A. Hearst, "Why phishing works," in *CHI*, R. E. Grinter, T. Rodden, P. M. Aoki, E. Cutrell, R. Jeffries, and G. M. Olson, Eds. ACM, 2006, pp. 581–590.

[14] H. Dobbertin, "Cryptanalysis of MD5," Presented at a rump session of EuroCrypt '96, 1996.

[15] H. Dobbertin, A. Bosselaers, and B. Preneel, "RIPEMD-160: A Strengthened Version of RIPEMD," in *FSE*, ser. Lecture Notes in Computer Science, D. Gollmann, Ed., vol. 1039. Springer, 1996, pp. 71–82.

[16] W. Feller, *An Introduction to Probability Theory and Its Applications, Volume 1.* Wiley, 1968.

[17] M. Gebhardt, G. Illies, and W. Schindler, "A Note on the Practical Value of Single Hash Collisions for Special File Formats," in *Sicherheit*, ser. LNI, J. Dittmann, Ed., vol. 77. GI, 2006, pp. 333–344.

[18] I. N. W. Group, "The Transport Layer Security (TLS) Protocol," http://tools.ietf.org/html/rfc4346, 2006.

[19] P. Hawkes, M. Paddon, and G. G. Rose, "Musings on the Wang et al. MD5 Collision," Cryptology ePrint Archive, Report 2004/264, 2004, http://eprint.iacr.org/.

[20] S. Indesteege, Persoonlijke communicatie, Februari 2008.

[21] A. Joux and T. Peyrin, "Hash functions and the (amplified) boomerang attack," in *CRYPTO*, ser. Lecture Notes in Computer Science, A. Menezes, Ed., vol. 4622. Springer, 2007, pp. 244–263.

[22] V. Klima, "Tunnels in Hash Functions: MD5 Collisions Within a Minute," Cryptology ePrint Archive, Report 2006/105, 2006.

[23] A. K. Lenstra, "Further progress in hashing cryptanalysis," http://cm.bell-labs.com/who/akl/hash.pdf, 2005.

[24] G. Leurent, "Message Freedom in MD4 and MD5 Collisions: Application to APOP," in *FSE*, ser. Lecture Notes in Computer Science, A. Biryukov, Ed., vol. 4593. Springer, 2007, pp. 309–328.

[25] C. H. Lim and P. J. Lee, "A Study on the Proposed Korean Digital Signature Algorithm," in *ASIACRYPT*, ser. Lecture Notes in Computer Science, K. Ohta and D. Pei, Eds., vol. 1514. Springer, 1998, pp. 175–186.

[26] F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen, "On the Collision Resistance of RIPEMD-160," in *ISC*, ser. Lecture Notes in Computer Science, S. K. Katsikas, J. Lopez, M. Backes, S. Gritzalis, and B. Preneel, Eds., vol. 4176. Springer, 2006, pp. 101–116.

[27] F. Mendel and V. Rijmen, "Colliding Message Pair for 53-Step HAS-160," in *ICISC*, ser. Lecture Notes in Computer Science, K.-H. Nam and G. Rhee, Eds., vol. 4817. Springer, 2007, pp. 324–334.

[28] ——, "Weaknesses in the HAS-V Compression Function," in *ICISC*, ser. Lecture Notes in Computer Science, K.-H. Nam and G. Rhee, Eds., vol. 4817. Springer, 2007, pp. 335–345.

[29] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography.* CRC Press, 1996.

[30] R. C. Merkle, "A Certified Digital Signature," in *CRYPTO*, ser. Lecture Notes in Computer Science, G. Brassard, Ed., vol. 435. Springer, 1989, pp. 218–238.

[31] B. M. Moret, *The Theory of Computation.* Addison-Wesley Longman Publishing Co., Inc., 1998.

[32] K.-H. Nam and G. Rhee, Eds., *Information Security and Cryptology - ICISC 2007, 10th International Conference, Seoul, Korea, November 29-30, 2007, Proceedings*, ser. Lecture Notes in Computer Science, vol. 4817. Springer, 2007.

[33] N. I. of Standards and Technology, "Cryptographic Hash Algorithm Competition," http://www.nist.gov/hash-competition.

[34] N. K. Park, J. H. Hwang, and P. J. Lee, "HAS-V: A New Hash Function with Variable Output Length," in *Selected Areas in Cryptography*, ser. Lecture Notes in Computer Science, D. R. Stinson and S. E. Tavares, Eds., vol. 2012. Springer, 2000, pp. 202–216.

[35] B. Preneel, "Analysis and design of cryptographic hash functions," Ph.D. dissertation, Katholieke Universiteit Leuven, 1993.

[36] G. W. Reitwiesner, "Binary arithmetic," *Advances in Computers*, vol. 1, pp. 231–308, 1960.

[37] V. Rijmen and E. Oswald, "Update on SHA-1," in *CT-RSA*, ser. Lecture Notes in Computer Science, A. Menezes, Ed., vol. 3376. Springer, 2005, pp. 58–71.

[38] F. I. P. Standards, "FIPS 186-2: Digital Signature Standard (DSS)," http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf, 2000.

[39] M. Stevens, A. K. Lenstra, and B. de Weger, "Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, M. Naor, Ed., vol. 4515. Springer, 2007, pp. 1–22.

[40] M. M. J. Stevens, "On Collisions for MD5," Master's thesis, Eindhoven University of Technology, 2007.

[41] D. Wagner, "The Boomerang Attack," in *FSE*, ser. Lecture Notes in Computer Science, L. R. Knudsen, Ed., vol. 1636. Springer, 1999, pp. 156–170.

[42] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu, "Cryptanalysis of the Hash Functions MD4 and RIPEMD," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, R. Cramer, Ed., vol. 3494. Springer, 2005, pp. 1–18.

[43] X. Wang, Y. L. Yin, and H. Yu, "Finding Collisions in the Full SHA-1," in *CRYPTO*, ser. Lecture Notes in Computer Science, V. Shoup, Ed., vol. 3621. Springer, 2005, pp. 17–36.

[44] X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, R. Cramer, Ed., vol. 3494. Springer, 2005, pp. 19–35.

[45] H. Yu, G. Wang, G. Zhang, and X. Wang, "The Second-Preimage Attack on MD4," in *CANS*, ser. Lecture Notes in Computer Science, Y. Desmedt, H. Wang, Y. Mu, and Y. Li, Eds., vol. 3810. Springer, 2005, pp. 1–12.

[46] A. Yun, S. H. Sung, S. Park, D. Chang, S. Hong, and H.-S. Cho, "Finding Collision on 45-Step HAS-160," in *ICISC*, ser. Lecture Notes in Computer Science, D. Won and S. Kim, Eds., vol. 3935. Springer, 2005, pp. 146–155.