Faculteit
Ingenieurswetenschappen

Departement
Elektrotechniek – ESAT

KATHOLIEKE
UNIVERSITEIT
LEUVEN

Illustratieve software voor symmetrische cryptografie

Eindwerk voorgedragen tot het behalen van het diploma van Burgerlijk werktuigkundig-elektrotechnisch ingenieur, richting elektrotechniek, optie multimedia en signaalverwerking

Igor Jacques

Promotor:
Prof. Dr. Ir. Bart Preneel
Dagelijkse begeleiding:
Ing. Jan Cappaert
Ir. Sebastiaan Indesteege

2007 – 2008

Inhoudsopgave

Voorwoord

Tijdens de ontwikkeling van mijn programma EduCrypt heb ik veel programmatorische ervaring opgedaan. Het werken met C++ en later Qt ging in het begin niet altijd even vlot. Hoewel mijn thesis veel weggeeft van een thesis aan het departement computerwetenschappen, heb ik tijdens mijn thesis een vollediger beeld van de cryptografie gekregen. Terwijl de werking van symmetrische algoritmes zoals DES en AES vorig jaar nog redelijk abstract was voor mij, zijn de verschillende onderdelen van deze algoritmes nu veel duidelijker. Hoewel cryptanalyse niet echt aan bod kwam in deze thesis, heb ik nu ook een concreet beeld van wat met deze wetenschap precies bedoeld wordt.

Bij deze wou ik mijn promotor Bart Preneel, alsook mijn dagelijkse belegeiders bedanken voor de ervaringen die ik heb opgedaan het afgelopen jaar. Mede dankzij de hulp van Sebastiaan Indeestege heb ik de C++-taal grondig leren kennen. Bij deze bedank ik ook Jan Cappaert, mijn andere dagelijkse begeleider, die mij net als Sebastiaan geholpen heeft met tal van suggesties en correcties ter verbetering van mijn programma.

Naast mijn thesisbegeleiders wil ik ook Marijke Bouvyn bedanken voor het nalezen van deze thesistekst.

Abstract

In deze thesis wordt een nieuw educatief programma voorgesteld. Het programma is ontworpen om de gebruiker beter vertrouwd te maken met symmetrische cryptografische operaties. Met het voorgestelde programma kan de gebruiker alle mogelijke symmetrische cryptografische cijfers aanmaken. Dit gebeurt op interactieve wijze, in een grafische omgeving, die zowel met het toetsenbord als de muis bediend kan worden. Een cryptografisch algoritme wordt er voorgesteld door een netwerk van blokken die onderling verbonden worden. Onder de verschillende blokken onderscheiden we ingangen, uitgangen en operaties. De verbindingen bepalen de doorstroming van de bitwoorden doorheen het operatienetwerk. Het is mogelijk berekeningen uit te voeren doorheen de aangemaakte operatienetwerken. Het inkapselen van operatienetwerken is ondersteund. Men kan operatienetwerken, alsook sommige operaties, opslaan om ze later terug in te laden. Het gebruikte bestandsformaat is gebaseerd op XML. Dit formaat is universeel, waardoor men opgeslagen netwerken of operaties zowel in een Linux- als een Windows-omgeving kan gebruiken.

Het programma werd geschreven in de programmeertaal C++. Voor de gebruikersinterface werd de toolkit Qt van Trolltech gebruikt. Het programma werd op objectgerichte wijze geschreven, waardoor het toevoegen van nieuwe functionaliteit geen probleem kan vormen. Het programma heeft een introductiehandleiding, deze bevindt zich in bijlage D.

In het eerste deel van de thesistekst wordt ingegaan op de structuur waarin een operatienetwerk intern wordt voorgesteld. De belangrijkste ontwerpbeslissingen worden verdedigd, het algoritme waarmee de berekeningen doorheen een operatienetwerk gebeuren komt aan bod. Ook het opslaan en inladen van operatienetwerken en operaties wordt besproken. In een volgend deel wordt de grafische gebruikersinterface besproken. Dit gebeurt vanuit het standpunt van de gebruiker, er is vooral aandacht voor de functionaliteit in het werkblad. Vervolgens worden nog een aantal operaties en cryptografische algoritmes beschreven. Deze algoritmes maken standaard deel uit van het programma en illustreren het gebruik van de verschillende operaties, alsook de mogelijkheden tot inkapseling. Tot slot worden enkele mogelijke uitbreidingen voorgesteld. Er wordt gedacht aan extra functionaliteit om eenvoudige differentiële cryptanalyses te illustreren of aan een uitbreiding die het mogelijk moet maken om op basis van operatienetwerken volautomatisch implementaties te genereren in programmeertalen als C en VHDL.

Hoofdstuk 1

Inleiding

In deze thesis wordt een nieuw educatief programma besproken. Het programma wordt in deze inleiding eerst geplaatst in zijn kennisdomein, namelijk de cryptografie. Vervolgens wordt toegelicht vanuit welke context het idee voor deze thesis is ontstaan. Nadien wordt in grote lijnen overlopen welke mogelijkheden de ontwikkelde software allemaal biedt. Eventuele uitbreidingen komen kort aan bod. Tot slot wordt ingegaan op de structuur van deze thesis.

Deze thesis behoort tot het kennisdomein van de cryptografie. Dit domein beoogt typisch een bepaalde boodschap om te vormen tot een onleesbaar geheel, gebruikmakende van een publiek algoritme dat gebruik maakt van een geheime code. Dit omvormen noemt men meestal encrypteren en de naamgeving 'geheime code' wordt meestal vervangen door 'sleutel'. Andere cryptografische technieken maken het mogelijk digitale handtekeningen te creëren of te controleren of bepaalde informatie werd gewijzigd, bijvoorbeeld na transmissie. In de praktijk is ook het protocol waarin een algoritme wordt aangewend zeer belangrijk, maar daarop wordt hier niet verder ingegaan.

Men kan in de cryptografie twee domeinen onderscheiden: de symmetrische en de asymmetrische cryptografie. Toepassingen die gebruik maken van symmetrische cryptografie encoderen en decoderen informatie met dezelfde sleutel, het encoderings- en decoderingsproces zijn elkaars inverse. De gebruikte operaties zijn meestal eenvoudig; logische operaties worden veelvuldig gebruikt. Bij asymmetrische cryptografie verloopt het decoderingsproces anders dan het encoderingsproces. Bovendien wordt asymmetrische cryptografie vaak geassocieerd met moeilijke wiskundige operaties, in tegenstelling tot de eenvoudige operaties uit de symmetrische cryptografie. Crypografische algoritmes die niet encrypteren, zoals bijvoorbeeld hash-functies, worden meestal tot de symmetrische cryptografie gerekend.

Beide methodieken verschillen dus grondig. Bovendien zijn er grote verschillen tussen de technieken waarmee beide geanalyseerd worden. Cryptografische algoritmes worden geanalyseerd om te onderzoeken of ze aan bepaalde veiligheidscriteria voldoen. Er zijn verschillende technieken om cryptografische algoritmes te analyseren. De differentiële

cryptanalyse is een veel gebruikte techniek. Hier worden verschillen van bits doorheen het cijfer bestudeerd in een statistische context. Een andere methodiek is de lineaire cryptanalyse. Deze techniek probeert een cijfer of een onderdeel ervan te benaderen in een lineaire ruimte.

De cryptografische wereld is net zoals andere sterk wetenschappelijke kennisdomeinen niet zo toegankelijk voor derden. Vanuit een wiskundige achtergrond bedenken wetenschappers nieuwe encryptiemethodes, die ze vervolgens analyseren op bepaalde eigenschappen. Mensen die interesse tonen in dit vakgebied zijn meestal aangewezen op papers en boeken. Sommige boeken zijn heel informatief en bespreken een verzameling van cryptografische algoritmes en hun eigenschappen. Andere literatuur is dan weer heel mathematisch en alleen geschikt voor ingewijden. Hoewel beide soorten literatuur elkaar aanvullen, is er toch een kloof tussen beide.

Met deze thesis wordt getracht een deel van deze kloof op te vullen. Deze thesis omvat het ontwerp en de implementatie van een educatief programma. Dit programma geeft de gebruiker de kans de in de literatuur besproken symmetrische cijfers zelf na te maken en ze zelf uit te proberen. Uit opzoekingswerk blijkt dat een dergelijk programma nog niet publiek beschikbaar is. Er bestaan wel veel programma's die gebruik maken van cryptografie. In de eerste plaats denken we aan encryptieprogramma's zoals PGP [1], maar ook gewone programma's zoals browsers die in hun beveiligingsprotocollen cryptografische algoritmes gebruiken. Programma's waarmee men een symmetrisch cryptografisch algoritme kan simuleren en vervolgens alle tussenresulaten bestuderen zijn echter nog niet publiek beschikbaar.

Het programma beperkt zich tot de symmetrische cryptografische operaties. Deze beperking kwam tot stand omwille van volgende redenen. Er bestaan zeer grote verschillen tussen de rekenkundige operaties gebruikt in de symmetrische en de symmetrische cryptografie. Het maken van een programma dat beide types operaties aankan is zeker mogelijk, maar programmatorisch zouden slechts weinig routines herbruikt kunnen worden. De keuze voor symmetrische algoritmes laat ook toe om later eenvoudige plugins voor cryptanalyse aan het programma toe te voegen.

Het voorgestelde programma laat toe gelijk welk symmetrisch algoritme aan te maken. Het aanmaken van het algoritme gebeurt grafisch, in een werkblad. De gebruiker plaatst de ingangen, uitgangen en operatieblokken in een werkblad en verbindt ze op de gewenste manier. Voorbeelden van operatieblokken zijn permutaties en logische operaties. Het eindresultaat is een netwerk van operaties. Dit alles gebeurt interactief. De gebruiker kan de verschillende eenheden verplaatsen, kopiëren en verbinden met de muis. Ook kunnen bestaande algoritmes die standaard in het programma aanwezig zijn, worden ingeladen. Het is bovendien mogelijk om operatienetwerken in te kapselen.

Het gemaakte algoritme kan worden uitgeprobeerd door bepaalde reeksen van bits, bitwoorden genoemd, aan de ingangen aan te leggen en vervolgens de uitgangen te berekenen. Niet alleen de bitwoorden aan de uitgang zijn beschikbaar voor de gebruiker, ook tussenresultaten overal in het operatienetwerk kunnen bestudeerd worden. Hoewel de

gebruiker zelf verantwoordelijk is voor de consistentie van een operatienetwerk, voorziet het programma de nodige feedback indien bepaalde operaties op inconsistente wijze zijn verbonden. Een aangemaakt operatienetwerk kan worden opgeslagen in een XML-document, om het dan later terug te kunnen inladen.

De ontwikkelde software heeft niet als doel een cryptograaf te helpen bij het ontwerpen van nieuwe cijfers. De rol van het programma is louter educatief. Centraal staat het beter leren begrijpen van symmetrische algoritmes. Het programma kan tevens gebruikt worden om bepaalde varianten op bestaande algoritmes te tekenen en te simuleren. Dit alles heeft uiteraard niet de bedoeling het algoritme daadwerkelijk te verbeteren, maar eerder om bepaalde inzichten te verkrijgen. Het is ook belangrijk op te merken dat de operatienetwerken van de standaard aanwezige algoritmes meestal niet overeen stemmen met de implementatiestructuren die in de praktijk gebruikt worden. De standaard aanwezige algoritmes werden opgebouwd met als doel de structuur van het algoritme duidelijk weer te geven, er werd geen aandacht besteed aan efficiëntie, hergebruik van geheugen of mogelijk parallellisme. De Advanced Encryption Standard [2], of kortweg AES, wordt in de praktijk vaak geïmplementeerd in combinatie met opzoekingstabellen. Vanuit didactisch oogpunt is het echter niet zo nuttig AES voor te stellen gebruikmakende van opzoekingstabellen.

Het programma heeft de naam EduCrypt gekregen. Dit staat voor Educatieve Cryptografie. Het is ontworpen met aandacht voor mogelijke uitbreidingen. Het werd op een objectgerichte manier ontworpen in C++ en er werd veel aandacht besteed aan het ontwerp. Voor de grafische gebruikersinterface werd gebruikt gemaakt van Qt [3], een software toolkit van Trolltech [4]. Mogelijke uitbreidingen zijn het toevoegen van eenvoudige cryptanalyse-plugins, alsook het genereren van een implementatie in C of VHDL [5] op basis van een getekend operatienetwerk. Ook andere mogelijke uitbreidingen worden in de thesistekst besproken.

In deze paragraaf wordt de inhoud van deze thesistekst kort toegelicht. In het eerste hoofdstuk wordt een probleemstelling geformuleerd. Er wordt gespecifieerd aan welke vereisten het programma moet voldoen. Vervolgens wordt het ontwerp besproken van de klasses waarmee een operatienetwerk intern wordt bijgehouden. De belangrijkste beslissingen worden toegelicht. Daarna wordt uitgelegd hoe berekeningen doorheen een operatienetwerk gebeuren. In het vierde hoofdstuk wordt de in hoofdstuk drie ontworpen structuur uitgebreid met klasses voor grafische informatie. Er is er ook aandacht voor de grafische voorstelling van de verschillende elementen in een operatienetwerk. Daarna wordt uitgelegd hoe operatienetwerken, alsook sommige operaties, in een XML-document worden opgeslagen. In het zevende hoofdstuk wordt de werking van de grafische gebruikersinterface besproken. De verschillende onderdelen van het programma worden toegelicht. De nadruk ligt op de gebruiksvriendelijke bediening van het programma. In het achtste hoofdstuk worden enkele algoritmes en operaties besproken die deel uitmaken van het programma. Deze algoritmes en operaties kunnen ten allen tijde worden gebruikt. Daarna worden enkele mogelijke uitbreidingen van het programma voorgesteld. In het laatste hoofdstuk wordt een algemeen besluit geformuleerd.

Hoofdstuk 2

Achtergrond

2.1 Inleiding

In dit hoofdstuk wordt eerst uitgelegd in welke context de ontwikkelde software nuttig kan zijn. Dit gebeurt in sectie 2.2. In sectie 2.3 worden de vereisten voor de software besproken. In een laatste sectie wordt een besluit geformuleerd.

2.2 Probleemstelling

Voor educatieve doeleinden in de cryptografische wereld zijn boeken het meeste gebruikte medium. In didactisch verantwoorde boeken over cryptografische cijfers wordt een cijfer meestal in detail voorgesteld met behulp van schematische voorstellingen. Vervolgens worden enkele eigenschappen van het cijfer kort besproken. Vaak is er aandacht voor enkele interessante cryptanalytische opmerkingen. Dit is onder andere het geval in [6] en [7]. Voor de modale lezer houdt het opzoekingswerk over cijfers vaak op na het lezen van enkele van deze boeken. In de bibliotheek of op het internet zal de lezer tevens gespecialiseerde literatuur vinden. In deze boeken of papers worden cijfers geanalyseerd of worden meestal moeilijke aanvallen voorgesteld. Deze literatuur schrikt vele lezers af, omdat de kandidaatlezers nog onvoldoende vertrouwd zijn met de werking van cijfers en de eenvoudige cryptanalytische principes. Er is dus een didactische kloof tussen de inleidende boeken en de meer gespecialiseerde literatuur.

Deze kloof is louter didactisch. Het is dus niet zo dat bepaalde kennis om de omschakeling naar gespecialiseerde literatuur te maken, ontbreekt. De genoemde kloof kan worden gedicht door actief te oefenen en wordt in het universitair onderwijs typisch gedicht tijdens oefenzittingen. Het is echter niet vanzelfsprekend oefenzittingen over cijfers te beperken tot oefeningen op papier. Indien men cijfers aan het werk ziet en bepaalde statistische eigenschappen zelf ondervindt, zal men bepaalde inzichten sneller verwerven.

Deze thesis speelt in op deze behoefte. In deze thesis wordt een bespreking gegeven van het ontwerp en de werking van een grafische gebruikersinterface om symmetrische cryptografische cijfers te tekenen en er berekeningen mee uit te voeren. Na een berekening zijn alle tussenresultaten voor handen. Reeds bestaande cijfers zoals AES en SHA-1 kunnen worden ingeladen. Bovendien is de ontwikkelde software klaar voor uitbreidingen.

Dat dergelijke educatieve programma's nog niet bestaan, heeft waarschijnlijk de maken met de beperkte doelgroep. De modale multimedia-ingenieur kan echter prima gebruik maken van dit eenvoudige programma, al is het maar één enkele keer. Hij kan er zijn kennis over symmetrische cryptografie mee concretiseren.

Het programma is niet gemaakt om data te encrypteren. Het programma kan daarvoor wel gebruikt worden, maar er bestaan professionele programma's om dit te realiseren. Hierbij denken we bijvoorbeeld aan PGP of browsers die het SSL-protocol gebruiken. Bovendien bieden de meeste van deze programma's betere veiligheidsgaranties en behoeven de gebruikers weinig kennis om ze te gebruiken.

De concrete invulling van het programma wordt in de volgende sectie besproken.

2.3 Ontwerpvereisten

Het programma moet aan een aantal voorwaarden voldoen. In de inleiding werd een algemeen beeld gegeven van de functionaliteit van het programma. Dit wordt nu geconcretiseerd in enkele duidelijke vereisten. Er is ook aandacht voor toekomstige uitbreidingen.

De vereisten worden hieronder opgesplitst in een aantal groepen. Vervolgens worden ze verder gespecifieerd.

Algemeen

- De gebruiker beschikt over een werkblad waarin hij een operatienetwerk kan tekenen.
- De gebruiker heeft de keuze uit de belangrijkste in de symmetrische cryptografie gebruikte operaties. De meeste symmetrische cryptografische algoritmes kunnen getekend worden. Naast de operatieblokken zijn er ook ingangsblokken en uitgangsblokken.
- Inkapseling moet mogelijk zijn. Het moet dus mogelijk zijn een zogenaamde zwarte doos in te voeren, die de functionaliteit van een reeds bestaand operatienetwerk omvat.

Gebruiksvriendelijkheid

- De basisvoorzieningen inzake opslaan zijn voorzien. Indien de gebruiker een nieuw document wil openen of het programma wil sluiten, krijgt hij de kans om zijn werk

op te slaan.

- De gebruiker kan het operatienetwerk met de muis tekenen. Verbindingslijnen kunnen met de muis worden getekend. Blokken kunnen geselecteerd, versleept en gekopieerd worden.
- De gebruiker kan het operatienetwerk dat is ingekapseld in een zwarte doos op eenvoudige wijze opvragen, zodat dit ingekapseld operatienetwerk verschijnt in het werkblad.
- Instellingen van blokken kunnen opgeroepen worden door dubbel te klikken op het blok.
- De gebruiker kan in een invoerbalk een bitwoord ingeven.
- De gebruiker kan het bitwoord horende bij een bepaalde verbinding te allen tijde bekijken.

Berekeningen

- De gebruiker kan zelf gekozen bitwoorden aan de ingangen aanleggen. Daarna kan hij een berekening starten. Nadien zijn alle uitgangen en alle tussenresultaten beschikbaar.
- Indien een berekening vastloopt omdat het getekende operatienetwerk niet consistent is, wordt de gebruiker geïnformeerd over de aard van de fout en waar ze zich bevindt.
- Na berekening kan de gebruiker het aantal bits dat hoort bij een connectielijn bekijken.

Opslag

- De gebruiker kan een operatienetwerk opslaan naar een extern bestand. Alle nodige parameters die constant zijn in het operatienetwerk moeten mee worden opgeslagen naar dit extern bestand. De gebruiker kan het gemaakte operatienetwerk terug inladen of gebruiken in een zwarte doos in een ander operatienetwerk.

Comptabiliteit

- De gebruiker kan het programma zowel in een Linux-omgeving, als een Windows-omgeving gebruiken.
- De in een Linux-omgeving opgeslagen bestanden zijn ook bruikbaar in een Windows-omgeving en omgekeerd.

Uitbreidbaarheid

- Het programma is op een objectgerichte manier ontworpen, zodat het programma uitbreidbaar is. Het moet vrij duidelijk zijn hoe een nieuwe module kan worden toegevoegd. Om deze uitbreidingen toe te laten moeten hoogstens enkele methodes worden toegevoegd aan de bestaande klasses.
- Het moet mogelijk zijn om met een nieuwe module een operatienetwerk te exporteren naar een implementatie in C of VHDL.

2.4 Besluit

In dit hoofdstuk werd uitgelegd in welke context het ontwikkelde programma gebruikt kan worden. Vervolgens werden de vereisten voor de software besproken. In het volgende hoofdstuk komen de verschillende stappen in het ontwerp van de netwerkstructuur aan bod. De netwerkstructuur is de structuur waarin een operatienetwerk intern wordt bijgehouden. Er zal meermaals verwezen worden naar de in dit hoofdstuk geformuleerde vereisten.

Hoofdstuk 3

Ontwerp van de structuur van een operatienetwerk

3.1 Inleiding

In dit hoofdstuk wordt eerst en vooral uitgelegd welke programmeertaal gebruikt wordt en waarom. Nadien wordt uitvoerig ingegaan op het ontwerp van de achterliggende structuren van een operatienetwerk. De klasses verantwoordelijk voor de grafische weergave worden pas in hoofdstuk 5 besproken.

In sectie 3.2 wordt de keuze van de programmeertaal besproken. In sectie 3.3 wordt de opbouw van het ontwerp beschreven. De belangrijkste klasses komen er aan bod, belangrijke ontwerpbeslissingen worden gemotiveerd. Het eerste ontwerp wordt in sectie 3.4 verduidelijkt aan de hand van het objectdiagram van een operatienetwerk. In sectie 3.5 wordt het eerste ontwerp op enkele punten aangepast en uitgebreid. Dit resulteert in een finaal ontwerp, hetwelke in sectie 3.6 verduidelijkt wordt met een objectdiagram. Daarna worden in sectie 3.7 alle geïmplementeerde operaties kort besproken. In sectie 3.8 wordt een belangrijk implementatieprobleem besproken. In de laatste sectie wordt een conclusie geformuleerd.

3.2 Programmeertaal

Vooraleer begonnen werd met het schrijven van een programma dient een programmeertaal gekozen te worden. Software wordt het best geschreven in een objectgerichte programmeertaal, zeker indien het programma uitbreidbaar dient te zijn. Informatie over de basisconcepten van het objectgericht programmeren, namelijk inkapseling, overerving en polymorfisme, kan men vinden in [8]. Meer uitgebreide informatie over objectgericht programmeren, toegepast op de programmeertaal Java, kan men vinden in het boek

"Objectgericht programmeren met Java" van Steegmans [9].

Om het programma te schrijven werd eerst gedacht aan de programeertaal Java [10]. Deze taal heeft echter een aantal tekortkomingen. Het is algemeen bekend dat Java-programma's niet altijd even deterministisch zijn wat betreft de uitvoeringstijd. Java werkt immers met een Garbage Collector". Dit is een proces dat in de achtergrond van elk Java-programma nu en dan de overbodig geworden objecten uit het geheugen verwijdert. Dit kan op willekeurige momenten voor grafische storingen zorgen. Bovendien kan een Java-programma alleen functioneren indien een Java Virtuele Machine (JVM) is opgestart. Het gebruik van Java heeft dan weer twee grote voordelen. Ten eerste zijn geheugenlekken onmogelijk. Ten tweede kan een Java-programma in elk besturingssysteem gebruikt worden, weliswaar alleen nadat de Java Virtuele Machine is opgestart.

Een alternatief is het gebruik van C++ [11]. Deze taal is ook objectgericht, maar staat veel dichter bij de hardware dan Java. Het geheugenbeheer is meer in handen van de programmeur en kan dus geoptimaliseerd worden. Naast de keuze van de programmeertaal zelf, is ook een toolbox nodig om de grafische gebruikersinterface te kunnen maken. Een dergelijke toolbox bevat voorgedefinieerde klasses om programma's te ontwerpen. Een extra voorwaarde is het beschikken over een grafische module. Het is immers de bedoeling dat de gebruiker op interactieve wijze operatienetwerken kan tekenen. Een redelijk populaire, volledige en wel onderhouden toolkit is Qt [4]. In deze software toolkit werd recentelijk een vernieuwde grafische module geïntroduceerd, genaamd Graphics View [12]. Qt biedt ook verschillende modules om externe bestanden in te laden en informatie erin weg te schrijven. Dit is vooral handig wanneer een operatienetwerk moet worden opgeslagen.

Met de combinatie van C++ en de Qt software toolbox kan het programma dus geschreven worden. Om het programma te schrijven werd Qt versie 4.3 gebruikt. Documentatie over alle modules en klasses kan men vinden in [3].

De software werd geschreven in de Engelse taal. Zowel de grafische gebruikersinterface, als de code is in het Engels. Door deze keuze kunnen meer mensen van het programma gebruik maken of de code uitbreiden.

3.3 Opbouw van het ontwerp

In dit deel wordt het ontwerp van de netwerkstructuur besproken. De netwerkstructuur bestaat uit de klasses waarmee een operatienetwerk intern wordt bijgehouden. De belangrijkste ontwerpbeslissingen worden op opbouwende wijze voorgesteld. Belangrijke beslissingen worden telkens afgewogen tegenover alternatieven. Het ontwerpproces werd opgedeeld in een aantal onderdelen, telkens staat een bepaalde klasse centraal. De naam van deze klasse staat telkens tussen haakjes in de titel van het onderdeel.

3.3.1 Een bitwoord (BitWord)

Het spreekt vanzelf dat een applicatie die berekeningen doet op bitwoorden, een structuur nodig heeft die een bitwoord kan bewaren. Een klasse BitWord werd ontworpen. Deze eenvoudige klasse houdt een vector van positieve gehele getallen (unsigned integers) bij. In de meeste processorarchitecturen bestaat een geheel getal uit 32 bits. Aangezien de lengte van een vector kan variëren, is ook het aantal bits in een bitwoord variabel. Het totaal aantal bits wordt nog eens afzonderlijk bijgehouden.

Een rij van gehele getallen werd verkozen boven een vector van bits. C++ bevat een aantal templates die standaard in de taal aanwezig zijn. Deze templates maken deel uit van de C++ Standard Template Library, of kortweg STL. Deze bibliotheek biedt immers een formaat bitset [13]. Het gebruik van deze template werd overwogen. Deze template biedt tal van voordelen, maar ook een aantal nadelen. De voor- en nadelen zijn hieronder neerschreven.

Voordelen van het STL-formaat bitset:

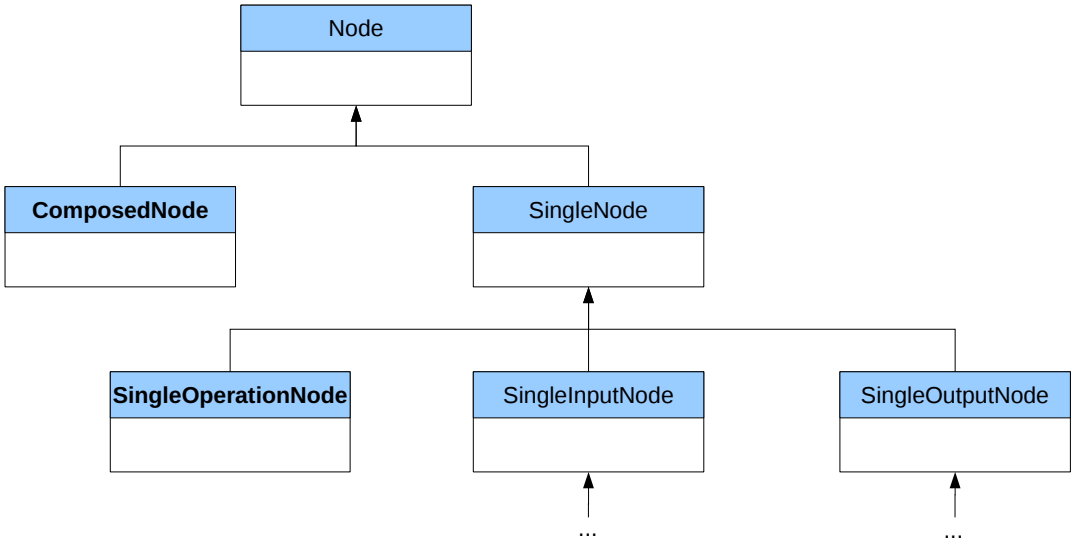
- Het STL-formaat biedt een sterk geoptimaliseerde implementatie van de logische operaties.
- Operatoren om een STL-bitset te verschuiven zijn reeds aanwezig.

Nadelen van het STL-formaat bitset:

- Bepaalde operaties zoals het optellen en vermenigvuldigen van bitwoorden worden inefficiënt indien de STL-template gebruikt wordt. Hoewel de template bitset intern ook positieve gehele getallen gebruikt, voorziet deze template geen methodes om op efficiënte wijze over te gaan naar een voorstelling met positieve gehele getallen.
- Het toewijzen van bepaalde waarden aan een bitset, moet bit per bit gebeuren, dat is niet efficiënt.
- Er zijn geen methodes om een aantal bits uit een bitset te kopiëren naar een nieuwe bitset.
- Het aantal bits in een bitset is niet variabel. Een bitset is immers geen dynamische structuur.

Uiteindelijk werd gekozen voor de eigen klasse met een bitwoord-voorstelling in een vector van positieve gehele getallen. Het spreekt voor zich dat deze keuze berust op een eigen afweging van de voor- en nadelen.

Nadat het programma reeds bijna geschreven was, werd ook het bestaan ontdekt van een template voor dynamische bitwoorden. De template dynamic_bitset is een deel van de bibliotheek Boost [14]. Deze template is een beter alternatief voor de eigen klasse BitWord die gebruikt wordt in het programma.



Figuur 3.1: Voorlopige erfelijkheidsstructuur van de klasse Node.

3.3.2 Een blok (Node)

Centraal in een netwerkstructuur staan natuurlijk de netwerkelementen. In deze thesistekst worden deze blokken genoemd. Een blok is een object van de klasse Node. De klasse Node is een abstracte klasse. De voorlopige erfelijkheidsstructuur van deze klasse wordt voorgesteld in figuur 3.1. De namen van de niet-abstracte klasses staan in het vet.

Er wordt onderscheid gemaakt tussen enkelvoudige blokken (SingleNode) en samengestelde blokken (ComposedNode). Samengestelde blokken zijn blokken waarachter een volledig ander operatienetwerk schuil gaat. Ze zijn een zwarte doos voor een ander operatienetwerk. In tegenstelling tot de klasse ComposedNode, is de klasse SingleNode nog steeds abstract. Er zijn drie types enkelvoudige blokken: ingangs-, uitgangs- en operatieblokken. De klasses SingleInputNode en SingleOutputNode zijn nog steeds abstract, ze worden verder in deze sectie nog onderverdeeld.

Een blok houdt bij welke naam het draagt. Twee blokken mogen dezelfde naam hebben. Om de verschillende blokken te kunnen onderscheiden, heeft elk blok een nummer, namelijk het dynamisch bloknummer. Het bloknummer is dynamisch in die zin dat het dynamisch toegekend wordt; eenmaal een blok een bepaald nummer heeft gekregen, verandert dit nummer niet meer. Het bloknummer is als een sleutel voor het blok en zal later gebruikt worden om een netwerkstructuur op efficiënte wijze op te slaan. Dankzij dit uniek nummer kunnen de verbindingen tussen de verschillende blokken apart van de informatie over de blokken zelf worden opgeslagen. In sectie 6.4.1 wordt hierover meer uitleg gegeven.

Een blok moet op één of andere wijze ook bijhouden met welke andere blokken het is

verbonden. Hoe dit gebeurt, wordt beschreven in het volgende onderdeel.

3.3.3 De I/O van een blok (Connection)

Elk blok houdt bij met welke andere blokken het is verbonden. Enerzijds zijn er de verbindingen die aankomen bij het blok, anderzijds de verbindingen die vertrekken naar een ander blok. Bovendien moet een volgorde worden bijgehouden. Indien zowel blok `A` als blok `B` verbonden zijn met een ingang van blok `C`, dan is het voor blok `C` belangrijk te weten welk blok met de eerste ingang verbonden is en welk blok met de tweede ingang. Dit is niet belangrijk voor commutatieve operaties, maar bijvoorbeeld wel indien blok `C` een samengesteld blok is dat een AES-encryptie voorstelt, waarbij aan de eerste ingang het datablok en aan de andere ingang de sleutel verwacht wordt.

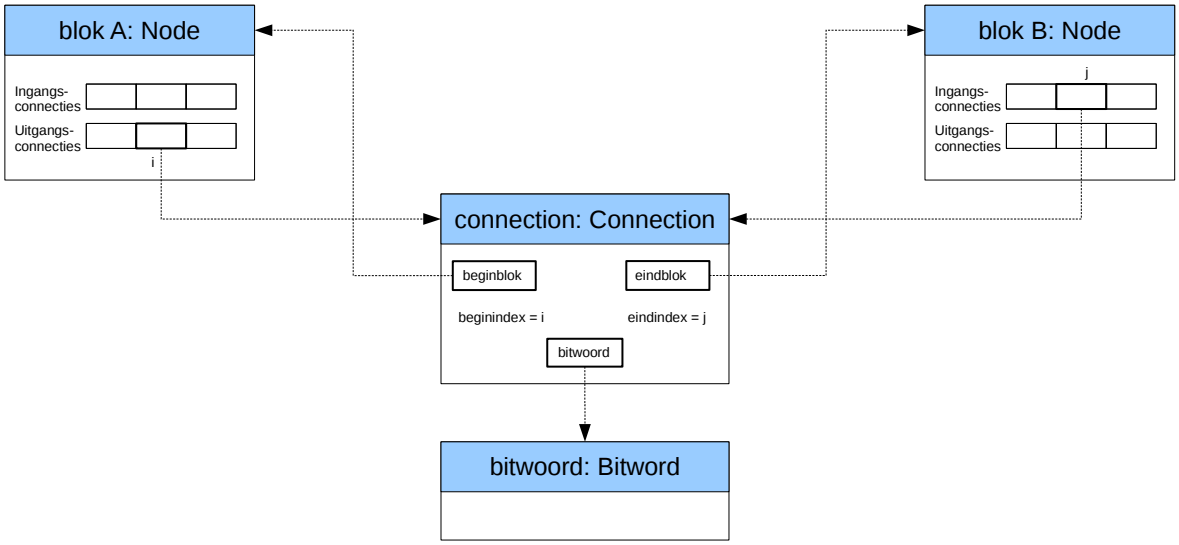
Elk blok houdt dus twee vectoren bij. Eén vector om bij te houden welke blokken verbonden zijn met de ingangen en één vector om bij te houden welke blokken verbonden zijn met de uitgangen. In deze vector zou men referenties naar de verbonden blokken kunnen bijhouden. Dit is echter niet aan te raden, want op deze manier is het voor een blok nogal omslachtig om te achterhalen op welke ingang van een ander blok, zijn uitgang is aangesloten. Er is dus nood aan een nieuwe klasse.

Deze nieuwe klasse heet Connection. In deze klasse wordt een referentie naar een start- en een eindblok bijgehouden, alsook telkens een index die aangeeft op de hoeveelste in- en uitgang de connectie is aangesloten.

Om het programmatorisch beheer van de connecties te vergemakkelijken, zal een nieuw blok met bijvoorbeeld twee uitgangen, de twee uitgangsconnecties automatisch construeren, nog tijdens zijn constructie. Aan de eindblok-referentie van deze connectie wordt de nulreferentie toegekend. Dit automatisch genereren van connecties gebeurt alleen bij de uitgangen, niet bij de ingangen. Stel een nieuw blok met twee ingangen heeft een ingangsvector van lengte twee. Deze vector bevat twee nulreferenties. Indien de `i`-de uitgang van node `A` verbonden wordt met de `j`-de ingang van node `B`, dienen drie zaken gerealiseerd te worden:

1. De eindblok-referentie in de connectie dient vanaf nu te wijzen naar blok `B`.
2. De eindindex van de connectie dient op `j` gezet te worden.
3. Het `j`-de element in de vector van ingangsconnecties van node `B` dient een referentie naar de connectie te bevatten.

Doorheen de netwerkstructuur dienen ergens bitwoorden te worden bijgehouden. Dit gebeurt in de connecties. Naast de referenties naar de blokken en de indexen, houdt een connectie dus ook een referentie naar een bitwoord bij. Indien een berekening doorheen de netwerkstructuur propageert, zullen alle tussenresultaten bijgehouden worden in de bitwoorden van elke connectie. Hoe een berekening precies gestart wordt en hoe ze propageert doorheen een operatienetwerk, wordt in detail beschreven in sectie 4.3.



Figuur 3.2: Verband tussen de variabelen in een connectie en de variabelen in de blokken die de connectie verbindt.

De verschillende variabelen van een connectie en het verband met de variabelen in de blokken die de connectie verbindt, zijn zichtbaar in figuur 3.2. Merk op dat de indexen i en j starten bij nul.

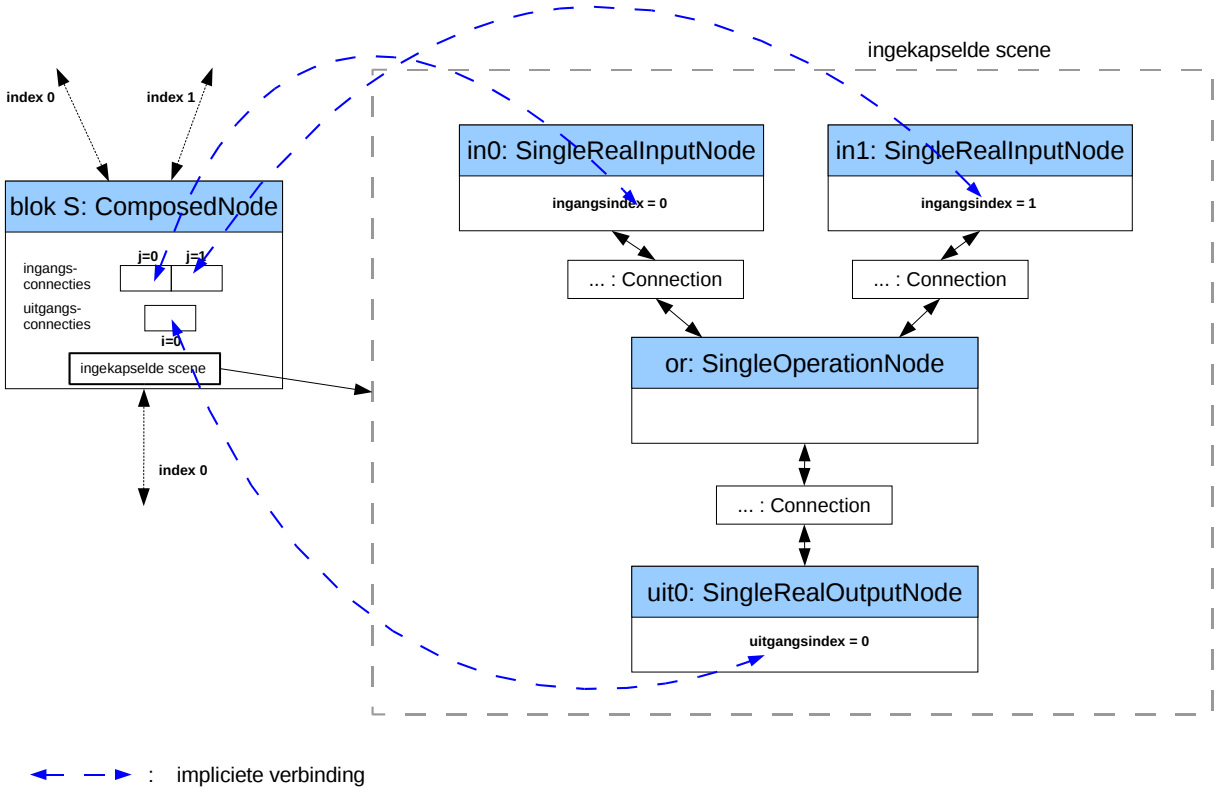
3.3.4 Een scène (ScenePointerCollection)

Er is een klasse nodig om een operatienetwerk bij te houden. Deze klasse heet ScenePointerCollection. Voor de eenvoud wordt een object van deze klasse in het vervolg van deze tekst een 'scène' genoemd. Deze klasse houdt referenties bij naar alle blokken in het operatienetwerk. Dit gebeurt in een vector. Deze klasse lijkt voorlopig van weinig belang, maar ze is cruciaal om te voldoen aan de encapsulatievereiste. Het is ook deze klasse die meegegeven wordt aan de routine die een operatienetwerk opslaat. Daarover wordt meer uitleg gegeven in sectie 6.4.

Elke scène houdt bij of ze standaard is of niet. Een standaard scène werd ingeladen uit een standaard in het programma aanwezig document.

3.3.5 Een blok (Node): vervollediging erfelijkheidsstructuur

In bovenstaand onderdeel werd de klasse ScenePointerCollection ingevoerd. Aan elk blok wordt een referentie naar deze scène toegevoegd. Dit is zeker noodzakelijk, want indien

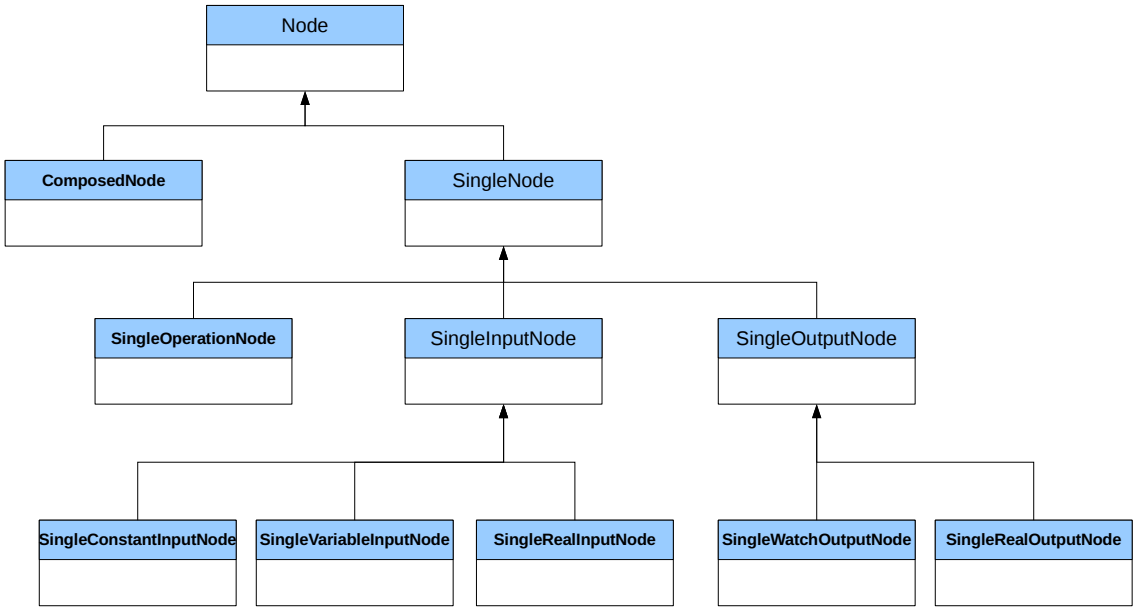


Figuur 3.3: Verband tussen een samengesteld blok en de I/O-ingangen en I/O-uitgangen in de ingekapselde scène.

een gebruiker een blok wil verwijderen, moet de scène dit blok verwijderen uit haar lijst van blokken. Het blok dat verwijderd wordt, meldt dus vanuit de destructor aan de scène dat het niet langer aanwezig is in de scène.

In de erfelijkheidsstructuur van figuur 3.1 zijn de abstracte klasses SingleInputNode en SingleOutputNode nog niet verder onderverdeeld. Dit zal nu gebeuren.

Er wordt onderscheid gemaakt tussen de normale in- en uitgangen en de I/O-in- en uitgangen van een scène. De bitwoorden horende bij de I/O-ingangen van de scène komen van buiten de scène, het zijn dus externe ingangen. De bitwoorden horende bij de I/O-uitgangen van de scène zijn externe uitgangen. Een scène met minstens één I/O-ingang en minstens één I/O-uitgang kan gebruikt worden als ingekapselde scène van een samengesteld blok. De in- en uitgangen van dit samengesteld blok zijn dan impliciet verbonden met de I/O-ingangen en I/O-uitgangen van de ingekapselde scène. Elke I/O-ingang heeft een nummer. Dit nummer bepaalt met welke ingang van het samengestelde blok de I/O-ingang overeenkomt. Ook de I/O-uitgangen hebben een gelijkaardig nummer. Dit alles wordt onder de vorm van een voorbeeld voorgesteld in figuur 3.3.



Figuur 3.4: Volledige erfelijkheidsstructuur van een blok.

Naast de I/O van de scène, zijn er nog de normale in- en uitgangsblokken. Er zijn twee soorten normale ingangen. Er zijn de constante ingangen (SingleConstantInputNode) en de variabele ingangen (SingleVariableInputNode). Beide blokken verschillen maar weinig, de bitwoorden horende bij de constante ingangen worden altijd mee opgeslagen, terwijl bij de variabele ingangen alleen het aantal bits wordt opgeslagen. Het bitwoord horende bij het ingangsblok wordt telkens opgeslagen in het bitwoord van de uitgangsconnectie. Alle ingangsblokken hebben precies één uitgangsconnectie en nul ingangsconnecties.

Er is slechts één normaal uitgangsblok (SingleWatchOutputNode); het bitwoord dat dit blok na een berekening bereikt, verschijnt onmiddellijk op het blok, op voorwaarde dat het aantal bits in het bitwoord niet groter is dan acht. Dit blok wordt een observatie-uitgang genoemd. Uitgangsblokken hebben precies één ingangsconnectie en één of geen uitgangsconnectie.

De volledige erfelijkheidsstructuur van een blok wordt in figuur 3.4 weergegeven. De namen van de niet-abstracte klasses staan in het vet.

3.3.6 Een samengesteld blok (ComposedNode)

Het samengestelde blok werd geïmplementeerd in de klasse ComposedNode. Dit blok kapselt een andere scène in. Een samengesteld blok bevat bijgevolg niet alleen een referentie naar de scène waarin het zich bevindt, maar ook een referentie naar de scène die het inkapselt. Het aantal I/O-ingangen en I/O-uitgangen van deze ingekapselde scène, komt exact overeen met het aantal in- en uitgangen van het bijhorende samengestelde

blok.

Merk op dat de ingekapselde scène van een samengesteld blok opnieuw samengestelde blokken kan bevatten. Er ontstaat dus een hiërarchie van scènes.

In een samengesteld blok dient extra aandacht besteed te worden aan het correct laten propageren van berekeningen. In onderdeel 4.3.2 wordt de gevolgde werkwijze uitgelegd.

3.3.7 Uitbreidingen van de scène (ScenePointerCollection)

Uit het vorige onderdeel blijkt dat een samengesteld blok een referentie naar de ingekapselde scène bijhoudt. Om deze binding bidirectioneel te maken, wordt in een scène ook een referentie naar een samengesteld blok toegevoegd. Indien de scène niet is ingekapseld en dus op zichzelf staat, wordt deze referentie gelijkgesteld aan de nulreferentie.

Tot dusver bevat de scène precies één vector, met referenties naar alle blokken. Er worden nu enkele vectoren toegevoegd. Deze vectoren zijn alleen nuttig indien men de gemaakte structuur gebruikt om berekeningen te maken.

Een vector met alleen de I/O-ingangen van de scène en een vector met alle normale ingangen van de scène worden toegevoegd. Deze vectoren vergemakkelijken de start van een berekening. Een berekening wordt gestart door aan alle ingangen van de scène een berekeningsopdracht te geven, alle ingangen van een scène zijn dankzij deze nieuwe vectoren onmiddellijk opvraagbaar. Merk op dat de normale ingangen in een andere vector geplaatst worden dan de I/O-ingangen. Op die manier zijn de I/O-ingangen van de scène onmiddellijk opvraagbaar. Er is ook een extra vector voor de I/O-uitgangen. Op deze manier zijn de I/O-uitgangen van de scène eveneens onmiddellijk opvraagbaar.

Hoe het berekenen precies in zijn werk gaat en waarom het starten van berekeningen op deze manier gebeurt, wordt besproken in sectie 4.2.

3.3.8 Een operatieblok (SingleOperationNode)

De klasse SingleOperationNode bevindt zich onder de klasse SingleNode in de erfelijkheidsstructuur van een blok in figuur 3.4. Deze klasse wordt gebruikt voor operatieblokken. In de klasse SingleOperationNode wordt een nieuwe variabele ingevoerd. Het betreft een referentie naar een operatie (Operation). Elk operatieblok heeft een eigen operatie.

Er werd besloten het soort operatie niet op te nemen in de erfelijkheidsstructuur van de klasse Node. Dit alternatief werd verworpen omdat, indien de operatie het type van het blok zou bepalen, een verandering van operatie zou resulteren in het verwijderen van het oorspronkelijke operatieblok en het creëren van een ander operatieblok.

De beslissing om een nieuwe klasse Operatie in te voeren is dus logisch. Indien de informatie over de operatie geherbergd wordt in een afzonderlijke klasse, volstaat het de

oude operatie te verwijderen en een nieuwe operatie te creëren.

3.3.9 Een operatie (Operation)

Alle operaties krijgen een eigen klasse. Elke operatieklasse erft van de abstracte klasse Operation. Aan een operatie kan gevraagd worden hoeveel ingangen ze nodig heeft. Het aantal ingangen is niet noodzakelijk constant. Er is bijvoorbeeld ruimte voor een OR-operatieblok dat meer dan twee ingangen aanvaardt.

Een operatie bevat een referentie naar een operatieblok. De operatie en het operatieblok zijn dus bidirectioneel gebonden.

Elke operatie heeft een methode *calculate*. Indien deze methode wordt opgeroepen, worden de bitwoorden in de ingangsconnecties van operatieblok opgevraagd en worden de uitkomsten geplaatst in de bitwoorden van de uitgangsconnecties van het operatieblok. De meeste operaties hebben slechts één uitgang. Er zijn echter operaties met meerdere uitgangen, bijvoorbeeld de kopie-operatie.

Meer details over de geïmplementeerde operaties zijn te vinden in sectie 3.7.

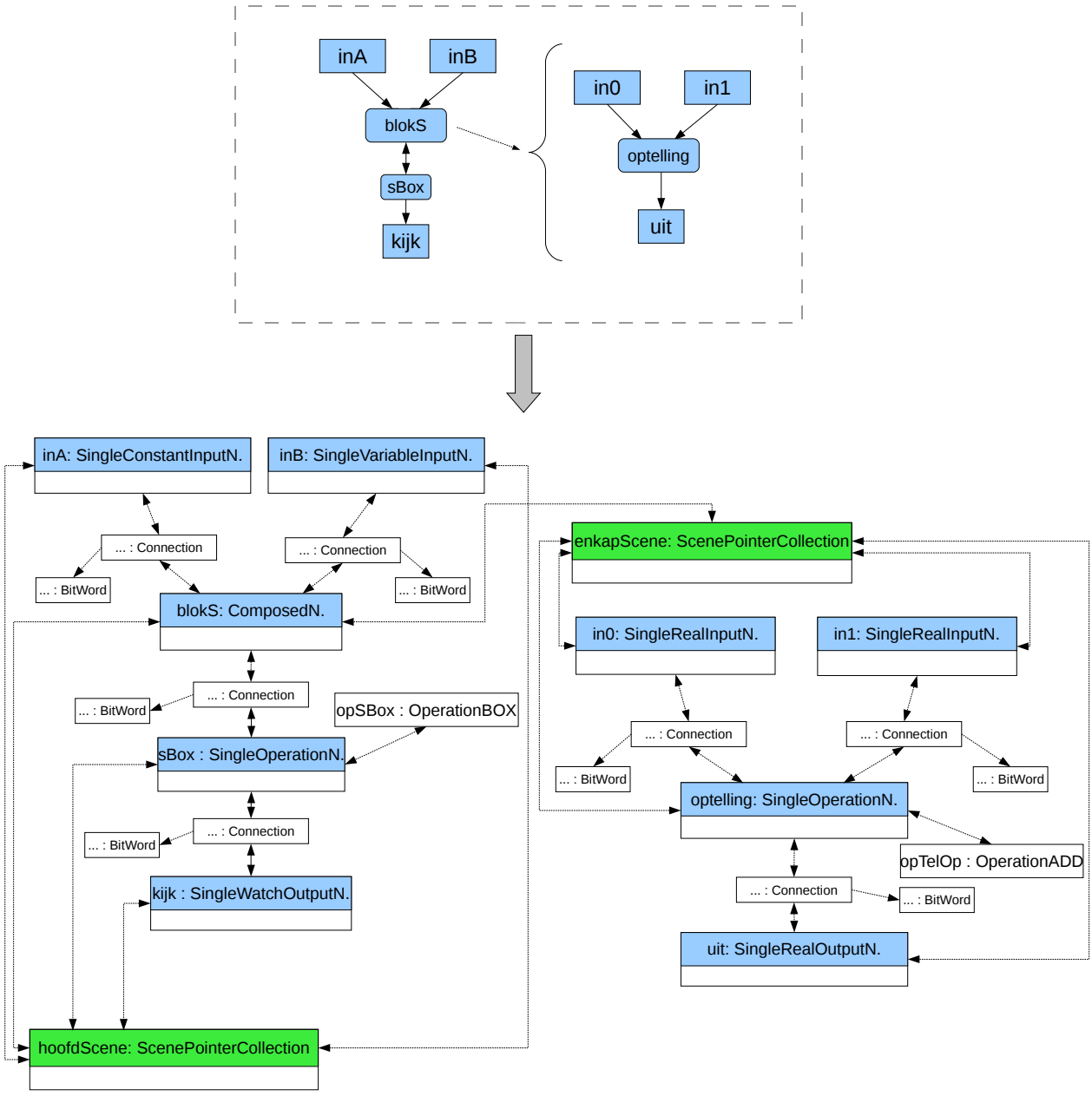
3.4 Overzicht van het ontwerp

In sectie 3.3 werden de belangrijkste variabelen en referenties in elke klasse besproken. In dit deel worden al deze ontwerpbeslissingen op samenvattende wijze voorgesteld in één grote afbeelding, namelijk in figuur 3.5. De afbeelding bevat het objectdiagram van een eenvoudig operatienetwerk. Alle soorten blokken zijn vertegenwoordigd. Het operatienetwerk bevindt zich bovenaan.

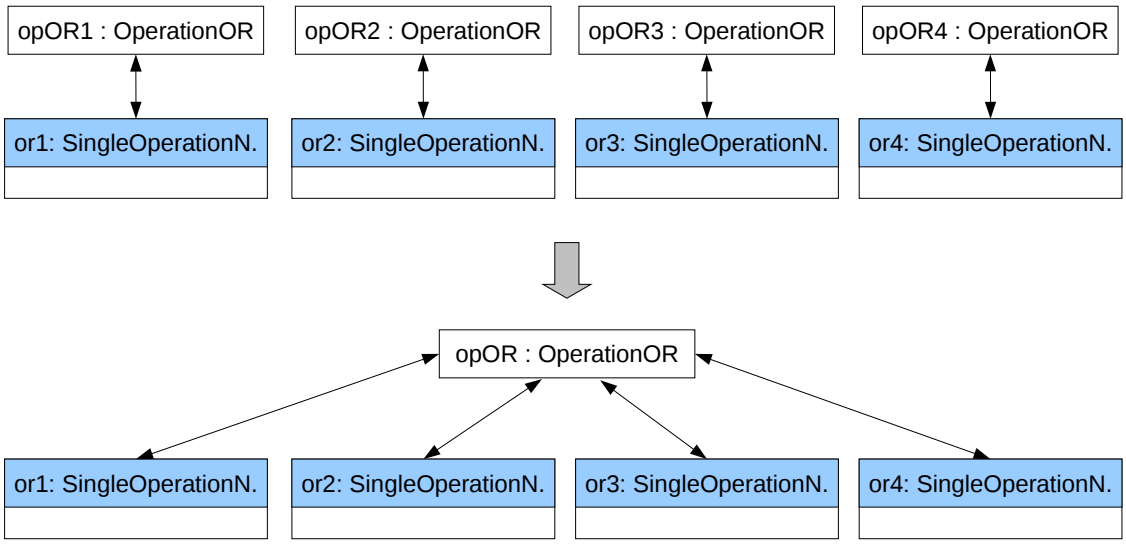
In deze thesistekst wordt vrijwel nooit ingegaan op de programmeercode, maar een kort codefragment waarin het objectdiagram uit figuur 3.5 wordt opgebouwd, kan zeker helpen bij het beter begrijpen van het ontwerp. Dit codefragment bevindt zich in bijlage A.

3.5 Aanpassingen van het ontwerp

Op een zeker moment tijdens het programmeren werden enkele belangrijke veranderingen aangebracht aan het ontwerp van de netwerkstructuur. Een eerste grote aanpassing houdt verband met de binding tussen operatieblok en operatie. De tweede aanpassing heeft betrekking op de binding van een samengesteld blok en de bijhorende ingekapselde scène. In dit deel worden beide veranderingen afzonderlijk aangebracht. De lezer zal merken dat beide aanpassingen analoge doelstellingen beogen. In deel 3.6 wordt het finale ontwerp van de netwerkstructuur overzichtelijk samengevat.



Figuur 3.5: Objectdiagram van een eenvoudig operatienetwerk. Het operatienetwerk bevindt zich bovenaan.



Figuur 3.6: Relatie tussen operatie en operatieblok: bovenaan zonder gelinkte operaties, onderaan met gelinkte operaties.

3.5.1 Gelinkte operaties

In het eerste ontwerp is er een bidirectionele binding tussen elk operatieblok en haar operatie. In cryptografische cijfers is het soms zo dat een operatienetwerk parallelle takken bevat met gelijklopende operaties. In de SubBytes-stap van het AES-algoritme [2] bijvoorbeeld, wordt op elk van de zestien bytes van een blok dezelfde substitutie-box toegepast. In een dergelijk geval zou het interessant zijn dat de betrokken operatieblokken met éénzelfde operatie-object gebonden zijn. Dit zou resulteren in minder geheugengebruik. Tevens zou het operatienetwerk op een compactere manier kunnen worden weggeschreven naar een extern bestand. Dit is niet het belangrijkste voordeel. Indien verschillende operatieblokken verbonden zijn met eenzelfde operatie, kan men de eigenschappen van deze operatie of de volledige operatie voor alle operatieblokken tegelijkertijd veranderen. Op deze manier wordt het programma gebruiksvriendelijker. Operatieblokken die een operatie delen, worden ‘gelinkte’ operatieblokken genoemd.

Om het linken van operatieblokken mogelijk te maken, wordt de referentie in een operatie naar een operatieblok vervangen door een vector van referenties naar operatieblokken. In figuur 3.6 wordt de relatie tussen een operatieblok en een operatie getoond, voor vier operatieblokken met dezelfde OR-operatie. Bovenaan in de figuur is de werkwijze zonder gelinkte operaties te zien, onderaan is de werkwijze met gelinkte operaties te zien.

3.5.2 Gelinkte samengestelde blokkken

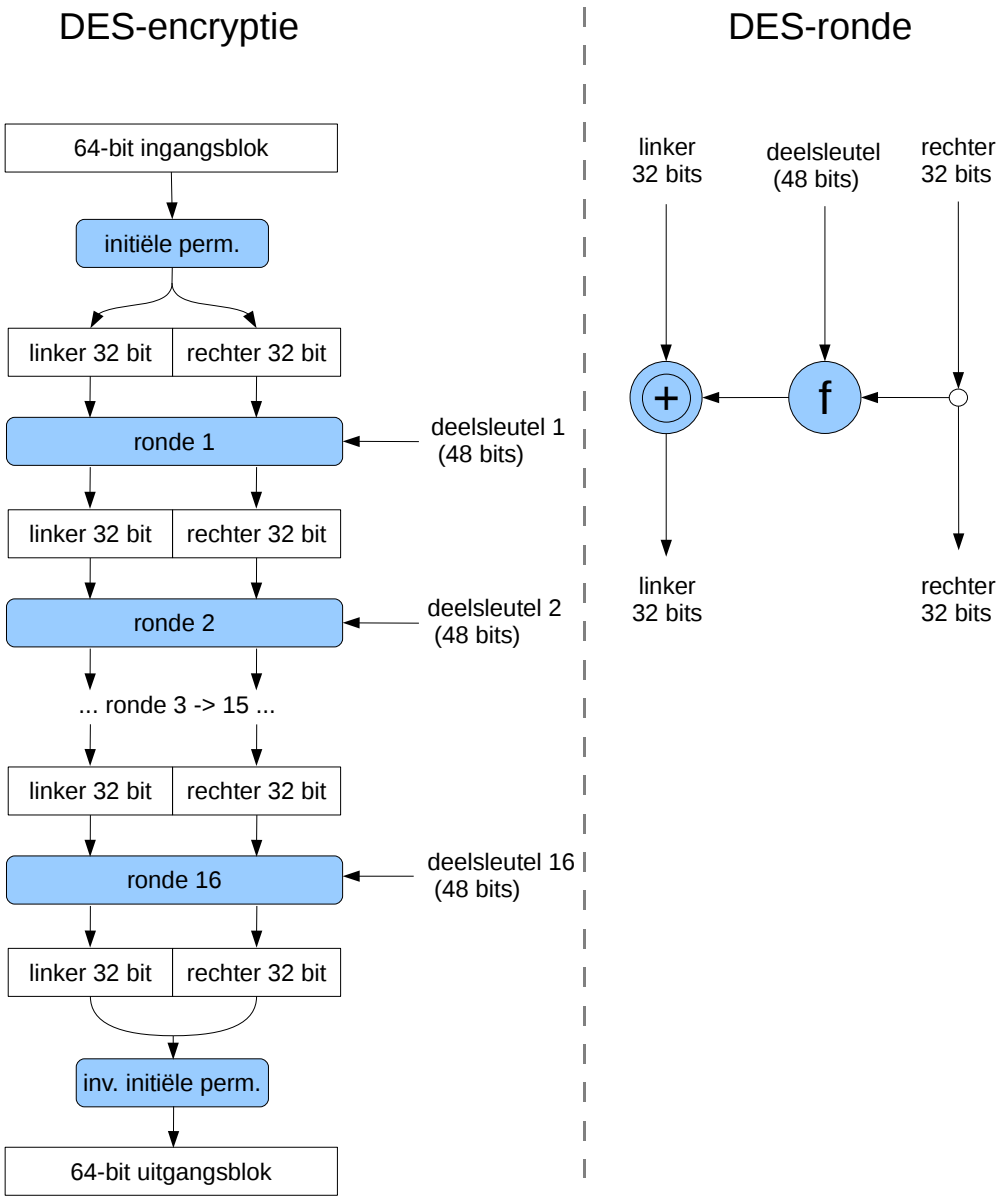
Net zoals bij operatieblokken, is het vaak zo dat bepaalde delen van een operatienetwerk meermaals voorkomen. Uiteraard kan dit wederkerend deel ingekapseld worden, gebruikmakende van enkele samenstelde blokken. Het eerste ontwerp biedt echter niet de mogelijkheid om expliciet uit te drukken dat deze samengestelde blokken dezelfde functionaliteit hebben. Er zijn tal van situaties waarin dit nuttig kan zijn. Het DES-algoritme (Data Encryption Standard) bijvoorbeeld bestaat uit verschillende rondes met telkens dezelfde functionaliteit, dit wordt weergegeven in figuur 3.7. De functie Feistel functie f is eigen aan DES en wordt nader gespecifieerd in [15].

In dit ontwerp wordt het dus ook mogelijk om uit te drukken dat de verschillende rondes in het DES-algoritme uit figuur 3.7 dezelfde zijn. Elke ronde wordt één samengesteld blok. Al deze samengestelde blokken kunnen worden gelinkt. Het linken van samengestelde blokken komt neer op het linken van ingekapselde scènes. Het is belangrijk in te zien dat het linken van deze scènes niet zo eenvoudig is als het linken van operaties. Twee gelinkte samengestelde blokken mogen in geen geval verwijzen naar eenzelfde scène. Waarom dit zo is, wordt nu uitgelegd.

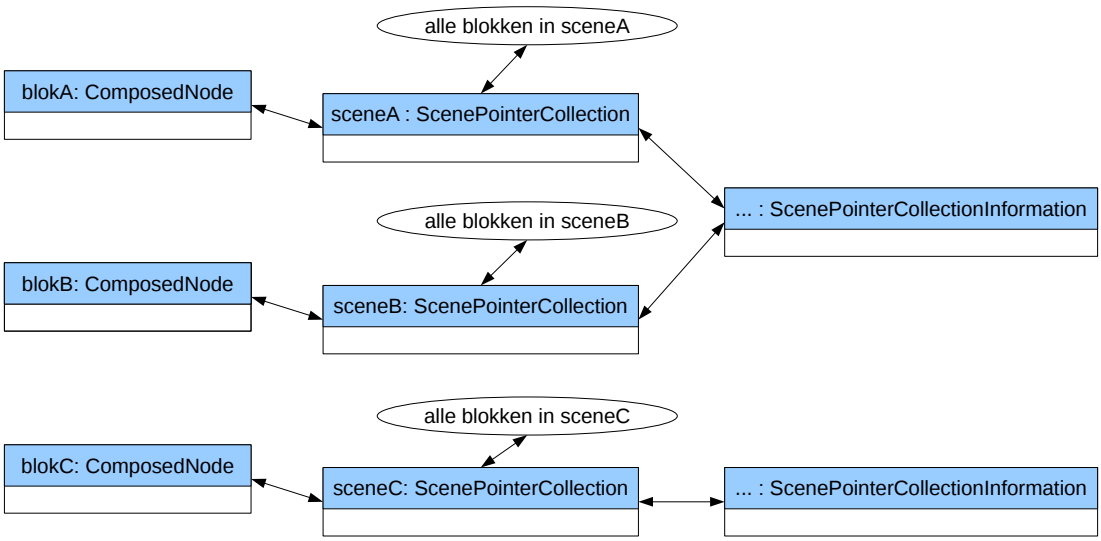
Stel dat twee samengestelde blokken verwijzen naar eenzelfde ingekapselde scène. Indien een berekening gestart wordt in de scène waarin deze samengestelde blokken zich bevinden, zullen de berekeningen twee maal propageren doorheen diezelfde ingekapselde scène. De bitwoorden aanwezig in de connecties van de ingekapselde scène zullen diegene zijn van de laatste propagatie. De tussenresulaten uit de eerste propagatie doorheen de ingekapselde scène gaan verloren. Het is dus niet aan te raden twee gelinkte samengestelde blokken te laten wijzen naar eenzelfde scène, gezien de vereiste dat alle tussenresultaten beschikbaar moeten zijn.

Elk samengesteld blok wijst dus naar een andere ingekapselde scène. Het is natuurlijk de bedoeling dat gelinkte samengestelde blokken over gelijke ingekapselde scènes beschikken. Indien de samengestelde blokken `blokA` en `blokB` gelinkt zijn, dan moet de inhoud van de ingekapselde scène van `blokB` mee wijzigen, indien de ingekapselde scène van `blokA` gewijzigd wordt.

Op een zekere manier moet worden uitgedrukt dat twee of meerdere samengestelde blokken gelinkt zijn. Dit gebeurt met behulp van een nieuwe klasse, genaamd ScenePointerCollectionInformation. Verder in deze tekst wordt naar deze klasse verwezen met de term 'scène-informatie'. Elke scène (ScenePointerCollection) krijgt een referentie naar een object van deze klasse. Twee gelinkte scènes hebben een referentie naar eenzelfde scène-informatie. Algemene informatie over de scène wordt verhuisd van de scène naar de scène-informatie. In de volgende paragraaf wordt uitgelegd over welke informatie het gaat. De vectoren met referenties naar de blokken in de scène blijven in het scène-object. De relatie tussen een scène en diens scène-informatie wordt verduidelijkt in figuur 3.8. In de figuur zijn de samengestelde blokken `blokA` en `blokB` gelinkt met elkaar. De scènes `sceneA` en `sceneB` zijn dus identiek qua inhoud. Het samengestelde blok `blokC` is niet



Figuur 3.7: De algemene structuur van het DES-algoritme [16].



Figuur 3.8: Relatie tussen een scène en diens scène-informatie.

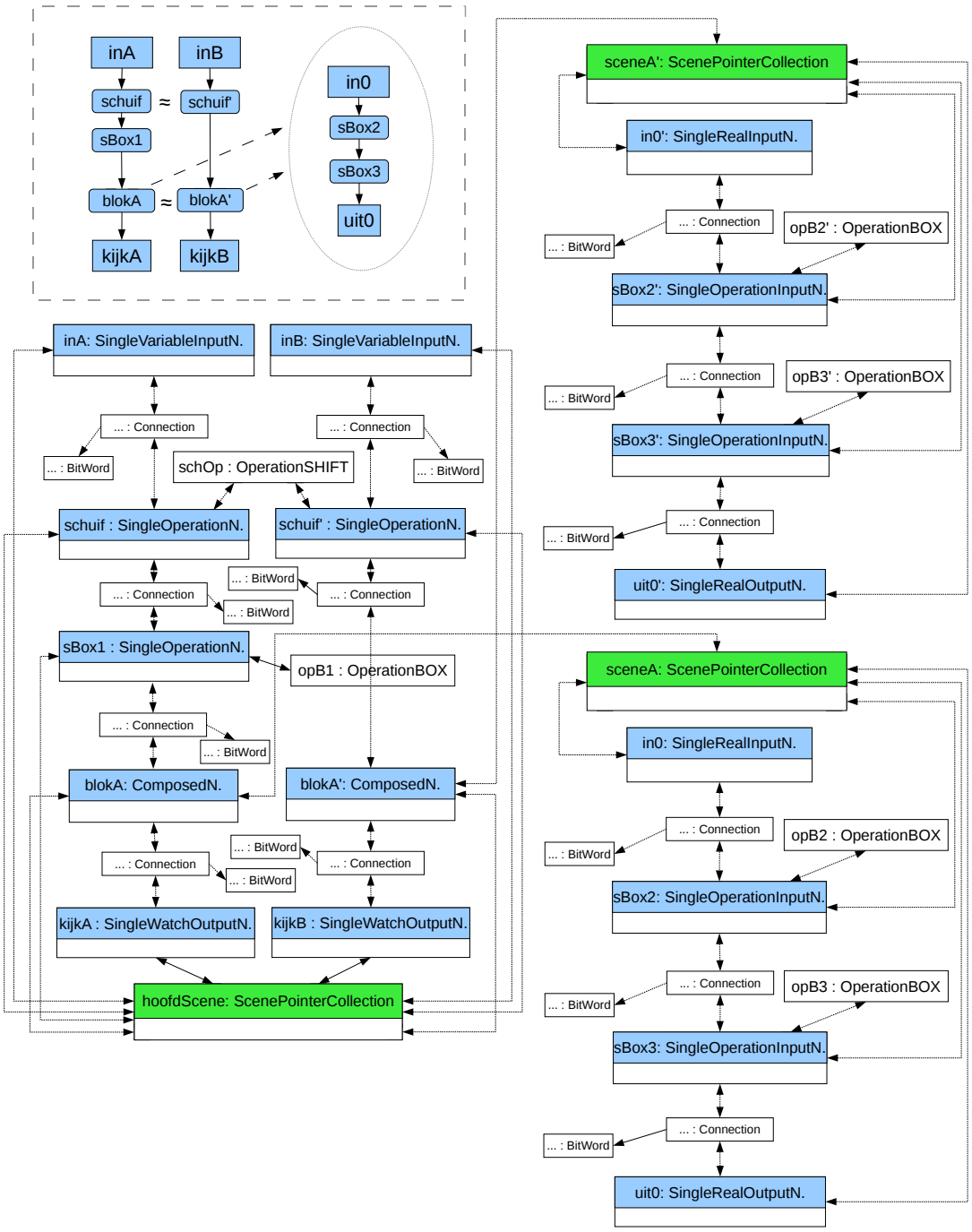
gelinkt.

Een scène-informatie heeft een vector met referenties naar de gelinkte scènes. Daarnaast houdt een scène-informatie bij of de gelinkte scènes standaard scènes zijn en of de gelinkte scènes vergrendeld zijn of niet. Standaard scènes zijn vergrendeld en kunnen dus niet gewijzigd worden. Ze kunnen ontgrendeld worden, maar dan zijn ze niet meer standaard.

Het linken van samengestelde blokken is dus complexer dan het linken van operaties. Vooral het verzorgen van de voorwaarde dat alle gelinkte scènes dezelfde inhoud moeten hebben, is soms nogal rekenintensief.

3.6 Overzicht van het finale ontwerp

Om de finale structuur van een operatienetwerk te verduidelijken, wordt hier opnieuw een objectdiagram voorgesteld. Het objectdiagram bevindt zich in figuur 3.9. Het overeenkomstige operatienetwerk is bovenaan in de figuur getekend. Op de twee variabele ingangen wordt eerst een schuif-operatie toegegepast. De twee schuif-operatieblokken zijn gelinkt, ze zijn dus gelijk. Later in het operatienetwerk worden twee gelinkte samengestelde blokken gebruikt, namelijk `blokA` en `blokA'`. De inhoud van de ingekapselde scènes van beide blokken is gelijk.



Figuur 3.9: Objectdiagram van een operatienetwerk met gelinkte operaties en gelinkte samengestelde blokken. Het operatienetwerk bevindt zich links bovenaan.

3.7 Ondersteunde operaties

In deze sectie worden alle geïmplementeerde operaties voorgesteld. Ze kunnen onderverdeeld worden in vier groepen: de logische operaties, de Galois veld operaties, de nepoperaties en de overige operaties. Elke operatie erft over van de abstracte klasse Operatie. Elke groep operaties wordt in de volgende onderdelen besproken.

3.7.1 De logische operaties

De meest triviale operaties zijn de logische operaties. In volgende lijst worden alle geïmplementeerde logische operaties vermeld:

- OR
- AND
- NOT
- XOR
- NOR
- NAND

De klasses die geassocieerd zijn met deze operaties bevatten geen extra parameters. Merk op dat de meeste van deze klasses operatieblokken met drie of meer ingangen ondersteunen. Het kan bijvoorbeeld gaan om een OR-operatie op drie ingangen tegelijkertijd. Alleen de NOT-operatie kan slechts op één ingang worden toegepast.

3.7.2 Operaties in een Galois veld

3.7.2.1 Voorbeelden van Galois operaties in de cryptografie

In de symmetrische cryptografie worden soms operaties in een Galois veld gebruikt. Deze operaties bevinden zich op een hoger niveau dan de tot nu toe besproken operaties. Ze kunnen met andere woorden gemaakt worden met behulp van andere operaties. Meer informatie over Galois velden en operaties in deze velden kan men vinden in [17].

Galois velden van karakteristiek twee worden het vaakst gebruikt. De elementen in dergelijke velden kunnen worden voorgesteld als veeltermen met binaire coëfficiënten. Hoe deze omzetting gebeurt, wordt uitgelegd in bijlage B.1. De optelling en de aftrekking in binaire Galois velden zijn eenvoudige XOR-operaties. Het berekenen van het product van twee elementen en de inverse van een element is meestal moeilijker.

In AES [18] wordt gewerkt in $GF(256)$, de genererende veelterm is er $x^8 + x^4 + x^3 + x + 1$. Dit veld wordt het Rijndael-veld genoemd.

Deze Galois velden zijn vooral belangrijk bij het ontwerp van cijfers. De eerste stap bij het encrypteren van data met AES, genaamd SubBytes, is een permutatie. Deze S-box van deze permutatie wordt in tabel 3.1 voorgesteld. Dit gebeurt niet in een opzoekingstabel, maar in een tweedimensionale permutatietabel. Uitleg over hoe men een tabelvoorstelling van een substitutie-box moet interpreteren, is te vinden in bijlage C. In deze initiële stap wordt elke byte vervangen door een andere byte. De gebruikte permutatietabel kent echter een wiskundige achtergrond.

Elke byte wordt eerst geïnverteerd in het Rijndael-veld (`0x00` wordt afgebeeld of zichzelf). Vervolgens ondergaat deze inverse een matrix vermenigvuldiging, opnieuw in het Rijndael-veld. In een laatste stap wordt een XOR-bewerking met `0x63` doorgevoerd. Deze berekeningen worden hieronder gedemonstreerd voor de byte `0xa2`, het overeenkomstig resultaat werd in het vet aangeduid in tabel 3.1. De eerste bit in een bit-vector is de minst significante bit, de achtste bit is de meest significante bit.

$$\texttt{0xa2} \rightarrow (x^7 + x^5 + x)$$

$$(x^7 + x^5 + x)^{-1}\ mod\ x^8 + x^4 + x^3 + x + 1 = x^5 + x^3 + x^2 + x$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \rightarrow \texttt{0x3a}$$

Verderop in het AES-algoritme worden meermaals vermenigvuldigingen in het Rijndael-veld gebruikt. Voor een bespreking van de volledige structuur van AES kan de lezer terecht in onderdeel 8.2.1. Ook andere cryptografische algoritmes, bijvoorbeeld de Whirlpool hash-functie [19], gebruiken operaties in een Galois veld.

Merk op dat operaties in een Galois veld in de praktijk geïmplementeerd worden als een samenstelling van logische operaties. Vanuit educatief oogpunt is het echter interessant om sommige Galois operaties op een hoger niveau voor te stellen.

3.7.2.2 Implementatie van de Galois operaties

De mogelijke Galois velden worden beperkt tot de binaire Galois velden ($GF(2^q)$, $q>0$). Op deze manier is het verband tussen een bitwoord en een element in het veld éénduidig. Het is niet nuttig om voor alle operaties in een Galois veld nieuwe operatieblokken te

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5d	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	4e	0b	db
a	e0	32	**3a**	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Tabel 3.1: AES SubBytes stap: permutatietabel (hexadecimaal)

voorzien. Er worden twee nieuwe operaties geïmplementeerd. Met de eerste kan men twee Galois elementen vermenigvuldigen. Met de tweede kan men de inverse van een Galois element berekenen.

Voor de implementatie van deze operaties wordt telkens gebruik gemaakt van een logaritme- en een exponenttabel. Op deze manier kunnen berekeningen in het veld heel snel gebeuren. Voor elk Galois veld zijn dus twee tabellen nodig; de grootte ervan is evenredig met het aantal elementen in het veld. Deze methode kan dus niet gebruikt worden in combinatie met grote velden, anders wordt het geheugengebruik te groot. De velden die beschikbaar zijn in het programma werden vooraf bepaald. Op deze manier wordt de gebruiker niet geconfronteerd met de nood aan een generator van het veld. Een generator is nodig om de vermelde tabellen te genereren. In bijlage B kan de lezer meer uitleg vinden over de wiskundige achtergrond, het genereren en het gebruik van deze tabellen.

Vooraleer werd overgegaan tot een eigen implementatie van deze operaties, werden een aantal pakketten voor Galois veld operaties geraadpleegd. De beschikbare pakketten bleken echter niet geschikt voor gebruik in de applicatie. Een eerste pakket [20], gemaakt door Partow functioneert alleen met velden waarin het element twee een generator is. In een ander pakket, geschreven door Plank [21], werd het Galois veld statisch opgeslagen. Het statisch opslaan van een veld maakt het onmogelijk om twee operatieblokken die werken in twee verschillende velden samen te gebruiken. Bovendien is het gebruik van statistisch opgeslagen velden programmatorisch onverantwoord.

Omdat de beschikbare pakketten niet voldeden, werd bijgevolg overgegaan tot een eigen

implementatie. De informatie over het binaire Galois veld bevindt zich in een object van de klasse GaloisField. Daarnaast is er nog een klasse GaloisElement om een element van een binair veld bij te houden. Elk object van de klasse GaloisElement bevat een verwijzing naar een object van de klasse GaloisField. De methodes om de logaritme- en exponenttabel van een veld te genereren zijn gebaseerd op de implementatie in het bestand `gf2.cpp` uit het pakket van Plank [21].

3.7.3 De nepoperaties

Er zijn zes zogenaamde nepoperaties. Deze operaties komen niet overeen met een wiskundige operatie.

De eerste is een kopie-operatie. Met deze operatie kan men een verbindingslijn opsplitsen in meerdere verbindingslijnen met telkens hetzelfde bitwoord. Vanuit de uitgang van een blok kan slechts één verbindingslijn vertrekken. Daarom is de kopie-operatie een onmisbare operatie.

De tweede nepoperatie is de verbind-operatie. Dit is een kopie-operatie met slechts één ingang en één uitgang. Deze operatie kan worden gebruikt om verbindingslijnen om te leiden. Deze operatie kan niet op de gewone manier worden toegevoegd. In onderdeel 7.4.1 wordt uitgelegd hoe men deze operatie kan gebruiken.

De derde nepoperatie laat de bitwoorden horende bij de verschillende ingangen te concateneren. Deze operatie wordt de smelt-samen-operatie genoemd. Met deze operatie kan men bijvoorbeeld een operatieblok maken dat een bitlijn van drie bits en een bitlijn van zeven bits, omzet in een bitlijn van tien bits. Het operatieblok uit dit voorbeeld heeft twee ingangen en één uitgang. Deze operatie werkt ook met meer dan twee ingangen. De klasse die overeenkomt met deze operatie heeft geen extra parameters. Het aantal ingangen is onbepaald en hangt af van het operatieblok waarmee de operatie geassocieerd is.

Met de vierde nepoperatie kan een verbindingslijn opgesplitst worden in verschillende verbindingslijnen met telkens een deel van de bits van de ingangsverbinding. Aan de ingang kan bijvoorbeeld een verbinding van zestien bits bevestigd zijn. De gebruiker kan er met deze operatie voor kiezen dat het geassosocieerde operatieblok drie uitgangen heeft, waarop respectievelijk de eerste vier, de volgende zes en de laatste zes bits van de ingangsverbinding geplaatst worden. Deze operatie wordt een splits-op-operatie genoemd. Deze operatie heeft wel parameters. De operatie houdt een vector bij waarin de bitverdeling opgeslagen wordt. De vector horende bij de operatie in voorgaand voorbeeld bevat de elementen vier, zes en zes. Eenmaal deze splits-op-operatie is ingesteld, werkt ze alleen correct indien aan de ingang een bitwoord met het correcte aantal bits aangelegd wordt. Indien dit tijdens een berekening niet het geval is, wordt de gebruiker onder andere op grafische wijze geïnformeerd over deze fout. Meer informatie hierover kan men vinden in onderdeel 5.3.2.

De vijfde nepoperatie maakt het mogelijk een bitwoord uit te breiden. Deze operatie heeft twee ingangen nodig. Het eerste bitwoord is het bitwoord dat wordt uitgebreid. Het tweede bitwoord wordt gebruikt om het eerste bitwoord uit te breiden tot een gespecifieerd aantal bits. De eerste ingang wordt zoveel keer geconcateneerd met de tweede ingang als nodig, om het gespecifieerd totaal aantal bits te bereiken.

De laatste nepoperatie geeft aan de uitgang het aantal bits aan de ingang. Het aantal bits waarmee dit aantal wordt voorgesteld, kan worden ingesteld.

Deze zes operaties zijn wiskundig gezien geen operaties. Toch werden ze beschouwd als operaties omdat hun werking analoog is aan die van een gewone operatie. Er is wel een subtiel verschil. Het aantal uitgangen van de kopie- en splits-op-operatie wordt niet gedetermineerd door het aantal ingangen. Dit is bij alle andere operatie wel het geval. Dit geeft echter geen programmatorische problemen.

3.7.4 De overige operaties

Er zijn operaties om bitwoorden op te tellen of te vermenigvuldigen. Tevens zijn er operaties om bitwoorden te verschuiven en te roteren. Tot slot zijn er nog de substitutie-box operatie (S-box-operatie) en de bit-herverdeler.

De operaties om bitwoorden op te tellen en te vermenigvuldigen bevatten geen speciale parameters.

De schuif-operatie laat toe bitwoorden te verschuiven naar links of naar rechts. De roteer-operatie laat toe bitwoorden te roteren, naar links of naar rechts. Elk van deze operaties heeft twee parameters: het aantal bits waarover geroteerd of geschoven wordt en de richting waarin dit moet gebeuren.

De S-box-operatie is de meest complexe operatie. Een substitutie-box vervangt het bitwoord aan de ingang door een ander bitwoord. Dit gebeurt aan de hand van een opzoekingstabel. Indien deze substitutie inverteerbaar is, is deze substitutie een permutatie. Verder wordt het aantal bits aan de uitgang bijgehouden. Om het aantal ingangsbits bij te houden worden twee variabelen voorzien. Zij komen overeen met het aantal bits horende bij de verschillende rijen, respectievelijk kolommen, in een tabelvoorstelling van een S-box. De som van beide is het aantal bits aan de ingang. Meer informatie over een tabelvoorstelling van een S-box is te vinden in bijlage C.

De bit-herverdeel-operatie (bit shuffle) maakt het mogelijk om de bits in het ingangsbitwoord te herverdelen over het uitgangsbitwoord. Het uitgangsbitwoord hoeft niet noodzakelijk uit evenveel bits te bestaan als het ingangsbitwoord. In het programma zijn er drie soorten bit-herverdelers:

Choice Permutation : Indien het aantal bits van het ingangsbitwoord groter is dan het aantal bits van het uitgangsbitwoord, dan wordt de term ‘choice permutation’ gebruikt. Ook wanneer het aantal bits in het ingangsbitwoord gelijk is aan het aantal

bits in het uitgangsbitwoord, maar de bit-herverdeel-operatie niet inverteerbaar is, wordt deze term gebruikt. Dit type bit-herverdeler wordt onder andere gebruikt in het sleutelgeneratie-algoritme van DES [16].

Permutation of bits : Indien het aantal bits in het ingangsbitwoord gelijk is aan het aantal bits in het uitgangsbitwoord, waarbij elke bit van het ingangsbitwoord exact eenmaal gebruikt wordt in het uitgangsbitwoord, spreekt men van een permutatie box (P-box). In het programma wordt de term 'permutation of bits' gebruikt. Merk op dat een P-box operatie een speciaal geval is van een substitutie-box.

Expansion Permutation : Indien het aantal bits in het ingangsbitwoord kleiner is dan het aantal bits in het uitgangsbitwoord, wordt de term 'expansion permutation' gebruikt. Dit type bit-herverdeler wordt onder andere gebruikt in de Feistel functie van het DES-algoritme [16].

De gegevens die de S-box-operatie en de bit-herverdeel-operatie bepalen, worden in een afzonderlijk bestand opgeslagen. Op deze manier wordt het mogelijk enkele standaard operaties te voorzien in het programma. De gebruiker kan deze standaard S-box- en bit-herverdeel-operaties op elk moment gebruiken. De parameters horende bij de andere operaties worden bij het opslaan van een operatienetwerk mee in het bestand van de scène opgeslagen. Meer details over het opslaan van operatienetwerken, S-boxen en bit-herverdelers is te vinden in hoofdstuk 6.4.

3.8 Implementatie

Tijdens het implementeren van de netwerkstructuur waren er enkele problemen. Het ene probleem was al hardnekkiger dan het andere, maar er was toch één probleem dat er bovenuit stak.

Het genoemde probleem is eigenlijk gewoon een eigenschap van de taal. In C++ worden tijdens de constructie van een object methodes nooit dynamisch aangeroepen.

De lezer herinnert zich de erfelijkheidsstructuur van een blok uit figuur 3.4. Stel dat een virtuele methode bepaalde variabelen van de klasse Node initialiseert bij constructie. Deze methode wordt dus opgeroepen in de abstracte klasse Node. In de abstracte klasse Node is een implementatie van deze initialisatiemethode voorzien. In de klasse ComposedNode (een samengesteld blok) wordt deze methode opnieuw gedefinieerd; de methode is immers virtueel. Men zou er vanuit kunnen gaan dat bij constructie van een samengesteld blok, de initialisatiemethode uit de klasse ComposedNode opgeroepen wordt vanuit de constructor in Node. Dit is echter niet het geval. Ongeacht het type van het nieuwe blok, altijd zal de initialisatiemethode uit de klasse Node worden opgeroepen.

Indien men deze initialisatiemethode afhankelijk wil maken van het type, moet men de oproep van deze methode verplaatsen naar een constructor op een lager niveau in de erfelijkheidsstructuur. Deze oproep moet dus in heel wat verschillende constructoren

worden geschreven.

De besproken eigenschap van de taal C++ heeft ook een belangrijk voordeel. De dynamische binding tijdens constructie wordt niet zomaar verhinderd. De programmeur wordt immers beschermd tegen een bepaald type fout. In één van vorige paragrafen werd het voorbeeld gegeven van een virtuele initialisatiemethode die zowel in de klasse Node als in de klasse ComposedNode werd geïmplementeerd. Stel nu dat de initialisatiemethode in de klasse ComposedNode per ongeluk variabelen gebruikt die pas geïnitialiseerd worden in de constructor van ComposedNode. Hierdoor dreigt een segmentatiefout, want zonder de genoemde beperking van C++, zou hier de initalisatiemethode in ComposedNode opgeroepen worden, nog voor de constructor in ComposedNode de in de initalisatiemethode gebruikte variabelen heeft geïnitialiseerd. Merk wel op dat dit probleem zich alleen zou voortdoen indien de programmeur onoplettend zou zijn.

Het detecteren van deze eigenschap was niet vanzelfsprekend, bij het compileren wordt immers geen waarschuwing gegeven. Het is voor een onervaren C++-programmeur moeilijk om in te zien dat het type van een nieuw object bij constructie niet in rekening wordt genomen. In andere programmeertalen zoals C# [22] en Java [10] gebeurt dit trouwens wel.

3.9 Besluit

In dit hoofdstuk werd het ontwerp van de netwerkstructuur van een operatienetwerk voorgesteld. Dit gebeurde op een opbouwende manier, de belangrijkste ontwerpbeslissingen kwamen aan bod. Het ontwerp voldoet aan de vereisten die gesteld werden in sectie 2.3. Bovendien kan men operaties en samengestelde blokken linken, wat de gebruiksvriendelijkheid zeker ten goede komt. Alle nodige operaties werden voorzien, ook operaties in een Galois veld zijn beschikbaar.

In dit hoofdstuk werd niet uitgelegd hoe een berekening van ingangen naar uitgangen verloopt. In het volgende hoofdstuk wordt aandacht besteed aan dit berekeningsproces.

Hoofdstuk 4

Berekeningen maken

4.1 Inleiding

In dit hoofdstuk wordt uitgelegd hoe berekeningen doorheen een operatienetwerk worden uitgevoerd. In sectie 4.2 wordt een bespreking gegeven van de verschillende wijzen waarop de berekeningen kunnen gebeuren. De voor- en nadelen van de verschillende mogelijkheden worden afgewogen tegenover elkaar. Op basis daarvan wordt de berekeningsmethode gekozen. In sectie 4.3 wordt uitgelegd welke methodes gebruikt worden om deze berekeningsmethode te implementeren. Er is extra aandacht voor berekeningen die zich niet beperken tot één scène. In de laatste sectie wordt een conclusie geformuleerd.

4.2 Keuze van de berekeningsmethode

Het bitwoord dat hoort bij een verbindingslijn wordt bijgehouden door een object van de klasse Connectie. De blokken zelf houden geen bitwoorden bij. Voor de eenvoud wordt even aangenomen dat alle berekeningen zich in één scène afspelen. Er zijn dus geen samengestelde blokken met ingekapselde scènes. Er zijn een drietal vrij eenvoudige manieren om berekeningen doorheen een operatienetwerk uit te voeren:

1. Vertrek van een vector met alle blokken. Itereer over deze vector door achtereenvolgens elk blok te vragen of alle nodige ingangen zijn aangekomen. Indien dit het geval is, krijgt het blok de opdracht om haar uitgangen te berekenen. Vervolgens wordt het blok verwijderd uit de vector. Het iteratieproces wordt stopgezet wanneer de vector leeg is.
2. Vertrek van een vector met alle uitgangsblokken. Activeer alle blokken in de vector eenmaal. Een geactiveerd uitgangsblok vraagt aan de blokken die verbonden zijn met zijn ingangen, om hun uitgangen te berekenen. Op deze manier wordt een

recursieve opdracht gestart. De berekeningsopdracht propageert naar de ingangen. Vervolgens propageren de berekeningsresultaten terug naar de uitgangen.

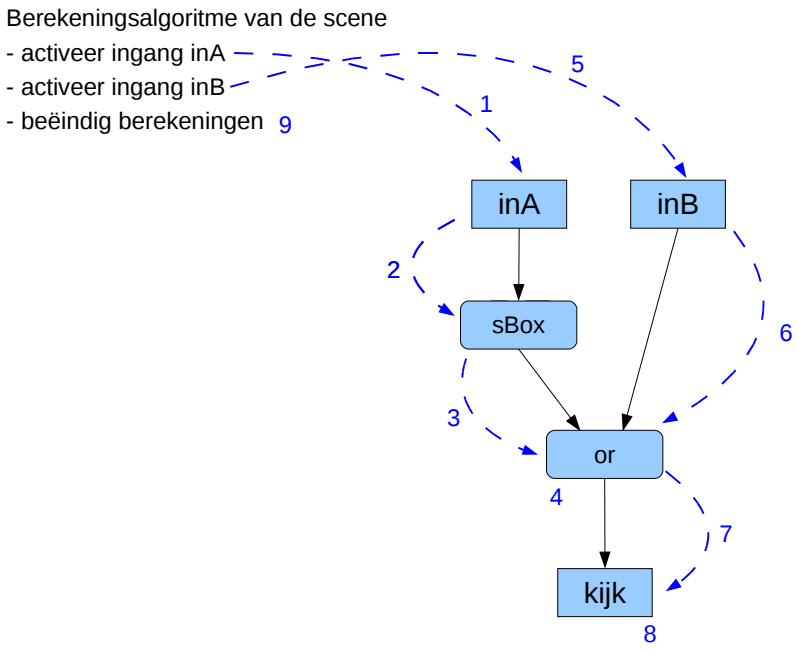
3. Vertrek van een vector met alle ingangsblokken. Activeer alle blokken in de vector eenmaal. Een geactiveerd ingangsblok geeft zijn waarde door aan het blok dat met zijn uitgang is verbonden. Op deze manier wordt een recursieve opdracht gestart. Indien een blok alle ingangen heeft ontvangen, berekent dit blok al zijn uitgangen. Indien alle blokken in het operatienetwerk correct zijn verbonden, zullen de uitgangen van alle blokken, ten laatste nadat alle ingangsblokken in de vector werden geactiveerd, beschikbaar zijn.

De eerstgenoemde methode is eenvoudig te implementeren. Zij is echter kwadratisch in orde, de uitvoeringstijd neemt kwadratisch toe met het aantal blokken, in scènes met veel blokken. De eerste methode is dus weinig efficiënt. De tweede methode lijkt een goede oplossing, maar er is een belangrijk probleem. De bedoeling is om alle mogelijke tussenresultaten te berekenen. Dit is één van de vereisten die geformuleerd werden in sectie 2.3. Indien er geen uitgangsblokken zijn, berekent de tweede methode echter niets. Bijgevolg kan de tweede methode onmogelijk voldoen aan de gestelde vereisten. De derde methode vertrekt bij de ingangen. Zij garandeert dat alle mogelijke tussenresultaten berekend worden. Bovendien is deze methode net als de tweede methode lineair in orde, de uitvoeringstijd neemt lineair toe met het aantal blokken, in scènes met veel blokken.

De derde methode wordt dus gekozen. De lezer kon deze keuze ergens al vermoeden, aangezien in onderdeel 3.3.7, in een scène, twee vectoren met enerzijds de I/O-ingangen en anderzijds de normale ingangen werden ingevoerd.

De beschrijving van de gekozen methode wordt nu met een voorbeeld verduidelijkt. In figuur 4.1 worden de verschillende stappen in een eenvoudig berekeningsproces verduidelijkt. De verschillende stappen in dit proces worden hieronder nader gespecifieerd:

1. De scène ontvangt een berekeningsopdracht. De ingangen worden overlopen. Er zijn twee ingangen. De eerste ingang wordt geactiveerd. Ingangsblok `inA` krijgt dus een berekeningsopdracht.
2. Ingangsblok `inA` geeft aan operatieblok `sBox` de opdracht een berekeningsopdracht te starten. De uitgang van `inA` is immers verbonden met de ingang van operatieblok `sBox`.
3. Operatieblok `sBox` ontvangt de berekeningsopdracht. Er wordt gecontroleerd of alle ingangen beschikbaar zijn, zodat de berekening kan worden gestart. Aangezien een S-box-operatie slechts één ingang heeft, is dit het geval. De berekening wordt gedaan en het resultaat wordt beschikbaar gesteld. Vervolgens vraagt operatieblok `sBox` aan operatieblok `or` een berekeningsopdracht te starten.
4. Operatieblok `or` ontvangt de berekeningsopdracht. Er wordt gecontroleerd of alle ingangen beschikbaar zijn, zodat de berekening kan worden gestart. Dit is niet het geval, want de OR-operatie heeft slechts één van haar twee ingangen ontvangen. De eerste recursieve berekeningsopdracht, die gestart werd bij ingangsblok `inA`, kent hier haar einde.



Figuur 4.1: Illustratie van de berekeningsmethode van een eenvoudig operatienetwerk.

5. De scène activeert de tweede ingang. Ingangsblok `inB` ontvangt dus een berekeningsopdracht.
6. Ingangsblok `inB` geeft aan operatieblok `or` de opdracht een berekeningsopdracht te starten. De uitgang van `inB` is immers verbonden met de tweede ingang van operatieblok `or`.
7. Operatieblok `or` ontvangt zijn tweede berekeningsopdracht. Er wordt gecontroleerd of alle ingangen beschikbaar zijn, zodat de berekening kan worden gestart. Beide ingangen zijn nu beschikbaar. De berekening wordt gedaan en het resultaat wordt beschikbaar gesteld. Vervolgens vraagt operatieblok `or` aan uitgangsblok `kijk` een berekeningsopdracht te starten.
8. Uitgangsblok `kijk` heeft geen uitgangen en dient niets te doen. De tweede recursieve berekeningsopdracht, die gestart werd bij ingangsblok `inB`, kent hier haar einde.
9. De scène beëindigt haar berekeningsopdracht. Alle tussenresultaten en eindresultaten van de berekeningsopdracht zijn beschikbaar in het operatienetwerk.

Op het ogenblik dat alle ingangen van een samengesteld blok beschikbaar zijn, worden de berekeningen in de ingekapselde scene eerst uitgevoerd. De gekozen methode gebruikt dus een gewijzigde vorm van het diepte-eerst-algoritme. Het diepte-eerst-algoritme wordt immers niet strikt toegepast, aangezien de berekeningen in de ingekapselde scene slechts van start gaan wanneer alle ingangen van het samengestelde blok beschikbaar zijn. In de volgende sectie wordt de concrete implementatie van de gekozen methode besproken.

4.3 Implementatie van de berekeningsmethode

Deze sectie is opgesplitst in drie delen. Het eerste deel is beperkt tot scènes zonder samengestelde blokken. In het tweede deel wordt uitgelegd hoe wordt omgegaan met berekeningsopdrachten over verschillende scènes heen. In het derde deel wordt een voorbeeld gegeven.

4.3.1 Berekeningen in een scène

In dit onderdeel worden samengestelde blokken nog niet beschouwd. Dit gebeurt pas in het volgende onderdeel.

Om berekeningen te kunnen uitvoeren, krijgt elk blok een methode genaamd *calculateRecursively*. Elk type blok heeft een eigen implementatie voor deze methode. Deze methode wordt voor elk ingangsblok in de scène eenmaal opgeroepen. Deze methode geeft telkens een waarde terug die feedback geeft over het berekeningsproces. Deze feedback kan slechts drie waarden aannemen. Ofwel zijn de berekeningen correct verlopen, ofwel zijn ze correct verlopen maar is een bepaalde uitgang niet verbonden, ofwel is er een fout opgetreden. Belangrijk is dat elke connectie ook een methode *calculateRecursively* heeft. Een blok zal een blok dat met één van zijn uitgangen is verbonden nooit zelf een berekeningsopdracht geven. Dit gebeurt via de connectie. De connectie zal de berekeningsopdracht op haar beurt doorsturen naar het blok aan de andere kant van de connectie. Een connectie houdt bij of er al dan niet een berekeningsopdracht is voorbij gekomen.

Elke blok krijgt tevens een variabele die bijhoudt hoeveel berekende ingangen reeds beschikbaar zijn. Een blok zal een berekeningsopdracht immers alleen uitvoeren en doorgeven, indien het zelf al zijn ingangen heeft ontvangen.

Het berekeningsproces wordt gestart vanuit een methode in de scène. Vooraleer de nieuwe berekeningsopdracht werkelijk van start gaat, dienen alle hierboven vermelde variabelen gereset te worden. Tijdens het berekenen wordt informatie over het berekeningsproces gelogd. Indien de beide bitwoorden aan de ingang van een OR-operatieblok bijvoorbeeld een verschillend aantal bits hebben, wordt een opmerking toegevoegd aan de log. Nadat het berekeningsproces in de scène is voltooid, wordt gecontroleerd of alle blokken in de scène betrokken werden bij de berekening. Opmerkingen hierover worden toevoegd aan de log. Ook de waardes die de verschillende *calculateRecursively*-oproepen terug hebben gegeven, worden gebruikt in de log. Het is de bedoeling dat bepaalde opmerkingen of foutmeldingen niet alleen gelogd worden, maar ook grafische gevolgen hebben in de betrokken blokken. Mogelijke grafische consequenties van het berekeningsproces worden besproken in onderdeel 5.3.2. Een voorbeeld van een berekeningsproces met bijhorende log bevindt zich in onderdeel 4.3.3.

4.3.2 Berekeningen over verschillende scènes heen

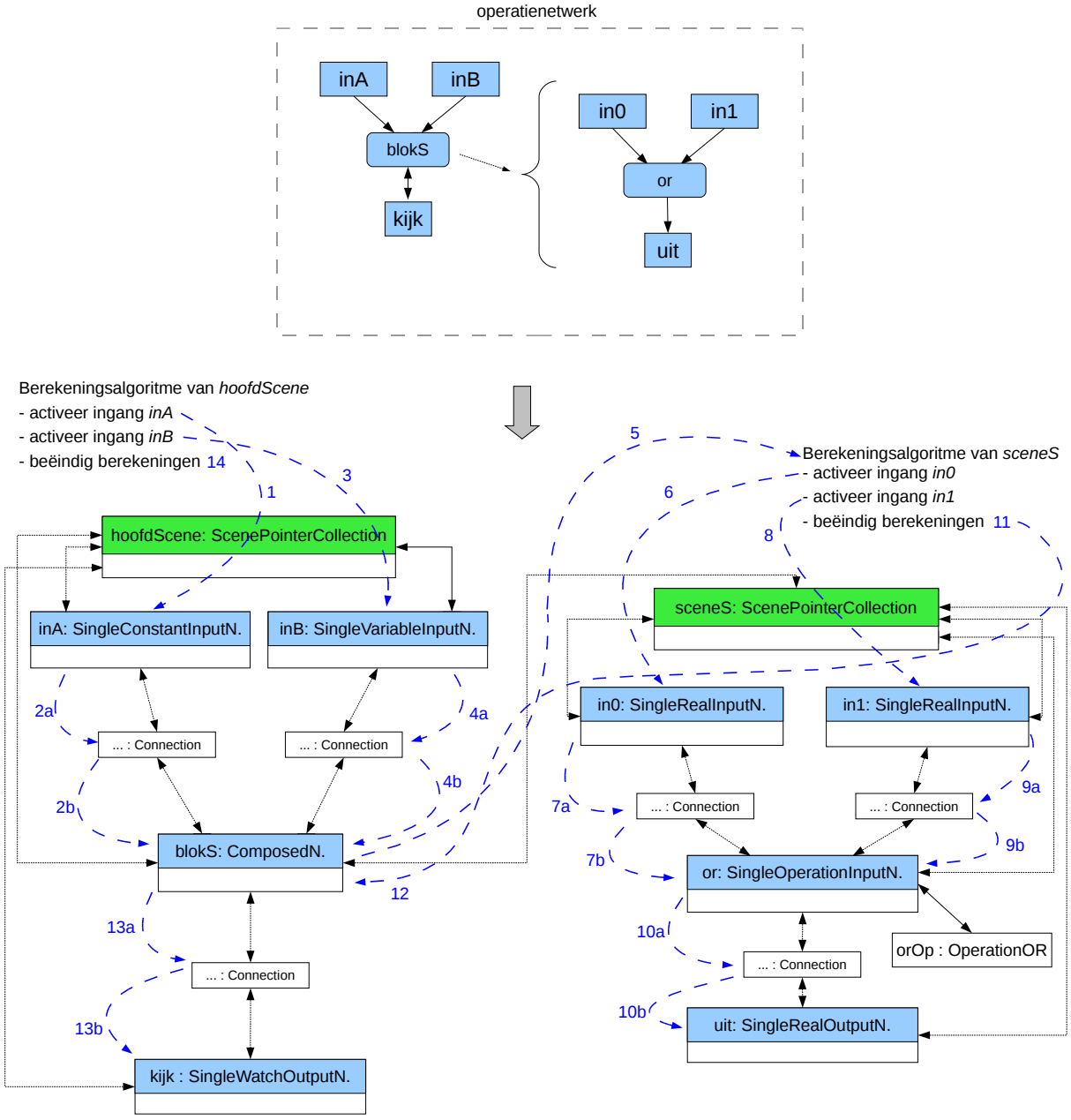
Indien een berekeningsopdracht aan een samengesteld blok wordt gegeven, dient deze opdracht doorgegeven te worden aan de ingekapselde scène. Dit gebeurt op het moment dat alle ingangen van het samengestelde blok beschikbaar zijn. Belangrijk is dat de bitwoorden aan de ingangen van het samengestelde blok moeten worden overgezet naar de I/O-ingangen van de ingekapselde scène. Het berekeningsproces in de ingekapselde scène verloopt op dezelfde manier als het berekeningsproces in een gewone scène. Op het einde van dit proces moeten de bitwoorden die horen bij de I/O-uitgangen van de ingekapselde scène worden overgezet naar de uitgangen van het samengestelde blok. Daarna kan het berekeningsproces in de scène worden verdergezet.

In het volgende onderdeel vindt de lezer een voorbeeld die de besproken manier van werken verduidelijkt.

4.3.3 Een voorbeeld van het berekeningsproces

In deze sectie worden de verschillende stappen in het berekeningsproces van een voorbeeld-operatienetwerk verduidelijkt. Dit gebeurt aan de hand van het objectdiagram van een eenvoudig operatienetwerk, waarop de opeenvolgende stappen in het berekeningsproces zijn aangeduid. Het objectdiagram is voorgesteld in figuur 4.2. Het berekeningsproces wordt nu overlopen; de nummering komt overeen met die van in de figuur.

1. De berekening wordt gestart door de eerste ingang te activeren.
2. Via de uitgangsconnectie van de eerste ingang, komt het samengestelde blok te weten dat de eerste ingang beschikbaar is.
3. De tweede ingang wordt geactiveerd.
4. Via de uitgangsconnectie van de tweede ingang, komt het samengestelde blok te weten dat de tweede ingang beschikbaar is.
5. Het berekeningsalgoritme in de ingekapselde scène wordt gestart.
6. De eerste I/O-ingang van de ingekapselde scène wordt geactiveerd. Het bitwoord dat hoort bij de eerste ingang van het samengestelde blok wordt meegegeven.
7. De eerste I/O-ingang geeft de berekeningsopdracht via zijn uitgangsconnectie door aan het OR-operatieblok.
8. De tweede I/O-ingang van de ingekapselde scène wordt geactiveerd. Het bitwoord dat hoort bij de tweede ingang van het samengestelde blok wordt meegegeven.
9. De tweede I/O-ingang geeft de berekeningsopdracht via zijn uitgangsconnectie door aan het OR-operatieblok.
10. Dit operatieblok heeft nu alle ingangen ontvangen en plaatst het berekende bitwoord op de uitgangsconnectie van het OR-operatieblok. Via de uitgangsconnectie van het OR-operatieblok, krijgt de enige I/O-uitgang nu een berekeningsopdracht.
11. Deze I/O-uitgang kan niets meer doen, de berekeningen in de ingekapselde scène zijn voorbij.



Figuur 4.2: Illustratie van de berekeningsmethode aan de hand van het objectdiagram van een eenvoudig operatienetwerk. Het operatienetwerk bevindt zich bovenaan.

12. Het programma gaat terug naar de berekeningsmethode in het samengestelde blok. Het samengestelde blok vraagt aan de I/O-uitgang in de ingekapselde scène of de berekeningen zijn aangekomen. Dit is het geval. Het bitwoord in de ingangsconnectie van de I/O-uitgang wordt opgevraagd en gekopieerd naar de eerste en enige uitgangsconnectie van het samengestelde blok.
13. Vervolgens geeft het samengestelde blok een berekeningsopdracht aan haar uitgangsconnectie, die op haar beurt een berekeningsopdracht geeft aan het uitgangsblok.
14. Het uitgangsblok kan niets meer doen. De berekeningen worden beëindigd.

Merk op dat elk van de twee in de scène gestarte berekeningen een resultaat teruggeeft. Aangezien alles correct is verlopen, zullen deze twee resultaten allebei het succes van de berekeningen bevestigen. Hieronder vindt de lezer de log na het uitvoeren van het daarnet beschreven berekeningsproces:

```
 >>> START CALCULATION [level 1]
Start calculation propagation
Start activating normal inputs

 >>> START CALCULATION [level 2]
 Start calculation propagation
 Start activating normal inputs
 Start activating I/O-inputs
 End calculation propagation
 Start enumerating connection problems
 End enumerating connection problems
 CalcResult = Calculation succeeded!
 >>> END CALCULATION [level 2]

 Start activating I/O-inputs
 End calculation propagations
 Start enumerating connection problems
 End enumerating connection problems

 CalcResult = Calculation succeeded!
 >>>END CALCULATION [level 1]
```

Een gebruiker die de log leest, wordt geconfronteerd met de manier waarop de berekeningen gebeuren. Dit helpt de gebruiker om sneller te vinden waar een eventuele fout zich bevindt. De gebruiker kan de output in de log beperken, indien hij niet wenst geconfronteerd te worden met de manier van berekenen. Informatie over fouten wordt sowieso gelogd. De beperkte log-output ziet er als volgt uit:

```
>>>START CALCULATION [level 1]

>>> START CALCULATION [level 2]
CalcResult = Calculation succeeded!
 >>> END CALCULATION [level 2]

CalcResult = Calculation succeeded!
 >>> END CALCULATION [level 1]
```

In het begin en op het einde van berekeningen in een scène wordt telkens vermeld op welk niveau (*level*) de berekeningen plaats vinden. Het niveau één verwijst naar de hoofdscène, deze scène is niet ingekapseld. Het niveau twee verwijst naar een scène die ingekapseld is in een samengesteld blok in de hoofdscène. Indien er zich tijdens het berekeningsproces bepaalde problemen voortdoen, wordt daarvan melding gemaakt in de log. Hoe deze log in de applicatie wordt weergegeven, wordt besproken in sectie 7.5.

4.4 Besluit

In dit hoofdstuk werd een keuze gemaakt uit enkele eenvoudige algoritmes om berekeningen in een operatienetwerk uit te voeren. De berekeningen starten bij de ingangen en planten zich volgens een aangepaste vorm van het diepte-eerst-algoritme voort naar de uitgangen. Op deze manier zijn alle mogelijke tussenresultaten beschikbaar. Het berekeningsproces werd geïllustreerd aan de hand van een voorbeeld.

In sectie 7.5 wordt uitgelegd hoe de gebruiker berekeningen kan uitvoeren gebruikmakende van de grafische gebruikersinterface van de applicatie.

Hoofdstuk 5

Uitbreiding van de netwerkstructuur met grafische informatie

5.1 Inleiding

In dit hoofdstuk wordt de structuur van een operatienetwerk uit hoofdstuk 3 uitgebreid met klasses die grafische informatie bevatten. Het resultaat is één grote netwerkstructuur die zowel de grafische als de data-informatie bevat. De klasses uit de netwerkstructuur die in hoofdstuk 3 werden besproken, worden voortaan de datastructuur genoemd. De klasses uit de netwerkstructuur die de grafische informatie bevatten vormen de grafische structuur. De datastructuur en de grafische structuur vormen samen de volledige netwerkstructuur. In sectie 5.2 wordt het ontwerp van deze grafische structuur besproken. Het verband met de datastructuur komt uitgebreid aan bod. In sectie 5.3 wordt de grafische voorstelling van alle elementen in een operatienetwerk voorgesteld. Daarna komen enkele enkele implementatieproblemen aan bod, dit gebeurt in sectie 5.4. In de laatste sectie wordt een conclusie geformuleerd.

5.2 Ontwerp van de grafische structuur

5.2.1 Grafische informatie

In dit hoofdstuk wordt de term grafische informatie meermaals gebruikt. Onder grafische informatie wordt verstaan alle informatie die niet vervat zit in de structurele voorstelling van een operatienetwerk. Meer concreet wordt onder grafische informatie verstaan:

- De layout en het uitzicht van de scène.
- De grafische weergave van een blok en de verbindingslijnen tussen de blokken.

- De plaats waar verbindingslijnen aankomen bij een blok of vertrekken naar een ander blok.
- Markeringen op een operatieblok, ten gevolge van selectie of een calculatiefout.
- Markeringen om aan te tonen welke blokken gelinkt zijn.

5.2.2 Scheiding van data- en grafische informatie

In hoofdstuk 3 werden de klasses om een operatienetwerk te kunnen voorstellen besproken. Deze klasses volstaan als voorstelling van een operatienetwerk in het geheugen, alle data-informatie wordt erin geherbergd. Al deze klasses vormen de datastructuur van een operatienetwerk. De in hoofdstuk 3 en 4 besproken voorbeelden tonen aan dat deze datastructuur gebruikt kan worden om een operatienetwerk op te bouwen en berekeningen uit te voeren. Dit volstaat echter niet. Het is de bedoeling dat de gebruiker beschikt over een visuele voorstelling van een operatienetwerk. De gebruiker moet het operatienetwerk ook kunnen aanpassen.

Men kan overwegen de grafische informatie in de reeds bestaande datastructuur op te slaan. De grafische informatie wordt in dat geval opgeslagen in de reeds ontworpen klasses. Indien de grafische informatie vermengd wordt met de data-informatie, wordt de code bijzonder onoverzichtelijk. Indien beide structuren gescheiden worden, is de code veel overzichtelijker. Tegerlijkertijd wordt de code ook complexer. Er werd gekozen voor een scheiding van de data- en de grafische informatie.

Een scheiding van beide structuren suggereert dat de grafische structuur en de datastructuur afzonderlijk gebruikt kunnen worden. Dit zou willen zeggen dat een operatienetwerk getekend kan worden, zonder dat de datastructuur in de achtergrond parallel aanwezig is. Ook het omgekeerde zou gelden. Geen van beide is het geval. Indien beide structuren parallel gebruikt worden, zoals het geval is in het programma, moet een bepaalde verandering in de grafische structuur, ook een verandering in de parallel aanwezige datastructuur teweegbrengen. Op deze manier is het operatienetwerk direct klaar om berekeningen uit te voeren.

Een kritische lezer kan opmerken dat dit geen noodzakelijke voorwaarde is. Men zou het programma zo kunnen ontwerpen dat er helemaal geen datastructuur is, terwijl de gebruiker een operatienetwerk tekent met de muis. Indien de gebruiker vervolgens een berekening start, kan de datastructuur gegenereerd en gebruikt worden bij de berekeningen. Maar zonder datastructuur zijn er geen bitwoorden, bevatten de ingangsblokken dus geen bitwoorden en kunnen de berekeningen niet plaats vinden. Bovendien zal de gebruiker tijdens het tekenen bepaalde (constante) ingangen willen voorzien van een bitwoord. Dit laatste vereist het bestaan van een datastructuur.

Gezien dit programma toch gemaakt wordt om berekeningen uit te voeren, wordt ervoor gekozen de datastructuur en de grafische structuur altijd naast elkaar te laten bestaan. Ook de grafische informatie van de blokken in de scènes die niet zichtbaar zijn, bijvoorbeeld in een ingekapselde scène, zijn voortdurend aanwezig in het geheugen. Een nadeel van

deze manier van werken is dus een hoog geheugengebruik, vooral wanneer heel uitgebreide operatienetwerken gebruikt worden. Er zijn uiteraard ook voordelen. Het visualiseren van ingekapselde scènes kan heel snel en de berekeningen kunnen direct van start gaan, zonder dat eerst nieuwe objecten in de structuur gegenereerd moeten worden. Bovendien zijn grafische aanduidingen over berekeningsfouten in ingekapselde scènes onmiddellijk zichtbaar, wanneer men na het berekenen overschakelt naar een ingekapselde scène. Dit is heel belangrijk aangezien het illustratieve software betreft.

5.2.3 De grafische structuur

In deze sectie wordt de grafische structuur voorgesteld. Dit gebeurt aan de hand van een klein objectdiagram in figuur 5.1. Het objectdiagram hoort bij de netwerkstructuur die zich bovenaan in de figuur bevindt. De datastructuur bevindt zich links, de grafische structuur bevindt zich rechts in de figuur. Bindingen tussen beide structuren zijn in het blauw aangeduid. In de volgende delen worden de in de figuur aangeduide relaties tussen de datastructuur en de grafische structuur verder verduidelijkt. Merk op dat de namen van de grafische klasses beginnen met de letters QE. De klasses die deel uit maken van de Qt zelf beginnen met de letter Q. De afkorting QE staat voor Q-Extended.

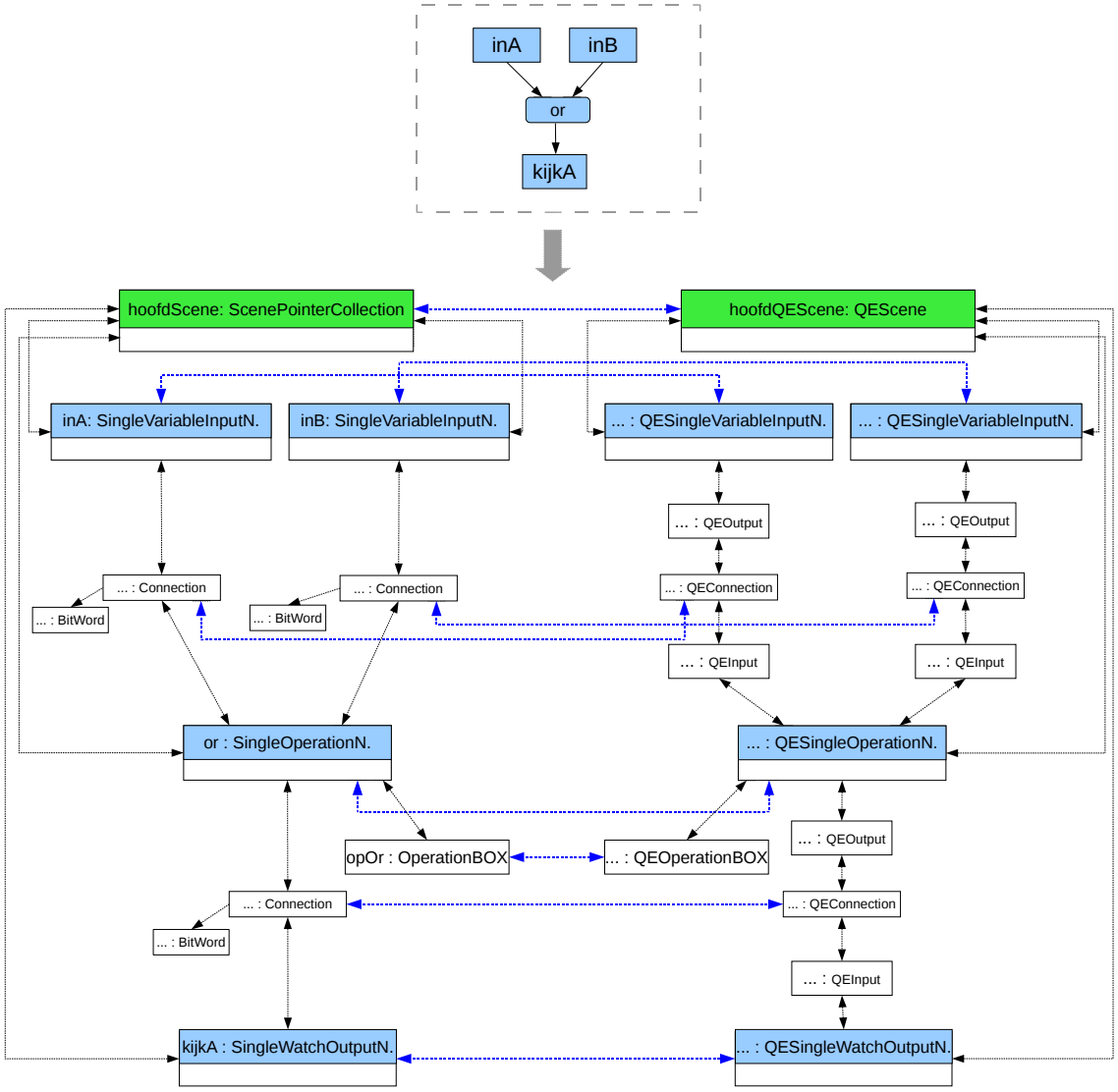
De grafische structuur wordt in de volgende paragrafen besproken aan de hand van het objectdiagram in figuur 5.1.

De scène

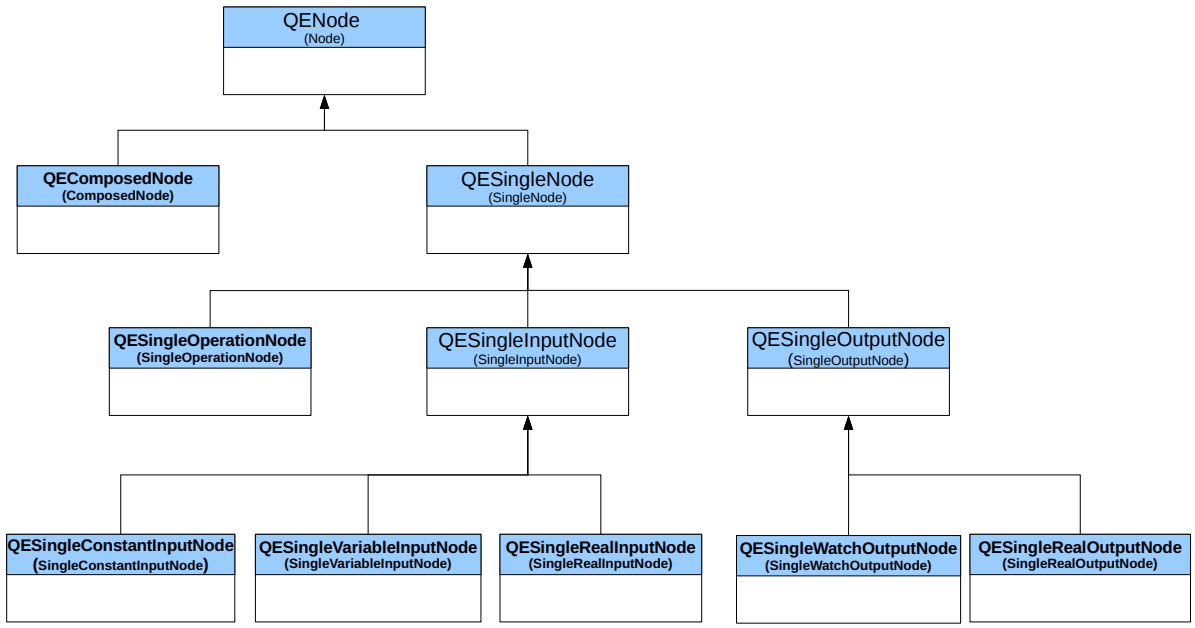
De grafische equivalent van een ScenePointerCollection (scène) is een QEScene. Er is een dubbele binding tussen beide. Opdat een scène weergegeven zou kunnen worden, erft de klasse QEScene over van de klasse QGraphicsScene [23], deze klasse maakt deel uit van de QGraphics View module van Qt [24]. In de klasse QEScene worden alle aanwezige blokken (QENode) en connecties (QEConnection) bijgehouden. Typische methodes in QEScene zijn methodes om blokken toe te voegen of om een groep blokken samen te selecteren. Alle input van de muis of het toetsenbord komt aan in de klasse QEScene en wordt eventueel doorgestuurd naar de correcte blokken.

Erfelijkheidsstructuur van een blok

In het objectdiagram in figuur 5.1 zijn drie verschillende soorten blokken aanwezig. Het betreft een operatieblok, twee ingangsblokken met variabele ingang en een observatie-uitgang. In hoofdstuk 3, meer bepaald in figuur 3.4, kwam de volledige erfelijkheidsstructuur van een blok aan bod. De grafische structuur heeft een gelijke erfelijkheidsstructuur. Deze is afgebeeld in figuur 5.2. De namen van de niet-abstracte klasses staan in het vet. Tussen haakjes staat telkens de overeenkomstige naam uit de datastructuur. Het equivalent van een Node (blok) is een QENode. Beide zijn telkens bidirectioneel gebonden.



Figuur 5.1: Een voorbeeld van een operatienetwerk, met bijhorend objectdiagram. De objecten die horen bij de datastructuur bevinden zich links, de objecten die horen bij de grafische structuur bevinden zich rechts.



Figuur 5.2: Erfelijkheidsstructuur van de klasse QENode, met telkens tussen haakjes de equivalente klasse uit de datastructuur.

De klassenamen QESingleVariableInputNode, QESingleOperationNode en QESingle-WatchOutputNode uit het objectdiagram in figuur 5.1, vindt men terug in figuur 5.2.

De connecties tussen blokken

De grafische equivalent van een Connection (connectie) is een QEConnection. In het objectdiagram in figuur 5.1 is te zien dat er zich tussen een QENode en een QEConnection nog een andere object bevindt, namelijk een object van de klasse QEInput of QEOutput. Beide erven over van de abstracte klasse QEInputOutput, waarin de gemeenschappelijke methodes geïmplementeerd zijn. Alle genoemde klasses erven over van de klasse QGraphicsItem [25], deze klasse maakt deel uit van de QGraphics View module van Qt [24]. Objecten van de klasse QGraphicsItem kunnen getoond worden in een QEGraphicsScene. In dit programma kunnen een QEConnection, een QENode, een QEInput en een QEOutput dus getoond worden in QEScene

De klasses QEInput en QEOutput komen met geen enkele klasse uit de datastructuur overeen. Objecten van deze klasses zijn als het ware een onderdeel van een blok, maar ze bepalen zelf hoe ze eruit zien en hoe ze reageren op bepaalde gebeurtenissen. In het programma zien ze eruit als blauwe cirkels. In de klasses QEInput en QEOutput zijn speciale reacties op bepaalde muisgebeurtenissen geïmplementeerd. Indien men klikt op een uitgang van een blok en de muis vervolgens versleept, kan men een connectielijn trekken naar de ingang van een andere blok. Deze connectielijn verschijnt tijdens het

slepen op het scherm. Indien men klikt op een blok, maar niet ter hoogte van een ingang of uitgang, en de muis vervolgens versleept, gebeurt er niets. Juist om dit verschil in reactie objectgericht te implementeren, worden de klasses QEInput en QEOutput gebruikt.

In de datastructuur is het zo dat de connectie die hoort bij de uitgang van een blok altijd aanwezig is, zelfs al is de uitgang niet verbonden. In het grafische gedeelte geldt hetzelfde. Toch is er een klein verschil, want in de grafische structuur wordt de connectie (QEConnectie) bijgehouden door een QEOutput. Een QEOutput zal bij constructie dus onmiddellijk een QEConnection aanmaken. Deze laatste wordt vervolgens dubbel gebonden met de connectie uit de datastructuur.

De operatie

De grafische weergave van een operatieblok wordt niet alleen bepaald door het operatieblok, maar ook door het type operatie en de parameters van de operatie. Er zijn twee mogelijkheden om hiermee rekening te houden.

In de methode die het operatieblok tekent, kan men eerst onderzoeken over welke operatie het gaat en gebruikmakende van de parameters van het type operatie, het blok een bepaald uiterlijk geven. Dit resulteert echter in weinig flexibiliteit.

Meer flexibiliteit bekomt men indien men elke operatie voorziet van een grafische klasse QEOperation. De klasse is abstract en bevat enkele verplichte methodes. Elk type operatie heeft haar eigen subklasse en implementeert de verplichte methodes. Er is een grote flexibiliteit. Elke operatie bepaalt zelf hoe ze eruit ziet. De klasse OperationOR bijvoorbeeld heeft als equivalent de klasse QEOperationOR.

In een operatieblok wordt de tekenopdracht dus gedeeltelijk doorgegeven aan een QEOperation. Hoe het operatieblok eruit ziet, hangt vooral af van de verschillende implementaties in QEOperation, maar hangt ook af van bepaalde markeringen op het blok. Indien het operatieblok geselecteerd is, wordt het immers op een andere manier weergegeven. Zo zijn er nog een aantal markeringen die te maken hebben met feedback over een berekeningsproces. Een lijst van de mogelijke markeringen is te vinden in onderdeel 5.3.2.

5.2.4 Constructie, wijziging en destructie van de data- en grafische structuur

In onderdeel 5.2.2 werd uitgelegd waarom de data- en de grafische structuur altijd naast elkaar aanwezig zijn. In deze sectie wordt kort toegelicht hoe een volledig operatienetwerk in het geheugen wordt opgebouwd. Eerst wordt de constructie besproken, vervolgens de destructie en tot slot wordt uitgelegd op welke manier een operatienetwerk wijzigingen ondergaat.

Constructie

De constructie vertrekt vanuit de datastructuur. De opbouw van een operatienetwerk zoals dat gebeurde in sectie 3.4 blijft volledig geldig.

Indien een scène geconstrueerd wordt, gebeurt dit vanuit de datastructuur. De QEScene wordt automatisch gecreëerd. Om een nieuw blok toe te voegen aan de scène, wordt de constructor van de klasse Node opgeroepen en vervolgens moet het blok worden toegevoegd aan de ScenePointerCollection. Wanneer het object van de klasse Node geconstrueerd wordt, gebeurt de constructie van het object van de klasse QENode automatisch. Het toevoegen van de Node aan de ScenePointerCollection geeft als onmiddellijk gevolg dat ook de overeenkomstige QENode toegevoegd wordt aan de QEScene.

Destructie

De destructie van een blok, een connectie of een deel van een netwerk gebeurt vanuit de grafische structuur. Dit is niet onlogisch, aangezien constructoren en destructoren in C++ in omgekeerde volgorde doorlopen worden. Om uit het objectdiagram van figuur 5.1 het operatieblok te verwijderen, krijgt de QESingleOperatieNode een verwijderopdracht. Dit zijn de opeenvolgende stappen in het verwijderingsproces:

1. De twee QEInputs van de QESingleOperationNode krijgen een verwijderingsopdracht. De ingangen van het operatieblok zijn elk verbonden met een QEConnection, dus krijgen deze beide QEConnection's eerst de opdracht zich van het operatieblok los te koppelen.
2. De QEOutput van de QESingleOperationNode krijgt een verwijderingsopdracht. De QEOutput verwijdert de QEConnection die erbij hoort.
3. De SingleOperationNode krijgt een verwijderingsopdracht. Er wordt voor gezorgd dat de twee connecties (Connection) aan de ingangen losgekoppeld worden. De twee uitgangsconnecties (Connection) worden verwijderd. Aan de operatie horende bij dit operatieblok wordt gemeld dat dit blok de operatie niet langer gebruikt. Aangezien de operatie hier niet gebruikt wordt door andere blokken, ze is immers niet gelinkt, zal de operatie zichzelf verwijderen. Tijdens dit verwijderingsproces verwijdert de operatie ook haar grafische equivalent, de QEOperation.

Merk op dat bovenstaande uitleg niet volledig is, alleen de belangrijkste acties in het verwijderingsproces werden vermeld.

Wijzigingen

Om een operatienetwerk te wijzigen, worden methodes uit de grafische structuur opgeroepen. Deze keuze is logisch, aangezien de meeste wijzigingen plaats vinden door acties van de gebruiker. De reacties op deze acties zijn immers geïmplementeerd in

de grafische structuur. Indien een gebruiker de naam van een blok wijzigt, wordt een methode opgeroepen in de grafische structuur. Deze methode wijzigt de naam in de datastructuur, want de naam van een blok wordt daar bijgehouden. Vervolgens zal deze methode ervoor zorgen dat de nieuwe naam correct op het scherm wordt weergegeven. Dit gebeurt door een bepaald deel van de scène opnieuw te tekenen. Voor andere wijzigingen zoals het tekenen van verbindingslijnen of het veranderen van het aantal ingangen van een blok, wordt op analoge wijze te werk gegaan. Er zijn ook opdrachten die geen gevolgen hebben voor de datastructuur, bijvoorbeeld het verplaatsen van een blok.

Merk op dat de destructie van een blok gezien kan worden als een speciaal geval van een wijziging. Aangezien zowel wijzigingen als verwijderingsopdrachten vertrekken vanuit het grafische gedeelte, is de manier van werken consistent.

5.3 Grafische voorstelling van de elementen in een operatienetwerk

In deze sectie wordt de grafische voorstelling van de elementen in een volledig operatienetwerk voorgesteld. In een eerste onderdeel komt de standaard grafische weergave van alle blokken aan bod. Onder standaard grafische weergave wordt verstaan de weergave zonder speciale markeringen. In een tweede onderdeel wordt een overzicht gegeven van alle wijzen waarop een blok gemarkeerd kan worden. Tot slot wordt in een derde deel de grafische voorstelling van een volledig netwerk voorgesteld.

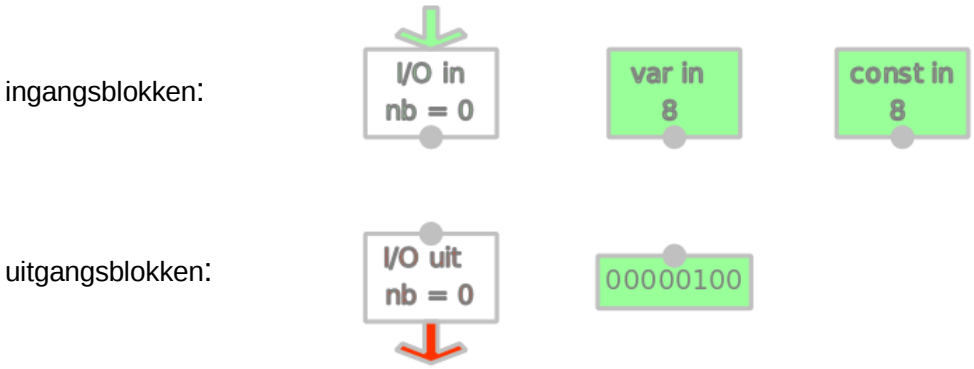
5.3.1 Grafische voorstelling van de blokken

De blokken worden onderverdeeld in drie groepen. Elke groep wordt in een afzonderlijke paragraaf voorgesteld. De eerste groep bevat alle in- en uitgangen. De tweede groep bevat alle operaties. De derde groep bevat enkel het samengestelde blok.

De in- en uitgangen van elk blok worden telkens getekend op de blokken. Ze worden voorgesteld door een blauwe cirkel. De grootte van deze cirkels kan echter verschillen. De grootte ervan hangt af van het type blok. Bij operaties hangt de grootte van elke cirkel ook af van het type operatie. De ingangen van elk blok bevinden zich bovenaan het blok, de uitgangen telkens onderaan het blok.

In- en uitgangen

In figuur 5.3 zijn de vijf verschillende in- en uitgangen weergegeven. De verschillende in- en uitgangen werden besproken in onderdeel 3.3.5. Links in de figuur zijn de I/O-ingang en de I/O-uitgang afgebeeld. Daarnaast zijn er nog de variabele ingang, de constante ingang en het uitgangsblok om een tussenresultaat in het operatienetwerk te bekijken



Figuur 5.3: De grafische voorstelling van de verschillende in- en uitgangsblokken; bovenaan: I/O-, variabele en constante ingang; onderaan: I/O- en kijk-uitgang.

(observatie-uitgang). Merk op dat deze laatste bestaat in twee versies, één met één uitgang en één zonder uitgang. De variabele en constante ingangen in de figuur bevatten telkens acht bits, vandaar het cijfer acht op deze blokken.

Operaties

In figuur 5.4 zijn de verschillende operatieblokken getekend. Sommige operatieblokken bevatten informatie over de parameters van de operatie. Dit is niet het geval bij de logische operaties. Elk operatieblok heeft zijn eigen typische kleur. Op deze manier kan de gebruiker in een oogopslag gelijke operaties herkennen.

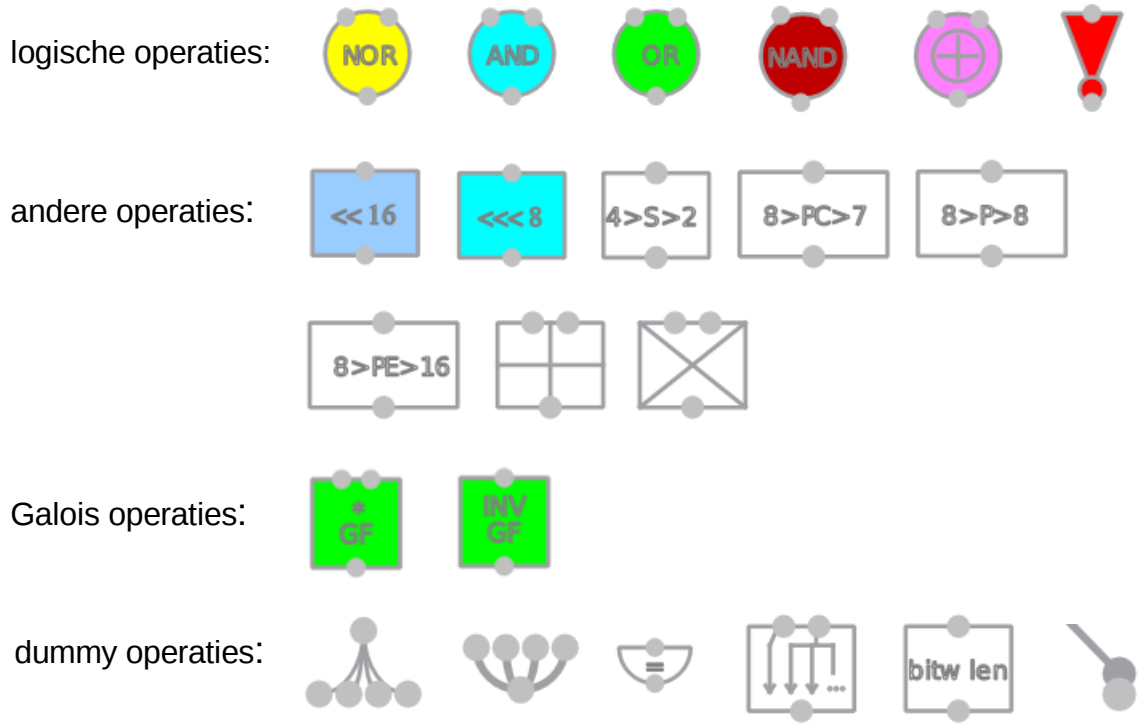
Samengesteld blok

In figuur 5.5 is een samengesteld blok met drie ingangen en twee uitgangen getekend. Indien het aantal in- of uitgangen van het blok vergroot wordt, zullen de overeenkomstige cirkels automatisch verkleinen.

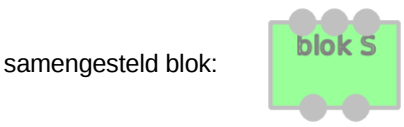
5.3.2 Markeringen op blokken

In voorgaande sectie werd de normale grafische voorstelling van de blokken voorgesteld. De blokken worden in sommige situaties op een iets andere manier getoond. In onderstaande lijst wordt een overzicht gegeven. De verschillende markeringen zijn genummerd. In figuur 5.6 wordt van elke markering een voorbeeld gegeven.

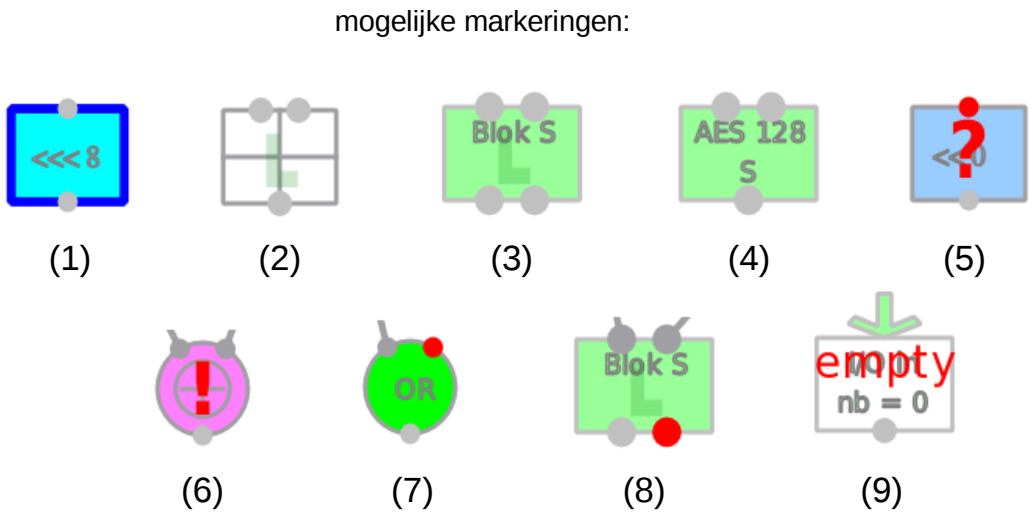
1. Een blok kan geselecteerd worden. Alle lijnen die deel uitmaken van de voorstelling van het blok worden blauw, de lijnen worden iets dikker.



Figuur 5.4: De grafische voorstelling van de verschillende operatieblokken; logische operaties: NOR, AND, OR, NAND, XOR en NOT; andere operaties: verschuiving, rotatie, S-box, 'choice permutation', P-box, expansion permutation', optelling, vermenigvulding; Galois operaties: vermenigvuldiging en inverse-berekening; dummy operaties: splits-op, smelt-samen, kopie, breid-uit, aantal-bits en verbind.

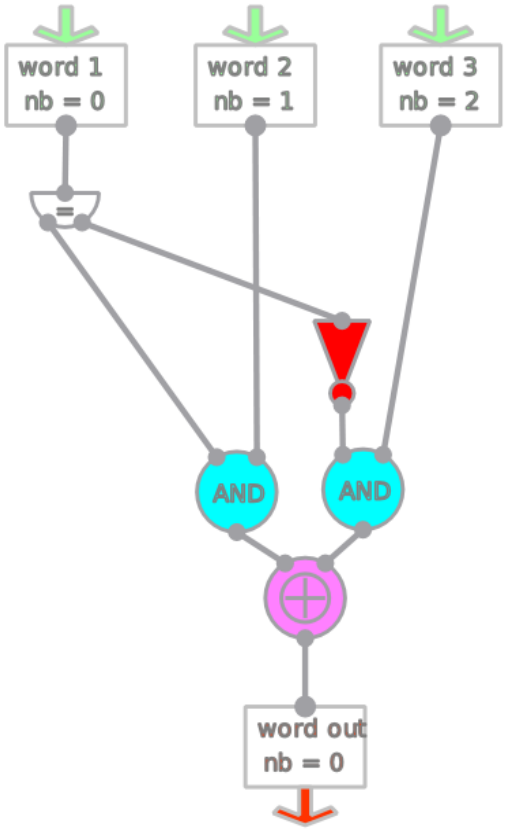


Figuur 5.5: De grafische voorstelling van een samengesteld blok met drie ingangen en twee uitgangen.



Figuur 5.6: De verschillende soorten markeringen op blokken.

2. Twee of meer operatieblokken zijn gelinkt. Er verschijnt een groene transparante letter L in het midden van alle betrokken operatieblokken.
3. Twee of meer samengestelde blokken zijn gelinkt. Er staat een groene transparante letter L in het midden van alle betrokken blokken.
4. Een samengesteld blok is standaard. Er staat een groene letter S onderin het blok.
5. Er werd een berekening uitgevoerd, maar het blok werd niet betrokken in het berekeningsproces. Er verschijnt een rood vraagteken in het midden van het blok.
6. Er wordt een berekening uitgevoerd, maar er is een probleem met het aantal bits dat aangelegd wordt aan één of meerdere ingangen van het blok. Er verschijnt een rood uitroepingsteken op het blok. Het kan bijvoorbeeld gaan om een OR-operatieblok met een verschillend aantal bits aan beide ingangen.
7. Er wordt een berekening uitgevoerd. Het blok wordt betrokken bij de berekeningen, maar kan niets doen, omdat niet al zijn ingangen beschikbaar zijn. De ingangen die overeenkomen met de niet-beschikbare ingangen worden rood gekleurd.
8. Er wordt een berekening uitgevoerd. De ingangen van een samengesteld blok zijn beschikbaar, maar door onregelmatigheden in de ingekapselde scène is een uitgang van het samengestelde blok niet beschikbaar. De uitgang die overeenkomt met deze uitgang wordt rood gekleurd.
9. Er wordt een berekening gestart in een scène met I/O-ingangen. De gebruiker is vergeten om aan deze ingangen een bitwoord toe te kennen. Het woord *empty* verschijnt in het rood op het I/O-ingangsblok.



Figuur 5.7: Grafische weergave van het operatienetwerk van de functie F_{if}, deze functie wordt gebruikt in deze eerste twintig rondes van het SHA-1-hash-algoritme.

5.3.3 Grafische weergave van een operatienetwerk

In voorgaande onderdelen werd beschreven hoe de verschillende blokken weergegeven worden op het scherm. In deze sectie wordt de grafische weergave van een operatienetwerk voorgesteld. Op die manier wordt ook duidelijk hoe verbindingen eruit zien. Het operatienetwerk wordt afgebeeld in figuur 5.7. Het betreft een implementatie van de functie F_{if} horende bij de eerste twintig rondes van het SHA-1-algoritme. Meer informatie over het SHA-1-algoritme en het volledige operatienetwerk ervan is te vinden in sectie 8.2.2.

Elke verbinding in het operatienetwerk van figuur 5.7 is een rechte lijn. Het zou echter handig zijn voor de gebruiker indien er ook samengestelde lijnen zouden zijn. Er zijn twee mogelijkheden om dit te realiseren.

Ten eerste zou men de verbinding tussen een uitgang en een ingang kunnen opsplitsen in horizontale en vertikale verbindingslijnen. In dat geval moeten ook algoritmes geïmplementeerd worden om deze verbinding mooi te tekenen terwijl een blok versleept

wordt. Bovendien moet rekening gehouden worden met de andere blokken, opdat de samengestelde verbinding tijdens het verslepen geen andere blokken zou doorkruisen. Deze optie is dus redelijk complex.

Een tweede mogelijkheid maakt gebruik van nepblokken. In dat geval kan een gebruiker klikken op de uitgang van een blok, de muis slepen naar een bepaalde plaats en daar een welbepaalde toets indrukken. Op die plaats wordt dan een nepblok gecreëerd. Vervolgens kan de gebruiker vanuit dat nepblok een verbinding tekenen naar de ingang van een ander blok.

Deze tweede mogelijkheid werd geïmplementeerd. Indien de gebruiker klikt op een uitgang van een blok, vervolgens de muis versleept naar een vrije plaats in de scène en de muisknop loslaat terwijl hij de control-toets indrukt, verschijnt een nepblok. Dit blok bestaat uit één ingang en één uitgang, deze beide overlappen gedeeltelijk, zie ook figuur 5.4. Dit nepblok werd geïmplementeerd als een nepoperatie. Deze operatie, de verbind-operatie, werd reeds besproken in onderdeel 3.7.3.

5.4 Implementatie

Hoewel in deze thesistekst minder aandacht besteed wordt aan de grafische structuur dan aan de datastructuur, was het implementeren van de grafische structuur zeker niet makkelijker dan het implementeren van de datastructuur. Dit heeft vooral te maken met de nieuwe grafische module QGraphics View in de software toolkit Qt. De specificaties van deze toolkit zijn over het algemeen zeer goed. Toch zijn de specificaties van de grafische module niet altijd even duidelijk. Dat is niet zo verwonderlijk aangezien deze grafische module nog maar pas werd vernieuwd en Qt in de opeenvolgende tussenversies nog steeds verbeterd en aangevuld wordt.

In de volgende twee onderdelen worden twee implementatieproblemen besproken. Het eerste probleem houdt verband met het doorgeven van gebeurtenissen van de scène naar de blokken. Het tweede probleem behandelt het niet terecht komen van een gebeurtenis.

Onduidelijkheid over het doorgegeven van gebeurtenissen

De grafische module heeft heel wat methodes om gebeurtenissen automatisch door te geven aan de juiste objecten. Indien men met de muis klikt in het werkblad, op een plaats waar een blok getekend is, krijgt de momenteel zichtbare scène de melding dat er geklikt werd op die bepaalde positie. De standaard implementatie van een scène zal deze melding automatisch doorgeven aan de grafische entiteit die op die plaats in de scène staat. In dit geval is dit een blok. Qt biedt echter de mogelijkheid om de methodes die zorgen voor dit doorgeven van gebeurtenissen opnieuw te implementeren. Om de correcte werking van het doorgeven van gebeurtenissen doorheen de scène te verzekeren is men

verplicht de oorspronkelijke implementatie van deze methode in de eigen implementatie van de methode op te roepen. Dit is echter niet voldoende duidelijk in de specificatie.

Er zijn verschillende methodes die een reactie op een gebeurtenis van de muis of het toetsenbord implementeren. Indien men sommige van deze methodes van een eigen implementatie voorziet, zonder oproep naar de oorspronkelijke implementatie, dan kunnen de andere originele (private) methodes op één of andere manier in de war geraken en vertonen ze na verloop van tijd vreemde gedragingen. Dit komt waarschijnlijk omdat de oorspronkelijke implementatie van deze methodes private variabelen en private caches gebruiken, die op een bepaald ogenblik hun consistentie met de wel publiek toegankelijke en wijzigbare variabelen verliezen.

Kortom, het zou beter zijn indien in de specificatie van deze methodes vermeld wordt dat de oorspronkelijke implementatie van deze methodes verplicht moet worden opgeroepen. Dit was niet onmiddellijk duidelijk en daardoor werd het implementeren van deze methodes bemoeilijkt. De specificatie van de betrokken klasse, QEGraphicsScene, vindt de lezer in [23].

Bepaalde gebeurtenissen komen niet terecht

Bepaalde grafische identiteiten in een scène kunnen op non-actief gezet worden. In dat geval wordt deze identiteit wel getekend, maar kan ze onmogelijk gebeurtenissen ontvangen. Indien men een blok toevoegt aan de scène, komt dit blok terecht in een kleine rechthoek die zich links bovenaan bevindt. Indien men vervolgens met de muis dit blok wou verslepen, bleek dit soms niet te lukken. Na onderzoek, bleek de scène de melding dat er geklikt werd wel degelijk te ontvangen, maar niet altijd door te geven aan het blok. Dit probleem stelde zich niet altijd, maar eerder nu en dan. Vermoedelijk wordt de gebeurtenis soms doorgegeven aan het inactieve blok en gaat ze dan verloren juist omdat het blok op inactief staat.

Er bestaat echter een eenvoudige oplossing. In een scène is het mogelijk om aan elke grafische identiteit een waarde toe te kennen, die bepaalt welke identiteit boven welke andere identiteit getekend moet worden. Indien men de genoemde rechthoek achter alle andere blokken en connecties tekent, stelt het genoemde probleem zich niet.

5.5 Besluit

In dit hoofdstuk werd de netwerkstructuur uit hoofdstuk 3 aangevuld met een parallelle grafische structuur. De volledige netwerkstructuur bestaat dus uit de datastructuur, dit is de structuur die in hoofdstuk 3 werd toegelicht, en de grafische structuur, dit is de structuur die in dit hoofdstuk werd toegelicht. De data-informatie werd gescheiden van de grafische informatie om de code overzichtelijk te houden. Er werd argumenteerd waarom beide structuren altijd samen aanwezig zijn. Daarna werd uitgelegd hoe een

operatienetwerk in het geheugen opgebouwd, gewijzigd en verwijderd wordt. Tevens werd de grafische weergave van de verschillende blokken voorgesteld. Er was aandacht voor de mogelijke markeringen op de verschillende blokken. Tot slot werden enkele implementatieproblemen besproken.

De implementatie van de grafische structuur bevat ook heel wat methodes die de gebruiker toelaten om interactief met een operatienetwerk om te gaan. Ze kwamen slechts even aan bod in de sectie over de implementatieproblemen, namelijk sectie 5.4. De verschillende mogelijke interacties van de gebruiker met de scène, worden in onderdeel 7.4.1 uitgebreid besproken.

Hoofdstuk 6

Opslaan en inladen van een operatienetwerk

6.1 Inleiding

In dit hoofdstuk wordt eerst bepaald in welk formaat de informatie over een operatienetwerk wordt opgeslagen, dit gebeurt in sectie 6.2. Er wordt gekozen voor een XML-formaat. In sectie 6.3 worden de twee belangrijkste standaarden om XML-bestanden op te bouwen, te wijzigingen en in te laden besproken. In sectie 6.4 wordt voorgesteld op welke manier de informatie in een XML-document wordt opgeslagen. In sectie 6.5 komt het terug inladen aan bod. Dit gebeurt telkens eerst voor de substitutie-box en de bit-herverdeler, daarna pas voor een volledig operatienetwerk. Telkens wordt gespecifieerd welke interface gebruikt wordt. In de laatste sectie wordt een conclusie geformuleerd.

6.2 Keuze van het formaat

Bij de keuze van het formaat waarin een operatienetwerk wordt opgeslagen, wordt rekening gehouden met de vereisten uit sectie 2.3. Het formaat moet dus platvormonafhankelijk zijn.

Er wordt gekozen voor het XML-formaat. De afkorting XML staat voor Extensible Markup Language en is platvormonafhankelijk. Bovendien wordt deze standaard breed ondersteund. Ook Qt, de gebruikte software toolkit, heeft een XML-module. Deze kan gebruikt worden om de inhoud van XML-documenten in te laden of om informatie erin weg te schrijven. Meer informatie over XML is te vinden in [26].

Een XML-document heeft ook als voordeel dat de informatie in het document door iedereen bekeken en geïnterpreteerd kan worden. Dit geldt uiteraard alleen indien de informatie in het XML-document op een duidelijke en gestructureerde manier is

```
<scene-information>
  <node>
     <name>naamvanblokje</name>
     <nrinputconnections>2</nrinputconnections>
  </node>
</scene-information>
```

Figuur 6.1: Voorbeeld van een XML-fragment.

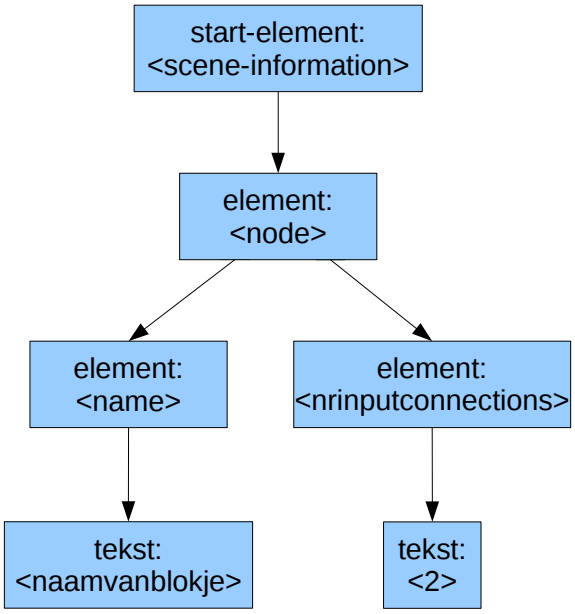
opgebouwd.

6.3 Beschikbare API's

In deze sectie worden een aantal API's besproken. API staat voor Application Programming Interface. Er is een API nodig om XML-documenten in te laden, te wijzigen en aan te maken. Men kan twee manieren van werken onderscheiden, elke manier komt overeen met een bepaald type API. Beide worden in de volgende paragrafen kort geïllustreerd aan de hand het XML-fragment in figuur 6.1. In het programma worden beide API-types gebruikt.

Objectgeoriënteerde API's

Het eerste type API is gebaseerd op een boomnetwerk. Elk XML-document herbergt een hiërarchische structuur en kan dus voorgesteld worden als een boomnetwerk. In figuur 6.2 wordt het boomnetwerk dat hoort bij het XML-fragment uit figuur 6.1 afgebeeld. Om een XML-document op te bouwen, wordt in het geheugen een boomnetwerk, of kortweg boom, gecreëerd. Vervolgens wordt de boom weggeschreven naar een XML-bestand. Om een XML-document te lezen, wordt de volledige boom telkens in het geheugen opgebouwd op basis van de informatie in het XML-document. Voor grote bomen vergt dit veel geheugen, dit is meteen het grootste nadeel van deze manier van werken. Het gebruik van een boom is wel heel flexibel, het vergt weinig programmeerwerk om bepaalde gegevens in een XML-document te wijzigen. Elk element van de boom kan snel opgevraagd worden. Een objectgeoriënteerde API is vooral interessant indien er belangrijke verschillen zijn tussen de structuur van het XML-document en de structuur van de overeenkomstige informatie in het programma. De meest gebruikte API van dit type is DOM, wat staat voor Document Object Model. Qt is voorzien van klasses die DOM implementeren. Meer informatie over DOM is te vinden in [27].



Figuur 6.2: DOM-voorstelling van het voorbeeld-XML-fragment in figuur 6.1.

Gebeurtenis gedreven API's

Een tweede type API werkt met gebeurtenissen. Bij het lezen van het XML-document, wordt een opeenvolging van gebeurtenissen gegenereerd. De opeenvolgende gebeurtenissen die horen bij het XML-fragment in figuur 6.1 zijn de volgende:

1. Startelement gedetecteerd: scene-information
2. Startelement gedetecteerd: node
3. Startelement gedetecteerd: name
4. Karakters gedetecteerd: "naamvanblokje"
5. Eindelement gedetecteerd: name
6. Startelement gedetecteerd: nrinputconnections
7. Karakters gedetecteerd: "2"
8. Eindelement gedetecteerd: nrinputconnections
9. Eindelement gedetecteerd: node
10. Eindelement gedetecteerd: scene-information

Het lezen van een XML-document gebeurt dus sequentieel. Het opbouwen van een XML-document verloopt op een gelijkaardige manier. Dit type API is niet geschikt om enkele specifieke wijzigingen aan te brengen in een XML-document. Dit type API is vooral geschikt om XML-documenten weg te schrijven, waar de data in de juiste volgorde beschikbaar wordt. Een gebeurtenis gedreven API is dus vooral interessant indien er weinig verschil is tussen de structuur van het XML-document en de structuur van de

overeenkomstige informatie in het programma. Het grootste voordeel van deze manier van werken is het lage geheugengebruik. De meest gebruikte standaard van dit type API is SAX, wat staat voor Simple API for XML. Meer informatie over SAX kan men vinden in [28].

6.4 Opslaan in een XML-document

Deze sectie is verdeeld in twee onderdelen. In een eerste onderdeel wordt het opslaan van substitutie-boxen en bit-herverdelers toegelicht. In een tweede onderdeel wordt het opslaan van een volledig operatienetwerk toegelicht.

Opslaan van een substitutie-box en een bit-herverdeler

De informatie die hoort bij een substitutie-box of een bit-herverdeler wordt niet in het XML-document van het operatienetwerk opgeslagen. De gegevens over deze operaties worden weggeschreven in afzonderlijke XML-documenten. Elke substitutie-box en elke bit-herverdeler wordt opgeslagen in een afzonderlijk XML-document. In onderdeel 3.7.4 werd uitgelegd waarom dit gewenst is.

Indien een S-box in een scène getekend wordt, is de informatie over deze S-box alleen aanwezig in het geheugen. Het creëren van een S-box in het geheugen gaat dus niet gepaard met het creëren van een XML-document. Een S-box kan wel op om het even welk moment opgeslagen worden in een XML-bestand. Indien een operatienetwerk waarin zich een S-box bevindt, wordt opgeslagen, wordt het XML-bestand dat de S-box definieert automatisch aangemaakt. Waar dit bestand precies wordt geslagen, wordt toegelicht in het volgende onderdeel. Dit alles geldt ook voor de bit-herverdeler.

Het opslaan van een substitutie-box of een bit-herverdeler kan gebeuren met SAX. De gegevens over deze operaties worden dus sequentieel weggeschreven.

De datagegevens die horen bij de S-box worden niet in een lineaire lijst opgeslagen. Het opslaan gebeurt overeenkomstig de structuur waarmee de S-box werd ingevoerd in het programma. De datagegevens van de S-box worden in het programma immers voorgesteld door middel van een tabel en niet door middel van een lineaire lijst. Een tabelvoorstelling heeft als voordeel dat alle elementen tegelijkertijd zichtbaar zijn op het scherm. Ter verduidelijking wordt in bijlage C uitgelegd hoe een tabelvoorstelling van een substitutie-box geïnterpreteerd dient te worden. Het in een tabel voorstellen van een S-box in het programma impliceert uiteraard niet dat de S-box in een tabel in het XML-document moet worden opgeslagen. Dit is echter wel het geval, want zo is de inhoud van het XML-document leesbaarder voor de gebruiker.

Het XML-document van een bit-herverdeler is op dezelfde manier opgebouwd als dat van een S-box. Een bit-herverdeler wordt volledig bepaald door het aantal ingangsbits,

het aantal uitgangsbits en een rij van bitposities. Deze rij bevat evenveel elementen als er uitgangsbits zijn. Indien het i-de rij-element gelijk is aan j, dan verschijnt de j-bit van de ingang op de i-de bitpositie van de uitgang.

In het XML-document van zowel een substitutie-box als een bit-herverdeler, wordt informatie over het al dan niet standaard zijn van de operatie opgeslagen. In het programma zijn een aantal standaard substitutie-boxen en een aantal standaard bit-herverdelers aanwezig. Voorbeelden zijn de S-boxen en bit-herverdelers uit DES. De XML-documenten die horen bij deze standaard operaties zijn te vinden in de map `standardOperations`. Deze map bevindt zich in de map `standardPackages`. Deze laatste map is een onderdeel van het programma.

Het is voor de gebruiker onmogelijk een operatie op te slaan als een standaard operatie. Indien dit wel mogelijk zou zijn, kan een gebruiker operatienetwerken tekenen met zijn zogezegde standaard operaties. Indien hij een dergelijk operatienetwerk opslaat en vervolgens probeert te openen, zal het programma echter melden dat het inlezen mislukt is. Het programma gaat er immers vanuit dat alle standaard operaties zich in de map `standardOperations` bevinden.

Standaard operaties, namelijk standaard S-boxen en standaard bit-herverdelers, kunnen enkel correct worden toegevoegd door ze op te slaan in de map `standardOperations`. Vervolgens moet de gebruiker het XML-document manueel aanpassen door aan het attribuut ‘standard’ de waarde ‘true’ toe te kennen.

In sectie 6.5.1 en 6.5.3 wordt respectievelijk uitgelegd hoe S-boxen en bit-herverdelers worden ingeladen en op welke manier standaard operaties heel eenvoudig gebruikt kunnen worden.

6.4.1 Opslaan van een operatienetwerk

Bij elke scène hoort precies één XML-document. Een operatienetwerk wordt niet noodzakelijk in één XML-bestand opgeslagen. Indien het operatienetwerk dat wordt opgeslagen een samengesteld blok bevat, met een ingekapselde scène die niet standaard is, wordt deze ingekapselde scène in een afzonderlijk XML-document opgeslagen. Tijdens het opslaan van een scène kunnen dus automatisch nog andere XML-bestanden gecreëerd worden. Waar precies deze ingekapselde scène dan wordt opgeslagen en onder welke naam, wordt het best uitgelegd aan de hand van een voorbeeld.

Een scène `sceneA` bevat verschillende blokken, waaronder één samengesteld blok met de naam `testBlok`. Scène `sceneA` wordt opgeslagen in de map `home` onder de bestandsnaam `sceneA.xml`. Omdat de scène een niet-standaard ingekapselde scène bevat, wordt in de map `home` een submap `sceneA_ComposedNodes` aangemaakt. In deze map wordt de ingekapselde scène opgeslagen onder de naam `testBlok.xml`. Deze naam is gebaseerd op de naam `testBlok` van het overeenkomstige samengestelde blok. Deze werkwijze is recursief. Indien de ingekapselde scène op haar beurt een samengesteld blok met een

andere ingekapselde scène bevat, wordt opnieuw een submap gecreëerd.

Een scène kan substitutie-boxen en/of bit-herverdelers bevatten. Bij het opslaan van een scène worden al deze operaties in een afzonderlijk XML-document opgeslagen. Dit gebeurt niet indien de gebruikte substitutie-box of bit-herverdeler standaard in het programma aanwezig is. De XML-documenten met informatie over de substitutie-boxen en bit-herverdelers worden opgeslagen in een submap die aangemaakt wordt tijdens het opslaan van de scène. Indien de scène opgeslagen wordt onder de naam `sceneA.xml`, wordt de naam van de genoemde submap `sceneA_OperationInformation`. Merk op dat deze werkwijze analoog is aan de werkwijze bij het opslaan van ingekapselde scènes.

Indien een scène wordt opgeslagen in een XML-bestand, kan dit gepaard gaan met het creëren van maximaal twee nieuwe mappen. Een map voor XML-bestanden met informatie over één of meerdere operaties en een tweede map voor XML-bestanden horende bij de aanwezige ingekapselde scènes.

Het XML-document dat hoort bij een scène bestaat uit vijf delen:

1. Informatie over de scène. Er wordt onder andere bijgehouden of de scène al dan niet standaard of vergrendeld is. Ook de naam van de scène wordt bijgehouden.
2. Een lijst van alle blokken. Elk blok heeft een uniek nummer, namelijk het dynamisch bloknummer. Ook het type, de naam, de positie en het aantal in- en uitgangen van elk blok worden hier bijgehouden. Afhankelijk van het type blok worden nog andere gegevens bijgehouden. Bij een ingangsblok met constante ingang wordt het bitwoord opgeslagen. Indien het een operatieblok betreft, wordt een operatienummer bijgehouden. Dit nummer verwijst naar een operatie. De eigenschappen van de verschillende operaties worden in het vierde deel opgeslagen. Gelinkte operaties hebben dezelfde operatie en dus ook hetzelfde operatienummer.
3. De connecties. In dit deel worden alle connecties in een sequentiële lijst opgeslagen. Gebruikmakend van de unieke nummers van elk blok, worden de connecties gedefinieerd. Het kan bijvoorbeeld gaan om een verbinding van de tweede uitgang van blok 52 die verbonden is met de eerste ingang van blok 39.
4. De eigenschappen van alle operaties. Voor een rotatie-operatie bijvoorbeeld, wordt de rotatie-richting en het aantal te roteren bits bewaard. Voor een substitutie-box en een bit-herverdeler wordt de bestandsnaam van het afzonderlijke XML-document met de informatie over de operatie bewaard. Indien de operatie niet standaard is, wordt ook de mapnaam waarin dit XML-document is opgeslagen bewaard.
5. In een laatste deel wordt informatie over de ingekapselde scènes van de samengestelde blokken opgeslagen. Ingekapselde scènes worden in een apart bestand opgeslagen. Elke scène krijgt immers haar eigen XML-document. Indien de betrokken ingekapselde scène standaard is, wordt alleen de bestandsnaam van deze scène bijgehouden. Indien het een zelfgemaakte ingekapselde scène betreft, wordt zowel de bestandsnaam van het XML-document van de ingekapselde scène, als de map waarin dit bestand zich bevindt, bijgehouden.

Het opslaan van een operatienetwerk is dus een complexe zaak. De informatie over een operatieblok is verspreid over het XML-document. Daarom is het gebruik van DOM te verkiezen boven SAX. Het XML-document dat hoort bij een scène wordt dus eerst in het geheugen in een boom opgebouwd. Op basis van deze boom wordt vervolgens een XML-document aangemaakt.

In bijlage E.2 is het XML-document van een scène te vinden. Deze scène is standaard in het programma aanwezig. Het operatienetwerk in deze scène wordt gebruikt in het AES-algoritme. Het betreft het operatienetwerk van één encryptie-ronde. Een grafische voorstelling van een ronde in het AES-algoritme zoals weergegeven in het programma, alsook meer uitleg over de stappen in het AES-algoritme, is te vinden in onderdeel 8.2.1.

6.5 Inladen van een XML-document

In deze sectie wordt uitgelegd op welke manier XML-documenten worden ingeladen. In de eerste twee onderdelen wordt besproken op welke manier substitutie-boxen en bit-herverdelers, respectievelijk scènes, worden ingeladen vanuit een XML-document. In een derde onderdeel wordt kort besproken op welke manier standaard scènes en operaties bij het opstarten van het programma reeds gedeeltelijk worden ingeladen.

6.5.1 Inladen van een operatie met een XML-document

Het inladen van de gegevens van een substitutie-box of een bit-herverdeler is redelijk eenvoudig en kan dus sequentieel gebeuren. De SAX-API wordt gebruikt. Hoe de gebruiker operaties kan inladen wordt besproken in onderdeel 7.4.2.

6.5.2 Inladen van een operatienetwerk

Het inladen van een operatienetwerk start met een inlaadopdracht. Het XML-document wordt ingeladen in een boom in het geheugen. De DOM-API wordt dus gebruikt. Het volledige inlaadproces van een operatienetwerk kan opgesplitst worden in een aantal stappen:

1. De boom van het XML-document van de scène wordt ingeladen, zodat alle informatie in het XML-document beschikbaar is.
2. De scène wordt geconstrueerd. De eigenschappen van de scène worden bepaald aan de hand van de scène-informatie in het XML-document.
3. De informatie over alle operaties wordt ingeladen.
4. De informatie over alle ingekapselde scènes wordt één voor één ingeladen. Concreet wil dit zeggen dat voor elke ingekapselde scène, de zeven stappen in dit inlaadproces recursief gebruikt worden.

5. Alle blokken worden geconstrueerd. Operatieblokken worden voorzien van de juiste operatieparameters, deze informatie werd reeds in stap drie ingeladen. Samengestelde blokken worden voorzien van de juiste ingekapselde scènes, deze werden reeds ingeladen in stap vier.

6. De in- en uitgangen van de verschillende blokken worden verbonden volgens de in het XML-document aanwezige informatie over de connecties.

7. De boom van het XML-document horende bij de scène wordt uit het geheugen verwijderd.

In stap drie werden alle operaties ingeladen. Indien de scène substitutie-boxen en/of bit-herverdelers bevat, worden de XML-documenten van deze operaties in stap drie ingeladen. Zoals in sectie 6.5.1 werd vermeld, gebeurt dit met de SAX-API.

6.5.3 Inladen van standaard operatienetwerken en operaties

In de vorige twee onderdelen werd besproken hoe operatienetwerken en operaties met een eigen XML-document worden ingeladen. Na het inladen zijn de scènes, respectievelijk de operaties, klaar voor gebruik. Ze werden immers volledig ingeladen. Het is echter ook mogelijk om scènes of operaties slechts gedeeltelijk in te lezen. In dat geval wordt bij scènes onder andere de naam en de beschrijving van de scène ingelezen. Bij operaties wordt onder andere de naam van de operatie ingelezen. Al deze gegevens bevinden zich telkens bovenaan in het XML-document. Met behulp van de SAX-API kunnen deze gegevens dus snel ingeladen worden.

Deze verkorte manier van inladen wordt gebruikt om bij het opstarten van het programma drie lijsten aan te maken. De eerste lijst bevat alle standaard aanwezige scènes, de tweede bevat alle standaard substitutie-boxen en de derde lijst bevat alle standaard bit-herverdelers. Om een standaard scène te gebruiken, kan de gebruiker de gewenste standaard scène selecteren uit de lijst waarin alle standaard scènes met naam en beschrijving zijn opgenomen. Dit is ook zo voor de twee andere lijsten, namelijk de lijst met substitutie-boxen en de lijst met bit-herverdelers.

De XML-documenten van alle standaard scènes en standaard operaties bevinden zich elk in een vaste map. Bij het opstarten van het programma worden deze twee mappen gescand. Elk XML-document wordt telkens gedeeltelijk ingeladen, de ingeladen gegevens komen terecht in de respectievelijke lijsten.

In figuur 7.4 kan de lezer zien hoe de lijst met standaard S-boxen in het programma tot uiting komt. In sectie 8.2 worden de standaard aanwezige algoritmes en de standaard XML-documenten van deze algoritmes besproken. In sectie 8.3 komen de standaard aanwezige substitutie-boxen en bit-herverdelers kort aan bod.

6.6 Besluit

In dit hoofdstuk werd XML als bestandsformaat gekozen om operatienetwerken op te slaan. De twee groepen API's om een XML-document aan te maken of in te lezen werden kort besproken. Vervolgens werd uitgebreid toegelicht hoe scènes, substitutie-boxen en bit-herverdelers worden opgeslagen en ingeladen. Telkens werd uitgelegd welke API gebruikt werd en waarom. Tot slot werd uitgelegd dat XML-documenten niet noodzakelijk volledig hoeven te worden ingeladen. Het is immers mogelijk om snel enkele belangrijke eigenschappen van een scène of operatie in te lezen.

Hoofdstuk 7

Grafische gebruikersinterface

7.1 Inleiding

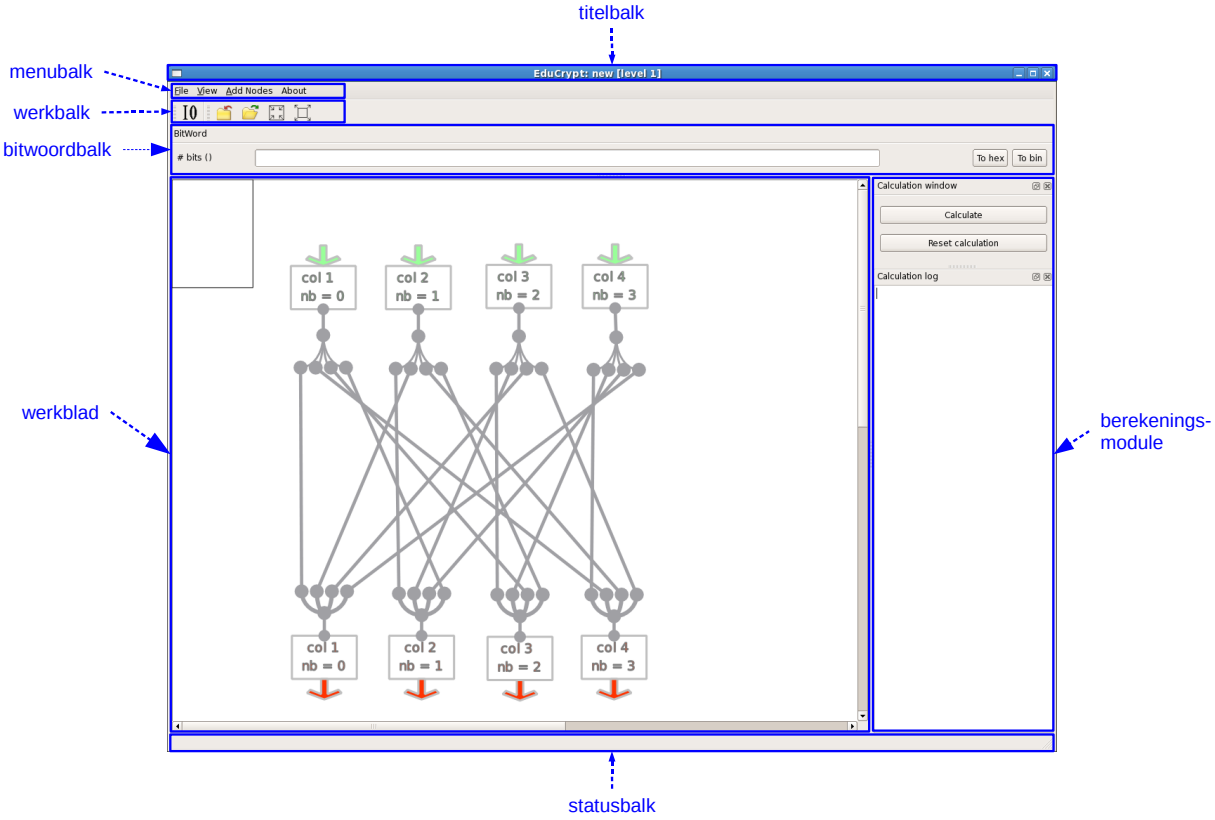
In dit hoofdstuk worden de verschillende onderdelen van de grafische gebruikersinterface besproken. In sectie 7.2 wordt de gebruikersinterface voorgesteld, de verschillende onderdelen worden voorgesteld. De meest belangrijke onderdelen worden elk in een afzonderlijke sectie besproken. In sectie 7.3 komt de bitwoordbalk aan bod. In sectie 7.4 worden het werkblad en de functionaliteit in dit werkblad uitgebreid besproken. In sectie 7.5 wordt de berekeningsmodule besproken. In sectie 7.6 wordt de handleiding van dit programma voorgesteld. In de laatste sectie wordt een besluit geformuleerd.

7.2 Verschillende onderdelen

In deze sectie komen de verschillende onderdelen van de grafische gebruikersinterface aan bod. De verschillende onderdelen zijn de volgende:

- De titelbalk
- De menubalk
- De werkbalk
- De statusbalk
- De bitwoordbalk
- Het werkblad
- De berekeningsmodule

In figuur 7.1 wordt de grafische gebruikersinterface van het programma weergegeven. De verschillende onderdelen zijn aangeduid in deze figuur.



Figuur 7.1: Overzicht van de grafische gebruikersinterface van het programma.

In de volgende onderdelen van deze sectie worden de eerste vier onderdelen van de grafische gebruikersinterface kort besproken. Aan de belangrijkste onderdelen, namelijk de bitwoordbalk, het werkblad en de berekeningsmodule, wordt telkens een afzonderlijke sectie gewijd. Ze worden respectievelijk in sectie 7.3, 7.4 en 7.5 besproken.

7.2.1 Titelbalk

Helemaal bovenaan in het programma bevindt zich de titelbalk. Deze titelbalk bevat informatie over het programma en de momenteel geopende scène. De tekst in de titelbalk bevat respectievelijk:

- De naam van het programma, namelijk EduCrypt. EduCrypt staat voor educatieve cryptografie.
- Daarna volgt het pad van de geopende scène. Indien deze scène nog niet werd opgeslagen, wordt het woord 'new' (nieuw) weergegeven.

- Daarna wordt vermeld op welk niveau de in het werkblad zichtbare scène zich bevindt. Indien geen ingekapselde scènes worden gebruikt, is dit niveau sowieso één. Indien men een ingekapselde scène bekijkt, bevindt men zich op het tweede niveau. Het aantal niveaus is onbeperkt, maar in de praktijk beperkt men zich meestal tot enkele. Hoe men zich kan begeven naar een bepaald niveau wordt uitgelegd in onderdeel 7.4.3.
- Informatie over het vergrendeld zijn van de momenteel zichtbare scène. De titeltekst bevat het woord 'locked' indien de scène is vergrendeld. Een scène die vergrendeld is, kan niet worden gewijzigd. Standaard scènes zijn altijd vergrendeld.
- Informatie over het al dan niet gelinkt zijn van de zichtbare scène. Indien enkele samengestelde blokken gelinkt zijn, bevat de titeltekst het woord 'linked' telkens wanneer één van de ingekapselde scènes van deze samengestelde blokken in het werkblad wordt weergegeven. Merk op dat een scène op niveau één nooit gelinkt kan zijn.

Indien men bijvoorbeeld het XML-document van het AES-encryptie-algoritme opent, dan is een mogelijke inhoud van de titelbalk de volgende:

```
Educrypt: /home/jigor/thesis/bin/standardPackages/standardComposedScenes/
       AES_128_encrypt.xml [level 1: locked]
```

7.2.2 Menubalk

De menubalk bevat vier menu's:

1. File
2. View
3. Add Nodes
4. About

De verschillende items in deze vier menu's worden nu kort besproken.

Het menu File :

New document: Sluit het huidige document en open een nieuw document.
Open: Open een bestaand document uit een XML-bestand.
Save all levels: Sla alle wijzigingen op in het bestand dat momenteel geopend is.
Export all levels as: Sla het momenteel geopende document op in een nieuw XML-bestand.
Export current level: Sla een deel van het document op in een nieuw XML-bestand. Alleen de momenteel zichtbare scène wordt opgeslagen, uiteraard samen met alle hogere niveaus.

Export scene to image: Sla de volledige inhoud van de momenteel zichtbare scène op in een beeldbestand (PNG-formaat).

Export scene-screenshot to image: Sla de momenteel zichtbare inhoud van de momenteel zichtbare scène op in een beeldbestand (PNG-formaat).

Quit: Verlaat het programma.

Het menu View :

Calculation buttons: Het deel van de berekeningsmodule met de berekeningsknop en de resetknop kan verborgen worden. Nogmaals klikken op dit item zorgt ervoor dat de berekenings- en resetknop terug verschijnen rechts in het scherm.

Calculation log: Het deel van de berekeningsmodule met de log van het berekeningsproces kan verborgen worden. Nogmaals klikken op dit item zorgt ervoor dat de log terug verschijnt rechts in het scherm.

Limit calculation log output Zorgt ervoor dat de informatie over een berekeningsproces die gelogd wordt, beperkt wordt het strikt noodzakelijke. Deze functionaliteit kwam reeds aan bod in sectie 4.3.3.

Het menu Add Nodes :

Dit menu bevat items om blokken toe te voegen aan de scène. De blokken zijn onderverdeeld in zeven categorieën. De eerste vier categorieën komen overeen met de in sectie 3.7 gedefinieerde groepen van operaties. De laatste twee categorieën zijn de normale ingangen en de normale uitgangen. Merk dat I/O-ingangen en I/O-uitgangen niet toegevoegd kunnen worden met behulp van dit menu. Dit moet op een andere manier gebeuren, deze manier wordt besproken in onderdeel 7.4.3.

Logical operations: OR, AND, NOR, NAND, XOR en NOT

Other operations: shift, rotation, substitution box, bit shuffle, addition, multiplication

Galois operations: Galois multiplication, Galois inverse

Dummy operations: split, melt, copy, extend, bitword length

Inputs: constant input, variable input

Outputs: watch output

Composed: composed node

Het menu About :

Dit menu bevat slechts één item, namelijk ‘About’. Bij het aanklikken van dit item verschijnt er een venster met wat uitleg over het programma.

7.2.3 Werkbalk

De werkbalk bevat vijf items. Hun functie wordt hieronder kort toegelicht.

Werkbalkknoppen :

I/O settings: Er opent zich een dialoogvenster, waar de I/O-ingangen en I/O-uitgangen van de momenteel zichtbare scène weergegeven worden. Het is ook hier dat deze in- en uitgangen toegevoegd, aangepast en verwijderd kunnen worden. Via dit dialoogvenster kan men tevens andere scènes inladen in het huidig zichtbare niveau. Meer uitleg over dit dialoogvenster is te vinden in sectie 7.4.3.

To composed scene: Indien momenteel een samengesteld blok geselecteerd is, wordt de scène in het werkblad veranderd naar de ingekapselde scène van dit samengesteld blok. Het niveau van de in het werkblad zichtbare scène is met één verhoogd.

To upper scene: Indien de scène die momenteel weergegeven wordt een ingekapselde scène is, die hoort bij een samengesteld blok op een lager niveau, wordt de scène in het werkblad veranderd naar de scène waarin dit samengesteld blok zich bevindt. Het niveau van de in het werkblad zichtbare scène is met één verlaagd.

Make current scene smaller: Verklein de grootte van de scène in het werkblad. De grootte van het werkbladvenster blijft gelijk. Deze actie heeft meestal als gevolg dat de schuifbalken herschaald worden. De scène wordt alleen verkleind indien hierbij geen objecten aan de rand van de scène gedeeltelijk of volledig onzichtbaar worden. Deze functie heeft niets te maken met zoomen in een werkblad.

Enlarge current scene: Vergroot de grootte van de scène in het werkblad. Deze actie heeft meestal als gevolg dat de schuifbalken herschaald worden. De grootte van het werkbladvenster blijft gelijk. Er is een maximumgrootte van 2500 x 2500 pixels ingesteld, om te vermijden dat de scène excessief vergroot wordt. Deze functie heeft niets te maken met zoomen in een werkblad.

7.2.4 Statusbalk

De statusbalk bevat begeleidende informatie. De inhoud van de statusbalk is vooral interessant nadat een blok is aangeklikt, want daarna verschijnen de eigenschappen van het blok in tekstvorm op de statusbalk.

Figuur 7.2: Grafische weergave van de bitwoordbalk.

7.3 Bitwoordbalk

7.3.1 Invoeren en bekijken van bitwoorden

De bitwoordbalk wordt gebruikt om bitwoorden weer te geven, in te geven of aan te passen. In figuur 7.2 wordt de bitwoordbalk weergegeven. De bitwoordbalk werkt zowel binair als hexadecimaal. Rechts in de bitwoordbalk kan de gebruiker de voorstellingswijze veranderen.

Om bitwoorden weer te geven klikt de gebruiker met de rechter muisknop op een in- of uitgangsblok of een verbindingslijn (connectie). Vervolgens verschijnt het overeenkomstige bitwoord in de bitwoordbalk. Het aantal bits van dit bitwoord wordt tussen haakjes vermeld, links in de bitwoordbalk. Informatie over het bitwoord bevindt zich rechts in de bitwoordbalk. Indien het in de bitwoordbalk zichtbare bitwoord afkomstig is van een ingangsblok, kan de gebruiker het bitwoord aanpassen. De invoer kan gebeuren zowel in binaire, als hexadecimale notatie, beide kunnen door elkaar gebruikt worden. Het aantal bits dat momenteel aanwezig is in de bitwoordbalk staat links in de bitwoordbalk. Dit aantal wordt tijdens de invoer voortdurend aangepast. Het wijzigen van een bitwoord horende bij een ingangsblok wordt bevestigd door op de enter-toets te drukken. Bitwoorden horende bij connecties kunnen bekeken, maar niet gewijzigd worden.

Na een berekening kan men het aantal bits horende bij een verbindingslijn bestuderen door de muispositie een tijdje vast te houden boven een verbindingslijn.

7.3.2 Automatische aanpassing van bitwoorden

Indien het in de bitwoordbalk getoonde bitwoord wijzigt, wordt de bitwoordbalk automatisch aangepast. Dit kan bijvoorbeeld gebeuren wanneer een nieuwe berekening wordt uitgevoerd. Indien de entiteit, bijvoorbeeld een ingangsblok, die hoort bij het momenteel zichtbare bitwoord wordt verwijderd, verschijnt het woord ‘empty’ in de bitwoordbalk.

Dit automatisch aanpassen van bitwoorden gebeurt op basis van een uitbreidbaar principe. Indien een klasse een bepaalde interface implementeert, kan deze vragen een bepaald bitwoord in de gaten te houden en te worden verwittigd indien het bitwoord wordt gewijzigd of verwijderd. Dit kan heel handig zijn voor eventuele uitbreidingen.

7.4 Werkblad

7.4.1 Functionaliteit in een werkblad

In dit onderdeel wordt een overzicht gegeven van alle interactieve mogelijkheden in het werkblad. Het veranderen van de instellingen van blokken en het gebruik van ingekapselde scènes wordt in de volgende twee onderdelen besproken.

De verschillende mogelijkheden om het werkblad interactief te gebruiken zijn opgesplitst in zeven types van handelingen. Deze types van handelingen worden nu overlopen.

Blokken aanmaken en verwijderen

Blokken kunnen enkel worden aangemaakt via het menu 'Add nodes' in de menubalk. Blokken verschijnen telkens links bovenaan in het werkblad, in de daartoe voorziene rechthoek. Het verwijderen van een blok kan door het blok te selecteren en vervolgens de delete-toets in te drukken. De gebruiker kan verschillende blokken tegelijkertijd selecteren. Wanneer hij vervolgens de delete-toets indrukt, worden alle geselecteerde blokken verwijderd.

Verplaatsen en kopiëren van blokken

Blokken kunnen verplaatst worden met de muis. De gebruiker klikt op het blok en versleept het blok naar de gewenste locatie. Twee blokken kunnen nooit overlappen. Indien de gebruiker het blok sleept over een andere blok en vervolgens de muisknop los laat, gebeurt er niets. Het kopiëren van een blok kan op dezelfde wijze gebeuren, maar bij het loslaten van de muisknop dient men de control-toets ingedrukt te houden.

Het verplaatsen van blokken kan tevens met het toetsenbord. Door op een blok te klikken kan men het selecteren. Vervolgens kunnen de vier pijltoetsen gebruikt worden om het blok te verplaatsen. Indien men tijdens het verplaatsen ook de control-toets indrukt, worden het blok telkens slechts één pixel verplaatst.

Sommige nepoperaties kunnen geroteerd worden. Dit gebeurt met de toets 'r'. Dit gaat alleen met de kopie-, verbind-, smelt-samen- en splits-op-operatie.

Operaties kunnen nog op een andere manier gekopieerd worden. Indien men vertrekt van twee operatieblokken met verschillende operaties, kan men de operatie van het ene blok kopiëren naar het andere operatieblok. Dit kan door het eerste blok te selecteren en de toetsencombinatie 'control + c' in te drukken. Vervolgens dient de gebruiker het andere operatieblok te selecteren en de toetsencombinatie 'control + v' in te drukken.

Verbindingen maken en verwijderen

Het verbinden van blokken gebeurt door te klikken op een uitgang van een blok, de muis te slepen naar de gewenste ingang en vervolgens de muisknop los te laten. Een verbinding kan wel verlegd worden, door te klikken op een verbonden ingang/uitgang, de muis te slepen naar een andere ingang/uitgang en vervolgens de muisknop los te laten. Men kan ook samengestelde verbindingen creëren, dit is vooral handig indien men een verbinding wenst om te leiden, omwille van grafische redenen. Dit kan door te klikken op een uitgang, de muis te slepen naar een vrije plaats in het werkblad en vervolgens de muisknop los te laten terwijl men de control-toets ingedrukt houdt. Deze actie heeft als gevolg het ontstaan van een verbindingsblok. Er kunnen verschillende verbindingsblokken na elkaar gecreëerd worden.

Een verbinding kan verwijderd worden door ze te selecteren en vervolgens de delete-toets in te drukken.

Het is mogelijk om aan te geven dat uitgangen niet zullen worden gebruikt. Dit kan door dubbel te klikken op een uitgang. Op deze manier kan men vermijden dat de uitkomst van een berekeningsproces aangeeft dat niet alle uitgangen zijn verbonden.

Linken van blokken

Operatieblokken of samengestelde blokken kunnen gelinkt worden. Dit kan op twee manieren. Indien de twee te linken blokken al bestaan, selecteert de gebruiker één van de blokken en voert de toetsencombinatie ‘control + c’ in. Vervolgens selecteert de gebruiker het andere blok en drukt hij de toetsencombinatie ‘control + l’ in. Er verschijnt een groene transparante letter ‘l’ op beide blokken, beide blokken zijn gelinkt. Indien een gebruiker een gelinkte kopie van een blok wil maken, kan dit door op het blok te klikken, de muis te slepen naar een willekeurige lokatie in het werkblad en tijdens het loslaten van de muisknop de toetsencombinatie ‘control + alt’ ingedrukt te houden. Een link kan ongedaan gemaakt worden door een gelinkt blok te selecteren en vervolgens de letter ‘u’ in te drukken.

Indien de gebruiker wenst te controleren welke blokken met elkaar gelinkt zijn, kan hij een blok selecteren en de letter ‘l’ indrukken. Vervolgens worden alle blokken die gelinkt zijn met het geselecteerde blok ook geselecteerd.

In- en uitzoomen

De gebruiker kan in- en uitzoomen in het werkblad. Dit kan door de control-toets ingedrukt te houden en tegelijkertijd met de muis te scrollen.

Vergroten en verkleinen van de scène

In het werkblad is telkens een grote rechthoek getekend. Alleen in deze rechthoek kunnen zich netwerkelementen bevinden. Om de werkbare ruimte te vergroten moet men de scène vergroten. Ook het verkleinen van de scène is mogelijk. Beide acties kunnen gebeuren met de knoppen in de werkbalk. Deze functionaliteit werd al eerder besproken in onderdeel 7.2.3.

Weergeven van bitwoorden

Door rechts te klikken op een ingangs - of uitgangsblok kan men het overeenkomstige bitwoord laten weergeven in de bitwoordbalk. Dit kan men ook doen met verbindingslijnen. Dit werd reeds besproken in onderdeel 7.3.1.

7.4.2 Eigenschappen van blokken wijzigen

De eigenschappen van de verschillende blokken komen grotendeels overeen met de belangrijkste parameters van het blok. In het geval van operaties, komen deze eigenschappen overeen met de parameters in de overeenkomstige operatie-klasse. In onderstaande lijst worden alle blokken vermeld, samen met de eigenschappen die veranderd kunnen worden:

constante ingang: aantal bits, naam, bitwoord
variabele ingang: aantal bits, naam, bitwoord
I/O-ingang: aantal bits, naam, bitwoord, (index)
observatie-uitgang: naam, uitgang of niet
I/O-uitgang: naam, (index)
operatie:
- **logische operaties** aantal ingangen (niet bij NOT)
- **verschuiving:** aantal bits, richting
- **rotatie:** aantal bits, richting
- **substitutie-box:** aantal ingangsbits (aantal rijen, aantal kolommen), aantal uitgangsbits, tabelgegevens, permutatie of niet
- **bit shuffle:** aantal ingangsbits, aantal uitgangsbits, tabelgegevens
- **optelling:** (geen)
- **vermenigvuldiging:** (geen)
- **Galois vermenigvuldiging:** Galois veld
- **Galois inverse:** Galois veld
- **opslitsing:** aantal uitgangen, bitverdeling
- **samensmelting:** aantal ingangen
- **kopie:** aantal kopies
- **verbinding:** (geen)

uitbreiding: totaal aantal bits, veelvoud van totaal aantal bits toegelaten (booleaans)

bitwoordlengte: aantal bits uitgang

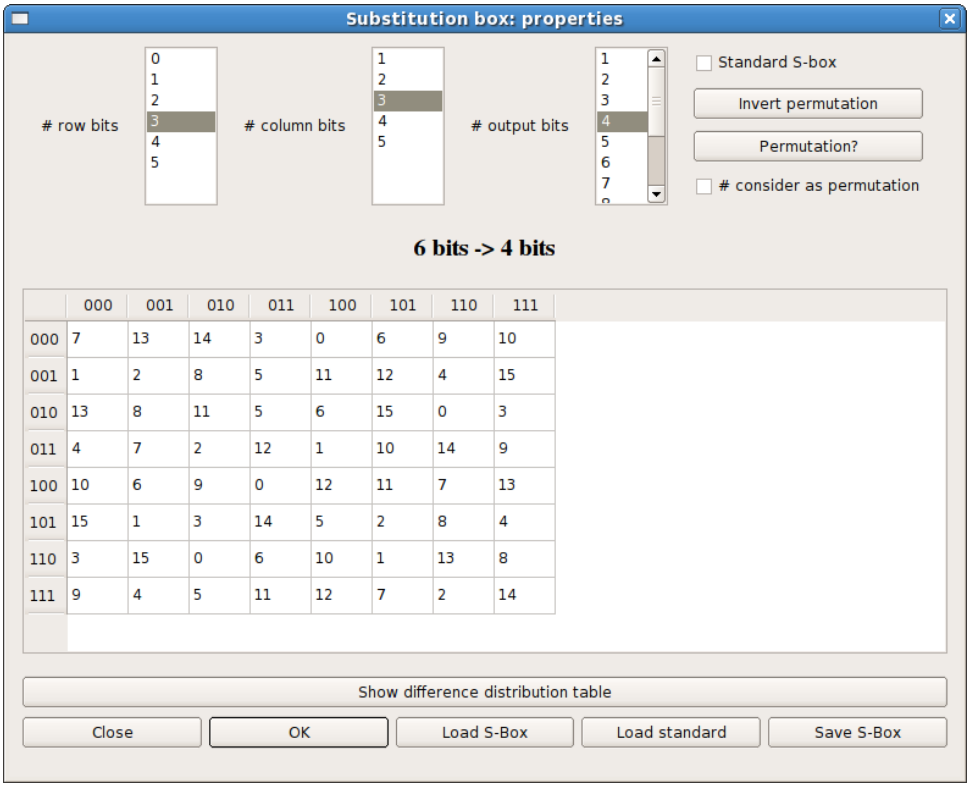
samengesteld blok: naam, aantal in- en uitgangen, ingekapselde scene (zie ook 7.4.3)

Een samenvatting van de eigenschappen van een blok wordt zichtbaar in het werkblad wanneer men de muispositie een tijdje vast houdt boven een blok. Dit is vooral interessant voor samengestelde blokken. Indien een samengesteld blok twee ingangen heeft, is het niet altijd duidelijk welke verbindingslijn met welke ingang van het samengestelde blok verbonden moet worden. De samenvatting van de eigenschappen van een samengesteld blok vermeldt de naam van elke I/O-ingang in de ingekapselde scène, die overeenkomt met elke ingang van het samengestelde blok.

De logische operaties, op de NOT-operatie na, hebben een variabel aantal ingangen. Dit aantal kan twee, drie of vier bedragen. De gebruiker kan dit aantal veranderen door het blok te selecteren en vervolgens de toetsen + en - te gebruiken. De andere wijzigbare blokken hebben telkens hun eigen eigenschap-dialoogvenster. Dit dialoogvenster verschijnt wanneer men dubbel klikt op het blok. Ter illustratie wordt het meest complexe dialoogvenster, dat van de S-box-operaties kort besproken. De andere dialoogvensters zijn gelijkaardig, maar meestal beperkter in omvang. Het dialoogvenster dat hoort bij het samengestelde blok wordt besproken in het volgende onderdeel, samen met het gebruik van ingekapselde scènes. Het dialoogvenster horende bij de S-box-operatie is afgebeeld in figuur 7.3.

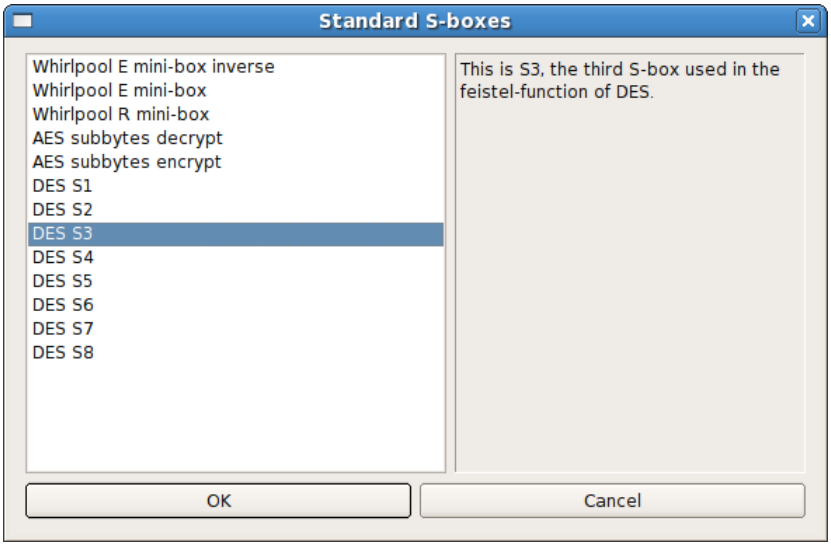
Bovenaan in het dialoogvenster kunnen de dimensies van de invoertabel van de S-box worden ingesteld. Daarnaast kan ook het aantal bits van de uitgang van S-box worden ingesteld. Dit aantal geeft een beperking op de waarden in de tabel. De waarden in de tabel zijn decimaal. Indien het aantal bits aan de ingang van de S-box gelijk is aan het aantal bits aan de uitgang, kan gecontroleerd worden of de S-box een permutatie is. Indien dit het geval is, kan de S-box geïnverteerd worden. Onder de tabel bevindt zich een knop om de verschil-distributietabel van de S-box weer te geven, meer uitleg hierover wordt gegeven in de volgende paragraaf. Onderin het dialoogvenster bevinden zich een aantal knoppen om gewone en standaard S-boxen in te laden. Indien de gebruiker ervoor kiest een standaard S-box in te laden, verschijnt een nieuw dialoog venster met een lijst van de beschikbare S-boxen. Dit venster wordt weergegeven in figuur 7.4. De instellingen van de S-box kunnen ook worden weggeschreven naar een XML-bestand.

De verschil-distributietabel van een S-box wordt gebruikt in de differentiële cryptanalyse [29]. Lezers die niet vertrouwd zijn met de basisbeginselen van de differentiële cryptanalyse kunnen terecht in [30]. In deze paper wordt op een heel duidelijke manier een differentiële aanval op een eenvoudig substitutie-permutatie netwerk van vier rondes uitgelegd. De verschil-distributietabel wordt er gedefinieerd en het gebruik ervan wordt er toegelicht.



	000	001	010	011	100	101	110	111
000	7	13	14	3	0	6	9	10
001	1	2	8	5	11	12	4	15
010	13	8	11	5	6	15	0	3
011	4	7	2	12	1	10	14	9
100	10	6	9	0	12	11	7	13
101	15	1	3	14	5	2	8	4
110	3	15	0	6	10	1	13	8
111	9	4	5	11	12	7	2	14

Figuur 7.3: Het dialoogvenster dat hoort bij een substitutie-box.



Figuur 7.4: Het venster met de lijst van standaard S-boxen.

Figuur 7.5: De knoppen in de werkbalk om de eigenschappen van de momenteel zichtbare scène op te vragen (1) en om te navigeren doorheen de verschillende scène-niveaus (2 en 3).

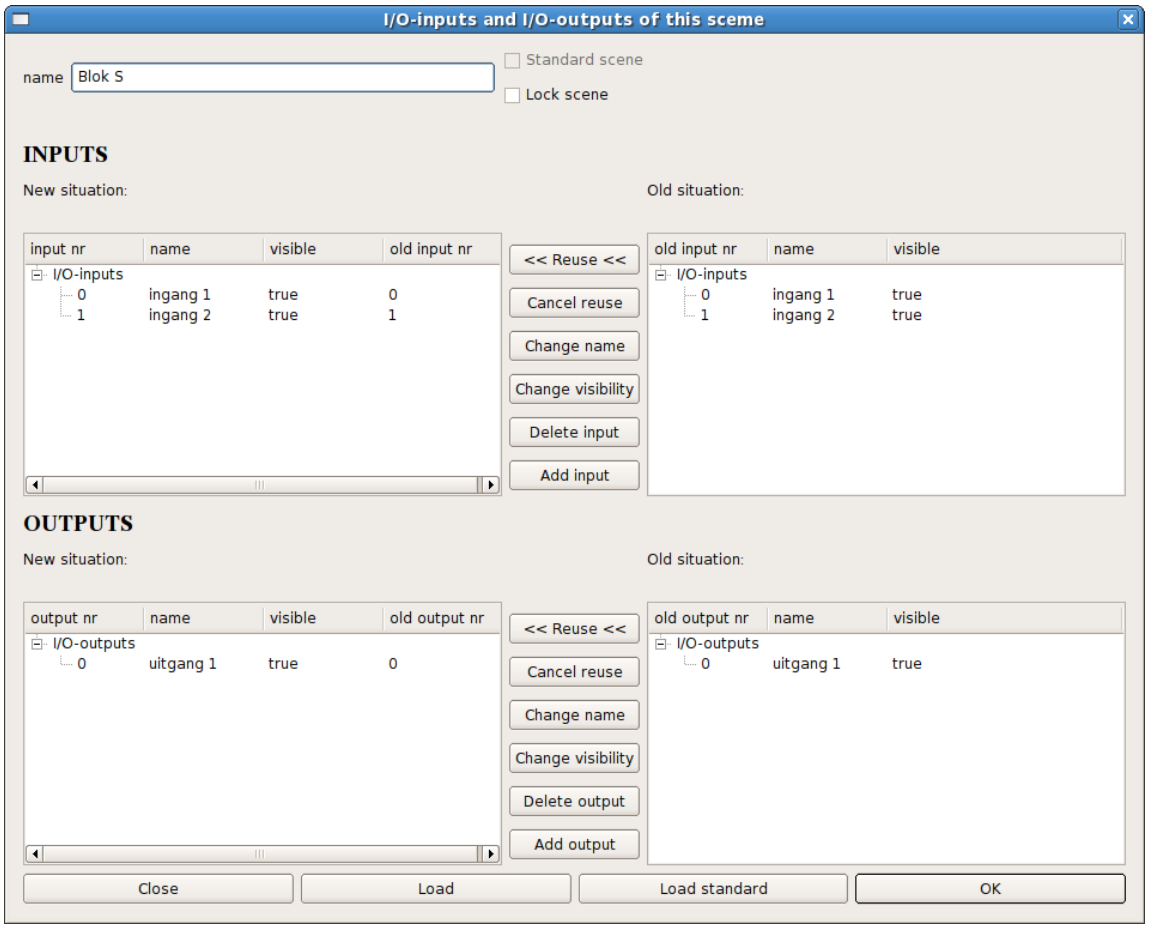
7.4.3 Gebruik van ingekapselde scènes

In deze uitleg wordt het gebruik van ingekapselde scènes besproken. Bij elke scène hoort een dialoogvenster. Het dialoogvenster horende bij de momenteel zichtbare scène kan geopend worden door de eerste knop in de werkbalk aan te klikken. Het dialoogvenster horende bij de ingekapselde scène van een samengesteld blok, kan geopend worden door dubbel te klikken op het samengestelde blok. Navigeren doorheen de verschillende scène-niveaus kan met behulp van de tweede en de derde knop in de werkbalk. Navigeren naar een ingekapselde scène kan door een samengesteld blok te selecteren en vervolgens de derde knop in de werkbalk aan te klikken. De aangehaalde knoppen worden voor de duidelijkheid afgebeeld in figuur 7.5.

Een mogelijk uitzicht van dit dialoogvenster is weergegeven in figuur 7.6. Dit dialoogvenster bestaat uit vier delen. Het bovenste deel herbergt enkele eigenschappen van de scène, zoals het vergrendeld of standaard zijn van de scène. Daarna volgen twee gelijkaardige delen waarin respectievelijk de in- en uitgangen van de scène ingesteld kunnen worden. Het onderste deel van het dialoogvenster bevat een aantal knoppen.

In het bovenste deel wordt weergegeven of een scène vergrendeld en/of standaard is. Vergrendelde scènes kunnen wel bekeken, maar niet gewijzigd worden. Standaard scenes zijn altijd vergrendeld. Eénmaal een scène niet meer standaard is, kan deze niet zomaar standaard gemaakt worden. In dit deel van het venster kan ook een naam ingesteld worden. Deze naam is de naam van het overeenkomstig samengesteld blok.

De middelste twee delen maken het mogelijk de I/O-ingangen en I/O-uitgangen van de scène aan te passen. Rechts staat telkens de oorspronkelijke toestand, links staat de nieuwe toestand. Gebruikmakende van de vijf knoppen in het midden, kunnen de ingangen aangepast worden. Elke in- en uitgang heeft een index die overeenkomt met de index van de in- of uitgang van het overeenkomstig samengesteld blok, indien het een ingekapselde scène betreft. De knop ‘reuse’ maakt het mogelijk bestaande in- of uitgangsblokken opnieuw te gebruiken onder een andere index. Verder kan de naam van een in- of uitgang worden aangepast en kan een blok tijdelijk onzichtbaar gemaakt worden. In- en uitgangen kunnen toegevoegd of verwijderd worden. Het maximaal aantal in- en uitgangen is elk acht, dit omwille van grafische redenen.

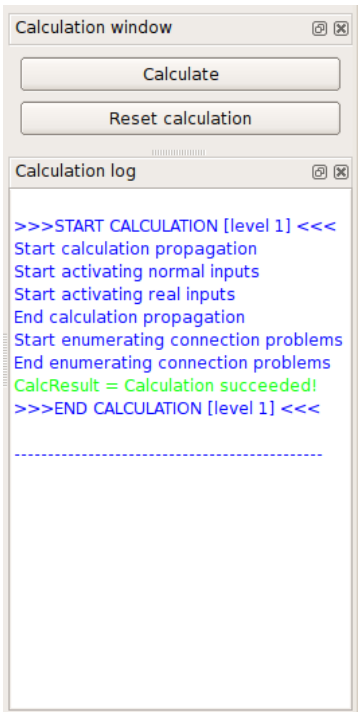


Figuur 7.6: Het dialoogvenster horende bij een scène.

Met de knoppen onderaan in het dialoogvenster kunnen zowel normale als standaard scènes worden ingeladen. Indien de gebruiker ervoor kiest een standaard scène in te laden, verschijnt een lijst met de beschikbare standaard scènes.

7.5 Berekeningsmodule

Een grafische weergave van de berekeningsmodule bevindt zich in figuur 7.7. De berekeningsmodule bestaat uit twee delen. Het eerste deel bevat twee knoppen. Met de eerste knop kan het berekeningsproces gestart worden, met de tweede worden de berekeningen gereset. Na het resetten van de berekeningen gaan de tussenresulaten verloren en verdwijnen de markeringen die ten gevolge van het berekeningsproces op de scène werden aangebracht. Het tweede deel bevat de log met feedback over het uitgevoerde berekeningsproces.



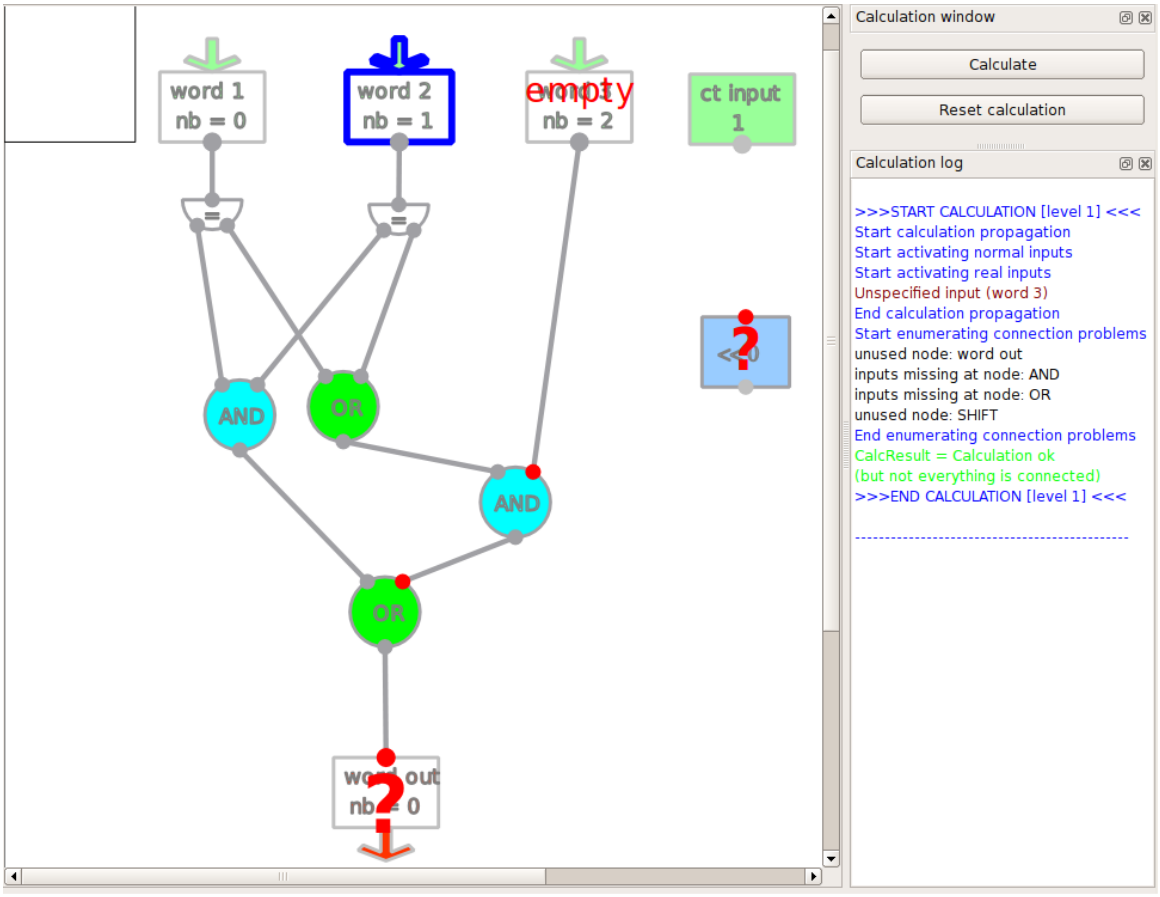
Figuur 7.7: Grafische weergave van de berekeningsmodule.

In deze sectie wordt een voorbeeld van een berekening gegeven, waarvan de feedback en de markeringen te zien zijn. In figuur 7.8 is een schermafbeelding te zien van het programma, nadat de berekeningen zijn uitgevoerd.

Er wordt duidelijk aangegeven welke blokken niet gebruikt worden en welke blokken niet correct verbonden zijn. Dit gebeurt zowel in de log, als in de scène zelf. Merk op dat eventuele markeringen ook in ingekapselde scènes zichtbaar zijn. Na de berekeningen kan de gebruiker zich begeven naar een ingekapselde scène en daar eventuele fouten in het berekeningsproces lokaliseren en verbeteren. Dit is niet van toepassing op het voorbeeld in de figuur, omdat in dit voorbeeld het operatienetwerk beperkt is tot één scène.

De log en de twee knoppen om de berekeningen te starten kunnen beiden verborgen worden. Dit kan via het menu “View” in de menubalk. In dit menu is er tevens een item waarmee men de gelogde informatie over het berekeningsproces kan beperken tot het strikt noodzakelijke.

Details over het berekeningsproces werden al gegeven in sectie 4.3. Een volledig overzicht van de verschillende markeringen op blokken is terug te vinden in onderdeel 5.3.2.



Figuur 7.8: Grafische weergave van het programma nadat een berekening werd uitgevoerd.

7.6 Handleiding

Met het oog op het werkelijk gebruiken van het programma, werd een compacte handleiding geschreven. Deze handleiding stelt de toekomstige gebruiker in staat snel kennis te maken met de principes die gehanteerd worden in het programma. Er wordt vooral aandacht besteed aan de functionaliteit in het werkblad, die in onderdeel 7.4.1 uitgebreid aan bod kwam.

Deze handleiding bevindt zich in bijlage D.

7.7 Besluit

In dit hoofdstuk werd de gebruikersinterface van het programma voorgesteld. De verschillende onderdelen van het programma werden vooral vanuit het standpunt van de gebruiker besproken. Op de implementatie van de grafische gebruikersinterface werd niet

rechtstreeks ingegaan. In dit hoofdstuk werd ook verwezen naar de compacte handleiding van dit programma, deze handleiding bevindt zich in bijlage D.

In het volgende hoofdstuk worden enkele cryptografische algoritmes die standaard in het programma aanwezig zijn besproken.

Hoofdstuk 8

Standaard cryptografische algoritmes en operaties

8.1 Inleiding

In dit hoofdstuk worden de cryptografische algoritmes, substitutie-boxen en bit-herverdelers die standaard in het programma aanwezig zijn kort besproken. In sectie 8.2 komen de volgende cryptografische algoritmes aan bod: AES, SHA-1 en Whirlpool. De structuur van elk algoritme wordt telkens kort voorgesteld en er wordt voor elk algoritme uitgelegd op welke manier het in verschillende operatienetwerken werd opgedeeld. In sectie 8.3 komen de operaties aan bod die standaard in het programma aanwezig zijn. In de laatste sectie wordt een besluit geformuleerd.

8.2 Cryptografische algoritmes

In het programma zijn drie veel gebruikte algoritmes standaard aanwezig. Deze algortimes kunnen in andere operatienetwerken opnieuw gebruikt worden. Na een berekening zijn de tussenresultaten in deze algoritmes vrij bestudeerbaar. Er werd één blokcijfer geïmplementeerd, namelijk AES. Daarnaast bevat het programma ook twee hashalgoritmes, namelijk SHA-1 en het meer veilige Whirlpool. In de volgende onderdelen wordt elk van deze algoritmes kort besproken. Er wordt ook telkens uitgelegd hoe het algoritme voorgesteld wordt in het programma.

8.2.1 Advanced Encryption Standard

De Advanced Encryption Standard wordt meestal afgekort als AES en staat geregistreerd onder “Processing Standards Publication 197” [18].

```
AESEncryption(byte in[16], byte out[16], 16 bytes subkeys[11])
begin
    byte state[4,4]
    state = in
    AddRoundKey(state, subkeys[0])
    for round = 1 to 9
       SubBytes(state)
       ShiftRows(state)
       MixColumns(state)
       AddRoundKey(state, subkeys[round])
    end for
    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, subkeys[10])
    out = state
end
```

Figuur 8.1: Het AES-encryptie-algoritme met een 128-bit-sleutel, in pseudocode.

Deze standaard specifieert hoe data geëncrypteerd en gedecrypteerd kan worden met een sleutel van 128, 192 of 256 bits. Ze is gebaseerd op het Rijndael-algoritme [31]. De klaartekst wordt telkens in blokken van 16 bytes geëncrypteerd. In deze tekst en ook in het programma wordt enkel de AES-versie met een 128-bit-sleutel beschouwd.

AES is een typisch voorbeeld van een substitutie-permutatie netwerk. AES stapt af van het in DES gebruikte Feistel netwerk [32]. DES is de afkorting van Data Encryption Standard en was de voorganger van AES. De ontwerpers van het AES-algoritme hielden rekening met de tot dan toe bekende aanvalstechnieken, alsook met mogelijkheden tot parallellisme zowel in software als hardware. Tot nu toe werden er nog geen zwaktes aangetoond. Het algoritme kan worden opgesplitst in twee delen. Er is het sleutelgeneratie-algoritme ('key schedule') en het algoritme om een datablok van zestien bytes te encrypteren gebruikmakende van de deelsleutels.

Het sleutelgeneratie-algoritme breidt de 128-bit-sleutel uit tot elf 128-bit-deelsleutels. Deze deelsleutels zijn het resultaat van een recursief proces, dat zowel lineaire als substitutie-operaties gebruikt.

Het encryptieproces zelf wordt in figuur 8.1 in pseudocode voorgesteld. Het datablok van zestien bytes wordt voorgesteld in een vierkante matrix *state*, dit refereert naar de term 'toestandsmatrix'. Deze matrix heeft vier rijen en vier kolommen, elk element is een byte. Het datablok van zestien bytes wordt rij per rij in de matrix geschreven.

Centraal in dit algoritme staan de vier stappen in de iteratielus: SubBytes, ShiftRows,

MixColumns en AddRoundKey. Deze vier stappen worden nu kort besproken.

stap 1: SubBytes In deze stap wordt elke byte in de toestandsmatrix vervangen door een andere byte. Het betreft een substitutie van bytes. De gebruikte substitutie-box en de opbouw ervan werd al besproken in onderdeel 3.7.2. Deze transformatie is in hoge mate niet-lineair.

stap 2: ShiftRows In deze stap wordt elke rij in de toestandsmatrix op zich beschouwd. Er vanuit gaande dat de rijen geïndexeerd worden vanaf nul, wordt de i-de rij, i bytes naar links geroteerd. De eerste rij blijft dus onveranderd. De andere rijen worden respectievelijk 1, 2 en 3 bytes naar links geroteerd. Deze transformatie zorgt voor een onderlinge menging van de bytes, zodat de MixColumns-stap die hierop volgt telkens andere bytes transformeert.

stap 3: MixColumns In deze stap wordt elke kolom in de toestandsmatrix op zich beschouwd. Elke kolom wordt beschouwd als een vector met vier elementen van het Galois veld GF(2^8) modulo $x^8 + x^4 + x^3 + x + 1$. Dit veld wordt ook wel eens het Rijndael-veld genoemd. Deze stap doet een lineaire transformatie in dit veld. De gebruikte transformatie kan worden voorgesteld met een matrix T en gebeurt als volgt:

$$\begin{pmatrix} s'_{0j} \\ s'_{1j} \\ s'_{2j} \\ s'_{3j} \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} s_{0j} \\ s_{1j} \\ s_{2j} \\ s_{3j} \end{pmatrix}$$

De gebruikte matrix is een cyclische MDS-matrix ('Maximum Distance Separable'). Deze stap is lineair en zorgt voor diffusie.

stap 4: AddRoundKey In deze stap wordt de deelsleutel gebruikt. De deelsleutel is 128 bits lang en wordt rij per rij in een deelsleutel-matrix geschreven. Deze matrix heeft dezelfde dimensies als de toestandsmatrix. Deze stap bestaat uit een XOR-operatie van elke byte van de toestandsmatrix `state` met de overeenkomstige byte van de deelsleutel-matrix. Het resultaat van deze operatie is opnieuw een matrix, deze matrix wordt de nieuwe toestandsmatrix.

Dit encryptie-algoritme heeft een bijhorend decryptie-algoritme. Het decryptie-algoritme kan bekomen worden door elke stap in het algoritme afzonderlijk te vervangen door de inverse stap. De volgorde waarmee de deelsleutels worden toegepast dient uiteraard te worden omgedraaid. Een meer volledige beschrijving van AES kan men vinden in [18]. Uitleg over resistentie tegen bepaalde aanvallen is te vinden in [31]. In [33] vindt men een interessante tutorial waarin de verschillende onderdelen van het AES-algoritme interactief worden aangebracht.

AES 128
→ AES 128 key expansion
→ AES 128 key expansion part
→ AES subbytes 4 bytes
→ AES 128 round
→ AES subbytes 4 bytes
→ AES 128 shiftrows
→ AES 128 mixcolumns
→ AES 128 mix one column

Figuur 8.2: De XML-documenten van de scènes die samen het AES-encryptie-algoritme vormen.

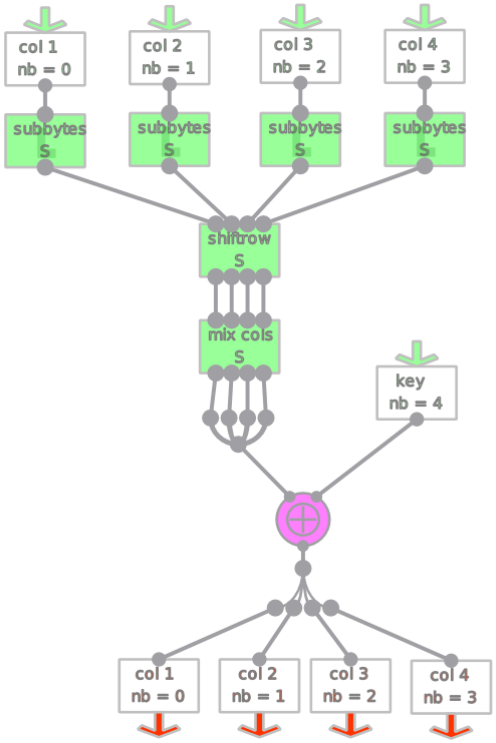
Structuur van AES in het programma

In het programma is zowel het encryptie-algoritme als het decryptie-algoritme aanwezig. Alleen de versie met 128 bits is standaard aanwezig. Het is echter mogelijk AES met een sleutellengte van 192 of 256 bits zelf aan te maken; heel wat ingekapselde operatienetwerken kunnen opnieuw gebruikt worden. In deze tekst wordt alleen het blokdiagram van het encryptie-algoritme toegelicht.

Om het overzicht te bewaren werd het blokdiagram opgesplitst in een aantal hiërarchische niveaus. Deze structuur dient de werkelijke structuur van het encryptie-algoritme te bewaren. Dit opsplitsen in hiërarchische niveaus resulteert dus in zogenaamde zwarte dozen, waarin functionaliteit geherbergd wordt. Een dergelijke zwarte doos komt overeen met een samengesteld blok, met achterliggende scène. Elk hiëarchisch onderdeel heeft een eigen XML-document. In figuur 8.2 is te zien welke XML-documenten gebruikt worden door de AES-encryptie. Deze XML-documenten zijn terug te vinden in de map `StandardPackages`. Deze map maakt deel uit van het programma.

In figuur 8.3 is de grafische voorstelling van één AES-encryptie-ronde te zien. Men kan de vier besproken stappen duidelijk herkennen. De gebruiker kan verder navigeren in de hiërarchie om bijvoorbeeld de structuur van het samengestelde blok ShiftRows te zien. In bijlage E.2 vindt de lezer het XML-document van het operatienetwerk in de figuur 8.3.

Het sleutelgeneratie-algoritme en het encryptie-algoritme zelf gebeuren in het programma na elkaar. Een structuur waarbij in elke stap een nieuwe deelsleutel gegenereerd wordt, is mogelijk. In de praktijk worden beide gebruikt, soms worden eerst alle deelsleutels aangemaakt en worden vervolgens alle encryptie-rondes doorlopen, in andere gevallen kan men de aanmaak van de deelsleutels parallelliseren met de encryptie. Dit hangt af van de hardware en/of software omgeving.



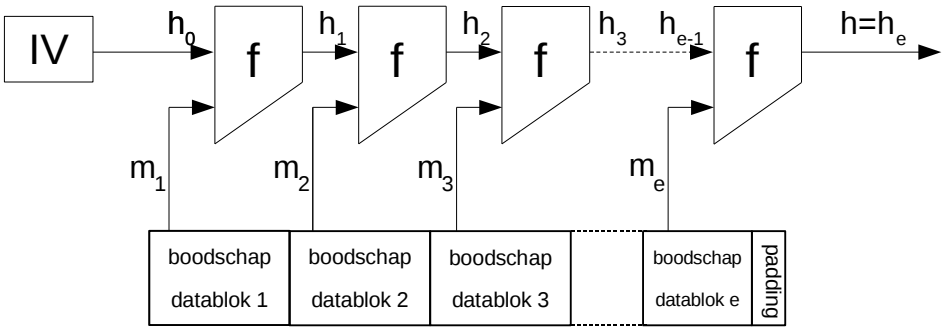
Figuur 8.3: Grafische weergave van een AES-encryptie-ronde in het werkblad.

8.2.2 Secure Hash Algorithm 1

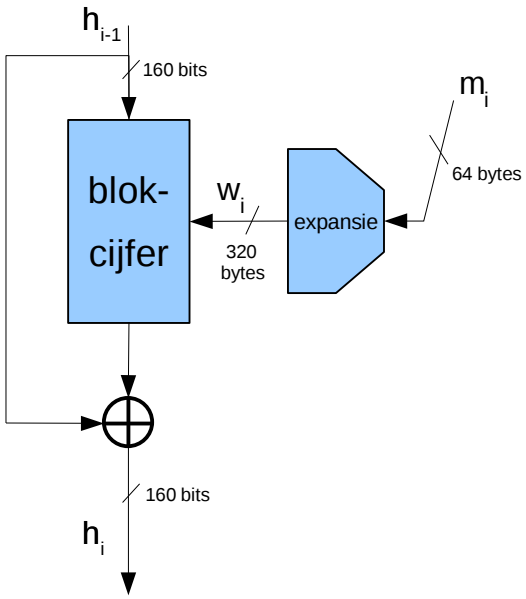
Het Secure Hash Algorithm wordt meestal afgekort als SHA-1 en staat geregistreerd onder "Processing Standards Publication 180-2" [34]. SHA-1 is tot op heden één van de meest gebruikte hash-functies. De veiligheid ervan komt wel in het gedrang, want in 2005 werd door Wang [35] een aanval op dit hash-algoritme gevonden. Het blijkt namelijk mogelijk te zijn om binnen redelijke tijd botsingen te vinden. De eerder genoemde aanval werd nog niet lang geleden concreter gemaakt in een paper van De Cannière and Rechberger [36]. Het is dus af te raden om SHA-1 te gebruiken in nieuwe applicaties.

De hash-functie SHA-1 transformeert een boodschap die kleiner is dan 2^{64} bits naar een hash-waarde van 160 bits. De eerste stap in het algoritme is de Padding-stap. Een boodschap `m` wordt in deze stap verlengd tot een boodschap `m'`, zodat de lengte van `m'` een veelvoud is van 512 bits. In de bits die de boodschap verlengen zit onder andere de lengte van de boodschap verwerkt, vandaar de naam Length Padding. De verlengde boodschap wordt vervolgens als ingang gebruikt voor een Damgård/Merkle iteratieve structuur [37, 38]. Deze structuur wordt gebruikt om cryptografische hash-functies te bouwen. De Damgård/Merkle iteratieve structuur wordt weergegeven in figuur 8.4.

De Damgård/Merkle iteratieve structuur maakt gebruik van een niet nader gespecifieerde functie f. Deze functie hangt af van de hash-functie en wordt de compressiefunctie van



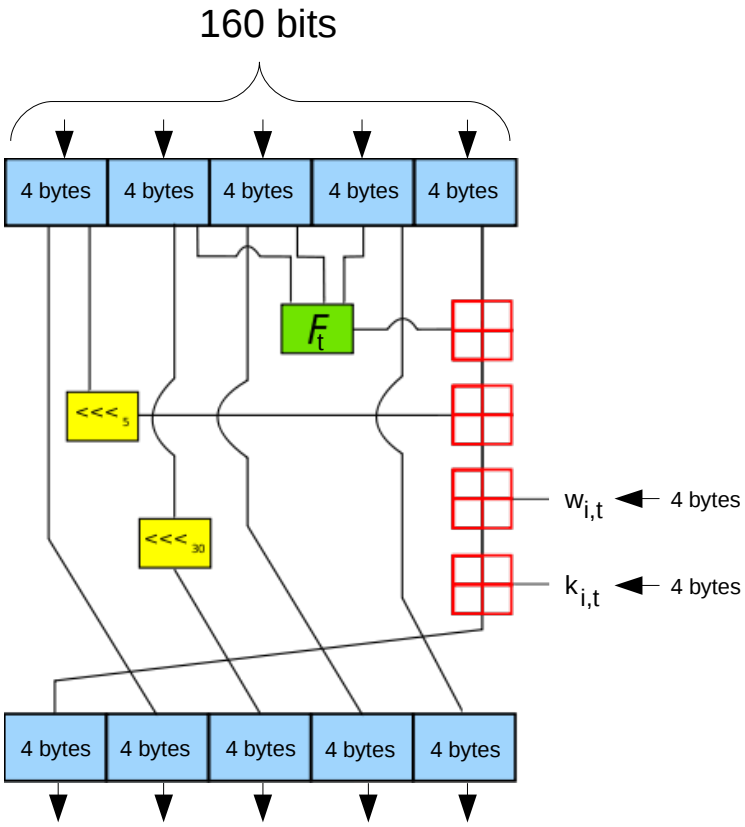
Figuur 8.4: De Damgård/Merkle iteratieve structuur met compressiefunctie f, toegepast op e datablokken.



Figuur 8.5: De compressiefunctie van SHA-1.

de hash-functie genoemd. De compressiefunctie van SHA-1 heeft 160 bits aan de ingang en 160 bits aan de uitgang. Ze wordt schematisch voorgesteld in figuur 8.5.

De boodschapsexpansie in de compressiefunctie breidt de ingang van 64 bytes uit tot 320 bytes met behulp van XOR- en rotatie-operaties. Het blokcijfer bestaat uit 80 rondes. In elke ronde worden 4 van de 320 bytes afkomstig van het expansie-blok gebruikt. In figuur 8.6 wordt één ronde voorgesteld. Aan de in- en uitgang van elke ronde bevinden zich inderdaad 160 bits, meer bepaald 5 woorden van 4 bytes.



Figuur 8.6: Ronde t in de i-de compressiefunctie van het SHA-1-algoritme; bron [39].

Rechts in figuur 8.6 bevindt zich de ingang $w_{i,t}$. Dit zijn de vier bytes, afkomstig van het expansie-blok van de i-de compressiefunctie, die gebruikt worden in ronde t. Er is ook nog een constante k die tevens afhankelijk is van het rondenummer. In figuur 8.6 is te zien dat optellingen en rotaties gebruikt worden, alsook een ronde-afhankelijke operatie F_t. De verschillende waarden voor k en de verschillende functies F worden in tabel 8.1 weergegeven.

Tabel 8.1: De verschillende waarden voor k en de verschillende functies $F(x, y, z)$ voor elke ronde in het SHA-1-algoritme; bron [35].

ronde	booleaanse functie F_t	constante k_t
1-20	IF: $(x \wedge y) \vee (x \wedge z)$	`0x5a827999`
21-40	XOR: $x \oplus y \oplus z$	`0x6ed6eba1`
41-60	MAJ: $(x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$	`0x8fabbcdc`
61-80	XOR: $x \oplus y \oplus z$	`0xca62c1d6`

SHA1 less than 448 bits
→ SHA1 padding less than 448 bits
→ SHA1 compression function
→ SHA1 20 iterations (steps 1-20)
→ SHA1 extend-step
→ SHA1 one iteration (for step 1-20)
→ SHA1 F (step 1-20)
→ SHA1 20 iterations (steps 21-40)
→ ...
→ SHA1 20 iterations (steps 41-60)
→ ...
→ SHA1 20 iterations (steps 61-80)
→ ...

Figuur 8.7: De XML-documenten van de scènes die samen het SHA-1-algoritme vormen.

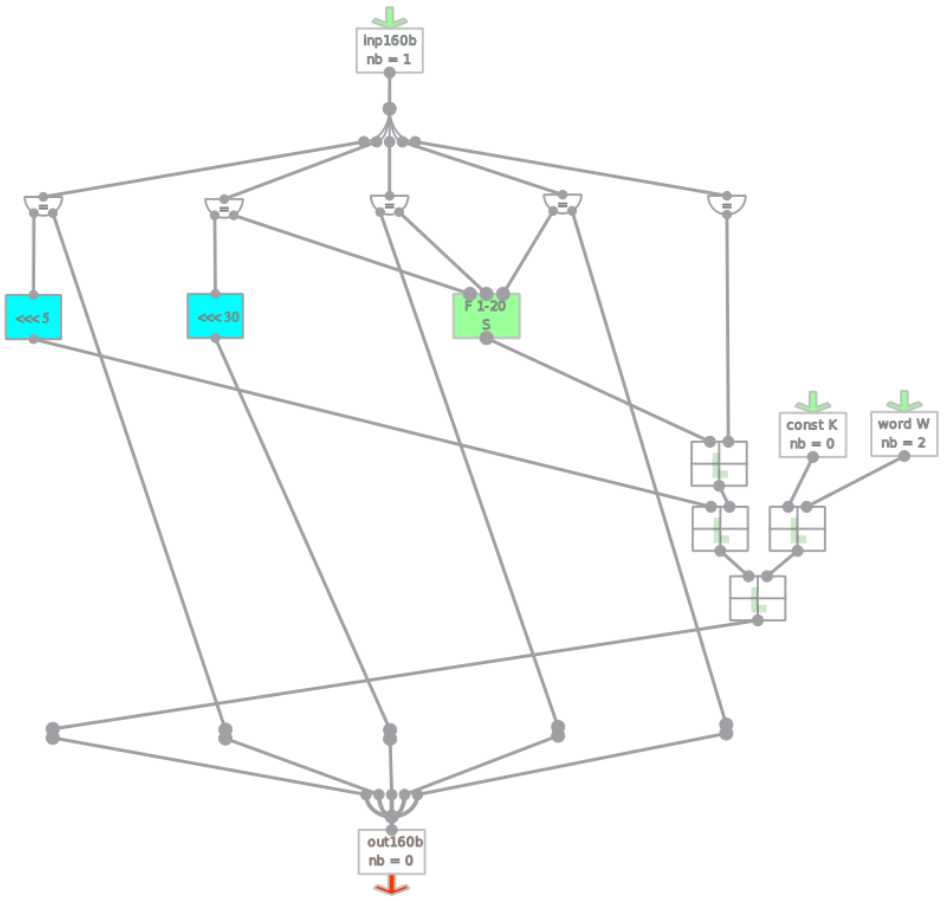
Voor details over dit hash-algoritme kan de lezer terecht in de specificaties van SHA-1 [34]. Eerder in dit onderdeel werd afgeraden om SHA-1 nog langer te gebruiken. Mogelijk alternatieven zijn SHA-256 en SHA-512. De hash-waardes van deze algoritmes zijn langer en deze algoritmes zijn momenteel nog veilig. Een ander hash-algoritme, waarvan het ontwerp geïnspireerd is op het Rijndael-algoritme, is Whirlpool. Dit hash-algoritme bevindt zich ook in het programma en wordt kort besproken in onderdeel 8.2.3. Meer informatie over alle in dit onderdeel besproken hash-functies vindt men terug in [19].

Structuur van Secure Hash Algorithm 1 in het programma

Het SHA-1-algoritme is standaard in het programma aanwezig. Het betreft een versie die slechts één iteratie in de Damgård/Merkle structuur ondersteunt. Met één compressie-functie in de Damgård/Merkle structuur kan men boodschappen van minder dan 448 bits hashen. In dit onderdeel wordt alleen het blokdiagram van het hash-algoritme toegelicht.

Het blokdiagram werd opgesplitst in een aantal hiërarchische niveaus. Elk hiërarchisch onderdeel heeft een eigen XML-document. In figuur 8.7 is te zien welke XML-documenten gebruikt worden door het SHA-1-algoritme.

In figuur 8.8 bevindt zich één SHA-1-ronde zoals ze in het programma wordt weergegeven. Het operatienetwerk komt overeen met het netwerk in figuur 8.6.

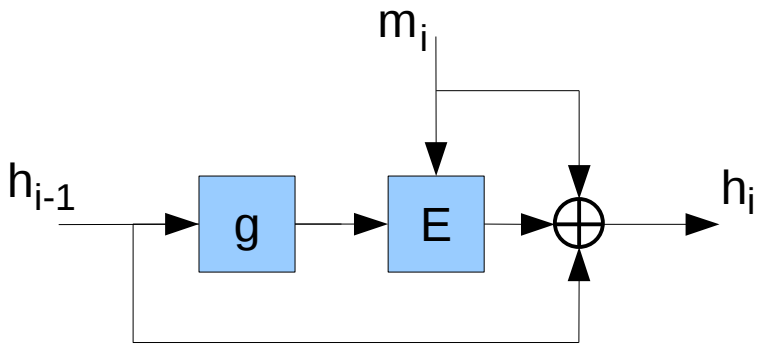


Figuur 8.8: Eén ronde in het SHA-1-algoritme; de functie F uit rondes 1 tot 20 wordt gebruikt.

8.2.3 De hash-functie Whirlpool

Naast SHA-1, is ook de hash-functie Whirlpool standaard aanwezig. Over deze hash-functie kan men heel wat documentatie vinden in [40]. Whirlpool is de naam van het resultaat van een meerjarig project. Er gingen twee versies aan vooraf, namelijk Whirlpool-0 en Whirlpool-T. De Whirlpool-T-versie werd aangepast nadat Shirai en Shibutani in 2001 een zwakheid vonden in de S-box die in het algoritme gebruikt werd. De uiteindelijke versie, die Whirlpool genoemd wordt, werd in 2003 gestandaardiseerd onder ISO/IEC 10118-3 [19]. Het algoritme werd later nog een keer aangepast, de circulaire diffusiematrix werd gewijzigd.

Net als bij SHA-1 wordt de te hashen boodschap eerst verlengd tot een veelvoud van 512 bits met Length Padding. Whirlpool ondersteunt grotere boodschaplengtes dan SHA-1, alle boodschappen kleiner dan 2^{256} bits worden ondersteund. Na het padden bekomt men een aantal blokken m_i van 512 bits. Deze blokken worden gebruikt in het



Figuur 8.9: De Miyaguchi-Preneel compressiefunctie.

Miyaguchi-Preneel schema. In dit schema wordt de compressiefunctie uit figuur 8.9 voor elk blok m_i herhaaldelijk toegepast. Elke h_i bestaat uit 512 bits. Het blok h_0 bestaat uit 512 0-bits.

In Whirlpool komen de functies g en E overeen met het blokcijfer W. De structuur ervan is gebaseerd op die van het Rijndael-algoritme. Deze korte bespreking beperkt zich tot het opsommen van de belangrijkste verschilpunten tussen het AES-algoritme en Whirlpool. Dit gebeurt in tabel 8.2. Uitleg over de volledige structuur van dit algoritme kan men vinden in [19] en [41].

Tabel 8.2: De belangrijkste verschilpunten tussen AES en het blokcijfer W dat in Whirlpool gebruikt wordt; bron [40].

	AES	**blokcijfer W**
Blokgrootte (bits)	128, 192 of 256	512
Aantal rondes	10, 12 of 14	10
Sleutelgeneratieschema	onafhankelijk recursief algoritme	gebruikt de ronde-functie
Datavoorstelling	4 x 4 byte-matrix, data kolom per kolom	8 x 8 byte-matrix, data rij per rij
Irreduceerbare polynoom	$x^8 + x^4 + x^3 + x + 1$	$x^8 + x^4 + x^3 + x^2 + 1$
Oorsprong S-box	inverseberekening in $GF(2^8)$, daarna affiene transformatie	recursieve structuur op basis van twee 4 x 4 mini S-boxen
Diffusielaag	vermenigvuldiging met circulante matrix cir(2, 3, 1, 1)	vermenigvuldiging met circulante matrix cir(1, 1, 4, 1, 8, 5, 2, 9)

Whirlpool less than 256 bits
→ Whirlpool padding
→ Whirlpool W block cipher
→ Whirlpool round function
→ Whirlpool S-box all 8 rows
→ Whirlpool S-box 1 row
→ Whirlpool S-box
→ Whirlpool rotate columns
→ Whirlpool linear diffusion layer
→ Whirlpool linear diffusion layer one row
→ Whirlpool dot product 8 bytes

Figuur 8.10: De XML-documenten van de scènes die samen het Whirlpool-algoritme vormen.

Structuur van Whirlpool in het programma

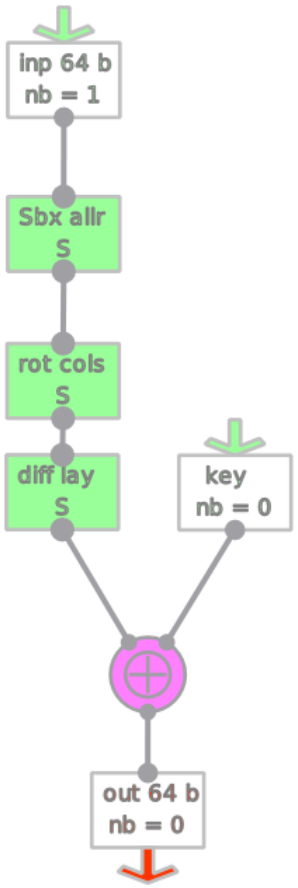
In het programma is het Whirlpool-algoritme standaard aanwezig. Het betreft een versie die slechts één iteratie in het Miyaguchi-Preneel schema ondersteunt. Met één compressiefunctie in het Miyaguchi-Preneel schema kan men boodschappen van minder dan 256 bits hashen. Boodschappen die minder dan 256 bits bevatten, worden tijdens het padden immers verlengd tot 512 bits. De gebruiker kan zelf een versie aanmaken die langere boodschappen aankan. Dit is vrij eenvoudig aangezien de XML-documenten die horen bij de compressiefunctie en het padding-schema opnieuw gebruikt kunnen worden.

Om het overzicht te bewaren werd het blokdiagram opgesplitst in een aantal hiërarchische niveaus. Elk hiërarchisch onderdeel heeft een eigen XML-document. In figuur 8.10 is te zien welke XML-documenten gebruikt worden door het Whirlpool-algoritme.

In figuur 8.11 wordt de grafische weergave van een ronde in de Whirlpool-functie W afgebeeld. Eerst is er de substitutie-stap, daarna de rotatie-stap, vervolgens de diffusie-stap in het veld $GF(2^8)$. De laatste stap, de XOR-operatie, is niet te zien in figuur 8.10 omdat deze operatie niet in een afzondelijke scène gebeurt.

8.3 Operaties

In het programma zijn een aantal S-boxen standaard aanwezig. Sommige van deze operaties worden gebruikt in de cryptografische algoritmes uit sectie 8.2. Er zijn echter ook een aantal operaties die deel uitmaken van het programma, maar die louter ter beschikking van de gebruiker staan. Het betreft de S-boxen en bit-herverdelers uit het



Figuur 8.11: Grafische weergave van één ronde in de Whirlpool-functie W.

DES-algoritme [15]. Het DES-algoritme is niet standaard aanwezig in het programma, maar kan door de gebruiker al dan niet gedeeltelijk zelf worden aangemaakt. Dit wordt vergemakkelijkt doordat de S-boxen en bit-herverdelers uit dit algoritme deel uitmaken van het programma.

De operaties die standaard deel uitmaken van het programma zijn:

- De S-box gebruikt in de AES-encryptie en diens inverse S-box
- De mini S-boxen gebruikt in de substitutie-stap van het Whirlpool-algoritme, alsook de volledige S-box uit deze stap
- Alle S-boxen en bit-herverdelers uit DES

8.4 Besluit

In dit hoofdstuk werden de standaard aanwezige cryptografische algoritmes besproken, dit gebeurde in sectie 8.2. Elk cryptografisch algoritme werd kort beschreven, de structuur ervan werd telkens besproken. Vervolgens werd uitgelegd hoe elk algoritme in het programma in verschillende hiërarchische onderdelen werd opgedeeld. In sectie 8.3 werd een overzicht gegeven van de S-boxen en de bit-herverdelers die standaard in het programma aanwezig zijn.

Zowel de standaard aanwezige operaties, als de standaard aanwezige algoritmes kunnen opnieuw gebruikt worden door de gebruiker. Ook de onderdelen waarmee de algoritmes zijn opgebouwd, kunnen allen opnieuw gebruikt worden.

Hoofdstuk 9

Mogelijke uitbreidingen

9.1 Inleiding

In dit hoofdstuk worden enkele mogelijke uitbreidingen voor het programma voorgesteld. Deze uitbreidingen werden opgesplitst in drie delen. Er zijn de uitbreidingen die vooral invloed hebben op de interne voorstelling van een netwerkstructuur, deze worden voorgesteld in sectie 9.2. Daarna worden in sectie 9.3 een aantal uitbreidingen voorgesteld die eerder verband houden met de grafische gebruikersinterface. In sectie 9.4 wordt uitleg gegeven over een plugin om een operatienetwerk te transformeren naar een implementatie in een programmeertaal. In de laatste sectie wordt een uitgebreid besluit geformuleerd.

9.2 Uitbreidingen van de netwerkstructuur

9.2.1 Geheugenelementen

Met het huidige programma kan men alleen operatienetwerken aanmaken die geen terugkoppelingen bevatten, er zijn immers geen blokken met een geheugen beschikbaar. Indien men over geheugenelementen beschikt, kan men stroomcijfers tekenen. In stroomcijfers gebruikt men vaak een Linear Feedback Shift Register of LFSR. In figuur 9.1 wordt een LFSR met vier geheugenelementen voorgesteld. Het is de LFSR met primitieve veelterm $x^4 + x + 1$. Uitleg over de veeltermvoorstelling van een LFSR is te vinden in [42].

Indien er geheugenelementen aanwezig zijn, kan het berekeningsproces onmogelijk voltooid zijn nadat elk blok eenmaal betrokken werd in dit proces. Het berekeningsproces moet worden aangepast en moet een aantal keer doorlopen worden. Men kan het bitwoord dat momenteel aanwezig is in het geheugenelement, weergeven op het geheugenblok, zoals dat gebeurt bij een observatie-uitgang.

Om de werking van een LFSR te demonstreren, moeten meerdere opeenvolgende ingangen

Figuur 9.1: Linear Feedback Shift Register met primitieve veelterm $x^4 + x + 1$.

aangelegd worden. Er is dus nood aan een nieuw type ingangsblok dat meerdere ingangen kan bevatten.

Het gebruik van geheugenelementen heeft ook belangrijke gevolgen voor het bijhouden van de tussenresultaten. Indien men het berekeningsproces vier keer uitvoert in het operatienetwerk van figuur 9.1, zijn langs sommige connecties vier verschillende bitwoorden voorbijgekomen. Aangezien één van de in sectie 2.3 geformuleerde vereisten van het programma is dat alle tussenresultaten moeten worden bijgehouden, moeten deze vier tussenresultaten alle vier worden bijgehouden. Het is dus noodzakelijk om in elke connectie niet één bitwoord bij te houden, maar een rij van bitwoorden. Men kan uiteraard een optie voorzien waarmee het bijhouden van alle vier de tussenresultaten wordt uitgeschakeld, om het geheugengebruik te beperken.

9.2.2 XML-formaat voor een verzameling bitwoorden

Het programma is momenteel enkel voorzien van enkelvoudige ingangen. Elke ingang bevat precies één bitwoord. In het vorige onderdeel werd voorgesteld om een nieuw type ingangsblok in te voeren dat meerdere ingangen kan bevatten. De uitbreiding in dit onderdeel bestaat erin een nieuw XML-formaat te definiëren waarin een verzameling van bitwoorden wordt bijgehouden. De bitwoorden in een dergelijk XML-document kunnen worden ingeladen in dat nieuwe ingangsblok. Men kan het formaat ook gebruiken om de verschillende tussenresultaten die horen bij een verbinding op te slaan.

9.2.3 Demonstreren van differentiële paden

Het programma kan worden uitgebreid met eenvoudige tools voor differentiële cryptanalyse. In sectie 7.4.2 werd reeds verwezen naar de verschil-distributietabel. De verschil-distributietabel van een S-box kan men met het huidige programma al opvragen. Om het vervolg van dit onderdeel te begrijpen, dient men vertrouwd te zijn met de beginselen van de differentiële cryptanalyse. Men kan daarvoor terecht in [29, 30].

In differentiële cryptanalyse wordt typisch in termen van koppels van bitwoorden gedacht. Het bestuderen van een eenvoudig substitutie-permutatie netwerk resulteert wel eens

in het vinden van een interessant XOR-ingangsverschil. Het ingangsverschil van het koppel (`ingang`$_1$,`ingang`$_2$) is `ingang`$_1 \oplus$ `ingang`$_2$. Een XOR-ingangsverschil is interessant wanneer men statistisch verantwoorde vermoedens heeft dat dit ingangsverschil zich op een bepaalde manier doorheen het cijfer zal voortplanten. Men kan het XOR-verschil van alle bitlijnen in het cijfer voor de twee ingangen $ingang_1$ en $ingang_2$ bestuderen. Deze XOR-verschillen bepalen wat men noemt het differentieel pad.

De uitbreiding bestaat er in de mogelijkheid te voorzien om differentiële berekeningen uit te voeren. Er moet dus een differentiële modus toegevoegd worden aan het programma. In deze modus kan men gebruikmaken van een nieuw type ingangsblok, dat twee bitwoorden bijhoudt met een specifiek ingangsverschil. Indien men berekeningen uitvoert in de differentiële modus, is het tussenresultaat dat hoort bij een bepaalde verbindingslijn gelijk aan het XOR-verschil van de tussenresultaten die horen bij deze verbindingslijnen indien men beide ingangen los van elkaar had toegepast.

De uitbreiding maakt het ook mogelijk een differentieel pad aan te duiden in een cijfer. Op deze manier kan de gebruiker experimenteren. Hij kan een differentieel pad invoeren en vervolgens voor verschillende ingangskoppels controleren of het pad gevolgd wordt. Merk op dat differentiële cryptanalyse op bitniveau gebeurt. De meeste verbindingslijnen in het programma zijn multi-bitlijnen. In de differentiële modus moet het mogelijk zijn de in- en uitgangen van sommige blokken op te splitsen per bit. Dit is vooral belangrijk bij de S-box. Een S-box heeft momenteel slechts één ingang en één uitgang. In een differentiële mode kan het interessant zijn het aantal in- en uitgangen te laten afhangen van het aantal in- en uitgangsbits van de S-box. Men kan de bits uiteraard ook op bitwoord-niveau bestuderen.

9.2.4 Flexibelere observatie-uitgangen

In het programma kan de gebruiker bitwoorden horende bij een verbinding op twee manieren bekijken. Dit kan door rechts te klikken op een verbinding, het overeenkomstige bitwoord verschijnt vervolgens in de bitwoordbalk. Een andere manier is het gebruiken van een observatie-uitgang. Indien het bitwoord bestaat uit acht bits of minder verschijnt het bitwoord in het blok. In dit geval is het bitwoord eenvoudig af te lezen bij het bekijken van het operatienetwerk.

Het kan interessant zijn om een flexibelere observatie-uitgang toe te voegen. De breedte van dit nieuwe observatie-uitgangsblok wordt automatisch aangepast aan de grootte van het bitwoord. De grootte van het bitwoord dient wel op voorhand te worden ingesteld. Bovendien is het mogelijk het bitwoord op te splitsen in bepaalde onderdelen, zodat bijvoorbeeld de vier bytes in een 32-bit-woord grafisch duidelijk tot uiting komen.

9.3 Uitbreidingen van de grafische gebruikersinterface

9.3.1 Meerdere vensters

In het huidige programma kan men slechts één document tegelijkertijd openen. Men zou het programma kunnen uitbreiden zodat het mogelijk is om meerdere documenten tegelijkertijd te openen, elk in een ander deelvenster. Deze uitbreiding is zeker haalbaar, vanwege de programmatorische structuur van de grafische gebruikersinterface.

9.3.2 Werkbalk voor het invoegen blokken

Blokken toevoegen kan enkel met menu 'Add Nodes'. Het nieuwe blok verschijnt telkens links bovenaan in de scène. Men kan het toevoegen van nieuwe blokken vergemakkelijken voor de gebruiker door een werkbalk toe te voegen, met voor elk type blok een knop. Een andere manier is het gebruik van een extra plugin, bijvoorbeeld links in het scherm. In deze plugin bevinden zich alle blokken onder elkaar in hun ware grafische voorstelling. Het invoegen van een nieuw blok kan dan gebeuren door een blok naar de gewenste locatie in het werkblad te slepen. Hiervoor kan men de klasse QToolBox gebruiken, voor een implementatievoorbeeld kan men terecht in [24].

9.3.3 Flexibelere blokken

Met het huidige programma beschikt de gebruiker over heel wat mogelijkheden om een operatienetwerk aan te maken. Deze mogelijkheden werden besproken in onderdeel 7.4.1. Wat betreft het omvormen van de blokken op zich, zijn de mogelijkheden echter beperkt. De plaatsing van de in- en uitgangen op de blokken suggereert een netwerkopbouw van boven naar beneden. Het is weliswaar mogelijk om sommige nepoperaties (kopieer, smelt-samen, splits-op en verbind) te roteren, maar het zou ook interessant zijn mochten ook de andere blokken geroteerd kunnen worden. Dit werd vooralsnog verhinderd. Het volstaat immers niet om de blokken te roteren, het is immers de bedoeling dat de tekst die zich in sommige blokken bevindt, leesbaar blijft.

9.3.4 Uitgebreidere zoom-functionaliteit

In het huidige programma kan men enkel inzoomen en uitzoomen door het muiswiel te gebruiken in combinatie met de toets 'control'. Men zou in de bovenste werkbalk enkele knoppen kunnen toevoegen om het in- en uitzoomen toegankelijker te maken.

9.4 Exporteren van een operatienetwerk

Een interessante uitbreiding zou een plugin kunnen zijn, waarmee men een volledig operatienetwerk kan exporteren naar een C-implementatie. Men kan bijvoorbeeld een cijfer-onderdeel tekenen met dit programma. De uitbreiding genereert een C-functie die dit cijfer-onderdeel implementeert. Dit is uiteraard niet evident, omdat deze automatisch gegenereerde implementatie toch enigzins efficiënt moet zijn.

In voorgaande paragraaf werd een C-implementatie voorgesteld, maar deze uitbreiding kan ook gericht zijn op een hardwarebeschrijving, bijvoorbeeld in VHDL [5].

9.5 Besluit

In dit hoofdstuk werden een aantal mogelijke uitbreidingen voorgesteld. Het betrof drie type uitbreidingen.

Eerst werden enkele uitbreidingen van de netwerkstructuur voorgesteld. Belangrijk was het toevoegen van geheugenelementen en een nieuw type ingangsblok. Dit ingangsblok bevat een verzameling van bitwoorden en kan gebruikt worden als ingang voor een LFSR. Om dit te realiseren zijn enkele wijzigingen nodig in de netwerkstructuur. Het aantal bitwoorden dat wordt bijgehouden door een connectie is niet langer één. Een tweede uitbreiding maakt het mogelijk een verzameling van bitwoorden op te slaan en in te laden in een speciaal daarvoor ontworpen XML-formaat. De meest interessante uitbreiding richt zich op het illustreren van een eenvoudig differentiëel pad. Het programma werkt hiervoor in een differentiële modus. In deze modus zijn de ingangen koppels van bitwoorden en zijn de tussenresultaten XOR-verschillen.

Daarna werden enkele uitbreidingen voorgesteld die verband houden met de grafische gebruikersinterface. Deze uitbreidingen verhogen de gebruiksvriendelijkheid. Er werden vier uitbreidingen beschreven. Een eerste was het gebruik van vensters, waardoor meerdere documenten tegelijkertijd geopend kunnen worden. Een tweede uitbreiding maakt het mogelijk blokken toe te voegen door ze in de scène te slepen. Het betreft een plugin van waaruit men de verschillende blokken in het werkblad kan slepen. Een derde uitbreiding maakt het mogelijk om alle blokken vrij te roteren. Tot slot kan men de zoomfunctionaliteiten die momenteel aanwezig zijn uitbreiden. De tweede uitbreiding, namelijk het mogelijk maken om nieuwe blokken in het werkblad te slepen, is het meest interessant.

Tot slot kwam een uitbreiding aan bod die het mogelijk moet maken om op basis van operatienetwerken volautomatisch implementaties te genereren in programmeertalen als C en VHDL.

Hoofdstuk 10

Algemeen besluit

In deze thesis werd een nieuw educatief programma besproken. Dit programma werd ontworpen om de gebruiker beter vertrouwd te maken met symmetrische cryptografische operaties. De gebruiker kan symmetrische cryptografische algoritmes tekenen in de vorm van een operatienetwerk. De verschillende elementen in het netwerk kunnen met de muis verbonden worden, de gebruikte operaties kunnen worden ingesteld. Vervolgens kunnen berekeningen doorheen het operatienetwerk worden uitgevoerd. Het programma werd op objectgerichte wijze ontworpen. Om de grafische gebruikersinterface te implementeren werd Qt gebruikt, een toolkit die voortdurend wordt uitgebreid. Dit alles maakt het programma geschikt voor eventuele uitbreidingen.

In het eerste deel werd vooral aandacht besteed aan het ontwerp van de interne structuur van een operatienetwerk. Er was aandacht voor het ontwerpproces en het uiteindelijke ontwerp werd verduidelijkt met behulp van objectdiagrammen. Operatienetwerken kunnen worden ingekapseld in zogenaamde zwarte dozen. Er werd extra aandacht besteed aan het algoritme om berekeningen doorheen een operatienetwerk uit te voeren.

Vervolgens werd de functionaliteit van de grafische gebruikersinterface besproken. Er werd uitgelegd hoe operatienetwerken en operaties worden opgeslagen en welke interfaces hiervoor gebruikt worden. Daarna werd uitvoerig ingegaan op de functionaliteit in het werkblad. Er werd uitgelegd hoe de gebruiker een hiërarchie van operatienetwerken kan creëren, door gebruik te maken van ingekapselde operatienetwerken. Enkele complexe dialoogvensters werden besproken. Ter illustratie van de mogelijkheden van het programma zijn enkele cryptografische algoritmes standaard in het programma aanwezig. De gebruiker kan de structuur van deze algoritmes bestuderen. Er zijn eveneens een aantal operaties standaard aanwezig, bijvoorbeeld de S-boxen van DES.

In het laatste deel werd ingegaan op enkele mogelijke uitbreidingen van het programma. Deze uitbreidingen zouden het programma nog interessanter maken voor gebruik tijdens oefenzitingen over symmetrische cijfers. Er was helaas geen tijd om deze uitbreidingen te implementeren. Het invoeren van geheugenelementen, alsook het definiëren van een bestandsformaat om een verzameling bitwoorden bij te houden kwam aan bod. Daarnaast

werd een uitbreiding besproken die het mogelijk moet maken differentiële berekeningen uit te voeren en om te controleren of een bepaald ingangspaar een differentieel pad al dan niet volgt. Tot slot werd een uitbreiding voorgesteld waarmee men op basis van operatienetwerken volautomatisch implementaties kan genereren in programmeertalen als C en VHDL.

Bibliografie

[1] The International PGP Home Page. PGPi Home. [Online]. Available: http://www.pgpi.org/

[2] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," September 1999, document version 2. [Online]. Available: http://citeseer.ist.psu.edu/daemen98aes.html

[3] Online Reference Documentation Qt 4.3. Trolltech. [Online]. Available: http://doc.trolltech.com/4.3/

[4] Trolltech homepage. Trolltech. [Online]. Available: http://trolltech.com

[5] The Designer's Guide to VHDL. Doulos. Last checked: 2008-05-10. [Online]. Available: http://www.doulos.com/knowhow/vhdl_designers_guide/

[6] W. Stallings, *Cryptography and network security: principles and practice*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1999.

[7] N. Smart, *Cryptography, An Introduction*. UK, Maidenhead: McGraw-Hill Education, January 2004.

[8] D. Z. Choudhury. (1999) Basic Concepts of Object Oriented Programming. [Online]. Available: http://homepages.north.londonmet.ac.uk/~chalkp/proj/ootutor/oopconcepts.html

[9] E. Steegmans and J. Dockx, *Objectgericht programmeren met Java*. Belgium, Leuven: Acco, 2000. [Online]. Available: http://www.doulos.com/knowhow/vhdl_designers_guide/

[10] The Source for Java Developers. Sun Microsystems. [Online]. Available: http://java.sun.com/

[11] L. Ammeraal, *Basiscursus C++*. The Netherlands, Schoonhoven: Academic Service.

[12] (2008) The Graphics View Framework. Trolltech. Last checked: 2008-05-10. [Online]. Available: http://doc.trolltech.com/4.3/graphicsview.html

[13] C++: Reference: STL Containers: bitset. cplusplus.com. Last checked: 2008-05-10. [Online]. Available: http://www.cplusplus.com/reference/stl/bitset/

[14] Boost C++ libraries: dynamic_bitset. Boost.org. Last checked: 2008-05-10. [Online]. Available: http://www.boost.org/doc/libs/1_35_0/libs/dynamic_bitset/dynamic_bitset.html

[15] W. Stallings, *Cryptografie en Netwerkbeveiliging, Beginselen en Praktijk*. The Netherlands, Schoonhoven: Academic Service, 2002, sec. 3.3 De Data Encryption Standard, pp. 73–83, translation from [6].

[16] *Data Encryption Standard*, National Institute of Standards and Technology, Federal Information Processing Standards Publication 46, January 1977.

[17] E. W. Weisstein. Finite Field. MathWorld. [Online]. Available: http://mathworld.wolfram.com/FiniteField.html

[18] *Specifications for the Advanced Encryption Standard*, National Institute of Standards and Technology, Federal Information Processing Standards Publication 197, November 2001. [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[19] *Information technology, Security techniques, Hash-functions, Part 3: Dedicated hash-functions*, International Organization for Standardization and International Electrotechnical Commission International standard final draft 10118-3, Rev. 2, 2004. [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43181

[20] A. Partow. (2006, January) Galois Field Arithmetic Library. Last checked: 2008-05-10. [Online]. Available: http://www.partow.net/projects/galois/

[21] J. Plank, "Fast Galois Field Arithmetic Library in C/C++," University of Tennessee, Tech. Rep. CS-07-593, April 2007. [Online]. Available: http://www.cs.utk.edu/~plank/plank/papers/CS-07-593

[22] Visual C# Developer Center. Microsoft. [Online]. Available: http://msdn2.microsoft.com/en-us/vcsharp/default.aspx

[23] (2008) QGraphicsScene Class Reference. Trolltech. Last checked: 2008-05-10. [Online]. Available: http://doc.trolltech.com/4.3/qgraphicsscene.html

[24] (2008) Diagram Scene Example. Trolltech. Last checked: 2008-05-10. [Online]. Available: http://doc.trolltech.com/4.3/graphicsview-diagramscene.html

[25] (2008) QGraphicsItem Class Reference. Trolltech. Last checked: 2008-05-10. [Online]. Available: http://doc.trolltech.com/4.3/qgraphicsitem.html

[26] XML Tutorial. W3Schools. Last checked: 2008-05-10. [Online]. Available: http://www.w3schools.com/xml/default.asp

[27] XML DOM Tutorial. W3Schools. Last checked: 2008-05-10. [Online]. Available: http://www.w3schools.com/dom/default.asp

[28] About SAX. SourceForge.net. [Online]. Available: http://www.saxproject.org/

[29] E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," in *Advances in Cryptology – CRYPTO '90 Proceedings*, vol. 537. Springer-Verlag, 1991, pp. 2–21.

[30] H. M. Heys, "A Tutorial on Linear and Differential Cryptanalysis," Memorial University of Newfoundland, pp. 3–5 and 19–29. [Online]. Available: http://www.engr.mun.ca/~howard/PAPERS/ldc_tutorial.ps

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

[32] W. Stallings, *Cryptografie en Netwerkbeveiliging, Beginselen en Praktijk*. The Netherlands, Schoonhoven: Academic Service, 2002, sec. 3.2 Principes van blokvercijfering, pp. 63–73, translation from [6].

[33] E. Zabala. (2004) Rijndael cipher 128 bit version, encryption (tutorial). Last checked: 2008-05-10. [Online]. Available: http://www.cs.bc.edu/~straubin/cs381-05/blockciphers/rijndael_ingles2004.swf

[34] *Specifications for the Secure Hash Standard*, Federal Information Processing Standards Publication 180-2, August 2002.

[35] X. Wang, Y. L. Yin, and H. Yu, "Finding Collisions in the Full SHA-1," in *Proceedings of the 25th Annual International Cryptology Conference on Advances in Cryptology*, vol. 3621. Springer-Verlag, August 2005, pp. 17–36.

[36] C. De Cannière and C. Rechberger, "Finding SHA-1 Characteristics: General Results and Applications," in *Advances in Cryptology – ASIACRYPT 2006*, vol. 4284. Germany, Berlin-Heidelberg: Springer, 2006, pp. 1–20.

[37] R. Merkle, "A certified digital signature," in *Advances in Cryptology – CRYPTO '89 Proceedings*, vol. 435. Springer-Verlag, 1989, pp. 218–238.

[38] I. Damgård, "A design principle for hash functions," in *Advances in Cryptology – CRYPTO '89 Proceedings*, vol. 435. Springer-Verlag, 1989, pp. 416–427.

[39] One iteration within the SHA-1 compression function. Wikipedia. [Online]. Available: http://en.wikipedia.org/wiki/Image:SHA-1.svg

[40] P. S. L. M. Barreto. The WHIRLPOOL Hash Function. Last checked: 2008-05-10. [Online]. Available: http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html

[41] P. S. L. M. Barreto and V. Rijmen, "The WHIRLPOOL Hashing Function," May 2003. [Online]. Available: http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html

[42] A. Burr, *Modulation and Coding for Wireless Communications*. Great Britain, Edinburgh Gate, Harlow, Essex: Pearson Education Limited, 2001, ch. 7.

Bijlage A

Opbouw van een netwerk: codefragment

Deze bijlage bevat een codefragment. Deze code bouwt het operatienetwerk dat besproken wordt in sectie 3.4 op.

```
// Maak de hoofdscene aan
ScenePointerCollection * hoofdScene = new ScenePointerCollection();

// Maak twee nieuwe constante ingangen van 16 bits
SingleConstantInputNode * inA = new SingleConstantInputNode(QString("inA"),16);
SingleConstantInputNode * inB = new SingleVariableInputNode(QString("inB"),16);
// Initialiseer de ingangen
inA->getInputBitWord()->setBitWordPart(0,0x0000300F);
inB->getInputBitWord()->setBitWordPart(0,0x0000111A);
// Voeg beide ingangen toe aan de hoofdscene
inA->connectToScenePointerCollection(mainScene);
inB->connectToScenePointerCollection(mainScene);

// Maak een samengesteldblokje aan met 1 uitgang en 1 ingang
ComposedNode * blokS = new ComposedNode(QString("blokS"),2,1);

ScenePointerCollection * enkapScene = composedNode->getComposedScenePointerCollection();

// Maak een operatie optelling
Operation * opTelOp = new OperationADD();
// Wijs de operatie toe aan een operatieblok, dit blok heeft 1 ingang
SingleOperationNode * optelling = new SingleOperationNode(QString("optelling"),opTelOp,2);
// Voeg het operatieblokje toe aan de hoofdscene
```

```
optelling->connectToScenePointerCollection(enkapScene);

// Vraag de I/O in- en uitgangen van de geëncapsuleerde scene op
SingleRealInputNode * in0 = encapsulatedScene->getRealInputNode(0);
SingleRealInputNode * in1 = encapsulatedScene->getRealInputNode(1);
SingleRealOutputNode * uit = encapsulatedScene->getRealOutputNode(0);

//Leg de verbindingen in de geëncapsuleerde scene
in0->getOutputConnection(0)->connectToEndNode(optelling,0);
in1->getOutputConnection(0)->connectToEndNode(optelling,1);
optelling->getOutputConnection(0)->connectToEndNode(uit,0);

// Maak een operatie substitutie-box
Operation * opSBox = new OperationBOX();
// Wijs de operatie toe aan een operatieblok, dit blok heeft 1 ingang
SingleOperationNode * sBox = new SingleOperationNode(QString("sBox"),opSBox,1);

// Maak een normaal observatie-uitgangsblok, met 0 uitgangen
SingleWatchOutputNode * kijk = new SingleWatchOutputNode(QString("kijk"),0);
// Voeg het blok toe aan de hoofdscene
kijk->connectToScenePointerCollection(hoofdScene);

//Leg de verbindingen in de hoofdscene
inA->getOutputConnection(0)->connectToEndNode(blokS,0);
inB->getOutputConnection(0)->connectToEndNode(blokS,1);
composedNode->getOutputConnection(0)->connectToEndNode(sBox,0);
sBox->getOutputConnection(0)->connectToEndNode(kijk,0);
```

Bijlage B

Operaties in GF(2^q) met behulp van tabellen

In deze bijlage wordt uitgelegd hoe opzoekingstabellen gebruikt kunnen worden om bepaalde operaties in een Galois veld van karakteristiek twee uit te voeren. Enkel de multiplicatieve operaties worden besproken, met name de vermenigvuldiging, de deling en de berekening van een inverse. In sectie B.1 gaat het over de veeltermrepresentatie van een element in een Galois veld. In de tweede sectie, sectie B.2, wordt de vermenigvuldiging gedefinieerd gebruikmakende van binaire veeltermen. Vervolgens wordt in sectie B.3 uitgelegd hoe de genoemde bewerkingen met behulp van twee tabellen kunnen gebeuren.

B.1 Veeltermrepresentatie in GF(2^q)

Uitleg over Galois velden kan men vinden in [21]. Een Galois veld van de vorm GF(2^q) telt 2^q elementen. Elk element kan worden voorgesteld door een decimaal getal in het bereik $[0, 2^q - 1]$. Handiger is deze voorstelling te transformeren naar een veelterm met binaire coëfficiënten. Er wordt vertrokken van de decimale voorstelling. Daarna wordt deze omgezet naar een binaire voorstelling in q bits. Deze bits komen overeen met de coëfficiënten van een veelterm van graad $q - 1$. De meest significante bit hoort bij de hoogste graadsterm, de minst significante bit is de constante in de veelterm. Dit wordt hieronder geïllustreerd aan de hand van het element 73 in een Galois veld met 256 elementen.

$$a = (73)_{10} = (01001001)_2 \rightarrow x^6 + x^3 + 1$$

B.2 Multiplicatieve berekeningen in GF(2^q)

Om multiplicatieve berekeningen te doen in een Galois veld van de vorm GF(2^q), dient een irreduceerbare veelterm ingevoerd te worden. Deze veelterm bevat louter binaire coëfficiënten en

is van graad q. In het AES-algoritme [18] wordt een Galois veld met 256 elementen gebruikt. De irreduceerbare veelterm is er $x^8+x^4+x^3+x+1$. Om twee elementen te vermenigvuldigen dient men eerst de twee overeenkomstige veeltermen te vermenigvuldigen. Men bekomt een veelterm van graad kleiner of gelijk aan $2*(q-1)$. Het resultaat van de vermenigvuldiging kan men bekomen door deze veelterm euclidisch te delen door de irreduceerbare veelterm; de rest van deze laatste deling is het resultaat van de vermenigvulding. Dit wordt hieronder geïllustreerd aan de hand van de vermenigvuldiging van de elementen 73 en 10 in het Rijndael-veld.

$$\begin{aligned} a &= (73)_{10} = (01001001)_2 \rightarrow x^6+x^3+1 \\ b &= (10)_{10} = (00001010)_2 \rightarrow x^3+x \end{aligned}$$

$$\begin{aligned} a\,b &= (x^6+x^3+1)\,(x^3+x) \mod x^8+x^4+x^3+x+1 \\ &= x^9+x^7+x^6+x^4+x^3+x \mod x^8+x^4+x^3+x+1 \\ &= x*(x^8)+x^7+x^6+x^4+x^3+x \mod x^8+x^4+x^3+x+1 \\ &= x*(x^4+x^3+x+1)+x^7+x^6+x^4+x^3+x \mod x^8+x^4+x^3+x+1 \\ &= x^5+x^4+x^2+x+x^7+x^6+x^4+x^3+x \mod x^8+x^4+x^3+x+1 \\ &= x^7+x^6+x^3+x^2+x \end{aligned}$$

$$a\,b = x^7+x^6+x^3+x^2+x \rightarrow (11001110)_2 = (206)_{10}$$

Indien men twee elementen a en b wil delen door elkaar, dient men de overeenkomstige polynomen euclidisch te delen in het Galois veld. De rest van de deling is het eindresultaat. Het berekenen van de inverse van een getal b, kan door 1 en b te delen door elkaar; 1 is het neutrale element van de vermenigvuldiging.

B.3 Opzoekingstabellen voor multiplicatieve berekeningen in GF(2^q)

Multiplicatieve bewerkingen in een Galois veld kunnen heel snel gebeuren indien men over twee vooraf aangemaakte tabellen beschikt. In volgende paragrafen wordt ingegaan op de wiskundige achtergrond van deze tabellen.

Het gebruik van tabellen steunt op de aanwezigheid van een multiplicatieve generator in het Galois veld. Elk element kan bijgevolg worden voorgesteld als een macht van deze generator.

De generator wordt voorgesteld door de letter g. In het Galois veld dat gebruikt wordt in AES (het Rijndael-veld) is 3 een generator. Voor elk element a bestaat dus een exponent e zodat:

$$a = g^e \mod x^8 + x^4 + x^3 + x + 1$$

Elk element a is dus volledig bepaald indien de exponent e gekend is. Het nulelement is een uitzondering, dit element kan niet worden voorgesteld als een macht, omdat het geen deel uitmaakt van de multiplicatieve groep. In de eerste opzoekingstabel kan men de exponent horende bij een element opzoeken, deze tabel wordt ook wel eens een logaritmetabel genoemd, omdat de tabel een element afbeeldt op het logaritme van dit element. In tabel B.1 bevindt zich een logaritmetabel van het Rijndael-veld. Als generator werd `0x03` gebruikt.

$$^g log(a) = {}^g log(g^e) = e$$

Een tweede opzoekingstabel wordt ook wel eens de exponenttabel genoemd en bevat de inverse afbeelding van de eerste tabel. Op basis van een exponent e kan het oorspronkelijke element g^e opgezocht worden. In tabel B.2 bevindt zich een exponenttabel van het Rijndael-veld. Als generator werd `0x03` gebruikt.

Beide opzoekingstabellen kunnen samen gebruikt worden om de vermenigvulding, de deling en de berekening van een inverse in een Galois veld uit te voeren. Dit gebeurt als volgt voor de vermenigvuldiging van twee niet-nul-elementen a en b.

$$a\,b = g^{e_a}\,g^{e_b} = g^{(logtabel[a]+logtabel[b]) \mod 2^q-1} = exptabel[(logtabel[a]+logtabel[b]) \mod 2^q-1]$$

Het delen van twee niet-nul-elementen a en b gebeurt als volgt:

$$\frac{a}{b} = g^{e_a}\,g^{-e_b} = g^{(logtabel[a]-logtabel[b])}$$

Indien de exponent van de generator in bovenstaande uitdrukking negatief is, moet deze vermeerderd worden met $2^q - 1$, namelijk de grootte van de multiplicatieve groep. Indien de exponent daarentegen groter dan of gelijk is aan $2^q - 1$, moet deze verminderd worden met $2^q - 1$. Vervolgens kan ook hier de exponenttabel gebruikt worden om het resultaat van de macht te bekomen. Indien een inverse dient berekend te worden, moet a gelijk aan 1 worden gesteld. Merk op dat $logtabel[1]$ altijd gelijk is aan 0, voor om het even welke generator.

Deze opzoekingstabellen hangen af van de irreduceerbare veelterm en de generator en bevatten elk $2^q - 1$ elementen. Deze methode kan alleen toegepast worden indien het veld niet te groot is, anders nemen de genoemde tabellen te veel geheugen in. Als generator wordt meestal een zo laag mogelijk getal gekozen, om het aanmaken van de tabellen niet nodeloos te vertragen. Om de logaritmetabel aan te maken, moeten immers alle machten van de generator berekend

worden. Indien de generatoren klein zijn, kan dit heel snel gebeuren. Indien de generator bijvoorbeeld twee is, moet men telkens vermenigvuldigen met twee. In een Galois veld van karakteristiek twee is dit een verschuiving, gevolgd door een conditionele XOR-operatie met de irreduceerbare veelterm van het veld. De exponenttabel kan eenvoudig bekomen worden door de logaritmetabel te inverteren.

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

Tabel B.1: Logaritmetabel van het Rijndael-veld, met generator `0x03` (hexadecimaal).

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

Tabel B.2: Exponenttabel van het Rijndael-veld, met generator `0x03` (hexadecimaal).

Bijlage C

Tabelvoorstelling van een substitutie-box

In deze bijlage wordt kort uitgelegd hoe een tabelvoorstelling van een substitutie-box geïnterpreteerd moet worden. Dit wordt gedaan aan de hand van de volgende substitutie-box. De waarden tussen haakjes zijn telkens de decimale voorstelling.

000 (0)	→	010 (2)
001 (1)	→	100 (4)
010 (2)	→	000 (0)
011 (3)	→	001 (1)
100 (4)	→	110 (6)
101 (5)	→	100 (4)
110 (6)	→	011 (3)
111 (7)	→	101 (5)

Deze opzoekingstabel van acht elementen kan bijvoorbeeld worden voorgesteld in een tabel met twee rijen en 4 kolommen:

	00	01	10	11
0	2	4	0	1
1	6	4	3	5

Om het resultaat van de substitutie-operatie te bekomen moet men de ingang binair voorstellen. De eerste bit van de ingang bepaalt de keuze van de rij in bovenstaande tabel. De twee volgende bits bepalen de kolom in bovenstaande tabel. Op deze manier bekomt één element in de tabel. Dit is het resultaat van de substitutie-operatie. Het binaire getal 100 bijvoorbeeld wordt afgebeeld op het element in de tabel dat zich bevindt in rij '1' en in kolom '00', het resultaat is 6 (=110).

Deze tabelvoorstelling is vooral handig bij substitutie-boxen met aan de ingang meerdere bits. Een S-box met 6 bits aan de ingang kan compact worden voorgesteld in een tabel met acht rijen en acht kolommen.

Bijlage D

Handleiding van het programma

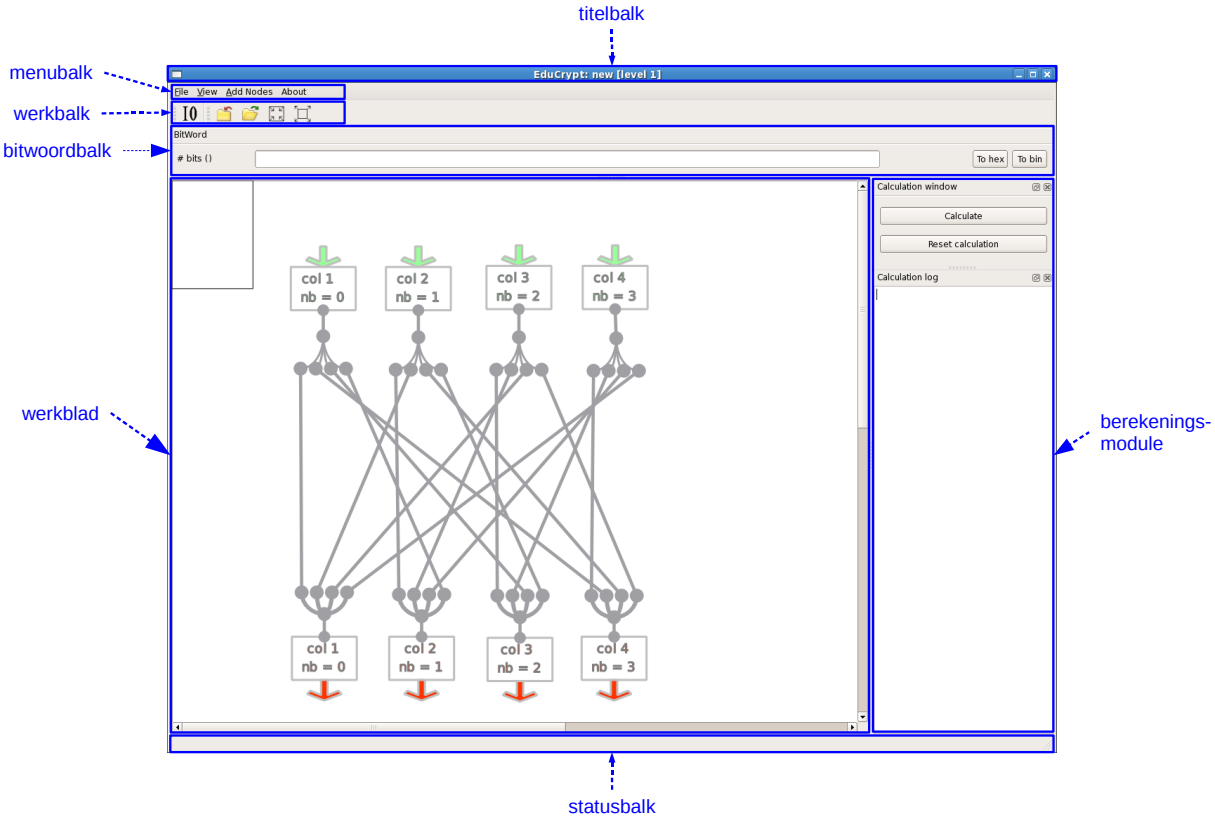
In deze bijlage bevindt zich de compacte handleiding van het programma. Deze handleiding maakt ook deel uit van het programma, ze staat onder de vorm van een pdf-bestand in de map van het uitvoeringsbestand. Er is zowel een Nederlandstalige als een Engelstalige versie. De Nederlandstalige versie wordt hieronder weergegeven.

Dit is de compacte handleiding van het programma EduCrypt. Deze handleiding volstaat om een gebruiker op weg te zetten. De handleiding is onderverdeeld in een aantal secties:

1. Beschrijving van het programma
2. Opbouwen van een operatienetwerk
3. Bediening van het programma
4. Opslaan en inladen
5. Auteursinformatie

D.1 Beschrijving van het programma

Dit programma biedt een grafische gebruikersinterface, waarmee de gebruiker om het even welk symmetrisch cijfer in de vorm van een operatienetwerk aan kan maken, opslaan en terug inladen. Het programma heeft een louter educatief doel en moet de gebruiker helpen bij het verwerven van inzicht in de symmetrische cryptografie. Een operatienetwerk bestaat uit blokken en verbindingen. Een verbinding is typisch een multi-bitlijn, onder de beschikbare blokken zijn er ingangen, uitgangen en operatieblokken. Het is tevens mogelijk om eigen blokkem aan te maken die gekoppeld zijn aan een ingekapseld operatienetwerk. De gebruiker kan een operatienetwerk gebruiken om berekeningen te maken. Bovendien kan de gebruiker alle tussenresultaten bestuderen. Indien het netwerk inconsistenties bevat, wordt de gebruiker hierover geïnformeerd. Het programma bevat hier en daar een kleine verwijzing naar de differentiële cryptanalyse, zo is het mogelijk om de verschil-distrubitietabel van een substitutie-box op te vragen.



Figuur D.1: Grafische gebruikerinterface van het programma EduCrypt.

In figuur D.1 wordt een screenshot van het programma weergegeven, de verschillende onderdelen zijn aangeduid in de figuur.

D.2 Opbouwen van een operatienetwerk

Centraal staat het werkblad. In het werkblad bevindt zich de scène waarin een operatienetwerk getekend kan worden. De gebruiker kan blokken toevoegen gebruikmakende van het menu 'Add Nodes'. De eigenschappen van elk blok kunnen veranderd worden door dubbel te klikken op een blok. Blokken worden verbonden met verbindingslijnen, die men kan verkrijgen door de uitgang van een blok te verbinden met de ingang van een ander blok. Dit kan door een sleepbewerking met de muis. Het aantal bits dat hoort bij een verbindingslijn wordt gedetermineerd door het startblok. De volgende blokken worden onderscheiden:

- Ingangsblokken: constante ingang (^{1}), variabele ingang en externe ingang (^{2})
- Uitgangsblokken: observatie-uitgang en externe uitgang (^{2})

- Operatieblokken, mogelijke operaties:
  - logische operaties: OR, NOR, AND, NAND, XOR, NOT
  - Galois operaties: vermenigvuldiging en de berekening van een inverse in een Galois veld
  - dummy operaties: samensmelting, splits op, kopie, aantal bits van de ingang, uitbreiding van een bitwoord tot een bepaald lengte, verbinding
  - overige operaties: S-box ([3]), bit-herverdeler ([3]), verschuiving, rotatie, optelling, vermenigvuldiging
- Samengestelde blokken: dit type blok is als een zwarte doos voor ingekapselde scènes ([4])

[1] De bits horende bij de constante ingang worden bewaard indien het document wordt opgeslagen, dit is niet zo bij variabele ingangen.

[2] Externe in- en uitgangen worden gebruikt in scènes die ingekapseld zijn in een samengesteld blok. Deze externe in- en uitgangen worden impliciet verbonden met de in- en uitgangen van dit samengesteld blok. Dit type in- en uitgangen hebben ook een index, opdat vast zou staan welke externe ingang/uitgang overeenkomt met welke ingang/uitgang van het samengestelde blok.

[3] Het dialoogvenster van een operatieblok met een S-box-operatie of een bit-herverdeel-operatie (dubbel klikken op het operatieblok) biedt de mogelijkheid een willekeurige S-box/bit-herverdeler in te voeren, alsook om een operatie uit een extern XML-bestand in te laden.

[4] In het dialoogvenster van een samengesteld blok (dubbel klikken op het samengesteld blok) kan men de externe in- en uitgangen van het samengesteld blok beheren. De gebruiker kan er ook scènes mee inladen, bijvoorbeeld een AES-128-bit-encrpytie. De inhoud van de ingekapselde scène kan vergrendeld worden door de variabele 'locked' bovenaan in het dialoogvenster in te schakelen.

D.3 Bediening van het programma

In het werkblad bevindt zich een scène. Deze scène kan vergroot en verkleind worden met de werkbalkknoppen 4 en 5. De externe in- en uitgangen van de momenteel zichtbare scène kunnen beheerd worden met werkbalkknop 1. Het invoegen van blokken gebeurt met het menu 'Add Nodes'. De eigenschappen van een blok kunnen gewijzigd worden door dubbel te klikken op het blok. Het verwijderen van een blok gebeurt door het blok te selecteren en vervolgens de delete-toets in te drukken. Verschillende blokken kunnen tegelijkertijd geselecteerd worden.

Bitwoorden horende bij ingangen kunnen worden ingevoerd in de bitwoordbalk, dit kan hexadecimaal of binair, beide notaties mogen door elkaar gebruikt worden. Door met de rechter muisknop te klikken op een ingangsblok verschijnt het huidige bitwoord in de bitwoordbalk.

Dit bitwoord kan worden aangepast, de aanpassing wordt bevestigd door de enter-toets in te drukken.

Alle blokken kunnen vrij met de muis worden verplaatst. Twee blokken kunnen niet overlappen. Indien de gebruiker een blok wil kopiëren in plaats van te verplaatsen dient hij de control-toets in gedrukt te houden net voor hij de muisknop loslaat. Het verplaatsen van blokkken kan ook door een blok te selecteren en vervolgens de pijltoetsen te gebruiken, al dan niet in combinatie met de control-toets. Operaties horende bij een operatieblok kunnen gekopieerd worden door 'control + c' in te drukken, het plakken in een ander operatieblok gebeurt met 'control + v'.

De eigenschappen van elk blok kunnen opgevraagd worden door dubbel te klikken op het blok. Het aantal ingangen van de logische operaties OR, NOR, AND, NAND en EXOR kan veranderd worden met de toetsen '+' en '-'.

Het verbinden van blokken gebeurt door te klikken op de uitgang van een blok, de muis te verslepen naar de ingang van een ander blok en daar de muis los te laten. Elke uitgang kan slechts één verbinding hebben, aftakkingen kunnen gecreëerd worden met een kopie-operatie-blok. Men kan een niet-rechte verbinding leggen indien men gebruik maakt van verbindingsblokken. Deze worden automatisch gecreëerd indien de gebruiker klikt op een uitgang, de muis sleept naar een vrij plaats en daar de muisknop loslaat terwijl hij de control-toets ingedrukt houdt.

De scène ingekapseld in een samengesteld blok komt tevoorschijn in het werkblad indien men het blok selecteert en vervolgens werkbalkknop 3 indrukt. Terugkeren naar het hogere (vorige) niveau kan door werkbalkknop 2 in te drukken.

Operatieblokken en samengestelde blokken kunnen gelinkt worden. Dit wil zeggen dat de inhoud van deze blokken altijd hetzelfde blijft. Indien drie AND-operaties gelinkt zijn en één operatie verandert in een OR-operatie, veranderen de andere operaties mee. Hetzelfde geldt voor de ingekapselde scènes van gelinkte samengestelde blokken. Blokken kunnen gekopieerd worden zodat de kopie gelinkt is met het origineel, dit kan door de toetsen combinatie 'control + alt' in te drukken tijdens het loslaten van de muisknop. Het verwijderen van een link kan door het blok te selecteren en de toets 'l' in te drukken. Twee reeds bestaande blokken kunnen gelinkt worden door 'control + l' in te drukken, en vervolgens het te linken blok te selecteren en 'control + v' in te drukken. Operatieblokken kunnen alleen met operatieblokken gelinkt worden, samengestelde blokken alleen met samengestelde blokken.

Berekeningen uitvoeren kan met de knop 'Calculate' in de berekeningsmodule. In de log van de berekeningsmodule verschijnt feedback over het berekeningsproces. Het berekeningsproces stuurt respectievelijk alle ingangen aan en de berekeningen sijpelen telkens zo ver mogelijk door naar de uitgangen. Om de resultaten van een berekeningsproces te bekijken kan de gebruiker rechts klikken op alle verbindingslijnen of op een uitgangsblok. Telkens verschijnt het overeenkomstige bitwoord in de bitwoordbalk.

D.4 Opslaan en inladen

Gebruikmakend van het menu 'File' kan de gebruiker documenten inladen of opslaan vanuit of naar een XML-document. Het is tevens mogelijk de huidig zichtbare scène, of een deel ervan,

op te slaan in een PNG-beeldbestand.

In het programma zijn een aantal scènes, substitutie-boxen en bit-herverdelers reeds standaard aanwezig. De gebruiker kan deze gebruiken door in het overeenkomstige dialoogvenster de 'Load standard' knop aan te klikken. Daarna verschijnt telkens een lijst met de beschikbare scènes of substitutie-boxen. Aan de hand van deze dialoogvenster kunnen ook niet-standaard scènes of substitutie-boxen worden weggeschreven of ingeladen.

D.5 Auteursinformatie

Het programma EduCrypt werd gemaakt door Igor Jacques, in het kader van een masterproef aan de Katholieke Universiteit Leuven.

Bijlage E

XML-documenten

E.1 XML-document van een substitutie-box

In deze bijlage wordt een voorbeeld gegeven van een XML-document van een S-box. De S-Box uit bijlage C wordt gebruikt.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE empty>
<OPERATION operationName="BOX" fileName="sboxvoorbeeld.xml"
name="voorbeeld substitutie-box" standard="false" description="">
    <data-information permutation="false">
        <nrRowBits>1</nrRowBits>
        <nrColumnBits>2</nrColumnBits>
        <nrOutputBits>3</nrOutputBits>
    </data-information>
    <data>
        <row>
            <column>2</column>
            <column>4</column>
            <column>0</column>
            <column>1</column>
        </row>
        <row>
            <column>6</column>
            <column>7</column>
            <column>4</column>
            <column>5</column>
        </row>
    </data>
</OPERATION>
```

E.2 XML-document van een operatienetwerk

In deze bijlage wordt een voorbeeld gegeven van een XML-documenten van een scène. De sscène in dit XML-document worden gebruikt in het AES-encryptie-algoritme. Ze bevat het operatienetwerk van één encryptie-ronde. Een grafische voorstelling van een ronde in het AES-algoritme zoals weergegeven in het programma, alsook meer uitleg over de verschillende stappen in het AES-encryptie-algoritme, is te vinden in het onderdeel 8.2.1.

Hieronder bevindt zich het XML-document dat hoort bij het operatienetwerk van één AES-encryptie-ronde, namelijk `AES_128_round.xml`. Zoals wordt uitgelegd in sectie 6.4.1, bestaat het XML-document uit vijf delen. Een eerste deel bevat algemene informatie over de sscène, het tweede deel bevat alle blokken, het derde deel bevat alle connecties, in het vierde deel worden alle operaties gedefinieerd en in het vijfde en laatste deel wordt verwezen naar de XML-documenten van de ingekapselde scènes.

```
<scene fileName="AES 128 round.xml" >
    <scene-information>
        <standard>true</standard>
        <locked>true</locked>
        <advisedName>AES 128 round</advisedName>
        <description></description>
        <size>
            <width>700</width>
            <height>700</height>
        </size>
    </scene-information>
    <nodes>
        <node>
            <nodeType nodeTypeName="COMPOSEDNODE" >1</nodeType>
            <name>subbytes</name>
            <nrConnections>
                <nrInputs>1</nrInputs>
                <nrOutputs>1</nrOutputs>
            </nrConnections>
            <nodeNr>15649</nodeNr>
            <position>
                <x>151</x>
                <y>128</y>
            </position>
            <scenepointercollectionnumber>15649</scenepointercollectionnumber>
        </node>
        <node>
            <nodeType nodeTypeName="COMPOSEDNODE" >1</nodeType>
            <name>subbytes</name>
            <nrConnections>
                <nrInputs>1</nrInputs>
```

```
            <nrOutputs>1</nrOutputs>
        </nrConnections>
        <nodeNr>15658</nodeNr>
        <position>
            <x>245</x>
            <y>127</y>
        </position>
        <scenepointercollectionnumber>15649</scenepointercollectionnumber>
    </node>
    <node>
        <nodeType nodeTypeName="COMPOSEDNODE" >1</nodeType>
        <name>subbytes</name>
        <nrConnections>
            <nrInputs>1</nrInputs>
            <nrOutputs>1</nrOutputs>
        </nrConnections>
        <nodeNr>15667</nodeNr>
        <position>
            <x>344</x>
            <y>127</y>
        </position>
        <scenepointercollectionnumber>15649</scenepointercollectionnumber>
    </node>
    <node>
        <nodeType nodeTypeName="COMPOSEDNODE" >1</nodeType>
        <name>subbytes</name>
        <nrConnections>
            <nrInputs>1</nrInputs>
            <nrOutputs>1</nrOutputs>
        </nrConnections>
        <nodeNr>15676</nodeNr>
        <position>
            <x>438</x>
            <y>127</y>
        </position>
        <scenepointercollectionnumber>15649</scenepointercollectionnumber>
    </node>
    <node>
        <nodeType nodeTypeName="COMPOSEDNODE" >1</nodeType>
        <name>shiftrow</name>
        <nrConnections>
            <nrInputs>4</nrInputs>
            <nrOutputs>4</nrOutputs>
        </nrConnections>
        <nodeNr>15685</nodeNr>
        <position>
```

```
            <x>298</x>
            <y>211</y>
        </position>
        <scenepointercollectionnumber>15685</scenepointercollectionnumber>
    </node>
    <node>
        <nodeType nodeTypeName="SINGLEOPERATIONNODE" >3</nodeType>
        <name>MELT</name>
        <nrConnections>
            <nrInputs>4</nrInputs>
            <nrOutputs>1</nrOutputs>
        </nrConnections>
        <nodeNr>15702</nodeNr>
        <position>
            <x>297</x>
            <y>348</y>
        </position>
        <operationnumber>15702</operationnumber>
    </node>
    <node>
        <nodeType nodeTypeName="SINGLEOPERATIONNODE" >3</nodeType>
        <name>xOR</name>
        <nrConnections>
            <nrInputs>2</nrInputs>
            <nrOutputs>1</nrOutputs>
        </nrConnections>
        <nodeNr>15703</nodeNr>
        <position>
            <x>346</x>
            <y>415</y>
        </position>
        <operationnumber>15703</operationnumber>
    </node>
    <node>
        <nodeType nodeTypeName="SINGLEOPERATIONNODE" >3</nodeType>
        <name>SPLIT</name>
        <nrConnections>
            <nrInputs>1</nrInputs>
            <nrOutputs>4</nrOutputs>
        </nrConnections>
        <nodeNr>15704</nodeNr>
        <position>
            <x>346</x>
            <y>467</y>
        </position>
        <operationnumber>15704</operationnumber>
```

```
</node>
<node>
    <nodeType nodeTypeName="COMPOSEDNODE" >1</nodeType>
    <name>mix cols</name>
    <nrConnections>
        <nrInputs>4</nrInputs>
        <nrOutputs>4</nrOutputs>
    </nrConnections>
    <nodeNr>15705</nodeNr>
    <position>
        <x>298</x>
        <y>284</y>
    </position>
    <scenepointercollectionnumber>15705</scenepointercollectionnumber>
</node>
<node>
    <nodeType nodeTypeName="SINGLEREALINPUTNODE" >6</nodeType>
    <name> col 1</name>
    <nrConnections>
        <nrInputs>0</nrInputs>
        <nrOutputs>1</nrOutputs>
    </nrConnections>
    <nodeNr>15894</nodeNr>
    <realInputNr>0</realInputNr>
    <position>
        <x>151</x>
        <y>66</y>
    </position>
</node>
<node>
    <nodeType nodeTypeName="SINGLEREALINPUTNODE" >6</nodeType>
    <name> col 2</name>
    <nrConnections>
        <nrInputs>0</nrInputs>
        <nrOutputs>1</nrOutputs>
    </nrConnections>
    <nodeNr>15895</nodeNr>
    <realInputNr>1</realInputNr>
    <position>
        <x>245</x>
        <y>65</y>
    </position>
</node>
<node>
    <nodeType nodeTypeName="SINGLEREALINPUTNODE" >6</nodeType>
    <name> col 3</name>
```

```
        <nrConnections>
            <nrInputs>0</nrInputs>
            <nrOutputs>1</nrOutputs>
        </nrConnections>
        <nodeNr>15896</nodeNr>
        <realInputNr>2</realInputNr>
        <position>
            <x>344</x>
            <y>64</y>
        </position>
    </node>
    <node>
        <nodeType nodeTypeName="SINGLEREALINPUTNODE" >6</nodeType>
        <name> col 4</name>
        <nrConnections>
            <nrInputs>0</nrInputs>
            <nrOutputs>1</nrOutputs>
        </nrConnections>
        <nodeNr>15897</nodeNr>
        <realInputNr>3</realInputNr>
        <position>
            <x>438</x>
            <y>64</y>
        </position>
    </node>
    <node>
        <nodeType nodeTypeName="SINGLEREALINPUTNODE" >6</nodeType>
        <name>  key</name>
        <nrConnections>
            <nrInputs>0</nrInputs>
            <nrOutputs>1</nrOutputs>
        </nrConnections>
        <nodeNr>15898</nodeNr>
        <realInputNr>4</realInputNr>
        <position>
            <x>432</x>
            <y>319</y>
        </position>
    </node>
    <node>
        <nodeType nodeTypeName="SINGLEREALOUTPUTNODE" >7</nodeType>
        <name>col 1</name>
        <nrConnections>
            <nrInputs>1</nrInputs>
            <nrOutputs>0</nrOutputs>
        </nrConnections>
```

```
    <nodeNr>15899</nodeNr>
    <realOutputNr>0</realOutputNr>
    <position>
        <x>237</x>
        <y>541</y>
    </position>
</node>
<node>
    <nodeType nodeTypeName="SINGLEREALOUTPUTNODE" >7</nodeType>
    <name> col 2</name>
    <nrConnections>
        <nrInputs>1</nrInputs>
        <nrOutputs>0</nrOutputs>
    </nrConnections>
    <nodeNr>15900</nodeNr>
    <realOutputNr>1</realOutputNr>
    <position>
        <x>313</x>
        <y>541</y>
    </position>
</node>
<node>
    <nodeType nodeTypeName="SINGLEREALOUTPUTNODE" >7</nodeType>
    <name> col 3</name>
    <nrConnections>
        <nrInputs>1</nrInputs>
        <nrOutputs>0</nrOutputs>
    </nrConnections>
    <nodeNr>15901</nodeNr>
    <realOutputNr>2</realOutputNr>
    <position>
        <x>382</x>
        <y>541</y>
    </position>
</node>
<node>
    <nodeType nodeTypeName="SINGLEREALOUTPUTNODE" >7</nodeType>
    <name> col 4</name>
    <nrConnections>
        <nrInputs>1</nrInputs>
        <nrOutputs>0</nrOutputs>
    </nrConnections>
    <nodeNr>15902</nodeNr>
    <realOutputNr>3</realOutputNr>
    <position>
        <x>458</x>
```

```
            <y>542</y>
        </position>
    </node>
</nodes>
<connections>
    <connection>
        <startNodeNumber startIndex="0" >15649</startNodeNumber>
        <endNodeNumber endIndex="0" >15685</endNodeNumber>
    </connection>
    <connection>
        <startNodeNumber startIndex="0" >15658</startNodeNumber>
        <endNodeNumber endIndex="1" >15685</endNodeNumber>
    </connection>
    <connection>
        <startNodeNumber startIndex="0" >15667</startNodeNumber>
        <endNodeNumber endIndex="2" >15685</endNodeNumber>
    </connection>
    <connection>
        <startNodeNumber startIndex="0" >15676</startNodeNumber>
        <endNodeNumber endIndex="3" >15685</endNodeNumber>
    </connection>
    <connection>
        <startNodeNumber startIndex="0" >15685</startNodeNumber>
        <endNodeNumber endIndex="0" >15705</endNodeNumber>
    </connection>
    <connection>
        <startNodeNumber startIndex="1" >15685</startNodeNumber>
        <endNodeNumber endIndex="1" >15705</endNodeNumber>
    </connection>
    <connection>
        <startNodeNumber startIndex="2" >15685</startNodeNumber>
        <endNodeNumber endIndex="2" >15705</endNodeNumber>
    </connection>
    <connection>
        <startNodeNumber startIndex="3" >15685</startNodeNumber>
        <endNodeNumber endIndex="3" >15705</endNodeNumber>
    </connection>
    <connection>
        <startNodeNumber startIndex="0" >15702</startNodeNumber>
        <endNodeNumber endIndex="1" >15703</endNodeNumber>
    </connection>
    <connection>
        <startNodeNumber startIndex="0" >15703</startNodeNumber>
        <endNodeNumber endIndex="0" >15704</endNodeNumber>
    </connection>
    <connection>
```

```
    <startNodeNumber startIndex="0" >15704</startNodeNumber>
    <endNodeNumber endIndex="0" >15899</endNodeNumber>
</connection>
<connection>
    <startNodeNumber startIndex="1" >15704</startNodeNumber>
    <endNodeNumber endIndex="0" >15900</endNodeNumber>
</connection>
<connection>
    <startNodeNumber startIndex="2" >15704</startNodeNumber>
    <endNodeNumber endIndex="0" >15901</endNodeNumber>
</connection>
<connection>
    <startNodeNumber startIndex="3" >15704</startNodeNumber>
    <endNodeNumber endIndex="0" >15902</endNodeNumber>
</connection>
<connection>
    <startNodeNumber startIndex="0" >15705</startNodeNumber>
    <endNodeNumber endIndex="0" >15702</endNodeNumber>
</connection>
<connection>
    <startNodeNumber startIndex="1" >15705</startNodeNumber>
    <endNodeNumber endIndex="1" >15702</endNodeNumber>
</connection>
<connection>
    <startNodeNumber startIndex="2" >15705</startNodeNumber>
    <endNodeNumber endIndex="2" >15702</endNodeNumber>
</connection>
<connection>
    <startNodeNumber startIndex="3" >15705</startNodeNumber>
    <endNodeNumber endIndex="3" >15702</endNodeNumber>
</connection>
<connection>
    <startNodeNumber startIndex="0" >15894</startNodeNumber>
    <endNodeNumber endIndex="0" >15649</endNodeNumber>
</connection>
<connection>
    <startNodeNumber startIndex="0" >15895</startNodeNumber>
    <endNodeNumber endIndex="0" >15658</endNodeNumber>
</connection>
<connection>
    <startNodeNumber startIndex="0" >15896</startNodeNumber>
    <endNodeNumber endIndex="0" >15667</endNodeNumber>
</connection>
<connection>
    <startNodeNumber startIndex="0" >15897</startNodeNumber>
    <endNodeNumber endIndex="0" >15676</endNodeNumber>
```

```
        </connection>
        <connection>
            <startNodeNumber startIndex="0" >15898</startNodeNumber>
            <endNodeNumber endIndex="0" >15703</endNodeNumber>
        </connection>
    </connections>
    <operations>
        <operation number="15702" >
            <standard>true</standard>
            <operationType>7</operationType>
        </operation>
        <operation number="15703" >
            <standard>true</standard>
            <operationType>12</operationType>
        </operation>
        <operation number="15704" >
            <standard>true</standard>
            <operationType>6</operationType>
            <nrBitPartitions>4</nrBitPartitions>
            <bitPartitions>
                <bitPartition>32</bitPartition>
                <bitPartition>32</bitPartition>
                <bitPartition>32</bitPartition>
                <bitPartition>32</bitPartition>
            </bitPartitions>
        </operation>
    </operations>
    <scenepointercollections>
        <scenepointercollection number="15649" >
            <standard>true</standard>
            <locked>true</locked>
            <fileName>AES subbytes 4 bytes.xml</fileName>
        </scenepointercollection>
        <scenepointercollection number="15685" >
            <standard>true</standard>
            <locked>true</locked>
            <fileName>AES 128 shiftrows.xml</fileName>
        </scenepointercollection>
        <scenepointercollection number="15705" >
            <standard>true</standard>
            <locked>true</locked>
            <fileName>AES 128 mixcolumns.xml</fileName>
        </scenepointercollection>
    </scenepointercollections>
</scene>
```