Elgadroid

Pim Hopmans

Bachelor Kunstmatige Intelligentie, Universiteit Utrecht
7,5 ECTS

Scriptiebegeleider:
Dr. Gerard Tel

Tweede
beoordelaar:
Dr. Frans Adriaans

9 mei 2017

Inhoudsopgave

1 Inleiding

De app Elgamal voor Android-smartphones is bedoeld ter demonstratie van het Elgamal-algoritme, ontworpen door Taher Elgamal in 1985.[1]

Met deze app kunnen leerlingen/studenten zelf ondervinden hoe het is om berichten, in de vorm van getallen, te versleutelen, te versturen, te ontvangen en te ontsleutelen. In deze app worden twee cryptografische algoritmes gedemonstreerd, namelijk Caesar-codering en Elgamal-codering. Caesar-codering is een simpel algoritme dat gebruik maakt van vermenigvuldiging en deling om berichten te versleutelen en te ontsleutelen. Elgamal-codering is een ingewikkelder algoritme dat gebruik maakt van een publieke en een geheime sleutel, vermenigvuldiging, deling en machtsverheffen. Verder kan de leerling/student met de app kort experimenteren hoe moeilijk het kan zijn om logaritmes en machten te vinden gegeven een getal.

De app kan gebruikt worden voor workshops, demonstraties op middelbare scholen en ter ondersteuning van colleges over cybersecurity. In dit artikel zal ik kort beschrijven wat de relevantie van zo een app is voor Kunstmatige Intelligentie.

2 Doel

In 2005 heeft Gerard Tel een applicatie geschreven voor de TI-83+ rekenmachine[2], dat bedoeld was om workshops te geven over cryptografie. Bij deze workshops konden de leerlingen kennis opdoen over cryptografie, met behulp van hun rekenmachine. Met de applicatie voor de TI-83+ kun je Caesar-codering, Elgamal-codering, machtsverheffen en twee algoritmes om de discrete logaritme uit te rekenen uitproberen. Dit is ook allemaal in de app Elgamal geprogrammeerd.

De app Elgamal is dan ook een nieuwere, modernere versie van het TI-83+ programma. Gerard Tel merkte op dat hij bij zijn colleges Security aan de Universiteit Utrecht expliciet moest vermelden dat de studenten het opvolgende college hun TI-rekenmachine mee moesten nemen. En aangezien de meerderheid van de studenten tegenwoordig een Android-telefoon heeft, is het een logische stap om een versie van het programma Elgamal te schrijven voor Androids.

Dus, het doel is om een app voor Android-telefoons te schrijven, die dezelfde functionaliteit heeft als het TI-83+ Elgamal-programma, zodat in plaats van het TI-83+ programma de app voor Android-telefoons gebruikt kan worden tijdens workshops/colleges/demonstraties, etc.

3 Studie

In de app moet dezelfde functionaliteit voorkomen als in het TI-83+ programma en hiervoor is het benodigde bestudeerd:

1. Caesar-codering.
2. Elgamal-codering.
3. Machtsverheffen (Exponentiation by Squaring).
4. Holemens Logaritme.
5. Pollard Rho-algoritme.

1 en 2 zijn beiden behandeld in de cursus Security van Gerard Tel. 3 tot en met 5 komt voornamelijk uit *Het programma* ***ELGAMAL*** door Gerard Tel.[3]

Daarnaast heb ik het dictaat van de cursus Mobiel Programmeren aan de Universiteit Utrecht, gegeven door Jeroen Fokker, doorgenomen om basiskennis op te doen van applicaties voor Androids programmeren.[4]

Hier volgt een beschrijving van de opgedane kennis over de algoritmes:

1. Caesar-codering is een simpel algoritme om berichten te versleutelen. Het wordt ook wel Caesarrotatie, of afgekort ROT, genoemd. Caesar-codering is een symmetrisch crypto-algoritme, wat inhoudt dat dezelfde sleutel voor het versleutelen als het ontsleutelen wordt gebruikt. Als gevolg hiervan, moet deze sleutel goed beschermd worden! Over het algemeen wordt dit algoritme gebruikt om woorden en zinnen te versleutelen, door de letters van het alfabet x plaatsen op te schuiven, waarbij x de sleutel is. Dus als je als sleutel het getal 3 gebruikt, dan krijg je de volgende substitutieregels:

   - $a \rightarrow d$
   - $b \rightarrow e$

     ...
   - $y \rightarrow b$
   - $z \rightarrow c$

   Merk op dat als de sleutel groter is dan 26, we de sleutel modulo 26 kunnen nemen om hetzelfde resultaat te krijgen! Het ontsleutelen van versleutelde berichten is het omgekeerde van versleutelen, dus kun je de substitutieregels omkeren en het originele bericht terug krijgen uit het versleutelde bericht.

   In het geval van de app die we gaan schrijven, gebruiken we een andere variant van Caesar-codering, namelijk de variant waarbij we getallen kunnen versleutelen. Hiervoor moeten we een getal als modulus afspreken. Dit is noodzakelijk omdat je met

een modulus het originele bericht niet terug kan rekenen zonder de sleutel te weten.

De versleutelprocedure van getallen gaat als volgt: Gegeven een sleutel z, een modulus m en een bericht x, kunnen we het cijferbericht y uitrekenen als:

$$y = x \cdot z \bmod m$$

Het ontsleutelen is wederom het omgekeerde: Gegeven de sleutel z, de modulus m en het cijferbericht y, kunnen we het originele bericht x uitrekenen als:

$$x = y/z \bmod m$$

Deze deling moet modulair worden uitgevoerd, dus gaan we de multiplicatieve inverse van z mod m gebruiken. Dit is mogelijk omdat volgens de Kleine Stelling van Fermat geldt dat $z^{m-1} = 1 \bmod m$. Hierbij moet m wel een priemgetal zijn. Als we dat aannemen, krijgen we als ontsleuteling:

$$x = y \cdot z^{m-2} \bmod m$$

Caesar-codering met letters is zeer onveilig. Als een aanvaller weet dat het bericht met Caesar-codering is gecodeerd, zou hij alle mogelijke verschuivingen van letters uit kunnen proberen en daar zelf het originele bericht uit kunnen herkennen. Caesar-codering met getallen is een stuk veiliger, mits de sleutel groot genoeg is en geheim gehouden is. Hierbij komt ook het voordeel dat als een aanvaller een brute-force aanval op het versleutelde bericht uitvoert, er bij elke mogelijke sleutel een getal uit komt. Zo kan de aanvaller nooit het originele bericht herkennen.

Het blijft echter een onveilige manier van encryptie. Uit een gelekt x, y-paar kan de gebruikte sleutel afgeleid worden, waardoor de aanvaller elk bericht tussen de twee corresponderende personen kan ontsleutelen, als zij geen andere sleutel afspreken.

2. Het crypto-algoritme Elgamal is een algoritme gebaseerd op de discrete logaritme in cyclische groepen. In tegenstelling tot Caesar-codering is Elgamal een asymmetrisch crypto-algoritme. Als twee personen met elkaar willen communiceren, hebben zij niet 1 gezamenlijke sleutel om te versleutelen en ontsleutelen, maar hebben ze allebei een eigen geheime en publieke sleutel. De geheime sleutel delen ze uiteraard met niemand en de publieke sleutel is openbaar voor iedereen. Als persoon A nu een bericht naar persoon B wilt sturen, versleutelt A zijn bericht met behulp van de publieke sleutel van B, waarna B het versleutelde bericht kan ontsleutelen met zijn eigen geheime sleutel.

   Voor het algoritme Elgamal zijn een paar variabelen, p, q en g, nodig. p wordt als modulus gebruikt, q als orde en g als generator. Aan deze variabelen zitten nog een paar eisen:

   - q moet een priemgetal zijn.
   - p moet een priemgetal zijn en $p - 1$ moet deelbaar zijn door q.
   - g is een willekeurig getal uit de orde q modulo p. Hier geldt de relatie $g^q = 1$. g mag niet gelijk zijn aan 1!

Nu we deze parameters hebben, kunnen we verder gaan met het versleutelen. Stel er zijn twee personen, Alice en Bob, die met elkaar willen communiceren. Eerst zullen zij ieder een geheime sleutel, a, moeten kiezen. a moet tussen 1 en $q-1$ liggen. Uit a wordt dan hun publieke sleutel b bepaald door $b = g^a$. Nu hebben Alice en Bob dus beiden een a en b.

Stel Bob wilt nu een bericht, x, sturen naar Alice. Dan versleutelt hij x als volgt:

- Kies een willekeurig getal k tussen 1 en $q-1$.
- Bereken $u = g^k$ en $v = x \cdot b_1^k$, waarbij b_1 de publieke sleutel van Alice is.
- Nu kan Bob het versleutelde bericht $y = (u, v)$ sturen naar Alice.

Als Alice het ontvangen bericht y van Bob nu wilt ontsleutelen, hoeft ze alleen maar te machtsverheffen met haar eigen geheime sleutel a, als volgt:

$$x = \frac{v}{u^{a_1}} = \frac{x \cdot b_1^k}{(g^k)^{a_1}} = \frac{x \cdot (g^{a_1})^k}{g^{k a_1}} = \frac{x \cdot g^{a_1 k}}{g^{a_1 k}}$$

Dit algoritme is veilig, omdat de geheime sleutel, a, niet terug gerekend kan worden uit b, mits de getallen groot genoeg zijn. Als een aanvaller dan het versleutelde bericht onderschept, zou hij het nooit terug kunnen vertalen naar het originele bericht zonder a. Er zijn een aantal algoritmen die de discrete logaritme uit kunnen rekenen, zoals het Holemens-algoritme en het Pollard Rho-algoritme, maar deze zijn niet efficiënt genoeg.

3. Machtsverheffen is een simpele wiskundige operatie. In g^x is g het grondtal en x de exponent, en wordt g x keer met zichzelf vermenigvuldigd. Bijvoorbeeld: $2^4 = 2 \cdot 2 \cdot 2 \cdot 2$. Dit algoritme heeft $x - 1$ vermenigvuldigingen nodig, terwijl het volgende algoritme een stuk sneller kan. Dit noemen we "Exponentiation by Squaring". Het werkt als volgt:

   Stel we hebben een machtsverheffing g^x. Als de exponent even is, dan wordt het gehalveerd en wordt het grondtal gekwadrateerd. Dit mag omdat:

   $$g^x = (g^2)^{\frac{x}{2}} = (g \cdot g)^{\frac{x}{2}} = g^{\frac{x}{2}} \cdot g^{\frac{x}{2}}.$$

   Als de exponent oneven is wordt de exponent met 1 verminderd, en wordt dat met het grondtal vermenigvuldigd, zoals:

   $$g^x = g \cdot g^{x-1}$$

   Het mag duidelijk zijn dat deze operaties het resultaat van de machtsverheffing niet veranderen. En dit gaat zo door zo lang de exponent van g groter is dan 0. Een voorbeeld:

   $$g^{15} = g \cdot g^{14}$$
   $$g^{14} = g^7 \cdot g^7$$
   $$g^7 = g \cdot g^6$$
   $$g^6 = g^3 \cdot g^3$$
   $$g^3 = g \cdot g^2$$
   $$g^2 = g \cdot g$$

$g = g \cdot g^0$
$g^0 = 1$

Dit zijn dus 7 vermenigvuldigen, tegenover de 14 vermenigvuldigingen die het originele algoritme nodig zou hebben.

4. Het Holemens-algoritme is een simpel algoritme om de logaritme van een machtsverheffing uit te rekenen. Dat wil zeggen, uit $g^x = y$ kun je x weer terug rekenen als je g en y weet. Het Holemens algoritme gaat simpelweg alle mogelijke exponenten, g^0, g^1, g^2, etc. vergelijken met y totdat er gelijkheid wordt geconstateerd. Elke stap van dit algoritme kost maar 1 vermenigvuldiging, omdat we voor elke iteratie het resultaat van de vorige iteratie alleen maar met g hoeven te vermenigvuldigen om het resultaat van de huidige iteratie te berekenen. Echter: Hoe groter x is, hoe meer vermenigvuldigingen benodigd zijn om x terug te vinden, en dus hoe langer dit algoritme erover doet om x terug te vinden.

5. Het Pollard Rho-algoritme is een stuk sneller algoritme om een logaritme uit te rekenen. Het algoritme zoekt op een slimme manier naar gelijkheden. Het is gebaseerd op het algoritme van Floyd voor het opsporen van cyclussen. We gaan twee reeksen maken, die in een cyclus lopen, waarbij de tweede reeks twee keer zo snel gaat als de eerste reeks. Op die manier wordt er snel een gelijk element gevonden van de twee reeksen.

   Bij het oplossen van $g^x = y$ worden willekeurige getallen w getrokken, waarvan de *representatie* bekend is, dat wil zeggen waarvoor we een u en v kennen waarvoor geldt dat:

   $w = y^u \cdot g^v$

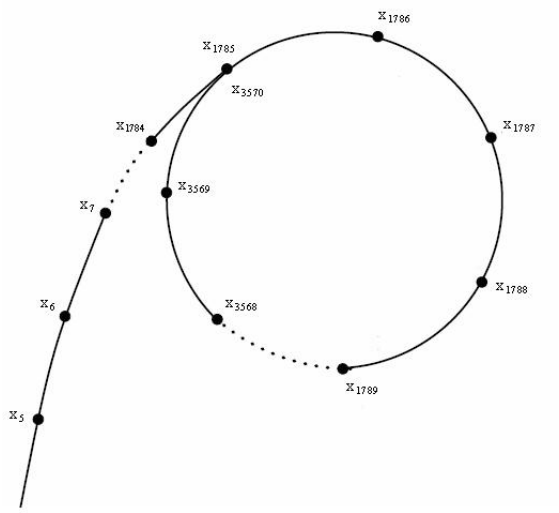
   Vervolgens wordt de groep van orde q in drie deelverzamelingen verdeeld van ongeveer dezelfde grootte. Nu definiëren we een functie die met behulp van de drie deelverzamelingen bepaald wat het volgende element van de reeks gaat worden. We blijven deze functie uitvoeren op twee reeksen, totdat we een element vinden in de eerste reeks dat gelijk is aan een element in de tweede reeks. Van dit element zijn dan twee representaties bekend, namelijk: $w = y^u \cdot g^v$ en $w = y^r \cdot g^s$. Uit die twee representaties is dan de logaritme te berekenen, omdat uit $y^u \cdot g^v = y^r \cdot g^s$ volgt:

   $y = g^{\frac{s-v}{u-r}}$

   Dus:

   $x = \frac{s-v}{u-r}$

   Figuur 1 laat zien hoe de reeksen eruit zien. Het algoritme is dan ook vernoemd naar de Griekse letter rho waar de reeksen op lijken.



Figuur 1: De reeksen van het Pollard Rho-algoritme[5]

4 Uitvoering

Bij het programmeren van de app in Android ben ik tegen een aantal dingen aangelopen, die ik hier zal toelichten. De app is geschreven in Java en hiervoor heb ik de Android-module van IntelliJ gebruikt.

Te beginnen met BigIntegers. Bij het programmeren van alle algoritmes, kwam al snel naar boven dat de primitieve types int en long lang niet genoeg opslagcapaciteit hebben om de tussenresultaten van de algoritmes op te kunnen slaan. Hierdoor heb ik gekozen om in plaats van de primitieve types, het voorgedefinieerde object BigInteger te gebruiken.

Het voordeel hiervan is dat BigIntegers geen bovenlimiet hebben, zoals ints en longs wel hebben (bovenlimiet van int: 2,147,483,647, bovenlimiet van long: 9,223,372,036,854,775,807). Bij kleine getallen als parameters en berichten zou een long wel voldoende zijn, maar dan zouden de crypto-algoritmes niet meer veilig zijn. Een BigInteger is dus vereist als je een veilige manier van communicatie wilt hebben. Een extra voordeel van BigIntegers is dat een modulaire exponentiatie al een voorgedefinieerde methode is van BigInteger.

Een nadeel van BigIntegers is de opslagruimte die het vergt. Ints en longs gebruiken respectievelijk 32 en 64 bits, waardoor de wiskundige operatoren hier efficiënt mee kunnen werken. Een BigInteger daarentegen gebruikt een array van ints om alles op te slaan. In principe wordt de BigInteger in kleine blokken gehakt waarvan elk wordt opgeslagen in de vorm van een int op de juiste plaats in de array. Omdat een BigInteger op deze manier wordt opgeslagen, kosten alle berekeningen met BigIntegers ook meer tijd. Op mijn eigen Android uit 2015 merk je het verschil echter niet. Het zou kunnen dat het bij

oudere modellen wel problemen oplevert.

Een andere bewuste keuze betreft het opslaan van de parameters. Een programmeur zou kunnen kiezen om dit in een superklasse van alle klassen, die gebruik maken van de parameters, op te slaan. Google heeft hier echter voor Androids al iets voor gemaakt, namelijk een PreferenceScreen. In een PreferenceScreen kun je zogenaamde preferences (onze parameters) implementeren, met informatie over wat voor preference het is (getal of een string), wat de standaardwaarde is, naam, omschrijving, etc. Ik heb hier voor gekozen, omdat een PreferenceScreen automatisch een layout maakt voor het scherm waar de parameters ingesteld kunnen worden. Dit vond ik dermate nuttig, dat ik niet voor het opslaan van parameters in de superklasse ben gegaan. Een PreferenceScreen is ook gemakkelijk aan te passen, mochten er later nog parameters bijkomen.

Een nadeel van een PreferenceScreen is, in het geval van de Elgamal-app, dat er bij een paar functies van het parameterscherm, meerdere of helemaal geen invoer nodig is. Dit is een nadeel omdat een Preference standaard één invoer gebruikt. Hiervoor heb ik een eigen MultiEditTextPreference gedefinieerd, wat met twee invoeren om kan gaan. Om met geen invoer om te kunnen gaan heb ik een ResetPreference gedefinieerd. De functie van die Preference is dan ook alle parameters naar de standaardwaarden van TI-83+ herstellen.

Een ander nadeel van een PreferenceScreen is dat er bepaalde eisen aan bepaalde parameters zitten, zoals eerder besproken. Een standaard PreferenceScreen kan hier niet mee omgaan, maar om dit te verhelpen heb ik de onPreferenceChangeListener van elke Preference aangepast. De onPreferenceChangeListener wordt aangeroepen op het moment dat een gebruiker een parameter wilt veranderen. Voordat de parameter aangepast wordt, controleert het eerst of de nieuwe waarde wel aan de eisen voldoet.

Eén bewuste keuze betreft de toolbar en NavigationDrawer van de app. Om dit netjes over alle schermen te krijgen, heb ik een superklasse gemaakt over alle klassen die dit nodig hadden, genaamd *DrawerActivity*. Als deze klasse niet had bestaan had elke klasse zijn eigen toolbar en NavigationDrawer gehad, wat redundante code zou zijn.

Deze klasse genereert dus de toolbar en NavigationDrawer en legt hem over bijna elk ander scherm heen. Zo een superklasse bood ook andere voordelen. Ik heb hier onder andere de methode voor het uitrekenen van de multiplicatieve inverse onder gezet, omdat dat voorkwam in meerdere klassen. Op die manier heb ik redundante code zo veel mogelijk geprobeerd te vermijden.

Waarschijnlijk de belangrijkste keuze die ik gemaakt heb, is de keuze tussen Bluetooth- en NFC-verbinding. Het idee hierachter is dat, omdat bijna elke Android Bluetooth- of NFC-verbinding kan maken, het versturen van versleutelde berichten naar een medestudent makkelijker maakt. Ik heb gekozen voor de NFC-verbinding, omdat dat eigenlijk

een impliciete Bluetooth-verbinding maakt zonder dat de gebruiker zelf hoeft te kiezen met welke andere Android hij communiceert. De studenten houden simpelweg hun Androids tegen elkaar en het bericht is verstuurd. Als er een Bluetooth-verbinding gebruikt zou worden, zou er tijdens een workshop op heel veel apparaten tegelijkertijd Bluetooth aan staan, waardoor het voor de studenten meer tijd kost om de juiste Android om mee te verbinden uit de lijst met mogelijke apparaten te kiezen. Daarbij komt ook dat er over NFC op Android meer documentatie te vinden is.

5 Resultaat

Het resultaat van de app Elgamal is te vinden in de Google Play Store en is beschikbaar voor Androids met softwareversie 4.0 of hoger. Bij het opstarten van de app verschijnt een scherm met basisinformatie over Taher Elgamal en de app. Via de menu-knop kan dan genavigeerd worden naar andere schermen, waar bijvoorbeeld parameters ingesteld of berichten versleuteld kunnen worden. Na het versleutelen van een bericht verschijnt er een dialoog, waar het versleutelde bericht staat. In dit dialoog staat ook de mogelijkheid om naar het NFC-scherm te gaan, waarna de NFC-connectie met een andere Android-smartphone gelegd kan worden. Hierbij is wel vereist dat beide smartphones NFC ondersteunen. Mocht een van de smartphones geen NFC ondersteunen, dan zullen de versleutelde berichten handmatig overgebracht moeten worden, dat wil zeggen: Dan zal de student zijn versleutelde bericht moeten opnoemen, waarna de mede-student het in kan typen op zijn smartphone om vervolgens het bericht te ontsleutelen met de ontsleutel-optie.

Op het moment dat een versleuteld bericht via NFC verstuurd is naar een andere smartphone, verschijnt op het scherm van de ontvanger een dialoog, waar hij de keuze heeft om het ontvangen bericht te ontsleutelen met Caesar-codering of Elgamal-codering. Het is wel belangrijk dat alle parameters op beide smartphones goed ingesteld zijn!

De app voor Androids is een bijna exacte replica van het TI-83+ programma van Gerard Tel. Het heeft dezelfde functies en algoritmes, maar kan, door de rekenkracht van Androids, met grotere getallen rekenen dan het programma op de TI-83+. De app kan weliswaar nog crashen als er toch met te grote getallen gerekend wordt. Op dat moment loopt de Android even vast, en zal de app automatisch afgesloten worden. Het is echter, door verschillende rekenkrachten op verschillende smartphones, per smartphone verschillend hoe groot de getallen kunnen worden.

De Android-app kan nog uitgebreid worden. Het Pollard Lambda-algoritme, wat vergelijkbaar is met het Pollard Rho-algoritme, kan nog toegevoegd worden. Het Pollard Lambda-algoritme genereert, in tegenstelling tot het Pollard Rho-algoritme, meerdere reeksen. Deze reeksen kruisen elkaar dan op een bepaald punt, wat gevisualiseerd wordt als een λ. Ook kan er nog een in-app handleiding toegevoegd worden, waar informatie in te vinden is over de algoritmes en wat ze precies doen. Dit zou een nuttige toevoeging

zijn, omdat er nu in de app weinig informatie te vinden is over de algoritmes. De app is functioneel, maar nog niet educatief genoeg.

De app is te vinden in de Google Play Store onder de naam Elgamal.[6] Appendix A is een handleiding voor de app Elgamal. De handleiding beslaat een groot deel van wat er in dit artikel is behandeld, maar is gericht op het daadwerkelijk gebruiken van de app Elgamal.

6 Relevantie voor KI

In deze sectie zal ik de relevantie van de Android-app Elgamal voor Kunstmatige Intelligentie beschrijven aan de hand van de volgende onderwerpen: Onderwijs en werving, complexiteit en ethiek.

6.1 Onderwijs en werving

Het voornaamste doel van de Android-app Elgamal is de ondersteuning bij groepsactiviteiten over cryptografie. De participanten kunnen nadat ze de benodigde theorie geleerd hebben, zelf de algoritmes uitproberen. De participanten krijgen hierdoor een beter beeld van de algoritmes, omdat ze het in de praktijk ervaren. De participanten kunnen zelf beredeneren waarom het werkt, en in het geval dat er iets fout gaat kunnen ze zelf beredeneren waarom het fout gaat. Dat de app voor Android is geprogrammeerd zal ze ook aanspreken, waardoor ze als gevolg langer met de app in de weer zullen gaan.

In het geval van werving van studenten naar de universiteit kan de app ook van pas komen. Omdat de app toegankelijk is voor iedereen, kan op een open dag, meeloopdag of matchingsdag een workshop worden opgezet waar een demonstratie wordt gegeven van de app. Hopelijk spoort dit de toekomstige studenten dan aan om Informatica of KI te gaan studeren en meer over cryptografie te leren.

Studenten Kunstmatige Intelligentie aan de Universiteit Utrecht kiezen regelmatig Security als keuzevak voor hun studieprogramma of volgen Security als onderdeel van hun minor Informatica. Hierdoor komen regelmatig KI-studenten in aanraking met cryptografie. Mocht de studenten hierna interesse behouden voor cryptografie, dan kunnen zij verder onderzoek doen naar de combinatie van KI met cryptografie. Een voorbeeld hiervan is het laten communiceren van twee agents zonder dat een derde agent weet waar ze het over hebben. En dat zonder dat de twee communicerende agents cryptografische algoritmes kenden om uit te proberen.[7]

6.2 Complexiteit

Cryptografie maakt gebruik van wiskundige problemen, zoals, in het geval van het Elgamal-algoritme, de discrete logaritme. Deze problemen zijn computationeel moeilijk uit te rekenen, omdat er nog geen efficiënt algoritme voor gevonden is. Beter bekend

als NP-problemen. De discrete logaritme is een NP-probleem, omdat de uitkomst van een logaritme niet in polynomiale tijd is uit te rekenen, maar wel in polynomiale tijd verifieerbaar is. Door weer te machtsverheffen kunnen we verifiëren of een uitgerekend logaritme klopt.

De aanname dat NP-problemen niet op te lossen zijn met de huidige rekenkracht van computers, is dus de basis van cryptografie. Al is dit in gevaar door de opkomst van kwantumcomputers. Deze computers moeten, volgens Grovers algoritme, een probleem, dat normaal in n stappen kost, in $\sqrt{n}$ stappen kunnen oplossen.[8] Dit houdt in dat Grovers algoritme een 128-bits symmetrische sleutel in 2^{64} iteraties zou kunnen brute forcen in plaats van in 2^{128} iteraties. Nu is dit in de praktijk nog ver te zoeken, maar er zijn tekenen dat het groot gaat worden.[9]

De complexiteit van problemen is een belangrijk deel van KI en Informatica. Hierdoor kunnen we van tevoren de rekentijd van algoritmes schatten. Zeker nu er meer en meer data opgeslagen wordt, bekend als Big Data, gaat de complexiteit een grotere rol spelen. Machine learning-algoritmes gebruiken de Big Data om patronen te herkennen, maar als de complexiteit van de algoritmes te hoog ligt, zouden de algoritmes veel minder nuttig zijn dan als ze een lage complexiteit zouden hebben. Bij een hoge complexiteit gaat het trainen van de algoritmes namelijk te lang duren.

6.3 Ethiek

Ethiek is een leidend deel van de filosofie van KI. In relatie met cryptografie stelt het vragen, zoals: Is een agent die zelf cryptografische algoritmes ontwikkelt wel wenselijk? Redeneert zo een agent met de belangen van de mens voorop? En wat als het algoritme dat het ontwikkeld heeft zo moeilijk is dat de mens het nooit zou kunnen begrijpen? Dit zijn belangrijke dilemma's. Aan de ene kant willen we heel graag een onbreekbaar algoritme, maar wat als dat algoritme in de verkeerde handen valt? Kunnen we verdachte mensen dan nog in de gaten houden? Zo zijn er situaties te bedenken waarin je geen onbreekbaar algoritme wilt hebben, maar is privacy dan nog gewaarborgd?

KI en cryptografie komen steeds meer in het nieuws, waarbij de vraag opkomt of het moreel verantwoordelijk is om bijvoorbeeld ächterdeurtjesïn encryptie te stoppen[10] of om zelfrijdende auto's daadwerkelijk zelf te laten rijden[11].

7 Samenvatting

De app Elgamal voor Androids is een nieuwere versie van het TI-83+ programma door Gerard Tel. Dat programma wordt gebruikt, tijdens workshops over cryptografie, om een praktische ervaring te bieden met cryptografie. Een nieuwere versie van het programma is nodig, omdat de TI-83+ tegenwoordig een schaars goed is. Androids zijn daarentegen overal, en het is dan ook een logische stap om een app te maken voor Androids die

hetzelfde kan als het TI-83+ programma. De app bevat dezelfde functies en algoritmes als het TI-83+ programma, namelijk het symmetrische cryptoalgoritme Caesar-codering, het asymmetrische cryptoalgoritme Elgamal-codering, machtsverheffen en het Holemens-algoritme en Pollard Rho-algoritme om logaritmes uit te rekenen. De app is werkzaam en beschikbaar voor Androids met softwareversie 4.0 of hoger. Er zijn nog verbeteringen voor de app mogelijk, zoals bijvoorbeeld het toevoegen van het Pollard Lambda-algoritme of een in-app handleiding voor de algoritmes. De code van de app Elgamal staat in een private repository op BitBucket. Mocht je de code willen inzien, dan kun je contact met mij opnemen via pim.hopmans@live.nl.

Appendices

A Handleiding voor de app Elgamal

A.1 Inleiding

Dit is een handleiding voor de Android app Elgamal, te vinden in de *Google Play Store*, geschreven door Pim Hopmans als bachelor eindproject onder begeleiding van Gerard Tel.
Als je de app opstart kom je op het toegangsscherm. Hier is wat informatie te vinden over Taher Elgamal, de ontwerper van het Elgamal-algoritme, en wat basisinformatie over het algoritme.
Als je op de menu-knop linksboven tikt, verschijnt het navigatiemenu. Vanuit hier kun je navigeren naar *Caesar Code*, *Elgamal Code*, *Log en Macht* en *Parameters*. Deze activiteiten kun je starten door er op te tikken. Uitleg hierover kun je in deze handleiding vinden.
Je kunt het programma te allen tijde stoppen door op de terug-knop te blijven drukken tot de app sluit, of door hem handmatig te eindigen.

A.2 De activiteit Caesar Code

De Caesar code gebruikt voor versleutelen en ontsleutelen dezelfde sleutel, een getal z, allebei berekend met resten. Versleutelen is vermenigvuldigen met z modulo p, en ontsleutelen is delen door z modulo p. De standaard modulus die wordt gebruikt is 95917. De modulus kun je zelf aanpassen in de activiteit *Parameters*.
De keuze *Sleutel kiezen* kiest een willekeurig getal voor z kleiner dan p. De keuze *Sleutel instellen* laat je zelf een sleutel instellen en de keuze *Sleutel tonen* laat je de huidige waarde van z en p zien. Bij de keuzes *Versleutelen* en *Ontsleutelen* kun je respectievelijk een bericht wat je wilt versturen en een ontvangen bericht invoeren en deze vermenigvuldigen of delen door z.
Omdat een deling met een modulus niet zo gemakkelijk te programmeren is, wordt het ontsleutelen berekend door eenmaal te machtsverheffen. Dat kan omdat p een priemgetal is, en volgens de Kleine Stelling van Fermat geldt dan $z^{p-1} \equiv 1 (\text{mod } p)$. Dat betekent dat de deling x/z, oftewel $x \cdot y^{-1}$, uit te drukken is als $x \cdot y^{p-2}$. Hoe het machtsverheffen precies werkt, kun je lezen in de sectie *De activiteit Log en Macht*.

A.3 De activiteit Elgamal Code

In deze activiteit zijn de verschillende berekeningen van het Elgamal geheimschrift aan te roepen. Hierbij zijn de parameters p, q en g belangrijk. Deze kun je aanpassen in de activiteit *Parameters*.
Met *Sleutel maken* kun je een sleutelpaar genereren, bestaande uit een geheime en een publieke sleutel. Je begint met het bepalen van je eigen geheime sleutel, a. Dit kun je zelf kiezen en invoeren, maar als je een 0 invoert neemt het programma een willekeurig

getal tussen 1 en $q-1$. Jouw publieke sleutel b wordt vervolgens berekend door $b = g^a$. Deze sleutels worden opgeslagen in je parameterinstellingen in de activiteit *Parameters*. Het is belangrijk voor de veiligheid van het algoritme dat je je geheime sleutel ook daadwerkelijk geheim houdt!
Met *Public key invoeren* kun je de publieke sleutel van de partij waarmee je wilt communiceren, laten we hem Bob noemen, invoeren. Dit wordt ook opgeslagen in je parameterinstellingen.
Met de keuze *Versleutelen* kun je jouw bericht invoeren, wat vervolgens wordt versleuteld met Elgamal. De cipher die hier uitrolt is een getallenpaar (u, v). Uiteraard wordt hier de publieke sleutel van Bob voor gebruikt, zodat alleen hij het kan ontsleutelen. De versleuteling kiest een random getal k en berekent $u = g^k$ en $v = x \cdot b^k$, waarbij b de publieke sleutel van Bob is. Niet je eigen publieke sleutel!
Met *Ontsleutelen* kun je een ontvangen getallenpaar (u, v) ontsleutelen. Dit gebeurt, zoals uitgelegd, door $m = v/u^a$. Hier zit weer een modulaire deling in en om dat niet te hoeven programmeren wordt dit in feite gedaan door $m = v \cdot u^{-k}$.
Met *Sleutels zien* kun je je eigen geheime en publieke sleutel en de ingestelde publieke sleutel van Bob zien.

A.4 Log en Macht

In de activiteit *Log en Macht* zijn drie rekenkundige algoritmes beschikbaar, waarmee je wat kunt experimenteren. Twee van de algoritmen rekenen de logaritme uit van een getal en een grondtal, en het derde algoritme rekent de macht uit van een grondtal gegeven een exponent. Hierbij worden weer dezelfde parameters p, q en g gebruikt. Je kunt deze parameters instellen in deze activiteit onder de keuze *Parameters instellen*.
Met *Machtsverheffen* kun je bij een getal x de waarde van g^x opvragen. Hierbij wordt verteld hoe veel vermenigvuldingen het programma hiervoor gebruikt heeft. Je kunt hierbij zien dat bij een exponent van k cijfers nooit meer dan $6.64k$ vermenigvuldigingen nodig zijn. Verderop kun je lezen hoe de macht precies uitgerekend wordt.
Met *Holemens logaritme* kun je van het resulterende getal de logaritme weer berekenen. Maar het Holemens algoritme is erg duur, omdat er achtereenvolgens de waarden g^0, g^1, g^2, g^3, etc. worden uitgeprobeerd totdat er gelijkheid wordt geconstateerd. Dus hoe groter de exponent was bij *Machtsverheffen*, hoe meer tijd het Holemens gaat kosten om hem terug te vinden. Een Android-telefoon is tegenwoordig doorgaans zo snel, dat het niet te merken is dat Holemens inefficiënt is.
Het algoritme van Pollard, dat je met *Pollard Rho* kunt aanroepen, werkt veel slimmer. Er worden 'willekeurige' getallen w getrokken, maar zo, dat er bij elk getal w getallen u en v bekend zijn waarvoor $w = y^u \cdot g^v$; die u en v heten een representatie van w. Hiermee gaat Pollard door, totdat er onder de trekkingen twee getallen 'toevallig' aan elkaar gelijk zijn. Van dat getal zijn dan twee representaties bekend: $w = y^u \cdot g^v$ en $w = y^r \cdot g^s$. Uit die representaties is dan de logaritme te berekenen, want uit $y^u \cdot g^v = y^r \cdot g^s$ volgt $y = g^{\frac{s-v}{u-r}}$.

Hoe werkt machtsverheffen in Elgamal?

Als voorbeeld bespreken we hier het berekenen van $x \cdot z^{p-2}$, maar machtsverheffen komt op diverse plaatsen in het programma voor, al is het steeds met andere variabelen. Het machtsverheffen begint met de waarden van het grondtal z en de exponent $p-2$ op te slaan in de variabelen o, respectievelijk e. De gewenste uitkomst $x \cdot z^{p-2}$ is nu gelijk aan $x \cdot o^e$, en omdat e een variable is mogen we de waarde ervan wijzigen. Het zou heel prettig voor ons zijn als e gelijk was aan 0, want dan was $x \cdot o^e$ namelijk gewoon x.

```
o = z; e = p - 2;
while (e > 0) {
    if ((e % 2) == 0) {
        e = e / 2;
        o = o * o;
    } else {
        e = e - 1;
        x = o * x;
    }
}
```

Wat er nu steeds in de while-loop gebeurt, is dat de waarde van e kleiner wordt gemaakt, zonder dat de waarde van $x \cdot o^e$ verandert! Als e even is, wordt e gehalveerd en o gekwadrateerd; dit is correct omdat $x \cdot (oo)^{e/2}$ weer gelijk is aan $x \cdot o^e$. Als e oneven is, wordt e verlaagd en x met o vermenigvuldigd; dit is correct omdat $(xo) \cdot o^{e-1}$ weer gelijk is aan $x \cdot o^e$. (Na elke vermenigvuldiging moet de rest bij deling door p worden bepaald, maar die bewerking is hier weggelaten.)
Als de while-loop termineert is $x \cdot o^e$ dus niet veranderd, oftewel, nog steeds gelijk aan de gewenste waarde. Maar omdat nu $e = 0$, wordt x als uitvoer gegeven.
Hoe vaak wordt de lus doorlopen? Als we beginnen met e een getal van k bits lang (dus: kleiner dan 2^k), dan kan e hoogstens k maal worden gehalveerd voordat de waarde 0 wordt; het Then gedeelte wordt dus hoogstens k maal uitgevoerd. Als een oneven getal met 1 wordt verlaagd ontstaat een even getal; daarom wordt, tussen twee uitvoeringen van het Then gedeelte, het Else gedeelte hoogstens eenmaal uitgevoerd. Het aantal rondes van de loop is daarom hoogstens $2k$.

A.5 Parameters

In deze activiteit kun je de verschillende parameters instellen die betrekking hebben op de Elgamal code. Als je op een parameter tikt, verschijnt er een schermpje waar je de nieuwe waarde in kan stellen. Er zijn wel een paar eisen aan de parameters:

- z en p moeten priemgetallen zijn.
- q moet een priemgetal en een deler van $p-1$ zijn.
- $g^q = 1 \pmod{p}$ moet gelden.

Verder heb je nog twee keuzes, *Parameters genereren* en *Parameters herstellen*. Onder *Parameters generen* kun je limieten voor p en q opgeven, waaruit vervolgens de grootste

p en q en een willekeurige g uit worden bepaald die voldoen aan alle eisen. Dat wordt op de volgende manier gedaan:

- *Het vinden van de orde:* Vanaf de opgegeven grens worden oneven getallen getest of ze priem zijn. Dat gebeurt door ze te delen door alle oneven getallen kleiner dan hun wortel; als een deler wordt gevonden probeert het programma een kleinere waarde van q.
- *Het vinden van de modulus:* De modulus moet een priemgetal zijn, en om straks een orde te kunnen vinden zorgen we ervoor q een deler is van $p-1$. Het programma probeert daarom oneven q-vouden plus 1, beginnend bij de opgegeven grens.
  Voorbeeld: als $q = 11$ en de grens op p is 170, dan probeert het programma achtereenvolgens 155, 133, 111, etc. Als geen priemgetal wordt gevonden (het programma komt dan uit bij 1), wordt een lagere waarde van q geprobeerd.
- *Het vinden van de generator:* We hebben nu priemgetallen p en q, en zoeken naar een getal g dat orde q heeft. Volgens de kleine stelling van Fermat geldt, als je modulo p rekent, voor elk getal w: $w^{p-1} = 1$. Omdat q een deler is van $p-1$ kun je $p-1$ schrijven als $p-1 = r \cdot q$. Dan zie je dat voor elk getal w geldt: $w^{rq} = 1$. Het programma neemt voor w getallen vanaf 100 en neemt daarvan de macht $(p-1)/q$. In het (weinig voorkomende) geval dat het resultaat 1 is, wordt de volgende waarde van w geprobeerd.

Met de maximale grenzen van 5000000 op q en 10000000 op p vind je $q = 4999523$ en $p = 9999047$.
En als laatste: Met de keuze *Parameters herstellen* kun je alle parameters herstellen naar de TI-84 waarden. Dat is als volgt:

- $p = 95917$
- $q = 7993$
- $g = 29609$

Referenties

[1] Taher Elgamal (1985),
A Public key Cryptosystem and A Signature Scheme based on discrete Logarithms,
IEEE Transactions on Information Theory.

[2] Gerard Tel (2005),
Universiteit Utrecht,
Workshop Geheimschrift op de TI-83+,
http://www.staff.science.uu.nl/ tel00101/Cryptografie/Elgamal/

[3] Gerard Tel (2005),
Universiteit Utrecht,
*Het programma **ELGAMAL***,
http://www.staff.science.uu.nl/ tel00101/Cryptografie/Elgamal/ElgProg.pdf

[4] Jeroen Fokker,
Universiteit Utrecht,
Mobiel Programmeren,
http://www.cs.uu.nl/docs/vakken/b1mop/diktaat/mop.pdf

[5] Figuur 1:
https://nl.wikipedia.org/wiki/Pollards_rho-algoritme#/media/File:Pollard_rho_cycle.jpg

[6] Pim Hopmans
Elgamal (2017)
https://play.google.com/store/apps/details?id=com.hopmans.pim.elgamal

[7] Martín Abadi, David G. Andersen,
Learning to Protect Communications with Adversarial Neural Cryptography,
Google Brain.

[8] Grover L.K.,
A fast quantum mechanical algorithm for database search,
Proceedings, 28th Annual ACM Symposium on the Theory of Computing, (May 1996)
p. 212

[9] Gershon, Eric,
New qubit control bodes well for future of quantum computing,
Phys.org (14-01-2013).

[10] NU.nl (04-05-2017),
'Britse overheid wil achterdeurtjes in encryptie verplicht maken',
http://www.nu.nl/internet/4669320/britse-overheid-wil-achterdeurtjes-in-encryptie-verplicht-maken.html

[11] NU.nl (03-05-2017),
PSA Peugeot Citroën gaat zelfrijdende auto's testen in Singapore,
http://www.nu.nl/internet/4664235/psa-peugeot-citroen-gaat-zelfrijdende-autos-testen-in-singapore.html