Verslag Project Codetheorie

Sibert Aerts Cédric De Haes

2017-04-28

1 Vigenère Plus

We zijn er in geslaagd de Vigenère Plus code te kraken.
We begonnen met het wiskundig uitschrijven van de gegeven encryptiemethode: Een Vigenère versleuteling gevolgd door een enkele kolom transpositie. We merkten meteen dat er geen voor de hand liggende wiskundige of statistische oplossing was zolang de kolom sleutel (of tenminste de lengte hiervan) en de lengte van de Vigenére sleutels onbekend waren.

Om Vigenère op te lossen analyseert men bepaalde attributen van de code: Het herhaaldelijke voorkomen van factoren in de afstanden tussen identieke substrings van 3 karakters, deze herhaalde factoren verklappen de lengte van de Vigenére sleutel. De kolom transpositie (of het omkeren van de transpositie met een foute sleutel) zorgt er echter voor dat deze onderliggende patronen verborgen blijven. Enkel als men een (quasi) juiste sleutel gebruikt om de kolom transpositie om te keren zal men een code verkrijgen waarin men zo patronen kan herkennen.

Met dit in gedachte bedachten we een methode om een arbitraire sleutel k een score te geven:

1. Gebruik de sleutel k om de enkele kolomtranspositie ongedaan te maken.
2. Analyseer de resulterende code en gebruik dit om een score te berekenen.
3. Associëer deze score met de sleutel k.

Het berekenen van de score in stap 2 gebeurt aan de hand van een heuristiek, in dit geval bedachten we een simpele score functie: De som van elke factor maal het aantal voorkomens van die factor. We pasten deze methode toe op een grote hoeveelheid willekeurig gekozen sleutels met lengtes tussen 2 en 10. Uit de scores viel onmiddellijk op dat de sleutels met de hoogste scores lengte 7 hadden, wat aangeeft dat dit waarschijnlijk de correcte sleutellengte is.

Nu kennen we echter enkel de lengte, en niet de sleutel zelf. Bij wijze van test namen we de hoogst scorende sleutel van lengte 7 (die zeker niet gegarandeerd was de juiste sleutel te zijn) en voegden het resultaat in de *Vigenère Cracking Tool* [1]. Deze gaf zeer duidelijk aan dat de Vigenère sleutel lengte 14 moest

[1] `http://www.simonsingh.net/The_Black_Chamber/vigenere_cracking_tool.html`

zijn. Dit is een enorm behulpzaam feit want 14 is een veelvoud van 7, wat een speciale eigenschap geeft aan de combinatie Vigenère en kolom transpositie:
Neem:

$c =$ de correcte kolom sleutel (lengte 7)

$v =$ de correcte Vigenère sleutel (lengte 14)

$text =$ de originele text

$code = Col_c \circ Vig_v(text) =$ de gegeven code

$c' =$ een arbitraire kolom sleutel (lengte 7)

$v' =$ de Vigenère sleutel (lengte 14) gevonden aan de hand van c'

$text' = Vig_{v'}^{-1} \circ Col_{c'}^{-1}(code)$

Dan kan men aantonen dat omwille van de lengtes van c en v:

$text'$ is een permutatie van $text$

v' is een permutatie van v

$code' = Col_{c'}(text') = Col_{c'} \circ Vig_{v'}^{-1} \circ Col_{c'}^{-1}(code)$

$= Col_{c'} \circ Vig_{v'}^{-1} \circ Col_{c'}^{-1} \circ Col_c \circ Vig_v(text)$

$= Col_c(text)$

Kort gezegd: Het vinden van de meest waarschijnlijke Vigenère sleutel voor een foute kolom sleutel die wel de juiste lengte heeft, staat ons toe om de Vigen'ere geheel teniet te doen. Het resultaat, $code'$, is een tekst die enkel versleuteld is met een kolom transpositie, en aangezien we de lengte van de sleutel kennen is het vinden van de oplossing hiervoor een simpele puzzel die we makkelijk manueel konden uitvoeren.

Uiteindelijk vonden we dus:

- De kolom sleutel: GBDAFCE
- De Vigenère sleutel: SASKIADECOSTER

2 PlayFair

Het is ons niet gelukt om de PlayFair code te kraken.
We probeerden dit te doen met behulp van de frequentie-analyse van bigrammen. Eerst hebben we een playfair decoder geïmplemeteerd. Deze gaat een code nemen en die decoderen volgens een gegeven vierkant. Deze hebben we nodig omdat de tekst te klein is om brute force de frequentie te nemen van elk bigram in de code te vervangen door het bigram met ongeveer gelijke frequentie. Daarom maken we in de start een random vierkant aan als sleutel en decoderen we de code met behulp van deze decoder. Dan wordt op deze gedecodeerde tekst een score geplakt. Deze score wordt berekend aan de hand van de frequentie-analyse gevonden op *fusiontables*[2] We hebben eerst geprobeerd dit te doen met

[2] `https://fusiontables.google.com/DataSource?docid=15jK-3WUD-JjQMdwLe-ipwqkUdjvf2JKK-D-s9as`

de 30 meest voorkomende bigrammen volgens *practicalcryptography*[3], maar dit was onvoldoende. Vanuit die tabel worden de frequenties van alle bigrammen berekend. De frequenties van bigrammen met een J worden opgeteld met die van overeenkomstige bigrammen met een I. Voor de dubbele letters in de tabel gaan we al de mogelijkheden gelijk stellen. Dit wil zeggen: AX en XA hebben dezelfde frequentie: XA + AX + AA. Dit doen we omdat de computer niet kan weten of het gaat om een dubbele letter of een enkele letter met een X. Dit moeten we zelf na gaan.
De score wordt berekend door de som te nemen van de 10 log van de frequenties de bigrammen in de gedecodeerde tekst. Deze score wordt dat vergeleken met de score van enkele andere dichtbijzijnde sleutels:

1. 1 rij omhoog
2. 1 rij omlaag
3. 1 rij links
4. 1 rij rechts
5. 1 rij spiegelen
6. 1 kolom spiegelen
7. 2 random rijen omwisselen
8. 2 random kolommen omwisselen
9. een random pair karakters omwisselen
10. 5 random pair karakters omwisselen
11. 2 random pair karakters omwisselen
12. een random triplet omwisselen
13. de laatste rij in alfabetische volgorde plaatsen

Als de score van van al deze sleutels kleiner is dan die van de oorspronkelijke sleutel, is de oorspronkelijke sleutel de correcte sleutel. Dit wordt dan nog een aantal keren herhaald omdat je werkt met random waardes. Als er een sleutel wordt gevonden met een betere of gelijke score, dan worden dichtbijzijnde sleutels bij die sleutel gekozen.
Op deze manier kwamen we relatief snel aan een oplossing, maar was deze steeds incorrect. Toen bedachten we ons dat de letters die niet in het codewoord staan in alfabetische volgorde achter het codewoord geplaatst worden om het vierkant op te vullen. Dus voegden we de voorwaarde toe dat de laatste 5 karakters van de sleutel in alfabetische volgorde moeten staan. Dit komt niet uit voor codewoorden met meer dan 20 verschillende letters (J niet meegerekend). Daarom

[3] `http://practicalcryptography.com/cryptanalysis/letter-frequencies-various-languages/`

dat we aannemen dat de sleutel toch correct is als in dezelfde sleutel 100 keer naar een betere sleutel gezocht werd. Toch bleeft het doorgaan, dus besloten we die voorwaarde van laatste rij in alfabetische volgorde te laten vallen en er een dichtbijzijnde sleutel van te maken. Zo bekwamen we een vierkant met een code die ons in de juiste richting leek. In de gedecodeerde tekst stonden woorden als "ONDAYIS", waarvan we dachten dat er enkel een "M" voor moest komen. Het vierkant zag er uit als volgt:

D	I	R	B	U
W	O	N	E	G
H	L	T	F	M
Y	S	K	Z	Q
X	A	C	V	P

Daaraan leek het ons duidelijk dat het codewoord "DIRECTION" of "DIRECTOR" zou kunnen bevatten.

3 ADFVGX

Het is ons niet gelukt om de ADFVGX code te kraken.

4 Enigma

Het is ons ook niet gelukt om de enigma code te kraken.
Eerst hebben we een Enigma machine geïmplementeerd. Met behulp van de code en de crib maakten we intern een graaf aan de hand van een dictionary van lijsten. Omdat we geen makkelijk algoritme ken om alle onafhankelijke cycles in een graaf te vinden. Omdat de graaf relatief klein is (24 paden) hebben we die manueel getekent en de cycles op deze figuur aangeduid. Zo bekwamen we volgende graaf.

Daaruit hebben we 4 onafhankelijke paden gevonden:

1. B $\rightarrow$ A $\rightarrow$ B
2. E $\rightarrow$ X $\rightarrow$ E
3. V $\rightarrow$ B $\rightarrow$ R $\rightarrow$ E $\rightarrow$ V
4. V $\rightarrow$ S $\rightarrow$ O $\rightarrow$ F $\rightarrow$ G $\rightarrow$ N $\rightarrow$ L $\rightarrow$ V

Er zijn dus fixpunten voor:

1. $\sigma\mathcal{E}_{k+11}\mathcal{E}_{k+23}\sigma$
2. $\sigma\mathcal{E}_{k+2}\mathcal{E}_{k+7}\sigma$
3. $\sigma\mathcal{E}_{k+5}\mathcal{E}_{k+8}\mathcal{E}_{k+15}\mathcal{E}_{k+10}\mathcal{E}_{k+21}\mathcal{E}_{k+19}\mathcal{E}_{k+14}\sigma$
4. $\sigma\mathcal{E}_{k+12}\mathcal{E}_{k+4}\mathcal{E}_{k+17}\mathcal{E}_{k+1}\sigma$

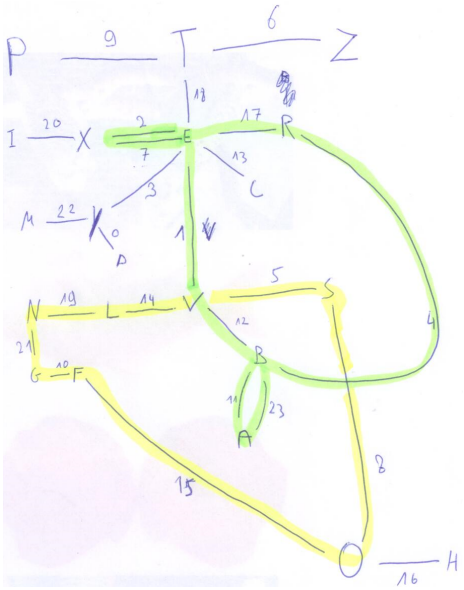

Figure 1: Enigma Graaf

Deze hebben we dan manueel in onze kraker gestoken. Deze kraker probeert dan in alle mogelijke beginstanden een karakter te vinden die op zichzelf afbeelden door deze door de reeks van $\mathcal{E}_{k+i}$ te sturen. Omdat die nog altijd relatief veel mogelijkheden zullen zijn, gaan laten we daarna de geëncrypteerde tekst met die mogelijke beginstanden decoderen en kijken we of de tweede, derde en vierde letter dezelfde zijn. Die zullen gelijk zijn omdat deze letters gelijk zijn in de crib en het stekkerbord letters altijd op eenzelfde letter zal afbeelden. Dit dachten we dat het aantal mogelijkheden fel zou doen dalen en hebben we 's nachts op een server laten draaien.
Daaruit bekwamen we nog altijd meer dan 300 mogelijkheden uit. Dit was te veel om met de hand na te kijken, dus lieten we er nog een check op uitvoeren. We laden de mogelijkheden in en gaan telkens een stekkerbord proberen te maken door te kijken naar de mapping van de crib tot de eerste letters van de gedecodeerde code. Daar mogen geen contradicties in zitten. Dit was jammer genoeg bij elke mogelijkheid wel het geval en uiteindelijk hadden we geen tijd meer om een andere oplossing te implementeren.
Al de 327 mogelijkheden staan in de result.txt in de map Enigma. Het is telkens de setting van de Enigma machine gevolgd door de uitgeschreven code. Wat ons wel opviel was dat het vooral het 7de karakter (de vierde E) was die voor een fout zorgde in de mapping.