Katedralskolan Växjö

Gymnasiearbete
(matematik)

Klassiska chiffer

En analys av dess svagheter

Namn: Anton Vrigborn
Klass: Na21C
VT 2024

Handledare: Åsa Kämpe

Abstract

This paper compares the vulnerability of mono and poly alphabetic substitution ciphers and transposition ciphers with regard to brute force and frequency analysis attacks. It will be based on both previous literature and an implementation of the algorithms in code. The algorithms will be examined with regard to twenty famous texts of varying lengths with the goal of gaining insight towards the algorithm's time complexity. The conclusion drawn was that the attack time was proportional to the text length for all tested attacks. It was also concluded that frequency analysis is multiple times more effective than brute force but Vigenère ciphers does only get marginal security improvements over caesar ciphers.

Sammanfattning

Denna rapport jämför hur utsatta mono- och polyalfabetiska substitutionschiffer samt transpositionschiffer är för brute force och frekvensanalys attacker. Detta genomförs genom både en referens till relevant fakta samt en kodimplementation av algoritmerna. Dessa algoritmer undersöks emot tjugo kända textstycken av varierande längd för att dra slutsatser om algoritmernas tidseffektivitet. Slutsatserna var att för alla attacker är attacktiden proportionell mot textlängden samt att frekvensanalys är mycket effektivare än brute force men Vigenérechiffer ger endast små säkerhetsförbättringar jämfört med ceasarchiffer.

Innehållsförteckning

Inledning

Kryptografi är grekiska för hemlig skrift och handlar om hur man skickar meddelanden utan att någon obehörig kan läsa dem. Kryptoanalys är läran om hur man attackerar kryptografiska algoritmer, så kallade chiffer. Chiffer bygger huvudsakligen på två tekniker, substitution där man byter ut tecken mot andra och transposition där man byter plats på tecken.

Syfte

Syftet med denna undersökning är att undersöka enkla chiffer och deras svagheter för att dra slutsatser om deras mer allmänna egenskaper. Detta är användbart eftersom det kan appliceras på mer komplexa chiffer vilket är viktigt för modern internetsäkerhet.

Frågeställning

Hur utsatta är transposition-, caesar- och vigenère- chiffer för de kryptografiska attackerna frekvensanalys och brute force och vilket samband har tiden för dessa attacker med chiffertexternas längd?

Hypotes

Caesarchiffer förväntas vara enklast att bryta, följt av vigenère medan transpositionschiffer kommer vara säkrast. Förväntan är att frekvensanalys är den bättre attacken för Caesarchiffer och Vigenérechiffer men att brute force kommer vara den enda attacken som bryter transposition. Detta förväntas eftersom frekvensanalys är effektivare än brute-force därför att den kräver mindre antal försök men transposition är inte utsatt för frekvensanalys.

Teori/Bakgrund

Chiffer har funnits i alla tider, från början användes det av militären för att skicka hemliga meddelanden men nu använder vi oss alla av det varje dag eftersom dagens internet är baserat på kryptering för att kunna skicka information säkert.

Teoretiska begrepp

Chiffer är ett system för att kunna skicka meddelanden utan att en tredje part ska kunna läsa dem. De flesta har nycklar och meddelanden, nycklarna är något bara de betrodda parterna har, medan *chiffertexten* är någonting även utomstående kan läsa. Problem kan uppstå om folk som ser chiffertexten kan meddelandet utan nycklar, det är vad attacker försöker göra. *Chiffertext*, även kallat *kryptotext*, är den text som blivit utsatt för något typ av chiffer. *Chiffernyckeln* eller bara *nyckeln* är den hemlighet som krävs för att kryptera och avkryptera meddelanden, oftast är *nyckeln* en lång slumpmässig rad av bokstäver.

Attacker eller attackmetoder är ett sätt exploatera ett chiffers svaghet och få ut det ursprungliga meddelandet utan att vara den tänkta mottagaren.

Ngram är en typ av textstatistik där man undersöker frekvensen av bokstavsrader av samma längd. Ett ngram har längden n. *Quadgram* (quad = fyra) är en underkategori av ngram där man arbetar med bokstavsraden av längd fyra. Monogram (mono = en) är en textstatistik där man söker textrader av längden ett, vilket motsvarar enskilda bokstäver, detta är vad begreppet frekvensanalys oftast syftar på.

Lista 1, 10 vanligaste quadgrams i svenska språket
TILL, NING, ANDE, LAND, NDER, ADES, FÖRS, UNDE, ERNA, TION
Källa: http://practicalcryptography.com/cryptanalysis/text-characterisation/quadgrams/.

Fitnessfunktion är ett välkänt begrepp eftersom det används mycket inom AI men finns även inom kryptering. Det är en funktion som värderar hur bra någonting är, alltså hur hög fitness det har. I relation till kryptering menas en algoritm som testar hur likt äkta text en potentiell attackerad text är. Fitnessfunktionen som implementeras i studien är baserat på

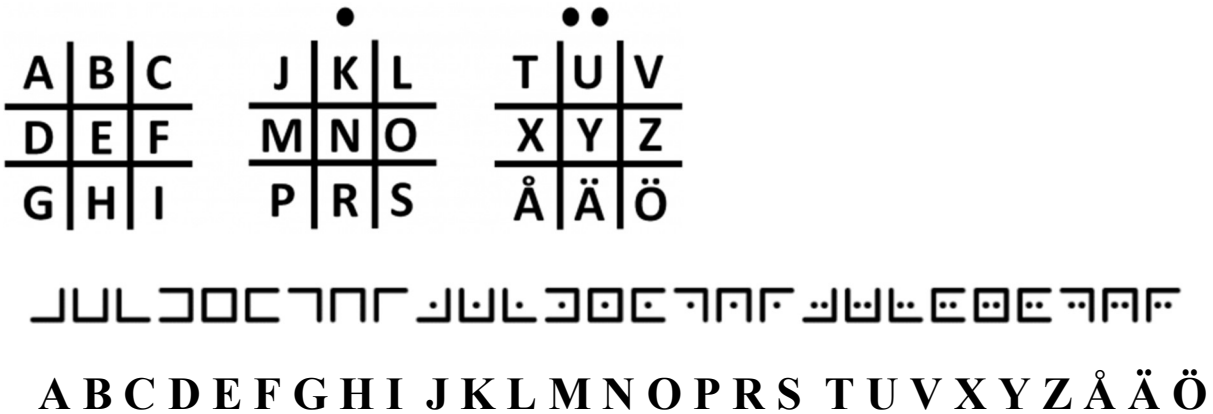
quadgramfrekvenser men det går även att basera dem på ordlistor. Quadgramfrekvenser fungerar som fitnessfunktion eftersom olika quadgramfrekvenserna är olika vanliga.

Chiffer

Substitutionschiffer

Ett substitutionschiffer byter ut en eller flera bokstäver mot andra tecken, genom ett nyckelalfabet som fås genom olika tekniker. Med ett nyckelalfabet för ett substitutionschiffer menas ett system av tecken där varje tecken korresponderar mot en bokstav. Oftast är dessa tecken bara våra vanliga bokstäver fast i annan ordning fast det går även skapa ett med främmande tecken. Vad som skiljer olika substitutionschiffer åt är sättet att bilda detta nyckel alfabet men delar generellt hur detta appliceras. För att försvåra attacker kan nulls(tecken utan betydelse), flera tecken för en bokstav och felstavningar implementeras. Även enkla former av substitutionschiffer är relativt starka mot brute force då det finns 29! ≈ 8,8 *10^30 olika nycklar men är betydligt enklare att bryta med frekvensanalys.

Bild 1, “Scoutchiffer”



från : https://hasselby.scout.se/for-scouter/chiffer/bradgardschiffer/

Här visas ett exempel på ett substitutionschiffer som kallas scoutchiffer eller brädgårdschiffer med dess korresponderande nyckelalfabete. Observera att q och w inte finns med i chiffret.

Caesarchiffer

Caesarchiffer är en underkategori till monoalfabetiska (ett alfabet) substitutionschiffer där man flyttar alla bokstäver x tecken framåt. Om exempelvis x=3 blir i chiffertexten A→ D, B→ E, C→F, D → G. Detta är ett mycket svagt chiffer som bara kan ha 29 olika nycklar och går därför enkelt att lösa om man känner till metoden som använts.

Tabell 1, Caesarchiffer bokstavsförflyttning / Nyckelalfabete.

Vanligt alfabete	a	b	c	d	e	…	ä	ö
Krypterat alfabete	d	e	f	g	h	…	b	c

Vigenèrechiffer

Vigenèrechiffer är ett exempel på ett polyalfabetisk (flera alfabeten) substitutionschiffer. Polyalfabetiska chiffer består av flera nyckelalfabeten. Det fungerar genom att man har ett nyckelord som repeteras för alla bokstäver i meddelandet och deras alfabetpositioner adderas ihop för att få bokstaven för chiffertexten. Det går även att beskriva som flera caesarchiffer som är sammansatta och man alternerar vilket man använder. Därför har det samma svagheter som caesarchiffer förutom att man måste hitta antalet alfabeten använda innan chiffret kan börja bli attackerat. Detta görs genom att söka efter upprepade bokstavskombinationer i chiffertexten och beräkna avståndet mellan dem. Detta eftersom det troligtvis är samma ord i meddelandet som dyker upp i chiffertexten. Om man vet flera sådana delar kan man beräkna deras största gemensamma delare (SGD) vilket motsvarar antalet alfabeten använda. Då kan man fördela bokstäverna i olika högar som motsvara varsitt caesarchiffer vilka kan brytas separat med frekvensanalys. Om man har en nyckellängd lika med meddelandelängd är detta ett säkert chiffer och kan inte brytas.

Tabell 2, Exempel av Vigenèrechiffer med nyckellängd tjugo

Meddelande	a 0	t 19	t 19	v 21	a 0	r 17	a 0	e 4	l 11	l 11	e 4	r 17	i 8	n 13	t 19	e 4	v 21	a 0	r 17	a 0
Nyckel	m 12	p 15	a 0	e 4	x 23	v 21	z 25	r 17	w 22	r 17	u 20	h 7	h 7	p 15	p 15	m 12	i 8	e 4	r 17	d 3
Kryptotext	m 12	f 5	t 19	z 25	x 23	j 9	z 25	v 21	e 4	ö 28	y 24	y 24	p 15	ö 28	f 5	q 16	a 0	e 4	f 5	d 3

Siffrorna under bokstäverna motsvarar deras position i alfabetet, observera hur kryptotextens siffra är summan av meddelandets och nyckelns.

Transpositionschiffer

Transpositionschiffer byter inte ut bokstäver utan ändrar istället ordningen på dem. Transpositionschiffer är säkra mot både frekvensanalys då det inte finns utbytta bokstäver men även brute force attacker eftersom nyckelantalet ökar med fakultet, men är svag mot anagramering[1] av meddelandet. Problemet med denna chiffertyp är att generellt sett kräver transpositionschiffer mer datorkraft än substitutionschiffer. Underkategorin av transpositionschiffer som implementerades i undersökningen kallas kolumntranspositionschiffer. I kolumntranspositionschiffer placeras meddelandet i en matris rad för rad, där kolumnordningen och matrisstorleken bestäms av nyckeln. Meddelandet läses sedan genom att chiffertexten placeras i en likformig matris kolumn för kolumn. Om meddelandet skulle ta slut innan sista raden är full så fylls den med skräpdata, detta händer om kolumnantalet inte är en delare av meddelandets längden.

Tabell 3, Exempel av transpositions matris.

6	7	2	3	1	5	4
A	M	I	D	S	U	M
M	E	R	N	I	G	H
T	S	D	R	E	A	M

[1] *Ett anagram betyder att man tagit ett ord och slängt om bokstäverna i det. Exempel är kela och elak eller engelskans "heart" och "earth". Detta fungerar även för meningar om man tar bort mellanslag. Denna teknik går att använda som en kryptografisk attack men detta analyseras inte i denna studie.*

Meddelandet i exemplet är "amidsummernightsdream". Chiffertexten blir "SIE IRD DNR MHM UGA AMT MES" fast utan mellanslag.

Moderna chiffer

Det tre viktigaste aloritmena som används av dagens datorer för kryptering är AES (advanced encryption algorithm), D-H (Diffie-Hellman) samt RSA (Ron Rivest , Adi Shamir och Leonard Adleman). AES är ett symmetriskt chiffer (likt de vi analyserat i denna studie) vilket betyder att både sändaren och mottagaren måste ha samma nyckel. D-H används för att på ett säkert sätt skicka det första meddelandet och byta nycklar. RSA är ett asymmetriskt chiffer vilket betyder att det har en offentlig och en privat nyckel, detta är användbart för bland annat signera meddelanden.

Attackmetoder

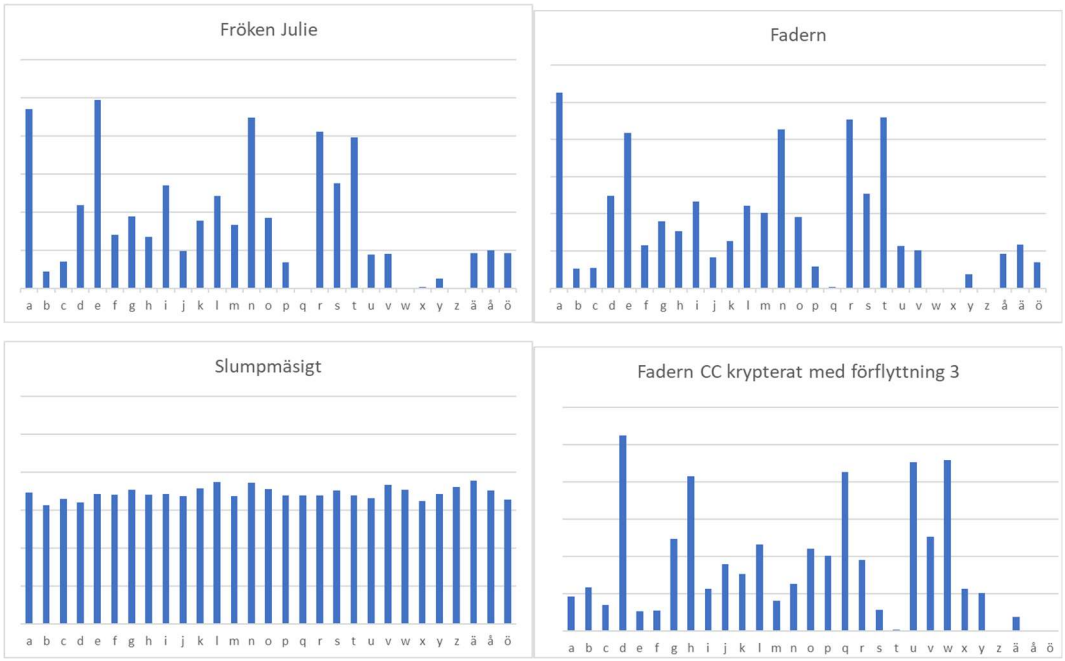
Frekvensanalys

Fungerar genom att bokstäverna har olika frekvenser i texter, alltså ett "t" är mer vanligt att stöta på än "q". Om man bara har bytt ut bokstäver mot varandra kan man alltså lätt se vad det ursprungligen var. Se diagram 1 till 4. Vi kan se att bokstavsfrekvensen är liknande mellan texter och att man därför kan dra slutsatser mellan dem, vilket inte går för en slumpmässig text. Det går att se på den krypterade Fadern texten att den har blivit förflyttad tre steg, alltså krypterad med caesarchiffer med nyckeln 3. Algoritmen för att beräkna detta är summan av absolutbeloppet av alla differenser mellan en standardfrekvens och chiffertextens för alla bokstäver. Summan jämförs sedan med liknande summor från samma text men där standardfrekvensen blivit förflyttad i relation med chiffertextfrekvensen. Den summa som är lägst är mest lik riktig text och då har även chiffernyckeln upphittats.

$$f(d) = \sum_{n=0}^{29} [S_{n+d} - C_n]$$

Här blir f funktionen, d förflyttningen, n bokstavsnumret, S_n bokstav n standardfrekvens och C_n bokstav n chiffertextfrekvens.

Diagram 1, Jämförelse av bokstavsfrekvens hos olika texter



Observera att diagrammen endast är skalenliga med sig själva och inte varandra.

Brute force

Det simplaste och minst effektiva sättet att bryta chiffer är brute force eller även kallat totalsökning. Denna algoritmen testar alla möjliga nycklar för ett chiffer tills dess att en text hittas som liknar svenska. Detta är mycket långsamt men för säkra chiffer är dock detta den enda möjliga attacken, om den enda information om meddelandet är chiffertexten.

Moderna attacker

De chiffer som används idag har inga uppenbara svagheter utom måste bli utsatta för en brute-force attack. AES (advanced encryption standard) är exempelvis baserat på primtalsfaktorisering vilket är långsamt för traditionella datorer, men blir möjligt att lösa med kvantdatorer i en överskådlig tid. Vilket gör att en brute force attack blir rimligt för attackerare. Detta betyder att vi behöver migrera från AES till ett annat säkrare chiffer inom den närmaste framtiden.

Tidskomplexitet och Ordo-notationen

Med att undersöka en algoritms tidskomplexitet menas att undersöka hur tiden det tar att köra algoritmen förändras när indatan ökar. Detta är mycket användbart att veta när man utvecklar algoritmer som skall skalas upp. Oftast bryr man sig endast om den termen som ökar snabbast eftersom för stora indata är detta den enda signifikanta termen. Detta är var ordo-notationen (engelska Big O) visar. Exempel är O(n) som motsvarar en linjärt tidskomplexitet och $O(n^2)$ som motsvarar en algoritm med kvadratisk tidskomplexitet. Oftast undersöker man bästa fall, genomsnittliga fall och sämsta fall för ens algoritm.

I teorin för attacker på caesar och vigenèrechiffer blir komplexiteten O(n) eftersom beräkningen för varje dataenhet inte är beroende på mängden intagen data. Men det går att göra en nästa lika pålitlig attack genom att bara se de första 5000 tecken vilket skulle kunna ge algoritmen en komplexitet på O(1) eftersom den tar lika lång tid att köra oavsett inmatning.

Material och Metod

Kodanalys

Det här stycket presenterar min implementation av algoritmerna i Java. IDE som användes var VS code eftersom jag hade tidigare erfarenheter om det och koden sparades på github. Programmets funktion är att kryptera 20 olika texter med de valda chifferna och testar sedan att attackera meddelanden samtidigt som data samlas in från programmet. Koden gör detta genom en simpel huvudstruktur med tre klasstyper, vilka är chifferklasser, attackklasser och utilityklasser. De två första är baserat på interfaces, vilket gör att en stor del av komplexitetet kan undangömas. Chifferklasser har alla enc, dec och setKey metoder, medan attackklasserna har en attackmetod per chiffer. Varje chiffer och attack har sin egen klass. Utility klasser är extra klasser som ger koden en bättre struktur där de två viktigaste är main- och statistic-klasserna men även Utility.java som innehåller algoritmer som återanvänds flertalet gånger i olika delar av koden.

För djupare förklaring av klasserna se Appendix 1 och för koden se Appendix 2 eller min github-sida: https://github.com/Miiroun/GA.

Urval och avgränsningar

Chifferena valdes ut för att få en relativt bred sammansättning av chiffertyper men även hålla komplexiteten låg. Attackerna valdes eftersom de både är relativt universella och “endast chiffertext” (ciphertext only attack) baserat vilket är både simplaste, mest kända attacktypen och applicerbar på alla chiffer. Eftersom alla implementerade attacker är baserade på att endast veta chiffertexten kan fler paralleller dras även ifall detta inte var fallet.

Resultat

Attacktider

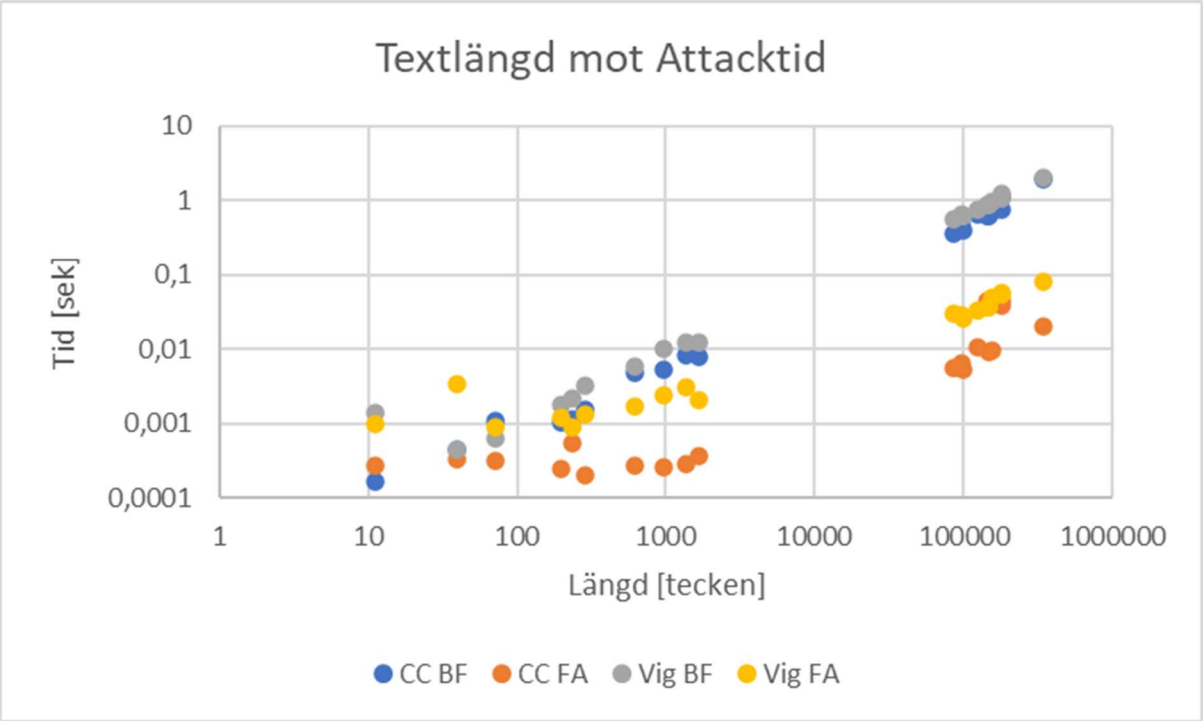
Tabell 4 Attacktider i Sekunder för olika textlängder

length	CC BF	CC FA	Vig BF	Vig FA
182864	1,134601	0,038789	1,213683	0,056448
146704	0,60808	0,044934	0,884944	0,03726
100554	0,398451	0,005367	0,610559	0,026159
149062	0,625313	0,009197	0,870184	0,037192
125304	0,660261	0,010774	0,739756	0,03305
182094	0,738056	0,045489	1,048118	0,05506
345093	1,88258	0,019948	2,038782	0,081504
98112	0,445673	0,006475	0,630934	0,028
86344	0,359109	0,005521	0,556865	0,02962
156888	0,720719	0,00962	0,943868	0,049193
1657	0,007889	0,000368	0,011993	0,002064
612	0,004817	0,000275	0,00581	0,001692
1377	0,008391	0,000287	0,012445	0,003093
286	0,001499	0,000202	0,003219	0,001351
964	0,005259	0,000255	0,010263	0,002429

39	0,00044	0,000327	0,000437	0,003442
199	0,001024	0,000243	0,00175	0,001203
237	0,001159	0,00053	0,002154	0,000907
71	0,001059	0,000309	0,000632	0,000891
11	0,000169	0,000278	0,001391	0,00098

Som ses i tabell ovan är värdena för både längden och tiden i olika storleksordningar och därför skulle ett linjärt diagram inte innefatta alla värden på ett meningsfullt sätt. Därför har det valts att presentera dem i en logaritmisk skala (med bas 10).

Diagram 5 Textlängd mot attacktider i logaritmisk skala



Eftersom vi inte lyckades hitta en effektiv algoritm för bryta kolumn transpositionschiffret så inkluderas inte dess värden här. Vi kan utläsa från detta diagram att frekvensanalys är mer generellt sätt mer effektivt än brute force, men om man attackerar ett polyalfabetiskt chiffer så tar detta mer tid än ett monoalfabetiskt chiffer av samma längd. Detta gäller dock inte för mycket korta texter eftersom attackerna då missade hitta rätt sluttext, detta gäller särskilt för

vigenérechiffer som i denna studie effektivt sätt arbetar med fyra stycken fjärdedel så lång chiffertext av motsvarande caesarchiffer text.

Enkryptions och dekryptionstid

Diagram 6 Enkryptionstiden för olika chiffer i logaritmisk skala

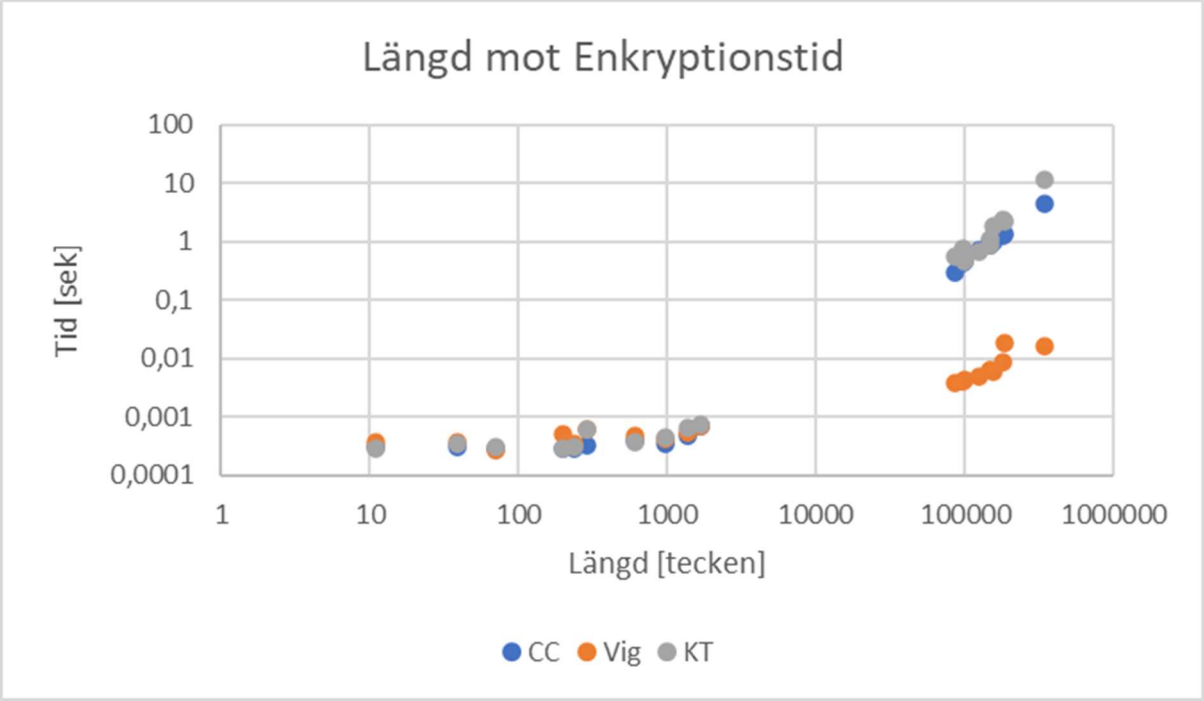
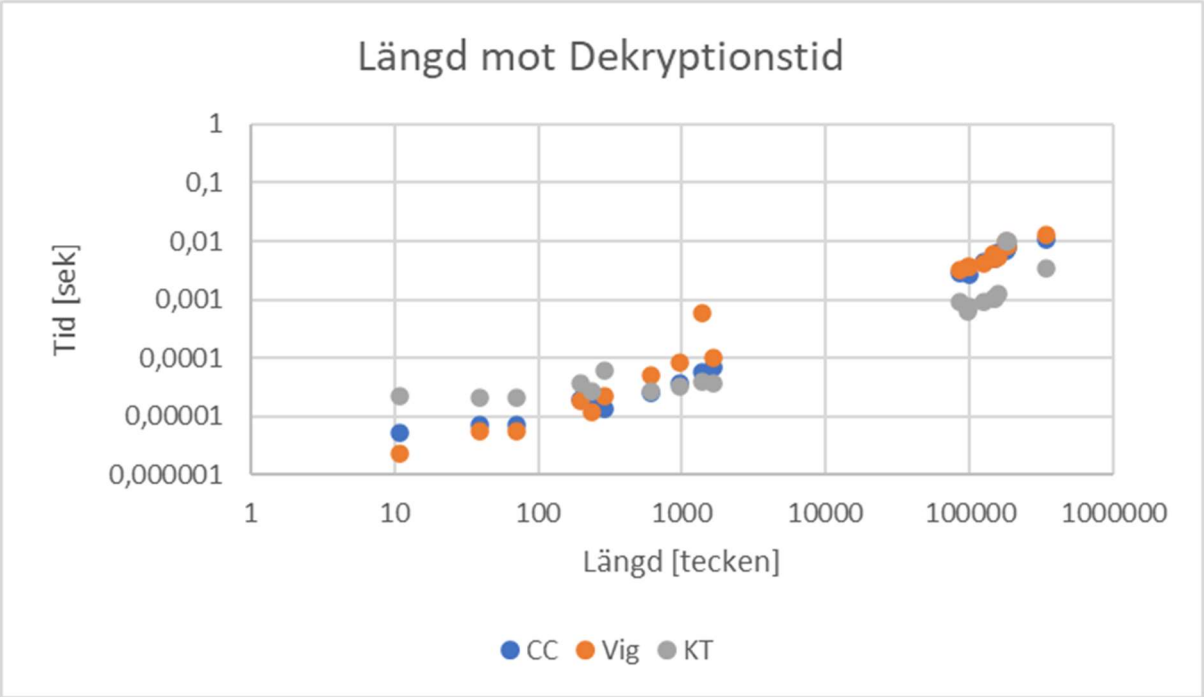


Diagram 7 Dekryptionstiden för olika chiffer i logaritmisk skala

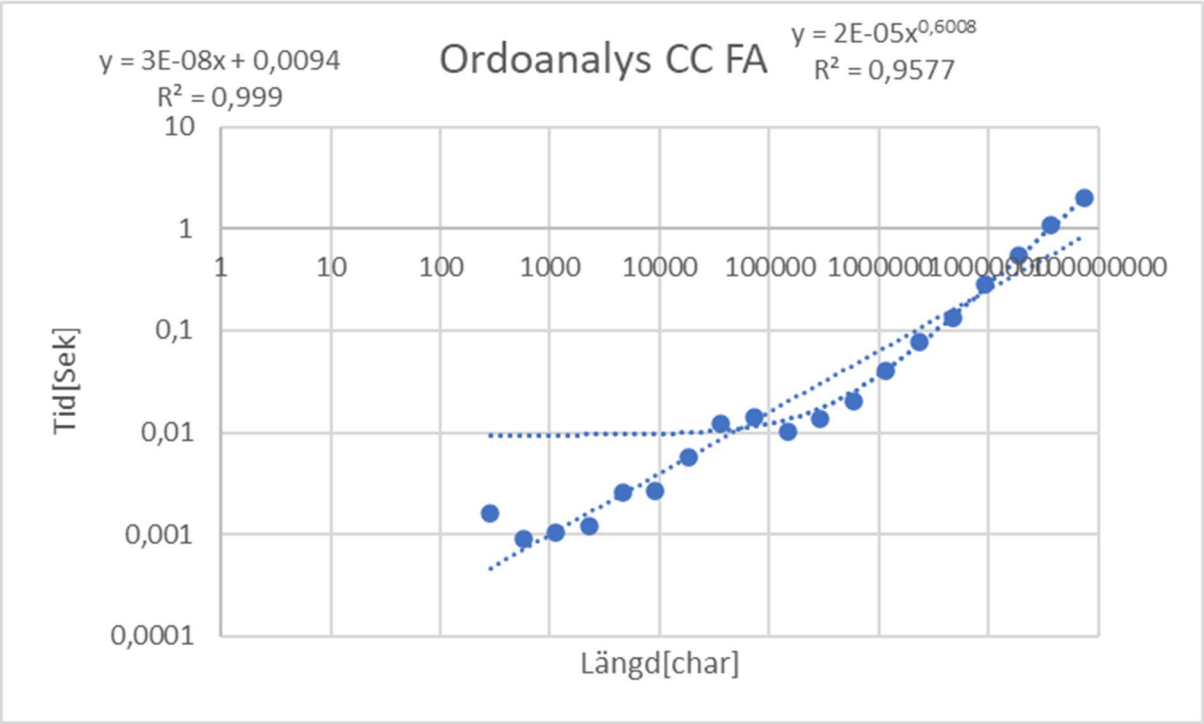


Tolkning av enkryptions och dekryptios data ger att tiden är mycket nära varandra, förutom för längre texter av enkryption, detta tyder på att det är andra delar av koden som tar tid och mäts i dessa steg än den rena krypteringen vilket orsakades av en bug i koden.

För rådata resultaten se appendix 3.

Komplexitet analys

Diagram 8 Ordoanalys



För data se appendix 4.

Attaktiden för frekvensanalys av ceasarchiffer tolkas som proportionell mot teckenlängden och har därför en komplexitet på O(n). Detta eftersom regressionen hade ett R^2 värde på 0,999 för den linjära regressionen vilket överensstämmer med teorin.

Diskussion

Slutsats

Sambandet mellan teckenlängd och tid är linjär för samtliga uppmätt data och alla algoritmer. Dessa har en liknande konstantterm men konstanttermen skiljer dem åt, men denna blir enbart signifikant efter ca 10´000 tecken. Ett monoalfabetiskt chiffer är inte mycket effektivare än ett polyalfabetsikt (caesar och vigenère) men träffsäkerheten är signifikant högre för kortare texter.(Se texter 15 16 & 18 för CC och Vig utsatta med FA). För längre texter är frekvensanalys ca 100 gånger effektivare än brute force för motsvarande text men för kortare texter är det ingen större skillnad för attacktider och osäkerheten är relativt hög. Frekvensanalys var även något träffsäkrare än brute force (Jämför BF och FA för text 17 vigenérechiffer). Alltså är frekvensanalys en bättre algoritm än brute force för de chiffer det är applicerbar på och Vigenérechiffer är i de flesta fall ekvivalent med caesarchiffer. De absolut säkraste är dock transpositionschiffer, eftersom det ej lyckades brytas i denna studie.

Metoddiskssion

Koden är relativt effektiv men kan ha problem och borde inte tas som standard för kodbrytning. Den är exempelvis kodat i Java och inte C, ett lägre språk skulle ökat algoritmens hastighet, samt effektivt använde av bitmanipulering och pointers kan antagligen öka algoritmens effektivitet flertalet gånger men kompetensen för detta saknades i studien.

Valet av att koda algoritmerna själv och inte använda en färdig implementation har tagit mycket tid men har givit bättre data samt en ökad förståelse av algoritmerna. Ett utmanande del av projektet var fitnessfunktionen. Detta eftersom terminologin är otydlig och implementationen försvårades markant av behovet av utomstående data och abstraktion av data vilket behövdes göras. Valet av användningen av ngram frekvens istället för ordboksuppslagning har givit algoritmen lägre träffsäkerhet men valdes bort då optimeseringarna nödvändiga för att få ordboksuppslagning att fungera effektivt bedömdes för svåra att implementera.

Källkritisk Diskussion

Då chiffer och algoritmer är ett område fokuserat runt dator finns mycket information koncentrerat på internetet, därför hämtades mycket av grundläggande informationen därifrån, (särskilt sidan crypto-it.net). Även mycket information om hur man kodar effektivt och löser vissa buggar hämtades från webben. När rapporten sedan skrevs så jämfördes informationen på dessa sidor med den från boken Kodboken av Simon Sight. Problemet med denna bok är att den är 25 år gammal och datasäkerhet är ett nytt ständigt uppdaterande ämne, även ifall algoritmerna diskuterade i rapporten är flera hundra år gamla. Så för att ge ett nutida perspektiv granskades internetsidorna som behövde användas mot varandra till den grad detta var möjligt.

Felkällor

Resultatet har stora felmarginaler för speciellt de kortare texterna. Detta hade kunnat lösas genom att upprepa experimentet fem gånger och ta medelvärdet av resultaten. De kan även bero på olika bakgrundsprogram som kördes på datorn under tiden data samlades in.

Ytterligare en felkälla är implementationen i koden. Efter data redan hade processas upptäcktes ett problem i implementationen av caesarchiffer, med en så kallad “unsafe conversion between char and string”. Detta borde inte påverka andra delar av koden än enkryption men liknande fel kan finns på andra ställen i koden.

Vidare forskning

För vidare forskning kan studier undersöka andra substitution och transpositionschiffer av liknande komplexitet och se hur säkra de är mot våra presenterade attacker. Man hade även kunnat studera hur säkra de är mot attacker av andra attacktyper (Dictionary attack, Known-plaintext attack, Chosen-key attack, Chosen-ciphertext attack).

Man kan även studera tekniker för att detektera vilket chiffer som har använts på en text.

Vidare studier kan även fokusera på att skapa en mer komplett bild av attackernas tidskomplexitet (Big O) genom att öka definitionsmängden för algoritmen men även testa detta mot ett transpositionschiffer.

Källförteckning

Digitala källor

1. Garg, Vibhav. https://medium.com/analytics-vidhya/how-to-distinguish-between-gibberish-and-valid-english-text-4975078c5688 2021-04-14. [2024-02-16]
2. Kowalczyk, Chris. https://www.crypto-it.net/eng/simple/index.html. 2020-03-09. [2024-01-15]
3. Nilsson, Stefan. https://www.csc.kth.se/utbildning/kth/kurser/DD1341/inda11/algorithms/asymptoter/. 2012-03-04. [2024-03-20]
4. Rifkin, Jeremy.https://felisphasma.github.io/FrequencyAnalysisOfNgrams/. 2017. [2024-02-16]
5. Timur. https://planetcalc.com/8045/. 2021. [2024-02-16]

Tryckta källor

1. Singh, Simon. Kodboken. Övers. Brogen Margareta. Nordstedt förlag.

Texter som analyserats

1. Långa texter
    a. Karin Boye:http://runeberg.org/
    b. August Strindberg. Carl J,L, Almqvist: https://litteraturbanken.se/txt/lb1722437/lb1722437.pdf
    c. Shakespear:https://www.folger.edu/explore/shakespeares-works/
    d. Charles Dickens: https://www.gutenberg.org/cache/epub/46/pg46.txt
2. Korta texter och utdrag
    a. LordOfTheRing:https://edgestudio.com/script/lord-of-the-rings-opening-monologue/
    b. TongueTwisters:https://www.engvid.com/english-resource/50-tongue-twisters-improve-pronunciation/

c. IHaveADream:https://historiskamedia.se/artiklar/i-have-a-dream-martin-luther-kings-berorande-tal/
d. TwoCities:https://lithub.com/how-many-of-the-100-most-famous-passages-in-literature-can-you-identify/
e. 1984:https://www.penguinrandomhouse.ca/books/326569/1984-by-george-orwell/9780735234611/excerpt
f. WaltDisney: https://blog.hubspot.com/sales/famous-quotes

nGram frekvenser källa

1. http://practicalcryptography.com/cryptanalysis/letter-frequencies-various-languages/

Användbara verktyg

1. https://cryptii.com/pipes/binary-to-base64
    a. Mycket bra när jobbar på bitnivå
2. https://encode-decode.com/des-encrypt-online/
    a. Ett sätt att pröva om ens algoritmer ger rätt chiffertext
3. https://www.random.org/strings/?num=1&len=32&loweralpha=on&unique=on&format=html&rnd=new
    a. Ett användbart verktyg för slumpmässiga bokstavsrader.

Appendix 1 - Förklaring av koden

Mainklassens funktion är att läsa in texter och hålla huvud-loopen. Den använder två enum:s för att bestämma vilket chiffer- och attack som ska användas.

Frekvensanalys använder en MAP för (Character, Integer) som har ett entry för alfabetets alla bokstäver. Funktionen loopar igenom hela texten och mäter upp frekvensen av alla karaktärer och beräknar sedan den relativa frekvensen. Detta leder vidare till referensvärden som jämförs, när attacken gått igenom skickas nyckeln genom dekryptionsalgoritmen. För vig chiffer antar vi att längden är fyra, då tiden saknades att implementera en metod för nyckellängdbestämning.

Brute force fungerar genom att skapa en lista av möjliga nycklar och testar dem alla. För att avgöra vilket meddelande som är det korrekta används fitness funktionen quadgram. Även här antas vigenerchifferlängden vara fyra då metoden ej är implementerad. Då antalet nycklar blir 29^n för vigenère så är detta inte en rimlig metod för längre nycklar.

Caesarchiffer använder en lista av alfabetet för att förskjuta bokstäver. Det finns även en annan variant i koden som ej används där vi adderar 1 till integer representationen av karaktären men problemet var att vi då fick icke rimliga tecken som output.

Vigenère delar på texten i fyra delar och sedan återanvänder caesarchiffret för själva kryptionen.

Komplexitetsanalysen är ett liknande experiment till attackanalysen men samma text har tagits och adderats till sig själv sådant att varje steg precis dubblar den tidigare längden. Detta gör att vi får mer välspridd data i det område vi önskar. Målet var att se mer exakt hur attacktiden är beroende av karaktärs längden på chiffertexten och skulle därför gå uppemot 1000 sekunders texter som längst, problemet var att programmet stötte på minnesproblem dessförinnan och kunde inte slutföra mer än texter på 1 sekund.

Tabell 5 Följande förkortningar används i koden.

Förkorting	Förklaring

Caesarchiffer	CC
Vigenérechiffer	Vig
Kolumntranspositonschiffer	CT (efter engelska columnar transposition cipher)
Brute Force	BF
Frekvensanalys	FA
Enkryption	enc
Dekryption	dec

Appendix 2 -koden

Main

```
package Other;

import java.io.File;
import java.io.FileInputStream;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Paths;

import Chiffers.CaesarCipher;
import Chiffers.ColumnarTransposition;
import Chiffers.Substitution;
import Chiffers.X_OR;
import Chiffers.AdvancedEncryption.DES;
import Chiffers.AdvancedEncryption.DiffieHellman;
import Chiffers.AdvancedEncryption.TriDES;
import Other.Interfaces.AttackInterface;
import Other.Interfaces.ByteEncrytionInterface;
import Other.Interfaces.StringEncrytionInterface;
import attacks.BruitForce;
import attacks.FrequencyAnalysis;

public class Main {

    public enum EncStandard {
        DES,
        TriDES,
        DH,
        CC,
        Vig,
        Sub,
        CT
    }

    public enum AttStandard {
        FA,
        BF,
        not
    }
```

```
    public static EncStandard encStandard;
    public static AttStandard attStandard;
    public static boolean doBytes; // else work with strings
    public static int blockSize = 64;
    public static int byteSize = blockSize / 8;
    public static int blockCount;
    public static Charset charset = Charset.forName("UTF-8");

    public static void varibleSetUp(EncStandard encStand, AttStandard
attStand) {
        encStandard = encStand;
        attStandard = attStand;

        if (encStandard == EncStandard.DES || encStandard ==
EncStandard.TriDES || encStandard == EncStandard.DH) {
            doBytes = true;
            blockSize = 64;
            byteSize = blockSize / 8;
            charset = Charset.forName("UTF-8");
        } else if (encStandard == EncStandard.CC || encStandard ==
EncStandard.Sub ||  encStandard == EncStandard.Vig || encStandard ==
EncStandard.CT) {
            doBytes = false;
            charset = Charset.forName("UTF-8");

        }


    }

    // setup
    public static byte[] inputArray;
    public static byte[] outputArray;
    public static byte[] keyArray;

    public static String inputString;
    public static String outputString;
    public static String keyString;

    public static byte[] readData(String path, boolean isText) {

        byte[] inputArray = new byte[0];
```

```
        // read file
        try {
            File file = new File(path);
            byte[] bytes = new byte[(int) file.length()];
            try (FileInputStream fis = new FileInputStream(file)) {
                fis.read(bytes);
            }
            inputArray = bytes;
            if (isText) {
                String input = new String(inputArray, charset);
                System.out.println("Input" + ":" + input);
            }
        } catch (Exception e) {
            System.out.println("An error '" + e + "' has occurred.");
        }
        return inputArray;
    }

    public static String readData(String path){
        String data = "";
        try{
            data = new String(Files.readAllBytes(Paths.get(path)),
charset);
        } catch (Exception e) {
            System.out.println("An error '" + e + "' has occurred.");
        }
        return data;

    }

    public static void writeData(String data, String path) {
        try{
            Files.writeString(Paths.get(path), data);
        } catch (Exception e) {
            System.out.println("An error '" + e + "' has occurred.");
        }
    }

    public static void writeData(byte[] byteArray, String path, boolean
isText) {

        try {
            Files.write(Paths.get(path), byteArray);
```

```
            if (isText) {
                String output = new String(byteArray, charset);
                System.out.println("Output:" + output);
            }
        } catch (Exception e) {
            System.out.println("An error '" + e + "' has occurred.");
            e.printStackTrace();
        }

    }

    public static byte[][] seperateIntoBlocks() {

        byte[][] blocks = new byte[blockCount][];

        int i = 0;
        int j = byteSize;
        while (i < blockCount) {
            byte[] data = new byte[byteSize];
            if (i == blockCount - 1) {
                j = inputArray.length + byteSize - outputArray.length;
            }
            System.arraycopy(inputArray, i * byteSize, data, 0, j);

            blocks[i] = data;

            i = i + 1;
        }
        return blocks;
    }

    public static void computeBlock(byte[] block, int i,
ByteEncrytionInterface encClass, boolean encrypt) {
        byte[] data = block;

        if (encrypt) {
            data = encClass.enc(data);
        } else {
            data = encClass.dec(data);
        }

        System.arraycopy(data, 0, outputArray, i * byteSize, byteSize);
```

```
    }

    public static void doCryotionBlocks(boolean encrypt) {
        blockCount = Math.floorDiv((inputArray.length - 1), byteSize) +
1;
        outputArray = new byte[blockCount * byteSize];
        byte[][] blocks = seperateIntoBlocks();
        ByteEncrytionInterface encClass;

        if (encStandard == EncStandard.DES) {
            encClass = new DES();
        } else if (encStandard == EncStandard.TriDES) {
            encClass = new TriDES();
        } else if (encStandard == EncStandard.DH) {
            encClass = new DiffieHellman();
        } else {
            // error, should never be here
            encClass = new DES();
        }

        for (int i = 0; i < blocks.length; i++) {
            computeBlock(blocks[i], i, encClass, encrypt);
        }
    }

    public static void doCryption(boolean encrypt) {
        StringEncrytionInterface encClass;

        if (encStandard == EncStandard.CC) {
            encClass = new CaesarCipher();
        } else if (encStandard == EncStandard.Vig) {
            encClass = new X_OR();
        } else if (encStandard == EncStandard.Sub) {
            encClass = new Substitution();
        }else if (encStandard == EncStandard.CT) {
            encClass = new ColumnarTransposition();
        } else {
            // error, should never be here
            encClass = null;
        }

        encClass.setKey(keyString);
```

```
        if (encrypt) {
            outputString = encClass.enc(inputString);
        } else {
            outputString = encClass.dec(inputString);
        }

    }

    public static void doAttacking() {
        AttackInterface attClass;

        if (attStandard == AttStandard.BF) {
            attClass = new BruitForce();
        } else if (attStandard == AttStandard.FA) {
            attClass = new FrequencyAnalysis();
        } else {
            // error, should never be here
            System.out.println("something wrong with attact standard
selection");
            attClass = new BruitForce();
        }

        //

        if (encStandard == EncStandard.CC) {
            outputString = attClass.attackCC(inputString);
        } else if (encStandard == EncStandard.Vig) {
            outputString = attClass.attackXO(inputString);
        } else if (encStandard == EncStandard.Sub) {
            outputString = attClass.attackST(inputString);
        } else if (encStandard == EncStandard.CT) {
            outputString = attClass.attackCT(inputString);
        } else {
            // error, should never be here
            throw new UnsupportedOperationException("something wrong
with attact standard selection");
        }

    }
    // algorithems

    public static void encryptData() {
```

```
        System.out.println("encrypting data");

        if (doBytes) {
            inputArray = readData("data/startText.txt", true);
            keyArray = readData("data/encryptKey.txt", false);

            doCryotionBlocks(true);

            writeData(outputArray, "data/encryptedText.txt", false);
        } else {
            inputString = readData("data/startText.txt");
            keyString = readData("data/encryptKey.txt");

            doCryption(true);

            writeData(outputString, "data/encryptedText.txt");
        }

        Statistics.recordStat("");
    }

    public static void decryptData() {
        System.out.println("decrypting data");

        if (doBytes) {
            inputArray = readData("data/encryptedText.txt", false);
            keyArray = readData("data/encryptKey.txt", false);

            //
            doCryotionBlocks(false);
            //

            writeData(outputArray, "data/endText.txt", true);
        } //
        else //
        { //
            inputString = readData("data/encryptedText.txt");
            keyString = readData("data/encryptKey.txt");

            //
            doCryption(false);
            //
```

```
            writeData(outputString, "data/endText.txt");
        }
        Statistics.recordStat("");
    }

    public static void attackData() {
        if (doBytes) {

        } else {
            System.out.println("attacking data");

            inputString = readData("data/encryptedText.txt");

            doAttacking();

            System.out.println("guess of oridginal text:" +
outputString);


            if (outputString.equals(readData("data/startText.txt")
)){System.out.println("The attack got the correct answer");
            } else {System.out.println("The attack got the wrong
answer");}

            writeData(outputString, "data/attackedText.txt");
        }
        Statistics.recordStat("");

    }

    public static String repetText(String str) {
        return  str + str + str +str + str + str + str + str +str +
str;
    }

    public static String[] generateTextOfDiffrentLengths(String str,
int n) {
        String[] texts = new String[n];
        texts[0] = str;

        for (int i = 1; i < texts.length; i++) {
            //texts[i] = repetText(texts[i-1]);
            texts[i] = texts[i-1].concat(texts[i-1]);
```

```
        }
        return texts;

    }


    public static void testKrypto() {
        Statistics.startCollecting();
        Statistics.openAttEncPair();
        Statistics.openTextWork();

        varibleSetUp(EncStandard.CC, AttStandard.FA);

        System.out.println("Staring...");

        encryptData();

        decryptData();

        if(attStandard != AttStandard.not) attackData();

        System.out.println("Done!");

        Statistics.endCollecting(false);
    }

    public static void anaCharFrec () {
        FrequencyAnalysis fa = new FrequencyAnalysis();
        String str = readData("data/texts/FrakenJulie.txt");
        //CaesarCipher cc = new CaesarCipher();
        //cc.setKey("3");
        //str = cc.enc(str);

        /*char[] strArray = new char[10000];
        Random rand = new Random();
        for (int i = 0; i < strArray.length; i++) {
            strArray[i] = Utility.alphabet[rand.nextInt(29)];
        }
        String str = new String(strArray);*/

        System.out.println(fa.analysLetters(str.toCharArray()));
    }
```

```
    public static void testEvaluate() {
        String str = readData("data/texts/hamlet.txt");
        //str =
"kzuojahsnqxfqkevsfftsbpuioyehpxbkzuojahsnqxfqkevsfftsbpuioyehpxbkzuoja
hsnqxfqkevsfftsbpuioyehpxbkzuojahsnqxfqkevsfftsbpuioyehpxbkzuojahsnqxfq
kevsfftsbpuioyehpxbkzuojahsnqxfqkevsfftsbpuioyehpxbkzuojahsnqxfqkevsfft
sbpuioyehpxbkzuojahsnqxfqkevsfftsbpuioyehpxbkzuojahsnqxfqkevsfftsbpuioy
ehpxbkzuojahsnqxfqkevsfftsbpuioyehpxb";
        //str =
"TobeornottobethatisthequestionWhethertisNoblerinthemindtosufferTheSlin
gsandArrowsofoutrageousFortuneOrtotakeArmsagainstaSeaoftroublesAndbyopp
osingendthemWilliamShakespeareHamlet";
        System.out.println(BruitForce.evaluteText(str));
    }

    public static void doDataCollecting(String[] texts, int i) {
        Statistics.openAttEncPair();
        for (int j = 0; j < texts.length; j++) { //loop throu all the
texts
            Statistics.openTextWork();
            System.out.println("Started on text:" + j + ", With enc:" +
encStandard+ ", and att:" + attStandard);//

            inputString = texts[j];
            doCryption(true);
            Statistics.closeSegmentInPair();

            inputString = outputString;
            doCryption(false);
            Statistics.closeSegmentInPair();
            String tempOut = outputString;

            if(attStandard != AttStandard.not) {
                doAttacking();
            }
            Statistics.closeSegmentInPair();

            Statistics.closeTextWork(outputString.equals(tempOut), j);
            Statistics.timeStamp("Typ:"+i+"_Text: "+j);

        }
        Statistics.closeAttEncPair(i);
```

```
    }

    public static void testBigO() {
        Statistics.startCollecting();

        varibleSetUp(EncStandard.CC, AttStandard.FA);
        keyString = "3";

        String orgString = readData("data/texts/" + "TwoCitys" +
".txt");
        String[] texts = generateTextOfDiffrentLengths(orgString, 22);
//22 högsta för integer bit limit, men krashar av minnesproblem vid
n:19 redan

        doDataCollecting(texts, 0);

        Statistics.endCollecting(true);
    }

    public static void collectData() {
        Statistics.startCollecting();

        varibleSetUp(EncStandard.Sub, AttStandard.BF);

        String[] texts = new String[20];
        String[] titles = {"Hamlet", "RaJ", "MND", "WinterTale",
"MerchantVenice", // shakspear: Hamlet, Romeo and Julijet, a Midsummer
nights dream, the winter tale, the merchant of venice
         "Carol", "DetGarAn", "FrakenJulie", "Fadern", "RodaRummet"//
chrismas carol, det går an, fröken julie, Fadern, röda rummet
        , "LoTR", "Tradet", "IHAD", "TwoCitys", "1984" //  5 middle
short text: lord of the ring opening monolog, trädet av Boye, utdrag ur
i have a dream, utdrag ur a tale of two citys, utdrag av utdrag av 1984
        , "ToBE", "PP", "BB", "WD", "HW" // 5 realy short texts: to be
or not to be, peter pipper; toung twister, start doing by wuiting-
Disney, Betty botter; toung twister, hellow world
        };

        for (int i = 0; i < 20; i++) {
            texts[i] = readData("data/texts/" + titles[i] + ".txt");
            //System.out.println(titles[i] +":" + texts[i].length());
        }
```

```
        for (int i = 0; i < 5; i++) {// looop throw encryption and
attack pattarns
            if(i == 5){
                attStandard = AttStandard.not;
            }else if(i % 2 == 0) {
                attStandard = AttStandard.FA;
            } else if(i % 2 == 1){
                attStandard = AttStandard.BF;
            }

            if(i == 0 || i == 1) {
                encStandard = EncStandard.CC;
                keyString = "3";
            } else if(i == 2 || i == 3 ) {
                encStandard = EncStandard.Vig;
                keyString = "data";
            } else if(i == 4 || i == 5 ) {
                encStandard = EncStandard.CT;
                keyString = "swindon";
            }


            doDataCollecting(texts, i);
        }

        //record data about my messages
        Statistics.endCollecting(true);
    }



    public static void main(String[] args) {
        //testKrypto();

        //anaCharFrec();
        //testEvaluate();

        //collectData();
        testBigO();
    }

}
```

Frekvensanalys

```
package attacks;

import java.util.Map;
import java.util.Random;
import java.util.TreeMap;

import Chiffers.CaesarCipher;
import Chiffers.ColumnarTransposition;
import Chiffers.Substitution;
import Chiffers.X_OR;
import Other.Utility;
import Other.Interfaces.AttackInterface;

import static java.util.Map.entry;

import java.util.ArrayList;

public class FrequencyAnalysis implements AttackInterface{

    public Map<Character, Integer> sweFrequency = Map.ofEntries(
        entry('a', 720),
        entry('b', 120),
        entry('c', 180),
        entry('d', 360),
        entry('e', 1120),
        entry('f', 180),
        entry('g', 170),
        entry('h', 610),
        entry('i', 620),
        entry('j', 10),
        entry('k', 90),
        entry('l', 440),
        entry('m', 290),
        entry('n', 590),
        entry('o', 800),
        entry('p', 140),
        entry('q', 10),
        entry('r', 560),
        entry('s', 600),
        entry('t', 870),
        entry('u', 300),
```

```
        entry('v', 80),
        entry('w', 210),
        entry('x', 10),
        entry('y', 220),
        entry('z', 10),
        entry('å', 10),
        entry('ä', 10),
        entry('ö', 10)
    );

    public Map<Character, Integer> analysLetters(char[] wordlist) {
        Map<Character, Integer> frequency = new TreeMap<>();
        //for (char c : Utility.alphabet)
{frequency.put(Character.toLowerCase(c), 0);} // sets each letter to
zero
        //for (char c : Utility.nummbers)
{frequency.put(Character.toLowerCase(c), 0);} // sets each nummber to
zero
        //for (char c : Utility.signs)
{frequency.put(Character.toLowerCase(c), 0);} // sets each sign to zero

        //

        for (char cha: wordlist) {
            char chaL = Character.toLowerCase(cha);
            Integer num = frequency.get(chaL); if(num == null) {num =
0;}

            frequency.put(chaL, num + 1);
        }

        //refaktor code to be in promille
        int size = 0; //wordlist.length ;//- 6079 - 28050 -742 -3193 -
442 - 2098;//- frequency.get(' ') - frequency.get('\n');
        for (char c : Utility.alphabet) {Integer frec =
frequency.get(Character.toLowerCase(c)); if (frec != null){size +=
frec;}} //counts total amout of letters

        for ( int i = 0; i < Utility.alphabet.length; i++) {
            Character cha = Utility.alphabet[i];
            Integer in = frequency.get(cha); if (in == null){in = 0;}
            int x = Math.round(((in * 10000 / size)));
            //x = in;
```

```
            frequency.put(cha, x);
        }

        return frequency;
    }

    public int analysCharArray(char[] charArray) {
        Map<Character, Integer> frequency = analysLetters(charArray);

        int lowestDisplasment = Integer.MAX_VALUE;
        int bestMatch = -1;
        for(int i = 0; i < 29; i++) //i = diffrent keys
        {
            int currentDis = 0;
            for(int j = 0; j < 29; j++)
            {
            char absChar = Utility.alphabet[(j) % 29];
            char locChar = Utility.alphabet[(j + i) % 29];
            int dis = Math.abs(sweFrequency.get(absChar) -
frequency.get(locChar));
            currentDis += dis;
            //System.out.println(dis);

            }
            if ( currentDis < lowestDisplasment) {lowestDisplasment =
currentDis;bestMatch = i;}
        }

        return bestMatch;
    }

    public String[] genArrayListFromCharComb(ArrayList<Character>[]
charComb, int i, String keys[]) {
        String[] tempList = keys;




            String[] prior = new String[1];
            if(i != 0){
                prior = genArrayListFromCharComb(charComb, i - 1,
keys); //not entierly correct, needs ti fix somthing
```

```
            }
            if(i == 29) {
                return prior;
            }
            tempList = new String[prior.length * charComb[i].size()];
            int j = 0;
            for (char c: charComb[i]) {
                for(int k = 0; k < prior.length; k++) {
                    tempList[j] = prior[j % prior.length] + c;
                    j++;
                }
            }

            //how should this work??


        return tempList;
    }

    public String attackCC(String data) {
        CaesarCipher cc = new CaesarCipher();

cc.setKey(Integer.toString(analysCharArray(data.toCharArray())));
        return cc.dec(data);
    }



    @Override
    public String attackST(String data) {
        char[] charArray = data.toCharArray();
        Map<Character, Integer> frequency = analysLetters(charArray);
        ArrayList<Character>[] charComb = new ArrayList[29];

        float percent = 20f;
        for (int i = 0; i < 29; i++) {
            char letter = Utility.alphabet[i];
            int value = frequency.get(letter); //problem här att blir
0, kanske fel från datan
            charComb[i] = new ArrayList<Character>();

            for (char answerLetter  : Utility.alphabet) {
                int answerValue = sweFrequency.get(answerLetter);
                if (value * (1f - percent) <= answerValue )
```

```
                    if(answerValue <= value * (1f + percent)){
                    {
                        charComb[i].add(answerLetter);
                    }
                }
            }

        }

        String[] keys = genArrayListFromCharComb(charComb, 29, new
String[1]); //problem med skapandet av nycklar


        Substitution st = new Substitution();
        String message[] = new String[keys.length];
        int j = 0;
        for (String key : keys) {
            st.setKey(key);
            message[j] = st.dec(data);
            j++;
        }

        return BruitForce.evaluteMessageArray(message);


    }

    @Override
    public String attackXO(String data) {
        char[] charArray = data.toCharArray();
        char[][] subArray = new char[0][];

        X_OR xo = new X_OR();

        char[] keyStr;

        aa:{//chould make it generic for length > 4
            int keyLen = 4;//length
            subArray = new char[keyLen][];
            for (int i = 0; i < subArray.length; i++) {
                int rounded = Math.floorDiv(charArray.length, keyLen)+
1;
                subArray[i] = new char[rounded];
```

```
            }

            for (int i = 0; i < subArray[0].length * keyLen; i++)
            {
                char c = charArray[Math.min(i, charArray.length-1)];
                subArray[i % keyLen][Math.floorDiv(i, keyLen)] = c;
            } //create strings than inclued every keyLenth character

            int[] values = new int[keyLen];
            for (int i = 0; i < subArray.length; i++) {
                values[i] = analysCharArray(subArray[i]);
            }

            keyStr = new char[keyLen];
            for (int i = 0; i < keyLen; i++){keyStr[i] =
Utility.alphabet[values[i]];}
            xo.setKey(new String(keyStr));
            if(BruitForce.evaluteText(xo.dec(data)) > 7) {break aa;}
//this check would be helpful if wanted to do for more than length 4
        }


        return xo.dec(data);
    }


    @Override
    public String attackCT(String data) {
        //do an (not anagram) attack here
        int[] div = Utility.dividers(data.length());
        ColumnarTransposition ct = new ColumnarTransposition();

        int[] values = new int[div.length];
        for (int i = 0; i < div.length; i++) {
            int[] key = new int[div[i]];
            for (int j = 0; j < key.length; j++) key[j] = j;
            ct.setKey(key);

            char[][] matrix = ct.createMatrixDec(data);

            values[i] = 0;
            for (int j = 0; j < matrix.length; j++) {
                String subMessage = new String(matrix[j]);
```

```
                values[i] += BruitForce.evaluteText(subMessage);
            } //maybe not evalute inside forloop but as a whole



        }
        int k = -1;
        int bestMatch = Integer.MIN_VALUE;
        for (int i = 0; i < values.length; i++) {
            if(values[i] > bestMatch) {
                k = i;
                bestMatch = values[i];
            }
        }

        //loop throw to check which key permutation is best
        int keyLen = div[k];

        //cheats and tells system current length, since above algorithm
not acuret enouth and throws errors
        keyLen = 2;
        int[] key = new int[keyLen];
        for (int j = 0; j < keyLen; j++) { key[j] = j;} //om blir för
bred kolumn blir det myckeyt eftersom n! långl, problemet här är att
överstrider integer bit limit

        if(Utility.factorial(keyLen) < 1000) {
            int[][] permutations = Utility.permutations(key);
//alldeles för långsamt att testa all möjliga permutationer,

            String[] message = new String[permutations.length];
            for (int i = 0; i < message.length; i++) {
                ct.setKey(permutations[i]);
                char[][] matrix = ct.createMatrixDec(data);
                message[i] = ct.decWithMatrix(matrix);
            }

            return BruitForce.evaluteMessageArray(message);
        } else {
            int[] lastKey = key.clone();
            double lastValue = Integer.MIN_VALUE;

            Random r = new Random();
```

```
            for (int i = 0; i < 100000; i++) {
                int[] nowKey = lastKey.clone();
                int rand1 = r.nextInt(keyLen-1);
                int rand2 = r.nextInt(keyLen-1);
                nowKey[rand1] = lastKey[rand2];
                nowKey[rand2] = lastKey[rand1];
                ct.setKey(nowKey);
                double nowValue = BruitForce.evaluteText(ct.dec(data));
                if(nowValue > lastValue) {
                    lastValue = nowValue;
                    nowKey = lastKey;
                }
            }

            ct.setKey(lastKey);
            return ct.dec(data);
        }
    }
}
```

Brute force

```
package attacks;

import java.math.BigInteger;

import Chiffers.CaesarCipher;
import Chiffers.ColumnarTransposition;
import Chiffers.Substitution;
import Chiffers.X_OR;
import Other.Main;
import Other.QuadGram;
import Other.Statistics;
import Other.Utility;
import Other.Interfaces.AttackInterface;

public class BruitForce implements AttackInterface {
    public static QuadGram swe4Grams;

    // atempts at DES
    public static void attackDES() {
```

```
        System.out.println("decrypting data");

        byte[] answer = Main.readData("data/startText.txt", false);
        Main.inputArray = Main.readData("data/encryptedText.txt",
false);
        Main.keyArray = new byte[8];

        for (int i = 0; i < Math.pow(2, 64); i++) {
            BigInteger bigInteger = BigInteger.valueOf(i);
            Main.keyArray = bigInteger.toByteArray();
            Main.doCryotionBlocks(false);

            if (Main.outputArray == answer) {
                System.out.println("found you");
            }

            if (i % 100000 == 0) {
                System.out.println(i);
            }
        }

        Main.writeData(Main.outputArray, "data/endText.txt", true);
        Statistics.recordStat("");

    }

    public double evaluateNGram(String data) {
        Statistics.addDataCount("evaText", 1);
        // not yet implemented

        if (swe4Grams == null){
            swe4Grams = new QuadGram();

swe4Grams.readNGrams("data/nGrams/swedish_quadgrams.txt");}


        String[] Grams = QuadGram.genNGram(data);
        QuadGram nGramValues = new QuadGram();
        nGramValues.countNGrams(Grams);

        double value = QuadGram.compareNGrams(swe4Grams, nGramValues,
Grams.length);
```

```
        return value;
    }

    public double evaluteIOC(String data) { //index of coincidence
        //does not work for transposition chifers, can just be used to
tell that been incrypted by substitution
        int alfLength = 29; // sometimes mutliply the ioc by alfabet
length (26 or 29)to get "nicer" nummers to work with, differ depeding
on source

        double sum = 0d;
        long n = data.length();
        char[] letterarray = data.toCharArray();
        int[] letterCount = new int[29];

        for (int i = 0; i < letterarray.length; i++){
            aa:{for(char c : Utility.alphabet){if(letterarray[i] ==
c)break aa;}letterarray[i] = 'q';}} //if char:i not in letterArray then
replace with q
        for (char c : letterarray) letterCount[Utility.indexOf(c)] +=
1;

        for (int i = 0; i < letterCount.length; i++){
            long mult = Math.multiplyExact(letterCount[i],
(letterCount[i] -1));
            sum += mult;
        }

        long div = Math.multiplyExact(n, n-1);
        return sum *(alfLength) / div ;
    }


    public static double evaluateQGProb(String data) {//Quadgram but
with proberbility
        if (swe4Grams == null){
            swe4Grams = new QuadGram();

swe4Grams.readNGrams("data/nGrams/swedish_quadgrams.txt");}

        String[] grams = QuadGram.genNGram(data.toLowerCase());
        double value = 0d;
```

```
        long length = swe4Grams.getLength();
        double log2 = Math.log10(length);
        for(String gram : grams) { //proberbly should implement
chaching, this is far too slow
            long standVal = swe4Grams.getNgramValue(gram);
            double prob;
            if(standVal != 0) {
                double log1 = Math.log10(standVal);
                prob = log1 - log2;
                //log ( val / (length) );
            } else {
                prob = 0;
            }
            value += prob;
            //value += Math.abs(prob);
        }

        value = Math.abs(value);
        double normalValue = (value)/grams.length;
        double removedSigns = -50; int k = 5;
        if(grams.length != 0)removedSigns = (k * ((4 * grams.length ) -
data.length()) )/ (grams.length);
        return normalValue + removedSigns;
    }

    public static double evaluteText(String data) {
        Statistics.recordStat("UseMesEva");
        //return evaluateNGram(data);
        //return evaluteIOC(data);

        double value = evaluateQGProb(data);
        return value;
    }

    public static int evaluteMessageArrayBestMatch(String message[]) {
        //problem om text som bara tre bokstäver långa och att message
blir förflyttade för transposition när körs flera gånger

        int bestMatch = -1;
        double bestValue = Double.MIN_VALUE; //change sign if focus on
low
        for (int i = 0; i < message.length; i++) {
```

```
            double value = evaluteText(message[i]);
            if(value >= bestValue) {// change sign if focus on low
                bestMatch = i;
                bestValue = value;
            }
        }

        if (bestMatch ==-1) {
            if(message[0].length() > 4) {
                System.out.println(message.length);
                throw new java.lang.NullPointerException("something
wrong with messageevaluation");
            } else {
                System.out.println("sub message length is less than 4
and quadgramprobibility doesnt work");
                bestValue = 0d;
                bestMatch = 0;
            }
        }
        return bestMatch;
    }

    public static String evaluteMessageArray(String message[]) {
        int bestMatch = evaluteMessageArrayBestMatch(message);
        return message[bestMatch];
    }

    public int attackCCforBestMatch(String data) {
        final int testLength = 29;
        String[] message = new String[testLength];
        CaesarCipher cc = new CaesarCipher();

        for (int i = 0; i < testLength; i++) {
            cc.setKey(Integer.toString(i- testLength));
            message[i] = cc.dec(data);
        }
        return evaluteMessageArrayBestMatch(message);
    }


    public String attackCC(String data) {
        final int testLength = 29;
        String[] message = new String[testLength];
```

```
        CaesarCipher cc = new CaesarCipher();

        for (int i = 0; i < testLength; i++) {
            cc.setKey(Integer.toString(i- testLength));
            message[i] = cc.dec(data);
        }

        return evaluteMessageArray(message);
    }

    @Override
    public String attackST(String data) {
        int key[] = new int[29];
        for (int i = 0; i < key.length; i++) key[i] = i+1;
        int keys[][] = Utility.permutations(key);


        Substitution sub = new Substitution();
        String[] message = new String[Utility.factorial(29)];
        int k = 0;//im breacking integer limit, finns 8*10^30
möjligheter
        for (int[] is : keys) {
            String inChar = "";
            for (int i = 0; i < is.length; i++) inChar +=
Utility.alphabet[is[i]];
            sub.setKey(inChar);
            message[k] = sub.dec(data);
            k++;

        }


       return evaluteMessageArray(message);
    }

    @Override
    public String attackXO(String data) {
        char[] charArray = data.toCharArray();
        char[][] subArray = new char[0][];

        X_OR xo = new X_OR();

        char[] keyStr;
```

```
        aa:{//chould make it generic for length > 4
            int keyLen = 4;//length
            subArray = new char[keyLen][];
            for (int i = 0; i < subArray.length; i++) {
                int rounded = Math.floorDiv(charArray.length, keyLen)+
1;
                subArray[i] = new char[rounded];
            }

            for (int i = 0; i < subArray[0].length * keyLen; i++)
            {
                char c = charArray[Math.min(i, charArray.length-1)];
                subArray[i % keyLen][Math.floorDiv(i, keyLen)] = c;
            } //create strings than inclued every keyLenth character

            int[] values = new int[keyLen];
            for (int i = 0; i < subArray.length; i++) {
                values[i] = attackCCforBestMatch(new
String(subArray[i]));
            }

            keyStr = new char[keyLen];
            for (int i = 0; i < keyLen; i++){keyStr[i] =
Utility.alphabet[values[i]];}
            xo.setKey(new String(keyStr));
            if(BruitForce.evaluteText(xo.dec(data)) > 7) {break aa;}
//this check would be helpful if wanted to do for more than length 4
        }


        return xo.dec(data);
    }

    @Override
    public String attackCT(String data) {
        int[] div = Utility.dividers(data.length());
        ColumnarTransposition ct = new ColumnarTransposition();

        String bestMatch = "";
        int bestValue = Integer.MAX_VALUE;

        int[][] keys = null;
```

```
        for (int num : div) { // div is all matrix dimentions
            int[] key = new int[num];
            for (int j = 0; j < num; j++) { // nummbers that appera in
key
                key[j] = j;
            }

            keys = Utility.permutations(key);
            // calculates all keys for current matrix length

            for (int j = 0; j < keys.length; j++) {
                // check the key and decrypt message
                ct.setKey(keys[j]);
                String text = ct.dec(data);
                double value = evaluteText(text);
                if (value <= bestValue) {
                    bestMatch = text;
                    bestValue = (int)value;
                }
            }

        }

        //

        return bestMatch;
    }

}
```

Caesarchiffer

```
package Chiffers;
import Other.Statistics;
import Other.Utility;
import Other.Interfaces.StringEncrytionInterface;

public class CaesarCipher implements StringEncrytionInterface {
    int key = 0;

    public char shitChar(char cha, int n) {
```

```
        return shiftCharAlfa(cha, n);
    }

    public char shiftCharAlfa(char cha, int n) {
    char lowChar = Character.toLowerCase(cha);
    char c;
    if(Utility.contains(Utility.alphabet, lowChar)) {
        int i = (Utility.indexOf(lowChar) + n  + (10 *
Utility.alphabet.length)) % (Utility.alphabet.length);
        c = Utility.alphabet[i];
    } else {
        c =' ';
    }

        return c;
    }

    public char shiftCharSign(char cha, int n) {
        char c = (char) (cha + n);

        return c;
    }

    public String enc(String data) {
        char[] message = new char[data.length()];

        char[] charArray = data.toCharArray(); //blir långsamare och
långsamare
        for (int i = 0; i < charArray.length; i++ ) {
            message[i] = shitChar(charArray[i], (1) * key);
        }

        return new String(message);
    }


    public String dec(String data) {
        //System.out.println("dec");
        Statistics.recordStat("CallDecMethod");
        char[] message = new char[data.length()];

        char[] charArray = data.toCharArray(); //blir långsamare och
långsamare
```

```
        for (int i = 0; i < charArray.length; i++ ) {
            message[i] = shitChar(charArray[i], (-1) * key);
        }

        return new String(message);
    }
    public void setKey(String keyString) {
        key = Integer.valueOf(keyString);
    }

}
```

Vigenèrechiffer

(X_OR.java eftersom tidig misförståelse på namn av algoritm)

```
package Chiffers;
import Other.Statistics;
import Other.Utility;
import Other.Interfaces.StringEncrytionInterface;

public class X_OR implements StringEncrytionInterface {
    public char[] key;


    public String crypt(String data, int factor){
        char[] dataArray = data.toString().toCharArray();
        char[] message = new char[dataArray.length];

        for (int i = 0; i < message.length; i++) {  //inte fel med
denna functionen, inte detta iallafall
            message[i] =
Utility.alphabet[(Utility.indexOf(dataArray[i])     + factor *
(Utility.indexOf(key[ i % key.length])) +29)%29];

        }

        String messString = new String(message);
        return messString;
    }
```

```
        public String enc(String data) {
            return crypt(data, 1);
        }

    public String dec(String data) {
        Statistics.recordStat("CallDecMethod");

        return crypt(data, -1);
    }



    public void setKey(String keyString) {
        key = keyString.toCharArray();
    }

}
```

Transpositionschiffer

```
package Chiffers;

import Other.Statistics;
import Other.Utility;
import Other.Interfaces.StringEncrytionInterface;

public class ColumnarTransposition implements StringEncrytionInterface
{
    public int[] key;

    public char[][] createMatrixEnc(String data) {
        int width = key.length;
        int hight = (int) Math.ceil(data.length() / width);
        char[][] matrix = new char[width][hight];

        for (int i = 0; i < hight; i++) {
            for (int j = 0; j < width; j++) {
                char c;
                if (i * width + j < data.length()) {
```

```
                    c = data.charAt(i * width + j);
                } else {
                    c = 'q';
                }
                matrix[j][i] = c;
            }
        }
        return matrix;
    }

    public char[][] createMatrixDec(String data) {
        int width = key.length;
        int hight = (int) Math.ceil(data.length() / width);
        char[][] matrix = new char[width][hight];

        for (int i = 0; i < width; i++) {
            int l = indexOfvalue(key, i);
            // l = i;

            for (int j = 0; j < hight; j++) {
                matrix[l][j] = data.charAt((i * hight) + j);;
            }
        }

        return matrix;
    }

    public int indexOfvalue(int[] data, int i) {
        Integer l = null;
        for (int k = 0; k < data.length; k++) {
            if (i == data[k]) {
                l = k;
            }
        } // translates i till l
        return l;
    }

    @Override // encryption works, not decrypt
    public String enc(String data) {
        char[][] matrix = createMatrixEnc(data);
        String message = "";

        for (int i = 0; i < key.length; i++) {
```

```
            int l = indexOfvalue(key, i);

            char[] cs = matrix[l];
            for (int j = 0; j < cs.length; j++) {
                char c = cs[j];
                message += c;
            }
        }

        return message;
    }

    @Override
    public String dec(String data) {
        Statistics.recordStat("CallDecMethod");

        char[][] matrix = createMatrixDec(data);
        return decWithMatrix(matrix);
    }

    public String decWithMatrix(char[][] matrix) {
        char[] message = new char[matrix.length * matrix[0].length];

        for (int j = 0; j < matrix[0].length; j++) {
            for (int i = 0; i < matrix.length; i++) {
                message[j * matrix.length + i] += matrix[i][j];
            }
        }

        return new String(message);
    }

    @Override
    public void setKey(String data) {
        key = new int[data.length()];
        char[] localAlphabet = Utility.alphabet;
        char[] dataArray = data.toCharArray();

        int i = 0;
        while (i < data.length()) {
            int k = 0;
            while (k < localAlphabet.length) {
                char alf = localAlphabet[k];
```

```
                for (int j = 0; j < data.length(); j++) {
                    char da = dataArray[j];
                    if (alf == Character.toLowerCase(da)) {
                        // i ++;
                        key[j] = i;
                        i++;
                    }
                }
                k += 1;

            }
        }

    }

    public void setKey(int[] data) {
        key = data;
    }

}
```

Statistics

```
package Other;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.time.LocalDate;
import java.time.LocalTime;

public class Statistics {

    //global var for entire program
    public static long timeStart;
    public static String dataTime;
    public static String dataText;

    //local var for each encryption attack pair
    public static String[] dataEncAttPair; // stores the data collected
in all messges
```

```
    //local var for each message
    public static int funIndex;

    //local var for each funtion type
    public static int[] useOfMessEva; //counts how many times this used
the evalute of message function
    public static int[] useOfDecMeth;
    public static long[] lastTime;

    public static void startCollecting() {
        timeStart = System.nanoTime();
        dataTime = "\"TimeStamp\":[\n";
        dataText = "\"AttackData\":[";

    }

    public static void timeStamp() {timeStamp("");}

    public static void timeStamp(String message) {
        long now = System.nanoTime();

        String mess = "\t{\"Time\":" + (now - timeStart) + ",
\"message\":\"" + message + "\"},";
        dataTime += mess + "\n";
        //System.out.println(mess);

    }


    public static void openAttEncPair() { //starts collection for; ex
CC and BF
        dataEncAttPair = new String[20];
    }

    public static void openTextWork() {
        //declare varbles for pair
        funIndex = 0;
        useOfMessEva = new int[3];
        useOfDecMeth = new int[3];

        lastTime = new long[4];
        lastTime[3] = System.nanoTime();
```

```
    }

    public static void closeSegmentInPair() { // closes the collection
for; ex encryption of ceasar chifer
        lastTime[funIndex] = System.nanoTime();
        funIndex ++;

    }

    public static void closeTextWork(Boolean correct, int j) {
        //sumerice text
        String str = "";
        str += "{";


        str += "\"Name\":";
        str += "\"Text"+ j + "\",";

        //encryotion
        str += "\"enc\":{";
        str += "\"Type\":\"" + (Main.encStandard)+ "\",";
        str += "\"Time\":" + (lastTime[0] - lastTime[3])+ "";
        str += "},";

        //decryption
        str += "\"dec\":{";
        str += "\"Time\":" + (lastTime[1] - lastTime[0])+ "";
        str += "},";

        //attack
        str += "\"att\":{";
        str += "\"Type\":\"" + (Main.attStandard)+ "\",";
        str += "\"mesEva\": " + useOfMessEva[2] + ",";
        str += "\"decMet\": " + useOfDecMeth[2] + ",";
        str += "\"Succes\":" + Boolean.toString(correct) + ",";
//kanske spara tiden här också??
        str += "\"Time\":" + (lastTime[2] - lastTime[1])+ "";
        str += "},";

        //summery
        str += "\"sum\":{";
        str += "\"mesEva\": " + (useOfMessEva[0] + useOfMessEva[1] +
useOfMessEva[2]) + ",";
```

```
        str += "\"decMet\": " + (useOfDecMeth[0] + useOfDecMeth[1] +
useOfDecMeth[2]) + ",";
        str += "\"textLength\":" + Main.inputString.length();
        str += "}";

        str += "}";
        dataEncAttPair[j] = str;
    }

    public static void closeAttEncPair(int i) { // close collection,
preper for new pair: ex CC and BF

        //summerice pair
        String str = "";

        str += "\n\t" + "[";
        //str += "\"Typ" + i + "\","
        str += "\n";
        for (String string : dataEncAttPair) {
            str += "\t\t" + string + ",\n";
        }
        str = str.substring(0, str.length()-2);
        str += "],";

        dataText += str;
        //record key used

    }


    public static void recordStat(String mess) {
        if( mess == "UseMesEva") {
            useOfMessEva[funIndex] += 1;
        } else if( mess == "CallDecMethod") {
            useOfDecMeth[funIndex] += 1;
        } else if( mess == "") {

        } else {
            System.out.println("recorded stat not found:" + mess);
        }
    }

    public static void addDataCount(String string, int i) {
```

```
        //increase a datafield with name "string" i times, or create it
if not exist
    }

    public static String compileData() {
        String data = "{";


        data += dataTime.substring(0, dataTime.length()-2) + "]" +
",\n";

        data += dataText.substring(0, dataText.length()-1) + "]" +
"\n";

        data += "}";
        return data;
    }


    public static void endCollecting(Boolean saveData) {
        String data = compileData();
        if(saveData){
            saveData(data);
        } else {
            System.out.println(data);
        }
    }

    public static void saveData(String data) {
        try {
            Path path = Paths.get("data/statistics/" + "stats" +
LocalDate.now().toString() + "_" + LocalTime.now().getHour() + "-"+
LocalTime.now().getMinute() + /* "-" + LocalTime.now().getSecond() +
"" +*/  ".json");
            Files.write(path, data.getBytes(Main.charset));
        } catch (Exception e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }

    }
```

```
}
```

QuadGram

```
package Other;

import java.util.ArrayList;

public class QuadGram {

    private long[][][][] quadGram;
    private long length;

    public QuadGram() {
        quadGram = new long[29][29][29][29];
        length = 0;
    }

    public long getLength() {
        return length;
    }

    public long[][][][] getQuadGram() {
        return quadGram;
    }

    public void setLength(long length) {
        this.length = length;
    }

    public void setQuadGram(long[][][][] quadGram) {
        this.quadGram = quadGram;
    }

    public void changeLength(long change) {
            this.length += change;

    }
```

```
    public static double compareNGrams(QuadGram qGrams1, QuadGram
qGrams2, int nG2Count) {
        long[][][][] nGrams1 = qGrams1.getQuadGram();
        long[][][][] nGrams2 = qGrams2.getQuadGram();

        double value = 0;
        double factor = (10000000/nG2Count);
        for (int i4 = 0; i4 < nGrams2.length; i4++) {
            for (int i3 = 0; i3 < nGrams2.length; i3++) {
                for (int i2 = 0; i2 < nGrams2.length; i2++) {
                    for (int i1 = 0; i1 < nGrams2.length; i1++) { //793
is precomupted and 10 000 / ngrams in data set
                        double temp = (nGrams1[i4][i3][i2][i1])  -
(nGrams2[i4][i3][i2][i1] * factor);
                        value = value + Math.abs(temp);
                    }
                }
            }
        }

        return value;
    }

    public void readNGrams(String path) {//shoould maybe redo method
        // reads the ngram in files
        QuadGram grams = new QuadGram();
        // int count = 0;

        String[] splits = Main.readData(path).toLowerCase().split("\n",
0);

        for (String split : splits) {
            String gram = split.substring(0, 4);
            long num = Long.valueOf(split.substring(5));
            grams.setNgramValue(gram, num);
            // count += Integer.valueOf(split.substring(5));
        }

        // 79367664
        // System.out.println(count); //sould divide by this and then
multiply by 1000
        quadGram = grams.getQuadGram();
        length = grams.getLength();
```

```
    }

    public long getNgramValue(String nGramS) {
        char[] nGram = nGramS.toCharArray();
        //for (int i = 0; i < nGram.length; i++) nGram[i] =
Character.toLowerCase(nGram[i]);

        //this for loop checks so that it is letters and not signs
        //this should not be needed
        /*
        for (char c : nGram) {
            aa: {
                for (char d : Utility.alphabet) {
                    if (c == d) {
                        break aa;
                    }
                }
                return 1000000000; // a big nummer to discurage signes
not known
            }
        }
        */

        long value =
quadGram[Utility.indexOf(nGram[0])][Utility.indexOf(nGram[1])][Utility.
indexOf(nGram[2])][Utility
                .indexOf(nGram[3])];

        return value;

    }

    public void setNgramValue(String nGramS, long value) {
        char[] nGram = nGramS.toCharArray();

        for (char c : nGram) {
            if(!Utility.contains(Utility.alphabet, c))
            throw new java.lang.NullPointerException("something wrong
with ngram trying to set:" + nGramS) ;
        }
```

```
        length += value -
quadGram[Utility.indexOf(nGram[0])][Utility.indexOf(nGram[1])][Utility.
indexOf(nGram[2])][Utility
                .indexOf(nGram[3])];

quadGram[Utility.indexOf(nGram[0])][Utility.indexOf(nGram[1])][Utility.
indexOf(nGram[2])][Utility
                .indexOf(nGram[3])] = value;
    }

    public static String[] genNGram(String data) {
        String str = Utility.removeSigns(data);//this is slow

        ArrayList<String> grams = new ArrayList<String>() {};

        for (int i = 0; i <= str.length() - 4; i++) {
            char[] temp = { str.charAt(i), str.charAt(i + 1),
str.charAt(i + 2), str.charAt(i + 3) };
            grams.add(new String(temp));
        }

        String[] gramArray = new String[grams.size()];
        gramArray = grams.toArray(gramArray);

        return gramArray;
    }

    public void countNGrams(String[] grams) {
        QuadGram nGramCount = new QuadGram();

        for (String gramS : grams) {
            nGramCount.setNgramValue(gramS,
nGramCount.getNgramValue(gramS) + 1);
        }

        quadGram = nGramCount.getQuadGram();
        length = nGramCount.getLength();

    }
}
```

Utility

```
package Other;
public class Utility {

    public static final char[] alphabet =
"abcdefghijklmnopqrstuvwxyzåäö".toCharArray();
    public static final char[] signs = " ,.;:!?'-
+=%<>[](){}/\r\n\t\"".toCharArray(); //Ãƒf†â¢Â,€¬¶¤-"œ„


//{'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q',
'r','s','t','u','v','w','x','y','z','å','ä','ö'};
    public static int indexOf(char c, char[] alf) {
        Integer num = null;
        for (int i = 0; i < alf.length; i++) {
            if (c == alf[i]) {
                num = i;
            }
        }

        if ( num == null){//throw new
java.lang.NullPointerException("didn't find character for:'" + c +
"'");}
        num = 0;}

        return num;
    }

    public static int indexOf(char c) {
        return indexOf(Character.toLowerCase(c), alphabet);
    }


    public static int factorial(int n){
        if (n == 0)
          return 1;
        else
          return(n * factorial(n-1));
       }

    public static boolean contains(char[] dataArray, char test) {
        boolean cont = false;
```

```
        for (char object : dataArray) {
            if( object == test) cont = true;
        }

        return cont;
    }

    public static int[][] permutations(int[] array) {
        // maybe create lookuptable if too slow

        if (array.length != 1) {
            int[][] perm = new
int[Utility.factorial(array.length)][];//breaks the integer bit limit,
not reasonable tactic
            int n = 0;

            int[] shortArr = new int[array.length - 1];
            System.arraycopy(array, 0, shortArr, 0, array.length - 1);

            int[][] tempPerm = permutations(shortArr);
            for (int i = 0; i < array.length; i++) {
                for (int j = 0; j < tempPerm.length; j++) {
                    int[] tempArr = new int[array.length];
                    if (i != 0) {
                        System.arraycopy(tempPerm[j], 0, tempArr, 0,
i); // copys start until select index
                    }
                    tempArr[i] = array[array.length - 1]; // makes
select index to last index
                    if (i != array.length - 1) {
                        System.arraycopy(tempPerm[j], i, tempArr, i +
1, array.length - 1 - i); // copys after select

// index
                    }

                    perm[n] = tempArr;
                    n++;
                }

            }
            return perm;
```

```
        } else {
            int[][] perm = new int[1][];
            perm[0] = array;
            return perm;
        }

    }


    public static int[] dividers(int nummber) {
        int[] div = new int[0];

        for (int i = 1; i < nummber; i++) {
            if (nummber % i == 0) {
                int[] tempDiv = new int[div.length + 1];
                System.arraycopy(div, 0, tempDiv, 0, div.length);
                div = tempDiv;
                div[div.length - 1] = i;
            }
        }

        return div;

    }

    public static String removeSigns(String data) {
        String message = data;
        for (char c : signs) {
            message = message.replace(Character.toString(c), "");
        }

        //for (char c : data.toCharArray()) {if(contains(alphabet, c))
message += Character.toString(c);}

        return message;
    }


}
```

Kommentar kod appendix

Vissa klasser har lämnats ur ur detta appendix av olika anledningar; Interfaces utlämnades för de var trivialt enkla medan vissa attackklasser utlämnades då de ej användes i slutprojektet och är färdigutvecklade till olika grad. DHS är helt klar och tog lång tid att utveckla men då säkerheten där var för tight för denna studiens resurser att använda så kunde den ej inkluderas. Se min github-sida för hela kodprojektet: https://github.com/Miiroun/GA.

Appendix 3 Rådata

Här är följande textlängder i antal bokstäver

text0:Hamlet:182864

text1:RaJ:146704

text2:MND:100554

text3:WinterTale:149062

text4:MerchantVenice:125304

text5:Carol:182094

text6:DetGarAn:345093

text7:FrakenJulie:98112

text8:Fadern:86344

text9:RodaRummet:156888

text10:LoTR:1657

text11:Tradet:612

text12:IHAD:1377

text13:TwoCitys:286

text14:1984:964

text15:ToBE:39

text16:PP:199

text17:BB:237

text18:WD:71

text19:HW:11

Observera att attacken på CT (column transposition) är inte frekvensanalys utan en typ av hill climbing algorith men namngavs FA för spara plats i koden. Tiden mäts i nanosekunder

Name	enc.Type	enc.Time	dec.Time	att.Type	att.Succes	att.Time	sum.textLength
Text0	CC	1548914700	10307300	FA	SANT	38788600	182864
Text1	CC	864338400	5948700	FA	SANT	44934100	146704

Text2	CC	376934200	2559200	FA	SANT	5366900	100554
Text3	CC	824728700	4017900	FA	SANT	9197100	149062
Text4	CC	588908500	5237800	FA	SANT	10773900	125304
Text5	CC	1296612800	8685500	FA	SANT	45488700	182094
Text6	CC	4368433000	10571200	FA	SANT	19948300	345093
Text7	CC	385519400	3164100	FA	SANT	6475300	98112
Text8	CC	299565100	2715300	FA	SANT	5521400	86344
Text9	CC	937994500	5197900	FA	SANT	9620000	156888
Text10	CC	699900	83900	FA	SANT	367900	1657
Text11	CC	353200	24300	FA	SANT	274600	612
Text12	CC	460300	54500	FA	SANT	286600	1377
Text13	CC	310300	13000	FA	SANT	201800	286
Text14	CC	388500	36100	FA	SANT	255400	964
Text15	CC	204700	8800	FA	SANT	327400	39
Text16	CC	275200	17700	FA	SANT	242800	199
Text17	CC	212300	18700	FA	SANT	530400	237
Text18	CC	258300	8100	FA	SANT	308700	71
Text19	CC	310800	4900	FA	FALSKT	278100	11

Text0	CC	1222321500	5560400	BF	SANT	1134601000	182864
Text1	CC	1210603600	4391500	BF	SANT	608079900	146704
Text2	CC	547045500	2889200	BF	SANT	398451000	100554
Text3	CC	900094100	5885400	BF	SANT	625313400	149062
Text4	CC	855406100	3692000	BF	SANT	660261300	125304
Text5	CC	1258767000	5172500	BF	SANT	738055500	182094
Text6	CC	4720010000	11068500	BF	SANT	1882579900	345093
Text7	CC	455442600	3099600	BF	SANT	445672600	98112
Text8	CC	291214800	2876000	BF	SANT	359109000	86344
Text9	CC	1042926400	7363200	BF	SANT	720718600	156888
Text10	CC	739500	58200	BF	SANT	7889100	1657
Text11	CC	462300	26800	BF	SANT	4816700	612
Text12	CC	518700	57200	BF	SANT	8391000	1377
Text13	CC	352700	13500	BF	SANT	1498800	286
Text14	CC	300800	39600	BF	SANT	5258600	964
Text15	CC	426200	5700	BF	SANT	440200	39

Text16	CC	313700	21300	BF	SANT	1023900	199
Text17	CC	356200	13800	BF	SANT	1158500	237
Text18	CC	328300	5900	BF	SANT	1059200	71
Text19	CC	295800	5400	BF	FALSKT	168600	11
Text0	Vig	27943700	9725200	FA	SANT	56447900	182864
Text1	Vig	7332700	6992600	FA	SANT	37260200	146704
Text2	Vig	4799600	3372800	FA	SANT	26158700	100554
Text3	Vig	7347100	5055500	FA	SANT	37192100	149062
Text4	Vig	5132800	4134000	FA	SANT	33050400	125304
Text5	Vig	8421700	10140900	FA	SANT	55059800	182094
Text6	Vig	19866700	14101500	FA	SANT	81504300	345093
Text7	Vig	4292800	3805500	FA	SANT	27999600	98112
Text8	Vig	4423400	3551800	FA	SANT	29619800	86344
Text9	Vig	6462400	5638200	FA	SANT	49193100	156888
Text10	Vig	839500	114700	FA	SANT	2064400	1657
Text11	Vig	480000	47500	FA	SANT	1692400	612
Text12	Vig	505400	1071500	FA	SANT	3092700	1377
Text13	Vig	715800	20500	FA	SANT	1350600	286
Text14	Vig	409700	84700	FA	SANT	2428700	964

Text15	Vig	238800	5900	FA	FALSKT	3442000	39
Text16	Vig	661900	21300	FA	FALSKT	1203200	199
Text17	Vig	285100	12600	FA	SANT	907100	237
Text18	Vig	307800	5400	FA	FALSKT	890600	71
Text19	Vig	265500	2400	FA	FALSKT	979900	11
Text0	Vig	9692400	6791600	BF	SANT	1213683300	182864
Text1	Vig	5473900	4962500	BF	SANT	884944300	146704
Text2	Vig	4030900	4021600	BF	SANT	610558500	100554
Text3	Vig	5686300	4956900	BF	SANT	870184200	149062
Text4	Vig	4881600	4145900	BF	SANT	739755600	125304
Text5	Vig	9119700	9338600	BF	SANT	1048118100	182094
Text6	Vig	12074700	11720500	BF	SANT	2038781700	345093
Text7	Vig	4089300	3286300	BF	SANT	630933600	98112
Text8	Vig	3326000	2843700	BF	SANT	556864900	86344
Text9	Vig	5743700	5185400	BF	SANT	943867900	156888
Text10	Vig	528900	81300	BF	SANT	11993000	1657

Text11	Vig	500300	55300	BF	SANT	5809700	612
Text12	Vig	576000	84300	BF	SANT	12445100	1377
Text13	Vig	505300	22800	BF	SANT	3218700	286
Text14	Vig	431500	84800	BF	SANT	10263200	964
Text15	Vig	490200	4900	BF	FALSKT	436600	39
Text16	Vig	378800	15500	BF	FALSKT	1749800	199
Text17	Vig	400100	11200	BF	FALSKT	2153700	237
Text18	Vig	253100	5700	BF	FALSKT	631500	71
Text19	Vig	467700	2200	BF	FALSKT	1391300	11
Text0	CT	2210449600	9980400	FA	FALSKT	623640400	182861
Text1	CT	1147597500	1060000	FA	FALSKT	345091300	146699
Text2	CT	454257300	743700	FA	FALSKT	2667773500	100548
Text3	CT	892870100	1056300	FA	FALSKT	4448731400	149058
Text4	CT	670867300	896100	FA	FALSKT	2541834800	125300
Text5	CT	2441433300	9731100	FA	FALSKT	2496554000	182091
Text6	CT	11799775300	3330300	FA	FALSKT	5454621800	345093
Text7	CT	770163100	623100	FA	FALSKT	2220568000	98112

Text8	CT	556435400	921500	FA	FALSKT	1037763100	86338
Text9	CT	1852535800	1214700	FA	FALSKT	3387282900	156884
Text10	CT	763600	36500	FA	FALSKT	3366100	1652
Text11	CT	373700	26300	FA	FALSKT	854500	609
Text12	CT	650300	38000	FA	FALSKT	4012000	1372
Text13	CT	610900	62000	FA	FALSKT	1586900	280
Text14	CT	455200	31900	FA	FALSKT	1039700	959
Text15	CT	342000	21400	FA	FALSKT	65300	35
Text16	CT	288600	36000	FA	FALSKT	375400	196
Text17	CT	303300	26200	FA	FALSKT	559000	231
Text18	CT	311100	21400	FA	FALSKT	160500	70
Text19	CT	286900	22300	FA	FALSKT	36500	7

Appendix 4 Komplexitet data

Längd [Char]	Tid[Sek]
286	0,001637
572	0,0008966
1144	0,001059
2288	0,0012045
4576	0,0025636
9152	0,0026493
18304	0,0057816
36608	0,0120712
73216	0,0139914
146432	0,0100346
292864	0,0135478
585728	0,0206004
1171456	0,040132
2342912	0,076192
4685824	0,1350339
9371648	0,281698
18743296	0,5504132
37486592	1,0730657
74973184	2,0200969