Kapitel 1

Indledning

1.1 Introduktion til steganografi

Overvågning af de digitale medier og de debatter, som dette medfører, er et emne, som i de senere år har spillet en stor rolle i samfundsdebatten. For at undgå denne overvågning og stadig gøre brug af de digitale medier, er systemer eller metoder for sikker kommunikation nødvendige. Formålet ved disse metoder er, at gøre det drastisk besværligt for uvedkommendes adgang til det kommunikerede data. Til dette formål kan *steganografi* anvendes.

Ordet steganografi kommer fra græsk og betyder, direkte oversat, skjult skrift. Som navnet antyder, muliggør steganografi sløring af data ved at indlejre dette i andet, upåfaldende dæk-data. Dette kan gøres eksempelvis ved at sende en besked til en modtager, hvori de første bogstaver i hver linje samlet skaber en ny besked. Moderne steganografi fungerer ved samme princip, men benytter computere og digitalt gemte medier til at muliggøre denne skjulte kommunikation. Da al data på computere repræsenteres binært, er der få grænser for hvilke former for meddelelser og data, der kan skjules, samt i hvilket digitalt data de kan skjules i. Dette betyder at både tekst, lyd, billeder mm. kan benyttes som både den hemmelige besked og det kommunikerede *dækmedie*, som beskeden gemmes i. Moderne steganografi fungerer således ved først at omdanne beskeden til et binært sprog, og så at ændre udvalgte binære værdier i dækmediet, således disse repræsenterer beskeden. Modtageren af dækmediet behøver blot at identificere det ændrede data, og så omdanne dette til en læselig besked igen. Dette medfører at tilsyneladende uvigtig kommunikation og filoverførsel kan bruges til at overføre sikkerhedsfølsom data.

Et eksempel på praktisk anvendelse af moderne steganografi er militærkommunikation, hvor data skjules i et støjfyldt lyd-/radiosignal. Til den uvidende lyder det som typisk radiostøj, men såfremt modtageren er indforstået med den anvendte steganografiske teknik, kan støjen fjernes og det gemte data udvindes. [21] I dette

projekt vil der være fokus på at sikre data steganografisk ved brug af billeder som dækmedie.

Steganografi kan anses som en metode, der er beslægtet med *kryptografi*, dog er der subtile forskelle på deres formål. Kryptografi går ud på at ændre det sikkerhedsfølsomme data til noget data, der er ulæselig, såfremt den præcise metode, der blev anvendt til at gøre dataet ulæselig ikke kendes. Dette muliggør også sikker kommunikation, dog skjuler det i modsætning til steganografi ikke det faktum, at der foregår kommunikation mellem de to parter. Da formålet ved kryptografi dog minder om steganografi, vil der i løbet af denne rapport blive anvendt kilder, som hovedsageligt omhandler den ene af de to metoder, til at beskrive den anden. [73]

1.2 Initierende problem

Der opstilles herunder nogle spørgsmål, som tilsammen udgør det initierende problem, der har til formål at styre projektets retning fremadrettet. Vi vil efterfølgende undersøge disse spørgsmål og begrænse, specificere og konkretisere problemet, således det til sidst kan løses ved et produkt. Det initierende problem er således udgangspunktet, som dette projekt udspringer af.

Det initierende problem er:

- *Hvordan kan steganografi og kryptografi bruges til at sende data skjult i billeder via sociale medier?*
- *Hvorledes forhindrer loven privatpersoner i at benytte sociale medier til at sende beskeder med indhold, som ejerne af mediet ikke har kendskab til? Hvilke konsekvenser har det for ejeren af mediet at brugerne benytter mediet til steganografisk skjult kommunikation?*
- *Hvordan bør man som programudvikler forholde sig til kryptografiske og steganografiske muligheder som et system til sikker kommunikation vil kunne give?*

Kapitel 2

Problemanalyse

2.1 Problemanalyse

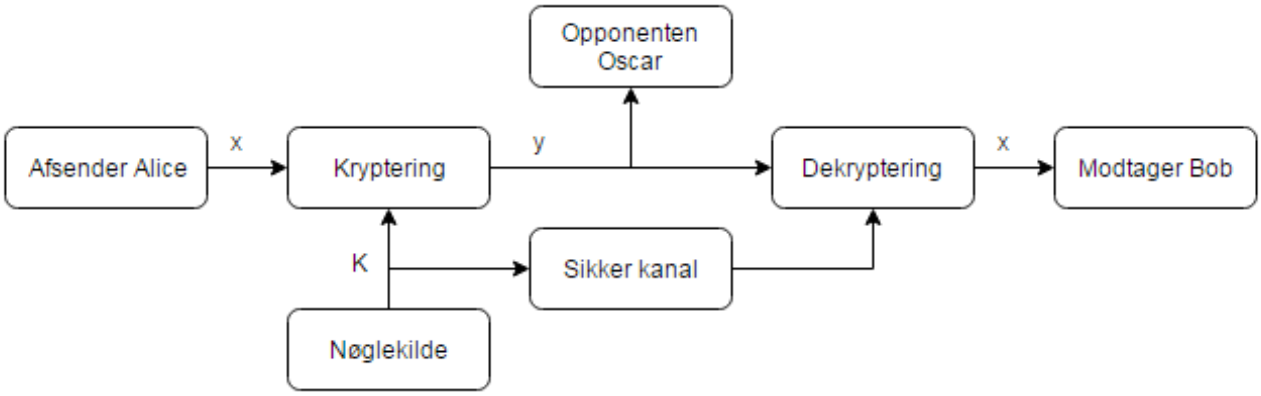
Herunder vil det initierende problem blive analyseret og belyst med henblik på at kunne opstille en specifik og konkret problemformulering, som så senere forsøges løst med et endeligt produkt.

2.1.1 Modeller for sikker kommunikation

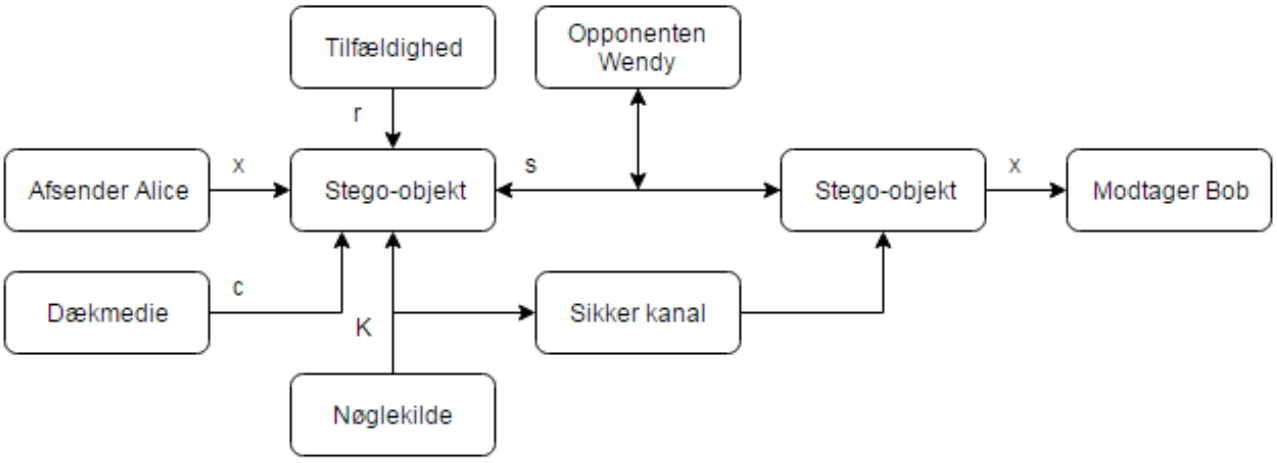
For at belyse det initierende problem og nå frem til en endelig problemafgrænsning og formulering, er det nødvendigt at specificere de elementer, som kan sikre sikker kommunikation. Ved sikker kommunikation menes primært, at kommunikation mellem to parter kan foregå, uden at indholdet på enkel vis kan blive kendt af en opponerende part. I dette afsnit introduceres to modeller for sikker kommunikation til at give en forståelse for emnets hovedelementer og fagtermer. På figur ses 2.1 en model for sikker envejskommunikation vha. kryptografi.

I denne model vil afsenderen Alice sende en besked til modtageren Bob, uden at indholdet af beskeden bliver kendt af opponenten Oscar. Alice starter ved at skrive en besked x, også kaldet *klarteksten*. Beskeden bliver krypteret til den ulæselige ciffertekst y ved brug af en nøgle K fra en nøglekilde. Bob modtager den krypterede besked og kan herefter dekryptere den ved at bruge nøglen K, som han har modtaget gennem en sikker kanal af Alice. Bob kan herefter læse beskeden x, som Alice skrev. Oscar kan også se den krypterede besked y, men kan ikke oversætte den til den læselige klartekst x, da han ikke besidder nøglen K, da den blot er delt mellem Alice og Bob.

En tilsvarende model for sikker envejskommunikation, dog steganografisk sikret i stedet for kryptografisk, ses på figur 2.2.



Figur 2.1: Model for kryptografisk sikret kommunikation [74].



Figur 2.2: Model for steganografisk sikret kommunikation [26].

I denne model vil afsenderen Alice også gerne sende en besked til modtageren Bob, men denne gang uden at opponenten Wendy overhovedet bemærker at en besked er blevet sendt. Alice starter ved at skrive en besked (klartekst) x, der optimalt gemmes ved brug af en nøgle K i et dækmedie c. Heri bliver klarteksten lagt evt. ved brug af tilfældighedselementer r, således der ikke altid laves steganografi på eksakt samme måde. Denne proces udføres meget omhyggeligt for at generere det endelige *stego-objekt* s. Alice kan nu sende stego-objektet til Bob, som nu kan bruge nøglen K til at få klarteksten x ud af stego-objektet s og læse beskeden, som Alice skrev. For at kommunikationen mellem Alice og Bob kan foregå ubemærket, er det vigtigt at dette stego-objekt ikke vækker opmærksomhed hos Wendy, og at beskeden ikke kan findes ved steganalyse. Det perfekte scenarie er, at der ikke kan ses nogen forskel på dækmediet c og stego-objektet s af hverken et menneske eller

en computer.

De ovenstående modeller på henholdsvis figur 2.1 og figur 2.2 kommer til at spille en central rolle for hele dette kapitel og vil blive refereret til løbende gennem de følgende afsnit. Disse benyttes som værktøjer til at skabe overblik over steganografiske og kryptografiske begreber og processer.

2.1.2 Kryptering og nøgler

I dette afsnit vil vi med udgangspunkt i modellen for sikker kommunikation på figur 2.1 belyse, hvilken rolle kryptering og nøgler spiller for at holde kommunikationen mellem Alice og Bob sikker.

Ciffertekst og klartekst

Ciffertekst er et centralt begreb i denne sammenhæng, og beskriver tekst, der er gjort ulæselig ved hjælp af kryptering. Ciffertekst ses således ofte som tekst bestående af en række tilfældige bogstaver, tegn og tal, der vha. den rette krypteringsnøgle repræsenterer den oprindelige tekst, men umiddelbart forekommer tilfældige. Den oprindelige tekst beskrives ved begrebet *klartekst*. Både ciffertekst og klartekst anvender begge et udvidet tekstbegreb og kan udover almen tekst også henvise til al mulig digital data, heriblandt billed- og lydfiler.

Teknikker til kryptering og nøglefordeling kan generelt opdeles i to metoder, *symmetrisk* og *asymmetrisk kryptering*, som vil blive gennemgået i de følgende underafsnit [4]. For at skabe et overblik over hvordan de to metoder fungerer, således vi kan vælge den bedste under design af projektets endelige løsning, vil vi tage udgangspunkt i modellen for sikker kommunikation under beskrivelsen af disse.

Symmetrisk kryptering

Symmetrisk kryptering fungerer ved at der genereres én enkelt nøgle, som er delt mellem Alice og Bob. Denne nøgle bruges både til at kryptere klarteksten til ciffertekst, og til dekryptering af cifferteksten til klartekst. Dette betyder, at hvis Alice vil sende en besked til Bob ved brug af denne metode, så genererer Alice først en symmetrisk nøgle, og benytter efterfølgende denne til at kryptere klarteksten til ciffertekst. Cifferteksten kan nu sendes over en kommunikationskanal til modtageren Bob, og opponenten Oscar ville ikke kunne oversætte indholdet af denne til klartekst uden brug af den rette nøgle. Problemet her er dog, at det er nødvendigt at Alice på en eller anden måde meddeler nøglens indhold sikkert til Bob for at kommunikationen kan foregå. Ideelt, så foregår dette på forhånd, eller over en kommunikationskanal, som Oscar ikke har mulighed for at overvåge. Bob kan herefter dekryptere cifferteksten tilbage til den klartekst, som Alice sendte. Der er også

andre måder hvorpå den symmetriske nøgle kan deles, der vil blive gennemgået i underafsnittet omhandlende deling af nøgler senere i dette afsnit.

Asymmetrisk kryptering

Asymmetrisk kryptering fungerer ved at der genereres et nøglepar. Dette par af nøgler bliver refereret til som henholdsvis en *offentlig nøgle* og en *privat nøgle*. Den offentlige nøgle bruges til at kryptere klartekst til ciffertekst, hvilket så sendes til modtageren. Den private nøgle kendes dog kun af modtageren, og er den nøgle, som kan dekryptere cifferteksten igen. Dette betyder, at hvis Alice ved brug af denne metode vil sende en besked til Bob, så skal Alice krypterer beskeden ved at bruge den offentligt tilgængelige nøgle, og efterfølgende sende den krypterede besked til Bob. Når Bob modtager cifferteksten kan han med sin private nøgle oversætte cifferteksten tilbage til den klartekst, som Alice sendte. Her vil opponenten Oscar, selvom han har adgang til den offentlige nøgle, ikke være i stand til at oversætte den krypterede besked tilbage til klartekst igen.

Deling af nøgler

I stedet for at anvende asymmetrisk kryptering på al kommunikationen, så kan en symmetrisk nøgle deles ved brug af asymmetrisk kryptering. En måde, hvorpå dette kan blive gjort, er hvis afsenderen Alice genererer en symmetrisk nøgle ud fra sin private nøgle og modtageren Bobs offentlige nøgle. Dog kan en symmetrisk nøgle også genereres på andenvis og behøver ikke anvende eksisterende nøgler. Efterfølgende kan Alice krypterer den genererede symmetriske nøgle med Bobs offentlige nøgle og sende nøglen til Bob. Herefter kan Bob, ligesom det var tilfældet med asymmetrisk kryptering, bruge sin private nøgle til at dekryptere den symmetriske nøgle. Efter dette har Alice og Bob en fælles symmetrisk nøgle, uden at opponenten Oscar har haft mulighed for at dekryptere den symmetriske nøgle. Dette kan være en fordel at bruge frem for kun at anvende asymmetrisk kryptering, da symmetrisk kryptering og dekryptering går hurtigere på standardhardware [63].

Endvidere er størrelsen af en symmetrisk og en offentlig nøgle ikke sammenlignelig i forhold til styrke, da en 128-bit symmetrisk nøgle kan være ligeså stærk, som en 3000-bit offentlig nøgle [87]. At den symmetriske nøgles størrelse er mindre, medfører også at den vil være nemmere at dele og ikke vække lige så meget opmærksomhed som en offentlig nøgle.

I følgende afsnit undersøges hvilke krav, der bør være opfyldt for at sikre sikker kommunikation.

2.1.3 Krav for sikker kommunikation

Ved systemer til sikker kommunikation, som beskrevet ved kommunikationsmodellen på figur 2.1, er det essentielt at undersøge de følgende egenskaber og i hvor høj grad kommunikationssystemets målgruppe ønsker disse sikret. Bemærk at den anvendte kilde [4] omhandler sikker kommunikation med fokus på systemer til sikker deling af krypteringsnøgler.

Autenticitet

Autenticitet er en sikkerhed om at afsenderen og modtageren af klarteksten, Alice og Bob, kan betro sig på at modpartens identitet er den som hævdes. Autenticitet kan ved kommunikationssystemet sikres gennem en autentifikationsproces, der for hver af de kommunikerende parter identificerer modparten, og herefter evt. låser op for kommunikation. Autenticitet af systemet kan også sikres ved at autentifikationsprocessen validerer, at det givende klartekst stammer fra den tilhørende kilde.

Autentifikation af de kommunikerende parter Alice og Bob kan foregå ved at undersøge elementer fra de følgende kategorier:

- Noget de ejer (eksempelvis et nøglekort)
- Noget de husker (eksempelvis et kodeord)
- Noget de er (eksempelvis et fingeraftryk)
- Noget de gør (eksempelvis en håndskriftstype)

Eksempelvis er NemID et system, hvis data er sikret af to faktorer, navnligt noget man husker (brugernavn og password), og noget man ejer (nøglekort). Der bør ved design af en autentifikationsproces tages hensyn til, at den personlige information, der bruges til selve verificeringen, ikke bør sendes over usikre kommunikationskanaler.

Integritet

Integritet er en sikkerhed om, at klarteksten ikke er blevet ændret. Sikker kommunikation kræver således en proces, der kan verificere integriteten af cifferteksten, således det kan vides, at det ikke er blevet ændret, gældende for både utilsigtede og målrettede ændringer, under hverken forsendelsen eller opbevaringen. Hvis denne verificering slår fejl, så bør modtageren af dataet meddeles om dette.

Fortrolighed

Fortrolighed er sikkerheden om at data i hele processen kun kan læses af den tiltænkte modtager. At kun modtageren af cifferteksten har adgang til dennes indhold, er derved blot en brøkdel af den samlede fortrolighed af et system. Det kræves derfor også eksempelvis at hvis Alice og Bobs personlige informationer udveksles af kommunikationssystemet for at verificere autenticitet, så bør denne information ikke være tilgængelig for de dele af systemet, som ikke direkte har brug for denne, for at undgå potentielle sikkerhedsbrud. Fortrolighed er et krav når data er lagret på et medie (f.eks. en computerharddisk), hvortil uautoriserede personer har adgang, når data er sikkerhedskopieret på et medie, som kan falde i hænderne på uautoriserede, og når data sendes over ubeskyttede netværk.

2.1.4 Overvejelser i forhold til sociale medier

Idet der i det initierende problem lægges fokus på brug af sociale medier som kommunikationskanal for steganografisk sikret kommunikation, så bør en række overvejelser i forhold til brugen af disse foretages. I dette afsnit undersøges primært muligheder og begrænsninger ved det sociale medie Facebook, da det, grundet dets overvældende popularitet, er baggrund for flere illustrative undersøgelser end de andre muligheder.

Fortrolighed

Da det kan antages at kommunikation udført vha. steganografiske metoder har til formål at være skjult overfor opponenter, er det relevant at undersøge hvorvidt privat kommunikation på sociale medier kan tilgås af udefrakommende som f.eks. efterretningstjenester.

Ifølge deres egne retningslinjer for udlevering af data til myndigheder, kan Facebook udlevere data hvis dette er i kontekst af en dommerkendelse eller efterforskning [17]. Mængden og typen af data, som Facebook udleverer, afhænger af den specifikke sag. Data som udleveres til myndigheder kan bl.a. indeholde brugerinformation som: navn, kreditoplysninger, email-adresser, IP-adresser, gemt indhold, beskeder, billeder, videoer, delinger og lokationsdata.

Det er altså muligt for myndigheder at tilgå en given brugers data på Facebook, såfremt dette sker som led af en efterforskning. Da Facebook er en amerikanskbaseret virksomhed, tages der udgangspunkt i amerikansk lovgivning, samt amerikanske myndigheder og retssystemer.

Facebook lavede i 2013, over en 6 måneders periode, en statistisk undersøgelse over statslige efterspørgelser af brugerdata, dvs. hvor mange gange Facebook har

modtaget officielle forespørgsler om udlevering af data fra statslige instanser. Det fremgår at der i disse seks måneder blev efterspurgt 11 danskeres data, hvoraf 55% af dette data blev udleveret. Af rapporten fremgår det også at op mod ca. 21.000 amerikaneres kontodata blev efterspurgt, med en successrate på 79%.[16]

Der er dermed en stor sandsynlighed for at steganografisk skjult kommunikation ville kunne blive udleveret af Facebook, hvis de kommunikerende personer kommer i søgelyset af myndigheder eller efterretningstjenester. Det kan siges, at det er her, hvor styrken af steganografien har betydning, dog er det ved design af et system for sikker kommunikation ulogisk at benytte en mindre sikker, frem for en mere sikker kommunikationskanal.

Dataintegritet

Ved sikker kommunikation med sociale medier som kommunikationskanal, er det essentielt at undersøge de tekniske problematikker, som dette medfører. Herunder er datakomprimering, der udgør en trussel for det steganografisk skjulte datas integritet.

Komprimering af data kan gøre at dette data fylder mindre, og anvendes ofte på data over websider, da dette mindsker datastrømmen, som siden skal håndtere, hvilket kan gøre denne mere responsiv over for brugeren. Facebook specifikt behandler forskellige billedfiltyper på deres webside forskelligt. Jo større det originalt uploadede billede er, desto mere agressiv kompression bliver anvendt på dette. Af denne grund anbefaler Facebook at man holder sig indenfor deres retningslinjer for billedopløsning [7].

Fordele ved et socialt medie som kommunikationskanal

Dog er der positive elementer ved at benytte et socialt medie som Facebook som platform for steganografisk kommunikation. De to primære fordele er mængden af data og udbredelsen af platformen, og dermed antal brugere, som der kan kommunikeres med. Se tabel 2.1 for et overblik over de primære fordele og ulemper ved brug af et socialt medie som kommunikationskanal.

Fordele	Ulemper
- Mængden af data på platformen *En stor strøm af data på en given platform kan være en fordel for sikker kommunikation, da der vil være mere data som opponenten skal undersøge. Dette kan betyde at færre af de sendte stego-objekter overhovedet udsættes for kontrol. Brugere på Facebook uploadede eksempelvis i år 2012 og 2013 i gennemsnit 300-350 millioner billeder om dagen. [32] [78] [8] [70]*	- Komprimering af data *Sociale medier komprimerer ofte bruger-uploadet data for at holde websitet optimeret. Dette har en negativ effekt på steganografiske beskeder, da tab af data kan forekomme ved komprimering.*[7] [16]
- Antal brugere *Sociale medier er fordelagtige at anvende som platform for kommunikationssystemer, da dette giver flere muligheder for modtagere, samt kan gøre målgruppen for produktet større. Facebook specifikt er en globalt tilgængelig platform (med få undtagelser, så som Kina og Nordkorea) med ca. 1,5 milliarder registrerede brugere. Baseret på det samlede antal brugere er Facebook det absolut største sociale medie.* [11]	- Udlevering af data *Sociale medier er ofte udsat for at efterretningstjenester, f.eks. FBI eller PET, i forbindelse med efterforskninger kræver enkelte brugeres personlige data udleveret. Ved anvendelse af sociale medier som kommunikationskanal bør det forventes at stego-objekterne kan tilgås og nøje undersøges af sådanne organisationer. Dette data forbliver derudover på Facebooks servere, og kan ikke slettes direkte af brugeren.*[17]

Tabel 2.1: Fordele og ulemper

Da Facebook er det sociale medie med størst brugerantal, har det derved flere af de ovenstående fordele (større datastrøm og bedre kommunikationsmuligheder) end alternativerne. Facebook vælges af denne grund som kommunikationsplatform til det endelige produkt. Fremadrettet er det derfor nødvendigt at en løsning til den ovenstående problemstilling, Facebooks implicitte billedkomprimering, findes.

Efterfølgende vil de etiske og lovmæssige overvejelser ved systemer til sikker kommunikation blive belyst. Heri indgår konsekvenser af sikkerhedsfølsom og/eller etisk tvivlsom kommunikation over sociale medier, samt hvilke konsekvenser dette kan have for ejeren af kommunikationskanalen.

2.1.5 Etiske grundprincipper

I denne del af problemanalysen undersøges et antal underemner, som alle tager udgangspunkt i etik. Da der findes flere filosofiske tilgange til at vurdere, hvad

der er etisk korrekt, er det først nødvendigt at specificere hvilke etiske principper, der tages udgangspunkt i. Ifølge Det Etiske Råd går etik ud på, hvad man bør gøre som individer og samfund. Af de forskellige tilgange, der er til vurdering af om en handling er etisk korrekt eller ikke, findes tre hovedtilgange, som alle befinder sig under den normative etik, som netop behandler spørgsmålet omkring hvad man bør gøre i visse situationer, eller hvordan man bør forholde sig til nye teknologier. De tre hovedtilgange er: *konsekvens-*, *pligt-* og *dydsetik*, dog vil vi ikke forholde os til den tredje af disse i rapporten. [68]

Konsekvensetik

Umiddelbart kan konsekvensetikken hjælpe med at svare på disse spørgsmål, da den grundlæggende tankegang i denne etik er, at man kan konsekvensvurdere sig frem til et svar. Det rigtige svar er, i princippet, meget simpelt at finde frem til, idet konsekvenserne af alle de mulige handlinger vurderes, og der herefter afgøres hvilken handling bringer det mest positive resultat til de berørte individer. Det kan til gengæld være svært at anvende, hvis man tænker i forhold til nye teknologier, da der f.eks. kan være konsekvenser, som først viser sig langt ude i fremtiden. Samtidigt skal der indsamles en stor mængde viden og data. Der skal laves matematiske modeller, som kan tage højde for forskellige udviklinger, og dermed hjælpe med at tage en beslutning. Dette kræver mange ressourcer og tid, og tilmed kan det være at det ikke er muligt at tage højde for alle faktorer, fordi omfanget af en bestemt teknologis anvendelsesmuligheder er banebrydende. [68]

Pligtetik

En anden vurderingsmulighed er pligtetik, der tager udgangspunkt i, at mennesker har en bestemt pligt, som skal overholdes, og at denne pligt bør bruges som grundlag for at foretage bestemte handlinger. Det kan herved godt være at en bestemt handling bringer positive resultater af sig, men hvis dette overtræder en etisk pligt, bør handlingen stadig ikke foretages. Selve pligterne, som er grundlaget for afgørelsen af en bestemt handling, kommer fra individet selv. [68]

2.1.6 Etiske problematikker

Formålet med dette afsnit er at undersøge nogle problematikker, med fokus på kryptografi, hvor de etiske grundprincipper benyttes. Dette gøres for at få en bedre forståelse af hvilke etiske problemstillinger der findes, for at der efterfølgende bedre kan tages stilling til steganografi, og dermed også det produkt, som bliver udviklet igennem dette projekt. Der tages hertil udgangspunkt i kryptografi, da denne teknologi er nært beslægtet steganografi, og der derved direkte kan overføres mange, hvis ikke alle, synspunkter.

Det er ikke ulovligt at benytte kryptografi i Danmark, og ifølge Ken Friis Larsen, lektor på Datalogisk Institut på Københavns Universitet, anvender den danske befolkning ikke kryptering i et stort nok omfang [14]. Kryptering giver til gengæld en række problemer for efterforskningsorganisationer, som f.eks. FBI og NSA [45].

NSA er eksempelvis beskyldt for at ønske, at man benyttede en særlig pseudorandom number generator, Dual_EC_DRBG (som bruges i krypteringsalgoritmer). Det er muligt, hvis man er ekspert i algoritmen, at gætte sig frem til rækkefølgen af tilfældige tal fra Dual_EC_DRBG, hvilket resulterede i et svagere sikkerhedssystem[19]. Det menes at NSA dermed kunne dekryptere filer, som var blevet krypteret med en algoritme, der brugte Dual_EC_DRBG til at genere tilfældige tal[10].

På den anden side er Apples CEO Tim Cook, der ligesom Friis Larsen mener [12], at kryptering er utrolig vigtigt at bibeholde, fordi det er den eneste måde at sikre almindelige borgers personlige information og friheder. Firmaer som Apple og efterforskningsorganisationer som FBI har således forskellige (modstridende) interesser.

Dette mener Phillip Rogaway også [64], som beskriver situationen som værende en balance mellem kollektiv sikkerhed og beskyttelsen af et individs privatliv. Hertil rejser sig problemet om man kan have maksimal beskyttelse af privatlivet, samtidigt med maksimal kollektiv sikkerhed? Rogaway mener at det ene nødvendigvis byttes for det andet. Der vil i projektet ikke videre blive taget stilling til dette fordybelsesområde, på grund af dets omfang og kompleksitet. Fremadrettet antages blot, at der altid vil være en konflikt mellem den kollektive sikkerhed og graden af et individs privatlivsbeskyttelse.

2.1.7 Etisk vurdering af et etableret steganografisk system

Formålet med dette afsnit er at finde ud af, hvordan et etableret steganografisk kommunikationssystem overordnet set kan påvirke myndigheders arbejde og hvilken betydning dette kan have for resten af samfundet. Desuden overvejes hvordan det har en effekt på den anvendte kanal, som bruges til distribution af beskederne, samt de etiske aspekter af at benytte en kanal til et formål, som kanal- ejeren ikke har viden om.

Der findes steganografisk software, som kan gemme en besked i et billede, som f.eks. QuickStego [52] og over 35 andre programmer med lignende funktionalitet [49]. Det etiske perspektiv bliver dog ændret, når man taler om et system, der automatiserer mange af de funktioner, som QuickStego giver. Et system, der automatisk kan gemme en tekstbesked i et billede, uploade billedet til et socialt medie, og som igen kan modtage et billede og uddrage teksten fra billedet, vil give mange praktiske fordele, idet det vil fungere på samme måde, som et almindeligt chatsystem, blot hvor selve kommunikationen foregår steganografisk.

Sådan et system vil i højere grad kunne anvendes i praksis eller i dagligdagen. Ud fra et konsekvensetisk synspunkt kan det siges, at det ville have en forværrende effekt på efterretningstjenester, da de ville få en ekstra udfordring, da de ikke ved hvem en bestemt person kommunikerer med. Spørgsmålet er dermed, hvad almindelige borgere vil få ud af et sådant system. Det kan siges at gøre det nemmere for kriminelle eller terrorister at gemme deres hensigter og planer. Til gengæld er der tvivl om terrorister overhovedet benyttede sig af kryptografi i terrorangrebet mod Paris i 2015, som et eksempel [75]. På den ene side kan det siges, at hvis det på nuværende tidspunkt er udfordrende at overvåge ikke-krypteret kommunikation, så vil det ikke gøre den store forskel hvis der bliver udviklet noget, som er endnu mere udfordrende. Det vil dermed ikke forværre den nuværende situation betydeligt, men det kan dog evt. have uforudsete fremtidige konsekvenser. På den anden side kan det gavne en befolkning betydeligt, da disse vil have større mulighed for at kommunikere frit, og dermed kan det hjælpe med at sikre ytringsfriheden for de almindelige borgere [64].

Ud fra et pligtetisk synspunkt kan der argumenteres på samme måde. I et samfund som Danmark har staten nogle særlige værdier, der sættes højere end andre. Disse kan opfattes som værende pligter, eksempelvis at staten har en pligt til at opretholde ytringsfriheden og grundlaget for samfundet og demokratiet. Med denne pligt vil det være at foretrække at betragte kryptografi og steganografi som afgørende værktøjer for samfundets integritet. Staten har flere essentielle pligter, som den bør varetage, som at beskytte samfundets borgere, hvilket gøres igennem politiet [53]. Ligesom tidligere nævnt kan det have en forværrende effekt på politiets arbejde, og dermed føre til en forringelse af sikkerheden. Det, der afgør valget, er op til borgerne og staten. Det, de mener, der skal vægtes højest, kommer til at afgøre den etisk korrekte handling. Rogaway argumenterer til gengæld for at sikringen af den første pligt er vigtigere, da det er afgørende for samfundets fremtidige forandring og udvikling at dets borgere har sikret ytringsfrihed [64].

2.1.8 Benyttelse af en offentlig kanal

Kommunikation over kommunikationskanaler kan have konsekvenser for ejeren af den givne kanal. Specifikt ved steganografi, hvor kanalejeren ikke nødvendigvis ved at der foregår kommunikation imellem de involverede parter, kan kanalejeren dermed ikke kan stå inde for hvad der kommunikeres om. Spørgsmålet er således, om det er etisk forsvarligt at udnytte sådan en kanal, uden at ejeren har kendskab til det.

Forholdet mellem kanalen og det steganografiske system kan beskrives ved at sidestille det steganografiske system med en parasitisk organisme. Denne type organisme vil udnytte en anden organisme (en vært) og vil være afhængig af denne

vært i resten af dens levetid [54]. Det steganografiske system er ligeledes afhængigt af dens vært (kanalen), ellers vil der være en brist i kommunikationen. Ved det konsekvensetiske syn, kan det siges at give problemer for kanalejeren hvis det er kriminelle eller lignende hændelser, der kommunikeres om. Sådan et steganografisk system vil ikke bringe noget positivt for den udnyttede kanal.

Problemstillingerne nævnt i disse afsnit kan have indflydelse på programudviklingen fremadrettet, og disse vil uddybes i afsnittet 2.2. Det samme gælder for det næste afsnit, men her står lovgivningen i centrum.

2.1.9 Lovgivning

Der er blevet fastlagt internationale og nationale regler for brugen af kryptografi. Ved udviklingen af et program til steganografisk sikret kommunikation er det essentielt at undersøge regler og krav for området. Da teknologierne er nært beslægtede, tages atter udgangspunkt i kryptering. Til start vil den internationale lovgivning blive dækket, da det er denne lovgivning, som bliver overholdt af det største antal lande.

Brugen af kryptering er dækket under Wassenaar-aftalen [77], der omhandler en række retningslinjer for kontrol af eksport af militært udstyr og våben, samt udstyr, der bruges af militæret og civile. Kryptografiske programmer er dækket herunder [3].

Under disse retningslinjer indgår krav om tilladelser til at eksportere det omtalte udstyr, og brud på dette kan medføre straf så som bøde eller fængsel i op til to år. Wassenaar-aftalen er dog ikke juridisk bindende, den er blot retningslinje for den kontrol, som landene selv udfører. Lande såsom Danmark, Canada og Singapore [6] har f.eks. ingen restriktioner på eksporten af kryptografiske programmer, da de ikke udfører kontrol herpå til at begynde med. Omvendt har lande som Kina, Egypten, Indien mm. [6], derimod indført restriktioner, hvor regeringen kræver en bevilling, før det er muligt at importere og bruge krypteringssoftware.

Andre lande, såsom Rusland, Irak og Nordkorea [6], har helt forbudt importen af udenlandsk krypteringssoftware. Dette kan skyldes, at disse lande har et anstrengt forhold til ytringsfrihed, da de vil undgå at deres befolkning kommunikerer udenom staternes overvågning. De enkelte lande kan også være bange for, at udenlandske regeringer har indført bagdøre i programmet, så de kan følge med i de krypterede samtaler. Dette kan de undgå ved at udvikle deres egne programmer og forbyde importen fra udlandet. Den første begrundelse virker dog mere sandsynlig, når de fleste lande med importforbud også er diktaturer. Der er dog ingen kilder over hvor effektivt disse restriktioner overholdes, da udbredelsen af internettet har gjort det muligt at hente krypteringsprogrammer selv.

Dernæst gennemgås de enkelte lande, hvis love er skrappere end hvad der kræves i Wassenaar-aftalen, både angående import, men også angående brugen af krypteret data. Lande såsom Australien [56], Frankrig [59] og England [57] har indført regler, der giver dommere beføjelse til at kræve det dekrypterede data eller dekrypteringsnøglen af den anklagede, ellers kan den anklagede blive dømt for at have forhindret efterforskningen. Dette data inkluderer krypterede samtaler. Lande såsom Belgien [55], Finland [58] og Holland [60] har også givet deres dommere ret til at kræve data dekrypteret, men dette data kan kun afkræves af vidner, ikke den anklagede selv.

USA [61] skal også nævnes i denne forbindelse, da den anklagede ikke skal frigive dekrypteret data. Derimod kan FBI, takket være PATRIOT act [62], udskrive såkaldte "national security letters", der gør det muligt at gennemsøge personers telefon, email og brevudveksling, uden at skulle bruge en dommerkendelse først, hvis FBI mener personen er en trussel mod national sikkerhed. Derudover har regeringen beføjelse til, grundet en udvidelse af loven om telefonovervågning, at overvåge trafikken på internetforbindelser for mistænkelig aktivitet. FBI har ikke lov til at overvåge selve samtalerne eller hvilke amerikanere, der besøger hvilke internetsider, men regeringen er blevet beskyldt for utilsigtet at havde opfanget denne data alligevel.

Selv med disse udvidelser af deres beføjelser, er institutionen NSA blevet grebet i at overtræde de nye retningslinjer fra PATRIOT, i forbindelse med Edward Snowden skandalen. Skandalen synliggjorde en praksis med skabelse af bagdøre i tjenesterne hos store firmaer, såsom Google. Disse bagdøre gør det muligt at tilgå det data og nettrafik, som firmaet håndterer, med den motivation at finde terrorister. Derudover skabte NSA også programmer (kodenavn PRISM), der direkte gik efter at optage krypterede samtaler og brugsvaner på nettet, både for amerikanere og andre brugere[2].

2.2 Interessenter

I de forrige afsnit om sociale medier 2.1.4, etik 2.1.5 og lovgivning 2.1.9 er et antal interessentgrupper løbende blevet nævnt, der potentielt vil være interesserede i det endelige produkt. Disse opdeles i de to hovedgrupper, angående om de hovedsageligt er interesserede i steganografi eller *steganalyse*, der er identificeringen af steganografiske objekter.

2.2.1 Interessenter for steganografi

Mennesker i samfund med undertrykkelse

Mennesker i samfund med undertrykkelse kan have gavn af et værktøj, der giver dem mulighed for at kommunikere steganografisk. Dette ville give en større frihed til at ytre sig uden at have et styre at forholde sig til. Et sådant værktøj kunne evt. også hjælpe aktivister i disse samfund på vej til at gøre en forskel og hjælpe til at sikre bedre vilkår for befolkingen, fx. ved at starte bevægelser med medlemmer, der har fælles holdninger. Et sådant værktøj kan også hjælpe samfundet med at kommunikere til verden udenfor landegrænserne, uden at samfundets styre bemærker dette.

Aktivister

Dog forsøger ikke blot regimer at få fat i deres borgeres kommunikation, vestlige/demokratiske regeringer gør også. Interessenter for et system for sikker kommunikation er således også blot folk, som sætter pris på at privat kommunikation forbliver privat. Dette går således ud over aktivister og kriminelle.

Journalister

Journalister kan være meget interesserede i at gemme deres data [72], da efterretningstjenester overvåger internettet [15] og dette øger risikoen for at journalisters data bliver overvåget. Af denne grund er journalister en interessent for steganografi, og andre måder at skjule data på [33].

Kriminelle

Kriminelle kan også være interesserede i steganografi, da det kan bruges som et værktøj at kommunikere med, der mindsker risici for at blive opdaget af politiet og lignende autoriteter. Derudover kan det bruges til at gøre data utilgængeligt for politiet, der ellers ville kunne blive brugt som bevise imod de kriminelle.

Terrorister

Terrorister adskiller sig fra de kriminelle ved at deres primære formål er at skabe frygt i en befolkning. Dette opnås oftest ved at udføre angreb eller attentater på højtsiddende embedsmænd eller den generelle civile befolkning. Som led i koordineringen af sådanne angreb benyttes digitale kommunikationsformer (ofte via internettet eller telefonservice). Et netværk af terrorister har dermed en interesse i at udføre deres kommunikation i en skjult facon, således at efterretningstjenester ikke opfanger kommunikationen. Dog er det ikke altid tilfældet at terrorister behøver at kommunikere skjult for at udføre et koordineret terrorangreb, som set

ved angrebet i Paris (2015) hvor kommunikation før og under angrebet blev foretaget via almindelig SMS [18]. Dog er kommunikation, eksempelvis koordinering og vidensdeling, imellem medlemmer af terrororganisationer, der er skjult fra omverdenen, favorabel set ud fra medlemmernes perspektiv.

2.2.2 Interessenter for steganalyse

Efterretningstjenester

Med efterretningstjenester menes myndigheder såsom Forsvarets Efterretningstjeneste (FE) og Politiets Efterretningstjeneste (PET), som er de to statslige myndigheder, som findes i Danmark. Ligeledes findes disse typer af organisationer i andre lande, hvor eksempelvis USA har FBI og Tyskland har BfV [51]. I denne undersøgelse tages der udgangspunkt i det danske PET, men fordi et stort antal lande har lignende organisationer, kunne disse også inddrages og betragtes som værende interessenter.

PET er en national sikkerhedsmyndighed, som har til opgave at bevare friheden, demokratiet og sikkerheden. Myndigheden er i besiddelse af mange afdelinger og værktøjer, som bruges til at løse opgaver, men den del af PET, der er særlig interessant, er afdelingen der beskæftiger sig med overvågning og IT, kaldet Teknisk Center. Dermed er PET en interessent, da steganografi er involveret i overvågningsarbejdet.

Det ville være en fordel for PET at have bedre værktøjer til overvågning, sådan at de bedre kan udføre deres opgaver, eksempelvis hovedformålet, der er at være forebyggende. Ved at være forebyggende, dvs. stoppe kriminalitet før det sker, kan samfundet spare ressourcer, i form af sociale, menneskelige og samfundsøkonomiske omkostninger [50].

Det, der er relevant for denne interessent, er analysen af steganografisk kommunikation, at de kan bevise, at der har fundet steganografisk kommunikation sted, og imellem hvilke personer kommunikationen foregik.

Ejere af kommunikationskanaler

Når der tales om ejerne af kommunikationskanalerne refereres der til ejerne af sociale websider. Dette dækker over alt fra store firmaer som Facebook til små fora drevet af enkelte personer, hvor folk kan snakke sammen og dele data, uden nødvendigvis at kende hinanden. Ejerne af det sociale medie, som der kommunikeres på via steganografi, kan have svært ved at håndhæve reglerne for tilladte tale-emner på websiden, når de ikke kan se samtalen til at begynde med.

2.3 Problemafgrænsning

I det følgende afsnit vil vi afgrænse projektets omfang, således vi kan formulere det egentlige problem, som vi vil forsøge at løse fremadrettet.

Vi har valgt at afgrænse produktet til hovedmålgruppen aktivister, defineret som "medlem af en bevægelse, der forsøger at fremme sin sag gennem aktioner, ofte udenomsparlamentariske"[47]. Dette gør vi da denne målgruppe kan drage gavn af steganografisk sikret kommunikation som modsvar på regeringers digitale overvågning imod dem [71] og resten af befolkningen [69]. De aktivister, som der specifikt tages udgangspunkt i, vil være aktivister, som benytter sig af sociale medier til at kommunikere med. Dette betyder selvfølgelig at de aktivister, der er målgruppen, skal have adgang til internettet og en computer, for at benytte det produkt, som vil blive udviklet til dette formål.

Historisk set er der præcedens for at regeringsinstitutioner misbruger viden om aktivister og samfundsbevægelser, som da FBI i 1964 sendte breve til den ikke voldelige aktivist for afrikansk-amerikanskes rettigheder, Martin Luther King Jr., i et forsøg på at drive ham til at begå selvmord [30]. Endvidere findes eksempler på at lignende overvågning stadig finder sted i dag, såsom Vodafone [5], der er et af verdens største mobilselskaber. Her udtalte Vodafone, at overvågning af selvskabets kunder var muligt for statslige myndigheder, hvor myndighederne havde adgang til at overhøre samtaler, der blev ført gennem Vodafones netværk. [25] [76].

Denne afgrænsning medfører at det steganografiske værktøj, der udvikles, bør opfylde aktivisters behov for sikker kommunikation.

Det er yderligere blevet afgrænset at Facebook vil blive brugt som kommunikationskanal og billeder som dækmedie. Grundet at Facebook komprimerer billeder uanset størrelse, gør at der skal bruges et andet medie, som ikke benytter komprimering, til at dele beskeden over Facebook med. Facebook, hvis brug som kommunikationskanal er analyseret i afsnit 2.1.4, vil gøre produktet tilgængeligt til en større målgruppe. Brug af billeder som stego-objekter vil gøre steganografisk indhold sværere at identificere, da de også vil være skjult i strømmen af billedoverførsler, som dagligt finder sted på Facebook.

2.4 Problemformulering

I det følgende afsnit definerer vi en præcis problemformulering for dette projekt, der vil blive benyttet som udgangspunktet for det videre arbejde. Projektets problem er det følgende:

Aktivister kan ikke kommunikere over offentlige kanaler uden risiko for at blive overvåget.

Hvordan kan vi opbygge et program til aktivister, der kan gemme klartekst i billeder og dele disse over sociale medier? Kommunikation via programmet skal opfylde kravene for sikker steganografisk kommunikation.

Problemformuleringen lægger op til nogle underspørgsmål, som er følgende:

- *Hvordan indlejres beskeden i dækmediet?*
- *Hvordan krypteres beskeden og hvordan dannes nøgler?*
- *Hvordan sikres integritet mht. filformat og komprimering?*

Disse spørgsmål vil på et senere tidspunkt udforme sig som en række krav, som programmet skal opfylde for at problemet kan siges at være løst. Viden omkring disse emner vil blive fundet under teknologianalysen, som danner grundlag for kravene og design af programmet med henblik på aktivister. Det endelige problem er nu defineret, og der kan nu foretages en undersøgelse af de teknologier, der ville være at foretrække for at udvikle et værktøj til steganografi for målgruppen aktivister.

Kapitel 3

Teknologianalyse

Det følgende kapitel tager udgangspunkt i problemformuleringen, og i afsnit 3.4 vil vi først undersøge hvilke metoder til steganografi, der bedst kan anvendes til at gemme data i billeder. Herefter vil vi i afsnit 3.9 undersøge hvilke metoder, der bedst kan kryptere klarteksten, som skal indlejres i dækmediet, således at fortrolighed af data kan sikres yderligere. Endvidere vil vi i afsnit 3.3 undersøge hvorledes vi kan anvende komprimering til at forbedre produktet på. Herefter, i afsnit 3.10, beskrives hvordan brug af Facebook som kommunikationskanal påvirker produktet, især i forhold til komprimering. Teknologianalysen vil ende ud i en kravspecifikation, der indeholder krav for det endelige program, som skal implementeres i C#.

3.1 Least Significant Bit

For at give et bedre indblik i hvordan steganografi på billeder kan fungere, vil vi beskrive *Least Significant Bit* (LSB). LSB er en simpel metode for steganografi, og udnytter det faktum, at antallet af farver i nogle filformater er større end det antal farver, som det menneskelige øje kan se forskel på. Man kan således foretage små ændringer i billedets enkelte pixels og gemme en besked i det uden at vække opmærksomhed.

Dette LSB fungerer ved at man gemmer de enkelte bits, som klarteksten består af, i de mindst signifikante bit, af dækmediet, mens de resterende bit forbliver uændret. Eksempelvis består værdien af et enkelt pixel i et gråtonebillede af et tal mellem 0 og 255. Dette tal repræsenteres på en computer af et enkelt byte, altså 8 bits. Den mindst signifikante bit er således den bit, der har mindst betydning for det endelige tal, i det følgende eksempel er det den bagerste. En ændring på den mindst signifikante bit vil derfor medføre en ændring af den endelige værdi på 1. Et eksempel på brugen af LSB til at gemme en byte af en besked følger.

En byte kan ved brug af LSB gemmes i otte byte fra det valgte dækmedie, se:

11010010

(a) Byte der skal gemmes

01110001	11000010	10101010	11010001
10000111	01010010	10001000	00011101

(b) Viser otte byte fra dækmediet hvori den tidligere byte fra tabel 3.1a skal skjules

01110001	11000001***1***	10101010	11010001
10000110	01010010	10001001	00011100

(c) Viser de otte byte fra dækmediet hvori den tidligere byte fra tabel 3.1a er blevet gemt (ændringer er markeret ved fed og kursiv)

Tabel 3.1: Viser en byte, der gemmes i otte byte ved brug af LSB

Det, som tabel 2.1 viser, er at det byte i tabel 3.1a skal fordeles over de mindst signifikante bits af otte bytes fra dækmediet, der ligger i forlængelse af hinanden, som vist i tabel 3.1b. Den mindst signifikante bit i den første byte fra tabel 3.1b stemmer overens med den første bit fra tabel 3.1a, og derfor skal der for denne byte ikke foretages nogle ændringer. Den mindst signifikante bit i den anden byte i tabel 3.1b stemmer derimod ikke overens med den anden bit i tabel 3.1a, og der skal derfor ske en ændring. Alle de mindst signifikante bits i tabel 3.1b bliver altså ændret så de stemmer overens med bitværdierne fra tabel 3.1a. Efter dette er gjort, bliver resultatet det, som ses i tabel 3.1c.

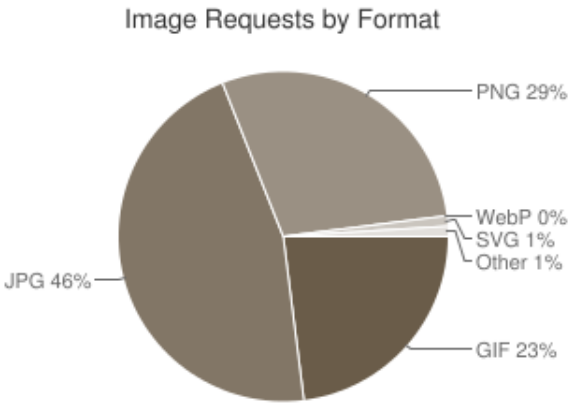
For atter at få den originale byte-meddelelse ud fra resultatet, skal de mindst signifikante bits fra alle byte i tabel 3.1b blot samles. Dette medfører selvfølgelig også at det ved brug af LSB er meget let for opponenter at læse beskeden. For at undgå dette, kan der bruges andre former for digital steganografi på billeder; metoder til dette vil blive behandlet i senere afsnit.

3.2 Analyse af filformater

I dette afsnit undersøges hvilken filtype, der er mest velegnet at anvende som dækmedie i den endelige løsning. Da steganografi består i at gemme data i tilsyneladende uskyldig data, kan det være nyttigt at benytte et filformat, som er udbredt og ofte benyttes alment, så det ikke skiller sig ud fra mængden. Derudover er det også relevant at benytte en filtype, som håndterer data på en hensigtsmæssig måde, idet filtypen skal være kompatibel med steganografisk indkodning.

I dette afsnit belyses hvorledes de udbredte billedformater JPEG, PNG, GIF, TIFF og BMP fungerer, og om de er oplagte som dækmedie for steganografiske beskeder.

Valget af JPEG, PNG og GIF er baseret på figur 3.1, som viser hvorledes JPEG, PNG og GIF er de dominerende filformater på nettet. Derudover inddrages BMP samt TIFF formaterne, da BMP er et simpelt filformat i sammenligning med disse andre, i forhold til opbygningen af filtypen, og TIFF er en *tabsfri* standard, som tidligere har været populær indenfor billedmanipulation og publikation (scannere, faxmaskiner osv.).



Figur 3.1: Denne graf viser filformaternes popularitet baseret på antallet af HTTP-requests [28].

3.2.1 JPEG

JPEG (Joint Photographic Experts Group) er et *tabsgivende* filformat. Dette betyder at JPEG billeder er udsat for komprimering idet JPEG-indkodningen foretages, hvori der forekommer datatab. Denne billedtype er populær, da filstørrelserne kan være meget små uden at miste essentiel visuel data. Mild JPEG-komprimering minimerer filstørrelsen, uden at fratage billedet dets synlige detaljer. Kraftig JPEG-komprimering kan kraftigt minimere filstørrelsen, men skaber dog synlige artefakter (tab af essentiel visuel data).

Da JPEG er et tabsgivende format, er det "ikke essentielle" data allerede fjernet, hvilket kan skabe store problemer for LSB-algoritmer. Måden hvorpå information gemmes i JPEG-formatet er en yderligere udfordring for steganografisk indkodning.

Dog er det muligt at benytte steganografiske metoder til at indlejre beskeder i et allerede indkodet JPEG billede. Dvs. data kan modificeres i JPEG filen. Men gennemgår det steganografisk indkodet billede endnu en JPEG indkodning, så vil det indlejrede data gå tabt, idet JPEG benytter en tabsgivende komprimering. [84]

3.2.2 PNG

PNG (Portable Network Graphics) er et filformat, som er populært på websider, og udsættes for tabsfri komprimering ved indkodning. Der sker således ikke datatab ved at benytte PNG-formatet. PNG tillader desuden, modsat andre formater, kontrol over billedets enkelte pixels gennemsigtighed. Den implicitte komprimering, der ikke formindsker billedkvaliteten, samt mulighed for gennemsigtighed (mulighed for at bruge billeder, som ikke fremstår firkantede), gør PNG til en foretrukken standard indenfor mange felter.

PNG har forskellige standarder for at håndtere pixeldata. Figur 3.2 viser hvorledes bit fordeles alt efter hvilken "color option" benyttes til indkodning af PNG-filer. Kanalen, som definerer gennemsigtighed (alpha), er ikke medtaget i alle muligheder.

Bits per pixel						
Color option	Channels	Bits per channel				
		1	2	4	8	16
Indexed	1	1	2	4	8	
Grayscale	1	1	2	4	8	16
Grayscale and alpha	2				16	32
Truecolor	3				24	48
Truecolor and alpha	4				32	64

Figur 3.2: Denne figur viser hvorledes PNG filer er opbygget, alt efter hvilken farvetype den er. [85]

I forhold til steganografiske metoder, så er PNG-filtypen et godt valg. Da der ikke sker datatab i processen, er det ikke nødvendigt direkte at forholde sig til om hvorvidt den steganografiske besked skulle gå tabt. Tages der udgangspunkt i "truecolor and alpha" standarden til LSB-indkodning, er der mange bit at indkode data i. Der vurderes derfor at PNG er et oplagt filformat for et dækmedie.[85]

3.2.3 TIFF

TIFF (Tagged Image File Format) var tidligere et populært format, indenfor publikation af medier, idet formatet ejes af Adobe Systems [86]. Endvidere understøttes formatet også i .NET. Virksomheder og programmer, som arbejdede med billedhåndtering, benyttede tidligere dette format som standard. Formatet er dog

efterhånden udfaset og erstattet af mere moderne alternativer. TIFF er et tabsfrit filformat ligesom PNG.

3.2.4 GIF

GIF (Graphical Interchange Format) er et populært valg af billedformat på websider. Dette er grundet at GIF ikke fylder meget. GIF giver kun mulighed for 256 farver. Dvs. at billeder med mange farver eller nuancer ikke er egnet til at bruge GIF formatet. Derimod giver GIF mulighed for animationer. GIF kan indkodes med flere sekventielle billeder i en container, hvorved billedet kan afspilles og gentages som en animation. Denne mulighed for animation, samt begrænsningen for 256 farver gør at GIF er udbredt brugt på websider. GIF benyttes sjældent i en kontekst udenfor websider. [82]

GIF-indkodningen benytter en algoritme til tabsfri komprimering, LZW [31], som ikke vil være et problem for indlejring af data, via steganografiske metoder.

3.2.5 BMP

BMP (Bitmap) er et filformat, udviklet af Microsoft til brug i Windows. BMP formatet er et simpelt format idet det gemmer pixeldata i et struktureret data-array. Der findes altså specifik bitdata, som beskriver hver enkelt pixel i billedet. BMP, som andre formatter, indeholder en header-sektion hvori filen beskrives, men herefter findes det rå pixeldata.

BMP benytter bit-padding i sin filopsætning. Efter hver datalinje i filen findes en mængde bit, sat til side til at agere som padding. Længden af en datalinje er defineret af den horisontale dimension af billedet, som bliver defineret i filheaderen. Disse paddingbit kan potentielt bruges til at indkode data. På denne måde ændrer en indkodet besked ikke nødvendigvis den visuelle repræsentation af billedet.

BMP er derfor, ligesom PNG, et oplagt format at benytte til steganografi, da pixeldata og formatteringen af filen er simpel og lineær (data befinder sig i logisk rækkefølge). [81]

3.3 Metoder til komprimering

Metoder for komprimering, herunder komprimering af billeder, kan opdeles i tabsgivende og tabsfri komprimering. JPEG, og PNG er kun to af mange billedformater, som implicit benytter sig af komprimeringsalgoritmer. Der henvises til afsnit 3.2 for mere information om hvilke billedformater, der hører under hvilken kategori. Grunden til at billeder bliver komprimeret skyldes, at der er et ønske om at mindske den mængde data et billede fylder, således det er nemmere at håndtere, når billedet skal kommunikeres via internettet [66].

3.3.1 Tabsgivende komprimering

Et billede, der er blevet komprimeret tabsgivende, er et billede, som har været udsat for elimination af hovedsageligt redundant information. Denne information er gået tabt og kan ikke skaffes tilbage. Det er i visse tilfælde svært for det menneskelige øje at se, om noget er blevet slettet, men med en computer er det oplagt ved sammenligning med det originale billede [67]. Grunden til at mennesket ikke vil kunne se forskel på det originale og det komprimerede billede skyldes, at det menneskelige øje kan se ændringer på lyse og mørke farver, altså gråtoner, men hvor øjet er mindre effektiv til at se ændringer på farver (RGB). Visse filformater, i dette tilfælde JPEG, udnytter dette. [20]

3.3.2 Tabsfri komprimering

Tabsfri komprimering er i modsætning til tabsgivende komprimering, en komprimeringsmetode, som garanterer at man helt uden tab kan genskabe det data, som var blevet komprimeret. [67]

Disse to komprimeringsmetoder har hver deres fordele og det er vigtigt at vide hvilken metode de forskellige billedformater bruger. Det er blevet afgjort at et tabsfrit filformat er foretrukken, da eventuelle datatab er svære at undgå og, hvis disse optræder, kan det skade dataets integritet betydeligt.

Det er dog stadig muligt at bruge filformater, som er tabsgivende, såsom JPEG, da komprimeringsproblemet blot opstår når der konverteres fra et filformat til et andet. Det vil sige at steganografi godt kan anvendes på JPEG filformatet, dog er det her essentielt at filen beholdes i JPEG filformatet i hele processen, da der ellers kan mistes data. Det er dog stadig muligt at beholde størstedelen af dataet efter indlejring ved komprimering fra et tabsfri til tabsgivende filformat, men aldrig med 100 procent præcision. [13]

3.4 Metoder til steganografi

For at vælge en passende steganorafisk metode, gennemgår vi tre forskellige metoder på et generelt plan, hvor idéen bag metoden forklares, og derefter hvilke fordele og ulemper disse metoder besidder i forhold til problemformulering. Metoderne, som vi har valgt at gennemgå, blev valgt med udgangspunkt i, at de kan anvendes på billedfiler. Metoden, som arbejdes med fremadrettet, vælges på baggrund af de beskrevne fordele og ulemper.

3.4.1 Hamming kode

Hamming kode bliver brugt til at finde og rette små fejl, bestående af enkelte bits i digitale opbevaringsmedier, såsom CD-er og harddiske[83]. Hamming kode kan

også bruges til at gemme data steganografisk og gør det muligt at gemme adskillige bits i dækmediet, ved kun at ændre på enkelte bits[79].

Metoden virker ved, at der konstrueres en *Hamming matrix* $(r \times k)$, der skal indeholde samtlige mulige kombinationer af tal indeholdene 1- og 0-taller. Hvert unikke tal står i sin egen kolonne i matricen, hvor antallet af cifre i hvert tal er bestemt af antallet af rækker (r) i matricen. Antallet af rækker i matricen er igen bestemt af antallet af bits, der ønskes gemt. Denne blok af bits, der gemmes, er en ud af flere blokke, der tilsammen udgør alle bits i den hemmelige besked.

Vi skal konstruere en Hamming matrix, der bliver nøglen til at kryptere og dekryptere beskeden med. Hvis man ønsker at gemme 3 bits af gangen i dækmediet, skal en Hamming matrix derfor have 3 rækker. Hver kolonne (k) i matricen skal indeholde et tal, der er en unik kombination af cifre bestående af 1- og 0-taller. Antallet af disse kolonner, der er nødvendige for at indeholde samtlige kombinationer, findes ved hjælp af en simpel formel: $kolonner = 2^r - 1$

Efter matricen er blevet konstrueret, skal der aflæses bits fra dækmediet. Disse bits aflæses i blokke, hvor antallet af bits i hver blok er bestemt af antallet af kolonner i Hamming matricen. Hver blok ganges på matricen, hvilket giver en vektor $(r \times 1)$, hvis cifre svarer til de bits, der ønskes gemt i dækmediet.

Hvis vektoren ikke svarer til de ønskede bits, skal der ændres på enkelte bits fra dækmediet, indtil vektoren passer overens med de bits, der ønskes gemt. Takket være Hamming matricen er det kun få bits, så lidt som en eller to, der skal ændres i dækmediet. Processen gentages for hver blok, der aflæses fra dækmediet.

For at aflæse den hemmelige besked, skal bits fra dækmediet aflæses og ganges på den Hamming matrix, der har det samme antal rækker og kolonner som den matrix, der blev brugt til at indkode beskeden med. Herefter kan matrix-vektorproduktet aflæses for de hemmelige bits.

Fordele

Sikkerhed Metoden er i stand til at gemme en besked, ved at ændre på færre bits i dækmediet, end antallet af bits i den hemmelige besked. Samtidig bliver disse ændrede bits fordelt tilfældigt ud i dækmediet, hvorefter de ikke kan skelnes fra almindelig støj i mediet. Dette sikrer, at beskeden ikke bliver detekteret af statistiske analyser.

Ulemper

Bæreevne Hvor sikkert beskeden er gemt i dækmediet, påvirker pladsen til beskeden. Dette skyldes, at sikkerheden afhænger af størrelsen på Hamming

matricen, hvor jo større matricen er, jo færre bits skal der ændres ud i hele dækmediet, før beskeden er gemt. Omvendt betyder det, at der skal aflæses flere bits per blok, for at kunne indlejre beskeden. Takket være formlen $kolonner = 2^r - 1$ stiger antallet af aflæste bits eksponentielt. En Hamming matrix på 3 rækker har således brug for 7 bits per blok, en matrix på 4 rækker har brug for 15 bits af gangen og en matrix på 5 rækker har brug for 31 bits i hver blok. Dette betyder, at programmøren kan blive tvunget til at ofre sikkerhed på grund pladsmangel, hvilket er en uønsket situation.

Nøgle Den eneste nøgle der kræves, for at låse beskeden op, er en Hamming matrix med den rigtige størrelse. Størrelsen på Hamming matricer vokser eksponentielt, takket være formlen $kolonner = 2^r - 1$. Eftersom antallet af kolonner i matricen vokser eksponentielt i forhold til antallet af rækker, vil tidsforbruget for multiplikation på en Hamming matrix også stige eksponentielt, jo større blokke af bits der ønskes gemt. Dvs. at det ikke er muligt at gennemgå samtlige mulige Hamming matricer, før matricerne bliver så store, at det vil tage for lang tid for en computer at kryptere eller dekryptere en besked. Denne grænse gælder både for sender, modtager og opponent, hvorfor det er muligt for opponenten at låse op for beskeden, ved at prøve samtlige matricer, som det er muligt at arbejde med. Det er derfor op til programmøren at implementere et nøglesystem, der forhindrer opponenten i at låse op for beskeden.

3.4.2 Paletbaseret metode

Den paletbaserede metode [23] er en relativ simpel steganografisk metode til at gemme data i et palettebaseret billedformat, såsom GIF. For at gemme en besked, tages modulo(2) på én udvalgt pixels sammenlagte farveværdier (summen af RGB værdierne), hvorefter modulo-værdien sammenlignes med den bit, der ønskes gemt.

Den enkelte pixel udvælges af et nøglesystem, der ikke er nærmere beskrevet i kilden. Hvis dette bit og modulo-værdien ikke er ens, bliver der søgt efter en ny pixel i billedet, hvis farveværdi er tæt på den originale pixel, men som har en modulo-værdi, der svarer til de bit, der skal gemmes. Den originale pixel og den nye pixel-plads i billedet skrives op, og processen gentages, indtil rækkefølgen af modulo(2)-værdier fra billedets pixels svarer til rækkefølgen af bits fra det hemmelige data.

Det hemmelige data kan så aflæses igen ved at tage modulo-værdierne for enkelte pixels i billedet, der er udvalgt af nøglen, og oversætte disse til de tilsvarende bits.

Fordele

Detektion Forhindrer detektion fra statistisk analyse, da beskeden gemmes ved at bytte rundt på eksisterende pixels, og dermed ikke ved at ændre på LSB i de enkelte pixels. Den samlede værdi af alle pixels forbliver således den samme.

Implementation Den bagvedliggende matematik er relativ simpel. Den eneste matematiske forståelse, der kræves, er brugen af modulo og afstandsformlen. Dette vil hjælpe med at simplificere implementeringen af funktionen i selve programmet.

Ulemper

Nøgle Fridrich [23] har selv implementeret et nøglesytem, men går ikke i nærmere detalje om dens konstruktion, hvilket gør at at det er op til programmøren selv at implementere et nøglesystem.

Bæremedie Metoden er udviklet til kun at arbejde med RBG-pixels. Det er muligt at arbejde med grayscale pixels, men dette beskrives ikke nærmere i [23] og det vil dermed være op til os selv, hvis vi ønsker at bruge denne metode på grayscale billeder.

3.4.3 Grafbaseret metode

Metoden beskrevet i kilde [27] er mere avanceret end den paletbaserede metode, da den er i stand til at skjule beskeder i så forskellige dækningsmedier som billeder- og lyd-filer. Denne korte gennemgang vil dog fokusere på brugen af billeder som dækmedie.

Kerneidéen i *den grafbaserede metode* er at bytte om på pixels, sådan at de kommer til at svare til den hemmelige besked. Der arbejdes dog ikke på enkelte pixels, men på et bestemt antal af dem, som samlet skal svare til en mindre del den hemmelige besked.

Det har den fordel, at man har flere valgmuligheder, når det kommer til at skulle bytte en pixel med en anden, da der skal bruges flere pixels for at repræsentere en del af den hemmelige besked.

Idéen med metoden er at konstruere en graf, der består af knuder og kanter. Knuderne i grafen består hver af en mindre gruppe af pixels, hvor der i [27] bruges 2-4 pr. gruppe. Den enkelte knude vil blive sammenlignet med andre knuder i grafen med lignende værdier, for at finde pixels, der kan bytte plads med de første.

Kanterne i grafen forbinder knuder, der kan bytte plads. Der kan indføres restriktioner, sådan at ikke alle knuder kan danne en kant imellem hinanden. En restriktion kunne være, at en pixel ikke må byttes med en anden pixel, hvis der er for stor afstand imellem deres respektive farveværdier. Dette kan forhindre at det er let at se at billedet er blevet modificeret. Restriktionen medfører at der f.eks. ikke byttes en sort pixel ud med en rød pixel.

For at kunne aflæse beskeden igen skal pixels fra billedet blot aflæses i knuder af den samme størrelse, som de blev indlæst med, hvorefter der tages modulo på knuderne, så de hemmelige bits kan aflæses.

Fordele

Sikkerhed Fordi der bliver byttet rundt på dataet i dækmediet i stedet for at ændre i de enkelte bits, er det ikke muligt at detektere beskeden via statistisk analyse, hvilket gør metoden mere sikker. Desuden, fordi der bliver sat restriktioner på hvilke ombytninger, der kan foretages, er det med til at gøre, at det menneskelige øje ikke kan skelne imellem et stego-objekt og det ubehandlede dækmedie [29].

Bæreevne Hvis metoden bruges på hver enkel af de tre farvekanaler i en pixel, er det muligt at gemme tre bits per pixel, hvilket giver en større datatæthed end den paletbaserede metode og Hammingmetoden.

Fleksibilitet Metoden er forholdvis fleksibel, idet der kan anvendes forskellige dataformater som input.

Ulemper

Kompleksitet Den grafbaserede metode er matematisk mere kompliceret end Hamming- og den palettebaserede metode, hvilket kan gøre at implementationen af metoden i programmet er mere kompliceret end de to andre.

3.4.4 Valg af metode

Vi har valgt at arbejde videre med den grafbaserede metode, da den tilbyder den bedste udnyttelse af plads i bæremediet, samtidig med at tilstedeværelsen af beskeden er svær at detektere for opponenten. For at kunne implementere metoden i vores program vil vi i det følgende afsnit foretage en mere detaljeret gennemgang af den grafbaserede metode.

3.5 Gennemgang af den grafbaserede metode

For at kunne implementere den grafbaserede metode beskrevet i [27], skal der præcist defineres, hvordan metoden virker og hvordan den bruges. Derfor vil der først være et underafsnit, der beskriver hvilke begreber, der bruges, og hvordan de er defineret. Efterfølgende kan disse begreber anvendes, sådan at vi beskrive konstruktionen af en graf ud fra et dækmedie.

3.5.1 Begrebsdefinition

Den mindste enhed, som der arbejdes med, kaldes for en *sample, s*. Hvis der arbejdes med et billede, som dækmedie, vil en sample svare til én pixel. Hvis billedet er et truecolor billede, vil denne sample have en værdi for henholdsvis R, G og B, hvorimod et grayscale billede kun vil have én værdi pr. pixel. Dermed kan en sample have flere forskellige værdier, da dækmedierne kan variere. Den mængde af værdier, som en sample kan have, betegnes med $\mathbb{S}$. Det samlede dækmedie består dermed af et array af samples.

Det første der gøres for et nyt dækmedie, er at der tildeles en indlejret værdi for enhver sample. Hvis man har et dækmedie $C = \langle s_1, ..., s_N \rangle$ og en mængde $P \subseteq \{1, ..., N\}$, så vil *frekvensen* af *sampleværdien* $x \in \mathbb{S}$ i mængden P være:

$$|\{i \in P | s_i = x\}|$$

Herefter antager vi at en funktion har tildelt en indlejret værdi, for hver sample, for et givet dækmedie.

$$v : \mathbb{S} \rightarrow \{0, ..., m-1\} \tag{3.1}$$

Hvis m sættes til 2, udføres der traditionel LSB, da sampleværdierne, ud fra v, kun kan være *0* eller *1*. Dermed kan m varieres, i forhold til det benyttede dækmedie.

I stedet for at indlejre data i en enkelt sampleværdi, bruges der k sampleværdier, sådan at der er flere valgmuligheder, når sampleværdierne skal modificeres. For at beregne denne værdi, tages de indlejrede værdier fra tidligere, hvorefter de indlejrede værdier summeres og der tages modulo m af dem.

Antag at vi har et dækmedie C.

$$C = \langle s_1, ..., s_N \rangle \tag{3.2}$$

Antag desuden at vi med funktion v har tildelt indlejrede værdier, til hver individuel sample.

Den indlejrede værdi, for k samples, defineres for $1 \leq i \leq \frac{N}{k}$ på følgende måde:

$$V_i(C) = v(s_k \cdot (i-1)+1) \oplus_m ... \oplus_m v(s_k \cdot (i-1)+k) \tag{3.3}$$

i vil altid være større eller lig med 1, da den mindste dataenhed, som der kan arbejdes på, er 1 pixel. Desuden skal i være mindre end eller lig med $\frac{N}{k}$, da dette tal ($\frac{N}{k}$) angiver antallet af samlede indlejrede værdier.

Hemmelige besked

Betegnelsen for den hemmelige besked er som følger:

$$E_m = (e_1, ..., e_n) \text{ hvor } e_i \in \{0, ..., m-1\} \text{ for } i \in \{1, ...n\}$$

Grunden til at den hemmelige besked beskrives på denne måde, er for at sikre, at det indlejrede data er indkodet som cifre, som ikke bliver større end m.

Selve grafen består af en mængde af knuder og en mængde af kanter. Grafen betegnes ved G, knuderne med V, og kanterne med E. Således består G af *(V,E)*, hvor $E \subseteq V \times V$. En graf kan enten være orienteret eller uorienteret, hvor der i dette tilfælde udelukkende arbejdes med uorienterede grafer. Hvis der findes en kant mellem x og y, $(x, y) \in E$, så vil det være den samme kant, som $(y, x) \in E$, hvis grafen er uorienteret.

Indførelse af restriktioner

En restriktion, som vi beskrev tidligere i afsnit 3.4.3, beskrives herfra, som en vægt. Kanter kan tildeles vægte, som betegnes med $c(e)$ for $e \in E$.

Pardannelse

En pardannelse defineres som $M \subseteq E$, hvor der ikke eksisterer to kanter e_1, $e_2 \in M$ og en knude $v \in V$ sådan at e_1 og e_2 er forbundne til v.

Parring

En kant er *parret*, i forhold til en eller anden parring M, hvis $e \in M$.

Parring af en knude

En knude er kaldet parret i M, hvis der er en kant i M, som danner en forbindelse med (incident to) v.

Der bruges også anden notation for at beskrive, at en knude er parret i M, hvilket skrives på følgende måde: $v \in M$ for en knude $v \in V$. Hvis knuden ikke er parret i forhold til M, beskrives det med følgende notation: $v \notin M$

Perfekt parring

Hvis alle knuder i en graph er blevet parret, så bruges betegnelsen perfekt parring for at beskrive dette.

Da de nødvendige begreber er blevet defineret, kan vi nu konstruere en graf, baseret på et dækmedie.

3.6 Konstruktion af en graf

Formålet med dette afsnit er at beskrive fremgangsmåden til at udføre steganografi vha. den grafbaserede metode. For at gøre dette konstrueres en graf ud fra et dækmedie. Desuden defineres størrelsen af en knude, dvs. hvilket data en enkelt knude består af. Herefter beskrives hvordan vi opretter kanter mellem knuder, hvorefter der indføres en vægt til den enkelte kant, som udregnes på baggrund af to sampleværdier.

3.6.1 Grafen

I denne sektion beskrives hvordan en graf konstrueres.

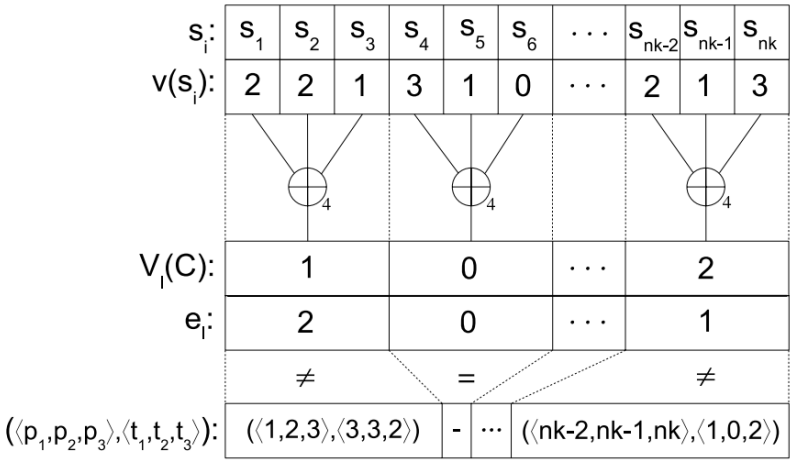
Først indhentes et dækmedie C.

$$C = \{s_1, ..., s_N\} \tag{3.4}$$

$k \geq 1$, hvor k og m kan varieres i forhold til dækmediet. Desuden beskrives k, som værende forholdet mellem samples og knude. Dvs. at hvis $k = 3$, så bruger vi tre samples pr. knude

Generel beskrivelse af en knude
En knude har følgende struktur (P, T) hvor $P = \langle p_1, ...p_k \rangle \in \{1, ..., N\}^k$ er en *k-tupel* af positioner i dækmediet og $T = \langle t_1, ...t_k \rangle \in \{0, ..., m-1\}^k$ er en *k-tupel* af målværdier.

Én af sampleværdierne, s_{pi}, hvor $i \in \{1, ..., k\}$ har brug for at blive ændret til en sampleværdi s_i^*, hvor $v(s_i^*) = t_i$ skal virke som indlejringen af en bestemt del af den hemmelige besked.



Figur 3.3: [27] Eksempel på konstruktionen af en knude, med $k = 3, m = 4$

Det ovenstående beskriver strukturen for én enkelt knude, hvor vi mangler at beskrive hvordan den resterende mængde af knuder skal konstrueres. Dvs. at der skal laves én knude, for hver *k-tupel* af sampleværdier, hvor der kun skal ændres én enkelt af disse k sampleværdier. Figur 3.3 er et eksempel på en konstruktion, for et helt dækmedie C og en hemmelig besked $E_4 = \langle e_1, ..., e_n \rangle$

3.6.2 Gennemgang af figur

Vi foretager en kort gennemgang af figur 3.3, sådan at det står klart hvordan alle beregninger foretages, og dermed hvordan man kommer frem til de tal, som der står i figur 3.3.

I den første række af en knude ligger hver sample, og hvis der arbejdes med et billede som dækmedie, vil det være en pixel. I anden række ligger den indlejrede værdi, for enhver sample, og på dette tidspunkt er der ikke foretaget nogle ændringer i dækmediet. Måden man kommer frem til den indlejrede værdi, er ved at definere en funktion v, som kan tage farveværdien på en pixel, og på baggrund af denne farveværdi tildele en indlejret værdi.

I tredje række ligger den værdi, som skal svare til den hemmelige besked. Værdien betegnes med $V_i(C)$. Måden man kommer frem til denne værdi, er ved at summere de indlejrede værdier, som indgår i den pågældende knude, og herefter tage modulo m af summen. Dvs. at $V_i(C) = (2 + 2 + 1) \mod 4$ for den første knude.

Herefter sammenlignes $V_i(C)$ med den i'te del af den hemmelige besked (e_i). Hvis de er ens, som det er i tilfælde nummer to på figur 3.3, skal der ikke foretages mere og der oprettes ikke en knude. Hvis de derimod ikke er ens (som det er tilfældet i den første og den tredje knude), oprettes der en knude i den sidste række, ved at beregne nogle målværdier for enhver sample. De målværdier beregnes ved først at beregne en forskel d

$$d = e_i \ominus mV_i(C) \tag{3.5}$$

Herefter adderes d til hver enkel $v(s_i)$, hvilket resulterer i en målværdi (t_i), for hver enkel sample $v(s_i)$. Grunden til at vi laver disse beregninger er, at hvis en knude skal ændres sådan at $V_i(C) = e_i$, så skal den ene $v(s_i)$ i en knude, ændres til dens korresponderende målværdi t_i.

3.6.3 Ombytninger og farveforskel

Dette afsnit har til formål at forklare hvordan kanter benyttes i den konstruerede graf, samt hvordan der tages højde for forskelle i farve blandt de pixels der måtte være en kant imellem, sådan at ombytninger, der hvor der kan ses forskelle i farve, formindskes.

Lad v og w være to knuder med
$v = (\langle p_1, ..., p_k \rangle, \langle t_1, ..., t_k \rangle)$ og $w = (\langle q_1, ..., q_k \rangle, \langle u_1, ..., u_k \rangle)$ og lad $i, j \in \{1, ..., k\}$. Her vil der være en kant, der forbinder den i-ende *samplеværdi* af v og den j-ende *samplеværdi* af w skrevet som $(v, w)_{i,j} \in E$ hvis:

$$v(s_{p_i}) = u_j$$

$$v(s_{q_j}) = t_i$$

En kant forbinder to knuder og bliver mærket med et index af en sampleværdi for hver knude. En ombytning af disse sampleværdier vil resultere i indlejring af begge knuder. Det er endvidere muligt for to knuder at have mere end én kant mellem dem. Da en parring kun kan indeholde en kant, vil der i dette tilfælde kun vælges en af kanterne, for ikke at indlejre de samme knuder mere end én gang.

Den ovenstående definition tillader ombytninger hvor det kan lade sig at gøre, men tager ikke hensyn til at ombytningerne kan skabe visuelle forskelle fra det originale dækmedie, eksempelvis hvis en sort og blå pixel ombyttes, vil dette være tydeligt at se. For at undgå dette skal der defineres en restriktion på mængden af kanter, som tager hensyn til visuelle forskelle på de forskellige pixels. Hertil kan der defineres en afstandsformel $d : \mathbb{S} \times \mathbb{S} \to \mathbb{R}$, som har til formål at finde den visuelle distance mellem de to pixels. Hertil defineres en lighedsrelation for visuel

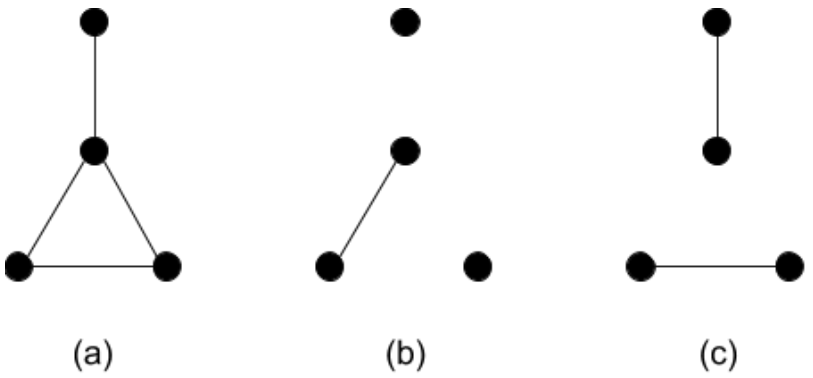
lighed $\sim$ ud fra afstandsbegrebet $s_1, s_2 \in \mathbb{S}$ som: $s_1\ s_2 \Leftrightarrow d(s_1, s_2) \leq r$ for en valgt radius r. Formlen for mængden af par med restriktion af $\sim$ kommer ud fra dette til at være følgende:

$$E^{\sim} = \{(v, w)_{i,j} \in E \mid s_{p_i} \sim s_{q_j}\}$$

Dette medfører at der sættes en begrænsning på hvor langt væk de to pixels må være fra hinanden med hensyn til farveværdier. Dette vil medføre at hvis en blå pixel kan bytte med en sort, vil dette (afhængig af r) ikke ske, hvis forskellen mellem den blå og sorte pixel er større end r.

3.7 Pardannelse

Efter at vi har konstrueret en graf, bestående af en mængde af knuder og en mængde af kanter, er der det tilbage, at der skal beregnes en perfekt parring.



Figur 3.4: Eksempel på tre forskellige uorienterede grafer

Figur 3.4 illustrerer forskellen mellem en perfekt parring (c) og en maksimal parring (b), for en graf (a). Når vi beregner en graf, vil én enkelt knude have flere kanter, ligesom (a). Vi ønsker en algoritme, der kan beregne en perfekt parring, sådan at alle knuder bliver indlejret ved hjælp af ombytning, sådan at der ikke er nogle knuder som skal modificeres, hvilket er hele idéen med den grafbaserede metode.

3.7.1 Pardannelsesproblemet

Pardannelsen for uorienterede grafer er et NP-fuldstændigt beslutnings problem, som har den betydning at der formodentligt ikke findes en algoritme, som kan finde en perfekt pardannelse indenfor polynomial tid.[24]

Der er flere tilgange der kan bruges til at finde pardannelser, hvor vi vælger en heuristisk tilgang, for at undgå en eksponentiel tidskompleksitet.

Først sorteres alle knuder i grafen, efter hvor mange kanter (muligheder) den enkelte knude har, sådan at når vi begynder processen, som skal finde den udvalgte kant, så tager vi de knuder med færrest muligheder først. Desuden vælges den kant med mindst vægt, ud af den mængde kanter, som en knude besidder for så vidt muligt at sikre, at stego-objektet ligner dækmediet. Dette er principperne fra pseudokode algoritmen "The static minimum degree construction heuristic (SMD)"[27], som vi også vil benytte os.

3.7.2 Pardannelsesalgoritme på pseudokode

Algoritmen er taget fra [27], dog med en mindre ændring, som tager højde for den datastruktur, som vi benytter. Ændringen består i, at vi kun skal aktivere nogle bestemte knuder. Helt præcist, skal det være de knuder i listen af knuder, som ikke allerede svarer til den hemmelige besked af ren tilfældighed.

Input: Graf $G = (V, E)$
Output: Parring M på G

Alle knuder sorteres efter antal kanter i stigende rækkefølge, $\langle v_i, ..., v_n \rangle$;
Initialisér $M = \emptyset$;
Marker de knuder, som ikke allerede er parret ved tilfældighed, som aktive;

for *i = 1,...,n* **do**
  if *Hvis* v_i *er aktiv og* $valens(v_i) > 0$ **then**
    Sæt $e = (v_i, w) =$ *korteste kant af* v_i;
    Sæt $M = M \cup \{e\}$;
    Marker v_i *og* w *som inaktiv og slet alle deres kanter fra* E
  end
end
return *M;*

Algoritme 1: Algoritme for pardannelse

3.8 Nøgle

I sammenhæng med den grafbaserede algoritme, er der brug for en nøgle, således at det skjulte data kan forblive skjult, på trods af oponenten, som ikke besidder nøglen, men måske kender til algoritmen. På denne måde er den indlejrede besked holdt sikker.

Elementet i algoritmen, som tillader at generere og benytte en nøgle, består i udpegningen af pixels i billedet. I et billede med n pixels benytter vi kun en del af disse. De pixels, som bliver udpeget til at indgå i algoritmen, er bestemt ud fra en streng af tilfældigt genererede tal. Strengen af tilfældige tal er bestemt ud fra en tilfældighedsgenerator (i .NET). Generatoren kan benytte sig af et såkaldt *seed* (frø), til at generere et mønster af tilfældige tal, og så længe dette seed er det samme, vil tilfældighedsgeneratoren udvikle akkurat samme mønster af tilfældige tal. Tallene er dermed ikke tilfældige, men pseudo-tilfældige (Se 3.8.3).

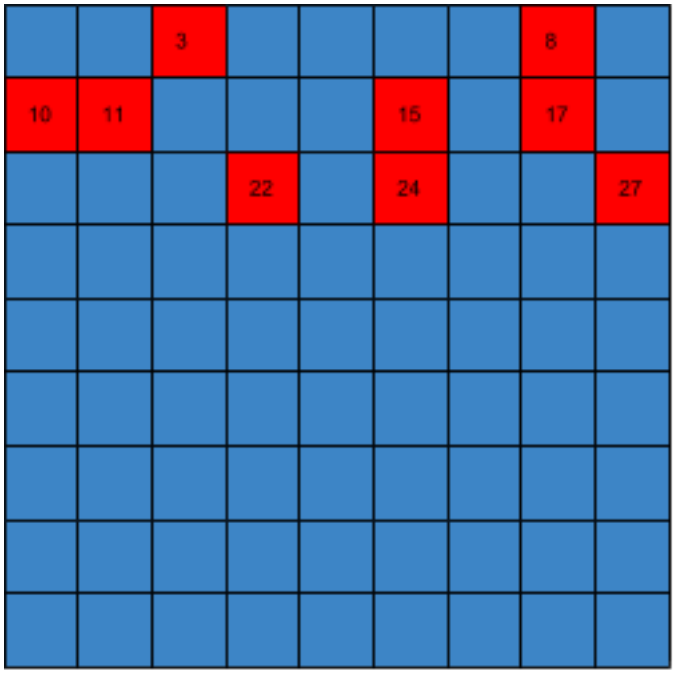
Vi har besluttet at implementere to forskellige nøglegenereringsmetoder, der tager samme inputparametre, og hvor begge metoder returnerer en liste af pixels, som er blevet udvalgt fra dækmediet, sådan at vi kan udskifte disse metoder med hinanden. Vi har valgt at implementere begge, i tilfælde af at den enes udvælgelsesproces senere viser sig ikke at være tilstrækkelig til formålet. Idet de to metoders fremgangsmåde er forskellige, er der forskellige variabler som kan ændres, for at ændre deres opførsel. I dette afsnit beskrives disse metoder.

3.8.1 Metode 1

Et array af tilfældige tal bliver genereret (indenfor interval $[x, y]$), med arraylængde n.

Eksempel:
Array = [3, 5, 2, 1, 4, 2, 5, 2, 3] med interval *[1;5]* og $n = 9$.

Denne streng af tal er unik og kan derfor benyttes som nøgle til algoritmen. Således kan vi benytte disse tal til at udpege pixels i billedet, som vi indrager i algoritmen. Vi læser hver værdi og hopper ligeledes denne værdi frem i billedet. Dvs. at hvis vi følger den ovenstående nøgle, starter vi med pixel nr. 3. Herefter vi hopper 5 pixelpladser frem, og har nu fat i pixel nr. 8. På denne måde kan vi udpege specifikke pixels i et billede med denne talrække (Se figur 3.5). Når talrækken er gennemgået, kan den gentages uden risiko for at samme pixel rammes igen, i modsætning til alternative måder til at udvælge tilfældige pixels, som vi har overvejet.



Figur 3.5: Denne figur viser hvorledes enkelte pixels udpeges ud fra en tilfældig talrække. Der tages udgangspunkt i talrækken fra det tidligere eksempel.

Har en udefrakomne person ikke denne talrække, skal pågældende gætte sig frem til de specifikke værdier i denne specifikke rækkefølge og længde. Alle variabler (x, y og n) samt de enkelte talværdier er ukendte for den udefrakomne person. Tidskompleksiteten for at finde frem til denne streng af tal, er $O(n^2)$.

Hvis n bliver større bliver dette kun endnu sværere, idet mønstrets længde vil blive forøget (den pseudo-tilfældige talrækkes periode, se 3.8.3). Denne metode har dog det problem at der vil være "huller"af pixels i billedet og dermed vil kapaciteten af et billede være formindsket, da denne metode ikke kan udpege alle pixels i et billede. Disse "huller"i billedet er afhængige af intervallet *[x; y]*, idet det største hul defineres af intervallet. Er intervallet f.eks: *[1; 5]*, vil det største hul være 4 pixels langt.

3.8.2 Metode 2

Som i første metode generes en række af tilfældige tal. Disse tal bliver benyttet til at udpege specifikke pixels i dækmediet.

Denne metode tager udgangspunkt i konceptet om en periode (se 3.8.3). Denne metode vælger to tal fra listen af psuedo-tilfældige tal. Denne liste har længden af frøets periode, for at undgå gentagelser, og dermed undgå at udpege samme pixel flere gange. De to udvalgte tal, forståes som X- og Y-værdier, som korresponderer

med en position i et billede. Denne pixel udpeges, tilføjes til listen, og herefter findes den næste pixel.

Denne metode vælger en tilfældig pixel i billedet, som defineres ud fra tilfældighedsgeneratorens output. Hvis output f.eks. er 30, så vælges pixel nr. 30. Hertil er det relevant at betragte perioden (se 3.8.3) af et frø, idet hvis perioden f.eks. er på 10.000 og der skal bruges 11.000 pixels, så bliver de sidste tusind pixels udpeget to gange, hvilket vil resultere i en fejl.

Da denne metode er begrænset af frøets periode, har den begrænsninger, som vi skal sikre os overfor. Hvis frøet til gengæld genererer en periode som er lang nok, vil det give et godt resultat, i forhold til hvilke pixels, der bliver udpeget i billedet.

3.8.3 Pseudo-tilfældighed

Såkaldt "rigtig tilfældighed"kan kun opnås ved at måle støj, idet støj består af hvad man kalder rigtig tilfældighed. Dvs. at støj ikke matematisk kan forudsiges; altså der ligger ingen algoritme bag genererigen af støj-værdier. F.eks. støj på et fjernsyn, som kommer fra den passive stråling fra universet. [1]

Tilfældige tal som kommer fra en algoritme (tilfældighedsgenerator) er såkaldt pseudo-tilfældige. Dvs. at tallene umiddelbart virker tilfældige. De er altså svære at spotte et mønster i, og dermed forudsige, hvis man på simpel vis observerer talene. Men matematisk set er de fuldstændigt forudsigelige, da denne talrække er dannet af en algoritme, hvorfor det er muligt at forudsige hvad algoritmen vil producere, hvis man kender til startværdien (frøet). [1]

Pseudo-tilfældige talrækker gentager sig selv efter et bestemt antal genererede tal, hvor antallet af disse tal kaldes for en periode. Perioden er bestemt ud fra længden af frøet. Er frøet længere, vil algoritmen kunne danne en længere talrække, før den gentager denne talrække igen. Omvendt, er frøet kort, vil algoritmen kun kunne lave en talrække med en lille periode. En rigtig tilfældig talrække (ikke-pseudo) vil aldrig gentage den samme talrække, og har derfor ingen periode. [1]

Vi kan benytte os af en pseudo-tilfældighedsgenerator, fordi når Alice vil sende en besked til Bob, skal Bob vide hvilke samples der blev brugt til at indkode beskeden i, for at kunne aflæse beskeden. Hvis Bob så har frøet, som Alice brugte til at indkode en besked med, så kan Bob finde frem til præcist de samme samples, da pseudo-tilfældighedsgeneratoren med samme frø, altid vil finde frem til de samme samples.

Vi benytter tilfældighedsgeneratoren i .NET, som er baseret på modificeret version af Donald E. Knuth's "subtractive random number generator algorithm"[39] [65].

3.9 Valg af krypteringsalgoritme

I problemformuleringen 2.4 blev det fundet frem til, at klarteksten skal krypteres til en ciffertekst. Dette gøres for at tilføje et ekstra lag af sikkerhed til det endelige program for sikker kommunikation. Det medfører at hvis brugen af steganografi opdages af en opponent, vil teksten, der bliver udvundet af dækmediet, stadig være krypteret. Det er hermed hensigten at det kun vil være den tiltænkte modtager, som kan oversætte cifferteksten tilbage til klartekst. Grundet at en symmetrisk nøgle er nemmere at dele end en asymmetrisk nøgle, som nævnt i afsnit 2.1.2, vil der i dette afsnit kun blive fokuseret på symmetrisk kryptering, eftersom dette vil tilgodese steganografiens formål om ikke at vække unødig opmærksomhed bedst, hvis nøglen deles på en usikker kanal.

Da kryptering ikke er det primære fokuspunkt for dette projekt, er det blevet valgt at benytte indbyggede biblioteker med krypteringsalgoritmer i .NET framework'et [40], frem for at en krypteringsalgoritme skulle importeres eller skrives fra bunden. Dette vil lette arbejdsbyrden for at implementere kryptering i programmet for sikker kommunikation, og hjælpe til at der kan fokuseres yderligere på andre dele af programmet, som vægter højere end kryptering.

En krypteringsalgoritme, der ville være relevant at implementere i denne sammenhæng, er Rijndael-algoritmen, som senere blev udnævnt til Advanced Encryption Standard (AES) af National Institute of Standards and Technology (NIST). Dette har medført at denne algoritme er blevet, og bliver, brugt af en lang række virksomheder til kryptering af data [46]. Grundet den udbredte brug af algoritmen, må det antages at denne vil være et fornuftigt valg at bruge til kryptering i programmet for sikker kommunikation i dette projekt. Derfor er det også blevet valgt at denne algoritme skal implementeres i programmet. Ud over dette er det blevet valgt ikke at tage hensyn til delingen af nøgler til kryptering og dekryptering i dette projekt. Dette ville have givet anledning til en række problemer i form af etablering af en sikker kanal til deling af nøgler, hvor en opponent ikke skulle have adgang, og er derfor ikke taget med.

3.10 Komprimeringsproblemet

Facebook behandler billedfiler med kompressionsalgoritmer, og dette skal der tages højde for, idet et stego-objekt, der udsættes for kompression, vil miste dets data, og dermed ødelægge beskeden. I dette afsnit tages der udgangspunkt i LSB-metoden, og der overvejes metoder om hvorvidt man kan håndtere kompression med datatab, uden at beskeden tager alvorlig skade.

LSB udnytter, som nævnt i 3.1, de mindst betydelige bit, dvs. de bit som ikke har en direkte synlig effekt. Da disse bit ikke bidrager væsentligt til billedets detajle-

grad, udnytter kompressionsalgoritmer ofte disse bit til at mindske filstørrelsen. Kompressionsalgoritmer udvælger data som anses som værende "unødvendige", og derefter fjerner denne data. Dermed er det de mindst betydelige bit, som står for skud når kompressionsalgoritmer mindsker filer. Da LSB-metoden benytter netop disse bit til at indkode steganografiske beskeder, vil der ske skade på den steganografiske besked, da en betydelig del af de betydningsfulde bit for beskeden vil blive fjernet af kompressionsalgoritmen.

For at komme udenom dette problem kan man gemme sin steganografiske besked i højere bits. Modificeres de højere, mere betydningsfulde bits, vil kompressionsalgoritmen ikke skade den indkodede data. Dette afhænger af hvor kraftig en kompressionsalgoritme som benyttes. Dvs. hvor agressiv algoritmen er i forhold til at udvælge overflødigt data. LSB-metodens grundprincip kan stadig bruges, men der benyttes istedet mere betydningsfulde bits. Da disse 'højere' bits er mere betydnigsfulde for billedet, kan der opstå synlige ændringer i billedet ved at modificere disse bits.

Som sagt kan højere bits benyttes, hvis man ønsker at kompression ikke fjerner eller skader. Dette vil skabe visuelle forskelle i billedet fra det originale billede til det steganografiske indkodede billede. Men hvis selve billedet består af støj (tilfældigt fordelte pixels med tilfældige farveværdier), uden nogen genkendelige former eller mønstre, kan man benytte de højere bits til indkodning. Der vil stadig ske muligt synlige ændringer i billedet. Men da billedet består af støj til at begynde med, kan der ikke umiddelbart ses en forskel. Dog kunne det vække opmærksomhed hos en opponent, hvis billeder af ren støj deles.

Problemet omkring Facebooks komprimering af uploadede billeder er et problem for steganografiens overlevelse. Men dette ligger udenfor scopet af dette projekt, da dette projekts afgrænsning ikke fokuserer på at omgå komprimerings-algoritmer. Brugen af Facebook er dog ikke udelukket, det bliver dog nødvendigt at bruge det på en anden måde. En måde at komme udenom komprimeringsproblemet, er ved at benytte en side, som ikke benytter komprimering af billeder, til at lægge stegoobjektet op på. Så kan der sendes et link til billedet over Facebook og modtageren kan herved stadig tilgå billedet gennem Facebook.

3.11 Kravspecifikation

I dette afsnit vil kravene til programmet blive specificeret. Kravene er lavet på baggrund af de erfaringer der er blevet gjort under teknologianalysen. Her vil kravene blive inddelt i to kategorier, henholdsvis hårde krav, som er krav der skal opfyldes, og bløde krav, som ville være gode at opfylde, men er ikke essentielle for funktionaliteten af programmet.

Kravene er ikke baseret på et specifikt brugersegment. Dermed er kravene ikke ment til at passe på specifikke brugssituationer eller én type bruger. Kravene er derfor ikke udarbejdet med fokus på målgruppen, aktivister. Kravene for programmet vil være de følgende:

Hårde krav til programmet

Steganografi Dette krav udspringer af spørgsmålet om hvodan en besked indlejres i et dækmedie. Dette opnås ved at programmet skal indeholde mindst én steganografisk algoritme til indkodning af en klartekst i et dækmedie. Dette er det mest centrale krav i dette projekt.

Brug af AES-kryptering Dette krav udspringer af spørgsmålet om hvodan beskeden krypteres. Dette opnås ved at programmet skal indeholde et ekstra niveau af sikkerhed, i form af en krypteringsalgoritme. Dette er et krav for at højne sikkerheden af den steganografiske kommunikation. Dette højner brugerens sikkerhed, i tilfælde af at kommunikationen bliver opdaget.

Brug af tabsfri filformater Dette krav udspringer af spørgsmålet om hvordan integriteten af den indlejrede besked sikres. Da mange steganografiske algoritmer til brug på billeder gør brug af de mindst signifikante bits er det væsentligt at vælge et filformat hvor disse ikke går tabt eller bliver ændret uden at det var intentionen.

Modulært systemdesign Programmet skal være opbygget med en modulær stuktur. Dvs. at dele af programmet skal kunne udskiftes, uden at resten af programmet skal omskrives. Altså skal man kunne implementere nye algoritmer, uden at brugerflade eller kontrolstruktur skal ændres. Dette vil også give mulighed for at udskifte eller lave ændringer i brugerfladen, uden at kernen af programmet skal omskrives. Dette opnås ved at benytte flere lag i programmet. En kerne, et translations/kontroller-lag og derpå en brugerflade. Herved kan hver af disse dele udskiftes, uden at være direkte afhængige af hinanden.

Lav tidskompleksitet De implementerede algoritmer skal have så lav en tidskompleksitet som muligt. Det er vigtigt at programmet kan eksekveres hurtigt, for

at brugeren ikke skal vente, og for at programmet kan benyttes på mindre kraftige computere.

Bløde krav til programmet

GUI implementation Vi ønsker at implementere en grafisk brugerflade. Dette er ikke centralt for programmets funktionalitet, men kan dog en fordel for brugere af programmet (en ikke nærmere defineret brugergruppe). Denne brugerflade implementeres ved brug af Windows Forms, en del af .NET-platformen. Grundet kompleksiteten af programmet er det favorabelt at bruge en grafisk brugerflade, i stedet for en konsol-baseret brugerflade.

Komprimering af klartekst For at kunne passe så meget tekst som muligt, ind i et dækmedie, kan det være favorabelt at kunne komprimere klarteksten. Et givet dækmedie har en maksimalt acceptabel grænse for hvad kan indkodes. Hvis klarteksten overskrider denne grænse er det et problem. Dette er dog et blødt krav, da det ikke er en essentiel specifikation.

Kapitel 4

Design af program

Det følgende kapitel vil indeholde vores beslutninger i forhold til design af det endelige program. Herunder vil vi komme ind på hvordan de førnævnte krav til det endelige program specifikt vil kunne blive opfyldt. Dette vil således give indsigt i det designede programs styrker og endvidere hvilke svagheder der måtte være for programmet. Afsnit 4.1 vil omhandle brugergrænsefladen i programmet. Afsnit 4.2 om programflow og design vil beskrive gennemløbet af programmet når det køres. Afsnit 4.3 vil beskrive muligheden for brug af komprimering i programmet og afsnit 4.4 vil beskrive data ekstrahering fra forskellige billedfiltyper, for at kunne benytte steganografi gennem disse.

4.1 Brugergrænseflade

I dette afsnit beskriver vi programmets brugergrænseflade (også kaldet GUI) med fokus på den grafiske brugerflade. Grundet vores krav omkring brugerflade (3.11), er det nødvendigt at dokumentere planen bag grænsefladen, samt dens opbygning.

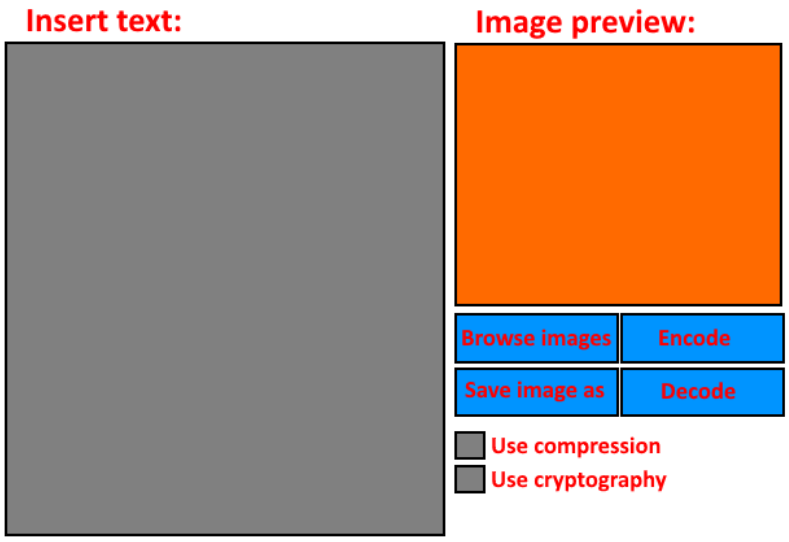
Vi har valgt ikke at tage højde for vores målgruppes ønsker for en brugergrænsefladen. Dette skyldes, at vores målgruppe ikke er defineret nærmere end at det er mennesker, der ønsker at kommunikere hemmeligt, for at de ikke bliver opdaget som værende aktivister. Det er derfor også usandsynligt, at kunne undersøge målgruppens ønsker for brugerflade, da denne jo vil bruge programmet i hemmelighed. Vi vil derfor selv designe en brugerflade. Desuden fokuserer vi brugerfladen til udelukkende at være lavet til computerskærme af almindelige størrelser uden touch-egenskaber.

Visionen med brugerfladen består i at effektivt kommunikere til brugeren, visuelt, samtidig med at brugeren bliver præsenteret for relevante informationer. Det vil sige brugeren ikke ser irrelevant information, som kan nedsætte overblikket og forståelsen af programmet, og dermed hæmme brugen af programmet.

Den grafiske brugerflade er opdelt i to hoveddele. Den ene består af tekst-input, og den anden af billedinformation. Brugeren bliver præsenteret for et stort tekstfelt, som klarteksten kan skrives i. Brugeren bliver her informeret om antallet af karakterer i klarteksten. Dette er relevant, da brugeren også bliver præsenteret for et estimat, for kapaciteten af det indlæste billede, i forhold til hvor meget tekst som kan indkodes i billedet.

Den anden del, billedinformation, viser en thumbnail af billedet, som brugeren har indlæst i programmet, samt information om dette. Herunder findes et kontrolpanel, hvor brugeren kan se de muligheder som han/hun har for at bruge programmet. Funktioner som at indlæse, gemme, indkode og dekode er vist med knapper. Hvis brugeren ønsker ekstra tilføjelser til indkodningen kan disse bestemmes i tjekbokse. F.eks. om hvorvidt der ønskes komprimering eller kryptering som del af indkodningsprocessen.

Se 4.1 for det planlagte brugerinterface.

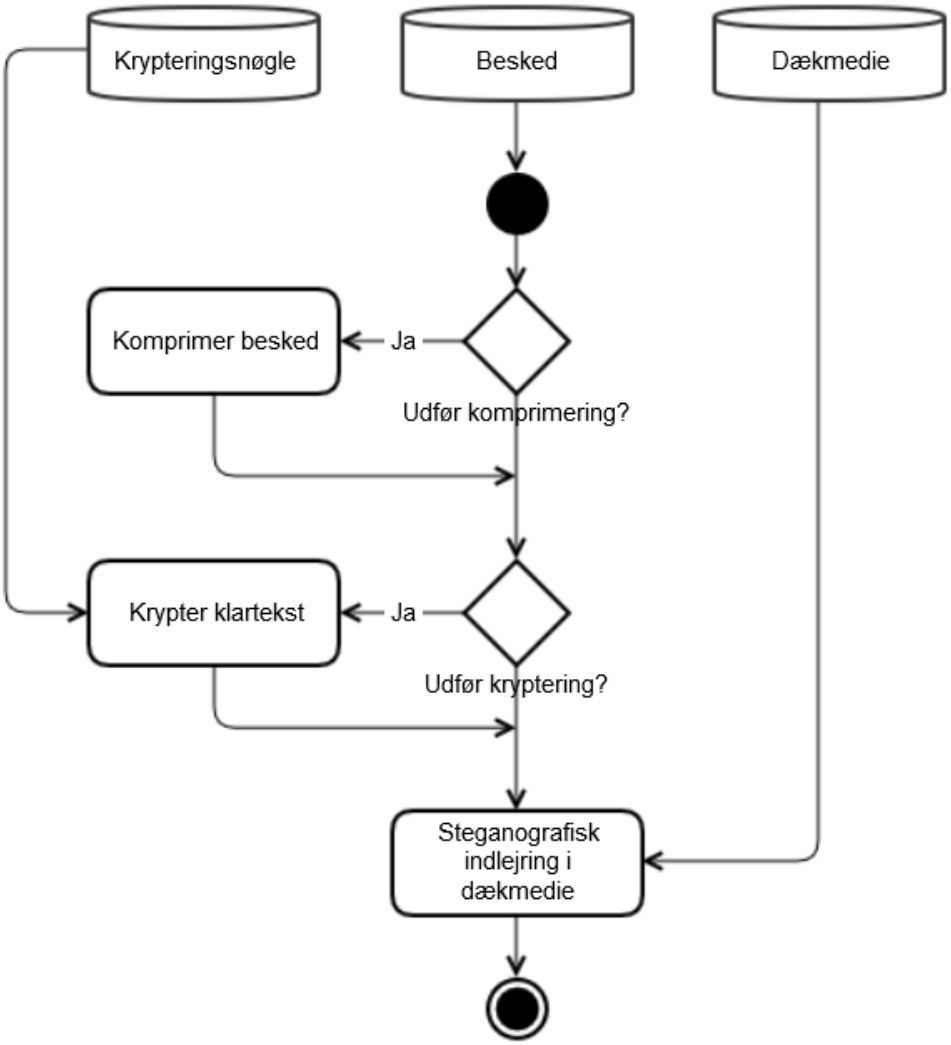


Figur 4.1: Planlagt brugerinterface

4.2 Programflow og design

4.2.1 Flowchart

Når man ved det endelige program trykker på knappen, der indlejrer beskeden i billedet, så udføres følgende programflow, se 4.2.



Figur 4.2: Flowchart over indlejring af besked.

- Den indtastede klartekst bliver loadet ind i programmet.
- Teksten komprimeres.

- Teksten krypteres til ciffertekst, evt. ved brug af nøgle indtastet af bruger.
- Cifferteksten lægges ind i det brugervalgte billede, der vil fungere som dækmedie.
- Den valgte algoritme til steganografi anvendes til indlejring af cifferteksten i billedet.
- Stego-objektet er dannet og kan nu hentes af brugeren.

4.3 Komprimering

Beskederne komprimeres før de indlejres i dækmediet. I dette afsnit undersøges hvordan komprimering af klarteksten kan implementeres.

Til dette formål søges specifikt tabsfri komprimeringsmetoder, der kan bruges på simpel binær data. Der findes en masse open-source biblioteker til dette, eksempelvis [9], dog kan det være meget tidskrævende at forstå hvorledes disse algoritmer fungerer, uden blot at forlade sig på ukendte kilders udtalelser. Desuden er hastighed af programmet og størrelse af filer endnu ikke et problem for projektet, hvilket gør det unødvendigt at sammenligne mange algoritmer blot for at opnå små forbedringer. Der vælges derfor at der blot vil blive anvendt biblioteker, som allerede befinder sig i .NET frameworket, da disse er blevet implementeret af Microsoft [37], hvilket betyder at algoritmerne burde fungere pålideligt og deres funktionsbeskrivelser er troværdige.

4.3.1 ZipFile

System.IO.Compression.ZipFile er en klasse, der indeholder metoder til at gemme filer i .zip formatet. Det specielle ved denne klasse, i modsætning til de andre, er, at denne kan foretage operationer på enkelte filer og foldere i den endelige komprimerede fil. Dette er, i forhold til det endelige produkt, dog et problem, da det blot er en simpel tekststreng, der skal komprimeres. Ved brug af denne klasse skal tekststrengen altså først gemmes som fil, komprimeres og så aflæses igen. Dette er ikke blot sværere at implementere, men forårsager også at strengen bliver gemt på computeren før den er indlejret i billedet, hvilket kan formindske dennes fortrolighed. [44]

4.3.2 DeflateStream

System.IO.Compression.DeflateStream klassen indeholder metoder til at komprimere data, dog kan dette ikke læses af traditionelle zip-programmer. DeflateStream tager

dog blot filstrømme som input, hvilket gør det nødvendigt først at omdanne tekststrengen til dette før den kan komprimeres. Dette kan dog let gøres, eksempelvis ved brug af klassen *System.IO.MemoryStream*. [35]

4.3.3 GZipStream

GZipStream bruger den samme algoritme som DeflateStream til at indpakke og udpakke data, dog tilføjer GZipStream information om filtypen .gz i starten, hvilket gør at den kan åbnes af de traditionelle zip-programmer. Desuden så tjekkes det imens algoritmen kører om der er forekommet datakorruption, og de enkelte dele komprimeres atter hvis nødvendigt. [36]

GZipStream anvendes i det endelige produkt på grund af dets datasikkerhed og altså herved opfylder at integritet af data er sikret hvilket blev fundet væsentligt i problemformuleringen i afsnit 2.4.

4.4 Fremskaffelse af data fra filer

I dette afsnit beskrives hvordan de nødvendige input ekstraheres fra de forskellige filformater, sådan at det bliver muligt, at bruge en steganogafisk algoritme. I afsnit 3.4.3 beskrives den primære steganografiske metode til at indlejre en besked i et dækmedie. Det kræver noget input for at denne algoritme skal kunne fungere. Der findes flere metoder, som kan anvendes, for at få adgang til det data, der befinder sig i en given fil. Det, der skal tages hensyn til, er filformatet, da det data, som er relevant for algoritmen, ikke nødvendigvis er alt det data, som ligger i en billedfil. Derfor vil det være besværligt at behandle et billede direkte, ved hjælp af f.eks. en inputstream. Det kan lade sig gøre, men problemet er, at det eneste, som er interessant for algoritmen, er pixelværdierne. Denne får man adgang til ved eksempelvis inputstreams, dog får man også meget data, som er ubrugeligt. Der skal også skelnes imellem hver billedfils type, da pixelværdierne ikke ligger samme sted i filen for hvert filformat. Dette problem kan løses ved at anvende Windows Graphics Design Interface (GDI+). Dette gør at et billede kan behandles som et objekt med Bitmap klassen, som understøtter følgende filformater[41]:

- BMP
- GIF
- JPEG
- PNG
- TIFF

Ud fra denne klasse kan der instantieres et objekt ud fra et eksisterende billede, som er et af de førnævnte filformater. Klassen indeholder et antal metoder, herunder `GetPixel()` og `SetPixel()`, som kan bruges til at manipulere de enkelte pixles, som er kerneopgaven for den steganografiske algoritme. På denne måde understøtter bitmapklassen, de forskellige anvendte algoritmer[34].

Umiddelbart kan med den grafbaserede algoritme der steganograferes på følgende dataformater:

- Truecolor
- Palette
- JPEG
- Waveform
- μ-law

4.4.1 Truecolor

Truecolor beskriver filtyper, som understøtter et bestemt antal bytes for at repræsentere én pixel. Når et billede understøtter truecolor, vil det sige, at et billede bruger 3 byte pr. pixel. Da én byte kan repræsentere 256 farver, kan man med med tre bytes repræsentere mere end 16 millioner forskellige farver, som følgende beregning viser:

$$256^3 = 16777216 \tag{4.1}$$

Forskellige filtyper understøtter forskellige måder at repræsentere farver på, truecolor er blot en af dem. De filtyper, der understøtter truecolor, er følgende[22]:

- PNG
- JPEG
- BMP
- TIFF

4.4.2 Palette

Palette er en defineret måde for billedfiler at lagre billeddata på, som dog ikke alle billedfiler understøtter. I stedet for at lagre data for hver individuel pixel, laves en palette, som indeholder enhver farve, som billedet gør brug af. Efterfølgende vil hver pixel have et tal, som svarer til en farve på paletten [80]. Den eneste filtype, som understøtter palette, som også understøttes af bitmap klassen, er GIF.

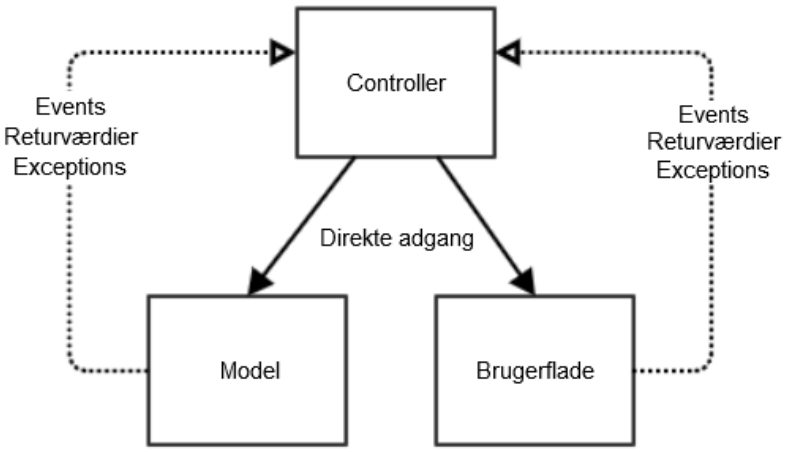
Kapitel 5

Implementation af program

I dette kapitel vil vi dokumentere hvordan vi er kommet fra vores design til det endelige program. Programmet vil følge ModelViewController princippet, hvor fokuset vil være at dele programmet i tre adskillige klasser henholdsvis Modellen, UI'en og Controlleren. I afsnit 5.1 vil vi komme ind på de tre forskellige klasser og hvad deres funktionalitet er, hvad forholdet imellem dem er, og hvordan de kommunikerer med hinanden. I afsnit 5.2 vil selve steganografidelen, som befinder sig i modellen, blive beskrevet, således der er styr på hvad de enkelte metoders input og output er. UI'en, controlleren og Modellen vil i afsnit 5.3, 5.5 og 5.4 blive beskrevet og implementeret.

5.1 Programstruktur

Vi har valgt at dele programmet op i tre hoveddele: *modellen*, *brugergrænsefladen* og en *controller*. Modellen indeholder programmets hovedfunktionalitet, så som at indlejre en besked i et billede. Brugergrænsefladen bestemmer hvorledes brugeren kan styre og få adgang til denne funktionalitet, eksempelvis ved at muliggøre tryk på en knap, der igangsætter indlejring af den indtastede besked i det valgte billede. Controlleren er den klasse, som binder disse to sammen. Controlleren modtager information fra brugergrænsefladen hver gang en knap trykkes, og styrer herefter hvilke metoder, der skal udføres. Eksempelvis modtager controlleren fra brugergrænsefladen besked om at der er blevet trykket på Encode-knappen, hvilket gør at controlleren kalder modellens metode for at indlejre en besked, og herefter kalder brugergrænsefladens metode for at vise et vindue, som viser resultatet af operationen. Se figur 5.1 for hvorledes modellen og brugergrænsefladen holdes adskilt fra hinanden, og blot kan tilgås af controlleren.



Figur 5.1: Overordnet plan for klassestruktur.

På koden i listing 5.1 ses hvorledes denne plan overordnet vil blive implementeret i programmets `main`.

Listing 5.1: Programmets main

```
namespace StegomaticProject
{
    static void Main(string[] args)
    {
        IStegoSystemModel stegoModel = new StegoSystemModelClass();
        IStegoSystemUI stegoUI = new StegoSystemWinForm();
        IStegoSystemControl stegoController = new
           StegoSystemControl(stegoModel, stegoUI);

        stegoUI.Start();
    }
}
```

Fordi det kun er controlleren, der har adgang til de andre klasser, og fordi der ikke foregår direkte kommunikation mellem klasser i hver deres hoveddel, bliver det lettere at holde klasserne uafhængige af hinanden. Dette gør det lettere at undgå at der evt. forekommer fejl i modellen, når der ændres i brugergrænsefladens kode.

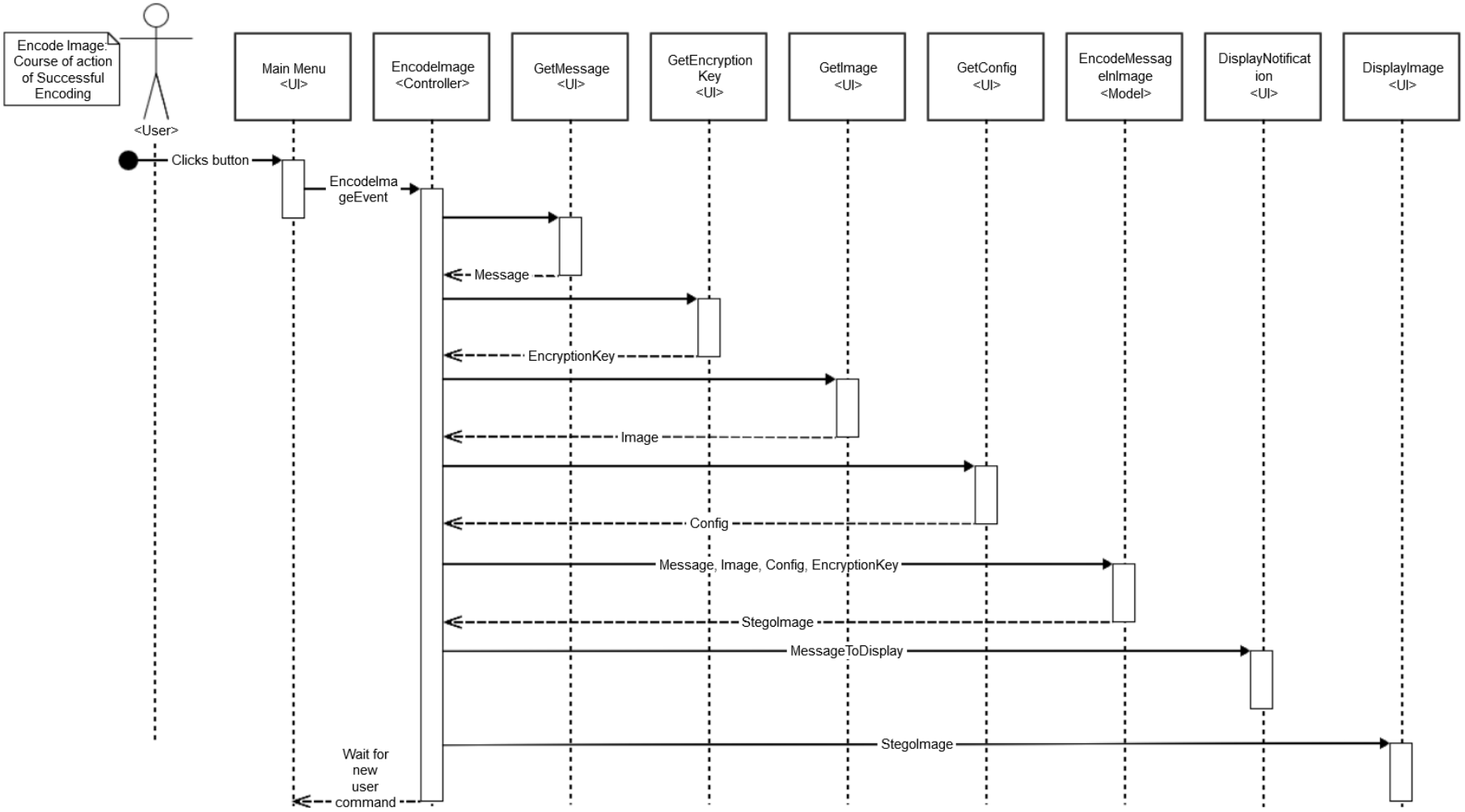
5.1.1 Kommunikation mellem klasser

Brugerfladen, skrevet som en WinForm applikation, modtager information om brugerens betjening af knapperne implicit ved hjælp af events [43]. Herefter skal dette

kommunikeres videre til controlleren, der så udfører de tilsvarende metoder. Det er dog kun controlleren, der har adgang til brugerfladens metoder, ikke omvendt. Så snart et WinForm-event registreres, er det altså nødvendigt at kommunikere ud af klassen, så controlleren kan få besked om denne hændelse. Dette kan gøres ved events, der, så snart disse aktiveres, tjekker om andre metoder er interesserede i de specifikke events. Hvis der er interesserede metoder, såkaldte subscribers, så udføres de interesserede metoder, før metoden, der aktiverede det enkelte event, fortsættes.

Events vil især blive brugt ved kommunikation mellem brugergrænsefladen og controlleren, dog kan de også anvendes til andet, hvis returværdier og exceptionhandling viser sig ikke at være tilstrækkelige.

På sekvensdiagrammet i på figur 5.2 ses hvordan kommunikation mellem klasserne vil komme til at foregå når brugeren betjener Encode-knappen på brugerfladen.



Figur 5.2: Kommunikation mellem klasserne og demonstration af controllerens funktion ved indlejring af en besked i et billede.

5.2 I/O Kontrakter

Modellen indeholder de mest essentielle dele af funktionaliteten og bør derfor være meget vel planlagt og forstået, før den implementeres. I dette afsnit vil vi sætte helt klare krav for modellens vigtigste metoders input og output. Udover større overskuelighed muliggør disse krav større modularitet af programmet. Således ved vi hvilke roller de enkelte metoder spiller i det endelige produkt, selv når disse ikke er blevet implementeret endnu. Det betyder at såfremt disse kontrakter overholdes, så bør de enkelte dele kunne integreres med hinanden uden at dette skaber yderligere fejl. I tabel 5.1 ses de to metoder, der vil blive kaldt af `StegoControlleren` når Encode- og Decode-knapperne betjenes på brugerfladen.

Method name	Input	Output
EncodeMessageInImage	Bitmap, string, string, string	Bitmap
DecodeMessageFromImage	Bitmap, string, string, bool, bool	string

Tabel 5.1: *Modellens* offentlige metoder.

`EncodeMessageInImage()` modtager et dækmedie i bitmapformatet, samt en besked og to nøgler som strenge og returnerer et stego-objekt i bitmap. `DecodeMessageFromImage()` modtager et dækmedie og to nøgler og returnerer en besked.

De ovenstående offentlige metoder kalder de private metoder, som ses i 5.2.

Method name	Input	Output
StringToByteArray	string	byte[]
ByteArrayToString	byte[]	string
Compress	byte[]	byte[]
Decompress	byte[]	byte[]
Encrypt	string, string	string[]
Decrypt	string, string	string
Encode	Bitmap, string, byte[]	Bitmap
Decode	Bitmap, string	byte[]

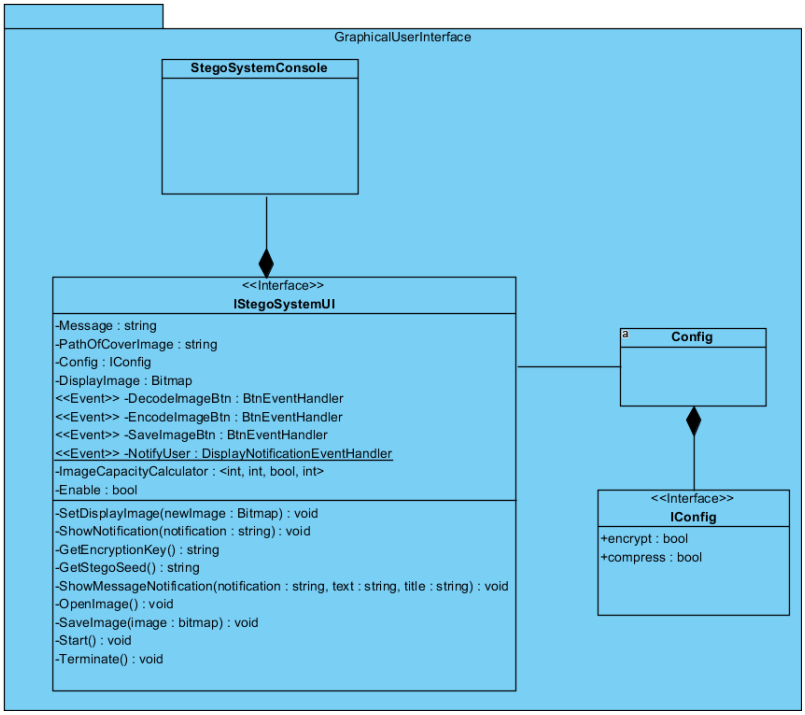
Tabel 5.2: Input og output for de metoder, som vil blive kaldt af modellens metoder `EncodeMessageInImage` og `DecodeMessageFromImage`.

Når `EncodeMessageInImage()` kaldes tager `Encrypt()` klarteksten og en krypteringsnøgle, hvorpå den krypterer klarteksten og returnerer ciffertekst. Cifferteksten konverteres derefter til dataformatet byte[] og sendes til `Compress()`, hvor den bliver komprimeret og returneret. Dernæst sendes den komprimerede ciffertekst og dækmediet til `Encode()`, hvor cifferteksten bliver indkodet i billedet ved hjælp af en steganografialgoritme og stego-objektet returneret.

For at afkode beskeden fra stego-objektet kaldes `DecodeMessageFromImage()`, der sender stego-objektet og den rette stego-nøgle til Decode(), der ekstraherer cifferteksten fra dækmediet. Denne ciffertekst sendes videre til `Decompress()` hvor det bliver dekomprimeret. Dernæst bliver cifferteksten i `Decrypt()` dekrypteret med samme nøgle, som blev brugt under krypteringen. Til sidst konverteres beskeden i `ByteArrayToString()` tilbage til dataformatet string, dette gøres internt i når cifferteksten dekrypteres i `Decrypt()`.

5.3 Brugergrænseflade

Brugergrænsefladen er blevet opbygget som illustreret ved klassediagrammet på figur 5.3



Figur 5.3: Klassediagram for brugergrænsefladen og hvilke klasser og interfaces der gøres brug af.

Herunder ses at brugerfladen implementerer et interface, hvilket er den måde controlleren tilgår klassen på.

Brugerfladen implementeres med Windows Forms. Winforms implementeres vha. klasser, der er opdelt i programmørens del og en autogenereret del. I den genererede del befinder sig kode, som programmøren generelt ikke har brug for at vide om eller ændre i. Herunder f.eks. at en winform knaps metode i programmørens del, der skal udføres når knappen trykkes, subscriber til det event, som bliver aktiveret når knappen trykkes af en bruger.

Specielt ved vores måde at implementere Winforms på er, at vi har valgt at sende mange af de opfangede events ud til klassen `stegoSystemWinForm` i stedet for at håndtere disse direkte i programmørens del af den opdelte klasse. Dette gøres for overskuelighedens skyld, især da WinForms har utrolig mange private events og metoder.

Ved at have brugerfladens funktionalitet udenfor selve formen undgår vi at miste overblikket over hvilke af disse er relevante, og at vi tvinger os selv til at overveje hvilke af formens metoder vi har brug for er synlige. `StegoSystemWinForm`-klassen sender herefter selv de fleste af disse events længere ud til `StegoSystemController`-klassen. Dette gøres, så det alene er controlleren, der styrer hvad tryk på de enkelte knapper gør.

Fremadrettet giver dette os mulighed for at bytte om på og ændre brugerfladens knappers funktioner uden at skulle genstrukturere koden. En simpel ombytning af to events kan eksempelvis få Encode-knappen til at åbne et billede i stedet for at indkode en besked i billedet. Se koden under Listing 5.2 for hvorledes vi modtager formens events i `StegoSystemWinForm`. Mange af disse sendes herefter videre til controlleren, og nogle håndteres direkte i `StegoSystemWinForm`.

Listing 5.2: Events opfanges af StegoSystemWinForm

```
private void SubscribeToEvents()
{
    _mainMenu.NotifyUser += new
        DisplayNotificationEventHandler(this.ShowNotification);
    _mainMenu.EncodeBtnClick += new
        BtnEventHandler(this.EncodeBtnClick);
    _mainMenu.DecodeBtnClick += new
        BtnEventHandler(this.DecodeBtnClick);
    _mainMenu.OpenImageBtnClick += new
        BtnEventHandler(this.OpenImageBtnClick);
    _mainMenu.CompressionCheckToggle += new
        BtnEventHandler(this.ForceUpdateImageCapacity);
}
```

En undtagelse på dette er config-klassen, der viser information om brugeren vil bruge kryptering og komprimering ved indkodning og afkodning af beskeder. Dette gør vi, da vi gerne vil have at hver eneste mainMenu-form, der instantieres, besidder sin egen config. Udover dette tager vi os selvfølgelig ikke af de events, som vi ikke explicit vil styre, som fx. at tryk på det røde kryds bør lukke programmet.

Kode, som er helt specifikt til brugerfladens funktionalitet, håndteres stadig i formen selv. Herunder er det bedste eksempel statuslinjen i bunden af brugergrænsefladen, der viser hvor meget kapacitet, der er tilovers i billedet, se koden i listing 5.3.

Listing 5.3: Implementation af billedkapacitetsbarren

```
private bool _previouslyOverLimit = false;

private void CapacityBarUpdateValidImage()
{
    // Updates the capacity bar character count and then tries to
       update the capacity bar visually.
    // If this fails then an exception is thrown and the bar's
       visual value is set to maximum, though only
    // if it has not previously been done. This causes the input
       of too many characters to only throw one
    // exception for each time it crosses the maximum threshhold.

    double input = txtbox_input.Text.Length;
    double capacity = Convert.ToDouble(label_capacity.Text);
    progressBar1.Visible = true;
    label_char.Text = "Characters: " + input + " / " + capacity;

    try
    {
        progressBar1.Value =
            Convert.ToInt32((input / capacity) * 100);
        _previouslyOverLimit = false;
    }
    catch (ArgumentOutOfRangeException)
    {
        if (!_previouslyOverLimit)
        {
            progressBar1.Value = progressBar1.Maximum;
            _previouslyOverLimit = true;
            throw;
        }
    }
}
```

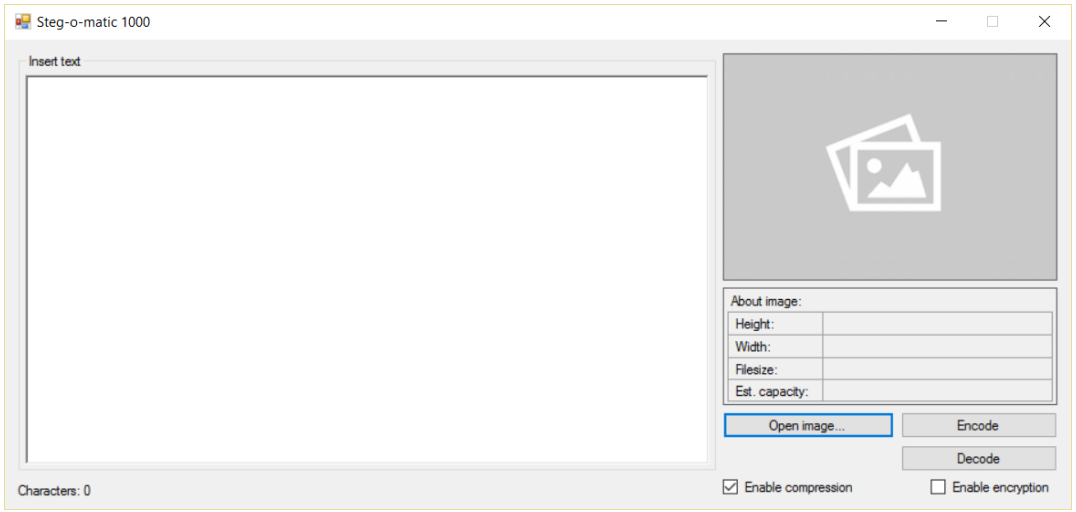
`CapacityBarUpdateValidImage()` bliver kaldt hver gang brugeren har redigeret i tekstboksen og der ligger et billede i displayet. Herunder udregnes antal procent som tekst i tekstboxen udgør af den samlede kapacitet af billedet, og dette indsættes som værdi i selve kapacitets-statuslinjen. Hvis denne værdi ikke er tilladt (over 100), og dette ikke var tilfældet ved sidste redigering, så kastes en `ArgumentOutOfRangeException` og kapacitets-statuslinjen sættes til maksimum.

5.3.1 Multithreading

WinForms har dog den større ulempe angående performance, at brugerinterfacet kun kører i en enkelt tråd. Dette betyder at hvis programmet udfører beregninger, som ikke inddrager brugerfladen, så kan brugerfladen ikke svare på brugerinput, og det kan se ud som om programmet er gået i stå. Vi har løst dette ved at implementere en backgroundworker, som vi kan sætte i gang med at lave de beregningsmæssige tunge kommandoer for indkodning og afkodning af en besked af billedet. Denne backgroundworker giver besked om at den er blevet færdig gennem et event, som vi så håndterer. Hvorledes denne backgroundworker er blevet implementeret vises i afsnittet om `StegoSystemControlleren`, se afsnit 5.5.2.

5.3.2 Den endelige brugergrænseflade

Se figur 5.4 for den endelige brugergrænseflade.

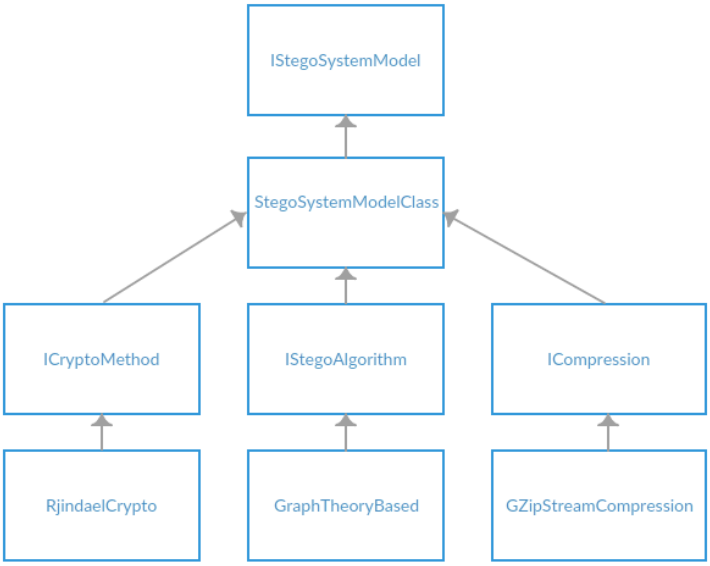


Figur 5.4: Brugergrænsefladen i det endelige program.

I forhold til det planlagte er der nogle få ændringer. Den vigtigste af disse er at vi har fjernet "Save image"knappen, da man kun har brug for at gemme billedet når man er færdig med at indkode sin besked. Derudover er komprimeringen aktiveret når programmet starter, da den gør at pladsen i billedet er udnyttet bedre og ikke har nogle ulemper for brugeren. Herudover har vi senere i implementationen også tilføjet en `OpenImage()` funktion, da det var nemmere ikke at håndterede dette inde i selve WinFormen. Desuden har vi tilføjet en bool property `Enable`, der låser og låser op for at brugeren kan interagere med brugerfladen.

5.4 Model

Modelklassen er ud fra vores overvejelser i designprocessen blevet opdelt som illustreret på figur 5.5. Hvert af disse led i modelklassen implementerer således et interface, som klasserne vil blive tilgået igennem. Komprimeringsklassen vil ikke blive beskrevet, da det blot er GZipStream, en standard .NET komprimeringsmetode, der er blevet anvendt.

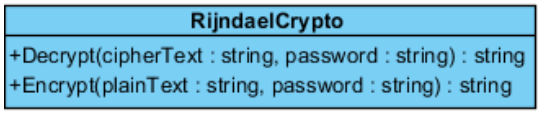


Figur 5.5: Oversigt over sammenhæng mellem klasserne for modellen.

5.4.1 Kryptografi

I programmet er et af kravene, at der skal benyttes kryptografi for at skabe et ekstra lag af sikkerhed. Til dette formål skal der laves en klasse der bliver tildelt navnet `RijndaelCrypto`.

Metoder Denne klasse kommer til at indeholde to metoder; en til kryptering af klartekst og en til dekryptering af en ciffertekst. De to metoder bliver henholdsvis kaldet `Encrypt` og `Decrypt`. Begge disse metoder kommer til at tage to strenge som parameter, hvor den ene af disse er `password` og den anden er enten klarteksten eller cifferteksten, afhængig af metoden der kaldes. Et klassediagram for denne klasse kan ses på figur 5.6.



Figur 5.6: Klassediagram for kryptografiklassen i programmet

5.4.2 Steganografi

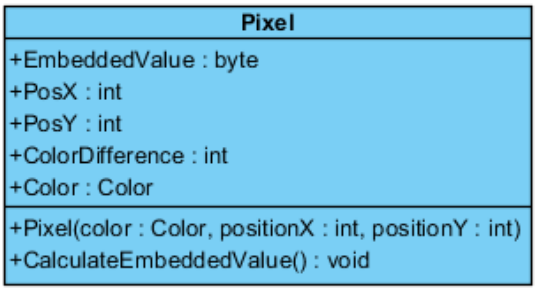
Konstruktionen af en graf ved brug af den grafbaserede metode blev gennemgået i afsnit 3.6. I de følgende afsnit vil vi beskrive hvilke klasser, der indgår i konstruktionen af en graf og udførelsen af steganografi. Disse klasser er: `Pixel`-klassen, som indeholder information om hver pixel i dækmediet, `Edge`-klassen, som indeholder information om kanten mellem to knuder, og `Vertex`-klassen, som indeholder information om pixels der indgår i knuden, samt kanterne mellem de forskellige knuder og `GraphTheoryBased`-klassen, der konstruerer grafen og hjælper til at udføre steganografi på dækmediet.

Pixelklasse

Som nævnt i introduktionen i afsnittet skal denne klasse indeholde information om hver enkelt pixel i dækmediet. Den information, som hver pixel indeholder, kan beskrives med de følgende variabler og metoder:

Variabler Hver instans af denne klasse kommer til at indeholde sin farve under variablen `Color`, som er et Color struct, og sin nuværende indlejrede værdi i variablen `Embeddedvalue`, der repræsenteres af en byte. Derudover indeholder hver pixel også information om dens position gennem variablerne `PosX` og `PosY` der er integers. Endvidere er der også en variabel `ColorDifference`, som indeholder en værdi for farveforskellen mellem to pixels.

Metoder Pixelklassen har metoden `Pixel`, som læser al nødvendig information fra en given pixel i et billede over i dets tilsvarende variabler i klassen. Dertil er også en metode `CalculateEmbeddedValue()`, der beregner den givne pixels indlejrede værdi ud fra farven fundet i `Color` variablen. Klassediagrammet for denne klasse kan ses på figur 5.7.



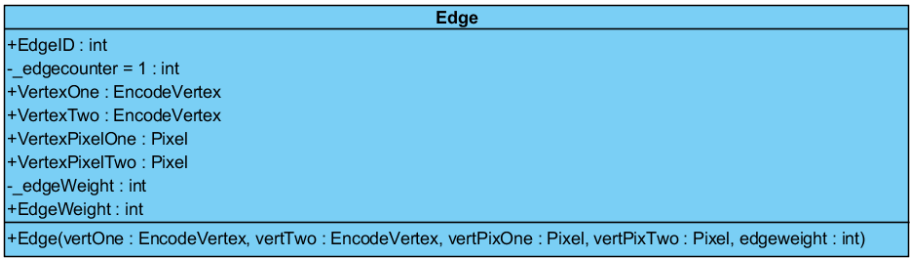
Figur 5.7: Klassediagram for pixel klassen

Kantklasse

Denne klasse skal indeholde information om den enkelte kant. Ligesom det var tilfældet med Pixel klassen, bliver der for hver kant også lavet en ny instans af denne klasse. Informationen i denne klasse kan beskrives med de følgende variabler og metoder:

Variabler Hver kant har et unikt ID som bliver holdt i variablen `EdgeID` som en integer, dette ID bliver bestemt ud fra `_edgecounter` som tæller op for hver ny instans af klassen der laves. Hver instans af kantklassen indeholder også to knuder, da det er disse der findes en kant mellem. De to knuder findes i henholdsvis `VertexOne` og `Vertextwo` der begge er af typen vertex og er altså hver især en instans af knudeklassen. Klassen indeholder også to pixels `VertexPixelOne` og `VertexPixelTwo`, der repræsenterer en pixel fra hver knude og er af typen pixel. Det er disse pixels hvor en ombytning kan ske, da det er disse som kanten er mellem. Endvidere har hver kant også en vægt `EdgeWeight`, som er en byte der beregnes ud fra `_edgeWeight` der sætter et tal på, hvor god ombytningen mellem de to pixels vil være.

Metoder Kantklassen indeholder constructoren `Edge()`, der laver en ny kant mellem to pixels og udfylder al nødvendig information i variablerne til klassen. Klassediagrammet for denne klasse kan ses på figur 5.8 nedenfor.



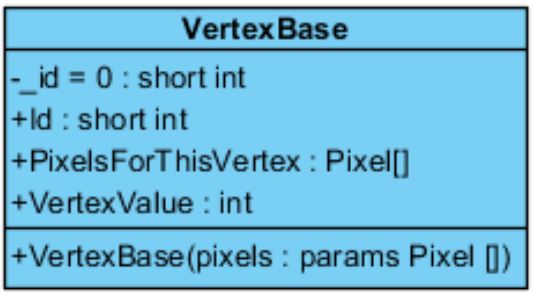
Figur 5.8: Klassediagram for kant klassen

Knudeklasse

Denne klasse skal indeholde information om den enkelte knude. Ligesom det var tilfældet med både pixel klassen og kant klassen, bliver der for hver knude lavet en ny instans af denne klasse. Informationen i denne klasse består af de følgende variabler og metoder:

Variabler En id variabel, `id`, som bliver givet for hver instans af klassen ud fra `_id`, sådan at forskellige knuder kan identificeres. Derudover indeholder klassen også pixels under `PixelsForThisVertex`, der er et array af pixels, som knuden består af. Til slut indeholder variablen `VertexValue`, der med en integer repræsenterer knudens indlejrede værdi, som kan ændres ved at bytte pixels med andre knuder.

Metoder Knudeklassen indeholder metoden `VertexBase`, som udfylder den nødvendige information, der skal til for at lave en ny instans af knudeklassen, ud fra et variabelt antal pixels, givet som parameter til metoden. Klassediagrammet for denne klasse kan ses på figur 5.9 nedenfor.



Figur 5.9: Klassediagram for knude klassen

Steganografiklasse

Denne klasse indeholder information om de tre tidligere nævnte klasser, samt kryptering og komprimering, for at kunne udfører steganografi på et givet dækmedie i form af et billede. Informationen i denne klasse kan beskrives med de følgende variabler og metoder:

Konstanter Denne klasse indeholder en række konstanter som bruges i algoritmen. Disse er *SamplesVertexRatio* som definerer antal samples pr. knude. I dette tilfælde er det hvor mange pixels der er knyttet til en knude. *Modulo* definerer modulo-værdien som algoritmen bruger. *MaxEdgeWeight* definerer et loft for den maksimalt tilladte vægt for en kant. Tilsidst definerer *PixelsPerByte* hvor mange pixels vi bruger for at repræsentere en byte.

Metoder `Encode()` står for at indlejre beskeden i dækmediet og returnerer et bitmap, hvor beskeden er indlejret når metoden er gennemført. `Decode()` står omvendt for at få beskeden ud af stego-objektet, som blev lavet af `Encode()`. `Decode()` metoden returnerer et array af bytes bestående af beskeden der blev taget ud af stego-objektet. Metoden `CalculateEdgeWeight()` bruges til at beregne vægten på en given kant.

Metoden `ShortenAndParsePassphraseToInt32()` står for at tage det kodeord, som brugeren har indtastet og laver dette om til en int, der efterfølgende kommer til at fungere som frøet *(seed)* til at udvælge tilfældige pixels. De næste to metoder er metoder til at udvælge tilfældige pixels med et givet frø. De to metoder hedder henholdsvis `GetRandomPixelsAddtoList1()` og `GetRandomPixelsAddtoList2()`. Metoden `CombineArrays()` står for at sætte arrays sammen til et samlet array med de værdier som de individuelle arrays holdte på. Metoden `ByteArrayToValues()` står for at læse et array af bytes over til værdier mellem $0 - 3$ for hvert par af bits i en byte.

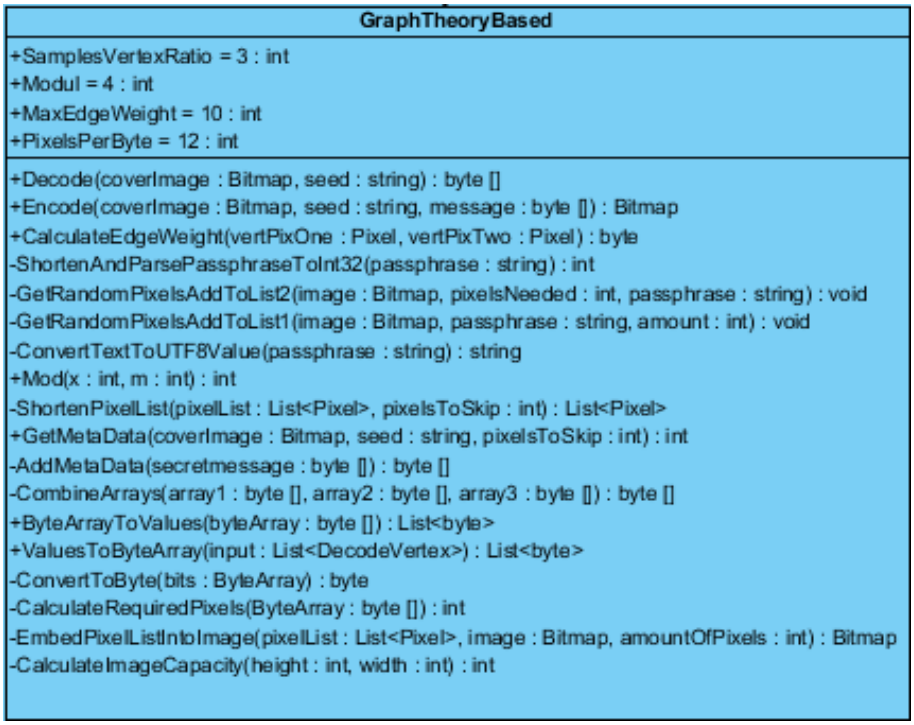
Metoden `ValuesToByteArray()` gør det modsatte af `ByteArrayToValues()` og tager en liste af værdier læst fra stegoobjektet og konverterer værdierne til par af bits for at kunne lave et array af bytes, der består af beskeden der tages ud af stego-objektet. Metoden `ConvertToByte` tager et array af otte bits og omsætter dette til en byte, denne metode bruges når `ValuesToByteArray` bruges.

Metoden `CalculateRequiredPixels()` bestemmer, ud fra længden af beskeden der skal indlejres, hvor mange pixels der skal bruges til dette. Metoden `EmbeddedPixelListIntoImage()` står for at overskrive de modificerede og byttede pixels ind i det originale dækmedie. Metoden `CalculateImageCapacity()` står for at beregne hvor stor en besked et givet dækmedie kan holde. Klassediagrammet for denne klasse kan ses på figur 5.10.

Metoden `AddMetaData()` tilføjer information til den hemmelige besked, efter kryptering og komprimering. Det der tilføjes, er hvor mange bytes den hemmelige besked består af, og en stopklods, som adskiller metadata delen fra den hemmelige besked.

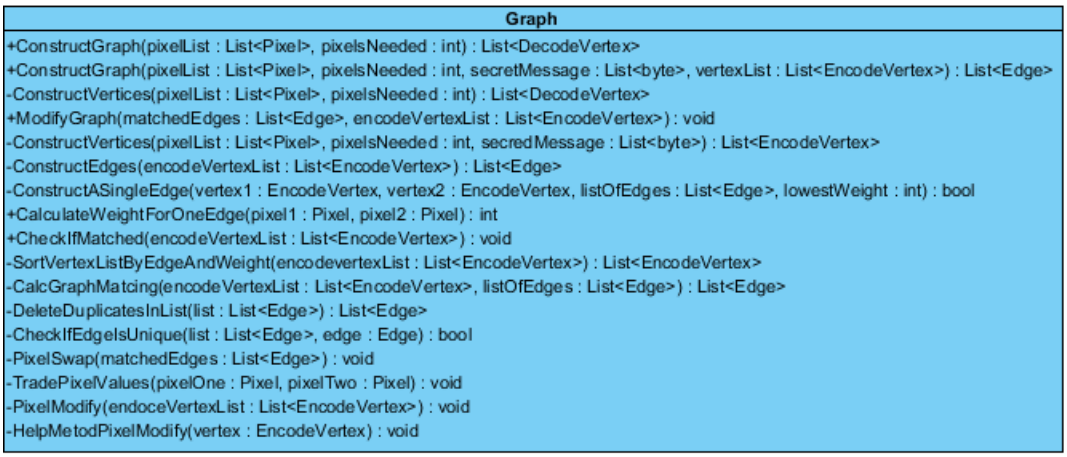
Metoden `GetMetaData()` er metoden der læser metadata fra et stego-objekt når det skal afkodes gennem programmet. Dette består i at de ti første tegn afkodes, for at finde ud af, hvor meget data der er indkodet i billedet, sådan vi efterfølgende kan aflæse en korrekt mængde af pixels. Efterfølgende opdeles strengen, så metadata adskilles fra den hemmelige besked.

Metoden `Mod()` er vores egen definerede modulo-metode. Den er implementeret da vi mødte problemer ved måden hvorpå C#'s modulo-operator håndterede visse tal. Dette gjorde at algoritmen ikke fungerede som den skulle, og derfor definerede vi en modulo-metode som opførte sig som forventet.



Figur 5.10: Klassediagram for GraphTheoryBased klassen

Klassen `Graph` indgår også under programmets model, det er i denne klasse en graf bliver konstrueret og modificeret ud fra det givne dækmedie og den givne besked. Klassediagrammet for denne klasse med dets metoder kan ses på figur 5.11 nedenfor:



Figur 5.11: Klassediagram for Graph klassen

I `Graph` klassen indgår en række metoder der tilsammen konstruerer en graf ud fra dækmediet og indlejrer den brugerdefinerede besked i dækmediet, som ender med at være stego-objektet. En gennemgang af nogle udvalgte metoder fra denne klasse bliver gennemgået i afsnittet kan ses i afsnittet nedenfor:

Præsentation af udvalgte metoder fra Graph-klassen

I dette afsnit præsenterer vi to centrale metoder fra `Graph` klassen, henholdsvis `ConstructEdges()` og `CalcGraphMatching()`.

Listing 5.4: Metoden `ConstructEdges()`, konstruerer alle de edges, som der måtte være for et givent dækmedie. Dette gøres ud fra de knuder, som bliver konstrueret af `ConstructVertices()`.

```
private List<Edge> ConstructEdges(List<EncodeVertex>
    encodeVertexList)
{
    List<Edge> listOfEdges = new List<Edge>();

    foreach (EncodeVertex item1 in encodeVertexList)
    {
        int lowestWeight = GraphTheoryBased.MaxEdgeWeight;
        int edgeWeight;
        int amountOfEdges = 0;

        foreach (EncodeVertex item2 in encodeVertexList)
        {
            if (item1.Active == true && item2.Active == true)
            {
```

```
                bool b = ConstructASingleEdge(item1, item2,
                    listOfEdges, out edgeWeight);
                if (b == true)
                {
                    if (edgeWeight <= lowestWeight)
                    {
                        lowestWeight = edgeWeight;
                    }
                    amountOfEdges++;
                }
            }
        }
        item1.LowestEdgeWeight = lowestWeight;
        item1.NumberOfEdges = amountOfEdges;
        item1.Active = false;
    }
    return listOfEdges;
}
```

Metoden beskrevet i 5.4 tager én formel parameter, som er en liste af knuder (*encodeVertexList*) og returnerer en liste af kanter (*listOfEdges*).

I metoden skal vi sammenligne en knude i listen *encodeVertexList* med en anden knude i samme liste. For at gøre dette laver vi et dobbelt foreach loop. Det efterfølgende *if*-statement vil kun være sandt, hvis begge knuder er aktive, hvilket de kun er, hvis der skal oprettes en kant. Derefter kaldes en hjælpe metode `ConstructASingleEdge()`, som kun opretter én kant mellem to knuder. Hvis der bliver oprettet en kant, returnerer `ConstructASingleEdge()` true, hvorefter der udføres to ting:

- Der tjekkes for om den kant der lige er blevet lavet *edgeWeight* er mindre end *lowestWeight*, som er den lavest vægtede kant, som en knude har, indtil videre. Hvis dette er sandt, bliver den hidtil laveste vægt for en knude, sat til *edgeWeight*. Dette har det formål, at efter vi har konstrueret alle kanter for en knude, så ved vi hvilken vægt, den bedste kant har, for den pågældende knude.
- Hvis der blev oprettet en kant, bliver antallet af kanter for den pågældende knude (item1), forøget med en.

Til slut tildeles den laveste vægt, samt hvor mange kanter knuden har og knuden (item1) sættes til at være inaktiv, da denne er blevet sammenlignet med alle andre knuder (item2). Desuden har det den effekt, at den ikke vil blive udvalgt igen, da den nu er inaktiv. Når alle kanter er blevet oprettet, returneres listen af kanter.

Metoden `CalcGraphMatching()` (5.5) finder den bedste kant for en given knude, og vælger denne knude, som senere skal bruges til at foretage en ombytning. Denne metode er derfor den centrale udvælgelse i den grafbaserede algoritme. Metoden tager en liste af knuder, samt en liste af kanter. Der undersøges og vælges den bedst mulige kant per knude. Herefter tilføjes den udvalgte kant til en liste af udvalgte kanter, som returneres af metoden.

Listing 5.5: `CalcGraphMatching()` en af de centrale metoder i algoritmen. Den udregner matches mellem knuder i grafen.

```
private List<Edge> CalcGraphMatching(List<EncodeVertex>
    encodeVertexList, List<Edge> listOfEdges)
{
    encodeVertexList =
        SortVertexListByEdgeAndWeight(encodeVertexList);

    List<Edge> tempMatched = new List<Edge>();

    foreach (EncodeVertex vert in encodeVertexList)
    {
        if (vert.Active == true && vert.NumberOfEdges > 0)
        {
            List<Edge> InternalEdgeList = new List<Edge>();
            foreach (Edge edge in listOfEdges)
            {
                if (edge.VertexOne == vert || edge.VertexTwo ==
                    vert)
                {
                    InternalEdgeList.Add(edge);
                }
            }

            List<Edge> SortedInternalList =
                InternalEdgeList.OrderBy(o =>
                o.EdgeWeight).ToList();

            if (SortedInternalList.FirstOrDefault() == null)
            {
                // In case of this, skip
            }
            else
            {
```

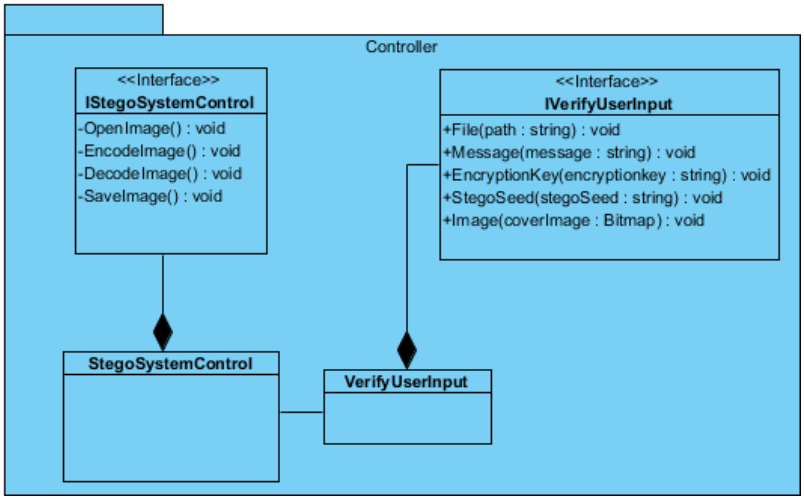
```
                Edge M = SortedInternalList.First();
                tempMatched.Add(M);
            }
        }
    }
    List<Edge> matchedEdges = tempMatched;

    return matchedEdges;
}
```

Idet `CalcGraphMatching()` returnerer en liste af de udvalgte kanter, har vi nu en komplet liste af kanter som kan håndteres af `PixelSwap`-metoden, hvor de to knuder, (som er relateret til én given kant) gennemgår en ombytning, således at deres værdier matcher med værdien som ønskes indkodet.

I den returnerede liste vil der forekomme gentagelser, i form af at samme kant kan fremgå flere gange. Men det gør ikke at den bliver byttet flere gange, da vi deaktiverer en knude når den er modificeret via ombytning, og der tjekkes om begge knuder er aktive for hver kant. Denne metode er derfor det afgørende element i algoritmen, der afgør hvilke specifikke knuder skal håndteres videre i algoritmen.

5.5 Controller

I dette afsnit vil vores controller blive gennemgået. Det er denne, der tager imod events fra brugergrænsefladen og dermed står for at styre programmet ved brug af brugerens input. Når brugeren interagerer med brugerfladen, bliver en metode kaldt, der svarer til hvad brugeren forespurgte. Controlleren består af de klasser, som kan ses på figur 5.12:



Figur 5.12: Klassediagram for controlleren.

5.5.1 Indkodning af besked

Controllerklassens essentielle metoder er bundet til brugergrænsefladens events ved subscribing, se listing 5.6.

Listing 5.6: Controllerens metoder subscribes til brugergrænsefladens events.

```
private void SubscribeToEvents()
{
    _stegoUI.NotifyUser += new
       DisplayNotificationEventHandler(this.ShowNotification);
    _stegoUI.EncodeBtn += new BtnEventHandler(this.EncodeImage);
    _stegoUI.DecodeBtn += new BtnEventHandler(this.DecodeImage);
    _stegoUI.OpenImageBtn += new BtnEventHandler(this.OpenImage);

    SubscribeBackgroundWorkerEvents();
}
```

Det er på denne måde at `EncodeImage()` i controlleren bliver kaldt, hver gang et `EncodeBtn`-event bliver rejst i brugergrænsefladeklassen. På figur 5.7 ses kontrollerens `EncodeImage()`-metode, der kaldes når der skal gemmes en besked i et valgt billede. Det første, der sker, når `EncodeImage()` kaldes gennem kontrolleren, er at der oprettes de forskellige referencer til de interfaces, der skal bruges i UI delen af programmet. Efterfølgende verificeres det om brugeren benytter et brugbart billede, altså at det indsatte billede er en billedfil. Hvis dette er tilfældet kan programmet fortsætte med at køre `EncodeImage()`.

Det næste trin er at tjekke om der benyttes kryptering og komprimering. Hvis ja, køres disse processer i encoding-processen, hvis ikke, undlades de. Efter at dette er fastlagt bedes brugeren om et kodeord som omdannes til et seed, der bestemmer hvordan beskeden indkodes i dækmediet. Det er det samme kodeord der bruges til at få den indkodede besked ud af stego-objektet igen, da der genereres det samme seed fra det samme kodeordet. Det næste der sker i denne proces, er at beskeden og kodeordet fra brugeren verificeres, før den egentlige indkodning kan forløbe. Hvis verificeringen ikke lykkes, kastes en undtagelse til brugesren og processen stoppes.

Hvis alle kriterier imødekommes, begynder indkodningen af beskeden i dækmediet ud fra det givne kodeord, repræsenteret som et seed, efter at alle argumenter er sat ind i en tupel kaldet `args`. Herefter kaldes metoden `RunWorkerAsync()` med den konstruerede tuple som parameter, der ender ud med at konstruere det endelige stego-objekt, hvori beskeden er indkodet ved hjælp af den grafteoretiske metode.

Listing 5.7: Metoden, der styrer indkodningen af en besked i et billede

```
public void EncodeImage(BtnEvent e)
{
    try
    {
        IConfig config = _stegoUI.Config;
        string message = _stegoUI.Message;
        Bitmap coverImage = _stegoUI.DisplayImage;
        string encryptionKey = string.Empty;

        _verifyUserInput.Image(coverImage);
        if (config.Encrypt)
        {
            encryptionKey = _stegoUI.GetEncryptionKey();
            encryptionKey =
                _verifyUserInput.EncryptionKey(encryptionKey);
```

```
        }
        string stegoSeed = _stegoUI.GetStegoSeed();

        message = _verifyUserInput.Message(message);
        stegoSeed = _verifyUserInput.StegoSeed(stegoSeed);

        var args = Tuple.Create<Bitmap, string, string, string,
            bool, bool>(coverImage, message, encryptionKey,
            stegoSeed, config.Encrypt, config.Compress);

        _backgroundWorkerProgressBar.Show();
        // Show that we're working on it!

        _stegoUI.Enable = false;
        // Disable the main-window, so the user click on anything
            they're not supposed to.

        _encodeWorker.RunWorkerAsync(args);
        // When worker is done and event will fire and
            ThreadedEncodeComplete() will be executed, which
        // will start a save-dialog when the encoding-process is
            completed.
    }
    catch (NotifyUserException exception)
    {
        ShowNotification(new DisplayNotificationEvent(exception));
    }
    catch (AbortActionException)
    {
    }
}
```

5.5.2 Backgroundworker encoding

For at undgå at programmet ser ud til at være gået i stå når det arbejder på de tidskrævende beregninger i modelklassen, implementerer vi en *backgroundworker*. BackgroundWorker har til formål at få brugerinterfacet til at køre i en anden tråd end `EncodeImage()`. Dette gøres, da brugerinterfacet ellers ikke kan svare på operativsystemets forespørgsler, mens `EncodeImage()` kører. Eftersom operativsystemet ikke kan komme i kontakt med programmets interface, fejltolkes det til at programmet er frosset og der sendes en fejlmeddelelse til brugeren om dette, på

trods af at programmet stadig kører. Se listing 5.8 for hvordan *backgroundworkeren* er blevet sat til at interagerer med de andre dele af controlleren.

Listing 5.8: Backgroundworker subscribing

```
private void SubscribeBackgroundWorkerEvents()
{
    //Backgroundworker for encoding
    _encodeWorker.DoWork += new DoWorkEventHandler(ThreadedEncode);
    _encodeWorker.RunWorkerCompleted += new
        RunWorkerCompletedEventHandler(ThreadedEncodeComplete);

    //Backgroundworker for decoding
    _decodeWorker.DoWork += new DoWorkEventHandler(ThreadedDecode);
    _decodeWorker.RunWorkerCompleted += new
        RunWorkerCompletedEventHandler(ThreadedDecodeComplete);
}
```

`ThreadedEncode` køres i en ny tråd så snart backgroundworkerens event `_encodeWorker.DoWork` rejses, da `ThreadedEncode()` subscriber til denne gennem en `DoWorkEventhandler`. På samme måde subscriber `ThreadedEncodeComplete()` gennem `RunWorkerCompletedEventHandler()` på backgroundworkerens event `ThreadedEncodeComplete()`. Dette event rejses når `ThreadedEncode()` er færdig med at køre og forårsager at `ThreadedEncodeComplete()` kaldes i den nye tråd.

`ThreadedEncode()`, der igangsættes når `EncodeImage()` i controlleren kalder `_encodeWorker.RunWorkerAsync(args)`, ses i listing 5.9.

Listing 5.9: Metoden, som kaldes af backgroundworker, i en anden tråd

```
private void ThreadedEncode(object sender, DoWorkEventArgs e)
{
    Tuple<Bitmap, string, string, string, bool, bool>
        EncodingArgument = e.Argument as Tuple<Bitmap, string,
        string, string, bool, bool>;

    Bitmap coverImage = EncodingArgument.Item1;
    String message = EncodingArgument.Item2;
    string encryptionKey = EncodingArgument.Item3;
    string stegoSeed = EncodingArgument.Item4;
    bool encrypt = EncodingArgument.Item5;
    bool compress = EncodingArgument.Item6;
```

```
    try
    {
        Bitmap stegoObject =
            _stegoModel.EncodeMessageInImage(coverImage, message,
            encryptionKey, stegoSeed, encrypt, compress);
        var EncodingInfo = new Tuple<Bitmap, string, string, bool,
            bool>(stegoObject, encryptionKey, stegoSeed, encrypt,
            compress);
        e.Result = EncodingInfo;
    }
    catch (NotifyUserException exception)
    {
        throw new Exception(exception.Message);
    }
}
```

Da vi ved at objektet, som vi har sendt med ind til backgroundworkerens metodekald, består af en Bitmap, to strenge og to boolean-værdier, kan vi bare anskue argumentobjektet som værende af denne type og få vores indsatte værdier ud af objektet igen. Nu har backgroundworkeren al den information, som den har brug for, til at indkode en besked i et billede.

Efter vi har modtaget informationen, udføres modellens `EncodeMessageInImage()`-metode, og det endelige stegoObjekt genereres. Herefter lægges al den information, som vi har brug for, når backgroundworkeren bliver færdig, ind i et nyt objekt og gemmes som `DoWorkEventArgs`' resultatobjekt.

Når backgroundworkeren er færdig, rejses et event, som kalder `ThreadedEncodeComplete()`, der fungerer meget lig den ovenstående. Først pakkes de resultatværdier, som vi har lagt i `DoWorkEventArg`'s resultatobjekt, ud af objektet og gemmes i tilsvarende variabler. Herefter ændres brugergrænsefladens skærmbillede til det nye stegoObjekt, en dialog til at gemme billedet åbnes, og brugeren notificeres om at indkodningen er foregået succesfuldt.

Kapitel 6

Test af program

For at lave et robust og velfungerende program er det essentielt at teste det systematisk. I dette kapitel vil test af det skrevne program blive gennemgået. Her vil vi se på to forskellige typer af tests, henholdsvis *unit tests* og *integrationstests*. Hvad disse tests indebærer vil blive beskrevet nærmere i afsnit 6.1. Hele programmet vil blive testet, men dette kapitel vil kun indeholde tests af de essentielle metoder og klasser i programmet, samt tests af hvordan disse klasser spiller sammen for at danne det endelige program. Med hensyn til steganografi vil der i afsnit 6.2 blive gennemgået, hvordan beskeden kan gå fra et byte-array til at blive klar til indlejring i dækmediet. Til kryptering vil der i afsnit 6.3 blive gennemgået en test, der viser denne klasses funktionalitet i programmet. Dette kapitel vil lede op til en vurdering der skal vise om det udviklede program overholder de krav der blev sat til det i afsnit 3.11 og altså kan anses for at være en fornuftig løsning på problemet der blev fundet.

6.1 Unit- og integrationstesting

Programmer kan testes med *unit tests*. Formålet ved unit tests er at teste de enkelte dele af programmet for sig selv. Unit tests skrives som metoder, der kalder de metoder eller klasser, som de forsøger at teste. For at gøre testprocessen hurtigere og mere systematisk anvendes testframeworks. I dette projekt anvendes visual studio tilføjelsen NUnit til at skrive og udføre vores unit tests. Integrationstests i dette projekt udfører vi selv, når vi samler de enkelte klasser i programmet og kombinerer dem til at udfører større delopgaver i programmet.

Testprincip for unit testing

Systematiske unit tests i Visual Studio laves ved først at oprette et nyt projekt [42]. Projektet får en henvisning til programmet, der skal testes, således at den kan kalde

programmets klassers offentlige metoder. Herefter laves i testprojektet en testklasse svarende til hver klasse, der skal testes. Klassen `StegoSystemWinForm.cs` i projektet `StegomaticProjekt` har således en unit test klasse, der hedder `StegoSystemWinFormTests.cs` og befinder sig i `StegomaticProject.Tests`.

Unit testing er især godt, da det giver god mulighed for at fange fejl tidligt i implementation af programmet. Hvis dette undlades, kan det lede til fejl, der eller ville være blevet undgået. Hvis der opstår problemer under den kombineret brug af af to metoder, så ved man ved systematisk brug af unit testing ofte præcist i hvilken del af programmet fejlen befinder sig, da man kan se om de enkelte metoder fungerer hver for sig. Uden god brug af unit testing har man derimod følgende mulige årsager til fejlen:

- En fejl i første metode.
- En fejl i anden metode.
- En fejl i begge metoder.
- En fejl i grænsefladen mellem de to metoder.

At finde fejl i programmer kan blive langt mere besværligt, når de forskellige metoder er blevet implementeret i det samlede program, frem for at de isoleret bliver testet inden implementationen og derefter også testes som en samlet enhed.

I testklasse oprettes enkelte metoder, som står for at teste den tilsvarende klasses metoder i et antal forskellige scenarier. Herefter vil det være muligt at få Visual Studio til at udføre alle disse tests og se hvilke af disse er blev gennemført med succes og hvilke ikke gjorde. For at gøre det lettere at overskue disse tests og finde fejlen i programmet, er det nødvendigt at have gode og beskrivende navne. Testklassernes navngivningen bør give en umiddelbar forståelse for hvad metoden tester. Hvis der så opstår en fejl, vil det være nemmere at gennemskue hvor fejlen er opstået og hvorfor.

Til dette navngives testmetoderne i overensstemmelse med Roy Osheroves navngivningskonvention, der fungerer på følgende måde:
MetodeUnderTest_Scenarie_ForventetResultat [48]. I listing 6.1 ses unit tests af komprimering i det endelige program samt initialisering af testklassen.

Listing 6.1: Setup af testklassen og fire unit tests af Compress-metoden i komprimeringsklassen

```
[TestFixture]
public class GZipStreamCompressionTests
{
```

```
    private GZipStreamCompression _gZipStreamTest;

    [OneTimeSetUp]
    public void Initialize()
    {
        _gZipStreamTest = new GZipStreamCompression();
    }

    [TestCase("12345678901234567890123456789012345678901234567890")]
    [TestCase("abcdefghijklmnopqrstuxyzabcdefghijklmnopqrstuxyz")]
    [TestCase("ABCDEFGHIJKLMNOPQRSTUXYZABCDEFGHIJKLMNOPQRSTUXYZ")]
    [TestCase("!#&/()=?!#&/()=?!#&/()=?!#&/()=?!#&/()=?")]
    [TestCase("1aA!1aA!1aA!1aA!1aA!1aA!1aA!1aA!1aA!")]
    public void Compress_String_ResultSizeLessOrEqual(string text)
    {
        byte[] byteText = ConvertToByteArray(text);
        byte[] compressedByteText =
            _gZipStreamTest.Compress(byteText);

        Assert.Less(compressedByteText.Length, byteText.Length);
    }
}
```

Hver gang testklassens testmetoder udføres, så køres `Initialize()` en enkelt gang og opretter et `GZipStreamCompression` objekt, som testmetoderne så kan anvende. Herefter udføres metoden `Compress_String_ResultSizeLessOrEqual()` en gang for hvert TestCase med det tilsvarende input. Input converteres til et byte array og komprimeres. Herefter undersøges om det resulterende byte array er mindre end det initierende. Testen er successfuld hvis dette er sandt og den fejler hvis ikke. Udover de viste tests er der i det endelige program flere tests af komprimeringsmetoderne, eksempelvis en, der tester om tekst, der er blevet komprimeret og så dekomprimeret, er lig den oprindelige tekst.

God unit testing kræver således at testmetoderne er velovervejede, da hver test kun bør teste et enkelt scenarie. I tilfælde af at der er behov for at teste private metoder for at opnå dette, bør man overveje om det er bedre at skrive metoderne på en alternativ måde, eksempelvis ved at lave de private metoder til offentlige metoder i klasser for sig selv.

Testprincip for integrationstest

Når et programs metoder er blevet unit testet, skal disse implementeres i det egentlige program. For at teste om helheden fungerer, udføres integrationstests. Princip-

pet bag integrationstests er at teste for, om de forskellige dele i et skrevet program kan interagere på den måde, som det var tænkt under designet af programmet.

Der findes endvidere flere forskellige måder at foretage integrationstests på [38], *top-down-*, *bottom-up-* og *paraplymetoden*.

Top-down Top-down tilgangen til integrationstests kræver at de største dele af programmet bliver integreret, før de mindre dele bliver. Denne type af integrationstests medfører, at den større logik i programmet bliver testet i starten og bliver fastlagt tidligt i integrationsforløbet. En ulempe ved denne form for integrationstests er dog, at der bliver brugt midlertidige metoder for de mindre dele af programmet, som betyder at små metoder først bliver testet sent i integrationsforløbet. Dette medfører også at der er en begrænset funktionalitet i programmet, indtil de mindre dele integreres.

Bottom-up Bottom-up tilgangen til integrationstests kræver, at de mindre metoder i programmet bliver testet og integreret som det første i integrationsforløbet. Dette medfører at den større funktionalitet i programmet bliver testet hurtigt i forløbet, mens de mindre dele i mindre dele efterhånden bygges op til en større funktionalitet. Dette medfører at der er en begrænset funktionalitet i programmet tidligt i integrationsforløbet.

Paraplymetoden Denne tilgang til integrationstests kan anses for at være en blanding af de to tidligere nævnte tilgange. Første trin i denne metode er, at input til metoder indsættes efter bottom-up tilgangen og efterfølgende bliver output fra de forskellige metoder sat ind efter top-down tilgangen. Fordelen ved denne metode er at brugen af midlertidige metoder minimeres, og derved sikrer funktionalitet, så snart de forskellige metoder implementeres. En ulempe er derimod, at denne tilgang er mindre systematisk end de to tidligere nævnte og kan derfor lede til, at flere tests skal gennemføres gennem implementationen.

Hovedsageligt har vi foretaget integrationstest med bottom-up tilgangen, dog med den undtagelse af at vi anvendte top-down tilgangen usystematisk under den initielle implementation af kommunikation mellem klasser. Til dette lavede vi en simpel consol-brugerflade, som implementerede interfacet `IStegoSystemUI`, og vi satte denne ind i stedet for klassen `StegoSystemWinForm`. Den simple consolbrugerflade brugte vi så til at få kommunikation mellem tryk på knapper i brugerfladen til at rejse events, der bliver opfanget af `StegoSystemController`, som så skriver en besked ud på brugerfladen. Så snart dette fungerede, så vidste vi at vores plan for programmets struktur ville kunne fungere. Udover dette har vi dog gennem implementationen af vores program har skrevet og implementeret de mindre metoder først, og efterfølgende bygget programmet op ud fra disse.

På listing 6.2 ses fire af vores systematiske integrationstests. Selvom disse er blevet implementeret vha. testframeworket nunit, så er det ikke enkelte "units"af funktionalitet de tester. I stedet testes programmets enkelte deles sammenspil i `StegoSystemModelClass`.

Listing 6.2: Fire integrationstests af `StegoSystemModelClass`

```
[TestCase(true, true)]
[TestCase(true, false)]
[TestCase(false, false)]
[TestCase(false, true)]
public void EncodeDecode_Configuration_NoLossOfData(bool encrypt,
    bool compress)
{
    Bitmap stegoObject;
    string decodedMessage;
    stegoObject = _stegoModel.EncodeMessageInImage(new
        Bitmap(_image), _message, _key, _seed, encrypt, compress);
    decodedMessage =
        _stegoModel.DecodeMessageFromImage(stegoObject, _key,
        _seed, encrypt, compress);
    Assert.AreEqual(decodedMessage, _message);
}
```

I disse fire tests undersøges specifikt om brug af kryptering og komprimering gør nogen forskel for den afkodede besked, og derved om integration af disse i programmet er passende. Alle variabler, der starter med et underscore, er private klassevariabler. I dette tilfælde er det blot nogle standardværdier.

6.2 Test af behandling af besked

Dette afsnit vil omhandle test af hvordan tekstbeskeden, som skal gemmes i dækmediet, behandles i programmet. Beskeden bliver i programmet behandlet som et array af bytes, når den skal indlejres i dækmediet. For at beskeden kan indlejres, bliver hver byte først opdelt i fire par af bits, som repræsenterer værdierne mellem nul og tre. Herefter bliver værdierne tilføjet til en liste af værdier, der samlet udgør beskeden. Metoden, der udfører dette, ses på listing 6.3.

Listing 6.3: Behandling af besked i bytes

```
public List<Byte> ByteArrayToValues(byte[] byteArray)
{
```

```
    List<Byte> Values = new List<byte>();

    foreach (Byte item in byteArray)
    {
        BitArray bitValues =
        new BitArray(BitConverter.GetBytes(item).ToArray());
        for (int index = 7; index > -1; index -= 2)
        {
            if (bitValues[index] == true &&
            bitValues[index - 1] == true)
            {
                Values.Add(3);
            }
            else if (bitValues[index] == true &&
            bitValues[index - 1] == false)
            {
                Values.Add(2);
            }
            else if (bitValues[index] == false &&
            bitValues[index - 1] == true)
            {
                Values.Add(1);
            }
            else
            {
                Values.Add(0);
            }
        }
    }
    return Values;
}
```

Denne kode splitter en byte op i fire bit-par og ser derefter hvilke værdier disse har. For eksempel hvis et par af bits er: 11, der også kan betragtes som *true* og *true*, vil værdien på decimalform være 3. Hvis dette er tilfældet, vil denne værdi blive tilskrevet til en liste, der holder værdierne for alle parrene i beskeden. Denne proces bliver udført for alle bytes, som beskeden består af. Et eksempel på brugen af dette er hvis karakteren *z* skal ændres til denne repræsentation. *z* har, ved ASCII-bitrepræsentation, decimalværdien 122. I form af en byte er dette værdien 01111010, som findes i det array, som `ByteArrayToValues()` modtager. De bit-par, der vil blive dannet ud fra denne byte, vil være 01, 11, 10 og 10, ifølge koden.

På decimalform vil værdierne være 1, 3, 2 og 2, der er det forventede output og stemmer over ens med det output, der blev givet af metoden.

En ting, der er værd at lægge mærke til, er at for-løkken i `ByteArrayToValues()` tæller ned og ikke op. Dette er grundet at en byte i form at et array af bits benytter *little endian*. Dette betyder at det er det mindst signifikante bit, der har den første plads i arrayet, altså plads 0, hvorfor arrayet skal indeles i par bagfra, for ikke at ændre på beskeden. Hvis ikke der var taget højde for dette, vil værdien for z blive læst som 01011110, hvilket svarer til *cirkumfleks-tegnet* og ikke z.

6.3 Test af kryptering

Som bekendt har vi i dette program valgt at bruge kryptering af tekstbeskeden der skal indlejres i dækmediet. Kryptering og dekryptering foregår i hver sin metode i programmet. Disse to metoder minder om hinanden og der vil i dette afsnit kun blive taget udgangspunkt i en af de to metoder, dette vil være metoden til at kryptere en klartekst. Denne metode tager to strenge som parametre. Den ene af disse er beskeden og den anden er et kodeord, hvor metoden så returnerer en streng i form af en ciffertekst. Det samme gælder for metoden der dekrypterer, her er det blot en ciffertekst frem for en klartekst der bruges som den ene parameter, og her bliver den originale klartekst returneret frem for en ciffertekst, hvis kodeordet stemmer overens med de oprindelige. Metoden for kryptering af en klartekst kan ses på 6.4.

Listing 6.4: Kryptering af klartekst

```
/*Method for encryption of the plaintext*/
public string Encrypt(string plainText, string password)
{
    byte[] encrypted;
    string cipherText = null;

    /*New instance of the AES-class*/
    using (RijndaelManaged aesAlg = new RijndaelManaged())
    {
        byte[] salt = new byte[] { 0x26, 0xdc, 0xff, 0x00, 0xad,
            0xed, 0x7a, 0xee, 0xc5, 0xfe, 0x07, 0xaf, 0x4d, 0x08,
            0x22, 0x3c };
        Rfc2898DeriveBytes key = new Rfc2898DeriveBytes(password,
            salt);

        aesAlg.Key = key.GetBytes(32); //256-bit Key
```

```
        aesAlg.IV = key.GetBytes(16); //128-bit IV

        /*Creates encrypter*/
        ICryptoTransform encryptor =
            aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

        /*Streams for encryption*/
        using (MemoryStream msEncrypt = new MemoryStream())
        {
            using (CryptoStream csEncrypt = new
                CryptoStream(msEncrypt, encryptor,
                CryptoStreamMode.Write))
            {
                using (StreamWriter swEncrypt = new
                    StreamWriter(csEncrypt))
                {
                    /*Writes all data to the stream*/
                    swEncrypt.Write(plainText);
                }
                /*Byte-array to encrypted text*/
                encrypted = msEncrypt.ToArray();
            }
        }
    }

    /*Returns ciphertext as string*/
    cipherText = Convert.ToBase64String(encrypted);
    return cipherText;
}
```

Beskeden bliver behandlet gennem en krypteringalgoritme, her Rijndael-algoritmen (AES), for at danne en ciffertekst der kan skjules i dækmediet ved brug af steganografi. Klarteksten bliver gennem krypteringsalgoritmen behandlet som et array af bytes og kommer derfor også ud som et array af bytes når krypteringen er gennemført. Dette byte-array bliver efterfølgende oversat til en tekststreng der udgør cifferteksten. Til at gennemføre krypteringen bliver der brugt en nøgle `key` og en initialization vector `IV`. Disse bliver genereret ud fra det indtastede kodeord samt et array af bytes kaldet `Salt`, der er faste hexadecimale variable i programmet i dets nuværende tilstand. Output af en test af de to metoder ses på listing 6.5.

Listing 6.5: Output de to krypteringsmetoder

```
Original input:   A text for AES encryption!
Ciphertext:       zamqvlTpBmNTv3xmzXHhgVF5PtJACr+SQhWr19ewFlk=
Output:           A text for AES encryption!
```

6.4 Data fra indlejringsprocessen

I dette afsnit har vi samlet det relevante data, som der måtte være under indlejringen. Herunder antallet af aktive knuder, dvs. dem der ikke allerede tilfældigvis svarer til det indlejrede data, antallet af kanter der bliver oprettet i alt, efterfølgende de kanter som `CalcGraphMatching()` finder, som svarer til de dannede par, og også hvor mange knuder faktisk bliver indlejret ved hjælp at ombytning og hvor mange procent det var ud af det aktive knuder. Desuden er størrelsen af det indlejrede data taget med.

6.4.1 Data

For hver enkel tabel er der blevet brugt et almindeligt 8 megapixel billede, hvor der for hver indlejring altid er brugt et ubehandlet billede og samme seed (stegokodeord) uden anvendelse af kryptering eller komprimering.

Datastørrelse(byte)	Aktive knuder	Kanter i alt	Dannede par	Knuder indlejret
504	1516	12.101	1391	984 (64.9%)
1005	2992	48.586	2870	1806 (60.3%)
1505	4488	107.458	4351	2568 (57.2%)
2005	5977	189.487	5835	3214 (53.7%)
4005	11.986	771.492	11.845	5312 (44.3%)
8005	23.970	3.096.560	23.830	8362 (34.8%)

Tabel 6.1: Data for indlejring med input: Png. Output: Png. PixelMetode: `GetRandomPixelsAddToList2()`

Datastørrelse(byte)	Aktive knuder	Kanter i alt	Dannede par	Knuder indlejret
504	1518	31.263	1467	808 (53.2%)
1005	2662	101.166	2608	1268 (47.6%)
1505	3026	128.876	4479	1762 (38.8%)
2005	6036	518.443	5978	2108 (34.9%)
4005	11952	2.043.492	11898	3112 (26.0%)

Tabel 6.2: Data for indlejring med input: Jpeg. Output: Png. PixelMetode: `GetRandomPixelsAddToList2()`

Datastørrelse(byte)	Aktive knuder	Kanter i alt	Dannede par	Knuder indlejret
504	1732	249.132	1728	358 (20.6%)
1005	3006	736.310	2994	470 (15.6%)
1505	4493	1.635.079	4480	548 (12.1%)
2005	6007	2.890.372	5993	624 (10.3%)

Tabel 6.3: Data for indlejring med input: Png. Output: Png. PixelMetode: `GetRandomPixelsAddToList1()`

6.5 Test af indlejringen af data

I dette afsnit opstiller vi en hypotese, for hvordan vi mener udviklingen bør se ud, i forhold til hvor mange knuder der bliver indlejret ved hjælp af pardannelsen og hvor mange der bliver modificeret. Efterfølgende præsenterer vi noget data, baseret på vores implementation, der viser sammenhængen mellem hvor stor en procentdel af knuderne bliver indlejret ved pardannelse, i forhold til nogle forskellige størrelser af data, som vi ønsker at indlejre.

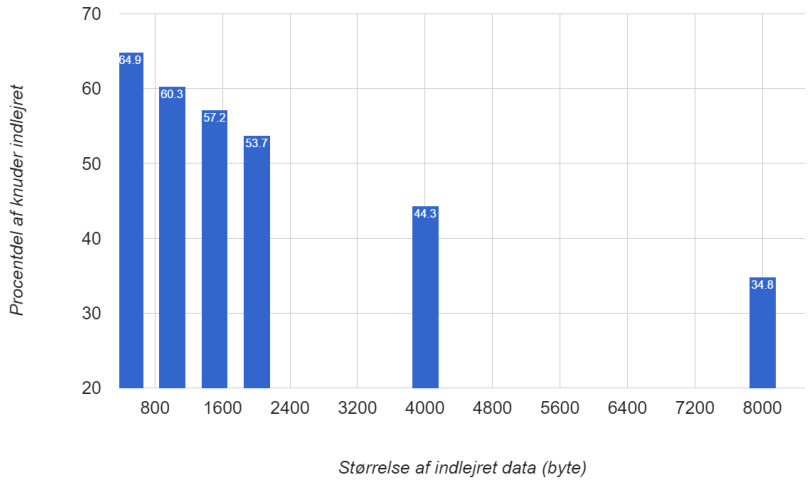
6.5.1 Hypotese

Denne hypotese omhandler vores forventning omkring størrelsen af det indlejrede data, i forhold til mængden af knuder, som blev indlejret ved hjælp af ombytning, som bestemmes ud fra kanterne fundet via `CalcGraphMatching()`. Vi forventede at se en forøgelse af antallet af kanter, jo større mængde data der skulle indlejres og at indlejringen af knuder vha. ombytning ville være nogenlunde stabil.

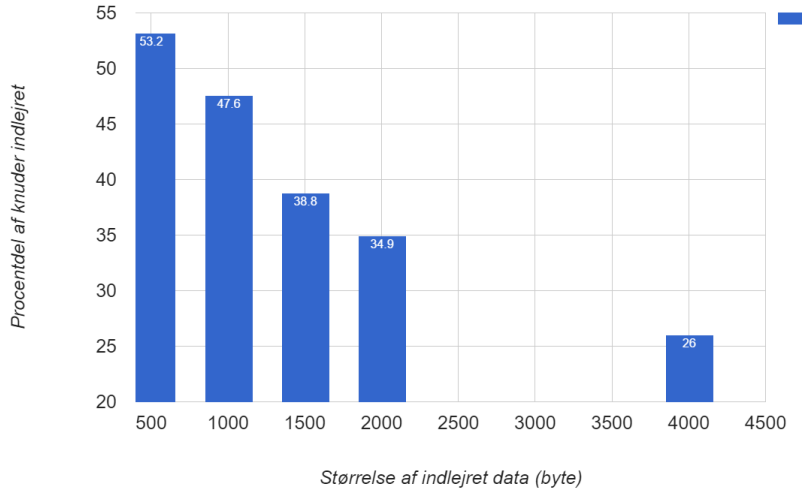
Grunden til at vi forventer flere kanter når datamængden bliver større, er at der bliver oprettet flere knuder, og dermed vil der være flere muligheder for at danne kanter. Dette burde til gengæld ikke have en effekt på den procentvise indlejring af knuder (antallet af par), da der altid dannes par for den knude, med færrest kanter først.

6.5.2 Resultater

De følgende figurer (6.1 og 6.2) er baseret på det data, som ligger i bilag 6.4.



Figur 6.1: Indlejring af data for Png, hvor filformatet var Png både for dækmediet og stego-objektet.



Figur 6.2: Indlejring af data for Png, hvor filformatet var Jpeg for dækmediet og Png for stego-objektet.

Ud fra figur 6.1 og 6.2 kan vi se at vores implementation ikke stemmer overens, med vores anden forventning, idet vi får indlejret færre knuder vha. ombytning, når datamængden stiger. Vores første forventning passer dog i forhold til data, som kan ses på tabellerne i afsnit 6.4, som var at vi får flere kanter, når datamængden stiger.

Med denne overordnede program test kan vi dermed sige, at der kan optimeres på metoden `CalcGraphMatching()`, idet der dannes et et antal kanter (tabel 6.1, afsnit 6.4), som bliver større eftersom mængder af data stiger, men hvor antallet af dannede par er faldende.

6.6 Opsummering af test

I dette afsnit vil vi opsummere hvordan tests af programmet er gået, samt hvordan tests af programmet har bidraget til at forøge kvaliteten af det skrevne program.

Under kapitel 6 fremgår det, at vi har foretaget både unittests og integrationstests. Vi har brugt unittests til at teste hver enkelt metode der er blevet skrevet, samt brugt integrationsstets til at teste flere metoder, der er blevet sat sammen. Efter unittests på de enkelte metoder og integrationstests på de sammensatte metoder, er de efterfølgende blevet tilføjet til det fulde program for projektet.

Under de foretagede unittests og integrationstests er der ikke blevet opdaget nogle større fejl, som var fatale for programmets udførsel. Dette kan skyldes at datastrukturen for programmet var meget fastlagt, før at den egentlige programmering begyndte, så der var på forhånd nået frem til hvad de enkelte metoder skulle opfylde, samt hvad der skulle tages ind som parametre, og hvad de enkelte metoder skulle returnere.

Dog har unittests og integrationstests været med til at spare en del tid når programmet skulle rettes for små fejl, og når der skulle laves optimeringer af de enkelte metoder. Udførelsen af tests har nemlig hjulpet til, at vi kunne vide os sikre på at de skrevne metoder opførte sig på den måde, som det var intentionen at de skulle, frem for bare at have en formodning om deres egentlige opførsel i programmet.

Kapitel 7

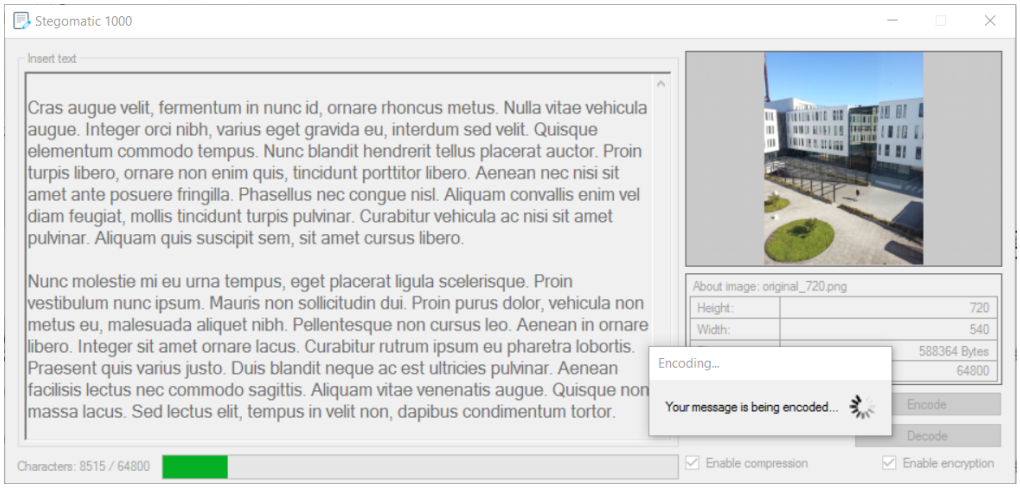
Afrunding

I dette kapitel vil vi konkludere på projektet, samt diskutere den valgte løsning på problemet og perspektivere til, hvad der videre kunne arbejdes med, hvis dette projekt ville fortsættes. Konklusionen vil tage udgangspunkt i problemformulering der findes i afsnit 2.4, kravene der findes i afsnit 3.11 og det skrevne program. Her vil disse blive vejet op mod hinanden for at vurdere, om det valgte problem kan siges at være løst.

7.1 Det færdige program

Til slut er det endelige program naturligvis blevet testet manuelt og usystematisk, og ændringer er blevet foretaget indtil fejl og mangler er blevet udbedret og programmet var som vi havde tiltænkt os. I dette afsnit demonstreres det endelige programs hovedfunktionalitet. Denne demonstration udføres med både komprimering og kryptering aktive. Det forsøges at indkode 1275 ord (8484 tegn) i et billede med krypteringsnøgle og steganografikodeord begge sat til 150.

På figur 7.1 ses programmet under indkodningsprocessen.

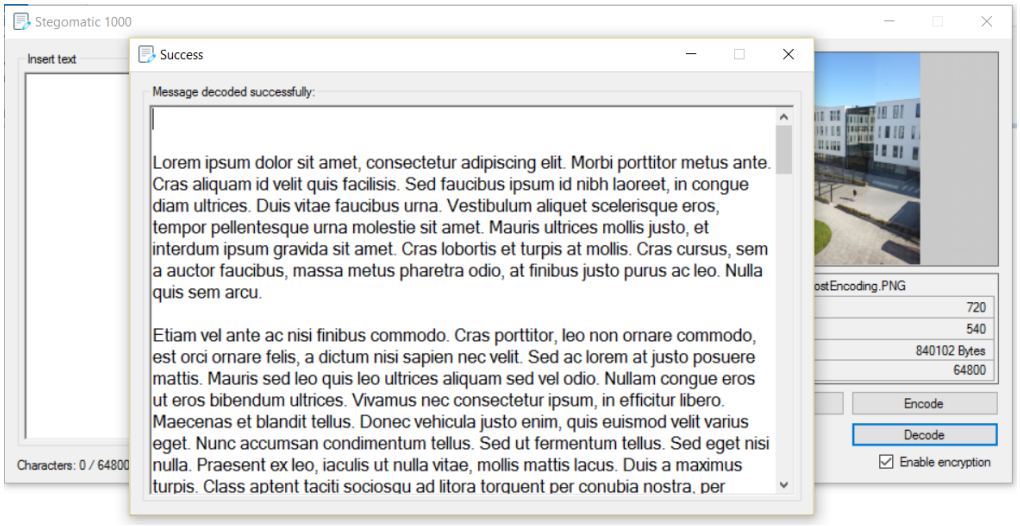


Figur 7.1: Data indlejres i billedet.

På figur 7.2 ses billedet efter indlejring af de 8484 tegn side om side med billedet før indlejring.

Figur 7.2: Billedet før og efter indkodning af 8484 tegn.

På figur 7.3 ses programmet ved successfuld afkodning af beskeden.



Figur 7.3: Programmet efter afkodning af billedet.

7.2 Diskussion

I dette afsnit vil vi diskutere det endelige produkt og dets svagheder. Herunder vil der blive redegjort for nogle af de overvejelser og fejl, som ledte dertil. Herunder vil vi desuden beskrive alternative løsningsmetoder.

Produktets målgruppe

Under afgrænsningen i afsnit 2.3 i problemanalysen blev der, som hovedmålgruppe for dette projekt, valgt aktivister. Der var dog ikke meget fokus på denne interessant senere i projektforbløbet og i afsnit 3.11 om krav i teknologianalysen blev det valgt, at der i produktet ikke længere skal tages hensyn til hvilke ønsker en aktivist måtte have til et sådant produkt. Dette valg har haft den betydning for projektet, at produktet ikke er lavet med henblik på et bestemt brugersegment, men derimod er lavet for at demonstrere steganografi ved brug af grafteori, samt en algoritme med en lav tidskompleksitet, for at løse pardannelsesproblemet beskrevet i 3.7. En grund til at fokus på et bestemt brugersegment, som aktivister, blev tilsidesat var pga. manglende tid til at undersøge målgruppen ordenligt. Der blev derfor ikke lavet en større undersøgelse med et specifikt fokus på målgruppen aktivister. Hvis dette projekt eller et lignende skulle fortsættes eller laves, eller hvis der havde været mere tid, kunne det være en fordel at kontakte en målgruppe for projektet for at få input, der kunne være med til at danne krav der var mere målrettet mod et bestemt brugersegment. Dette kunne, i bedste fald, også føre til at produktet kunne være til gavn for en bestemt målgruppe, frem for bare at være lavet.

Sociale medier og komprimeringsproblemet

Som det fremgår i afsnit 3.10 om komprimeringsproblemet, blev det valgt ikke at tage hensyn til komprimering fra kanaler, som vi ikke har mulighed for at kontrollere. Hvis dette projekt skulle fortsættes, et lignende skulle startes op, eller hvis der havde været mere tid, ville dette have været et punkt, der kunne fokuseres mere på. Hvis dette problem kunne løses med fokus på et bestemt medie som Facebook, ville dette være med til at sikre at et stego-objekt ikke kun kan deles på kanaler, som ikke udfører tabsgivende komprimering på dets datastrøm.

Pixelombytningsprocent ved brug af grafteori

Ud fra kapitel 6, der omhandler tests af programmet, fremgår det, at programmet godt kan udføre steganografi ved brug af grafteori, samt kryptering og dekryptering ved brug af AES-algoritmen og endvidere også komprimering og dekomprimering af data til indkodning og afkodning. I det endelige produkt bliver der, i algoritmen der udfører steganografi, modificeret flere pixels i dækmediet end der ønskes, når en besked indlejres for at danne et stego-objekt. Dette har vi en

formodning om kan have noget at gøre med den måde, hvorpå kanter mellem knuder bliver lavet i algoritmen, men vi har ikke haft tilstrækkelig meget tid til at få denne hypotese testet ordenligt igennem. Programmet kan, afhængigt af farvediversiteten i billedet opnå ca. 60% ombytninger af pixels, hvorefter resten bliver modificeret for at beskeden er indkodet succesfuldt og kan dekodes igen ved brug af programmet. Hvis dette projekt eller et lignende skulle laves eller fortsættes, eller hvis der havde været mere tid, ville vi have undersøgt dette problem nærmere med henblik på at opnå en endnu højere procentsats for ombytninger af pixels. Ud fra dette kan det også konstateres, at der kunne være lagt endnu mere fokus på at undersøge den grafteoretiske tilgang til steganografi.

Upræcis programmering

En anden svaghed ved vores program er at de enkelte komponenter ikke er fuldstændigt afstemte med hinanden, og at programmet derved udfører enkelte operationer dobbelt. Eksempelvis konverteres beskeden i `RjindaelCrypto` mellem tekststrenge og byte-arrays, hvilket er unødvendigt, da `ByteConverter` også bliver kaldt i `StegoSystemModelClass`.

Desuden er backgroundworkeren også blevet implementeret som en eftertanke, og var ikke blevet overvejet grundigt nok på forhånd. Eksempelvis var det en fejl at implementere backgroundworkeren i `StegoSystemControl`, da det hverken gav os mulighed for at afbryde operationen eller opdaterer en form for statuslinje.

Unit testing og algoritmens private metoder

Vi har under implementation af programmets enkelte dele, især den grafteoretiske metode, ikke tænkt tilstrækkeligt over hvordan vi havde planlagt at teste de vigtige dele. Dette betyder eksempelvis at vi har haft for mange metoder, som vi ikke har kunnet få adgang til at unit teste, da de var private metoder. Mange af disse bør nok have været skrevet som offentlige metoder i klasser, der er indlejrede i de overordnede klasser.

Desuden har vi også haft for mange metoder i klassen `GraphTheoryBased`, som vi ikke vidste hvordan vi skulle unit teste. Metoderne er svære at forstå og har ikke meget funktionalitet for sig selv, hvilket gjorde det svært at skrive unit tests. Dette gjorde at algoritmen egentligt kun er blevet systematisk testet vha. integrationstest af `GraphTheoryBased`. Mange af algoritmens dele er naturligvis blevet testet usystematisk med `Console.WriteLine()`, dog ville det nok have været lettere at finde fejl hvis vi havde tænkt mere over unit tests under selve implementationen af algoritmen.

For store klasser og metoder

Programmets programmeringsstil og læselighed lider en smule af at enkelte klasser og metoder i løbet af implementationen har vokset sig større end forventet. For at udbedre dette problem burde vi have været mere konsekvente i at dele programmet op i mindre dele. Et eksempel på dette problem er den statuslinje på brugergrænsefladen, der viser billedets kapacitet. Det var tænkt som en mindre del af programmet og var derved ikke så vel planlagt. Denne statuslinje startede med blot at være en enkelt metode i brugerfladens hovedmenu, Winformklassen `Form1`. Herefter bemærkede vi dog at vi have brug for metoderne `ForceUpdateProgressBar` og `ForceUpdateCapacity` for at få statuslinjen til at opdatere når brugeren tænder og slukker for komprimering. Aktuelt består statuslinjen af fire metoder og en privat klassevariabel i `Form1`, hvorimod den burde have været en klasse for sig selv.

Ikke-troværdige værdier vises til brugeren

Programmets troværdighed og hvor let det er at anvende bliver skadet af at ikke-troværdige værdier vises til brugeren. Det bedste eksempel for denne svaghed er tallet for estimeret kapacitet af billedet. Dette tal er helt passende så længe der blot gøres brug af steganografi, men både kryptering og komprimering ændrer beskedens længe som funktion af beskeden. Med andre ord, så vides ikke hvor lang teksten bliver efter de operationer indtil man udført operationen. Problemet er ikke nødvendigvis det upræcise tal, men mere at brugeren ingen steder vises hvordan og hvorfor det er upræcist.

Kun tekst som input

Blot et 500 x 500 pixel billede kan uden komprimering og kryptering indeholde op til 20833 bytes data, hvilket også er 20833 tegn. Dette er en mængde tekst, der vil være svær for brugere at udnytte ordentligt så længe vi kun tillader brødtekst. For at programmets potentiale ikke ville blive spildt på denne måde, bør vi have givet mulighed for at overføre eksempelvis en eller flere filer.

7.3 Konklusion

I det følgende afsnit vil vi konkludere på dette projektforløb og tage stilling til hvor meget der blev opnået, og om dette projekt egentlig nåede sine mål. Til at besvare dette tages der udgangspunkt i de underspørgsmål, som blev stillet i problemformuleringen i afsnit 2.4. Underspørgsmålene var som følge:

- *Hvordan indlejres beskeden i dækmediet?*

- *Hvordan krypteres beskeden og hvordan dannes nøgler?*
- *Hvordan sikres integritet mht. filformat og komprimering?*

Det første spørgsmål har vi besvaret ved at anvende steganografi baseret på grafteori. Det har resulteret i et program, hvori pixels ombyttes eller modificeres, for at en given besked bliver indkodet i et dækmedie bestående af et billede. Beskeden kan endvidere ekstraheres ved brug af det samme frø, som der blev brugt til at indkode beskeden med.

Til det andet spørgsmål bliver der til kryptering af klarteksten, som skal indlejres i dækmediet ved brug af steganografi, brugt AES-kryptering. AES-algoritmen indgår i programmet som den indbyggede metode, der er givet i .NET fra Microsoft. Denne algoritme omdanner ved brug af en nøgle klarteksten til en uforståelig cifferteksten, og optræder derfor som et ekstra lag af sikkerhed i programmet. Nøglen genereres af programmet ud fra et brugerdefineret kodeord og en række indbyggede værdier og anvendes til både kryptering og dekryptering. Denne proces og kryptering generelt er dog ikke blevet undersøgt nærmere, da dette ikke har været det primære fokuspunkt for projektet.

Til det tredje spørgsmål sikrer vi integriteten ved at de tabsfri filformater *PNG* og*BMP* er de eneste, som vi tillader at eksportere fra programmet. Herved er det sikret at beskeden forbliver indlejret i billedet og kan læses igen, hvilket ikke er garanteret ved brug af tabsgivende filformater. Vi valgte senere i projektforløbet ikke at tage hensyn til komprimering fra domæner, som vi ikke er i kontrol over. Herunder er Facebook samt andre sociale medier, der komprimerer deres billeder.

Som nævnt i afsnit 6.5 om test af indlejring i dækmediet fremgår det, at vi ikke er helt tilfredse med hvor mange pixels der egentlig bliver byttet, i forhold til hvor mange kanter der bliver konstrueret mellem de forskellige knuder. Dette medfører at størstedelen af pixels i dækmediet bliver modificeret, frem for at blive byttet. Beskeden indlejres stadig, men ikke på den optimalt skjulte måde i forhold til grafteori. På tabellerne 6.1 til 6.3 i kapitel 6, fremgår det at der er næsten ligeså mange par, som der er aktive knuder, hvilket ikke burde kunne lade sig gøre. Hvis to knuder har dannet et par, så burde de ikke kunne indgå i nogle andre. Derfor burde der kun kunne være halvt så mange dannede par, som der er aktive knuder. En hypotese på hvad der sker, er at nogle par gentager sig, og der altså dannes flere af de samme par. En anden hypotese er, at knuderne kan bryde regelen og indgår i mere end et par hver, hvilket påvirker antallet af ombytninger på en ukendt måde, og forårsager at ombytningerne bliver så få.

Vi kan ud fra de ovenstående besvarelser af problemformuleringens spørgsmål konkludere at projektet har nået sit mål, med blot få mangler. Manglerne udgør hovedsageligt at der ikke er taget hensyn til en specifik målgruppe, og at der ikke er taget hensyn til komprimeringen på sociale medier.

7.4 Perspektivering

I dette afsnit vil vi perspektivere til ændringer, der måtte være nødvendige at foretage, hvis det fremstillede steganografi-værktøj skulle bruges af et reelt brugersegment, frem for bare at være fremstillet for at demonstrere grafbaseret steganografi.

Større fokus på målgruppe

Hvis dette projekt skulle fortsættes og anvendes i større omfang i virkeligheden, så ville det være en klar fordel, at der fra starten tages hensyn til et brugersegment eller en bestemt målgruppe, der i vores tilfælde ville have været aktivister. Målgruppen skulle så analyseres dybdegående og endvidere kontaktes, for at få indblik i, hvad for nogle krav de måtte have til kommunikationsværktøj af denne type. Dette er for at sikre at programmet, der efterfølgende bliver udviklet, vil opfylde de behov som målgruppen har, og kan bruges af den tiltænkte målgruppe. Det er specielt her at kontakten med målgruppen kan vurderes værende nødvendigt, da de vil blive inddraget i udviklingsprocessen, frem for at vi bare arbejder ud fra antagelser om målgruppens ønsker til programmet.

Flere filformater

Det ville endvidere være fordelagtigt hvis det skrevne program understøttede flere forskellige filformater når et stego-objekt laves, da programmet i dets nuværende tilstand kun kan eksportere de tabsfri filformater `PNG` og `BMP`. Her ville det være interessant at se om det også ville være muligt at eksportere tabsgivende filformater som `JPEG`, eller endda arbejde på helt andre typer af filer, som lyd eller video. Det vil dog kræve mere viden, da grafteori anvendes forskelligt på lyd og video end på billeder. Hvis programmet kan udføre steganografi på flere forskellige filtyper, vil det udvide programmets brugbarhed og fleksibilitet, alt efter hvilke filer målgruppen har til rådighed, og hvilke filer målgruppen ønsker at benytte.

Flere algoritmer til steganografi

Det kunne også være relevant at implementere flere algoritmer til steganografi i det skrevne program. Dette er igen for at kunne imødekomme en eventuel målgruppes behov til programmet. Eksempelvis kunne den palette-baserede metode til steganografi implementeres, eller den simple LSB-metode, alt efter hvad målgruppen har af behov.

Test og evt. maksimal billedstørrelse

For at undgå fejl i det færdige program, kan det desuden være meget nyttigt at teste hvorledes programmet fungerer når størrelsen af det anvendte billede bliver

forøget ud over det forventede. Hvis sammenhængen mellem billedstørrelse og tidsforbrug eller billedstørrelse og hukommelse er eksponentiel, så kan det evt. være meget nyttigt at definere en maksimal billedstørrelse.

Procesanalyse

Dette er procesanalysen for gruppe A419. Vi har valgt at strukturere analysen på følgende måde: Beskrivelse, analyse, vurdering og syntese. Herunder bliver alle relevante emner først beskrevet, hvorefter de bliver analyseret osv. I dette projektforløb har vi haft et samarbejde med to andre grupper, på tværs af studieretninger, hvilket vi referer til som klyngesamarbejdet eller samarbejdet i klyngen. Vi samarbejdede med matematikgruppen A402 og softwaregruppen A325a, og dette samarbejde reflekterer vi også over i denne analyse.

Beskrivelse

Samarbejde i gruppen

Vi startede med at udarbejde en gruppekontrakt, hvori vi fastslog retningslinjer for gruppesamarbejdet. Punkter såsom forventede mødetider, og at gruppen skulle kontaktes i tilfælde af fravær, blev klarlagt. Derudover blev det aftalt, at vi skulle have et opsamlingsmøde hver mandag, hvor vi ville gennemgå hvad der var blevet lavet siden sidste møde, hvem der skulle rette hvilke arbejdsblade, hvad der forventes at skulle sendes til vejlederen senere på ugen, samt hvad der skulle laves i løbet af ugen.

Arbejdsfordelingen i gruppen blev organiseret, således det enkelte medlem af gruppen fik ansvar for den del af arbejdsbladet, som han selv arbejdede på. Efter en rette-runde af alle arbejdsblade, der var blevet udarbejdet i løbet af ugen, der involverede hele gruppen, blev de sendt til vejlederen. Efter vi havde modtaget vejleders feedback, så rettede hvert gruppemedlem sine egne arbejdsblade ud fra dette. Retterunden var også primært der, hvor vidensdelingen i gruppen fandt sted, da der kunne spørges ind til de enkelte afsnit.

Det blev aftalt at rollen som referent skulle gå på skift fra møde til møde, og at rollen som korrespondent skulle ligge fast hos et enkelt medlem af gruppen. Der blev ikke valgt en specifik ordstyrer eller leder af gruppen, da der ikke var stemning for at have én, som dikterede arbejdet i gruppen.

Samarbejde med vejleder

Til start i projektarbejdet formulerede vi ud fra gruppemedlemmernes erfaringer fra P1 nogle forventninger og ønsker til vejledertype og form for feedback, som vi så talte med vejleder Hans Hüttel om. Vi havde ønsket at vejlederformen var produktorienteret, således at vi i vejledningen hovedsageligt fokuserede på kritik af de udarbejdede afsnit og produktfremstillingen. Dette havde vi håbet, idet der ud fra projektoplægget og studieordningen, blev langt op til at projektet ville have stor fokus på det endelige produkt. Vejlederen var dog uenig i det foreslåede og mente, at det ville være mere nyttigt med delt produkt- og process-vejledning. Dette efterkom vi ved at indarbejdede en fast dagsorden i de ugentlige vejledermøder. Dette krævede, at vi til hvert vejledermøde ville behandle eventuelle problemer i gruppens samarbejde og vidensdeling.

Vi aftalte derudover, at vi i slutningen af hver arbejdsuge ville sende materiale (arbejdsblade), samt en læsevejledning til vejleder, som vi ville tage op under følgende vejledermøde. Herefter ville vi så rette materialet i forhold til disse kommentarer, samt tilføje mere og så sende det samlet til vejleder ved ugens afslutning. Disse tidsfrister kunne dog ændres ved aftale, hvis arbejdsbyrden i nogle enkelte uger blev for stor. Udover ugentlige vejledermøder og arbejdsblade, som vi sendte, så har vores vejleder også været en del af enkelte klyngemøder (der vil blive uddybet senere) og vi har sendt spørgsmål om teori til ham, hvis dette var nødvendigt.

Vejlederens respons har, især i projektets start, ofte været fokuseret på relevansen af de skrevne afsnit og om de også var nyttige at medtage i den endelige rapport. Udover dette blev vi også rådet til at ændre vores struktur af vores projektarbejde. Grunden var, at dette ville gavne os, ikke blot i dette projekt, men også fremadrettet. Den største ændring vi fik foretaget i denne sammenhæng, var at vi fik omstillet vores tilgang til at skrive materiale til projektet. I stedet for at skrive og udfylde en forudbestemt rapportstruktur, skrev vi i stedet arbejdsblade, der indeholdte alt det nødvendige på daværende tidspunkt. På denne måde undgik vi fremadrettet en stor del af det overflødige skrivearbejde.

Projektplanlægning og projektstyring

Gennem projektet har vi i projektgruppen brugt værktøjer som Trello og Slack, samt en arbejdsplan for at planlægge og styre projektet. Endvidere er tjenesten ShareLaTeX blevet brugt til at lave og sammensætte skrevne arbejdsblade til den endelige projektrapport.

Mere specifikt har vi brugt værktøjet Trello til at danne overblik over hvilke opgaver de individuelle gruppemedlemmer varetog, hvilke opgaver der skulle laves mht. arbejdsblade og programmering, samt hvilke opgaver, der var klar til at blive

rettet af et andet gruppemedlem. Alt dette blev gjort med "kort", der oprettes og placeres under de forskellige kategorier, såsom "skal laves", hvortil én eller flere personer kunne tilknyttes kortet. Alt efter status på kortet kunne de flyttes over i andre kategorier såsom "skal rettes"eller "færdig".

Tjenesten Slack har vi brugt til at komme i kontakt med hinanden uden for arbejdstiden eller i tilfælde af sygdom. Gennem Slack har det eksempelvis været muligt at uddele opgaver til sygdomsramte gruppemedlemmer. Endvidere er Slack også blevet brugt til at uddele skrevne referater fra vejledermøder og kursusforelæsninger, samt generel fildeling. Delingen er sket gennem kanaler på Slack, der er tilknyttet de enkelte kurser, samt en kanal til vejledermøder. Slack er endvidere blevet brugt til at dele kilder og læsematerialle mellem gruppens medlemmer.

Der blev for dette projekt også lavet en arbejdsplan. Denne lavede vi i gruppen under projektforløbet, hvorpå arbejdsopgaver blev sat, som skulle laves inden projektet skulle afleveres, meget ligesom det blev gjort på Trello. Arbejdsplanens formål var at skabe et overblik over, hvor meget arbejde, der var tilbage, som skulle laves før aflevering. Opgaverne fra denne arbejdsplan blev lavet til "kort"på Trello efterhånden som der der var opsamlet nok viden om emnerne, og de blev relevante at lave i forhold til hvor langt i projektforløbet vi befandt os.

Gennem projektforløbet har vi i gruppen ikke benyttet os af en projektleder. I stedet for dette, er det blevet valgt, at alle gruppens medlemmer sammen skulle danne sig et overblik over hvad projektet skulle indeholde, hvorefter den føromtalte arbejdsplan blev udarbejdet. Hvis der opstod spørgsmål i forhold til opgaver under projektforløbet, blev dette taget op i fællesskab af gruppen og en beslutning blev truffet for hvad en mulig løsning kunne være og hvordan projektet burde ledes videre.

Læreprocessen

I dette projektforløb har vi haft begrænset samarbejde angående løsning af opgaver fra kurserne, da de flere gruppemedlemmer valgte ikke at møde op til forelæsninger i Diskret Matematik og til dels i Computerarkitektur. Når der har været samarbejde, var dette som regel i programmeringskurset, hvor samarbejdet bestod i at vi diskuterede nogle af opgaverne og hvordan vi havde tænkt os at løse dem, hvorefter selve programmeringen foregik individuelt. Desuden har vi arbejdet sammen omkring nogle af de udfordringsopgaver i computerarkitekturkurset, da det var mere naturligt at løse disse opgaver som gruppe.

Fordi vi har haft mindre samarbejde i forhold til kurser, har vi generelt også haft meget lidt diskussion omkring det, der blev forelæst om, hvilket har haft en betydning for hvordan vi hver især har prøvet at opnå forståelse for det viden fra

kurserne og for den teori, som vi har brugt i forbindelse med projektet. Resultatet var at vi hovedsageligt spurgte hjælpelærerne, vejleder eller brugte online ressourcer, hvis der var noget teori som vi havde svært ved, fremfor at komme frem til noget internt i gruppen.

Samarbejde i klyngen

Det var hensigten at vi skulle samarbejde på kryds af fagområderne og at vi derved kunne drage fagligt nytte af det gennem P2 projektet. I klyngesamarbejdet mødtes vi et par gange for at dele viden. I starten af projektet aftalte vi hvad vi ønskede at bruge klyngegruppen til og blandt grupperne blev vi enige om at matematik-gruppen ville tage sig af at udvikle den matematiske baggrund for den grafbaserede metode. Vi, sammen med softwaregruppen, besluttede os for at dele vores viden og perspektiver omkring implementeringen af en sådan algoritme. Meget af dette skete primært ved aftale af møder i klyngegruppen. Derudover blev der oprettet en Facebookgruppe, så vi kunne komme i kontakt med hinanden.

Grundet det, at matematikgruppen først havde en implementerbar algoritme relativt langt inde i projektperioden, tog vi og softwaregruppen to forskellige indgangsvinkler til vores implementeringer. Dermed er vores endelige implementeringer ikke ens og dermed heller ikke kompatible med hinanden. Vi fik dog delt vores viden omkring visse muligheder i C# og .NET som vi kunne drage nytte af. Softwaregruppen gik i gang med at opsætte en programstruktur og udviklede en LSB metode, som placeholder for algoritmen som matematikgruppen kom med senere. Vi tog en anden indgangsvinkel, idet vi begyndte at undersøge teorien bag den grafbaserede steganografi og derfra udtænkte en struktur til denne. Vi endte med at have en implementation, som overordnet lignede de algoritmer, som matematikgruppen havde udarbejdet. Fordi vi modtog deres algoritmer meget sent i projektet, kunne vi ikke præcist implementere deres algoritmer og dermed fik vi ikke gavn af dem.

Vurdering

Samarbejde i gruppen

Gode erfaringer

Der har været mindre stress i forhold til arbejdsopgaven, i dette projekt, sammenlignet med P1. Vi har arbejdet professionelt og fik lavet rapporten og programmet indenfor den normerede tid, uden at gruppen gik ind i en periode med overarbejde.

Mandagsmøderne var gode til at skabe overblik over, hvor langt vi var kommet i projektet og hvilke opgaver, der skulle påbegyndes.

Dårlig erfaringer

Selve retterunderne var ikke effektive nok til at fange stavefejl og uønsket sætningskonstruktioner i arbejdsbladene, før de blev sendt til vejlederen.

Samarbejde med vejleder

Selve vejledersamarbejdet gavnede os meget, især fremadrettet. Vejledersamarbejdets største styrke har været, at al interaktion i projektforløbet har været så struktureret og tidsmæssig præcis, at det fik os til altid at have lavet nyt materiale til næste gang. Eventuelle glip i kommunikationen og samarbejdet skyldtes gruppen selv, oftest da vores arbejdsprocesser ikke var lige så strukturerede som vejlederens.

Gode erfaringer

Vejleders respons til arbejdsblade og spørgsmål har i løbet af hele projektet været meget tilfredsstillende, om end tidskrævende at efterkomme. Specielt så blev der foreslået mange ændringer, der ville gavne os meget i fremtiden, især ved nye projekter. Det kan siges at vejleder var meget påpasselig med at give lappeløsninger til rapporttekniske problemer, selv sent i projektet. Herunder er omstillingen fra rapportskrivning til skrivning af arbejdsblade det mest tilsvarende eksempel.

Dårlige erfaringer

De dårlige erfaringer vi har haft ved samarbejdet var hovedsageligt at vi fra vores side af ikke altid havde arbejdsblad, der var vel rettet igennem, hvilket gjorde at vi ikke altid alle kunne stå inde for de indsendte arbejdsblades kvalitet. Dette gjorde nogle møder med vejleder mindre effektive.

Projektplanlægning og projektstyring

Vurderingen af projektplanlægning og projektstyring kan deles op i en række gode og dårlige erfaringer, i forhold til hvordan det er blevet valgt at planlægge og styre projektet på. De gode erfaringer og de dårlige erfaringer er de følgende:

Gode erfaringer

Det har været en stor fordel at bruge Slack og Trello, samt vores arbejdsplan som planlægnings- og styrings-værktøjer i dette projekt. Grunden til dette, er fordi det har været med til at give et godt overblik over hvad der skulle laves gennem hele projektforløbet og har været med til at sikre, at vi har fået udarbejdet vores produkt, bestående af en rapport og et program, på en ordenlig og tilrettelagt måde i den forstand, at de forskellige opgaver er blevet igangsat i den rigtige rækkefølge og har derfor formindsket mængden af gange hvor projektet har været gået i stå.

Dårlige erfaringer

En ulempe ved dette projektarbejde har været at planlægningsværktøjet Trello ikke var blevet opdateret så ofte, som vi har ønsket at det skulle være. Det har derfor til tider gennem projektforløbet stået uklart hvad enkelte gruppemedlemmer var i gang med at arbejde på, specielt hvis de var blevet ramt af sygdom og ikke kunne kontaktes over Slack.

En anden ulempe under projektarbejdet har været at den lavede arbejdsplan ikke er blevet opdateret løbende gennem projektforløbet. Dette har nogle gange medvirket til at overblikket over projektet kunne mindskes en smule hvor det ellers kunne have været undgået hvis den blev opdateret. Hvis dette var gjort kunne det også have medvirket til at der havde været mindre pres da projektet nærmede sig afleveringsfristen.

Læreprocessen

Gode erfaringer

Det har været en god erfaring, at der har været fleksibilitet i forhold til hvornår man kunne bruge tid på kurserne. Det har ikke været et krav, i forhold til gruppekontrakten, at skulle deltage i forelæsninger, hvilket har givet mulighed for alle i gruppen at bruge den nødvendige tid, når det passede dem bedst.

Dårlige erfaringer

Det har været en dårlig erfaring, at der har været meget lidt diskussion af emner og opgaver fra tre kurser igennem projektforløbet og at der har været meget lidt fælles opgaveløsning, som ellers kan give meget til selve indlæringen.

Samarbejde i klyngen

Gode erfaringer

Klyngesamarbejdet har været en positiv oplevelse i forhold til udarbejdelse af projektet og det godt kan gentages. Vi dragede stor nytte af vidensdelingen, samt de erfaringer som de andre grupper havde.

Dårlige erfaringer

På visse punkter har vi ikke benyttet klyngegruppen optimalt, idet vi mener at det kunne have været brugt mere i det daglige projektarbejde.

Analyse

Samarbejde i gruppen

At samarbejdet i gruppen gik godt kan skyldes, at vi var i stand til at uddele arbejdet og holde overblikket over projektet, selv om vi ikke valgte en leder til at styre dette. Det ugentlige møde har hjulpet med at holde medlemmerne opdateret på projektet, men vi har også delt viden i løbet af de enkelte uger. Denne vidensdeling har også hjulpet med at holde det enkelte medlem opdateret. Derfor opstod der ikke en stor klump af ufærdigt arbejde, der skulle håndteres til slut i projektet.

At retterunderne ikke kunne fange stavefejl, skyldes sandsynligvis flere faktorer. En faktor kunne være, at retterunderne ikke udnyttede den afsatte tid effektivt, da det er meget tidskrævende for hele gruppen at gennemgå alle arbejdsbladene for stavefejl. En anden faktor kunne være, at disse retterunder også hjalp med vidensdeling i gruppen. Hvis det enkelte medlem ikke skulle bruge energi på at forstå, men også rette, var det nemmere at finde stavefejl, samt dårlige sætningskonstruktioner gennem processen.

Samarbejde med vejleder

Samarbejdet med vejleder har især været godt, fordi vi har bestræbt os på at efterkomme vejleders kritik og foreslag til ændringer så hurtigt som muligt. Derudover har det gavnet samarbejdet at vejlederen har været så specifik og konsistent med den respons og vejledningstype, som han i starten havde sagt han ville give.

De få problemer, der har været med vejlederesamarbejdet, hovedsageligt tidsproblemer ved aflevering af arbejdsblade om Fredagen, som hele gruppen kunne stå inde for, kunne vi have løst på flere måder. En måde ville være at vi i større grad kunne have udnyttet vores mulighed for at flytte afleveringsdagen en smule frem/tilbage. En anden måde, måske mere hensigtsmæssig, ville være at have

været mere streng og opmærksom på en egen tidsplan. Vi kunne på denne måde have været mere effektive til at finde eventuelle tidsproblemer tidligt. Hvilket ville gøre det muligt for os at løse disse lettere. Dette kunne evt. gøres ved at delegere flere gruppemedlemmer til de enkelte opgaver, eller ved at udskyde enkelte dele til den følgende uge. Herefter kunne vi så sætte de gruppemedlemmer, der var ved med at arbejde på disse, i gang med at rette de andre gruppemedlemmers materiale. Således kunne vi sende nogle få ordentlige arbejdsblade i stedet for at gruppemedlemmerne haster med at skrive deres afsnit færdige til sidste øjeblik før aflevering.

Projektplanlægning og projektstyring

Projektets resultat, bestående af en rapport samt et program, nåede at blive færdiggjort inden afleveringsfristen for projektet. En af de faktorer, der har hjulpet til at dette kunne opnås har været planlægningsværktøjerne, da disse har været med til at give et overblik over projektet som en helhed. Dette betyder selvfølgelig ikke at dette område ikke kan forbedres i forhold til opdatering af arbejdsplan mm.

En grund til at arbejdsplanen ikke er blevet opdateret så meget, som det kunne være ønsket, kunne være fordi at der i gruppen ikke har været nogen gruppeleder. I dette projekt har alle været med til at danne sig et overblik og alle har været med til at planlægge projektarbejdet, hvilket har gjort at alle har dannet sig et fælles overblik og ikke har tænkt videre over arbejdsplanen efter den er blevet lavet. En anden grund til mangel på opdatering af arbejdsplanen kunne være, at da det blev relevant at opdatere arbejdsplanen var der for stort pres på projektet til at vi i gruppen kunne sætte tid af til at opdatere den. Arbejdsplanen blev derfor givet en lavere prioritet, og vi mistede som et resultat af dette en smule overblik over projektet.

Læreprocessen

Grunden til, at vi ikke har haft fælles opgaveløsning i kurserne, kan være fordi vi ikke gjorde det klart i starten, hvad vi forventede af hinanden, i forhold til deltagelse i kurserne. Nogle mener, at man tager til enhver forelæsning og at man efterfølgende regner opgaver. Hvorimod andre har foretrukket at arbejde på andre tidspunkter. Nogle i gruppen havde en gruppekontrakt i P1, hvori man forventede at alle mødte til forelæsninger, hvor det havde en positiv effekt på gruppens arbejde.

Samarbejde i klyngen

Grunden til at vi ikke fik benyttet klyngen optimalt, har at gøre med de få erfaringer vi har i det almindelige gruppearbejde og derfor brugte vi meget tid på, at få

vores eget gruppearbejde til at fungere, hvilket overlod meget lidt tid til at få koordineret klyngesamarbejdet. Det er vores indtryk at dette også er problematikken for de andre grupper.

Syntese

Samarbejde i gruppen

Vi har gjort nogle overvejelser om, hvad vi vil fortsætte med at gøre på næste projekt og hvad vi vil stoppe med.

De ugentlige møder har været hyppige og detaljerede nok til at sikre fokus i gruppen og projektet. Vi vil derfor fortsætte med den samme hyppighed af møder på P3.

Retterunderne skal dog enten restruktureres eller erstattes af en anden metode til at rette arbejdsbladene på, da den har været utilstrækkelig til vores eget, samt vejlederens, behov.

Samarbejde med vejleder

Et godt råd til næste projekt er at fastlægge en struktur for vejlederkommunikation tidligt i projektet, så alle får overblik over hvordan tiden i arbejdsugerne bør udnyttes og hvordan vi skal forberede os til møderne.

Derudover vil vi til start af næste projekt ikke forsøge at begrænse vejlederens indflydelse på projektet, som da vi forsøgte blot at få produktorienteret vejledning.

Fremadrettet vil brug af en tidsplan sandsynligvis gøre tidsproblemer vedr. deadline til aflevering af arbejdsblade, til vejleder lettere at håndtere. Derimod så vil vi på grund af den enorme mængde afsnit, som vi i projektet har lært er unødvendige, sandsynligvis være bedre til på forhånd at vurdere hvad vi bør bruge vores tid på at skrive, og dermed vejleders tid på at rette.

Projektplanlægning og projektstyring

Et godt råd til fremtidige semesterprojekter for projektplanlægning og projektstyring ville være at blive bedre til at holde projektplanlægnings- og projektstyringsværktøjer opdateret for at bevare det fulde overblik over projektet og sikrer endnu mindre pres til sidst i projektforløbet, hvilket kunne resulterer i et endnu bedre produkt, da der ville være mere tid til forbedring i slutningen af projektet, hvis der rammes færre stopklodser, som følger af god og opdateret planlægning.

Læreprocessen

Til næste projekt skal vi have en konkret aftale på plads, i forhold til opgaveløsning i kurser. Dette skal tages op når gruppekontrakten laves, og det vil også være et godt tidspunkt at tale om de forventninger man har til hinanden, sådan at det står klart fra starten, hvordan hverdagene kommer til at foregå.

Samarbejde i klyngen

Et godt råd til fremtidige semesterprojekter, i forhold til samarbejde i klynger, ville være at forbedre samarbejdet via en fælles tidsplan for projektforløbet, for at få et indblik i de andre gruppers planer for projektet og klarlægge gruppernes fokuspunkter, for at klyngearbejdet kan supplere de individuelle gruppers arbejde mere. Det ville endvidere også være en god idé stadig at have en kommunikationskanal, hvor det er muligt at kontakte de forskellige grupper igennem, da dette eliminerer tidspild i forhold til at komme i kontakt med de enkelte grupper. Kanalen kunne eksempelvis være en fælles opsætning via tjenester såsom Slack eller Trello, da disse giver flere muligheder for at opretholde en struktur og en kommunikation mellem grupperne.

Bibliografi

[1] Khan Academy. *Random vs Pseudorandom Number Generators*. URL: https://youtu.be/GtOt7EBNEwQ (sidst set 29.04.2016).

[2] National Security Agency. I: (2007). URL: https://en.wikipedia.org/wiki/PRISM_(surveillance_program) (sidst set 18.02.2016).

[3] Mads Bryde Andersens. *IT-retten*. Gjellerup, 2005. URL: http://www.it-retten.dk/it-retten.htm#LinkTarget_8476 (sidst set 22.02.2016).

[4] Steve Lloyd Carlisle Adams. "Core PKI Services: Authentication, Integrity, and Confidentiality". I: (1999). URL: https://technet.microsoft.com/en-us/library/cc700808.aspx (sidst set 02.03.2016).

[5] cbsnews.com. "Vodafone reveals scale of gov't snooping in phone networks". I: (2014). URL: http://www.cbsnews.com/news/vodafone-reveals-scale-of-govt-snooping-in-phone-networks/ (sidst set 10.03.2016).

[6] Electronic Privacy Information Center. "Restrictions on the import of cryptography". I: (2008). URL: http://gilc.org/crypto/crypto-survey.html (sidst set 17.02.2016).

[7] Facebook Help Center. *How can I make sure that my photos display in the highest possible quality?* URL: https://www.facebook.com/help/266520536764594.

[8] Casey Chan. *What Facebook Deals with Everyday: 2.7 Billion Likes, 300 Million Photos Uploaded and 500 Terabytes of Data*. URL: http://gizmodo.com/5937143/what-facebook-deals-with-everyday-27-billion-likes-300-million-photos-uploaded-and-500-terabytes-of-data.

[9] Codeplex. "C LZF Real-Time Compression Algorithm". I: (). URL: https://csharplzfcompression.codeplex.com/ (sidst set 14.04.2016).

[10] Lucian Constantin. "Overreliance on the NSA led to weak crypto standard, NIST advisers find". I: (Juli 15, 2014). URL: http://www.pcworld.com/article/2454380/overreliance-on-the-nsa-led-to-weak-crypto-standard-nist-advisers-find.html (sidst set 24.02.2016).

[11] Josh Constine. *Facebook Climbs To 1.59 Billion Users And Crushes Q4 Estimates With $5.8B Revenue*. URL: http://techcrunch.com/2016/01/27/facebook-earnings-q4-2015/.

[12] Tim Cook. "A Message to Our Customers". I: (Februar 16, 2016). URL: http://www.apple.com/customer-letter/ (sidst set 25.02.2016).

[13] Daniel L. Currie, Nab Little Creek og Cynthia E. Irvine. "Surmounting the Effects of Lossy Compression on Steganography". I: *In National Information System Security Conference*. 1996, s. 194–201.

[14] Nicolai Devantier. "Kryptering er et kæmpe-problem for danskerne: så alvorlig er situationen". I: (Juni 29, 2015). URL: http://www.computerworld.dk/art/234235/kryptering-er-et-kaempe-problem-for-danskerne-saa-alvorlig-er-situationen (sidst set 25.02.2016).

[15] Rowena Mason Ewan MacAskill Nicholas Watt. "UK intelligence agencies should keep mass surveillance powers, report says". I: (2015). URL: http://www.theguardian.com/world/2015/jun/11/uk-intelligence-agencies-should-keep-mass-surveillance-powers-report-gchq (sidst set 07.03.2016).

[16] Facebook. *Global Government Requests Report*. URL: https://www.facebook.com/about/government_requests.

[17] Facebook. *Information for Law Enforcement Authorities*. URL: https://www.facebook.com/safety/groups/law/guidelines/.

[18] Cyrus Farivar. *Paris police find phone with unencrypted SMS saying "Let's go, we're starting"*. URL: http://arstechnica.com/tech-policy/2015/11/paris-police-find-phone-with-unencrypted-sms-saying-lets-go-were-starting/.

[19] Niels Ferguson. *Cryptography Engineering: Design Principles and Practical Applications*. First Edition. Indianapolis: Wiley Publishing, Inc., 2010, s. 139, 140.

[20] Fileformat. "JPEG Compression". I: (). URL: http://www.fileformat.info/mirror/egff/ch09_06.htm.

[21] Matteo Fortini. "SSIS - Spread Spectrum Image Steganography". I: (). URL: http://www.lia.deis.unibo.it/Courses/RetiDiCalcolatori/Progetti98/Fortini/SSIS.html (sidst set 22.03.2016).

[22] Alejandro C. Frery og Talita Perciano. *Introduction to Image Processing Using R: Learning by Examples*. Springer Publishing Company, Incorporated, 2013, s. 27, 28. ISBN: 1447149491, 9781447149491.

[23] Jiri Fridrich. "A New Steganographic Method for Palette-Based Images". I: *in Proceedings of the IST PICS conference*. 1999, s. 285–289. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.4399&rep=rep1&type=pdf.

[24] M. R. Garey og David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN: 0-7167-1044-7.

[25] Juliette Garside. "Vodafone reveals existence of secret wires that allow state surveillance". I: (2014). URL: http://www.theguardian.com/business/2014/jun/06/vodafone-reveals-secret-wires-allowing-state-surveillance (sidst set 10.03.2016).

[26] F. Garzia. *Handbook of Communications Security*. WIT Press / Computational Mechanics, 2013, s. 237. ISBN: 1845647688, 9781845647681.

[27] Stefan Hetzl og Petra Mutzel. "A Graph–Theoretic Approach to Steganography". I: *Proceedings of the 9th IFIP TC-6 TC-11 International Conference on Communications and Multimedia Security*. CMS'05. Salzburg, Austria: Springer-Verlag, 2005, s. 119–128. ISBN: 3-540-28791-4, 978-3-540-28791-9. DOI: 10.1007/11552055_12. URL: http://dx.doi.org/10.1007/11552055_12.

[28] Httparchive.org. *Image requests by format*. URL: http://httparchive.org/interesting.php#imageformats.

[29] Günter Judd Deane B. Wyszecki. *Color in Business, Science and Industry*. Wiley Series in Pure and Applied Optics (3rd ed.) Wiley-Interscience, 1975. ISBN: 0-471-45212-2.

[30] Dia Kayyali. "FBI's "suicide letter"to Dr. Martin Luther King, Jr., and the Dangers of Unchecked Surveillance". I: (2014). URL: https://www.eff.org/deeplinks/2014/11/fbis-suicide-letter-dr-martin-luther-king-jr-and-dangers-unchecked-surveillance (sidst set 10.03.2016).

[31] *Lempel–Ziv–Welch*. URL: https://en.wikipedia.org/wiki/LempelâĂŞZivâĂŞWelch.

[32] Zephoria Digital Marketing. *The Top 20 Valueable Facebook Statistics*. URL: https://zephoria.com/top-15-valuable-facebook-statistics/.

[33] Sarah Marshall. "How journalists can enter the 'deep web' to stay secure". I: (2013). URL: https://www.journalism.co.uk/news/-wpe13-how-journalists-can-stay-secure-by-entering-the-deep-web-/s2/a554394/ (sidst set 07.03.2016).

[34] Microsoft. "Bitmap Class". I: (). URL: https://msdn.microsoft.com/en-us/library/system.drawing.bitmap(v=vs.110).aspx (sidst set 08.04.2016).

[35] Microsoft. "DeflateStream Class". I: (). URL: https://msdn.microsoft.com/en-us/library/system.io.compression.deflatestream\%28v=vs.110\%29.aspx (sidst set 14.04.2016).

[36] Microsoft. "GZipStream Class". I: (). URL: https://msdn.microsoft.com/en-us/library/system.io.compression.gzipstream\%28v=vs.110\%29.aspx (sidst set 14.04.2016).

[37] Microsoft. "How to: Compress and Extract Files". I: (). URL: https://msdn.microsoft.com/en-us/library/ms404280\%28v=vs.110\%29.aspx (sidst set 14.04.2016).

[38] Microsoft. *Integration testing*. URL: https://msdn.microsoft.com/en-us/library/aa292128(v=vs.71).aspx (sidst set 11.05.2016).

[39] Microsoft. *Random Class*. URL: https://msdn.microsoft.com/en-us/library/system.random.aspx (sidst set 29.04.2016).

[40] Microsoft. *System.Security.Cryptography namespace*. URL: https://msdn.microsoft.com/en-us/library/system.security.cryptography(v=vs.110).aspx (sidst set 05.04.2016).

[41] Microsoft. "Types of Bitmaps". I: (). URL: https://msdn.microsoft.com/en-us/library/at62haz6(v=vs.110).aspx (sidst set 08.04.2016).

[42] Microsoft. *Unit testing*. URL: https://msdn.microsoft.com/en-us/library/aa292197(v=vs.71).aspx (sidst set 12.05.2016).

[43] Microsoft. "Windows Forms Overview". I: (). URL: https://msdn.microsoft.com/en-us/library/8bxxy49h\%28v=vs.110\%29.aspx (sidst set 21.04.2016).

[44] Microsoft. "ZipFile Class". I: (). URL: https://msdn.microsoft.com/en-us/library/system.io.compression.zipfile\%28v=vs.110\%29.aspx (sidst set 16.04.2016).

[45] Ellen Nakashima. "Apple vows to resist FBI demand to crack iPhone linked to San Bernardino attacks". I: (Februar 17, 2016). URL: https://www.washingtonpost.com/world/national-security/us-wants-apple-to-help-unlock-iphone-used-by-san-bernardino-shooter/2016/02/16/69b903ee-d4d9-11e5-9823-02b905009f99_story.html (sidst set 24.02.2016).

[46] NIST. *Advanced encryption standard algorithm validation list*. URL: http://csrc.nist.gov/groups/STM/cavp/documents/aes/aesval.html (sidst set 05.04.2016).

[47] OrdNet.dk. *aktivist - Den Danske Ordbog*. URL: http://ordnet.dk/ddo/ordbog?query=aktivist.

[48] Roy Osherove. *Naming standards for unit tests*. URL: http://osherove.com/blog/2005/4/3/naming-standards-for-unit-tests.html (sidst set 12.05.2016).

[49] *Over 35 forskellige programmer, som anvender steganografiske metoder til at gemme beskeder*. URL: http://listoffreeware.com/list-of-best-free-steganography-software-for-windows/.

[50] PET. *Hvorfor forebygge?* URL: https://goo.gl/LeNOHA.

[51] PET. *Om PET*. URL: https://goo.gl/Ew14ez.

[52] *Program som benytter steganografiske metoder til at gemme beskeder i billeder*. URL: http://www.quickcrypto.com/free-steganography-software.html.

[53] Danmarks Radio. "Overblik: Sådan bliver Danmark styret". I: (Oktober 29, 2013). URL: https://www.dr.dk/ligetil/indland/overblik-saadan-bliver-danmark-styret (sidst set 03.03.2016).

[54] Redaktionen. "Parasitisme". I: (2016). URL: http://denstoredanske.dk/Natur_og_miljÃÿ/Zoologi/Parasitologi/parasitisme (sidst set 03.03.2016).

[55] Belgiens regering. "Loi du 28 novembre 2000 relative à la criminalité informatique." I: (2000). URL: http://cwisdb.kuleuven.be/pisa/fr/jur/infocrimewet.htm#Art.9 (sidst set 19.02.2016).

[56] Den Australske regering. "Cybercrime Bill". I: (2001). URL: https://www.comlaw.gov.au/Details/C2004B00915/Download (sidst set 17.02.2016).

[57] Englands regering. "Regulation of Investigatory Powers Act". I: (2000). URL: http://www.legislation.gov.uk/ukpga/2000/23/contents (sidst set 18.02.2016).

[58] Finlands regering. "Pakkokeinolaki 806". I: (Udgivelsesår: 2011). URL: http://www.finlex.fi/fi/laki/ajantasa/2011/20110806#a806-2011 (sidst set 18.02.2016).

[59] Frankrigs regering. I: (2001). URL: https://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000000222052 (sidst set 19.02.2016).

[60] Hollandske regering. "Wetboek van Strafvordering, Artikel 125k". I: (2015). URL: http://wetten.overheid.nl/BWBR0001903/2015-01-01#BoekEerste_TiteldeelIV_AfdelingZevende_Artikel125k (sidst set 18.02.2016).

[61] USA regering. "P.A.T.R.I.O.T. Act". I: (2001). URL: https://en.wikipedia.org/wiki/Patriot_Act (sidst set 17.02.2016).

[62] USA regering. "P.A.T.R.I.O.T. Act". I: (2001). URL: https://www.gpo.gov/fdsys/pkg/PLAW-107publ56/pdf/PLAW-107publ56.pdf (sidst set 20.02.2016).

[63] Tom Roeder. *Asymmetric-Key Cryptography*. URL: http://www.cs.cornell.edu/courses/cs5430/2013sp/TL04.asymmetric.html.

[64] Phillip Rogaway. *The Moral Character of Cryptographic Work*. Tek. rap. Department of Computer Science, University of California, Davis, USA, December 12, 2015, s. 25, 27. URL: http://web.cs.ucdavis.edu/~rogaway/papers/moral-fn.pdf.

[65] Rosettacode.org. *Subtractive Generator*. URL: http://rosettacode.org/wiki/Subtractive_generator (sidst set 29.04.2016).

[66] Margaret Rouse. "image compression". I: (). URL: http://whatis.techtarget.com/definition/image-compression.

[67] Margaret Rouse. "lossless and lossy compression". I: (). URL: http://whatis.techtarget.com/definition/lossless-and-lossy-compression (sidst set 06.2015).

[68] Det Etiske Råd. *Hvad er Etik?* 2013. URL: http://www.etiskraad.dk/etiske-temaer/introduktion-til-etik/undervisning-til-gymnasieskolen/hvad-er-etik (sidst set 21.02.2016).

[69] Julian Sanchez. "Wiretapping's true danger". I: (2008). URL: http://articles.latimes.com/2008/mar/16/opinion/op-sanchez16 (sidst set 10.03.2016).

[70] Cooper Smith. *Facebook Users Are Uploading 350 Million New Photos Each Day*. URL: http://www.businessinsider.com/facebook-350-million-photos-each-day-2013-9?IR=T.

[71] History.com Staff. "J. Edgar Hoover begins his legacy with the FBI". I: (2009). URL: http://www.history.com/this-day-in-history/j-edgar-hoover-begins-his-legacy-with-the-fbi (sidst set 10.03.2016).

[72] Stegano. *Network Steganography*. URL: http://stegano.net/network-steganography.html.

[73] "Steganography". I: (). URL: https://en.wikipedia.org/wiki/Steganography (sidst set 22.03.2016).

[74] Douglas R. Stinson. *Cryptography: Theory and Practice*. Third Edition. Boca Raton: Taylor og Francis group, LLC, 2006, s. 2.

[75] Mathias Koch Stræde. "Er Kryptering Problemet?" I: (November 16, 2015). URL: http://www.information.dk/552202 (sidst set 25.02.2016).

[76] Daniel Thomas. "Vodafone reveals scale of state surveillance demands on network". I: (2014). URL: http://www.ft.com/cms/s/0/d5eeae2a-ed44-11e3-abf3-00144feabdc0.html#axzz42UczWHeJ (sidst set 10.03.2016).

[77] Europæiske Union. "Wassenaar aftalen". I: (1998). URL: http://www.wassenaar.org/control-lists/ (sidst set 22.02.2016).

[78] Kurt Wagner. *Facebook Has a Quarter of a trillion User Photos*. URL: http://mashable.com/2013/09/16/facebook-photo-uploads/#OQdFw8ZU2Eqk.

[79] Shuozhong Wang Weiming Zhang Xinpeng Zhan. *Maximizing Steganographic Embedding Eciency by Combining Hamming Codes and Wet Paper Codes*. Tek. rap. School of Communication og Information Engineering, Shanghai University, Kina, Juli 30, 2014. URL: https://www.researchgate.net/publication/220722166_Maximizing_Steganographic_Embedding_Efficiency_by_Combining_Hamming_Codes_and_Wet_Paper_Codes.

[80] Wikipedia. "Palette (computing)". I: (). URL: https://en.wikipedia.org/wiki/Palette_(computing) (sidst set 14.04.2016).

[81] *Wikipedia - BMP*. URL: https://en.wikipedia.org/wiki/BMP_file_format.

[82] *Wikipedia - GIF*. URL: https://en.wikipedia.org/wiki/GIF.

[83] *Wikipedia - Hamming code*. URL: https://en.wikipedia.org/wiki/Hamming_code.

[84] *Wikipedia - JPEG*. URL: https://en.wikipedia.org/wiki/JPEG.

[85] *Wikipedia - PNG*. URL: https://en.wikipedia.org/wiki/Portable_Network_Graphics.

[86] *Wikipedia - TIFF*. URL: https://en.wikipedia.org/wiki/Tagged_Image_File_Format.

[87] Dr. Bill Young. *Foundations of computer security*. URL: https://www.cs.utexas.edu/users/byoung/cs361/lecture44.pdf (sidst set 23.03.2016).