Thesis no: BCS-2016-03

En jämförelse mellan teckenbaserade och grafiska lösenord

Fokuserad på användarvänlighet och säkerhet

Albin Norlin

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science: Computer Security. The thesis is equivalent to 10 weeks of full time studies.

Contact Information:
Author(s):
Albin Norlin
E-mail: alno13@student.bth.se

University advisor:
Dr. Veronica Sundstedt
DIKR – Department of Creative Technologies

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

ABSTRACT

Context. For users not to forget their password they tend to choose short passwords. That users tend to choose short passwords is a security risk that needs to be prevented. There exist research that suggest humans are better at recalling a picture compared to a text. If a graphical password in the type of an image were used in place of the character-based passwords, users would have the possibility to choose a more complex graphical pattern and still have good chances to remember the password.

Objectives. The project performed a comparison between character-based passwords and graphical password of the type DAS based on usability and security. For usability a comparison of the time it took to register and login between character-based and graphical password were conducted. Further a comparison on unsuccessful login attempts between the password variants were also conducted.

For the security aspect a comparison on the time it took to perform a successful bruteforce-attack on the password strings were conducted.

Methods. The first method used was a literature study on what previously has been done in the domain. The next method used was implementation of two programs that applied character-based and graphical password of type DAS. The implemented programs were used to perform an experiment with participants that returned values which made it possible to compare character-based and graphical password in the aspect of usability and security.

Results. The result from the experiment gave an indication that graphical passwords were better in both usability aspects. Similar to usability the security result also pointed towards the graphical passwords. The only value that indicated that character-based passwords were better was when the participants of the experiment were asked which password variant they preferred, in which they answered the character-based over graphical.

Conclusions. There is a possible indication from the experiments that the graphical passwords have both higher usability and security levels for the aspects which were compared. More experiments need to be done before it is possible to establish a conclusion on which password variant is the best for usability and security.

Keywords: Draw a secret, graphical passwords, usability, security.

ABSTRAKT

Kontext. För att inte glömma sitt lösenord så väljer användare korta lösenord. Att användare väljer korta lösenord är en säkerhetsrisk som behöver förebyggas. Det finns forskning som pekar på att det är lättare för människor att komma ihåg en bild jämfört med en text. Om grafiska lösenord i form av en bild används istället för teckenbaserade lösenord i form av text skulle användare kunna välja svårare lösenord i form av komplexa lösenordsmönster och samtidigt minska risken att glömma dem.

Mål. I projektet utförs en jämförelse mellan teckenbaserade och grafiska lösenord av typen DAS inom områdena användarvänlighet och säkerhet. Inom användarvänlighet så jämförs tiden det tar att registrera samt logga in med ett lösenord av respektive typ. Det jämfördes även hur många lyckade kontra misslyckade inloggningar som gjordes med respektive lösenordstyp.

För säkerhet så jämfördes tiden det tar att utföra en lyckad bruteforce-attack mot lösenordssträngarna.

Metoder. Den första metoden som användes var en litteraturstudie på vad som tidigare gjorts på området. Litteraturstudien uppföljdes av implementation av två program där det första tillämpade traditionella teckenbaserade lösenord och det andra tillämpade grafiska lösenord av typen DAS. De implementerade programmen användes i experiment med deltagare som gav värden att jämföra och analysera.

Resultat. Experimentresultat gav en indikation på att de grafiska lösenorden var bättre i båda aspekterna för användarvänlighet. Liknande indikation gavs för säkerhetsaspekten där även experimentet för de grafiska lösenorden returnerade bättre värde. Det enda värde som pekade för teckenbaserade lösenord var att deltagare av experimentet föredrog de teckenbaserade över grafiska lösenorden.

Slutsatser. Det ges en indikation för att grafiska lösenord kan vara mer användarvänliga samt säkrare jämfört med traditionella teckenbaserade lösenord för de aspekter som jämfördes. Det behöver göras fler och större experiment innan det går att fastslå en slutsats om vilken lösenordstyp som är bäst för användarvänlighet samt säkerhet.

Nyckelord: Draw a secret, grafiska lösenord, användarvänlighet, säkerhet.

INNEHÅLLSFÖRTECKNING

ORDLISTA

- DAS: Draw a secret, grafisk lösenordsvariant.
- Bruteforce-attack: Attack mot lösenord där angripare testar alla möjliga lösenordsmöjligheter tills rätt lösenord hittas.
- BDAS: Background draw a secret, DAS-variant där en bakgrundsbild används.
- QDAS: Qualitative draw a secret, DAS-variant där ett rutnät används för att registrera de ritade linjernas riktning.
- Pattern lock: Grafisk lösenordsvariant.
- Smartphone: Modern mobiltelefon.
- Grid selection: Teknik för att öka lösenordsspann för draw a secret lösenord.
- Snowballing: Metod för att hitta artiklar till litteraturstudie.
- IEEE: Databas innehållande bland annat forskningsartiklar.
- ACM: Databas innehållande bland annat forskningsartiklar.
- PHP: Serverbaserad scriptkod.
- HTML: Språk för webblayout.
- JavaScript: Klientbaserat script.
- C: Programmeringsspråk.
- C#: Programmeringsspråk, uppgradering av C.
- C-familjen: Programmeringsspråk som tillhör C eller utvecklingar, exempelvis C, C++ och C#.
- SQL: Programspråk för att arbeta med relationsdatabaser.
- SQL-injection: Attack där svagheter i kod utnyttjas för att manipulera SQL-anrop till att göra något annat.
- Hash-algoritm: Metod för att utföra en envägskryptering.
- SHA1: Secure hash algorithm 1, variant av hash-algoritm.
- SHA2: Secure hash algorithm 2, variant av hash-algoritm.
- MD5: Message-digest algorithm 5, variant av hash-algoritm.
- John the ripper: Bruteforce-program.
- Linux Kali: Operativsystem.
- Shoulder surfing: Teknik då angripare tittar och memorerar när en användare skriver in lösenord.

1 INTRODUKTION

Precis som lås på dörren så har datorvärlden metoder för att hålla oinbjudna aktörer ute, exempelvis lösenord. Lås på dörren har svagheter som angripare kan utnyttja och likväl så har även lösenord det. Användare har lätt för att glömma lösenord och för att motverka detta så väljer de ofta korta lösenord och/eller lösenord som har någon personlig koppling så som namn på husdjur som är lättare att komma ihåg [1]. Detta är en säkerhetsrisk på så sätt att när lösenordet blir kortare så blir antalet möjliga lösenordskombinationer också färre. Det gör att det blir lättare för en angripare att manuellt eller med hjälp av datorkraft testa alla möjliga lösenordskombinationer tills det rätta lösenordet hittas.
Problemet ligger i att användare har svårt att komma ihåg långa lösenord. Ett lösningsalternativ skulle kunna vara att byta ut de traditionella teckenbaserade lösenorden mot grafiska lösenord. Det finns nämligen forskning som visar att människor har lättare att minnas en bild jämfört med en text [2] [3] [4]. Om ett grafiskt lösenord i form av en bild är lättare för en användare att minnas så ger det möjlighet för användaren att välja ett längre lösenord i form av ett komplext mönster.

1.1 Projektbeskrivning

I det här projektet så kommer grafiska lösenord av typen ”draw a secret”, förkortat som DAS att användas. Den största utmaningen kommer vara att det är orimligt att kräva att en användare ska kunna återupprepa ett mönster ner på pixel precision. Där måste alltså tillåtas mindre felmarginaler. Utmaningen är att skriva en algoritm som tittar på vad, samt hur användaren ritade sitt mönster vid inloggning och jämför det mot hur det ritades in vid registrering för att sedan avgöra om det är likt till den grad att dem beviljas inloggning. En svår fråga att besvara är att avgöra var gränsen går för vad som är ett matchande mönster och vad som inte är. Konfigureras algoritmen att tillåta stora skillnader så blir det enkelt för användaren att logga in, men det blir även lätt för angripare att testa sig fram då programmet tillåter väldigt många närliggande lösenord. Om algoritmen istället ställs in att endast tillåta mycket små skillnader så blir det svårare för angripare att gissa sig fram till rätt lösenord, men det blir även svårare för de riktiga användarna att logga in. Det här är den klassiska frågan och balansgången mellan säkerhet och användarvänlighet.

Det finns forskning som pekar på att användare fortsätter att välja lätta och förutsägbara lösenord även för grafiska lösenord [5]. Det medför att lösenorden förblir sårbara mot angrepp. En teknik som angripare kan använda för att knäcka enkla grafiska lösenord är genom att använda en fil med vanliga grafiska lösenord och låta en dator testa om något av lösenorden matchar [6]. En teknik för att motverka detta till viss grad är att inte bara jämföra mönster, utan även jämföra hur användaren ritade in mönstret.
Detta ger ett nytt djup i säkerheten och det blir svårare för angripare när det inte räcker med att bara mönstret stämmer, utan att de även måste ritas in med samma teknik. Exempelvis, om man bortser från storlek så ser en kvadrat alltid ut på ett sätt, men det finns däremot väldigt många olika sätt att rita den.
För att genomföra projektet så är det uppdelat i fyra mindre moment:

- Första momentet kommer bestå av en litteraturstudie som kommer vara en förstudie på vad som tidigare har gjorts på närliggande område.
- Andra momentet kommer vara att implementera två program, där det ena tillämpar grafiska lösenord av typen DAS och det andra använder sig av traditionella teckenbaserade lösenord.
- De implementerade programmen kommer att användas under tredje momentet som är experimentdelen där användare får testa både teckenbaserade och DAS-lösenord.

- Fjärde och sista delen är huvudsakligen att göra analyser, jämförelser och dra slutsatser från experimentresultat.

De aspekter som kommer att jämföras är säkerhet och användarvänlighet. Säkerheten kommer avgränsas till att endast testa säkerheten när det gäller bruteforce-attacker på lösenordssträngen (en bruteforce-attack är när en dator testar alla möjliga lösenordskombinationer tills den hittar en match). Användarvänligheten avgränsas till hur lång tid det tar för en användare att registrera ett konto samt logga in med respektive lösenordstyp, men även antal lyckade kontra misslyckade inloggningsförsök.

1.2 Mål och frågeställningar

Målet med projektet är att undersöka, jämföra och svara på frågeställningar kring användarvänlighet samt säkerhet mellan traditionella teckenbaserade lösenord och grafiska lösenord av typen DAS. Det skulle vara möjligt att endast testa en av aspekterna. Anledningen till att båda testas är att dem är beroende av varandra. Det skulle vara simpelt att implementera grafiska lösenord som är väldigt användarvänliga, men de skulle vara väldigt osäkra då de skulle tillåta väldigt stora skillnader när mönster återupprepas vid inloggning. På samma sätt skulle det vara möjligt att implementera mycket säkra grafiska lösenord, men det skulle vara mycket svårt för en användare att logga in och användarvänligheten skulle vara låg. Därför ger det en bättre koppling till de problem som uppstår vid en verklig inloggningsfunktion och den svåra balansgången och avvägningen mellan säkerhet och användarvänlighet när båda testas.

För användarvänlighet är frågeställningen följande:

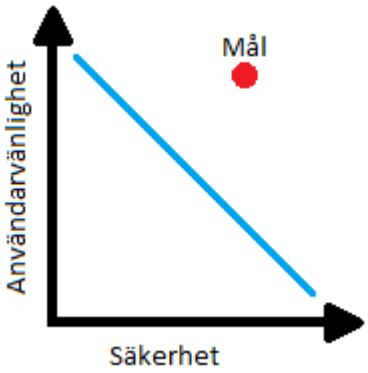
- Hur förhåller sig användarvänligheten mellan teckenbaserade och grafiska lösenord av typen DAS baserat på aspekterna, tid för registrering och inloggning samt antal lyckade kontra misslyckade inloggningsförsök?

För säkerhet så lyder frågeställningen:

- Hur förhåller sig säkerheten mellan teckenbaserade och grafiska lösenord av typen DAS mot bruteforce-attacker på lösenordssträngen? (Grafiska lösenord genererar även lösenordssträngar i form av olika värde från de jämförelseaspekter som testas).

2 RELATERAT ARBETE

För tillfället så är teckenbaserade lösenord klart störst på marknaden för lösenord. Grafiska lösenord är på väg då det finns en hel del forskning tillgänglig och det kommer kontinuerligt ny. Men än så länge har grafiska lösenord inte slagit igenom. Det grafiska lösenord som det går bäst för är pattern lock [7] som används i så kallade smartphones och går ut på att användaren ska rita ett mönster mellan fasta punkter. En av de stora anledningarna till att grafiska lösenord inte slagit igenom är att balansgången mellan säkerhet och användarvänlighet inte är bra nog [8]. För att grafiska lösenord ska slå igenom så behövs höga nivåer för både säkerheten och användarvänligheten, men idag finns ingen teknik eller metod som ger höga värde för båda aspekterna.



Figur 1: Linjediagrammet illustrerar möjlig balansgång mellan säkerhet och användarvänlighet samt önskat resultat.

Det finns flera olika varianter av grafiska lösenord [8]. Den variant som använts i detta projekt är en ren DAS som tillåter användaren att rita fritt. DAS går ut på att användaren ska återupprepa ett mönster som tidigare registrerats. Exempelvis, om användaren väljer att rita en cirkel vid registrering, så behöver en liknande cirkel att återupprepas för att inloggning ska lyckas. En direkt styrka med DAS är att det är oberoende av språk och kan därför köras över system med olika språkinställningar. Självklart finns det flera olika grupperingar och varianter av DAS-lösenord. En variant är background-DAS (BDAS), där läggs en bakgrundsbild till i det område där användaren ska rita sitt mönster. Det har visat att användare tar hjälp av detaljer i bilden för att bättre kunna återupprepa och minnas sitt grafiska lösenord [6] [9] [10].
En annan variant är qualitative-DAS (QDAS) [9] [10] [11]. QDAS går ut på att användaren får rita fritt i en ruta innehållande ett rutnät. Rutnätet används av programmet för att registrera när en användare rör musen mellan rutorna och över linjerna i rutnätet och med hjälp av detta kan programmet avgöra vilka riktningar som varje linje har.

En sak som forskare försöker göra är att utöka lösenordsspannet för DAS-lösenord, med andra ord öka antalet möjliga lösenord. Det är för att göra det svårare för angripare att gissa/testa sig fram till rätt lösenord, vilket det blir när antalet möjliga lösenord att testa ökar. En teknik som testas för att öka lösenordsspannet kallas grid selection [14]. Det går ut på att användaren börjar med att zooma in på en väldigt stor ruta tills den når en mindre storlek som ungefär matchar storleken på de rutor som ofta används för grafiska lösenord. Med andra ord så behöver användaren även zooma in på rätt ställe innan lösenordet ritas in. I och med att den totala rutan blir större och användaren kan rita in sitt lösenord på flera ställen så ökar även lösenordsspannet och säkerheten.

3 METOD

I denna del så beskrivs de metoder som användes under projektet. Det förklaras även varför just de metoder som användes valdes.

3.1 Litteraturstudie

Projektet inleds med en litteraturstudie för att undersöka vad som tidigare gjorts på området. Det är viktigt att få en inblick i problem och svårigheter med DAS-lösenord som andra forskare stött på tidigare samt förstå hur de har gått till väga för att hitta eventuella lösningar på problemen.
För att genomföra litteraturstudien kommer en metod vid namn snowballing [12] att användas. För att förklara den kortfattat så inleds den med att bygga ihop en söksträng som returnerar matchande artiklar för det område som litteraturstudien ska göras på. Nästa moment är att gå igenom de artiklar som sökningen returnerade och välja ut de som anses vara relevanta. Från de artiklarna som valdes ut studeras dess referenser och citeringar för att se om där finns några relevanta artiklar. Om där hittades några nya relevanta artiklar bland referenser och/eller citeringar så fortsätter man att titta på deras referenser och citeringar. Detta pågår tills inga nya relevanta artiklar hittas bland referenser och citeringar .
De databaser som används för att söka litteratur i är IEEE och ACM. Anledningen till att dessa databaser valdes är att dem innehåller mycket artiklar på området kring grafiska lösenord.

3.2 Implementation

Det valdes att implementera två program där det ena tillämpar teckenbaserade lösenord och de andra tillämpar grafiska lösenord av typen DAS.
Valet till att implementera två program som tillämpar teckenbaserade respektive grafiska lösenord är för att möjliggöra experiment med respektive. Experimenten kommer ge variabler och värden som kommer kunna jämföras, analyseras och slutligen användas för att svara på projektets frågeställningar.

Det finns flera val för miljö att utveckla teckenbaserade samt DAS-lösenord. En möjlighet var att använda en webbmiljö och språken PHP, HTML och JavaScript.
De andra alternativet som också valdes för projektet var att implementera programmet i ett av C-språken. Av C-språken så valdes C# och utvecklingsmiljön Visual Studio 2010/2015 [20]. Det skulle fungera att implementera programmen i båda alternativen men de senare alternativet valdes på grund av större förkunskaper i C-språken jämfört med webb. Större delen av DAS-lösenorden består av algoritmer vilket språken i C-familjen är passande för.

3.3 Testning av implementation

För att försäkra att de implementerade programmen fungerade väl innan experimenten så testades de av programmeraren under utvecklingens gång.
Det utfördes även pilottest av externa parter som inte visste hur programmen eller algoritmer i bakgrunden fungerade i detaljnivå. Deltagarna för pilottestet bestod av kollegor och vänner. Sammanlagt var det åtta personer i ålder 21 till 38 år som deltog i pilottest som utfördes fyra gånger. Testet gick ut på att dem skulle rita förutbestämda mönster och/eller rita fritt för att möjliggöra för programmeraren att se hur algoritmen fungerade när extern part använde den.

3.4 Experiment

Experimentet kommer ge variabler för både användarvänlighet och säkerhet. I den här delen så beskrivs vilka samt varför dessa variabler och värde jämförs.

3.4.1 Användarvänlighet

När en användare vill logga in på ett konto så vill de inte lägga onödig tid vid inloggning utan nå sitt mål direkt. Därför kommer det jämföras hur lång tid det tar för en användare att registrera ett konto samt logga in. Detta kommer göras med både teckenbaserade samt de grafiska lösenorden. Ytterligare kommer även att antalet misslyckade inloggningsförsök att jämföras. Värden från de teckenbaserade kommer jämföras med grafiska för att se hur de förhåller sig till varandra.

3.4.2 Säkerhet

Vanligtvis vid registrering så sparas användarnamn och lösenord i en databas. En säkerhetsrisk är om det finns sårbarheter som möjliggör att en angripare kan utföra en SQL-injection, vilket är ett av de största hoten inom dataintrång [15]. Det skulle då vara möjligt för en angripare att få tillgång till tabellen i databasen innehållande alla användarnamn och lösenord vilket medför tillgång till alla konton som är registrerade i databasen.
En teknik för att motverka skada om en angripare skulle få tag på alla användarnamn och lösenord via en SQL-injection är att hasha lösenordssträngarna.

Att hasha ett lösenord skulle kort kunna beskrivas som envägskryptering. Med envägs menas med att det går att kryptera, men inte att dekryptera. När en användare registrerar ett konto så hashas lösenordet och läggs in i databasen. När en användare loggar in så hashas de lösenord de försöker logga in med och jämförs med den lösenordshash som skapades vid registrering. Är det en match så är det samma lösenord och användaren tillåts inloggning. Om en angripare skulle få tag på inloggningstabellen i en databas med hashade lösenord så får de alltså inte lösenorden i klartext, utan istället en lista med hashar som inte går att dekryptera. Skulle en angripare få tag i en databas med hashade lösenord så är det inte oanvändbart för angriparen. Precis som vid inloggning så kan angriparen testa att hasha alla möjliga lösenord och jämföra de med hasharna från databasen. Angriparen använder alltså en bruteforce-attack för att knäcka de hashade lösenorden.

Det som kommer jämföras som säkerhetsaspekt i detta projekt är hur lång tid det skulle ta för en angripare att genomföra en lyckad bruteforce-attack på lösenordssträngar hashade med algoritmen SHA1 (secure hash algorithm 1) från både teckenbaserade och grafiska lösenord. Det finns ingen speciell anledning till att algoritmen SHA1 används och det skulle gå likväl att använda exempelvis MD5 (message-digest algorithm 5) eller SHA2 (secure hash algorithm 2).

4 IMPLEMENTATION

I den här delen så förklaras tillvägagångssätt och implementation på en djupare teknisk nivå. Det kommer i stor del inrikta sig på de grafiska lösenorden på grund av att dem är mer komplexa jämfört med de teckenbaserade lösenorden.

4.1 Val av jämförelseaspekter

För traditionella teckenbaserade lösenord så jämförs endast de tecken som skrivs in i lösenordsrutan med de som skrevs in vid registrering. När det gäller lösenord av typen DAS så är det lite mer komplicerat. Då måste programmet istället registrera variabler som beskriver hur användaren ritade in mönstret samt hur mönstret ser ut.

Till att börja med så måste det bestämmas vilka aspekter som jämförelsealgoritmen ska titta på. Vid litteraturstudien om vad som tidigare gjorts på området så var det vanligt att antal linjer, längd för varje linje samt startpunkt för varje linje jämfördes [13] [14]. Därför används även dessa aspekter i det här projektet. Utöver det kommer även slutpunkten för varje linje registreras och användas som en jämförelseaspekt. En av de vanligaste typerna av DAS är QDAS som använder ett rutnät för att följa riktningar för var/hur en användare drar linjen. QDAS rutnät har dock svagheter på så sätt att de tvingar användaren att förstå QDAS väl för att det ska fungera. Till exempel så måste användaren ta hänsyn så dem inte drar en linje ovanpå en rutnätslinje eller över ett hörn i rutnätet. Då är det slumpen eller minimal precision som avgör vilken ruta den ritade linjen är i [10], vilket ökar risken att inloggningen misslyckas som är en negativ faktor på användarvänlighet.
Projektet kommer därför inte att tillämpa QDAS eller använda sig utav ett rutnät för att avgöra vilken riktning en användare ritade i ett visst intervall i en linje. Istället kommer programmet att tillåta användaren att rita helt fritt utan att ta hänsyn till några rutnätslinjer. Det kommer höja användarvänligheten jämfört med QDAS men det kommer samtidigt öka komplexiteten för algoritmen som avgör lutning och riktning för varje linje.

De aspekter som jämförs för grafiska lösenord:

- Startpunkt för varje linje
- Slutpunkt för varje linje
- Längd för varje linje
- Antal linjer
- Lutning för varje linje (ett värde på lutning)
- Riktning för varje linje (linjens riktning i form av ökning eller minskning i x- och y-led)

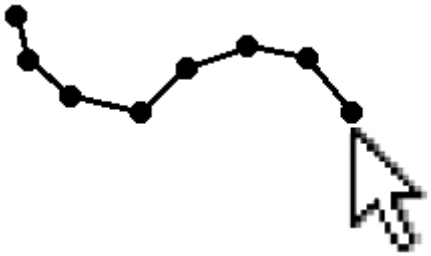
De jämförelseaspekterna skapar lösenordssträngarna för de grafiska lösenorden genom att dess värde läggs ihop. När de läggs ihop så adderas de inte utan istället läggs de efter varandra så de bildar en sträng. Strängen konstrueras på samma sätt vid registrering som vid inloggning, så om samma mönster ritas på samma sätt så blir även strängarna matchande.

4.2 Felmarginaler

Om en användare skulle vara tvungen att återupprepa sitt grafiska lösenord exakt på pixelprecision så skulle det vara mycket svårt. Det skulle inte fungera rent praktiskt då användarvänligheten skulle vara för låg. Lösningen blir att användarens mönster vid inloggning måste tillåtas skilja sig lite ifrån de som ritades vid registrering. Felmarginalernas värde testas först fram utav programmeraren och sedan testas dem även i pilottest av externa parter.

4.3 Lösenordsruta

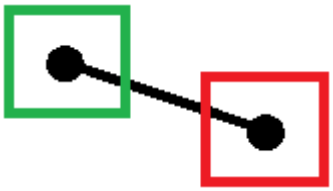
I C# finns det flera alternativ att välja mellan för att tillverka en ruta som det går att rita i. De som valdes var att använda två färdig klasser, Graphics och Pen. Klassen Graphics knyts till en panel i gränssnittet som kommer vara lösenordsrutan vilket i sin tur använder sig utav klassen Pen för att möjliggöra att det går att rita i den.
För att kontrollera ritfunktionen används olika event. Första eventet som används är när användaren trycker ner vänster musknapp, då registreras koordinaterna var musen befinner sig för tillfället. Nästa event är när muspekaren flyttas samtidigt som vänster musknapp är nedtryckt. Så fort det sker så registreras de nya koordinaterna för muspekarens nya position. Då aktiveras Graphics och Pen klassen som använder koordinaterna från första mustrycket och drar en linje till de nya koordinaterna som registrerades när muspekaren flyttades. Båda punkternas koordinater sparas för senare användning. Fortsätter användaren att flytta muspekaren med vänster musknapp nertryckt så registreras det nya värdet och en linje ritas från den näst senaste registrerade koordinaten till den senaste. Det medför att varje linje består av flera dellinjer precis som figur 2 nedanför illustrerar en linje bestående av sju dellinjer. Punkterna i linjen ska illustrera när eventet reagerar på att muspekaren flyttas med vänster musknapp nedtryckt och en ny dellinje påbörjas.



Figur 2: Illustrerar en linje bestående utan sju dellinjer ritade med hjälp av muspekaren.

4.4 Implementation av jämförelseaspekter

Koordinater för varje linjes startpunkt registreras med hjälp av eventet för när vänster musknapp trycks ner. Felmarginalen fungerar så att det är accepterat att trycka några pixlar fel i x- och y-led. För slutpunkt i linjen fungerar det på liknande vis med den skillnaden att koordinaterna registreras från eventet när vänster musknapp släpps upp. Förenklat skulle det kunna beskrivas så att användaren måste börja linjen inom ett visst område som är placerat efter var linjen började vid registrering. Precis som figur 3 nedanför illustrerar så är det gröna området accepterat startområde och det röda området accepterat slutområde för den ritade linjen (färgrutorna används endast för att illustrera och är inte synliga under användning). Så om nedanstående linje registreras så behöver användaren starta sin linje innanför det gröna området och avsluta den innanför det röda området för att kunna logga in. Felmarginalen testades fram till att tillåta 30 pixlars högre eller lägre värde i x- respektive y-led.

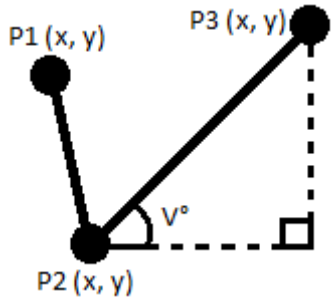


Figur 3: Grön och röd ruta ska illustrera accepterad felmarginal för start- respektive slutpunkt för ritad linje.

För att nå antalet linjer som ritas så används eventet för när vänster musknapp trycks ner. När vänster musknapp trycks ner så ökar antalet linjer med ett tills användaren väljer att registrera eller logga in. Denna jämförelseaspekt är unik på så sätt att det inte tillåts någon felmarginal. Användaren måste alltså använda exakt lika många linjer vid registrering som vid inloggning.

För att få varje linjes längd så läggs alla dellinjernas längd ihop. Varje dellinjes längd räknas ut med hjälp av Pythagoras sats [16]. För felmarginal så tillåts linjen att skilja sig till viss del i längd. Till en början så användes ett fast antal pixlar som felmarginal. En linje fick vara x pixlar längre respektive kortare än den registrerade linjen. Det medförde problem på så sätt att om en kort linje ritades så accepterades nästan alla korta linjer på grund av att felmarginalen var väldigt stor jämfört med linjens längd. Liknande problem uppstod om en lång linje ritades då felmarginalen var väldigt liten jämfört med den långa linjen och det blev väldigt svårt att återupprepa den långa linjen med bara några få pixlars felmarginal. Lösningen blir att använda ett procenttal baserat på den registrerade linjens längd som felmarginal. Det möjliggör för användaren att använda korta linjer samt att det underlättar när en lång linje ska återupprepas. Felmarginalen för linjers längd testades fram till 25%. En linje får alltså vara 25% kortare respektive längre än den som registrerades.

Lutningen är det svåraste momentet och har delats upp i flera delar. Det som används för att jobba med lutning är det listor som innehåller varje dellinjes koordinater samt varje dellinjes längd. För att få ut lutningen för en dellinje så tillämpas sinussatsen [17].



Figur 4: Bilden ovan föreställer en linje bestående av två dellinjer. Första dellinjen går mellan P1 (punkt 1) och P2 och andra dellinjen går mellan P2 och P3.

För att få reda lutningen eller graderna för vinkel V från figur 4 så behövs först längden på dellinje två, den mellan P2 och P3. Längden nås med hjälp av Pythagoras sats.

Ekvation 1: Pythagoras sats:

$$a^2 + b^2 = c^2$$

Ekvation 2: Omskrivning av Pythagoras sats för att nå hypotenusan i form av variabel c:

$$\sqrt{a^2 + b^2} = \mathrm{c}$$

Ekvation 3: Ekvation 2 med variabler från figur 4:

$$\sqrt{\big((P3.x) - (P2.x)\big)^2 + \big((P3.y) - (P2.y)\big)^2} = Längd\ på\ dellinje\ mellan\ P2\ och\ P3$$

När längden på hypotenusan, alltså dellinjens längd är tillgänglig så möjliggör det användning av sinussatsen för att räkna ut vinkeln som används för att jämföra lutningen för varje dellinje, som markeras med V i figur 4.

Ekvation 4: Sinussatsen:

$$\frac{\sin A}{a} = \frac{\sin B}{b} = \frac{\sin C}{c}$$

Ekvation 5: Omskrivning av sinussatsen för att få $\sin A$ själv på ena sidan av ekvivalenstecknet:

$$\sin A = \frac{\sin B}{b} \times a$$

Ekvation 6: Utveckling av ekvation 5 för att få vinkeln på $\sin A$:

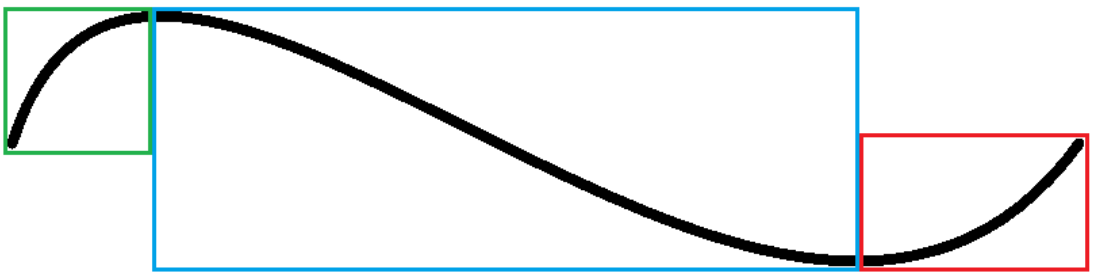
$$\sin^{-1}\left(\frac{\sin B}{b} \times a\right) = Vinkel\ A\,°$$

Ekvation 7: Ekvation 6 med variabler från figur 4 där P2P3 representerar längden på dellinjen mellan P2 och P3:

$$\sin^{-1}\left(\frac{\sin 90}{P2P3} \times (P3.y - P2.y)\right) = Vinkel\ V\,°$$

Med hjälp av Pythagoras sats och sinussatsen så är det nu möjligt att se varje dellinjes lutning. Kraftfull data för jämförelsealgoritmen men oanvändbar för tillfället då en linje som ser någorlunda rak ut inte är särskilt rak om man zoomar in och tittar på varje dellinje. Så om programmets lutningsdata skulle användas direkt som jämförelseaspekt skulle det bli väldigt svårt för en användare att logga in. Ytterligare ett problem är att varje dellinjes längd baseras på en kombination muspekarens hastighet samt datorns hastighet att registrera de musevent som används vid ritfunktionen. Drar användaren långsamt med muspekaren blir det många korta dellinjer kontra om användaren drar snabbt så blir det färre men längre. Om programmet skulle jämföra lutningsdata för varje dellinje så krävs det att användaren ritar in sitt mönster i exakt samma hastighet/tid samt på en dator som är exakt lika snabb på att registrera musevent och utöver detta att alla dellinjer inte har några avvikelser för lutning.

För att göra programmets lutningsdata användbar behövs fler algoritmer och funktioner. Istället för att jämföra varje dellinjes lutning så jämförs den genomsnittliga lutningen på alla dellinjer så länge dem går i samma riktning. Med riktning så menas exempelvis att så länge en linjes x-värde ökar (+) och y-värde sjunker (-) så håller linjen samma riktning (+-). Skulle y även börja öka (+) så skulle linjen byta riktning från +- till ++. Figur 5 nedanför illustrerar en linje som en användare har ritat. Den del av linjen, alltså de dellinjer som befinner sig i den gröna rutan har en ökning för x- och y-värden. Med andra ord så är dess riktning ++ inuti den gröna rutan. Inuti den blå rutan så går dellinjerna +- och för den röda rutan så är riktningen ++.



Figur 5: Bilden ovan illustrerar en linje och dess riktning vid en viss punkt där rutorna visar att punkterna i inuti respektive ruta har samma lutning.

För att ta reda på vilken riktning varje dellinje har så används dess start- och slutpunktskoordinater där dess x- respektive y-värde jämförs och med hjälp av detta är det

möjligt att avgöra riktningen. Det är viktigt att ta hänsyn till problemet med att en linje som ser rak ut ofta inte är rak om man zoomar in och granskar varje dellinje. Exempelvis om en linje går ++, alltså ökar i både x- och y-led så är det troligt att där är någon dellinje på någon enstaka pixel som går i en annan riktning. Detta skulle medföra att där finns ett eller flera område på ett fåtal pixlar som har en annan riktning. Det är väldigt svårt för en användare att både upptäcka men även återupprepa en sådan avvikelse. Lösningen för att gå runt problemet är att inte titta på en dellinje i taget. Istället är det bättre att även titta på närliggande dellinjer och avgöra vilken riktning som är vanligast inom det spann av dellinjer som undersöks och på så sätt undgå små avvikelser. När algoritmen avgör vilken riktning som är vanligast så är det viktigt att den inte jämför antalet dellinjer som går i en viss riktning utan istället jämför antalet pixlar som går i en viss riktning. Exempelvis om tre dellinjer på två pixlar vardera går ++ och två dellinjer på nio pixlar var går -+ så ska algoritmen välja de 18 (2×9) pixlar som går -+ över de sex (3×2) pixlar som går ++.
Det är alltså möjligt att dela upp linjerna i delar beroende på dess dellinjers lutning. Detta medför att den lutningsdata som hämtades tidigare med hjälp av sinussatsen kan användas. Istället för att jämföra varje dellinjes lutning så jämförs den genomsnittliga lutningen för varje närliggande dellinje som har samma riktning.

Exempelvis om figur 5 används igen så beräknas den genomsnittliga lutningen i den gröna, blå och röda rutan. Dessa tre lutningar används som jämförelseaspekt och behöver på ett ungefär återupprepas vid nästa inloggning. Utöver felmarginal för lutning så tillåts även ett område att vara fel. Exempelvis så tillåts en av rutorna i figur 5 att ligga utanför det accepterade lutningsintervallet. Anledningen till att ett fel tillåts är på grund av en upptäckt som gjordes av både programmeraren samt i pilottestet med externa testare. Användare slarvade ofta i slutet av linjer på så sätt att de släppte upp vänster musknapp för sent innan de flyttade muspekaren, vilket skapade en eller flera dellinjer i slutet på linjen. Detta var ett problem för användarvänligheten men löstes med hjälp av att sänka säkerheten i form av att tillåta ett fel i lutningen.

Då riktning räknas ut så jämförs även dessa så att varje linjes riktningsbyte sker i samma ordning. Här tillåts även ett fel med samma anledning som gavs för lutning.

5 EXPERIMENT

I denna del så förklaras varför experimentet utfördes, hur det planerades, tillvägagångssätt samt hur det utfördes.

5.1 Analys och syfte

För att kunna göra en utvärdering och analys behövs variabler och värden att jämföra. För att få fram jämförelsevariabler att analysera så utfördes ett experiment med teckenbaserade samt grafiska lösenord. Det som låg i fokus var att mäta tiden det tog för en användare att registrera samt logga in med teckenbaserade respektive grafiska lösenord. Det noterades även antalet misslyckade inloggningar som deltagarna genomförde på varje lösenord. De lösenordssträngar som användes eller genererades under experimentet sparades för jämförelse av säkerhet och bruteforce-test. Variablerna som skapades med hjälp av experimentet möjliggör jämförelse och beroende på resultat kan det ge en indikation på vilken typ av lösenord som är bättre i respektive aspekt.

5.2 Hypoteser

- **H0**: Grafiska och teckenbaserade lösenord är likvärdiga i säkerhet och användarvänlighet.
- **H1**: Grafiska lösenord är mer användarvänliga men osäkrare än teckenbaserade lösenord.

5.3 Val av deltagare

Kriterium för deltagande i experimentet är att personen har en viss datorvana och känner sig trygg med användning av mus och tangentbord. Deltagaren behöver även läsa igenom och skriva på ett dokument med information att de förstår deltagandevillkoren för experimentet.

5.4 Korrekthetsdiskussion

För att minska att slumpen ska påverka resultatet så ska så många deltagare som möjligt genomföra experimentet. Deltar för få i experimentet så är det större risk att slumpen gör så att de som deltar inte motsvarar genomsnittet, utan de exempelvis är bättre eller sämre än genomsnittet på att antingen rita in de grafiska lösenorden eller skriva in de teckenbaserade.
De deltagare som kommer utföra experimentet är vänner och kollegor varav samtliga är datorintresserade. Detta skulle kunna ha en påverkan på experimentresultatet då dem troligen inte motsvarar den genomsnittliga datoranvändaren utan troligen är lite mer erfarna.
Om en deltagare skulle ha hela eller delar av de slumpgenererade teckenbaserade lösenordet i exempelvis sitt egna lösenord eller på annat vis känner igen kombinationen så kan de sitta i personens muskelminne. Detta skulle medföra att personen skriver in ett lösenord snabbare än vad de skulle ske vid första användning. Det är väldigt osannolikt i och med att de fyra teckenbaserade lösenorden är slumpgenererade.
Det finns en risk att deltagare lär sig eller blir mer motiverade och snabbare allt efter som experimentet fortskrider. Det skulle medföra att om alla deltagare avslutade med grafiska lösenord att de skulle få en lägre tid och bättre resultat. För att motverka detta så börjar hälften av deltagarna med teckenbaserade lösenord och den återstående halvan att börjar med grafiska lösenord.

När det gäller säkerhetsjämförelse så kommer det avgränsas till att endast testa bruteforce-attacker mot lösenordssträngen. Det finns en intressant diskussion kring bruteforce-attacker mot grafiska lösenordssträngarna i och med att de tillåter felmarginaler. När felmarginaler accepteras så tillåts närliggande lösenord, med andra ord så tillåts flera lösenord vid inloggning. Hur många närliggande lösenord som tillåts beror på felmarginaler samt hur många linjer och riktningsbyte som lösenordet innehåller. Om en angripare vet i vilken ordning

som lösenordets jämförelseaspekter sätts ihop till lösenordssträngen samt hur stora felmarginalerna är så skulle det vara möjligt att förfina bruteforce-attacken till att testa sig fram med felmarginalernas storlek vilket skulle begränsa antalet möjliga lösenord och på så sätt snabba upp attacken betydligt. För det här projektet så kommer det begränsas till att en angripare inte har förståelse för hur lösenordssträngarna är uppbyggda eller felmarginalernas värde utan kommer utföra en traditionell bruteforce-attack där alla lösenordsvarianter planeras att testas.

Vid återupprepning av experiment är det viktigt att skärmupplösningen är 1920 x 1080 då flera jämförelsealgoritmer för de grafiska lösenorden jobbar mot pixlar och är kalibrerade för just denna skärmstorlek.

5.5 Utförande av experiment

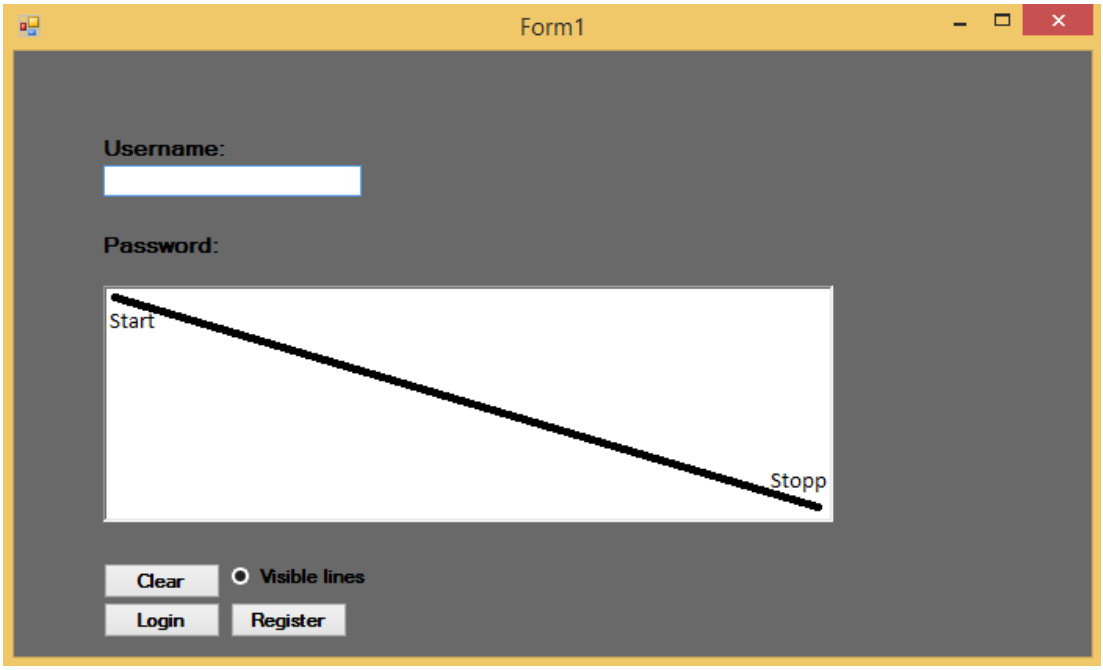
Deltagarna utförde experimentet enskilt med en handledares närvaro. Handledaren var närvarande för att inleda experimentet med en genomgång och se till att deltagaren förstod hur experimentet skulle utföras. Uppstod det frågor, oklarheter eller eventuella dator-/programproblem under experimentets gång så fanns handledaren tillgänglig för hjälp så länge inte hjälp gavs för att slutföra experimentet.
Experimentet genomfördes på en laptop som tillgodosågs av handledaren så att samtliga deltagare använde samma dator, tangentbord och mus.
Datorn som användes var i Windowsmiljö (Windows 10) då flest användare känner sig trygga med det.
Experimentet började med att deltagaren fick ett informationsblad med deltagandevillkor för experimentet som behövdes läsas igenom och skrivas på av både deltagare och handledare innan experimentet började. Om deltagandevillkoren accepteras och dokumentet skrivs på så gav handledaren en muntlig genomgång utav hur experimentet skulle utföras. Deltagaren fick även ett dokument med informationen i textform som det gavs tid till att läsa igenom (dokument finns i bilaga 10.2).
Experimentet gick ut på att deltagaren börjar med att registrera ett förutbestämt lösenord, för att sedan logga in med det. Det är viktigt att handledaren kontrollerar att deltagaren använder de förutbestämda lösenorden och inte skriver något annat. Om en användare av misstag skriver fel lösenord vid registrering så räknas det som ett misslyckat inloggningsförsök.
För teckenbaserade lösenord så slumpgenererades fyra lösenord bestående av stora och små bokstäver, siffror samt specialtecken. Lösenorden består av 6, 8, 10 respektive 12 tecken då de är vanliga lösenordslängder [1].

De teckenbaserade lösenord som användes var:

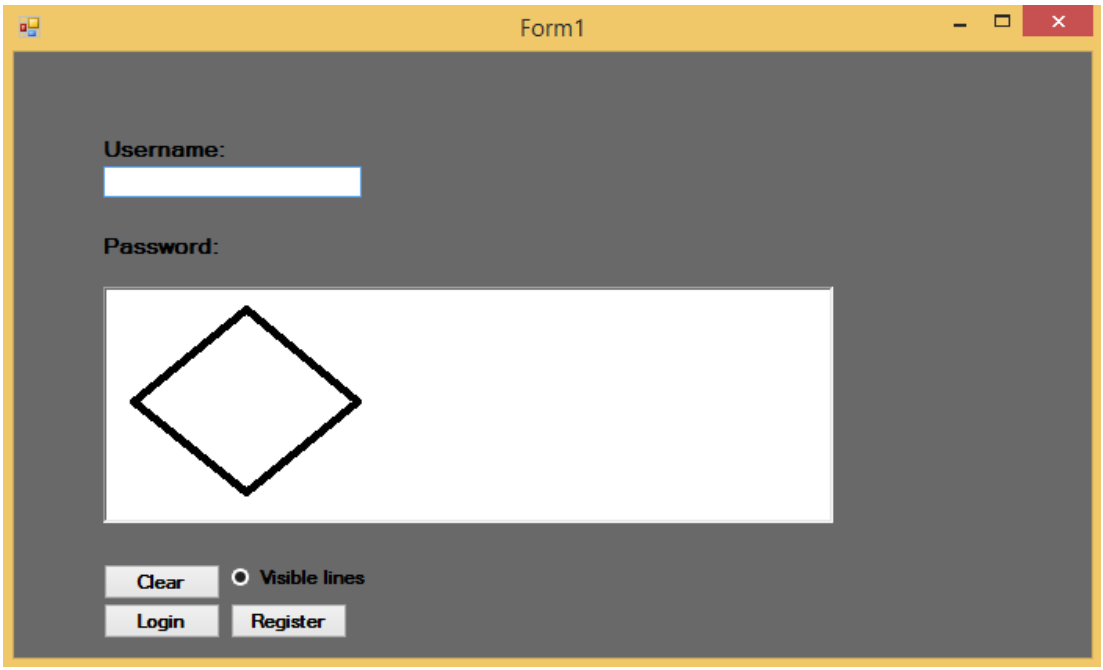
1. *a3!>VA*
2. *h9xY@+pV*
3. *?aQBn_!292*
4. *uV46Mu]=GL(E*

För de grafiska lösenorden så används bilder på grafiska mönster med tillhörande hjälptexter som deltagaren skulle följa för att försöka rita ett så likt mönster som möjligt. Deltagaren skulle då registrera sitt mönster/grafiska lösenord och försöka återupprepa det för att logga in.
Figur 6 nedanför representerar de grafiska gränssnittet som används för de grafiska lösenorden och är även bilden för de första grafiska lösenordet som deltagarna skulle använda.
Varje deltagare skulle börja med att försöka rita de förutbestämda mönstret i rutan och registrera det genom att trycka på registreringsknappen (Register). När det sker så töms lösenordsrutan på det som ritas. Deltagaren försökte sedan att återupprepa de som precis ritats samt logga in genom att trycka på login-knappen (Login). När deltagaren trycker på login-knappen får dem ett svar på datorskärmen om inloggningen var lyckad eller inte. Var inloggningsförsöket inte lyckat så får användaren testa igen tills lyckad inloggning sker. Skulle

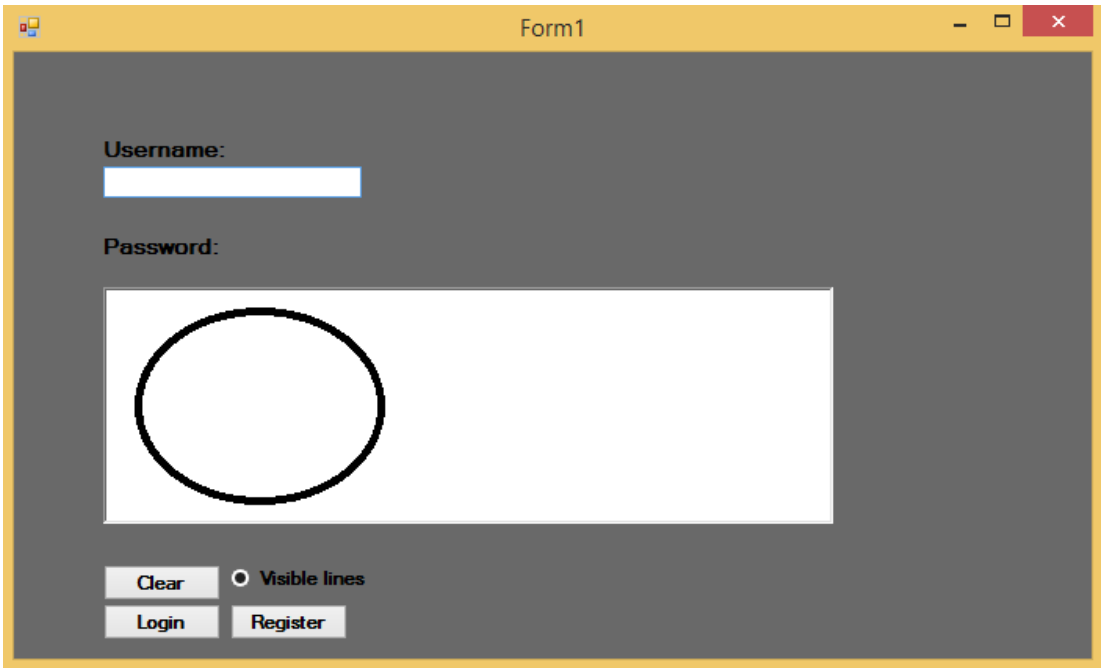
en deltagare rita något av misstag eller inte vara nöjd så finns det möjlighet till att rensa lösenordsrutan med hjälp av återställningsknappen (Clear).
Rutan för användarnamn (Username:) används inte i experimentet då endast lösenord är intressanta utan är endast där för att ge deltagaren en känsla av att det är en inloggning som sker. Även valet för synliga/osynliga linjer (Visible lines) används inte utan är till för framtida/fortsatt forskning.



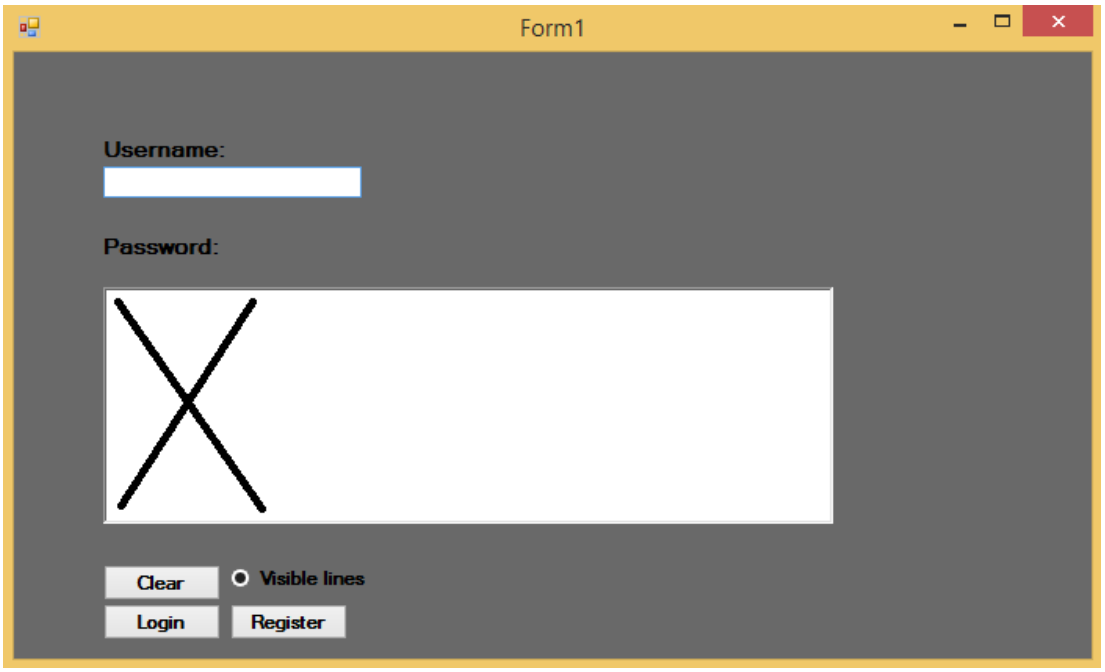
Figur 6: Första grafiska lösenordet som deltagarna skulle använda under experimentet.
Hjälptext: *Rita en diagonal linje från övre vänster hörn till undre höger hörn med hjälp av ett streck.*



Figur 7: Andra grafiska lösenordet som deltagarna skulle använda under experimentet.
Hjälptext: *Rita en romb med hjälp av ett streck.*



Figur 8: Tredje grafiska lösenordet som deltagarna skulle använda under experimentet.
Hjälptext: *Rita en cirkel med hjälp av ett streck.*



Figur 9: Fjärde grafiska lösenordet som deltagarna skulle använda under experimentet.
Hjälptext: *Rita ett kryss med hjälp av två streck.*

Varför just dessa mönster valdes att representera de grafiska lösenorden är för att dessa mönster har några specifika egenskaper som var intressanta att testa. De första grafiska lösenorden i form av den diagonala linjen valdes för att ha med ett väldigt simpelt lösenord som alla förhoppningsvis skulle klara utan problem. De andra lösenordet i form av en romb valdes för att testa hur deltagare klarade av att rita vinklar. De tredje lösenordet valdes för att testa hur väl en deltagare kunde rita jämna kurvor då det bestod av en cirkel. Fjärde och sista lösenordet som bestod av ett kryss valdes för att testa hur väl deltagare klarade det när där var flera linjer som behövde ritas.

När en deltagare var klar med både det teckenbaserade och det grafiska lösenorden så avslutades experimentet med att deltagaren skulle besvara några frågor kring de grafiska samt teckenbaserade lösenord som användes i experimentet (dokument finns i bilaga 10.3).
För att få ett värde på hur lång tid det skulle ta att utföra en lyckad bruteforce-attack så genomfördes inte bruteforce-attacker på lösenordenssträngarna då det skulle ta flera år med en genomsnittlig persondator. Istället så beräknades tiden det skulle ta att utföra en lyckad bruteforce-attack. För att möjliggöra den uträkningen så behövs något typ av hastighet på bruteforce, alltså hur många försök per sekund som en dator kan testa. För att få en hastighet så utfördes en bruteforce-attack mot en lösenordssträng, attacken slutfördes inte utan var endast till för att se vilken hastighet som uppnåddes.
Bruteforce-programmet som användes var John the ripper [18]. Programmet kördes på operativsystemet Linux Kali [19]. Hårdvara och programversioner som användes vid bruteforce-test och som benämnas som vanlig/genomsnittlig persondator i rapporten:

- CPU: Intel® Core™ i7-4770 CPU @ 3.40GHz (8 CPUs).
- GPU: AMD Radeon ™ R9 200 Series.
- RAM: 16 GB.
- Operativsystemsversion: Kali Linux 1.0.9 64-bit, Kernel Linux 3.14-kali1-amd64.
- John the ripper: Version 1.7.9-jumbo-7_omp [linux-x86-64].

5.6 Insamling av data

För insamling av data så kommer programmet att spara lösenordssträngarna samt meddela deltagaren och handledaren genom en text på skärmen om det var en lyckad inloggning eller inte. Handledaren kommer att använda ett stoppur för att mäta tiden varje deltagare lägger på varje lösenord.

6 RESULTAT OCH ANALYS

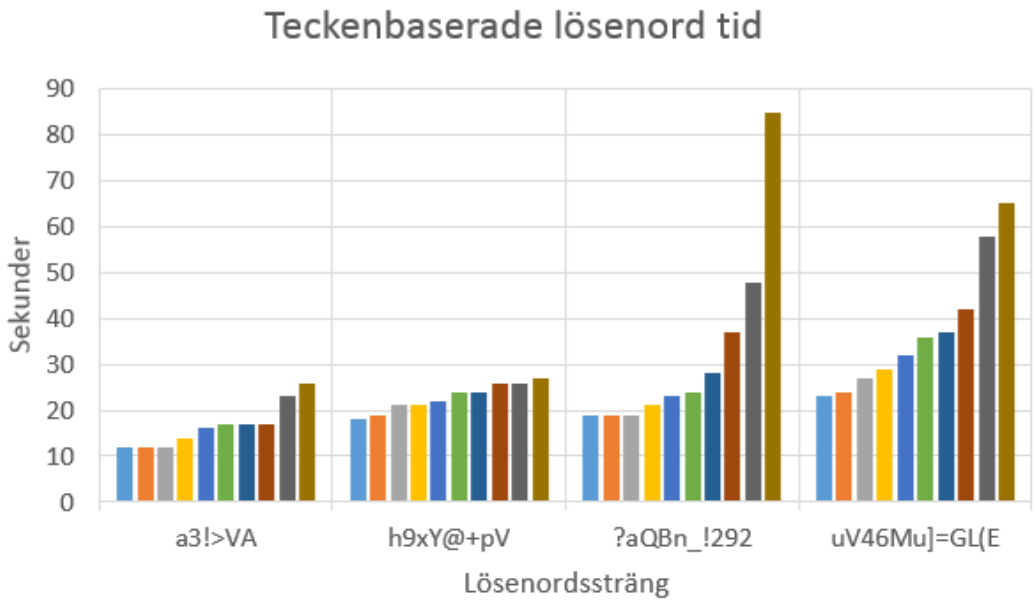
I denna del presenteras och analyseras resultat från utförda experiment för teckenbaserade och grafiska lösenord samt de följdfrågor som experimenten avslutades med.

6.1 Användarvänlighet

Användarvänligheten mäts och jämförs huvudsakligen i hur lång tid det tog för en användare att registrera och logga in med ett lösenord och hur många lyckade kontra misslyckade försök som gjordes.

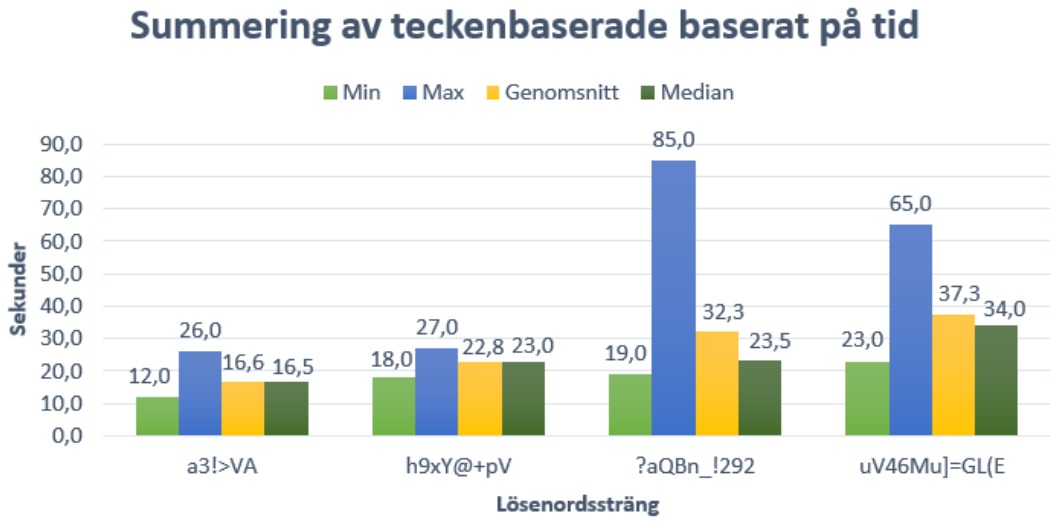
6.1.1 Teckenbaserade lösenord

Figur 10 nedanför illustrerar resultat i tid det tog för samtliga deltagare att registrera samt logga in med respektive teckenbaserat lösenord under experimentet. Värdena är sorterade efter tid för att underlätta jämförelsen. Det är med andra ord inte så att en deltagare är knuten till en viss färg.



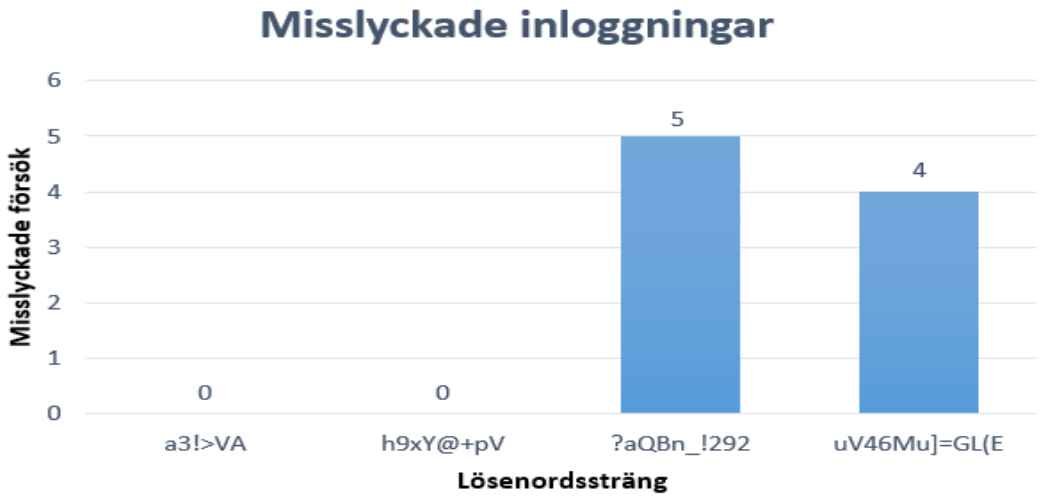
Figur 10: Stapeldiagrammet illustrerar antal sekunder det tog för samtliga deltagare att logga in med respektive teckenbaserat lösenord.

För att ge en bättre överblick för hur lång tid det tog för deltagarna att använda de fyra teckenbaserade lösenorden så har data sammanställts för kortast (min), längst (max), genomsnitt samt median avrundat till en decimal. Detta illustreras nedanför i figur 11.
En naturlig trend som grafen pekar på är att den genomsnittliga- och mediantiden ökar när antalet tecken ökar.



Figur 11: Stapeldiagrammet illustrerar sammanställd data från samtliga teckenbaserade lösenord grupperat på kortas, längst, genomsnitt samt median baserat på tid.

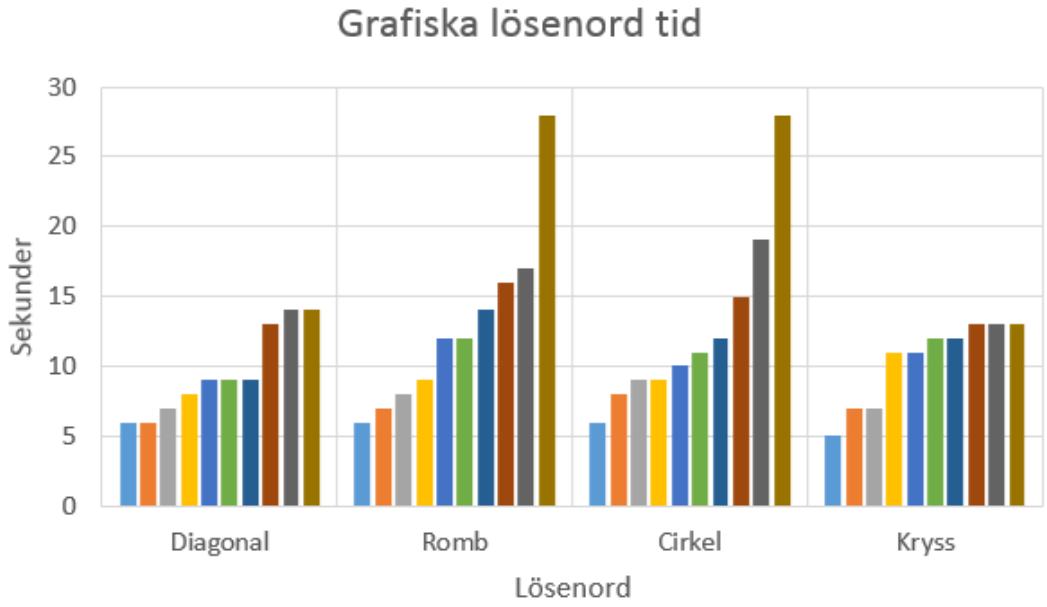
Antalet misslyckade inloggningsförsök som gjordes för varje teckenbaserat lösenord illustreras i figur 12 nedanför. En tydlig trend som syns i diagrammet är att några av deltagarna stötte på problem när lösenordenslängd i antal tecken uppgick till 10 eller fler. Varför det gjordes fler fel vid lösenordssträngen som innehöll 10 tecken än den som innehöll 12 tecken saknas det svar på. En hypotes skulle kunna vara att där var en kombination av tecken som deltagarna tyckte var svår att återupprepa. Det skulle även kunna vara att deltagarna lyckats med två lösenord i rad vilket kan ha gjort att de slappnade av och/eller tappade koncentrationen.



Figur 12: Stapeldiagrammet illustrerar antal misslyckade inloggningsförsök för teckenbaserade lösenord.

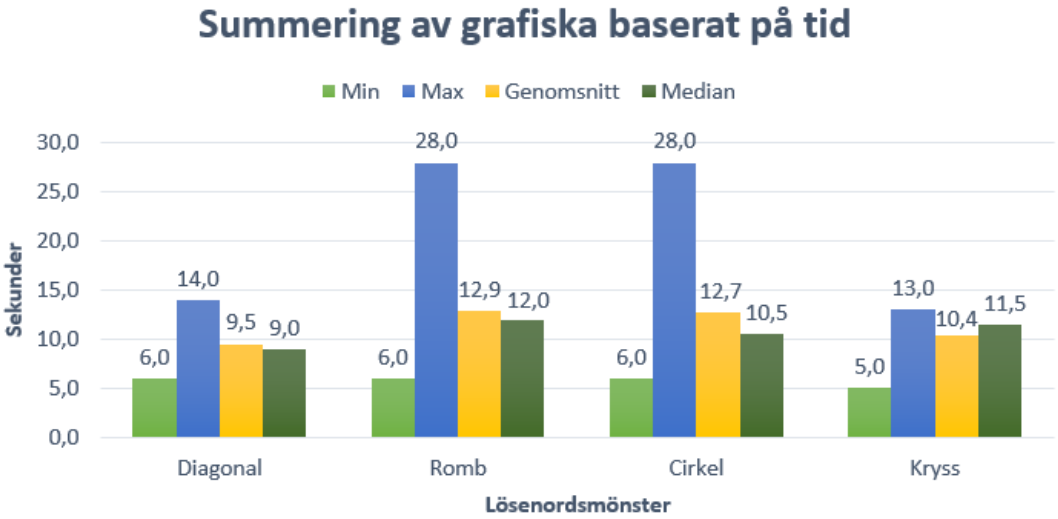
6.1.2 Grafiska lösenord

Figur 13 illustrerar resultat i tid det tog för samtliga deltagare att registrera samt logga in med respektive grafiskt lösenord under experimentet. Värdena är sorterade efter tid för att underlätta jämförelsen. Det är med andra ord inte så att en deltagare är knuten till en viss färg.



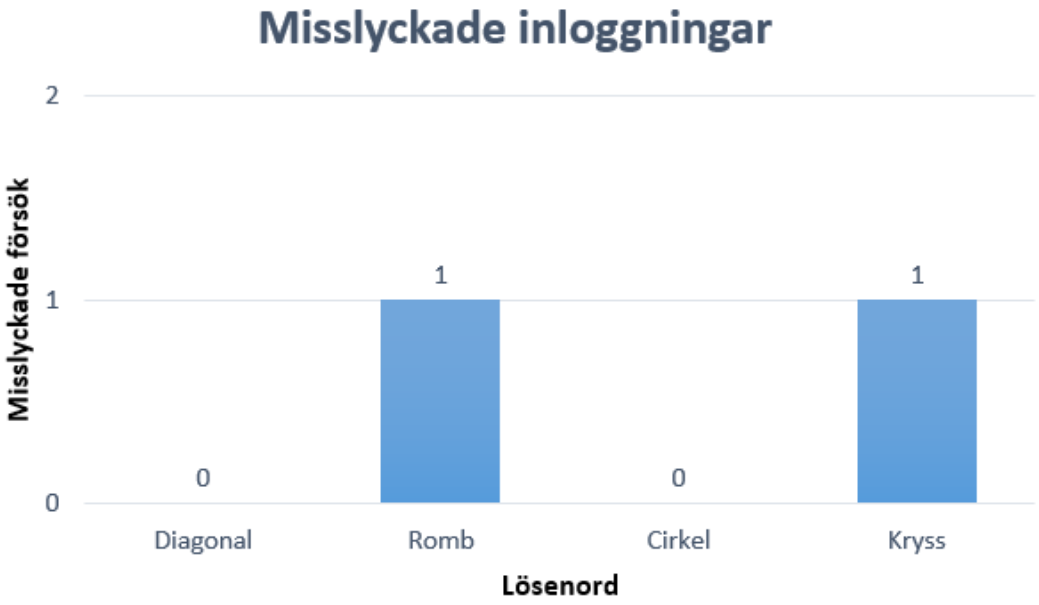
Figur 13: Stapeldiagrammet illustrerar antal sekunder det tog för samtliga deltagare att logga in med respektive grafiskt lösenord.

Precis som för teckenbaserade lösenord så har data sammanställts för att ge en bättre överblick för hur lång tid det tog för deltagarna att skriva in de fyra grafiska lösenorden. Data i form av tid i sekunder avrundat till en decimal har grupperats som kortast (min), längst (max), genomsnitt samt median. Detta illustreras nedanför i figur 14.
En spännande iakttagelse är jämförelsen mellan romb och cirkel. Då deltagarnas sammanställda värden är väldigt lika. Det kan vara att en romb och en cirkel är ganska snarlika i mönstret då man skulle kunna beskriva en cirkel som en romb där man jämnat ut hörnen. På så sätt blir linjen deltagarna ska rita för romb och cirkel ungefär lika lång vilket skulle kunna vara en avgörande faktor för tiden.



Figur 14: Stapeldiagrammet illustrerar sammanställd data från samtliga grafiska lösenord grupperat på kortas, längst, genomsnitt samt median baserat på tid.

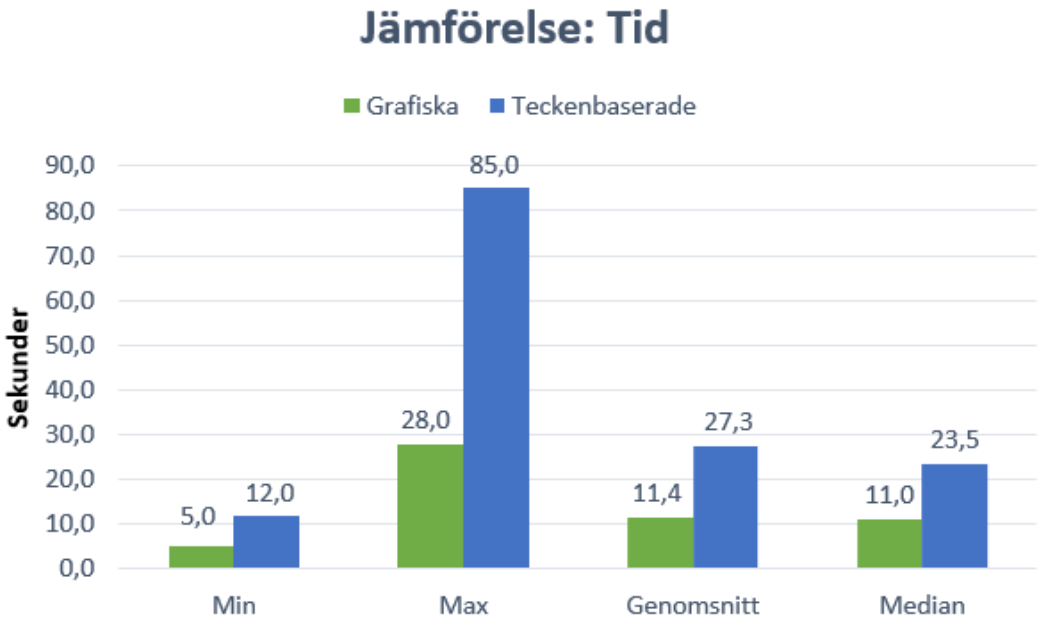
Antalet misslyckade inloggningsförsök som deltagare gjorde under de grafiska lösenorden illustreras i figur 15 nedanför.
Misstaget som gjordes för romb-lösenordet var att deltagaren hade förskjutit romben i sidled så att jämförelsealgoritmen reagerade på att linjen inte hade korrekt start- och slutpunkt.
Misstaget för kryss-lösenordet var att deltagaren började med linjen som startar uppe till höger vid inloggning, men vid registrering så började deltagaren med den andra linjen som börjar uppe till vänster. Jämförelse algoritmen reagerade då på att linjen/linjerna inte hade korrekta start- och slutpunkter.



Figur 15: Stapeldiagrammet illustrerar antal misslyckade inloggningsförsök för grafiska lösenord.

6.1.3 Jämförelse och analys av användarvänlighet

Den första aspekten som kommer jämföras är hur lång tid det tog för deltagarna att registrera och logga in med varje lösenord. Det data som illustreras i figur 16 nedanför är en sammanställning från samtliga deltagares resultat från både teckenbaserade och grafiska lösenord avrundat till en decimal.

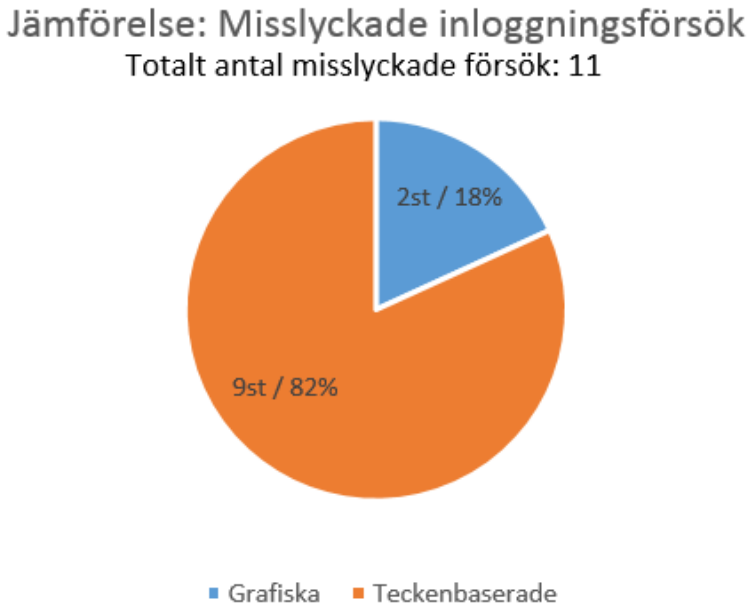


Figur 16: Stapeldiagrammet illustrerar en sammanställning av tid från både teckenbaserade och grafiska lösenord.

Det är väldigt tydligt att de grafiska lösenord som användes var bättre än de teckenbaserade som användes när det gäller tiden det tar för en användare att registrera och logga in. Om varje par av staplar i figur 16 studeras så går det att utläsa att deltagarna utförde grafiska lösenord på under halva tiden av vad de tog för dem att utföra teckenbaserade lösenord i samtliga kategorier.

Det kan vara att det är lättare och snabbare att använda ett nytt grafiskt lösenord för första gången jämfört med ett teckenbaserat. Om resultaten skulle följa samma trender om deltagarna fick träna in både de teckenbaserade och grafiska lösenorden innan experimentet utfördes är en spännande frågeställning och en möjlighet för framtida forskning.

Den andra aspekten inom användarvänlighet som jämförs är hur många misslyckade inloggningsförsök som gjordes. De data som illustreras i figur 17 nedanför är en sammanställning på misslyckade inloggningsförsök från samtliga deltagares inloggningsförsök från både teckenbaserade och grafiska lösenord.



Figur 17: Cirkeldiagrammet illustrerar samtliga misslyckade inloggningar från teckenbaserade och grafiska lösenord i både antal och procent.

Av de totala 80 antalet inloggningar som genomfördes under experimentet, 40 på teckenbaserade och 40 på grafiska, så gjordes sammanlagt 11 misslyckade inloggningar. Av de totala 11 misslyckade inloggningar som gjordes så var nio på teckenbaserade lösenord och två på grafiska lösenord. Det skulle motsvara att 82% av felen gjordes på de teckenbaserade lösenorden och de resterande 18% gjordes på de grafiska lösenorden.

6.2 Säkerhet

För att jämföra säkerheten mellan teckenbaserade och grafiska lösenord så kommer tiden beräknas för hur lång tid det skulle ta att utföra en lyckad bruteforce-attack på den genomsnittliga lösenordssträngen från varje lösenord.
En viktig aspekt för lösenord som är avgörande för tiden när det gäller en bruteforce-attack är lösenordssträngens längd. Desto längre lösenordssträng desto längre tid tar det att genomföra en lyckad bruteforce-attack.
För att få en hastighet på bruteforce så testkördes bruteforce-programmet John the ripper på tidigare nämnd dator under ungefär en timme mot ett hashat lösenord. Vid 18 tillfällen under denna timme så noterades hastigheten. Vilket i snitt blev 18489166 (18,5 miljoner) försök per sekund, som blir den hastighet som kommer att användas (skärmdump från testkörning finns i bilaga 10.1).
Det finns ytterligare en viktig aspekt för lösenord som är vilka tecken som är möjliga för användning. De tecken som lösenorden i detta experiment kan innehålla är små och stora bokstäver, siffror och specialtecken.

- Små och stora bokstäver: 58 tecken. (a-Ö)
- Siffror: 10 tecken. (0-9)
- Specialtecken: 23 tecken. (! @ # $ £ % & / ( ) = ? \ + }] [{ * < > - _)

Totalt kan ett lösenord bestå av 91 olika tecken. Detta medför att antalet lösenordskombinationer för ett lösenord blir $91^{(antal\ tecken\ i\ lösenordet)}$.

6.2.1 Teckenbaserade lösenord

För de teckenbaserade så använde samtliga deltagare samma förutbestämda lösenord. Lösenorden syns i tabell 1 i vänster kolumn. Där efter kommer lösenordets längd i antal tecken. Nästa kolumn är antal möjliga lösenordskombinationer som går att skapa med just den lösenordslängden. Sista kolumnen är hur lång tid mätt i antal år avrundat till fem decimaler det skulle ta att testa alla lösenordskombinationer. För att testa alla lösenordskombinationer med en vanlig persondator där antalet möjliga tecken som används i lösenordet är 91 och lösenordslängden är sex tecken så skulle det ta ungefär 0,00097 år, vilket motsvarar drygt åtta timmar. Om lösenordslängden skulle dubbleras till 12 tecken så skulle det istället för dryga åtta timmar ta ungefär 552848623 år att testa alla möjliga kombinationer.

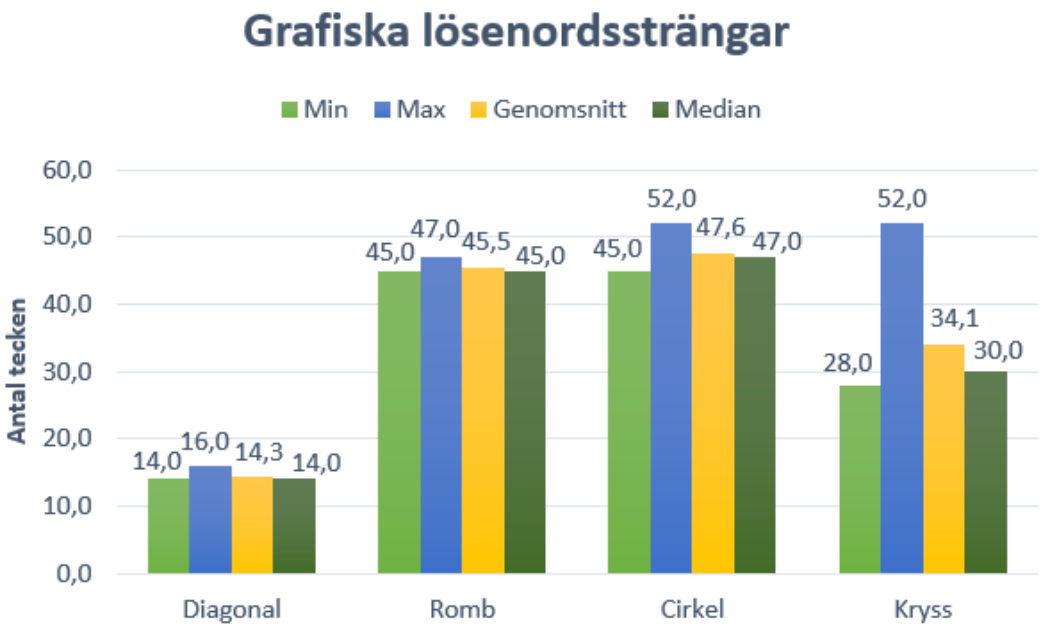
Lösenord	Längd	Kombinationer	Bruteforce tid
a3l>VA	6	567869252041	0,00097
h9xY@+pV	8	4702525276151520	8,06196
?aQBn_!292	10	38941611811810700000	66761,09448
uV46Mu]=GL(E	12	322475487413604000000000	552848623,34351

Tabell 1: Tabellen innehåller data för teckenbaserade lösenord grupperat på antaltecken, antal möjliga kombinationer samt tid i antal år det skulle ta att genomföra en bruteforce-attack med tidigare nämnd dator och programvara.

6.2.2 Grafiska lösenord

De grafiska lösenorden var även förutbestämda i form av vilka mönster deltagarna skulle använda. Skillnaden mot de teckenbaserade är att lösenordssträngarna genereras när deltagarna ritar in sitt mönster. Det medför att om inte de 10 deltagarna ritade exakt på pixelnivå samma i samtliga fyra grafiska lösenord som är väldigt osannolikt så kommer lösenordssträngarna skilja sig åt. De grafiska lösenordssträngarnas längd har sammanställts och illustreras i figur 18 nedanför (samtliga genererade lösenordssträngar för de grafiska lösenorden finns i bilaga 10.4).

Om varje stapelgrupp studeras så syns det att de diagonala lösenordet hade den kortaste lösenordssträngen för samtliga kategorier. Det beror på att det endast är en rak linje vilket bidrar till att inga riktningsbyte sker för linjen vilket medför att det inte läggs till några värde för det i lösenordssträngen. De tre resterande lösenorden har antingen vinklar, kurvor eller flera linjer som registreras och på så sätt blir deras lösenordssträng längre. Den lösenordslängd som kommer användas i bruteforce-testet för grafiska lösenord är den genomsnittliga längden avrundad till närmsta heltal.



Figur 18: Stapeldiagrammet illustrerar en sammanställning av de grafiska lösenordssträngarnas längd.

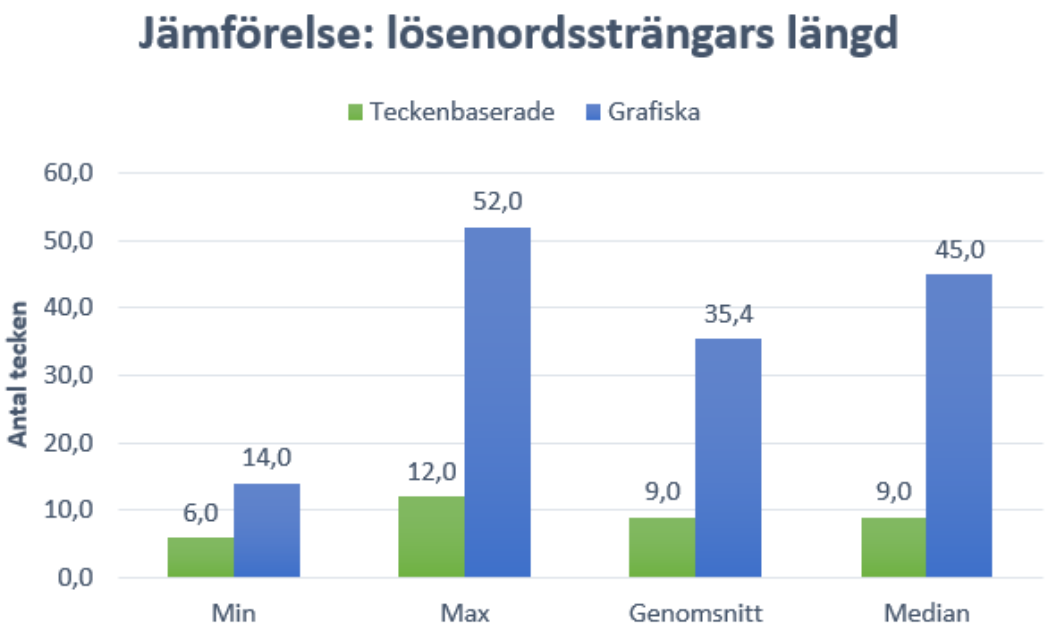
I och med att de grafiska lösenorden genererade väldigt långa lösenordssträngar så fungerade vanligt formaterade tal inte då de blev för stora, därför kommer bokstaven E att användas. Kortfattat vad som menas med E är att kommatecknet är flyttat ett steg till vänster för variabelstorleken som adderas med E (E + x). Exempelvis om tiden för att genomföra en brutreforce-attack på de grafiska lösenordet ”Cirkel” skulle formateras som ett vanligt tal så får värdet läsas ut ur tabell 2 nedanför och sedan flytta kommatecknet 79 steg/tecken åt höger, vilket bildar ett väldigt stort tal. Detta tal med 80 siffror till vänster om kommatecknet motsvarar antal år som det skulle ta att genomföra en bruteforce-attack med en vanlig persondator på samtliga lösenordskombinationer.

Lösenord	Längd	Kombinationer	Bruteforce tid
Diagonal	14	2,6704195113E+27	4,57814E+12
Romb	46	1,3058823340E+90	2,23879E+75
Cirkel	48	1,0814011608E+94	1,85394E+79
Kryss	34	4,0495553459E+66	6,94252E+51

Tabell 2: Tabellen innehåller data för grafiska lösenord grupperat på antaltecken, antal möjliga kombinationer samt tid i antal år det skulle ta att genomföra en bruteforce-attack med tidigare nämnd dator och programvara.

6.2.3 Jämförelse och analys av säkerhet

Det som jämförs är tiden det tar att genomföra en bruteforce-attack på lösenordssträngen. De lösenordsfaktorer som spelar roll för tiden är hur många möjliga tecken som kan användas samt hur många tecken varje lösenord innehåller. Antalet möjliga tecken är en konstant som är samma för både teckenbaserade och grafiska lösenord. Däremot så skiljer sig lösenordens längd som figur 19 nedanför visar. Data är sammanställd från samtliga deltagare och lösenord avrundat till en decimal. Det är tydligt att grafiska lösenord har längre lösenordssträngar än de lösenordslängder som valdes för de teckenbaserade.



Figur 19: Stapeldiagrammet illustrerar en sammanställning av lösenordslängd från både teckenbaserade och grafiska lösenord.

Det är tydligt att de grafiska lösenorden är säkrare mot bruteforce-attacker. För att ge ett bättre perspektiv så kommer den genomsnittliga lösenordslängden för teckenbaserade (9) samt grafiska (35) att jämföras i ungefär hur lång tid det skulle ta att genomföra en bruteforce-attack på samtliga kombinationer med en persondator:

- Teckenbaserade: 695 år.
- Grafiska: 631768918519666000000000000000000000000000000000000000000 år. (6,31768918519666E+53 år)

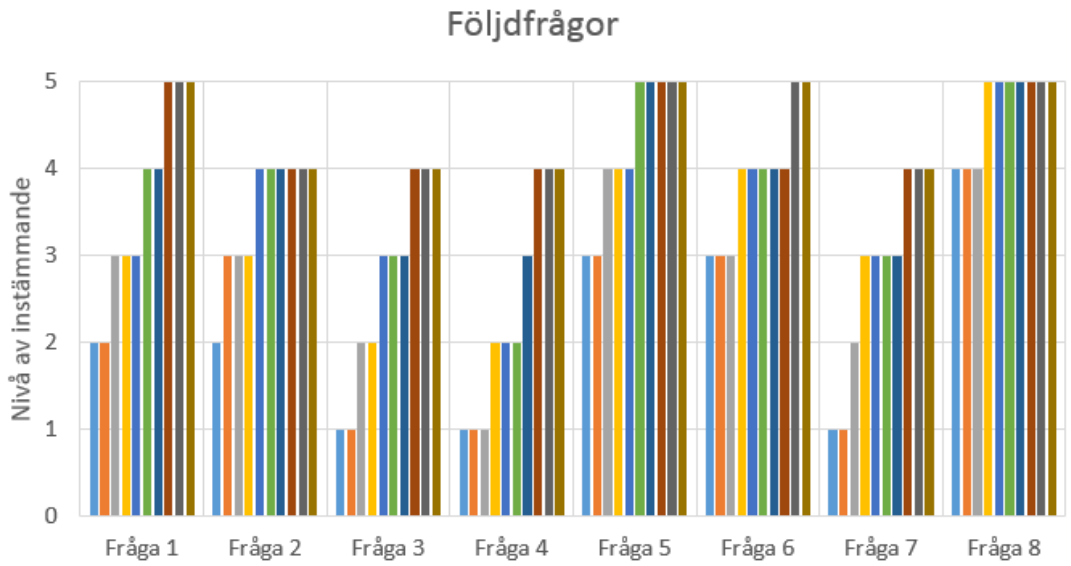
6.3 Följdfrågor

När deltagare i experimentet utfört både de teckenbaserade och grafiska lösenorden så avslutar de med att svara på några följdfrågor kring teckenbaserade och grafiska lösenord.
Processen går ut på att de får ett dokument med åtta påståenden där de ska svara med en skala från 1-5, där 1 motsvarar att de inte stämmer alls och 5 att det stämmer mycket väl (dokument finns i bilaga 10.3).

De påstående deltagarna skulle svara på var:

1. Jag upplever att teckenbaserade lösenord är användarvänliga.
2. Jag upplever att grafiska lösenord är användarvänliga.
3. Jag tyckte det var svårt att återupprepa teckenbaserade lösenord.
4. Jag tyckte det var svårt att återupprepa grafiska lösenord.
5. Jag upplevde teckenbaserade lösenords säkerhet som god.
6. Jag upplevde grafiska lösenords säkerhet som god.
7. Jag hade kunnat tänka mig att använda grafiska lösenord som autentisering vid inloggning av en dator.
8. Jag hade kunnat tänka mig att använda teckenbaserade lösenord som autentisering vid inloggning av en dator.

Samtliga deltagares svar är sammanställda i figur 20 nedanför.



Figur 20: Stapeldiagrammet illustrerar svar från samtliga deltagares följdfrågor. Där svarsnivå 1 är att påståendet inte stämmer medan nivå 5 är att det stämmer väl.

Vid jämförelse över samtliga frågor så kommer teckenbaserade ut som en vinnare. De staplar som kan vara intressanta att titta närmre på är fråga ett och två. Där frågades deltagaren om de upplevde teckenbaserade kontra grafiska lösenord som användarvänliga. Deltagarna svara visade att de tyckte att de teckenbaserade var lite mer användarvänliga jämfört med grafiska. Vid fråga fem och sex som frågar hur säkerheten upplevdes för teckenbaserade kontra grafiska så upplevde deltagarna att teckenbaserade var lite säkrare. Sedan när stapelparet för fråga sju och åtta studeras där det frågades om deltagaren kunde tänka sig att använda teckenbaserade kontra grafiska lösenord vid autentisering så väljer deltagarna teckenbaserade lösenord långt högre. Varför deltagare svarar så mycket högre för teckenbaserade lösenord över grafiska trotts att de upplever dem som väldigt nära i användarvänlighet och säkerhet kan bero på flera faktorer.
En hypotetisk faktor skulle kunna vara att de inte har använt grafiska lösenord innan och känner att de inte har koll på hur det fungerar och på så sätt faller tillbaka på det dem känner sig trygga med.

7 DISKUSSION

Användare vill inte lägga onödigt lång tid vid inloggningssekvenser utan vill att det ska gå så snabbt och smidigt som möjligt. Därför jämfördes tiden det tog för en användare att registrera och logga in med teckenbaserat samt grafiskt lösenord.
Vid analys av data som presenterades i kapitel 6 så var de grafiska lösenorden en klar vinnare. När genomsnittstiden för samtliga teckenbaserade och grafiska lösenord jämförs så tog teckenbaserade drygt dubbelt så lång tid med 27,3 sekunder mot de grafiska som endast tog 11,4 sekunder.
Förklaringen skulle kunna lyda att det var antingen de teckenbaserade lösenorden som var för långa och komplicerade och/eller att det var de grafiska lösenorden som var för korta som skapade differensen i tid. Motsägelsefullt med tanke på att de teckenbaserade var nio tecken långa i snitt vilket är en vanlig längd för teckenbaserade lösenord [1].
Då får de grafiska lösenorden undersökas om mönstren som användes var för enkla. Att några av de grafiska lösenorden var enkla är ett faktum, speciellt de första lösenordet som endast bestod av en diagonal linje. Men det som definierar om ett lösenord är för kort eller inte är lösenordssträngen. Med tanke på att det enklaste grafiska lösenordet genererade en lösenordssträng på 14 tecken jämfört med de längsta teckenbaserade lösenordet som bestod av 12 tecken så är det med stor sannolikhet inte de grafiska lösenorden för korta. Speciellt inte när flera fall av grafiska lösenord på över 50 tecken genererats vid experimentet. Om den genomsnittliga längden undersöks så innehöll teckenbaserade lösenord nio tecken jämfört med de grafiska lösenordens 35,375 tecken vilket motsäger att de grafiska lösenorden var för korta eller enkla.

De indikationer som kan tydas är att grafiska lösenord är mer användarvänliga för aspekterna tid för registrering och inloggning samt antal misslyckade inloggningsförsök.
Det finns däremot väldigt många tester och experiment inom området användarvänlighet som behöver utföras med ett fler antal deltagare för att kunna göra en faktisk bedömning.
Trotts experimentresultat så tycker deltagarna av experimentet de teckenbaserade lösenorden som lite mer användarvänliga än de grafiska.

För säkerhet och aspekten skydd mot bruteforce-attacker där angriparen inte vet hur felmarginaler och konstruktion av de grafiska lösenordssträngarna ser ut så ges det en indikation på att de grafiska lösenorden är starkare.
Skulle däremot en angripare veta hur felmarginaler och konstruktion av lösenordssträngarna ser ut så skulle angriparen kunna sänka det möjliga lösenordsspannet drastiskt. Detta skulle medföra att tiden att utföra en bruteforce-attack skulle sjunka.
Trotts att experimentresultatens variabler pekade på att grafiska lösenord var säkrare så upplevde experimentdeltagarna att de teckenbaserade lösenorden var säkrare.

8 SLUTSATS OCH UPPFÖLJANDE ARBETE

I denna del så diskuteras slutsats baserat på resultat och analys. Det gås även igenom framtida arbete och forskning inom området med bland annat delar som inte fick plats inom tidsramarna för detta projekt.

8.1 Återkoppling till frågeställningar för projektet

I den här delen så kommer sammankoppling mellan resultat, analys och frågeställning för projektet knytas samman så att eventuella slutsatser dras.

8.1.1 Användarvänlighet

När det gäller användarvänlighet baserat på inloggning och registreringstid samt lyckade kontra misslyckade inloggningsförsök så har resultat från experiment sammanställts och de pekar nästan alla i samma riktning:

- Ett teckenbaserat lösenord tog mer än dubbelt så lång tid för en deltagare att använda jämfört med de grafiska.
- Det skedde 4,5 gånger fler misslyckade inloggningsförsök med teckenbaserade lösenord jämfört med grafiska.
- När det uppstod problem för deltagarna så tog teckenbaserade som längst 85 sekunder jämfört med de 28 sekunder som var den längsta tiden för grafiska.
- Vid bästa fall så tog teckenbaserade 12 sekunder jämfört med grafiska som endast tog fem sekunder.
- Deltagarna av experimentet upplevde teckenbaserade lösenord som mer användarvänliga jämfört med de grafiska.

Under användarvänlighet så ges det en tydlig indikation på att grafiska lösenord skulle vara bättre för aspekterna registrering följt av inloggning samt lyckade kontra misslyckade inloggningar.

8.1.2 Säkerhet

För test av säkerhet så begränsades det till att endast testa bruteforce-attacker mot lösenordssträngarma och att angriparen inte vet hur de grafiska lösenordssträngarna är uppbyggda eller hur felmarginalerna ser ut.
Det som är avgörande för tid att genomföra en bruteforce-attack är lösenordets längd:

- Teckenbaserade lösenord hade en snittlängd på nio tecken.
- Grafiska lösenord hade en snittlängd på 35,375 tecken.

Vid en bruteforce-attack där alla möjliga lösenordskombinationer testas på genomsnittslängderna med en persondator så skulle det ungefär ta:

- 695 år för teckenbaserade.
- 631768918519666000000000000000000000000000000000000000 år för grafiska.

Den variabel som inte pekade för grafiska lösenord utan istället teckenbaserade var att deltagarna som utförde experimentet upplevde teckenbaserade lösenord som säkrare.

Under säkerhet så ges det en tydlig indikation på att grafiska lösenord är säkrare mot bruteforce-attacker så länge angriparen inte vet hur de grafiska lösenorden är sammansatta eller hur felmarginalerna fungerar.

8.2 Fortsatt forskning och framtida arbete

Det hade varit spännande att se om de grafiska lösenorden fortfarande är säkrare mot bruteforce-attacker på lösenordssträngen om angriparen vet hur och i vilken ordning de är uppbyggda samt hur felmarginalerna fungerar. Angriparen skulle då kunna begränsa bruteforce-programmet att följa vissa mönster så det inte testas lösenord som inte matchar de riktlinjer som jämförelsealgoritmen använder sig av när lösenordssträngen genereras. Det skulle minska antalet möjliga lösenord som behöver testas och på så sätt minska tiden för att utföra en bruteforce-attack.
Även att flera närliggande lösenord tillåts med hjälp av felmarginalerna i jämförelsealgoritmen. Har en angripare kunskap om hur stora felmarginalerna är och hur dem fungerar så skulle bruteforce-programmet även kunna konfigureras så det inte testar alla lösenord utan istället testar lösenord med felmarginalernas värde som mellanrum.

Det finns forskning som pekar på att grafiska lösenord är sårbara mot shoulder surfing [13] (shoulder surfing är när en person tittar och memorerar när en användare skriver in sitt lösenord).
En teknik att testa för att motverka shoulder surfing skulle kunna vara en knapp i gränssnittet som gör så att linjerna som ritas i de grafiska lösenordet bli osynliga. Det som skulle kunna testas är hur väl en användare kan återupprepa sitt lösenord utan att se vad som ritas jämfört med när de kan se vad som ritas. Det skulle även vara intressant att se hur en potentiell angripare skulle kunna återupprepa ett grafiskt lösenord där den ser linjerna jämfört när den inte.

Det skulle vara värdefullt att se hur tid påverkar minnet av de grafiska lösenorden. Vanligtvis med teckenbaserade lösenord så kommer användare antingen ihåg hela lösenordet eller inget av det. Snarlik är nog mönstret för grafiska, antingen kommer användaren ihåg vad den hade som lösenord, eller så gör man det inte. Skillnaden är att användaren även måste komma ihåg hur mönstret ritades. Kanske att användare skulle missa någon detalj, men det skulle vara spännande att se om dem faktiskt skulle rita alla linjer på samma sätt. Exempelvis om en linje dras från höger sida till vänster sida, om användaren återupprepar en linje som ser exakt lika ut till utseendet men är ritad från vänster sida till höger sida en längre tid efter den första linjen blev registrerad eller om den faktiskt ritas korrekt.

Skulle deltagarna vara lika duktiga på att återupprepa de grafiska lösenorden om de var mer avancerade eller skulle dem stöta på problem när lösenorden ska återupprepas. Det skulle kunna testas var samt med vad deltagare stöter på problem för med grafiska lösenord. Om det exempelvis är när antalet linjer blir fler eller dess längd blir längre.

9 REFERENSER

[1] G. E. Violettas and K. Papadopoulos, “Passwords to absolutely avoid,” in Applications of Digital Information and Web Technologies (ICADIWT), 2014 Fifth International Conference on the, 2014, pp. 60–68.

[2] R.N. Shepard, "Recognition memory for words, sentences, and pictures," Journal of Verbal Learning and Verbal Behavior, vol. 6, no. 1, pp. 156-163, 1967.

[3] D.L. Nelson, V.S. Reed, and J.R. Walling, "Pictorial superiority effect, " Journal of Experimental Psychology: Human Learning and Memory, vol. 2, no. 5, pp. 523-528, 1976.

[4] A. Paivio, T. B. Rogers, and P. C. Smythe, Why are pictures easier to recall than words? Psychonomic Science, 1968, 11(4), 137-138.

[5] M. D. Fatehah, M. Z. Jali, M. K. Wafa, and N. B. Anuar, “Educating users to generate secure graphical password secrets: An initial study,” in 2013 IEEE 5th Conference on Engineering Education (ICEED), 2013, pp. 26–31.

[6] P. Dunphy and J. Yan, "Do background images improve "draw a secret" graphical passwords?" In: Proceedings of the 14th ACM conference on Computer and communications security (CCS). New York, NY, USA: ACM, 2007, pp. 36-47.

[7] K. I. Shin, J. S. Park, J. Y. Lee, and J. H. Park, “Design and Implementation of Improved Authentication System for Android Smartphone Users,” in *2012 26th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 2012, pp. 704–707.

[8] R. Biddle, S. Chiasson, and P. C. Van Oorschot, “Graphical Passwords: Learning from the First Twelve Years,” *ACM Comput. Surv.*, vol. 44, no. 4, pp. 19:1–19:41, Sep. 2012.

[9] Y. Meng, “Designing Click-Draw Based Graphical Password Scheme for Better Authentication,” in 2012 IEEE 7th International Conference on Networking, Architecture and Storage (NAS), 2012, pp. 39–48.

[10] H. Gao, X. Guo, X. Chen, L. Wang, and X. Liu, “YAGP: Yet Another Graphical Password Strategy,” in Computer Security Applications Conference, 2008. ACSAC 2008. Annual, 2008, pp. 121–129.

[11] D. Lin, P. Dunphy, P. Olivier, and J. Yan, "Graphical passwords & qualitative spatial relations," In: Proceedings of the 3rd symposium on Usable privacy and security (SOUPS). New York, NY, USA: ACM, 2007, pp. 161-162.

[12] C. Wohlin, “Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering,” in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, New York, NY, USA, 2014, pp. 38:1–38:10.

[13] M. Oka, K. Kato, Y. Xu, L. Liang, and F. Wen, “Scribble-a-Secret: Similarity-based password authentication using sketches,” in 19th International Conference on Pattern Recognition, 2008. ICPR 2008, 2008, pp. 1–4.

[14] J. Thorpe and P. Van Oorschot, “Towards secure design choices for implementing graphical passwords,” in *Computer Security Applications Conference, 2004. 20th Annual*, 2004, pp. 50–60.

[15] A. Sadeghian, M. Zamani, and S. Ibrahim, “SQL Injection Is Still Alive: A Study on SQL Injection Signature Evasion Techniques,” in *2013 International Conference on Informatics and Creative Multimedia (ICICM)*, 2013, pp. 265–268.

[16] E. W. Weisstein, “Pythagorean Theorem.” [Internet]. Tillgänglig: http://mathworld.wolfram.com/PythagoreanTheorem.html. [Besöktes: 2016-04-20].

[17] E. W. Weisstein, “Sine.” [Internet]. Tillgänglig: http://mathworld.wolfram.com/Sine.html. [Besöktes: 2016-04-20].

[18] “John the Ripper password cracker.” [Internet]. Tillgänglig: http://www.openwall.com/john/. [Besöktes: 2016-04-24].

[19] “About Kali Linux,” 17-Dec-2012. [Internet]. Tillgänglig: https://www.kali.org/about-us/. [Besöktes: 2016-04-24].

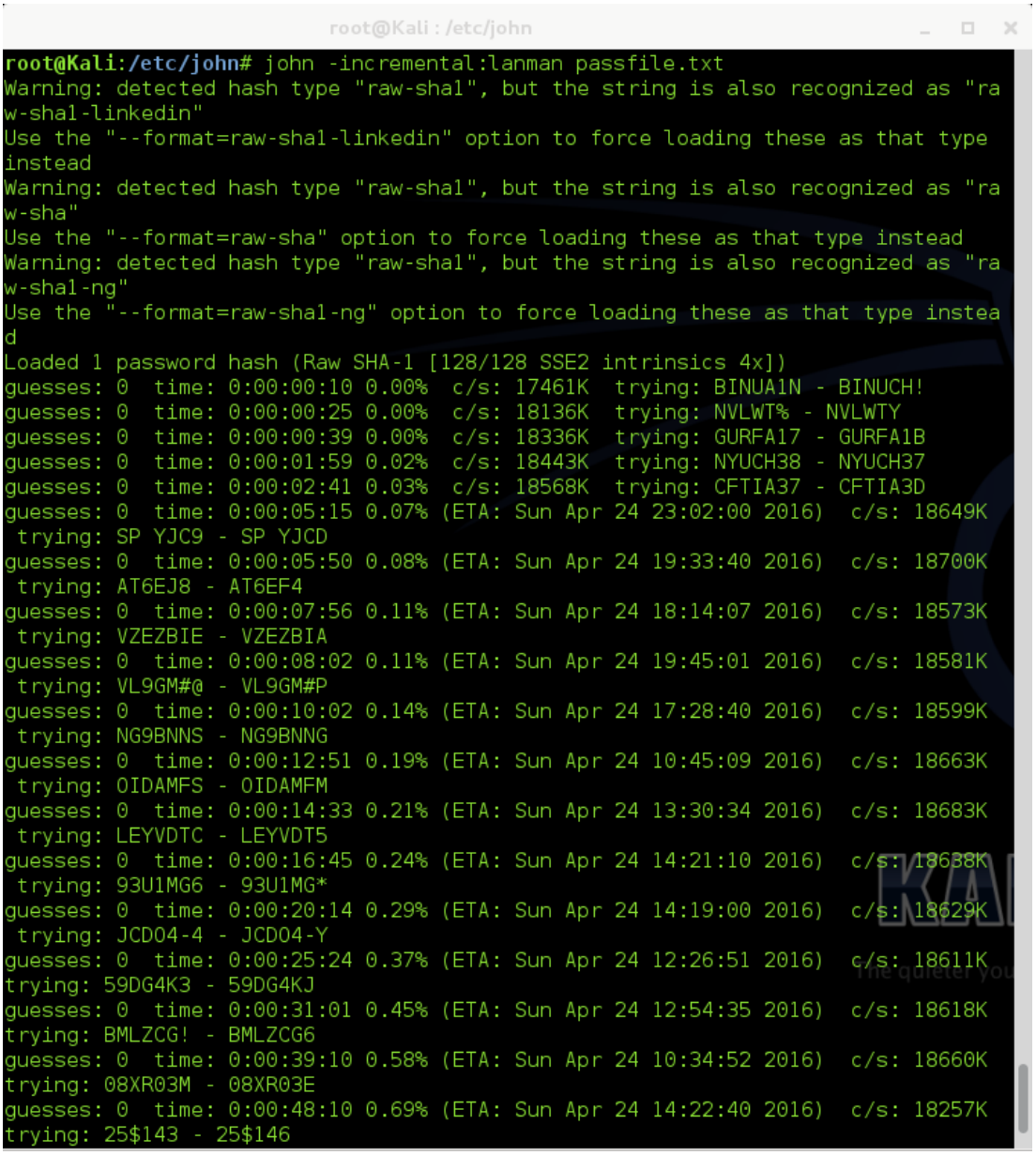
[20] ”Visual Studio – Microsoft Developer Tools.” [Internet]. Tillgänglig: https://www.visualstudio.com/ . [Besöktes: 2016-04-22]

10 BILAGOR

I denna del så finns samtliga bilagor i form av skärmdumpar, dokument, tabeller och kod som inte fick plats i rapporten.

10.1 John the ripper skärmdump

Skärmdump från test med John the ripper för att få en bruteforce-hastighet.
För att utläsa hastigheten så studeras vad som står efter ”c/s”, där antal försök per sekund visas i antal tusen. Incremental:lanman menas med vilka tecken som ska testas där lanman motsvarar att bokstäver, siffror och specialtecken testas. Passfile.txt är en fil som ligger i samma katalog som John the ripper och innehåller de hashade lösenord som bruteforce-testet körs mot.



10.2 Experimentinformation

Dokument innehållande information om hur experimentet skulle utföras som varje deltagare fick läsa igenom.

Experiment: grafiska och teckenbaserade lösenord

Användarnamnsrutan är med för att få känslan av en inloggning men kommer inte användas, därför ignorera rutan för användarnamn.

Teckenbaserade:
Den teckenbaserade delen kommer bestå av fyra slumpgenererade lösenord som ska återupprepas. Lösenorden kommer består av siffror, bokstäver och specialtecken. De har längderna 6, 8, 10 och 12 tecken.

- Börja med ett skriv (kopiera och klistra in är inte accepterat) in ett lösenord. Tryck sedan registrera.
- Skriv sedan in ordet en gång till. Tryck sedan logga in.
- Om fel sker vid inmatning av lösenordet så är det bara att försöka igen tills rätt lösenord bli inmatat.

Grafiska:
Den grafiska delen kommer bestå av fyra mönster som ska återupprepas.

- Börja med att försöka så bra det går att rita de som visas på bilden. Tryck sedan registrera.
- Försök att återskapa de som precis ritades. Tryck sedan login.
- En text kommer visas om det var tillräckligt lika för att accepteras eller inte. Om det inte är accepterat så är det bara att testa igen.

Att tänka på när man ritar sitt lösenord är att det inte bara ska vara likt bildmässigt, utan det ska även vara ritat på samma vis.
Exempelvis:

- Om man använder en linje för att rita en cirkel, så kan man inte logga in med en cirkel ritad med två linjer.
- Om man ritar en cirkel högervarv så kan man inte logga in med en cirkel ritad vänstervarv.
- Ritar man en triangel i övre höger hörna så kan man inte logga in med en triangel i nedre vänster hörn.

10.3 Dokument med följdfrågor

Dokument innehållande frågor som varje deltagare avslutade experimentet med att svara på.

Följdfrågor till experiment

Hur bra stämmer varje påstående, svara med en skala från 1 till 5 där 1 motsvarar att det inte stämmer och 5 att det stämmer mycket väl.
Grafiska samt teckenbaserade lösenord syftar endast på de som användes vid tidigare utfört experiment.

Fråga 1:
Jag upplever teckenbaserade lösenord är användarvänliga.

Fråga 2:
Jag upplever grafiska lösenord är användarvänliga.

Fråga 3:
Jag tyckte det var svårt att återupprepa teckenbaserade lösenord.

Fråga 4:
Jag tyckte det var svårt att återupprepa grafiska lösenord.

Fråga 5:
Jag upplevde teckenbaserade lösenords säkerhet som god.

Fråga 6:
Jag upplevde grafiska lösenords säkerhet som god.

Fråga 7:
Jag hade kunnat tänka mig att använda grafiska lösenord som autentisering vid inloggning av en dator.

Fråga 8:
Jag hade kunnat tänka mig att använda teckenbaserade lösenord som autentisering vid inloggning av en dator.

10.4 Samtliga grafiska lösenordssträngar

Denna del innehåller tabeller med samtliga grafiska lösenord som genererades under experimentet.

10.4.1 Diagonal

Lösenordssträngar som genererades vid grafiska lösenordet diagonal:

123469++431264
112475++457314
146458++417-293
135472++457341
147456++431284
175443++428356
113460++442329
11712345++313307
124454++421342
123444++406337

10.4.2 Romb

Lösenordssträngar som genererades vid grafiska lösenordet romb:

16232251+++-++-+--2325577059853803780-601-581
19812351+++-++-+--2655957981934813691-610-792
16719243+++-++-+--2627625854767848685-587-800
15517261+++-++-+--1832755368779928600-783-521
16726283+++-++-+--17517370561106875338-785-464
16917308+++-++-+--3930777765955849734-506-899
15425240+++-++-+--2229575957848698655-410-727
14183287++-++++---9631627958-1226-752236809-678
17328257+++-++-+--85546459452153798-111-85997
115176276++-++++---1658528252-590-760844727-847

10.4.3 Cirkel

Lösenordssträngar som genererades vid grafiska lösenordet cirkel:

19919376+++-++-+++-+10610671213549130695153331607877
13674230++-++++---2832703549-1064-549812625-459
17517397+++-++-+--2079126787813571145 9-696-753
17917381+++-++-+--1979111678773931519-1045-654
18520289+++-++-+--26511072960333822348-1058-804
17218280+++-++-+--1944974659147758580-916-617
18619411+++-++-+--306210410096229994548-578-484
110414371+++-++-+--25721145885249820358-807-681
18847239++--+-++-+--20336664946-1890694595-879-551
116173373++-++++---206410210963-780-437580531-815

10.4.4 Kryss

Lösenordssträngar som genererades vid grafiska lösenordet kryss:

2884032338169+++-++4615509681031346
2414232188176++++168159795350
28104816133135++++117114945340
212801211102101++++76851038505
21787182410291++++8474877354
2129172017143142+++-+++-5467111798028873050330
29931421126116++++10196887855
23114714126131++++82111868442
2263013323135137++-+++-+16104120104-473-860-5636-375
2281111118132134++++1021161017501

10.5 C# kod för grafiska lösenorden

C# kod för grafiska lösenord.

10.5.1 Lösenordsklass

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;


namespace Paint
{
class pwObj
{

// Errormargins
private int errorMarginStartXY = 30; // Px
private double errorMarginMaxLineLengths = 1.25; //%
private double errorMarginMinLineLengths = 0.75; //%
private int errorMarginAcceptedDirectionChangeErrors = 2; // Accepted errors (mod
value later)
private int errorMarginAcceptedLengthBetweenChanges = 85; // px
private int errorMarginAcceptedTiltValue = 550; // Tilted value


// Input values for algorithm use
private List<List<int>> sublineLengths = new List<List<int>>();
private List<List<string>> sublineDirection = new List<List<string>>();
private List<int> sublineDirectionValue = new List<int>();

// Login ---------------------------------------------------------------------------

// Number of total lines
private int nrOfLines;

// Start- and endpoints for each line
private List<int> startXs = new List<int>();
private List<int> startYs = new List<int>();

// Each lines length
private List<int> lineLengths = new List<int>();

// Subline(v2) direction algorithm values
private List<int> directionChangeAtPercent = new List<int>();
private List<string> directionChange = new List<string>();
private List<int> lengthBetweenChanges = new List<int>();
private List<float> avgDirectionValueBetweenChanges = new List<float>();

// Subline(v2) direction compare values
private List<List<int>> directionChangeAtPercentTable = new List<List<int>>();
private List<List<string>> directionChangeTable = new List<List<string>>();
private List<List<int>> lengthBetweenChangesTable = new List<List<int>>();
private List<List<float>> avgDirectionValueBetweenChangesTable = new
List<List<float>>();

// Direction data list/table
private List<List<string>> directionBetweenTable = new List<List<string>>();
private List<List<float>> directionValueBetweenTable = new List<List<float>>();
// Login end -----------------------------------------------------------------------
```

```
// Registrer ------------------------------------------------------------------------

// Number of total lines
private int nrOfLinesREG;

// Start- and endpoints for each line
private List<int> startXsREG = new List<int>();
private List<int> startYsREG = new List<int>();

// each lines length
private List<int> lineLengthsREG = new List<int>();

// Subline(v2) direction compare values
private List<List<int>> directionChangeAtPercentTableREG = new List<List<int>>();
private List<List<string>> directionChangeTableREG = new List<List<string>>();
private List<List<int>> lengthBetweenChangesTableREG = new List<List<int>>();
private List<List<float>> avgDirectionValueBetweenChangesTableREG = new
List<List<float>>();

// Direction data list/table
private List<List<string>> directionBetweenTableREG = new List<List<string>>();
private List<List<float>> directionValueBetweenTableREG = new List<List<float>>();

// Registrer end --------------------------------------------------------------------

// Holders/helpers
private int lengthBetweenChangesHolder = 0;
private int avgDirectionValueBetweenChangesHolder = 0;
private int lastChangeIndex = 0;

// returnvalue
private bool returnValue = false;


public pwObj()
{}

public bool inputValue(int nrOfLines, int startX, int startY, int lineLengths)
{
this.nrOfLines++;
this.startXs.Add(startX);
this.startYs.Add(startY);
this.lineLengths.Add(lineLengths);
return true;
}

public bool inputSubLineValues(int sublineDirectionValues, string sublineDirections,
int length, int currentLine)
{
if (this.sublineLengths.Count() < currentLine)
{
this.sublineLengths.Add(new List<int>());
}

this.sublineLengths.ElementAt((currentLine - 1)).Add(length);
```

```
if (this.sublineDirection.Count() < currentLine)
{
this.sublineDirection.Add(new List<string>());
}
this.sublineDirection.ElementAt((currentLine - 1)).Add(sublineDirections);
this.sublineDirectionValue.Add(sublineDirectionValues);

return true;
}

public bool enhanceLineDirection(int currentLine) // mouseUp
{
int errorMarginSkipStartAndTail = 15; // Skip first X and last X pixels in each line.
Keep it low or fast drawing will ruin algorihtm
string currentDirection = "++";
List<string> nextDirection = new List<string>();
int tempSum = 0;
float tempX = 0;
float tempY = 0;
bool currentWon = false;
List<int> winner = new List<int>();
List<int> neighborDirections = new List<int>(); // ++ +- -+ --

// Set to default
for (int i = 0; i < 4; i++)
{
neighborDirections.Add(0);
}

for (int lineIndex = errorMarginSkipStartAndTail; lineIndex <
(this.sublineLengths.ElementAt(currentLine).Count() - errorMarginSkipStartAndTail);
lineIndex++)
{
for (int offset = (lineIndex - errorMarginSkipStartAndTail); offset <= (lineIndex +
errorMarginSkipStartAndTail); offset++)
{
tempSum += this.sublineLengths.ElementAt(currentLine)[offset];
if (this.sublineDirection.ElementAt(currentLine)[offset] == "++")
{
neighborDirections[0] += this.sublineLengths.ElementAt(currentLine)[offset];
}
else if (this.sublineDirection.ElementAt(currentLine)[offset] == "+-")
{
neighborDirections[1] += this.sublineLengths.ElementAt(currentLine)[offset];
}
else if (this.sublineDirection.ElementAt(currentLine)[offset] == "-+")
{
neighborDirections[2] += this.sublineLengths.ElementAt(currentLine)[offset];
}
else if (this.sublineDirection.ElementAt(currentLine)[offset] == "--")
{
neighborDirections[3] += this.sublineLengths.ElementAt(currentLine)[offset];
}
}
for (int i = 0; i < 4; i++)
{
if (neighborDirections.Max() == neighborDirections[i])
{
winner.Add(i);
}
}
```

```
for (int i = 0; i < winner.Count(); i++)
{
if (winner[i] == 0)
{
if (currentDirection != "++")
{
nextDirection.Add("++");
}
else
{
currentWon = true;
}
}
else if (winner[i] == 1)
{
if (currentDirection != "+-")
{
nextDirection.Add("+-");
}
else
{
currentWon = true;
}
}
else if (winner[i] == 2)
{
if (currentDirection != "-+")
{
nextDirection.Add("-+");
}
else
{
currentWon = true;
}
}
else if (winner[i] == 3)
{
if (currentDirection != "--")
{
nextDirection.Add("--");
}
else
{
currentWon = true;
}
}
}

if (currentWon == false)
{
tempX = 0;
tempY = 0;
```

```
for (int walker = this.lastChangeIndex; walker < lineIndex; walker++)
{
tempX += this.sublineLengths.ElementAt(currentLine).ElementAt(walker); // variable to
% lenght
lengthBetweenChangesHolder +=
this.sublineLengths.ElementAt(currentLine).ElementAt(walker); // length
avgDirectionValueBetweenChangesHolder += sublineDirectionValue.ElementAt(walker); //
avg tilt
}

tempY = this.sublineLengths.ElementAt(currentLine).Sum();
this.directionChangeAtPercent.Add(Convert.ToInt32((100 / tempY) * tempX));

lengthBetweenChanges.Add(lengthBetweenChangesHolder);
if ((lineIndex - this.lastChangeIndex) != 0) // prevent divided by zero
{
avgDirectionValueBetweenChanges.Add(avgDirectionValueBetweenChangesHolder / (lineIndex
- this.lastChangeIndex)); // divided by 0
}
else
{
avgDirectionValueBetweenChanges.Add(avgDirectionValueBetweenChangesHolder);
}
this.directionChange.Add(currentDirection);
this.lastChangeIndex = lineIndex;

if (nextDirection.Count() > 0)
{
currentDirection = nextDirection.ElementAt(0);
}
}


// Reset values for next run
currentWon = false;
nextDirection.Clear();
neighborDirections.Clear();
lengthBetweenChangesHolder = 0;
avgDirectionValueBetweenChangesHolder = 0;
winner.Clear();

for (int i = 0; i < 4; i++)
{
neighborDirections.Add(0);
}

tempX = 0;
tempY = 0;


}

for (int cleanUp = this.lastChangeIndex; cleanUp <
(this.sublineLengths.ElementAt(currentLine).Count() - errorMarginSkipStartAndTail);
cleanUp++)
{
lengthBetweenChangesHolder +=
this.sublineLengths.ElementAt(currentLine).ElementAt(cleanUp); // length
```

```
avgDirectionValueBetweenChangesHolder += sublineDirectionValue.ElementAt(cleanUp);
//avg tilt
}
this.directionChange.Add(currentDirection);
this.directionChangeAtPercent.Add(100);
this.lengthBetweenChanges.Add(lengthBetweenChangesHolder);
if (((this.sublineDirection.ElementAt(currentLine).Count() -
errorMarginSkipStartAndTail) - this.lastChangeIndex) != 0)
{
this.avgDirectionValueBetweenChanges.Add(avgDirectionValueBetweenChangesHolder /
((this.sublineDirection.ElementAt(currentLine).Count() - errorMarginSkipStartAndTail)
- this.lastChangeIndex)); //Divide by zero
}
else
{
this.avgDirectionValueBetweenChanges.Add(avgDirectionValueBetweenChangesHolder);
}
this.lastChangeIndex = 0;

this.directionChangeTable.Add(new List<string>(this.directionChange));
this.directionChangeAtPercentTable.Add(new List<int>(this.directionChangeAtPercent));
this.lengthBetweenChangesTable.Add(new List<int>(this.lengthBetweenChanges));
this.avgDirectionValueBetweenChangesTable.Add(new
List<float>(this.avgDirectionValueBetweenChanges));

this.resetAlgorithmValues();

return true;
}


// Function only used to check values
public void printer()
{
Console.WriteLine("---<direction change>---");
for (int i = 0; i < this.directionChangeTable.Count(); i++)
{
for (int j = 0; j < this.directionChangeTable.ElementAt(i).Count(); j++)
{
Console.WriteLine("Line: " + (i + 1) + " Subline: " + (j + 1) + " Value: " +
this.directionChangeTable.ElementAt(i).ElementAt(j));
}
Console.WriteLine();
}
Console.WriteLine("---<direction change at %>---");
for (int i = 0; i < this.directionChangeAtPercentTable.Count(); i++)
{
for (int j = 0; j < this.directionChangeAtPercentTable.ElementAt(i).Count(); j++)
{
Console.WriteLine("Line: " + (i + 1) + " Subline: " + (j + 1) + " Value: " +
this.directionChangeAtPercentTable.ElementAt(i).ElementAt(j));
}
Console.WriteLine();
}
Console.WriteLine("---<length between changes>---");
for (int i = 0; i < this.lengthBetweenChangesTable.Count(); i++)
{
for (int j = 0; j < this.lengthBetweenChangesTable.ElementAt(i).Count(); j++)
```

```
{
Console.WriteLine("Line: " + (i + 1) + " Subline: " + (j + 1) + " Length: " +
this.lengthBetweenChangesTable.ElementAt(i).ElementAt(j));
}
Console.WriteLine();
}
Console.WriteLine("---<avg lutning between changes>---");
for (int i = 0; i < this.avgDirectionValueBetweenChangesTable.Count(); i++)
{
for (int j = 0; j < this.avgDirectionValueBetweenChangesTable.ElementAt(i).Count();
j++)
{
Console.WriteLine("Line: " + (i + 1) + " Subline: " + (j + 1) + " Lutning: " +
this.avgDirectionValueBetweenChangesTable.ElementAt(i).ElementAt(j));
}
Console.WriteLine();
}
Console.WriteLine("---<sublinelengths>---");
for (int i = 0; i < this.sublineLengths.Count(); i++)
{
for (int j = 0; j < this.sublineLengths.ElementAt(i).Count(); j++)
{
Console.WriteLine("Line: " + (i + 1) + " Subline: " + (j + 1) + " Length: " +
this.sublineLengths.ElementAt(i).ElementAt(j));
}
Console.WriteLine();
}
Console.WriteLine("---<start X>---");
for (int i = 0; i < this.startXs.Count(); i++)
{
Console.WriteLine("Line: " + (i + 1) + " X: " + this.startXs.ElementAt(i));
}
Console.WriteLine("---<start Y>---");
for (int i = 0; i < this.startYs.Count(); i++)
{
Console.WriteLine("Line: " + (i + 1) + " Y: " + this.startYs.ElementAt(i));
}
}

public void registrer()
{
this.printer();
// Number of total lines
this.nrOfLinesREG = this.nrOfLines;

// Start- and endpoints for each line
this.startXsREG = new List<int>(this.startXs);
this.startYsREG = new List<int>(this.startYs);

// each lines length
this.lineLengthsREG = new List<int>(this.lineLengths);

// Subline(v2) direction compare values
this.directionChangeAtPercentTableREG = new
List<List<int>>(this.directionChangeAtPercentTable);
this.directionChangeTableREG = new List<List<string>>(this.directionChangeTable);
this.lengthBetweenChangesTableREG = new
List<List<int>>(this.lengthBetweenChangesTable);
this.avgDirectionValueBetweenChangesTableREG = new
List<List<float>>(this.avgDirectionValueBetweenChangesTable);
```

```
for (int x = 0; (x < this.directionChangeTable.Count()) && (x <
this.directionChangeTableREG.Count()); x++)
{
for (int y = 0; (y < (this.directionChangeTable.ElementAt(x).Count()) +
sprinterFirstChar) && (y < (this.directionChangeTableREG.ElementAt(x).Count()) +
sprinterFirstCharREG); y++)
{
firstMatch = false;
secondMatch = false;

if ((y + sprinterFirstChar) < this.directionChangeTable.ElementAt(x).Count())
{
if (this.directionChangeTable.ElementAt(x).ElementAt((y +
sprinterFirstChar)).ElementAt(0) == '0')
{
firstMatch = true;
sprint++;
}
}
else if ((y + sprinterSecondChar) < this.directionChangeTable.ElementAt(x).Count())
{
if (this.directionChangeTable.ElementAt(x).ElementAt((y +
sprinterSecondChar)).ElementAt(1) == '0')
{
secondMatch = true;
sprint++;
}
}

if ((y + sprinterFirstCharREG) < this.directionChangeTableREG.ElementAt(x).Count())
{
if (this.directionChangeTableREG.ElementAt(x).ElementAt((y +
sprinterFirstCharREG)).ElementAt(0) == '0')
{
firstMatch = true;
sprintREG++;
}
}
if ((y + sprinterSecondCharREG) < this.directionChangeTableREG.ElementAt(x).Count())
{
if (this.directionChangeTableREG.ElementAt(x).ElementAt((y +
sprinterSecondChar)).ElementAt(1) == '0')
{
secondMatch = true;
sprintREG++;
}
}

if (((y + sprint) < this.directionChangeTable.ElementAt(x).Count()) && ((y +
sprintREG) < this.directionChangeTableREG.ElementAt(x).Count()))
{
if (this.directionChangeTable.ElementAt(x).ElementAt((y + sprint)).ElementAt(0) ==
this.directionChangeTableREG.ElementAt(x).ElementAt((y + sprintREG)).ElementAt(0))
{
firstMatch = true;
}
passwordString += this.directionChangeTableREG.ElementAt(x).ElementAt(y +
sprintREG).ElementAt(0);
}
```

```
if (((y + sprint) < this.directionChangeTable.ElementAt(x).Count()) && ((y +
sprintREG) < this.directionChangeTableREG.ElementAt(x).Count()))
{
if (this.directionChangeTable.ElementAt(x).ElementAt((y + sprint)).ElementAt(1) ==
this.directionChangeTableREG.ElementAt(x).ElementAt((y + sprintREG)).ElementAt(1))
{
secondMatch = true;
}
passwordString += this.directionChangeTableREG.ElementAt(x).ElementAt(y +
sprintREG).ElementAt(1);
}

if (firstMatch != true || secondMatch != true)
{
errorTemp++;
}
}
}
if (errorTemp > (this.errorMarginAcceptedDirectionChangeErrors - 1))
{
errorTemp = 0; //reset
for (int x = 0; (x < this.directionChangeTable.Count()) && (x <
this.directionChangeTableREG.Count()); x++)
{
for (int y = 0; (y < this.directionChangeTable.ElementAt(x).Count()) && (y <
this.directionChangeTableREG.ElementAt(x).Count()); y++)
{
if (this.directionChangeTable.ElementAt(x).ElementAt(y).ElementAt(0) ==
this.directionChangeTableREG.ElementAt(x).ElementAt(y).ElementAt(0))
{
firstMatch = true;
}
if (this.directionChangeTable.ElementAt(x).ElementAt(y).ElementAt(1) ==
this.directionChangeTableREG.ElementAt(x).ElementAt(y).ElementAt(1))
{
secondMatch = true;
}
}
if (firstMatch != true || secondMatch != true)
{
errorTemp++;
}
}

if (errorTemp > (this.errorMarginAcceptedDirectionChangeErrors))
{
this.returnValue = false;
}
}

errorTemp = 0;
for (int x = 0; (x < this.lengthBetweenChangesTable.Count()) && (x <
this.lengthBetweenChangesTableREG.Count()); x++)
{
for (int y = 0; (y < this.lengthBetweenChangesTable.ElementAt(x).Count()) && (y <
this.lengthBetweenChangesTableREG.ElementAt(x).Count()); y++)
{
```

```
if ((this.lengthBetweenChangesTable.ElementAt(x).ElementAt(y) +
this.errorMarginAcceptedLengthBetweenCahnges) <
this.lengthBetweenChangesTableREG.ElementAt(x).ElementAt(y)
|| (this.lengthBetweenChangesTable.ElementAt(x).ElementAt(y) -
this.errorMarginAcceptedLengthBetweenCahnges) >
this.lengthBetweenChangesTableREG.ElementAt(x).ElementAt(y))
{
errorTemp++;
}
passwordString += this.lengthBetweenChangesTableREG.ElementAt(x).ElementAt(y);
}
}
if (errorTemp > 1)
{
this.returnValue = false;
}
errorTemp = 0;
for (int x = 0; (x < this.avgDirectionValueBetweenChangesTable.Count()) && (x <
this.avgDirectionValueBetweenChangesTableREG.Count()); x++)
{
for (int y = 0; (y < this.avgDirectionValueBetweenChangesTable.ElementAt(x).Count())
&& (y < this.avgDirectionValueBetweenChangesTableREG.ElementAt(x).Count()); y++)
{
if ((this.avgDirectionValueBetweenChangesTable.ElementAt(x).ElementAt(y) +
this.errorMarginAcceptedTiltValue) <
this.avgDirectionValueBetweenChangesTableREG.ElementAt(x).ElementAt(y)
|| (this.avgDirectionValueBetweenChangesTable.ElementAt(x).ElementAt(y) -
this.errorMarginAcceptedTiltValue) >
this.avgDirectionValueBetweenChangesTableREG.ElementAt(x).ElementAt(y))
{
errorTemp++;
}
passwordString +=
this.avgDirectionValueBetweenChangesTableREG.ElementAt(x).ElementAt(y);
}
}
if (errorTemp > 2)
{
this.returnValue = false;
}

if (this.returnValue == false)
{
passwordString = "-1";
}

return passwordString;
}

public void resetAlgorithmValues()
{
this.sublineLengths.Clear();

this.directionChange.Clear();
this.directionChangeAtPercent.Clear();
this.lengthBetweenChanges.Clear();
this.avgDirectionValueBetweenChanges.Clear();
this.sublineLengths.Clear();
}
```

```
public void clear()
{
this.sublineLengths.Clear();
this.sublineDirection.Clear();

this.nrOfLines = 0;

this.startXs.Clear();
this.startYs.Clear();

this.lineLengths.Clear();

this.directionChange.Clear();
this.directionChangeAtPercent.Clear();
this.lengthBetweenChanges.Clear();
this.avgDirectionValueBetweenChanges.Clear();

this.directionChangeAtPercentTable.Clear();
this.directionChangeTable.Clear();
this.lengthBetweenChangesTable.Clear();
this.avgDirectionValueBetweenChangesTable.Clear();
}
}
}
```

10.5.2 Klass som arbetar mot gränssnittet

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Paint
{
public partial class Form1 : Form
{
Graphics graphics; //Graphic-class
Pen pencil = new Pen(Color.Black, 1); //Pen-class, color, brush, size etc.
Point start = new Point(0, 0); //start
Point end = new Point(0, 0); //end
bool mouseDown = false;
bool visibleLines = true;

pwObj passwordObj = new pwObj();

int nrOfLines = 0;

int startX = -1;
int startY = -1;

List<int> allLineLengths = new List<int>();
int currentLineLength;
int lastIndexX = -1;
int lastIndexY = -1;

int sublineDirectionEnhance = 1000;

//misc
int tempInt = -1;

string lastXdir = "0";
string lastYdir = "0";


public Form1()
{
InitializeComponent();
}

private void pwBox_MouseMove(object sender, MouseEventArgs e)
{
if (this.mouseDown == true)
{
if ((e.Location.X >= 0) && (e.Location.X <= (pwBox.Width - 5)) && (e.Location.Y >= 0)
&& (e.Location.Y <= (pwBox.Height - 5)))
{
if ((this.lastIndexX != e.Location.X) || (this.lastIndexY != e.Location.Y))
{
tempInt = this.pythagora(e.Location.X, e.Location.Y, lastIndexX, lastIndexY);
if (tempInt != -1)
{
```

```
this.currentLineLength += tempInt;
}

this.lastIndexX = e.Location.X;
this.lastIndexY = e.Location.Y;

}
}
end = e.Location;
this.graphics = pwBox.CreateGraphics();
this.graphics.DrawLine(pencil, start, end);

}
this.start = this.end;

}

private void pwBox_MouseDown(object sender, MouseEventArgs e)
{
this.start = e.Location;
if (e.Button == MouseButtons.Left)
{
this.startX = e.X;
this.startY = e.Y;
this.nrOfLines++;
this.mouseDown = true;
}
}

private void pwBox_MouseUp(object sender, MouseEventArgs e)
{
this.mouseDown = false;
this.passwordObj.inputValue((this.nrOfLines - 1), this.startX, this.startY,
this.currentLineLength);
this.passwordObj.enhanceLineDirection((this.nrOfLines - 1));
this.allLineLengths.Add(this.currentLineLength);
this.currentLineLength = 0;
this.lastIndexX = -1;
this.lastIndexY = -1;
this.resetAlgorithmValuesForm();
}

// (Clear button)
private void clearButton_Click(object sender, EventArgs e)
{
this.passwordObj.clear();
this.resetAlgorithmValuesForm();
pwBox.Refresh();
}


// (Login button)
private void loginButton_Click(object sender, EventArgs e)
{
```

```
string loginResult = this.passwordObj.login();
if (loginResult != "-1")
{
LoginText.Text = "Succes!           PW:" + loginResult;
userNameBox.Text = loginResult;
}
else
{
LoginText.Text = "Failed!";
}
}


// (Registrer button)
private void registerButton_Click(object sender, EventArgs e)
{
Console.WriteLine("Registrer-button");

this.passwordObj.registrer();
this.passwordObj.clear();
this.resetAlgorithmValuesForm();
pwBox.Refresh();
}

private void invisLinesRButton_CheckedChanged(object sender, EventArgs e)
{}

// (Invis-lines radio-button)
private void invisLinesRButton_MouseClick(object sender, MouseEventArgs e)
{
if (this.visibleLines == true)
{
this.pencil.Color = Color.White;
invisLinesRButton.Checked = false;
this.visibleLines = false;
}
else if (this.visibleLines == false)
{
this.pencil.Color = Color.Black;
invisLinesRButton.Checked = true;
this.visibleLines = true;
}
}

private void resetAlgorithmValuesForm()
{
this.nrOfLines = 0;
this.startX = -1;
this.startY = -1;
this.lastIndexX = -1;
this.lastIndexY = -1;
this.currentLineLength = 0;
this.allLineLengths.Clear();
}

private int pythagora(int newX, int newY, int lastX, int lastY)
{
int result = -1;
```

```
if (lastX > 0 && lastY > 0)
{
int xDif = 0;
int yDif = 0;
float floatX = 0;
float floatY = 0;

int sublineDirectionValue = 0;
double floatResultHolder = 0;
string sublineDirection = "";

if (newY > lastY)
{
yDif = (newY - lastY); //-Y
sublineDirection += "+";
this.lastYdir = "+";
}
else if (newY < lastY)
{
yDif = (lastY - newY); //+Y
sublineDirection += "-";
this.lastYdir = "-";
}
else
{
sublineDirection += this.lastYdir;
}

if (newX > lastX)
{
xDif = (newX - lastX); //-X
sublineDirection += "+";
this.lastXdir += "+";
}
else if (newX < lastX)
{
xDif = (lastX - newX); //+X
sublineDirection += "-";
this.lastXdir += "-";
}
else
{
sublineDirection += this.lastXdir;
}

floatX = (newX - lastX);
floatY = (newY - lastY);

// a^2 b^2
xDif = (xDif * xDif);
yDif = (yDif * yDif);

// square root of (a^2*b^2)
result = Convert.ToInt32(Math.Floor(Math.Sqrt((xDif + yDif))));
```

```
if (result != 0)
{
floatResultHolder = (Math.Asin(((Math.Sin(90) / result) * (newY - lastY))) *
sublineDirectionEnhance); // sinussatsen: arcsin(sin90/hyp) * yDif
}
sublineDirectionValue = Convert.ToInt32(Math.Floor(floatResultHolder));
this.passwordObj.inputSubLineValues(sublineDirectionValue, sublineDirection, result,
this.nrOfLines);

}
return result;
}

private void Form1_Load(object sender, EventArgs e)
{

}
}
}
```

10.5.3 Gränssnitt

```
namespace Paint
{
partial class Form1
{
/// <summary>
/// Required designer variable.
/// </summary>
private System.ComponentModel.IContainer components = null;

/// <summary>
/// Clean up any resources being used.
/// </summary>
/// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
protected override void Dispose(bool disposing)
{
if (disposing && (components != null))
{
components.Dispose();
}
base.Dispose(disposing);
}

#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
this.panel1 = new System.Windows.Forms.Panel();
this.pwBox = new System.Windows.Forms.Panel();
this.pwLable = new System.Windows.Forms.Label();
this.usernameLable = new System.Windows.Forms.Label();
this.userNameBox = new System.Windows.Forms.TextBox();
this.clearButton = new System.Windows.Forms.Button();
this.loginButton = new System.Windows.Forms.Button();
this.registerButton = new System.Windows.Forms.Button();
this.invisLinesRButton = new System.Windows.Forms.RadioButton();
this.LoginText = new System.Windows.Forms.Label();
this.SuspendLayout();
//
// panel1
//
this.panel1.Location = new System.Drawing.Point(14, 613);
this.panel1.Margin = new System.Windows.Forms.Padding(3, 4, 3, 4);
this.panel1.Name = "panel1";
this.panel1.Size = new System.Drawing.Size(708, 121);
this.panel1.TabIndex = 0;
//
// pwBox
//
this.pwBox.BackColor = System.Drawing.Color.White;
this.pwBox.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D;
this.pwBox.Cursor = System.Windows.Forms.Cursors.Hand;
this.pwBox.Location = new System.Drawing.Point(57, 150);
this.pwBox.Margin = new System.Windows.Forms.Padding(3, 4, 3, 4);
```

```
this.pwBox.Name = "pwBox";
this.pwBox.Size = new System.Drawing.Size(466, 151);
this.pwBox.TabIndex = 1;
this.pwBox.MouseDown += new
System.Windows.Forms.MouseEventHandler(this.pwBox_MouseDown);
this.pwBox.MouseMove += new
System.Windows.Forms.MouseEventHandler(this.pwBox_MouseMove);
this.pwBox.MouseUp += new System.Windows.Forms.MouseEventHandler(this.pwBox_MouseUp);
//
// pwLable
//
this.pwLable.AutoSize = true;
this.pwLable.Font = new System.Drawing.Font("Microsoft Sans Serif", 10.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.pwLable.Location = new System.Drawing.Point(54, 115);
this.pwLable.Name = "pwLable";
this.pwLable.Size = new System.Drawing.Size(82, 17);
this.pwLable.TabIndex = 2;
this.pwLable.Text = "Password:";
//
// usernameLable
//
this.usernameLable.AutoSize = true;
this.usernameLable.Font = new System.Drawing.Font("Microsoft Sans Serif", 10.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.usernameLable.Location = new System.Drawing.Point(54, 53);
this.usernameLable.Name = "usernameLable";
this.usernameLable.Size = new System.Drawing.Size(86, 17);
this.usernameLable.TabIndex = 3;
this.usernameLable.Text = "Username:";
//
// userNameBox
//
this.userNameBox.Location = new System.Drawing.Point(57, 73);
this.userNameBox.Name = "userNameBox";
this.userNameBox.Size = new System.Drawing.Size(165, 20);
this.userNameBox.TabIndex = 4;
//
// clearButton
//
this.clearButton.Cursor = System.Windows.Forms.Cursors.Hand;
this.clearButton.Location = new System.Drawing.Point(57, 327);
this.clearButton.Name = "clearButton";
this.clearButton.Size = new System.Drawing.Size(75, 23);
this.clearButton.TabIndex = 5;
this.clearButton.Text = "Clear";
this.clearButton.UseVisualStyleBackColor = true;
this.clearButton.Click += new System.EventHandler(this.clearButton_Click);
//
// loginButton
//
this.loginButton.Cursor = System.Windows.Forms.Cursors.Hand;
this.loginButton.Location = new System.Drawing.Point(57, 352);
this.loginButton.Name = "loginButton";
this.loginButton.Size = new System.Drawing.Size(75, 23);
this.loginButton.TabIndex = 6;
this.loginButton.Text = "Login";
this.loginButton.UseVisualStyleBackColor = true;
this.loginButton.Click += new System.EventHandler(this.loginButton_Click);
```

```
//
// registerButton
//
this.registerButton.Cursor = System.Windows.Forms.Cursors.Hand;
this.registerButton.Location = new System.Drawing.Point(138, 352);
this.registerButton.Name = "registerButton";
this.registerButton.Size = new System.Drawing.Size(75, 23);
this.registerButton.TabIndex = 7;
this.registerButton.Text = "Register";
this.registerButton.UseVisualStyleBackColor = true;
this.registerButton.Click += new System.EventHandler(this.registerButton_Click);
//
// invisLinesRButton
//
this.invisLinesRButton.AutoSize = true;
this.invisLinesRButton.Checked = true;
this.invisLinesRButton.Cursor = System.Windows.Forms.Cursors.Hand;
this.invisLinesRButton.Location = new System.Drawing.Point(138, 327);
this.invisLinesRButton.Name = "invisLinesRButton";
this.invisLinesRButton.Size = new System.Drawing.Size(92, 17);
this.invisLinesRButton.TabIndex = 8;
this.invisLinesRButton.TabStop = true;
this.invisLinesRButton.Text = "Visible lines";
this.invisLinesRButton.UseVisualStyleBackColor = true;
this.invisLinesRButton.CheckedChanged += new
System.EventHandler(this.invisLinesRButton_CheckedChanged);
this.invisLinesRButton.MouseClick += new
System.Windows.Forms.MouseEventHandler(this.invisLinesRButton_MouseClick);
//
// LoginText
//
this.LoginText.AutoSize = true;
this.LoginText.Location = new System.Drawing.Point(237, 327);
this.LoginText.Name = "LoginText";
this.LoginText.Size = new System.Drawing.Size(0, 13);
this.LoginText.TabIndex = 9;
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(7F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.BackColor = System.Drawing.Color.DimGray;
this.ClientSize = new System.Drawing.Size(688, 387);
this.Controls.Add(this.LoginText);
this.Controls.Add(this.invisLinesRButton);
this.Controls.Add(this.registerButton);
this.Controls.Add(this.loginButton);
this.Controls.Add(this.clearButton);
this.Controls.Add(this.userNameBox);
this.Controls.Add(this.usernameLable);
this.Controls.Add(this.pwLable);
this.Controls.Add(this.pwBox);
this.Controls.Add(this.panel1);
this.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.Margin = new System.Windows.Forms.Padding(3, 4, 3, 4);
this.Name = "Form1";
this.Text = "Form1";
this.Load += new System.EventHandler(this.Form1_Load);
```

```
this.ResumeLayout(false);
this.PerformLayout();

}

#endregion

private System.Windows.Forms.Panel panel1;
private System.Windows.Forms.Panel pwBox;
private System.Windows.Forms.Label pwLable;
private System.Windows.Forms.Label usernameLable;
private System.Windows.Forms.TextBox userNameBox;
private System.Windows.Forms.Button clearButton;
private System.Windows.Forms.Button loginButton;
private System.Windows.Forms.Button registerButton;
private System.Windows.Forms.RadioButton invisLinesRButton;
private System.Windows.Forms.Label LoginText;
}
}
```

10.6 C# kod för teckenbaserade lösenorden

C# kod för teckenbaserade lösenord.

10.6.1 Klass som arbetar mot gränssnittet

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace textPassword
{
public partial class Form1 : Form
{
string passwordREG = null;

public Form1()
{
InitializeComponent();
}

private void Blogin_Click(object sender, EventArgs e)
{
if (pwBox.Text == passwordREG)
{
loginReturn.Text = "Login success.   pw: " + passwordREG;
}
else
{
loginReturn.Text = "Login failed";
pwBox.Clear();
}
}

private void Bregister_Click(object sender, EventArgs e)
{
passwordREG = pwBox.Text;
pwBox.Clear();
}
}
}
```

10.6.2 Gränssnitt

```
namespace textPassword
{
partial class Form1
{
/// <summary>
/// Required designer variable.
/// </summary>
private System.ComponentModel.IContainer components = null;

/// <summary>
/// Clean up any resources being used.
/// </summary>
/// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
protected override void Dispose(bool disposing)
{
if (disposing && (components != null))
{
components.Dispose();
}
base.Dispose(disposing);
}

#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
this.label1 = new System.Windows.Forms.Label();
this.loginBox = new System.Windows.Forms.TextBox();
this.label2 = new System.Windows.Forms.Label();
this.pwBox = new System.Windows.Forms.TextBox();
this.Blogin = new System.Windows.Forms.Button();
this.Bregister = new System.Windows.Forms.Button();
this.loginReturn = new System.Windows.Forms.Label();
this.SuspendLayout();
//
// label1
//
this.label1.AutoSize = true;
this.label1.BackColor = System.Drawing.Color.DimGray;
this.label1.Font = new System.Drawing.Font("Microsoft Sans Serif", 10.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.label1.Location = new System.Drawing.Point(54, 53);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(81, 17);
this.label1.TabIndex = 0;
this.label1.Text = "Username";
//
// loginBox
//
this.loginBox.Location = new System.Drawing.Point(57, 73);
this.loginBox.Name = "loginBox";
this.loginBox.Size = new System.Drawing.Size(165, 20);
this.loginBox.TabIndex = 1;
```

```
//
// label2
//
this.label2.AutoSize = true;
this.label2.Font = new System.Drawing.Font("Microsoft Sans Serif", 10.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.label2.Location = new System.Drawing.Point(54, 115);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(77, 17);
this.label2.TabIndex = 2;
this.label2.Text = "Password";
//
// pwBox
//
this.pwBox.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.pwBox.Location = new System.Drawing.Point(57, 135);
this.pwBox.Name = "pwBox";
this.pwBox.PasswordChar = '*';
this.pwBox.Size = new System.Drawing.Size(165, 20);
this.pwBox.TabIndex = 3;
//
// Blogin
//
this.Blogin.Location = new System.Drawing.Point(57, 180);
this.Blogin.Name = "Blogin";
this.Blogin.Size = new System.Drawing.Size(75, 23);
this.Blogin.TabIndex = 4;
this.Blogin.Text = "Login";
this.Blogin.UseVisualStyleBackColor = true;
this.Blogin.Click += new System.EventHandler(this.Blogin_Click);
//
// Bregister
//
this.Bregister.Location = new System.Drawing.Point(147, 180);
this.Bregister.Name = "Bregister";
this.Bregister.Size = new System.Drawing.Size(75, 23);
this.Bregister.TabIndex = 5;
this.Bregister.Text = "Register";
this.Bregister.UseVisualStyleBackColor = true;
this.Bregister.Click += new System.EventHandler(this.Bregister_Click);
//
// loginReturn
//
this.loginReturn.AutoSize = true;
this.loginReturn.Location = new System.Drawing.Point(57, 223);
this.loginReturn.Name = "loginReturn";
this.loginReturn.Size = new System.Drawing.Size(0, 13);
this.loginReturn.TabIndex = 6;
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(7F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.BackColor = System.Drawing.Color.DimGray;
this.ClientSize = new System.Drawing.Size(666, 384);
this.Controls.Add(this.loginReturn);
this.Controls.Add(this.Bregister);
```

```
this.Controls.Add(this.Blogin);
this.Controls.Add(this.pwBox);
this.Controls.Add(this.label2);
this.Controls.Add(this.loginBox);
this.Controls.Add(this.label1);
this.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.Name = "Form1";
this.Text = "Form1";
this.ResumeLayout(false);
this.PerformLayout();

}

#endregion

private System.Windows.Forms.Label label1;
private System.Windows.Forms.TextBox loginBox;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.TextBox pwBox;
private System.Windows.Forms.Button Blogin;
private System.Windows.Forms.Button Bregister;
private System.Windows.Forms.Label loginReturn;
}
}
```