Kryptografiska hashfunktioner och säkerheten hos lagrade lösenord

Jonas Kylliäinen

Kandidatavhandling i datateknik

Handledare: Jan Westerholm

Fakulteten för naturvetenskaper och teknik

Åbo Akademi

2020

Referat

Lösenord är det vanligaste sättet för användare att autentisera sig till en dator eller datortjänst. Återanvändning av lösenord är därför inte alls ovanligt, eftersom användare hanterar idag flera olika konton. Detta orsakar ett säkerhetsproblem, eftersom ifall ett lösenord äventyras, kan det användas för att få tillgång också till andra konton. Därför är det viktigt att lösenorden hålls hemliga, och en stor betydelse i detta har formatet som lösenorden är lagrade i. Det säkraste sättet att lagra ett lösenord är att använda en kryptografisk hashfunktion för att beräkna ett hashvärde, och endast spara detta värde.

Kryptografiska hashfunktioner är mycket användbara vid lagring och verifiering av lösenord på grund av dess egenskaper. Flera olika funktioner existerar och används idag runt om världen för denna uppgift, men vissa kryptografiska hashfunktioner är bättre anpassade för detta än ändra. Dessa funktioner använder sig av nyckelsträckning för att öka på den beräkningsmässiga kostnaden av hashningen, vilket i sin tur gör hashvärdena av lösenorden mer tidskrävande att knäcka. Detta är redan nu och kommer att bli allt mer viktigare i framtiden, eftersom beräkningskraften av ny hårdvara fortsättningsvis växer.

Den använda kryptografiska hashfunktionen är dock inte den enda faktorn som inverkar på knäckbarheten av hashvärdena. Oerhört viktigt är också att ordentligt förbehandla lösenorden med tekniken saltning, för att förhindra attackerare från att knäcka hashvärden med hjälp av uppslagningstabeller. Dessa tabeller är mycket effektiva och kan dessutom skapas i förhand före ett eventuellt dataintrång ens har skett.

Även om en lämplig kryptografisk hashfunktion har använts och saltningen gjorts ordentligt, betyder det inte att hashvärdena skulle vara omöjliga att knäcka. Ifall det ursprungliga lösenordet är svagt, betyder det att attackerare kan också med hjälp av råstyrkeattacker gissa det relativt enkelt. Tjänsteleverantörer och organisationer kan i viss mån påverka detta, men slutliga ansvaret hör till användarna själva.

Nyckelord: Kryptografisk hashfunktion, hashvärde, lösenord, knäckning

Innehållsförteckning

1. Inledning

Sedan dess uppfinning på 1960-talet har lösenord varit den allmännaste metoden av autentisering för slutanvändare [1]. Lösenord används dagligen av miljarder av människor för att autentisera sig till olika tjänster, både på fritiden samt inom arbetslivet. Eftersom användare idag hanterar på tiotals olika konton, är det inte alls ovanligt att lösenord återanvänds mellan olika tjänster [2, 3, 4]. Detta orsakar i sin tur en säkerhetsrisk, eftersom ifall någon råkar få tag på ett lösenord från ett konto, kan detta eventuellt användas till för att få tillgång till andra.

På grund av detta finns det också ett incitament för hackare att hitta dessa lösenord, och därför är det otroligt viktigt att användarnas lösenord hålls hemliga. Ett stort ansvar för detta bär tjänsteleverantörerna och organisationerna som upprätthåller användarnas konton. Information om lösenordet samt annan användarinformation sparas vanligtvis i någon form i en databas. Ifall denna information äventyras i ett dataintrång, vilket är alltid en risk, är det oerhört viktigt att lösenorden är sparat i ett säkert format.

Det värsta och osäkraste formatet är att spara lösenorden i oformaterad text, eftersom det tillåter vem som helst som har tillgång till databasen att direkt läsa lösenorden [5, 6]. Även om detta är idag sällsynt bland tjänsteleverantörer, dyker det ibland upp fall där detta har skett. Till exempel Facebook blev i början av 2019 fast för att ha internt sparat lösenord av flera miljoner användare i detta format [7].

Ett annat alternativ är att kryptera lösenordet. Detta är ett säkrare alternativ jämfört med oformaterad text, eftersom krypterade lösenord går inte rakt att läsa, utan de måste först dekrypteras. Krypteringen och dekrypteringen sker med hjälp av en nyckel [8]. Problemet med detta alternativ är att nyckeln måste också lagras någonstans, eftersom den behövs varje gång då lösenord ska lagras och verifieras. Ifall ett dataintrång skulle ske, finns det en risk för att attackerarna också får tag på denna nyckel, vilket gör det möjligt att dekryptera alla lösenord, och på detta sätt få tillgång till dem [5, 6].

Det absolut vanligaste och säkraste sättet att lagra lösenord är med hjälp av kryptografiska hashfunktioner. Hur dessa funktioner fungerar och deras användning beskrivs i följande kapitel, men kort sagt tillåter de tjänsteleverantörer

att lagra lösenord i ett format där det lagrade lösenordet kan varken dekrypteras med hjälp av någon nyckel eller inverteras tillbaka till det ursprungliga lösenordet med hjälp av någon annan algoritm. På detta sätt kan ingen, inte ens tjänsteleverantören själv, rakt läsa vad lösenordet är [5].

Men även om lösenorden är lagrade i detta säkra format existerar det vissa metoder, vilka beskrivs i kapitel 3, som kan användas för att försöka få tag på de oformaterade ursprungliga lösenorden. Hur effektiva dessa metoder är beror på olika faktorer, varav den använda kryptografiska hashfunktionen spelar en viss roll.

I denna avhandling undersöks dessa faktorer och vilken roll de spelar. Flera olika varianter av kryptografiska hashfunktioner existerar, varav vissa är bättre anpassade för lagring av lösenord än andra. Funktionens inverkan på säkerheten minskar dock ifall lösenordet till exempel inte behandlas ordentligt. En viktig faktor spelar också det ursprungliga lösenordet, ifall det är svagt minskar andra faktorernas betydelse på säkerheten.

2. Vad är en kryptografisk hashfunktion?

En hashfunktion är en beräkningseffektiv funktion som returnerar en binär sträng med en given längd för indata av godtycklig längd. Värdet som hashfunktionen returnerar kallas för ett hashvärde [8]. Indata kan vara av vilken typ som helst, eftersom hashfunktionen bildar hashvärdet från den binära strängen som bygger upp själva data. Hashning är termen som används för att beskriva denna process.

Kryptografiska hashfunktioner är en speciell variant av hashfunktioner med ett antal egenskaper som gör dem mycket användbara inom flera olika områden inom datavetenskapen, speciellt inom informationssäkerhet och kryptografi. [8, 9]

Hashvärdet som kryptografiska hashfunktioner returnerar är alltid av en viss fixerad längd, oberoende på storleken av indatat [9]. Längden av hashvärdet beror på algoritmen som används, men vanliga längder är bland annat 128- och 256-bitar [6].

En annan viktig egenskap hos kryptografiska hashfunktioner är att de är kollisionsresistenta. Detta betyder att det anses vara beräkningsmässigt omöjligt att hitta två olika indata som skulle resultera i samma hashvärde. Men även om en funktion är kollisionsresistent så betyder det inte att kollisioner inte skulle finnas, det är bara extremt svårt att finna dem [8, 10]. Kollisioner har dock upptäckts i vissa äldre funktioner. Detta och mer om kollisionsresistens kommer att tas upp senare i denna avhandling.

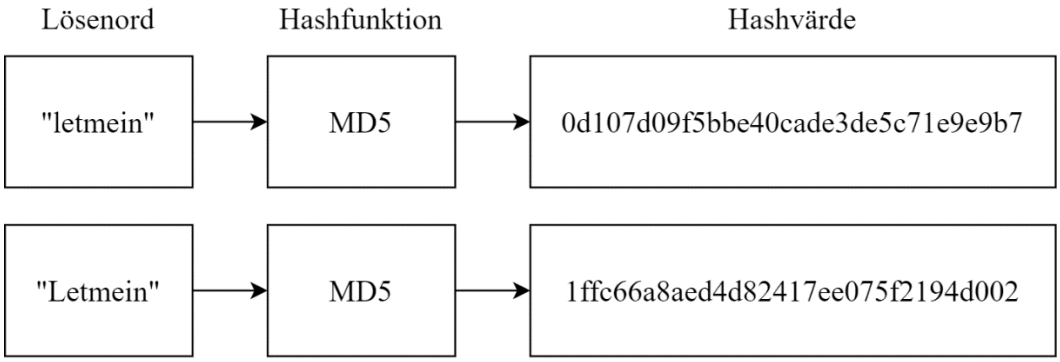
Den viktigaste egenskapen hos kryptografiska hashfunktioner är att de är enkelriktade. Funktionerna kan alltså snabbt beräkna hashvärdet för en viss mängd indata, men inverteringen av denna process anses vara beräkningsmässigt omöjligt [9]. Detta, tillsammans med lavineffekten, betyder att det utifrån hashvärdet varken går att beräkna eller dra slutsatser om de ursprungliga indata.

2.1. Användning i lagring och verifiering av lösenord

De kryptografiska hashfunktionernas egenskaper gör dem mycket användbara vid lagringen av lösenord. Orsaken är att de gör det möjligt att spara lösenorden i ett format som skyddar själva lösenordet ifall användarinformation äventyras,

samtidigt som det enkelt går att kontrollera ifall användaren matat in rätt lösenord [11].

När en användare till exempel skapar ett nytt konto och anger ett lösenord, körs detta lösenord först igenom en kryptografisk hashfunktion, vilket ger som output ett hashvärde. Exempel på detta kan ses i figur 1. Det resulterande hashvärdet lagras sedan i en databas. När användaren igen ska logga in till sitt konto körs strängen användaren matat in som lösenord genom samma hashfunktion, och det resulterande hashvärdet jämförs sedan med hashvärdet lagrat i databasen. Ifall det resulterad hashvärdet är ekvivalent med det i databasen, har användaren matat in rätt lösenord och inloggningen godkänns. Däremot ifall hashvärdena inte är ekvivalenta, godkänns inte inloggningen och användaren bes ge ett nytt lösenord [6, 11].



Figur 1 – Exempel på hashningsprocessen av två enkla lösenord med MD5

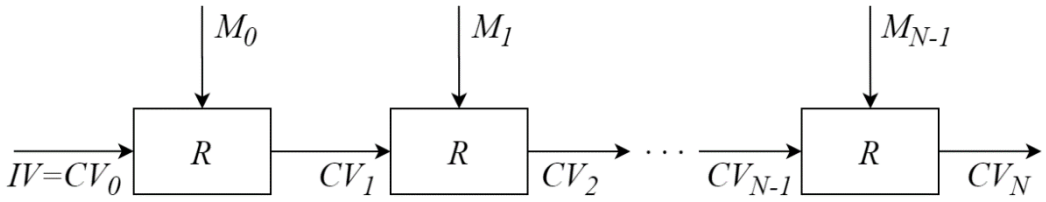
Ifall det skulle ske ett dataintrång i databasen, skulle hackarna endast få tag på hashvärdena, inte själva lösenorden. Ur figur 1 går det att se hur hashvärden får lösenord kan se ut. På grund av lavineffekten ser hashvärdena i figuren helt olika ut, även om skillnaden mellan de ursprungliga lösenorden är bara ett tecken. Eftersom det utifrån hashvärden rakt varken går att se eller beräkna de ursprungliga lösenorden, är de mycket säkrare än om lösenorden skulle vara lagrade i något annat format.

Detta betyder dock inte att det skulle vara omöjligt för en hackare att ta reda på de ursprungliga lösenorden. Båda lösenorden i figur 1 är mycket svaga, vilket i sin tur betyder att deras motsvarande hashvärden går enkelt att knäcka [5]. För detta finns det olika metoder, vilka kommer att tas upp i kapitel 3 av denna avhandling.

2.2. Traditionella strukturen av kryptografiska hashfunktionen

Strukturen på kryptografiska hashfunktioner varierar beroende på algoritmen i frågan, men de flesta av de mest använda funktionerna baserar sig på en iterativ struktur, som kan ses i figur 2. Merkle-Damgård konstruktionen, en allmän designmetod för kryptografiska hashfunktioner, bygger på denna struktur [12].

I denna struktur delas de ursprungliga indata N först in i block ($M_0, M_1 ... M_{N-1}$), där varje block består av b stycken bitar [9]. Ifall det sista blocket blir mindre än b bitar långt måste till bitar fyllas på tills blocket blir av korrekt storlek. Vanligtvis brukar dessa bitar vara nollor. Storleken på b beror på hashfunktionen som används.



Figur 2 – Den traditionella iterativa strukturen av kryptografiska hashfunktionen [9]

Efter indelningen startar funktionen en iterativ process där en enkelriktad kompressionsfunktion R används. Kompressionsfunktionen R tar in i varje iteration ett block M_{i-1} samt outputen från den tidigare iterationen CV_{i-1}. Både blocket M_{i-1} och outputen CV_{i-1} består av n bitar. I den första iterationen med det första blocket används istället en initialiseringsvektor (IV), eftersom ingen output existerar ännu i detta skede [9]. Initialiseringsvektorn består av ett eller flera kontanta värden som är bestämda i förhand, och varierar mellan olika kryptografiska hashfunktioner. Till exempel SHA-256, en mycket allmän hashfunktion, har åtta initialiseringsvektorer, vilka består av de 32 första bitarna av kvadratroten av de 8 första primtalen [13].

Efter att hashfunktionen itererat genom varje block fås värdet CV_N, vilket är det slutliga hashvärdet som funktionen ger som output. Storleken på hashvärdet är n bitar.

2.3. Enkelriktade kompressionsfunktionen

Kompressionsfunktionen spelar en viktig roll i kryptografiska hashfunktioner, eftersom de utför de egentliga beräkningarna. En kompressionsfunktion tar in 2 inputs av storleken b bitar och ger som output en binär sträng med samma storlek b bitar [8]. Funktionen har fått sitt namn från att den komprimerar två strängar till

en. Komprimeringen är dock enkelriktad, det vill säga den resulterande binära strängen går inte att dekomprimera tillbaka på något enkelt sätt.

Hur beräkningarna sker beror på kompressionsfunktionen som används, vilket i sin tur varierar mellan olika kryptografiska hashfunktioner. Gemensamt för kompressionsfunktionerna är att de först delar in blocken till mindre delar, vilka kallas för ord. Dessa ord bearbetas därefter av skilda logiska funktioner, ett för varje ord. I dessa logiska funktioner sker det ett antal olika bitvisa operationer, främst ”*exklusivt eller*” (XOR) samt ”*och*” (AND). Olika skiftningar är också vanliga. Outputen av kompressionsfunktionen är en sammansättning av orden som logiska funktionerna gett som output [8, 14, 15].

Kompressionsfunktionen orsakar också den så kallade lavineffekten i hashfunktionen, det vill säga även en liten ändring i indatat, till exempel en bits skillnad, resulterar till drastiska ändringar i det slutliga hashvärdet [16].

2.4. Olika kryptografiska hashfunktioner

Flera olika kryptografiska hashfunktioner har utvecklats genom tiderna. I följande underavsnitt presenteras några funktioner som har ofta använts för att hasha lösenord. Vissa av funktionerna påminner varandra på flera sätt, och de största skillnaderna har främst att göra med kompressionsfunktionen samt bitlängden på hashvärdet som funktionen ger som output.

2.4.1. MD5

I figur 1 skedde hashningen av lösenorden med kryptografiska hashfunktionen MD5 (Message Digest 5). Funktionen utvecklades i början av 1990-talet av Ronald Rives, professor på MIT [14], och har historiskt varit en av populäraste hashfunktionerna inom flera olika tillämpningar, bland annat lagring av lösenord [6]. MD5 ger som output ett 128-bitar långt hashvärde.

Funktionen anses dock idag vara kryptografisk osäker, vilket beror på dess dåliga kollisionsresistens [17, 18]. Därför bör funktionen numera undvikas inom olika säkerhetsrelaterade tillämpningar. Trots detta används MD5 fortfarande inom lagring av lösenord inom vissa system [19].

2.4.2. SHA-1

En annan allmän kryptografisk hashfunktion är SHA-1 (Secure Hash Algorithm 1). Algoritmen är utvecklad av NSA och utgiven av National Institute of Standards and Technology (NIST) år 1995. Algoritmen påminner på flera sätt MD5, men längden av det utgivna hashvärdet är längre, 160-bitar [20, 15].

Lik MD5, har SHA-1 också varit allmänt använd inom flera områden inom informationssäkerhet, bland annat lagring av lösenord [6]. Idag anses funktionen dock vara kryptografiskt osäker, eftersom kollisioner har upptäckts [21]. Detta har varit situationen redan några år, och i början av 2020 publicerade två forskare en artikel där de presenterade en metod för att attackera SHA-1 hashvärden som lika effektiv som metoder emot MD5 hashvärden år 2009 [22].

2.4.3. SHA-2

SHA-2 (Secure Hash Algorithm 2) är efterträdaren till SHA-1. Funktionen är utvecklad av NSA och utgiven av NIST år 2001 [15]. Till skillnad from SHA-1 är SHA-2 inte en algoritm, utan en mängd kryptografiska hashfunktioner som består av SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 och SHA-512/256.

Algoritmerna är på flera sätt lika, och skillnaderna gäller främst storleken på blocken som processernas samt bitlängden av det slutliga hashvärdet. Till exempel SHA-256 ger ett 256-bitar långt hashvärde som output, medan outputen av SHA-512 är 512-bitar långt. SHA-512/256 beräknar hashvärdet på samma sätt som SHA-512, men den slutliga outputen förkortas till 256-bitar. Alla SHA-2 algoritmer har olika initialiseringsvektorer [15].

2.4.4. bcrypt

Till skillnad från de tidigare funktionerna i detta avsnitt, är bcrypt en kryptografisk hashfunktion som har utvecklats specifikt för hashning av lösenord. Funktionen presenterades vid en konferens 1999 och är utvecklad av Niels Provos och David Mazières [23]. Bcrypt använder sig av tekniken nyckelsträckning för att göra hashningen mer tidskrävande. Nyckelsträckning och dess betydelse tas upp i mer detalj i kapitel 4, men kort sagt innebär det att hashningen upprepas ett antal gånger.

Funktionen tar som input ett lösenord, ett kostnadsvärde (mellan 4 och 31) samt ett saltvärde. Kostnadsvärdet inverkar på antalet iterationer som lösenordet hashas. Antalet iterationer beräknas med att ta 2 upphöjt med kostnadsvärdet [23]. Saltvärdet samt dess funktionalitet tas upp senare i kapitel 4.

Kompressionsfunktionen eksblowfish, som bcrypt använder sig av, är baserat på Blowfish blockchiffret [23]. Blowfish är en mer allmän krypteringsfunktion utvecklad i början av 1990-talet av Bruce Schneier. Blockchiffret har en komplex initialiseringsfas, vilket bcrypt använder till sin fördel i sin kompressionsfunktion för att göra hashningen mer tidskrävande [23, 24].

3. Hur knäcks hashvärden av lösenord?

I det förra kapitlet konstaterades det att kryptografiska hashfunktionernas egenskaper gör dem användbara då lösenord ska lagras på ett säkert sätt för verifiering. Av dessa egenskaper är den viktigaste enkelriktigheten, vilket betyder att utifrån hashvärdet går det varken att beräkna eller dra slutsatser om innehållet i de ursprungliga indata. Trots detta finns det olika metoder som kan användas för att försöka knäcka hashvärdet, det vill säga få tag på det ursprungliga innehållet.

3.1. Kollisioner

Kollisionsresistens är en av kryptografiska hashfunktionernas viktigaste egenskaper, och därför är kollisioner också ett mycket hett ämne inom kryptografi och informationssäkerhet. Kollisioner kan tillåta hackare att bland annat förfalska digitala signaturer och certifikat [25], men deras inverkan på säkerheten av lagrade lösenord är, kanske lite överraskande, relativt liten.

En kryptografisk hashfunktion anses vara osäker då kollisionsresistensen ”går sönder”, det vill säga en metod för att skapa kollisioner upptäcks som är snabbare än den traditionella födelsedagsattacken [12]. Hashvärdet för en kryptografisk hashfunktion består alltid av n bitar. För SHA-1 är $n = 160$. Detta betyder att det går att skapa 2^{160} ($\approx 1{,}4615*10^{48}$) olika unika hashvärden med SHA-1. Eftersom antalet olika indata är oändligt, betyder det att det måste finnas åtminstone två indata som skulle resultera till samma hashvärde, det vill säga $H(M_1) = H(M_2)$, där H är en kryptografisk hashfunktion och M_1 och M_2 är två olika indata.

Födelsedagsattacken är en metod för att hitta kollisioner som kan användas till på vilken kryptografisk hashfunktion som helst [8]. Attacken bygger på födelsedagsparadoxen, vilket är ett klassiskt problem inom sannolikhetsläran som har och göra med distribution. Paradoxen går som följande; ifall vi har en grupp människor, behövs det endast 23 personer för att sannolikheten att 2 personer har samma födelsedag i årskalendern ska bli 50% (då skottår inte räknas med och alla födelsedagar är lika troliga) [26]. Ifall detta generaliseras lite, blir sannolikheten att hitta två likadana element från en mängd med N stycken element hög då elementen väljs på måfå efter att cirka $O(\sqrt{N})$ val. Detta betyder att det går att hitta en kollision

med födelsedagsattacken till ett hashvärde med körtiden $O(\sqrt{2^n}) = O(2^{n/2})$, där n är antalet bitar hashvärdet består av [8]. För SHA-1 betyder detta att det går att hitta en kollision med hjälp av födelsedagsattacken med körtiden $O(2^{160/2}) = O(2^{80})$.

I det förra kapitalet nämndes det att SHA-1 anses idag vara osäker. I början av 2020 publicerade två forskare en artikel där de lyckats implementera hittills effektivaste kollisonsattackerna mot SHA-1 [22]. Detta var en förbättring av deras tidigare implementation från 2019 [12].

Kollisionsattacker kan delas in i två typer; klassiska kollisionsattacker och *chosen-prefix* kollisionsattacker. En klassisk kollisionsattack försöker finna två olika indata vilket resulterar till samma hashvärde, det vill säga $H(M_1) = H(M_2)$. Dessa attacker är används främst bara för att visa att det är möjligt att generera kollisioner på ett effektivt sätt. Den praktiska användningen är liten, eftersom det inte går att påverka innehållet i indata M_1 och M_2 [12].

Chosen-prefix kollisionsattacker är däremot farligare, eftersom de tillåter attackeraren att delvis inverka på indata, vilket kan ha vissa tillämpningar som kan missbrukas [25]. I denna attack bestämmer attackeraren först två valfria indata prefix P_1 och P_2. Målet är därefter att beräkna två indata bihang M_1 och M_2, så att $H(P_1 \,||\, M_1) = H(P_2 \,||\, M_2)$, där || betyder sammansättning [12, 25].

I den hittills effektivaste attacken mot SHA-1 implementerade forskarna båda kollisionsattackerna, den klassiska kollisionsattacken med körtiden $O(2^{61,2})$ och *chosen-prefix* kollisionsattacken med körtiden $O(2^{63,4})$ [22]. Detta är betydligt effektivare än födelsedagsattacken, vilket är orsaken till att algoritmen anses vara osäker.

Att implementera dessa kollisionsattacker mot SHA-1 är ändå inte en värst snabb eller enkel process. Forskarna använde sig av 900 stycken Nvidia GTX 1060 grafikkort för sin attack, och det tog dem 2 månader att beräkna kollisionerna. Kostnaden för en klassisk kollisionsattack beräknas vara cirka 11 000 USD och för en *chosen-prefix* kollisionsattack cirka 45 000 USD, då attackeraren hyr billiga grafikkort (Nvidia GTX 1060, 35 USD/månad) [22].

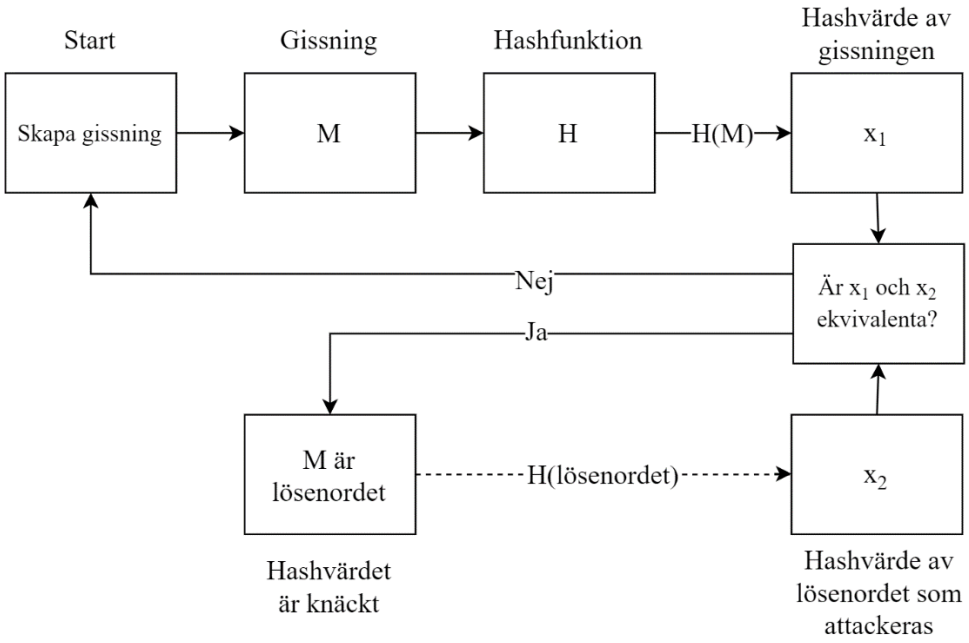
I båda av dessa attacker har attackeraren ingen kontroll över det slutliga hashvärdet. Attackeraren kan ha viss kontroll av indata vid en *chosen-prefix* kollisionsattack, men detta är begränsat till prefixen. För att i praktiken kunna använda dessa attacker mot ett visst lagrat lösenord, borde attackeraren kunna beräkna ut ett indata *M* så att $H(M) = x$, där x är hashvärdet för det lagrade lösenordet. Detta kallas för en *preimage* attack [8], och som av februari 2020 har en sådan attack endast i teorin bevisats för MD5 [27]. Körtiden för denna *preimage* attack beräknas vara $O(2^{123,4})$, vilket anses vara beräkningsmässigt omöjligt för dagens datorer.

3.2. Råstyrkeattacker

Det enklaste sättet att försöka knäcka hashvärdet av ett lösenord är att helt enkelt försöka gissa själva lösenordet [11]. Denna sorts attack mot ett hashvärde kallas för en råstyrkeattack (*brute-force attack)*, eftersom attackeraren med ”råstyrka” försöker hitta rätt lösenord. Antalet gissningar blir ofta mycket högt, ofta flera miljarder [5]. Eftersom *preimage* attacker för tillfället anses vara beräkningsmässigt omöjliga, hör råstyrkeattacker till de få metoder som kan användas för att försöka knäcka hashvärdet av ett lösenord.

Själva attackprocessen i en råstyrkeattack, som kan ses i figur 3, är enkel: Skapa en gissning, kör gissningen genom samma hashfunktion som det hashade lösenordet och jämför resultatet. Ifall hashvärdet från gissningen är ekvivalent med det som attackeras, är hashvärdet knäckt och motsvarande lösenord har hittats [11]. Om inte, börjar processen från början. Gissningarna och jämförelserna görs vanligtvis av ett program som är specialiserade för att knäcka lösenord [5], till exempel hashcat [28].

Råstyrkeattacker kan delas in i två olika varianter; ordboksattacker och klassiska råstyrkeattacker. I en klassisk råstyrkeattack testas alla olika kombinationer av tecken (det vill säga stora bokstäver, små bokstäver, siffor och symboler) tills motsvarande lösenord för hashvärdet hittas, eller tills attackeraren ger upp [5]. Denna typ av attack är beräkningsmässigt väldigt dyr och relativt ineffektiv, men den hittar nog rätt motsvarande lösenord för ett hashvärde, ifall det bara är möjligt att köra attacken tillräckligt länge [11].



Figur 3 – Processen för att knäcka ett hashvärde med råstyrkeattack

Användarnas lösenord består dock sällan bara av slumpmässiga tecken, utan oftast är lösenorden uppbyggda av ett eller flera ord. Detta utnyttjas av ordboksattacker, som tar in en lista innehållande ord, namn, fraser och vanliga lösenord, och utifrån detta skapar gissningarna. Dessa listor kan laddas ner från olika sidor runt internet [5]. Gissningarna som skapas innehåller ofta små modifikationer som användare i regel brukar göra, bland annat kan vissa bokstäver ersättas med siffror, till exempel så att ”morot” blir ”m0r0t” [11]. Ordboksattacker brukar vara effektivare än klassiska råstyrkeattacker, eftersom de ofta kan hitta motsvarande lösenord till ett hashvärde med ett mindre antal gissningar. Men ifall användaren har valt ett starkt lösenord, kan det hända att ordboksattacken aldrig lyckas knäcka hashvärdet, och då måste attackeraren övergå till en klassisk råstyrkeattack.

Prestandan av en råstyrkeattacks implementation mätts i regel i beräknade hashvärden per sekund (H/s). Eftersom moderna grafikkort idag kan skapa flera miljoner hashvärden per sekund, brukar prestandan vanligtvis betecknas i MH/s, vilket är mega (10^6 = en miljon) hashvärden per sekund. I juni 2019 publicerade en grupp forskare en artikel där de lyckats implementera en algoritm som genererar SHA3-512 hashvärden med hjälp av ett Nvidia GTX1080 grafikkort med en maximal genomströmning på 909,6 MH/s [29]. Noterbart är att SHA-3 är en relativt ny familj av kryptografiska hashfunktioner. Med äldre funktioner, som MD5, går det att generera hashvärden även snabbare. Mera om detta i kapitel 5.

Även om moderna grafikkort idag kan skapa hashvärden mycket snabbt, kan det ändå ta otroligt länge för en råstyrkeattack att hitta ett längre lösenord ifall det är långt. Detta är eftersom antalet olika kombinationer av tecken ökar exponentiellt när längden av lösenordet ökas. I kapitel 5 tas detta upp i mer detalj.

3.3. Uppslagningstabeller

En annan metod för att knäcka hashvärden är uppslagningstabeller. Med uppslagningstabeller är det möjligt att inom några sekunder hitta motsvarande lösenord till flera olika hashvärden [5], vilket gör dem mycket effektiva. Idén bakom uppslagningstabeller är att i förväg beräkna hashvärden av olika lösenord och därefter lagra hashvärden i en tabell tillsammans med de oformaterade lösenorden i hashvärde-lösenord par [30]. Efter att attackeraren lyckats få tag på lösenordshashen, till exempel efter ett dataintrång, kan attackeraren sedan slå upp hashvärdena i tabellen för att försöka hitta det motsvarande lösenordet [11].

Uppslagningstabeller innehåller ofta flera miljarder olika hashvärde-lösenord par, och därför tar det en lång tid att skapa dem [5]. De är dock mycket effektivare jämfört med råstyrkeattacker, eftersom attackeraren inte behöver börja spendera tid på att knäcka ett lösenord, utan kan slå upp alla på samma gång.

Problemet med uppslagningstabeller är att de tar mycket utrymme [30], eftersom både det ursprungliga lösenordet samt dess motsvarande hashvärde måste sparas i tabellen. Regnbågstabeller, en variant av uppslagningstabeller, löser delvis detta problem. Regnbågstabeller knäcker hashvärden långsammare jämfört med vanliga uppslagningstabeller, eftersom de kräver en del beräkningar för att leta paren, men de tar i sin tur mindre utrymme [31]. Detta betyder att en attackerare kan med hjälp av regnbågstabeller lagra flera lösenord-hashvärde par i samma utrymme, vilket gör dem mer effektiva och praktiska.

Regnbågstabeller byggs upp av så kallade hashkedjor [31]. En hashkedja består av ett startvärde, som är i oformaterad text, samt ett sluthash. Dessa hashkedjor kan innehålla flera miljoner hashvärden, men endast två strängar måste sparas i själva tabellen. Hashkedjorna skapas med hjälp av en kryptografisk hashfunktion samt med t stycken reduktionsfunktioner, där t är längden av kedjorna. En reduktionsfunktion är en enkelriktad funktion som tar som input ett hashvärde och

ger som output en sträng. Denna sträng kallas för en nyckel. Samma reduktionsfunktion ger alltid samma output för en viss input [31, 32].

För att skapa en hashkedja väls först ett startvärde, vilket sedan hashas. Det resulterande hashvärdet används sedan som input i reduktionsfunktionen, vilket resulterar till en nyckel. Därefter hashas nyckeln, vilket resulterar i ett nytt hashvärde. Denna process upprepas tills hashkedjan blir av längden *t*, varefter det sista hashvärdet sparas som motsvarande sluthash till startvärdet [32]. Storleken av *t* varierar mellan tabeller. Ifall hashkedjorna är långa, blir själva tabellens filstorlek mindre.

Sökningarna sker däremot med att först kolla ifall hashvärdet *h* finns bland sluthashen i regnbågstabellen. Ifall *h* inte finns bland sluthashen, används *h* som input i reduktionsfunktionen, vilket resulterar i en nyckel som hashas därefter. Det resulterande hashvärden jämförs därefter igen med sluthashen i tabellen. Ifall ingen träff hittas upprepas processen åter. Ifall en träff sker, betyder det att motsvarande lösenordet finns i hashkedjan vars sista hashvärde är sluthashet. För att hitta lösenordet, börjar man utifrån motsvarande startvärde gå igenom kedjan tills strängen *m* kommer emot, vilket är motsvarande lösenord till *h*, det vill säga $H(m) = h$. Detta betyder att desto längre hashkedjorna är, desto mera beräkningar måste göras för att genomföra sökningen, vilket gör processen långsammare [32].

En attackerare behöver nödvändigtvis inte spendera en lång tid på att skapa regnbågstabeller dem själva, eftersom de kan också laddas ner från internet. Vissa aktörer säljer dessa tabeller på nätet [33], medan andra tillåter användare att ladda ner deras gratis [34]. Storleken på regnbågstabellerna från *Free Rainbow Tables* varierar mellan några tiotals gigabyte till flera terabyte [34].

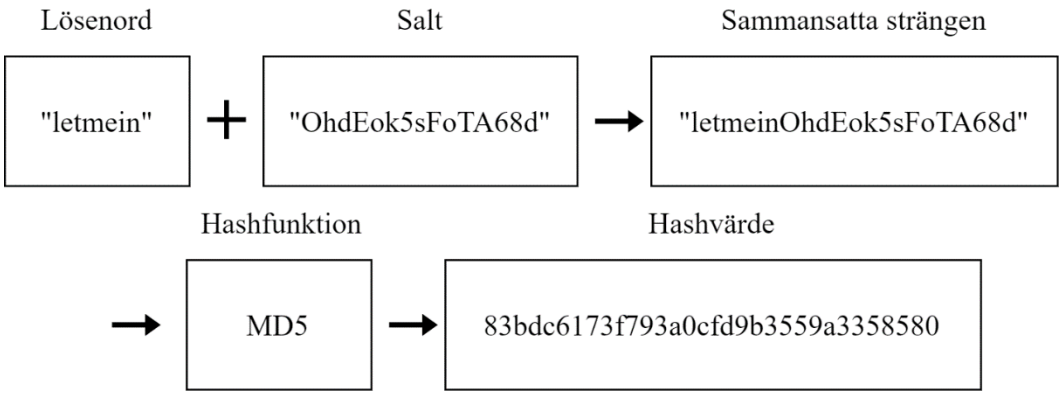
Uppslagningstabeller samt regnbågstabeller kan effektivt knäcka hashvärden, även för längre och mer komplicerade lösenord. Det finns dock en teknik som kan användas för att göra det omöjligt att använda dessa tabeller för att knäcka hashvärden av lösenord. Denna teknik kallas för saltning, och kommer att tas upp i följande kapitel av denna avhandling.

4. Tekniker för att förstärka hashvärden

Både råstyrkeattacker samt uppslagningstabeller kan användas för att försöka knäcka hashvärdet av ett lösenord. Effektiviteten av båda dessa metoder varierar. Råstyrkeattacker kräver ett stort antal gissningar för att eventuellt hitta rätt lösenord, och är därför beräkningsmässigt ineffektiva. Modern hårdvara kan dock idag redan generera hundratals miljoner hashvärden per sekund [29], och beräkningskraften kan antas bara öka i framtiden. Uppslagningstabeller kan redan idag användas för att snabbt knäcka flera hashvärden, och speciellt regnbågstabeller, med deras mindre storlek, kan också delas på internet. Därför är det avgörande att lösenorden behandlas och hashas med rätt tekniker.

4.1. Saltning

Saltning är en teknik där en slumpmässigt genererad sträng läggs till det ursprungliga lösenordet, varefter den sammansatta strängen körs igenom hashfunktionen. Denna slumpmässiga sträng kallas för ett saltvärde, eller kort också bara för ett salt [6, 11]. Processen för att salta ett lösenord kan ses i närmare detalj i figur 4. Saltvärdet orsakar den effekten att varje hashvärde som skapas kommer att vara unikt, även om flera användare skulle ha samma lösenord. Varje gång en användare skapar ett nytt lösenord, skapas också ett nytt saltvärde. Saltet måste lagras tillsammans med lösenordets hashvärde, eftersom samma salt måste också tillsättas till den sträng som användaren matar in som lösenord vid inloggning [6]. Ifall saltvärdet inte tillsätts till strängen kommer det inte att resultera till samma hashvärde som är lagrat i databasen, även om användaren skulle ha matat in rätt lösenord.



Figur 4 – Processen för att skapa ett saltat hashvärde av ett lösenord

Tilläggningen av saltvärdet gör det omöjligt för attackerare att försöka knäcka hashvärden med hjälp av någon som helst variant av uppslagningstabell [11]. Detta är eftersom uppslagningstabeller bygger på kryptografiska hashfunktionens egenskap att ett visst indata resulterar alltid till samma hashvärde. Därför är det möjligt att i förväg beräkna hashvärden for olika lösenord och spara dem i en tabell. Eftersom saltet orsakar effekten att varje hashvärde blir unikt, gör de det omöjligt att skapa uppslagningstabeller i förväg. Det är dock viktigt att saltvärdet som tilläggs inte är för kort, eftersom annars kan den sammansatta strängen också bli kort, vilket betyder att uppslagningstabeller kan knäcka hashvärdet.

Saltet sparas vanligtvis tillsammans med lösenordets hashvärde utan någon kryptering, ofta med att helt enkelt placera saltet framför eller bakom hashvärdet i samma sträng. Vissa forskare anser dock att en mer dynamisk placering skulle kunna öka på säkerheten av hashvärdena [6].

Orsaken till att saltvärdet lagras okrypterat är eftersom saltvärdet egentligen är ingen hemlighet, dess uppgift är endast att förhindra attackera från att använda uppslagningstabeller för att knäcka hashvärden [11]. Dessutom måste saltvärdet användas varje gång då lösenordsverifiering ska göras. Saltvärdet har därför ingen inverkan på effektiviteten av råstyrkeattacker, eftersom attackeraren får tag på saltvärdena samtidigt med hashvärden av lösenorden. Attackeraren kan därefter lägga saltvärdet till varje gissning för att försöka knäcka ett lösenord. Antalet gissningar som attackeraren måste göra förblir den samma, oberoende om hashvärdet är saltat eller inte.

4.2. Starkare lösenord

Det effektivaste sättet att bekämpa råstyrkeattacker är att göra lösenordet svårare att gissa för en dator. Det enklaste sättet att göra detta är att öka på lösenordets entropi, det vill säga öka på antalet bitar i lösenordet [35]. I praktiken betyder detta att längre lösenord har högre entropi, vilket gör dem säkrare. Längre lösenord innebär att det finns fler möjliga olika teckenkombinationer, vilket betyder att attackeraren måste göra fler gissningar för att eventuellt hitta rätt lösenord.

Även om ett lösenord har en hög entropi, kan det ändå vara sårbart för ordboksattacker [36]. Säkerheten av ett lösenord kan ökas genom användningen av

kombinationer av flera olika tecken. Genom att använda av kombinationer av flera olika små bokstäver, stora bokstäver, siffror samt symboler i lösenord, ökar antalet möjliga kombinationer ytterligare. Speciellt symboler används sällan av de flesta användare i deras lösenord [5].

De starkaste lösenorden består av slumpmässigt valda tecken för att tvinga attackeraren att använda än klassisk råstyrkeattack för att knäcka hashvärdet, eftersom de är den ineffektivare än ordboksattacker. Ett optimalt lösenord skulle därför till exempel kunna vara ”$SUX8H9@%q&yAMb3zPQr”. Detta lösenord har hög entropi och består av flera olika tecken, vilket betyder att det är otroligt svårt för en dator att gissa det [5].

Problemet är dock att den här sortens lösenord är mycket svåra att komma ihåg för de flesta användare. Majoriteten av användarna skapar lösenord som är enkla att komma ihåg, istället för att prioritera säkerheten [1], vilket leder ofta till korta och svaga lösenord.

Tjänsteleverantörer och organisationer kan, och i de flesta fall brukar, sätta vissa krav på användares lösenord. Ett av dessa krav kan till exempel vara att lösenordet måste ha en viss minimilängd och innehålla minst ett visst antal symboler [37]. På detta sätt går det att förhindra användare att skapa för enkla lösenord.

Problemet är att ifall kraven är för strikta, kan användarna bli irriterade, och ifall det är ifråga om en tjänst kan användarna också sluta använda den helt och hållet. Kraven kan därför inte häller vara för strikta, vilket betyder att användarna själv måste ansvara för att lösenordet är tillräckligt starkt.

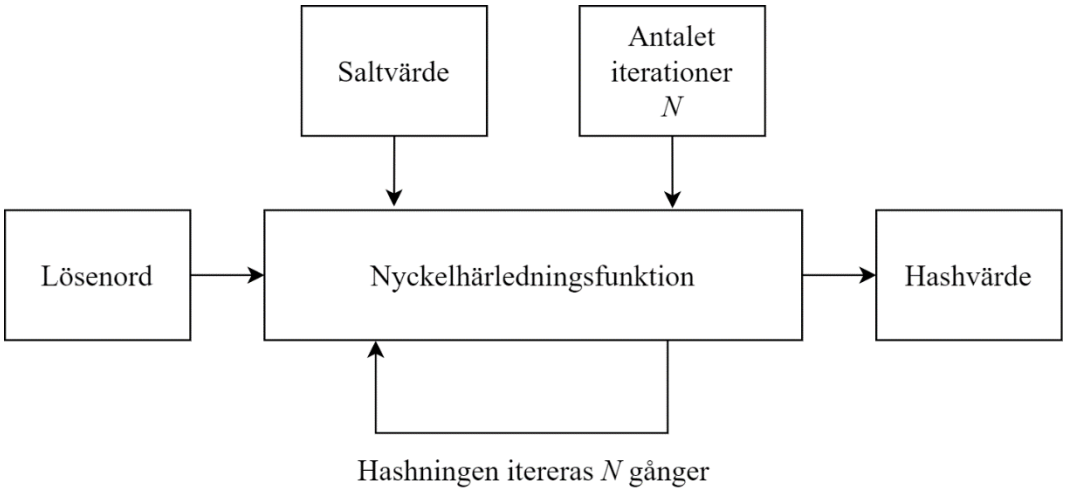
Vissa tjänsteleverantörer och organisationer tvingar användare också att byta sina lösenord efter en viss tid. Idén bakom detta är att ifall användarna återanvänder samma lösenord också i andra konton, vilket inte alls är ovanligt [2, 3, 4], hinner detta lösenord inte äventyras före det ”blivit gammalt” [38]. I princip ökar detta förfarande på kontots säkerhet ifall situationen är som beskrevs, men i praktiken ökar detta främst på antalet lösenord som användarna måste komma ihåg, vilket leder till svagare lösenord. Förfarandet tar också sällan i beaktan den ökade beräkningskraften av dagens hårdvara, vilka kan eventuellt knäcka lösenord relativt snabbt, beroende på hur de är lagrade samt hur starka lösenorden är [38]. Mera om detta i kapitel 5.

4.3. Nyckelsträckning

Datorer kan idag mycket snabbt beräkna hashvärdet för något visst indata, eftersom beräkningarna som görs av kryptografiska hashfunktionen är beräkningsmässigt mycket billiga. Detta har sina fördelar, till exempel vid beräkningen av checksummor, men gör det också möjligt för attackerare att skapa hundratals miljoner hashvärden per sekund i råstyrkeattacker [29]. Detta problem med kryptografiska hashfunktioner har redan varit känt i en längre tid, och därför har tekniken nyckelsträckning (*key stretching*) utvecklats [39].

Idéen bakom nyckelsträckning är att öka den beräkningsmässiga kostnaden för att beräkna hashvärdet [39]. När kostnaden ökar så ökar också tiden det tar att beräkna hashvärdet för något indata. Skillnaden i tiden är försumbar för enstaka användare som loggar in, men för en attackerare som försöker med rå styrka knäcka flera lösenord kan processen bli betydligt långsammare.

Nyckelsträckningen görs vanligtvis genom att iterera hashningsprocessen flera gånger, det vill säga hashvärdet från den tidigare iterationen körs igen genom samma hashfunktion. Funktioner som bygger på denna iterativa process kallas för nyckelhärledningsfunktioner (*key derivation functions)* [40].



Figur 5 – Den allmänna strukturen av en nyckelhärledningsfunktion

Flera olika varianter av nyckelhärledningsfunktioner existerar, men gemensamt för dem alla är den allmänna strukturen som kan ses i figur 5. Funktionerna tar som input åtminstone ett lösenord, ett saltvärde samt antalet iterationen som hashningen ska ske eller någon kostnadsfaktor. Eftersom antalet iterationer är anpassningsbart,

går det enkelt att öka på den beräkningsmässiga kostnaden med tiden då beräkningskraften av hårdvara ökar [40].

Vissa nyckelhärledningsfunktioner, som bcrypt [23] och Argon2 [41], är särskilda kryptografiska hashfunktioner, med sina egna kompressionsfunktioner, som har utvecklats specifikt för användning i hashning av lösenord. Däremot PBKDF2, en annan nyckelhärledningsfunktion, använder sig av valfri pseudo-slumpmässig funktion för att skapa hashvärdet. Denna pseudo-slumpmässiga består ofta av en allmän kryptografisk hashfunktion, till exempel SHA-1 [42].

5. Kryptografiska hashfunktionens inverkan på säkerheten av lagrade lösenord

Kryptografiska hashfunktioner används runt om i världen för att säkra lagrade lösenord. Flera olika funktioner används för detta mål, och de alla beräknar ut ett hashvärde som är mycket säkrare jämför med om lösenordet skulle sparas i något annat format. Hashvärden av lagrade lösenord är dock inte omöjliga att knäcka, vilket konstaterades i kapitel 3. Styrkan av ett lösenords hashvärde varierar dock beroende på flera olika faktorer, varav den använda kryptografiska hashfunktionen spelar en viss roll. Vissa kryptografiska hashfunktioner är bättre anpassade för användning i lagring av lösenord än andra, men även de optimalaste funktionerna kan inte skydda alla lösenord.

5.1. Snabba och långsamma funktioner

Kryptografiska hashfunktioner kan i viss mån delas in snabba och långsamma varianter [11, 43]. Skillnaden mellan de snabba och långsamma funktionerna är att de långsamma använder sig av nyckelsträckning för att göra hashningen beräkningsmässigt dyrare, vilket ökar på tiden det tar att beräkna ett hashvärde. Hashvärden som är beräknade med långsamma funktioner är därför mer resistenta emot råstyrkeattacker än de som är beräknade av snabba funktioner.

Skillnaden i tiden mellan snabba och långsamma kryptografiska hashfunktioner är anmärkningsvärd. För att påvisa skillnaden genomförde jag ett test där jag mätte hastigheten av några allmänna funktioner med hjälp av hashcat [28], ett verktyg för att knäcka hashvärden av lösenord, på min egen dator med ett Nvidia RTX 2070 grafikkort. Resultaten kan ses nedan i tabell 1:

Kryptografisk hashfunktion	**Antal iterationer**	**Hastighet**
MD5	(1)	27 191,5 MH/s
SHA-1	(1)	9 791,5 MH/s
SHA-256	(1)	4 054,2 MH/s
SHA-3-512	(1)	934,9 MH/s
bcrypt	32 (Kostnadsfaktor 5)	14 446 H/s
PBKDF2-HMAC-SHA-1	999	3 701,6 kH/s
PBKDF2-HMAC-SHA256	999	1 558,9 kH/s

Tabell 1 – Resultaten från testet

Från tabellen går det tydligt att skilja mellan de snabba och långsamma funktionerna. I de snabba funktionerna mäts hastigheten i enheten MH/s, vilket är miljon hashvärden per sekund, med de långsamma mäts i kH/s (kilo) eller bara H/s. Skillnaden mellan den snabbaste och den långsammaste funktionen i denna körning blev enorm. På samma tid som get tog att beräkna ett hashvärde med bcrypt beräknades cirka 1,88 miljoner hashvärden med MD5.

Noterbart är att hastigheten varierar beroende på hårdvaran som används. I detta test på min dator var bcrypt mycket långsammare jämfört med de andra funktionerna, men specialiserad hårdvara existerar som användas för att beräkna dessa hashvärden snabbare och effektivare [44].

I detta test användes också bara ett grafikkort. Flera grafikkort kan även köras parallellt för att öka på hastigheten ytterligare, vilket i sin tur gör det snabbare att knäcka hashvärden med råstyrkeattacker [45].

5.2. Osaltade hashvärden av lösenord

Ifall inget saltvärde tillsätts till lösenorden vid hashningen, betyder det att de resulterande hashvärdena kommer att vara sårbara mot uppslagningstabeller [5]. I kapitel 3 konstaterades det att uppslagningstabeller är mycket effektiva, eftersom de består av i förhand beräknade lösenord samt deras motsvarande hashvärde.

Vilken kryptografisk hashfunktion som används har därför inte lika stor betydelse, eftersom attackeraren alltid i förväg kan beräkna uppslagningstabellen. Ifall någon allmän funktion har använts för att hasha lösenorden kan det till och med vara möjligt för attackeraren att ladda ner regnbågstabeller [33, 34] och rakt börja knäcka hashvärden.

Om attackeraren måste skapa uppslagningstabellen själv, kan hastigheten av funktionen i viss mån påverka hur tabellens effektivitet. Tabeller med hashvärden av långsamma funktioner tar längre tid att skapa, vilket betyder att tabellen fylls långsammare jämfört med en tabell av en snabbare funktion. Eftersom det tar en mycket lång tid att skapa dessa tabeller oberoende av funktionen, kan långsamma hashfunktioner göra denna process ta ännu längre tid.

De långsamma funktionerna har dock vanligtvis inbyggd saltningsfunktionalitet [40], eftersom de är planerade just för hashning av lösenord, så osaltade hashvärden är sällan ett problem. Det är dock möjligt att ”missbruka” dessa hashfunktioner med att ge varje lösenord samma saltvärde, vilket i sin tur orsakar effekten att varje hashvärde inte längre är unikt. Eftersom saltvärdet i sig själv inte är någon hemlighet, betyder detta att attackerare kan skapa uppslagningstabeller med att lägga till saltvärdet till varje lösenord. Situationen är då i princip samma som om hashvärdet skulle vara osaltat [11].

Även om det kan ta mycket länge att skapa uppslagningstabeller så stoppar det inte hackare från att skapa dem. Kluster med flera grafikkort [45] samt specialiserad hårdvara [44] kan användas för att minska på tiden det tar att skapa tabellerna. Tiden är dessutom sällan ett större problem, så längde som attackerarna lyckas knäcka lösenorden. Därför är det oerhört viktigt att lösenorden saltas ordentligt före de körs igenom kryptografiska hashfunktionen, oberoende vilken funktion som används.

5.3. Svaga och starka lösenord

Ifall hashvärden av de lagrade lösenorden är saltade ordentligt, kan attackeraren inte använda sig av uppslagningstabeller, och då blir råstyrkeattacker det enda kvarliggande alternativet för att knäcka hashvärdena. I råstyrkeattacker spelar den använda kryptografiska hashfunktionen en mycket större roll jämfört med uppslagningstabeller.

Hur enkelt eller svårt det är att knäcka hashvärdet av ett lösenord med råstyrkeattacker beror på hur starkt det ursprungliga lösenordet är. Ifall lösenordet är svagt kan en dator också relativt enkelt också gissa det, medan om lösenordet är starkt kan det vara beräkningsmässigt omöjligt att knäcka hashvärdet [5, 36].

Råstyrkeattacker kan alltid användes mot hashvärden av lösenord, oberoende om vilken kryptografisk hashfunktion som har använts [11]. Skillnaden är i tiden det tar att utföra attacken. Desto komplexare lösenordet är, desto större blir skillnaden, när mer gissningar måste utföras. I tabell 2 kan detta tydligt ses. Tabellen visar maximala tiden det tar att knäcka ett hashvärde av ett lösenord av en viss längd med en klassisk råstyrkeattack, då antalet möjliga tecken är 62, det vill säga stora och

små bokstäver från a till z samt siffrorna 0 till 9. Hastigheten av kryptografiska hashfunktionernas är samma som i testet vars resultat finns i tabell 1.

Kryptografisk hashfunktion	**Hastighet**	**Tiden att knäcka ett lösenord med längden**				
		6	**7**	**8**	**9**	**10**
MD5	27 191,5 MH/s	2,1 s	2 m	2,2 h	5,76 dygn	357 dygn
SHA-256	4054,2 MH/s	14 s	15 m	15 h	38,65 dygn	6,56 år
bcrypt	14 446 H/s	45,5 dygn	7,73 år	479 år	29 712 år	1,8 miljoner år
PBKDF2-HMAC-SHA-1	3701,6 kH/s	4,26 h	11 dygn	1,87 år	116 år	7 190 år

Tabell 2 – Jämförelse mellan olika funktioner av tiden det tar att knäcka ett hashvärdet av ett lösenord av en viss längd med 62 möjliga tecken

Noterbart är att tiden som visas i tabell 2 är den maximala tiden, vilket är tiden det tar att beräkna hashvärdet för varje möjlig teckenkombination. I verkligheten behöver attackeraren i medeltal beräkna endast hälften av de möjliga kombinationerna för att knäcka ett hashvärde, vilket i sin tur betyder också att endast halva tiden behövs i medeltal [26]. Attackeraren kan också använda sig av flera grafikkort eller specialiserad hårdvara för att göra processen snabbare.

Från tabell 2 märks det att hashvärden av korta lösenord går att knäcka relativt snabbt, oberoende om vilken funktion som är används. Långsamma funktioner gör att processen tar längre tid, men tiden är inte oöverkomlig, speciellt om attackeraren använder sig av flera grafikkort. Däremot ifall attackeraren försöker knäcka flera hashvärden, till exempel efter ett dataintrång till en databas, kan det ta mycket längre för attackeraren att gå igenom lösenordshashen.

Tiden för att knäcka hashvärdet av ett lösenord växer exponentiellt då länden av lösenordet stiger, vilket kan också ses ur tabell 2. Detta betyder att då längden av lösenordet ökas tillräckligt, börjar det bli beräkningsmässigt omöjligt att knäcka hashvärdet med en klassisk råstyrkeattack, oberoende om vilken kryptografisk hashfunktion som används. Till exempel ifall lösenordet skulle bestå av 20 tecken, och lösenordets samt hashfunktionernas specifikationer skulle vara samma som i tabell 2, skulle det i medeltal ta cirka 410 biljarder ($4{,}1*10^{17}$) år att knäcka hashvärdet ifall den snabbaste funktionen (MD5) skulle användas.

Styrkan av lösenorden är dock något som är på användarnas eget ansvar. I viss mån går det att sätta krav på lösenorden, till exempel angående längden, men användarna brukar ofta välja ett lösenord som är lätt att komma ihåg istället för att säkert sådant [1]. Därför är bör en långsam kryptografisk hashfunktion alltid användas för att lagra lösenord. På detta sätt är lösenorden lagrade i det säkraste formatet möjligt. Hashvärden av svaga lösenord kan ändå knäckas, men åtminstone tar attacken en längre tid.

De långsamma funktionernas hastighet går att kontrollera i viss mån med att ändra på antalet iterationer som utförs. Desto mera iterationer, desto mera beräkningar behövs för att hasha lösenordet, som i sin tur gör också attackerna mot hashvärdet långsammare [40]. För lösenord som sparas lokalt i en dator kan antalet iterationer vara högt, eftersom tiden det tar att skapa och verifiera enstaka lösenord är försumbar. I webtjänster kan dock flera användare försöka logga in på samma gång, vilket i sin tur kan belasta servern mycket ifall antalet iterationer är högt. Detta kan i sin tur göra webtjänsten långsam för användarna, vilket inte är önskvärt.

Därför är det viktigt för tjänsteleverantörer att hitta en bra balans där antalet iterationer är tillräckligt högt så att hashvärden av lösenorden är säkra ifall ett dataintrång sker, samtidigt som tjänsten inte får bli långsam för användarna.

6. Avslutning

Kryptografiska hashfunktioner har ett antal egenskaper som gör dem mycket användbara inom flera olika områden inom IT. Längden av hashvärdet som funktionerna returnerar är alltid av en viss längd, vilket gör det omöjligt att dra slutsatser om storleken av ursprungliga indata. Kryptografiska hashfunktioner är också kollisionsresistenta [8], vilket betyder att det anses vara beräkningsmässigt omöjligt att hitta flera olika indata som skulle resultera till samma hashvärde. Den viktigaste egenskapen är dock enkelriktigheten, det vill säga det är omöjligt att invertera procesessen och ta reda på det ursprungliga indata utifrån hashvärdet [10].

Dessa egenskaper gör kryptografiska hashfunktioner mycket användbara inom lösenordslagring, eftersom de gör det möjligt att lagra lösenorden i ett säkert format, samtidigt som det går enkelt ett verifiera ifall en användare matat in rätt lösenord. Ifall ett dataintrång skulle ske, får attackerarna endast tillgång till hashvärden av lösenorden, vilka måste först knäckas före de kan användas till eventuell skada.

Hur enkelt det går att knäcka ett hashvärde av ett lösenord beror på flera olika faktorer, där den använda funktionen spelar en viss roll. Vissa äldre kryptografiska hashfunktioner anses idag vara kryptografisk osäkra, eftersom metoder att skapa kollisioner har upptäckts [17, 12]. Dessa sårbarheter kan dock åtminstone ännu inte användas för att hitta ett lösenord för ett visst hashvärde, vilket gör deras inverkan på säkerheten av lösenorden relativt låg. Det finns dock ingen orsak att fortsätta använda dessa äldre funktioner för att lagra lösenord, eftersom nya och säkrare alternativ existerar.

Hashvärden av lösenord kan knäckas med hjälp av råstyrkeattacker och uppslagningstabeller [5, 11]. Av dessa är uppslagningstabeller den effektivare metoden, eftersom de tillåter attackerare att knäcka flera hashvärden av vilken kryptografisk hashfunktion som helst inom en kort tid efter att tabellerna ha skapats. Regnbågstabeller [31] är en variant av uppslagningstabeller som tar mindre utrymme, och som kan till och med laddas ner gratis från internet [34]. Dessa tabeller kan dock göras oanvändbara genom saltning, vilket är en teknik där ett slumpat saltvärde läggs till lösenordet före det hashas [6]. Därför är det oerhört

viktigt att salta alla lösenord ordentligt, oberoende vilken kryptografisk hashfunktion som används.

Råstyrkeattacker kan dock användas för att försöka knäcka vilket hashvärde som helst, oberoende om det är saltat eller inte. Hur snabbt en råstyrkeattack knäcker ett hashvärde beror på ur starkt det ursprungliga lösenordet är. Ifall lösenordet är tillräckligt stark, kan det vara omöjligt att knäcka hashvärdet av det, oberoende av den använda kryptografiska hashfunktionen.

Eftersom lösenordets styrka faller på användarens eget ansvar, är det därför viktigt att försöka göra denna attack så tidkrävande som möjlig för alla hashvärden. Detta går att göra genom att använda en långsam kryptografisk hashfunktion, som använder sig av nyckelsträckning för att göra hashningen beräkningsmässigt dyrare [40]. På detta sätt går det att öka på tiden och resurserna som krävs för att knäcka ett hashvärde. Skillnaden blir tydligare desto fler hashvärden attackeraren försöker knäcka.

7. Referenser

[1] B. Grawemeyer och H. Johnson, ”Using and managing multiple passwords: A week to a view,” *Interacting with Computers,* vol. 23, pp. 256-267, 2011.

[2] S. Perman, J. Thoms, P. E. Naeini, H. Habib, L. Bauer, N. Christin, L. Faith Cranor, S. Egelman och A. Forget, ”Let's Go in for a Closer Look: Observing Passwords in Their Natural Habitat,” i *CCS '17: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, Dallas, 2017.

[3] R. Wash, E. Rader, R. Berman och Z. Wellmer, ”Understanding Password Choices: How Frequently Entered Passwords Are Re-used across Websites,” i *Proceedings of the Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, Denver, 2016.

[4] A. Adams och A. Sasse, ”Users Are Not the Enemy,” *Communications of the ACM,* vol. 42, pp. 40-46, 199.

[5] M. Burnett, Perfect Password : Selection, Protection, Authentication, Rockland: Elsevier Science & Technology Books, 2006.

[6] S. Boonkrong och C. Somboonpattanakit, ”Dynamic Salt Generation and Placement for Secure Password Storing,” *IAENG International Journal of Computer Science,* nr 43, pp. 27-36, 2016.

[7] Facebook, ”Keeping Passwords Secure,” 21 3 2019. [Online]. Available: https://about.fb.com/news/2019/03/keeping-passwords-secure/. [Använd 23 3 2020].

[8] A. J. Menezes, P. C. van Oorschot och S. A. Vanstone, Handbook of Applied Cryptography, Boca Raton: CRC Press, 1996.

[9] L. Peiyue, S. Yongxin och Y. Huaijiang, ”The parallel computation in one-way hash function designing,” i *2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering*, Changchun, 2010.

[10] P. Rogaway och T. Shrimpton, ”Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance,” i *Lecture Notes in Computer Science, vol 3017*, 2004.

[11] Defuse Security, ”Salted Password Hashing - Doing it Right,” [Online]. Available: https://crackstation.net/hashing-security.htm. [Använd 12 2 2020].

[12] G. Leurent och T. Peyrin, ”From Collisions to Chosen-Prefix Collisions Application to Full SHA-1,” i *Advances in Cryptology – EUROCRYPT 2019*, Springer, Cham, 2019, pp. 527-555.

[13] N. Sklavos och O. Koufopavlou, ”Implementation of the SHA-2 Hash Family Standard Using FPGAs,” *The Journal of Supercomputing,* vol. 31, pp. 227-248, 2005.

[14] R. Rivest, ”The MD5 Message-Digest Algorithm: RFC 1321,” Internet Engineering Task Force, 1992.

[15] National Institute of Standards and Technology, ”FIPS 180-4 Secure Hash Standard (SHS),” U.S. Department of Commerce, 2015.

[16] Y. Yang, F. Chen och X. Zhang, ”Research on the Hash Function Structures and its Application,” *Wireless Pers Commun,* vol. 94, pp. 2969-2985, 2017.

[17] Software Engineering Institute, ”MD5 vulnerable to collision attacks,” 31 12 2008. [Online]. Available: https://www.kb.cert.org/vuls/id/836068/. [Använd 21 2 2020].

[18] T. Xie, F. Liu och D. Feng, Fast Collision Attack on MD5, IACR Cryptology ePrint Archive, 2013.

[19] C. Ntantogian, S. Malliaros och C. Xenakis, ”Evaluation of password hashing schemes in open source web platforms,” *Computers & Security,* vol. 84, pp. 206-224, 2019.

[20] D. E. Eastlake, 3rd och P. E. Jones, ”US Secure Hash Algorithm 1 (SHA1): RFC3174,” Internet Engineering Task Force, 2001.

[21] S. M, B. E, K. P, A. A och M. Y, ”The First Collision for Full SHA-1,” i *Advances in Cryptology – CRYPTO 2017*, 2017.

[22] G. Leurent och T. Peyrin, ”SHA-1 is a Shambles - First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust,” Cryptology ePrint Archive, Report 2020/014, 2020.

[23] N. Provos och D. Mazières, ”A Future-Adaptable Password Scheme,” i *Proceedings of the FREENIX Track:1999 USENIXAnnual Technical Conference*, Monterey, 1999.

[24] B. Schneier, ”Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish),” i *Fast Software Encryption, Cambridge Security*, Cambridge, 1993.

[25] M. Stevens, A. Lenstra och B. de Weger, ”Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities,” i *Advances in Cryptology - EUROCRYPT 2007*, Barcelona, 2007.

[26] F. Mosteller, ”Understanding the Birthday Problem,” i *Selected Papers of Frederick Mosteller*, New York, Springer, 2006, pp. 349-353.

[27] Y. Sasaki och K. Aoki, ”Finding Preimages in Full MD5 Faster Than Exhaustive Search,” i *Advances in Cryptology - EUROCRYPT 2009*, Cologne, 2009.

[28] hashcat, [Online]. Available: https://hashcat.net/hashcat/. [Använd 14 3 2020].

[29] T. Nguyen Dat, K. Iwai, T. Matsubara och T. Kurokawa, ”Implementation of high speed hash function Keccak on GPU,” *International Journal of Networking and Computing,* vol. 9, pp. 370-389, 2019.

[30] H. Kumar, S. Kumar, R. Joseph , D. Kumar, S. Kumar Shrinarayan Singh, A. Kumar och P. Kumar, ”Rainbow table to crack password using MD5 hashing algorithm,” i *EEE Conference on Information & Communication Technologies*, Thuckalay, 2013.

[31] P. Oechslin, ”Making a Faster Cryptanalytic Time-MemoryTrade-Off,” i *Advances in Cryptology - CRYPTO 2003*, Santa Barbara, 2003.

[32] M. E. Hellman, ”A cryptanalytic time-memory trade-off,” *IEEE Transactions on Information Theory,* vol. 26, nr 4, pp. 401-406, 1980.

[33] RainbowCrack Project, ”Buy Rainbow Tables,” [Online]. Available: https://project-rainbowcrack.com/buy.htm. [Använd 5 3 2020].

[34] Distributed Rainbow Table Project, ”Free Rainbow Tables,” [Online]. Available: https://freerainbowtables.com/. [Använd 5 3 2020].

[35] J. L. Massey, ”Guessing and entropy,” i *Proceedings of 1994 IEEE International Symposium on Information Theory*, Trondheim, 1994.

[36] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. Faith Cranor och J. López, ”Guess again (and again and again):Measuring password strength by simulating password-cracking algorithms,” i *2012 IEEE Symposium on Security and Privacy*, San Francisco, 2012.

[37] R. Shay, S. Komanduri, P. Gage Kelley, N. Christin, M. L. Mazurek, L. F. Cranor, P. G. Leon och L. Bauer, ”Encountering stronger password requirements: user attitudes and behaviors,” *Proceedings of the Sixth Symposium on Usable Privacy and Security,* p. 1–20, 2010.

[38] S. Farrell, ”Password Policy Purgatory,” *IEEE Internet Computing,* vol. 12, nr 5, pp. 84-87, 2008.

[39] J. Kelsey, B. Schneier, C. Hall och D. Wagner, ”Secure applications of low-entropy keys,” *Lecture Notes in Computer Science,* vol. 1396, 2005.

[40] F. F. Yao och Y. L. Yin, ”Design and Analysys of Password-Baded Key Derivation Functions,” *Topics in Cryptology – CT-RSA 2005 – Lecture Notes in Computer Science,* vol. 3376, pp. 245-261, 2005.

[41] A. Biryukov, D. Dinu och D. Khovratovich, ”Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications,” i *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, Saarbrucken, 2016.

[42] A. Visconti, S. Bossi, H. Ragab och A. Calò, ”On the Weaknesses of PBKDF2,” i *Cryptology and Network Security, CANS 2015*, Marrakesh, 2015.

[43] D. Arias, ”Hashing Passwords: One-Way Road to Security,” Auth0, 30 9 2019. [Online]. Available: https://auth0.com/blog/hashing-passwords-one-way-road-to-security/. [Använd 13 3 2020].

[44] F. Wiemer och R. Zimmermann, ”High-speed implementation of bcrypt password search using special-purpose hardware,” i *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*, Cancun, 2014.

[45] K. Kim, ”Distributed password cracking on GPU nodes,” i *2012 7th International Conference on Computing and Convergence Technology (ICCCT)*, Seoul, 2012.