
KTH Computer Science
and Communication

# Autentisering av klient genom Proof-of-work och Challenge-response

SAMUEL WEJÉUS
wejeus@kth.se

Examensrapport vid NADA
Handledare: Mikael Goldmann
Examinator: Mads Dam
2011-06-14

# Referat

Denna essä kommer att analysera grunderna i två tekniker vid namn Proof-of-work och Challenge-response i syftet att se på hur kombinationen av dessa skulle kunna användas för att skapa ett autententiseringsprotokoll som kan säkerställa både identitet och intention hos en klient.

Jag kommer undersöka om, och i så fall hur, detta nya protokoll kan motverka Brute-force och Denial-of-service attacker genom att utnyttja viktiga säkerhetsrelaterade egenskaper från både Proof-of-work resp. Challenge-response.

En realisering av protokollet kommer att presenteras, både teoretiskt samt en implementation i form av en klient/server-modell. En analys av föreslaget protokoll kommer att ges, vilken kommer att visa på hur kombinationen av Proof-of-work och Challenge-response kan ge vissa säkerhetsmässiga fördelar men även skapar nya brister som i slutändan gör protokollet opassande för praktiskt bruk i dess nuvarande form.

# Abstract

## Authentication of a client using Proof-of-work and Challenge-response

This essay will analyze the basics of two techniques called Proof-of-work and Challenge-response with the purpose to see how the combination of these could be used to create a authentication protocol that can ensure both the identity and intention of a client.

I will examine whether, and if so how, this new protocol can prevent Brute-force and Denial-of-service attacks by taking advantage of key security features of Proof-of-work Challenge-response respectively.

A realization of the protocol will be presented, both theoretically as well as in the form of a implementation in created as a client/server model. An analysis of the proposed protocol will be provided, which will show how the combination of Proof-of-work and Challenge-response can provide some security benefits but also creates new flaws that ultimately makes the protocol unsuitable for practical use in its present form.

# Innehåll

# Kapitel 1

# Introduktion

Denna kandidatuppsatsen ingår i kursen DD143X - Examensarbete inom datalogi, grundnivå. Syftet med uppsatsen är att undersöka hur Challenge-response kan kombineras med Proof-of-work och därefter realiseras genom klient/server modellen samt visa på hur en implementation av detta kan se ut.

Challenge-response är ett abstrakt protokoll som beskriver hur en klient kan styrka sin identitet genom att besvara en auktoritets utmaning. Denna utmaning kan bestå av en förfrågan efter uppgifter som endast en unik klient har vetskap om med syftet att fastställa dess identitet.

Proof-of-work är ett liknande protokoll för verifiering av lösning till en ställd utmaning med den utmärkande skillnaden att inga antaganden tas huruvida en klient har någon kunskap om lösning för en efterfrågad utmaning i den inledande fasen av protokollet. Istället för att verifiera identitet är avsikten med Proof-of-work att bevisa en klients uppriktighet genom att denne först måste utföra ett beräkningstungt arbete som en auktoritet sedan enkelt kan bekräfta.

I resten av denna text kommer Challenge-response och Proof-of-work även att refereras till som CR respektive PoW.

## 1.1 Bakgrund

Proof-of-work föreslogs redan på -90 talet av Dwork och Naor [6] som en lösning på problemet med oönskad e-post. Detta genom att låta en användare utföra en beräkningsmässigt tung operation för varje meddelande denna önskar sända. Konceptet har utforskat vidare och har fått viss användning för bekämpning av bland annat Denial-of-service attacker [7]. Mark Handley och Adam Greenhalg [8] har lagt fram förslag om hur en miljö fri från DoS attacker i framtiden kan komma att realiseras, men under tiden finns det fort-

farande ett behov av skyddsmekanismer för att förhindra dessa. DoS-attacker har även blivit mer sofistikerade, angripare har nu upptäckt olika sätt att initiera DoS-attacker så högt som i lager 7 i OSI-modellen [9] vilket visar på att ett grundläggande DoS skydd, som från och med upprättandet av länken enbart skyddar sessionen, inte är tillräckligt och ytterligare motåtgärder bör implementeras även på applikationsnivå. Att ett progam skyddar sig själv på applikationsnivå innebär att inget antagande görs om att någon säkerhet garanteras från underliggande lager i OSI-modellen utan programmet själv tar ansvar för att säkerställa sin egen trygghet.

PoW konceptet har även utökats till att vara mer än bara ett skydd mot DoS attacker, detta bland annat genom arbetet utfört av Aura et al. [1] som visade hur en klient kan visa uppriktigheten i sin vilja att använda någon serverresurs genom PoW. Verifiering av intention genom PoW kommer förklaras ytterligare i sektion 3.1.

## 1.2 Problemformulering

I de situationer där det även är av intresse att säkerställa identiteten hos en klient krävs det ytterligare bevis för att klienten är vem denne utger sig för att vara. Alla former av autentisering kan generaliseras till att tillhöra en av tre följande kategorier [15], nämligen:

- Något en användare *är* (röstidentifiering, ögonscanning)
- Något en användare *har* (ID kort, smartcards)
- Något en användare *vet* (lösenord, PINs)

En vanlig lösning som uppfyller punkt tre är att använda sig av Challenge-response. Ett protokoll av denna typ kan implementeras och utökas på flertalet sätt och det är denna kategori av autentisering jag ämnar undersöka. Ett problem med denna typ av autentisering är att en angripare kan ta reda på någon annan klients kunskap, i detta fall ett lösenord, genom att helt enkelt göra en uttömmande sökning, s.k. Brute-force [3], av lösenordet dvs testa alla möjliga kombinationer tills den rätta har blivit funnen. En gemensam nämnare mellan tidigare diskuterade DoS och Brute-force attacker ligger i hur svagheter kan utnyttjas. I båda fallen kan en angripare initiera en massiv mängd uppkopplingar, för just fallet Brute-force attacker, innebär det att varje uppkoppling har som mål att gissa ett lösenord. Utifrån detta resonemang ges en antydan till hur ett skydd mot DoS attacker även kan mildra Brute-force attacker. Det är även möjligt att motverka denna typ av attack genom

att uppmuntra klienter att använda bättre och säkrare lösenord [4]. I denna rapport kommer ett antagandet göras nämligen att en klient redan använder ett säkert lösenord och en fråga jag ämnar försöka besvara är hur detta kan skyddas ytterligare.

Jag vill här understryka att det är en stor skillnad på den typ av verifiering som PoW resp. CR möjligör. Allmänt syftar verifikation på någon form av test med målet att säkerställa att information är korrekt och just den som förväntas. Verifikation genom Proof-of-work innebär ett test av en klients ändamål, eller uppriktighet, och jag kommer syfta till detta genom att referera tillbaka till dena typ av verifikation som *verifikation av intention*. *Verifikation av identitet* kan uppnås genom användning av CR vilket kan ge en bekräftelse av att en klients verkligen är den som den utger sig för att vara. Båda typer av verifikation kommer presenteras mer i detalj i kapitel 3.

Det huvudsakliga målet med denna uppsats är att kombinera de två koncepten, Proof-of-work och Challenge-response. Jag kommer även att undersöka om denna kombination kan användas för att säkra en användares identitet, förhindra DoS attacker samt skydda en användares inloggningsuppgifter genom att motverka Brute-force attacker. Kopplingen mellan de två begreppen Proof-of-work respektive Challenge-response är att jag kommer undersöka om, och i så fall hur, det är möjligt att autentisera en klient genom att slå samman de båda protokollen. Autentiseringsproceduren som kombinationen av PoW och CR mynnar ut i har som mål att säkerställa både identiteten och intention hos en klient.

Nödvändigheten i att kombinera de två nämnda koncepten ligger i det faktum att vid enbart användning av Proof-of-work är det endast möjligt att styrka en klients *intentioner*, men omöjligt att fastställa någon form av *identitet*. Kravet på identitet uppfylls, som senare kommer förklaras, av CR. Dock så lider enbart användning av Challenge-response av risken att utsättas för riktade Brute-force attacker som senare kommer att visas.

# Kapitel 2

# Metod

Proof-of-work är en ganska gammal idé som undersökts i flertalet akademiska texter samt fått viss praktisk användning i en del system [6, 8, 11, 1]. Att använda Proof-of-work har även mottagit en del negativ kritik då det bevisats att inte fungera för användning som en skyddsåtgärd mot tex. spam [10]. Det är med ett antagande att en viss modifiering av protokollet måste till för att eventuell kunna få någon praktisk nytta i ett autentiseringssystem.

Metoden som kommer att används för att undersöka hur CR kan kombineras med PoW är att låta indata för de två separata utmaningarna agera som gemensam indata för en enda utmaning som bör uppfylla kraven på verifiering av både identitet och intention. Kravet som kommer ställas på denna unifierade utmaning är att den måste uppfylla den funktionalitet som tillgodoses av CR och PoW i dess separerade former. Undersökningen innebär därmed att se på:

- Hur förändras egenskaperna för CR resp. PoW vid en sammanslagning?
- Vilka fördelar ges av en sammanslagning?
- Vilka nackdelar ges av en sammanslagning?

För att knyta tillbaka till problemformuleringen så inkluderar fördelar att skydd mot Brute-force och Denial-of-service samt skydd av inloggningsuppgifter. Nackdelar är uppenbart om någon av föregående fördelar bryts samt om ytterligare egenskaper uppstår som kan anses vara negativa ur ett säkerhetsperspektiv.

För att kunna analysera egenskaperna vid en ihopslagning kommer dels ett abstrakt protokoll skapas för att lättare kunna resonera teoretiskt om sammanslagningen samt en praktisk användbar implementation för undersöka

om en praktiska användningen kan visa på egenskaper som kan vara svåra att finna genom teoretisk analys.

# Kapitel 3

# Genomgång av använda koncept

Jag kommer här förtydliga de grundläggande koncept som kommer användas, nämligen Proof-of-work och Challenge-response, samt undersöka de egenskaper som dessa besitter. Avslutningsvis kommer jag undersöka hur de speciella egenskaperna som de två protokollen har, kan modifieras för att i slutändan kunna sammanfogas utan att förlora viktiga säkerhetaspekter. För de två begreppen; Proof-of-work och Challenge-response kommer ursprungsvarianten beskrivet i *DOS-resistant Authentication with Client Puzzles* [1] respektive *Challenge-response authentication* [5] att förklaras då detta ger en enklare koppling till tidigare forskning.

## 3.1 Proof-of-Work

Ett Proof-of-work är, som tidigare nämnt, en pusselbaserad utmaning mellan server och klient där klienten ombeds bevisa sin uppriktighet genom att lösa pusslet vid just det tillfället en server ger utmaningen till klienten. En beräkningstung operation, som först nämnt i introduktionen, kan vara ett anings svagt uttryck, vad som menas med detta är en operation som bör ta en viss *tid* för en klient att utföra. Ett pussel bör vara av det slaget att det inte skall vara möjligt att "fuska" eller rättare sagt: ta genvägar för att nå fram till lösningen. Enda möjligheten för att finna korrekt lösning måste därmed innebära att en klient måste använda sig av någon viss förutbestämd lösningsmetod.

Ett pussel bör vara helt unikt och det ska inte vara möjligt att beräkna lösningar i förhand, då detta skulle omintetgöra syftet med pusslet. Svårigheten på ett pussel bör kunna anpassas dynamiskt med intentionen att svårigheten för en klients pussel skall ökas då en attack mot denna klient identifierats. Med dynamisk syftas på tiden det tar att lösa detta pussel och bör kunna varieras från noll till oändlig.

Användningen av denna kostnad tvingar klienten att "betala" för varje försök till anslutning. Ju fler anslutningar klienten försöker göra, desto mer måste denna betala, och som ett resultat minskar risken för att en angripare som försöker utnyttja systemet lyckas fullfölja en attack. Ett bra pussel bör uppfylla följande egenskaper [6]:

- Att skapa ett pussel och validera dess lösning är billigt för en server.
- Kostnaden för att lösa ett pussel ska enkelt, på ett dynamisk sätt, kunna ökas från noll till omöjligt.
- Det skall inte vara möjligt att förberäkna lösningar till pussel.
- Ett pussel bör kunna lösas oavsett hårdvara (dock kommer det ta längre tid vid användning av långsammare hårdvara).

Det finns många typer av pussel som uppfyller kraven på bra pussel. Som exempel på bra pussel kan nämnas det svåra (men inte omöjliga) problemet att finna rötter till ett tal $a$ modulu något primtal $p$, finna inverser till hashfunktioner eller ett Fiat-Shamir-baserat schema (ett teoretiskt tyngre pussel som har likheter med båda tidigare nämnda pussel) [6]. Ett system där Proof-of-work används är *Hashcash* som är ett e-postsystem för vilket en användare måste köpa frimärken (genom att offra ett antal av sina CPU cykler) för att kunna sända e-post. Hashcash har nått viss framgång och existerar som plugins till diverse e-postklienter och som spamfilter i olika bloggsystem. Då syftet är att kombinera Proof-of-work med Challenge-response kommer endast ett, det så kallade hash-pusslet, att används i undersökningen på grund av enkelheten i dess implementation.

### 3.1.1 Hashbaserade pussel

I ett hash-pussel löser en klient ett hash-pussel genom att finna en sträng som genererar ett hashvärde vars strängrepresentation inleds med $m$ nollor [1], denna form av pussel är grafiskt beskriven in figur 3.1.1.

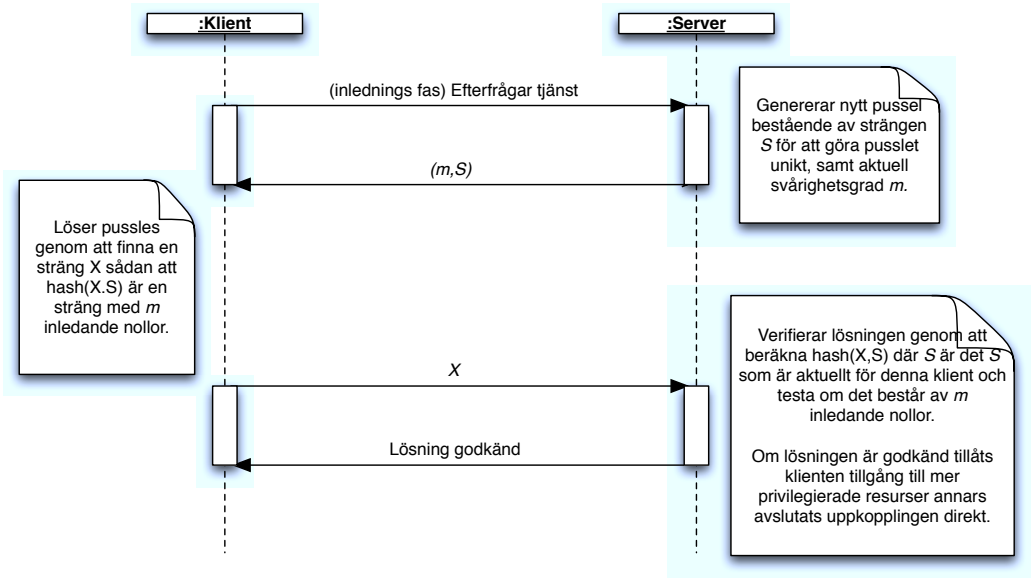
Denna sträng erhålls enklast genom Brute-force då det än så länge är det mest effektiva tillvägagångssättet att finna inverser till denna typ av funktioner. Valet av att finna ett hash, vars strängrepresentation, har den sökta formen med m inledande nollor är godtycklig och anledningen till valet är att då motsatsen, finna inverser till hash, är i praktiken (dvs. inom rimligt tid) omöjligt för strängar längre än några få tecken.

$$h(s, X) = \underbrace{00..00}_{m\ nollor} + \overbrace{Y}^{resterande\ hash}$$

**Figur 3.1.** Hashbaserat pussel

Här är $h$ en hash funktion, $m \in \mathbb{Z}$, samt s är en sträng. Utmaningen ligger således i att finna strängen $X$. Delstängen $s$ genereras och sänds ut från en server. Anledningen till att inkludera $s$ i pusslet är för att göra pussel unika och motverka att pussel kan förberäknas. En auktoritet kan enkelt verfiera en lösning genom att enbart beräkna ett hashvärde utifrån en klients påstådda lösning och jämföra denna med den korrekta.

Värt att notera, är att implementationen som kommer att presenteras är oberoende av vilket pussel som används så när som på mindre modifikationer. Nedanstående figur (figure: 3.2) illustrerar tillvägagångssättet för verifiering av intentention genom Proof-of-work med användning av ett hashbaserat pussel.

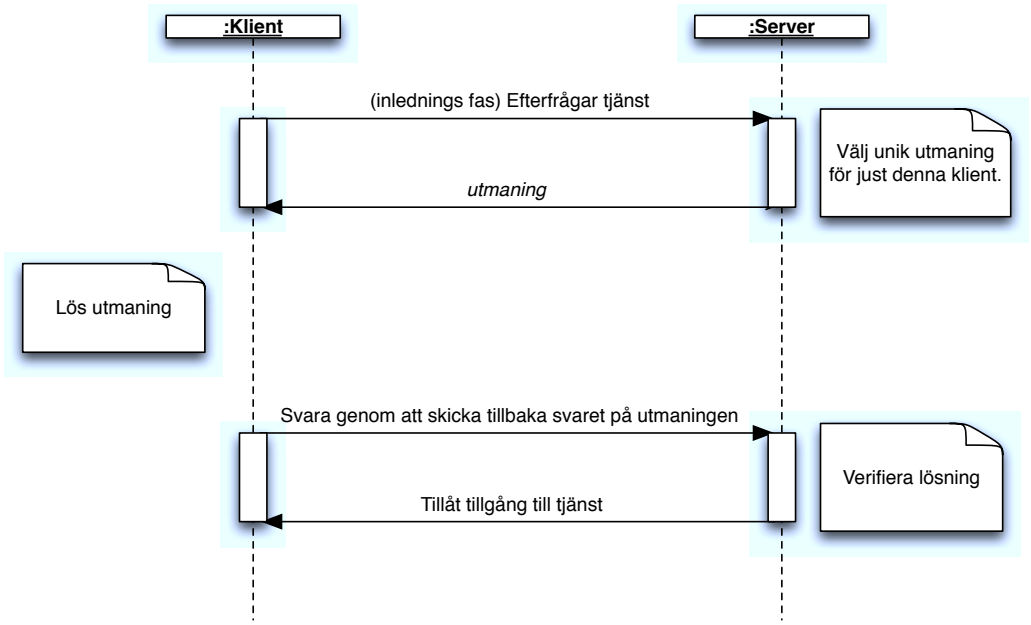



**Figur 3.2.** Proof-of-work protokoll

## 3.2 Challenge-response

Den enklaste formen av Challenge-response är ett autentiseringsprotokoll där en klient verfieras genom att en auktoritär komponent ställer en fråga ("challenge") och en tjänande komponent måste ge ett korrekt svar ("response") . Den vanligaste förekomsten av protokollet är lösenordsautentisering, där utmaningen är en förfrågan om lösenord och ett korrekt svar är det rätta lösenordet. CR kan även användas för att säkerställa svar på en utmaning som inte innehåller hemlig information, tex CAPTCHA's, vilket är en variant av Turingtestet [13].

Att utföra CR för att verfiera en klients identitet kan med fördel genomföras genom användning av ett *Zero-knowledge password proof* [2] genom att en klient inleder en handskakningsprocedur där denne utbyter användarnamn och påföljande lösenord i form av en hashsträng. På serversidan lagras samma lösenord, även här i hashad form, och en jämförelse av dessa kan således påvisa om en klient angett korrekt lösenord. En uppenbar svaghet är möjligheten till att bygga tabeller över vanliga lösenord och dess hashvärde med resultatet att en angripare effektivt kan göra en uppslagning på hashvärden och därmed få ut motsvarande lösenord, s.k. "Dictionary Attacks". En variant av sådana tabeller känt som "Rainbow Tables" [12] lagrar inte bara en viss mängd lösenord utan istället alla kombinationer av indata på ett kompakt sätt men till priset av att en uppslagning tar längre tid. Uppslagningen i ett Rainbow-table är endå betydligt snabbare än beräkning av invers till ett hashvärde. Detta har lett till utvecklingen, och användningen, av SALT's som motverkar denna typ av attacker genom tillägget av ett slumpgenererat värde till indata för varje lösenordshash [14]. Ett sådant tillägg omintetgör den praktiska nyttan av förberäknade värden på grund av mängden tabeller som måste upprättas och storleken på dessa. Figur 3.3 visar en grafisk representation av CR protokollet.




**Figur 3.3.** Challenge-response protokoll

# Kapitel 4

# Resultat

Att autentisera en klient, som beskrivet i den inledande problemformuleringen, består av två steg som förklarat tidigare. Först måste en server verifiera att en klient utfört ett visst arbete för att visa sin uppriktighet, samt även verifiera dennes inloggningsuppgifter för att kunna säkerhetsställa identiteten. Jag kommer nu att ge en beskrivning av hur dessa två steg kan kombineras till ett.

## 4.1 Autentisering av klient

Protokollet består av två faser. I initieringsfasen begär en klient att få upprätta en ny uppkoppling. Servern svarar genom att skicka ut ett nytt, individuellt anpassat, pussel. Pussel som genereras är tänkt att används som en grund för ett nytt pussel klienten sedan kommer modifiera originalpusslet till. Det är detta modifierade pussel som inkorporerar egenskaperna från Challenge-response. Den egenskap från CR som används i protokollet är *vetskapen om identitet* och detta görs genom att en klients lösenord inkluderas som indata i pusslet som skall lösas och således skapas en unik koppling mellan en klients lösenord och lösningen till pusslet. Det är denna koppling som en server sedan kommer att verifiera och genom en lyckad verifikation, kan då identitet samt intention för en klient styrkas.

### 4.1.1 Detaljerad beskrivning av autentiseringsprotokoll

Den del som är av störst betydelse för protokollet och dess säkerhetsrelaterade aspekter är det pussel som skapas. Pusslet som genereras av en server består av två delar: en global tidsbunden del bestående av en sträng som används vid samtliga klienters uppkopplingsförsök, samt en lokal som är unik för varje

klient, båda lagrade på serversidan. När pusslet sänds till en klient konkateneras de två delarna, som utgörs av två strängar, samman till pusselsträngen $S$ som beskrivet i sektion 3.1.1 och kan ses i figur 3.2 sidan 7. Svårighetsgraden på pusslet, $m$, utgörs av längden (antal tecken) av den sammanslagna pusselsträngen.

Syftet med de två olika delarna är att genom användning av den globala delen kan en attack riktad mot *servern* (dvs. inte någon specifik klient) enkelt motverkas genom att höja säkerheten (svårigheten på pusslet) globalt. Den globala delen är även låst till ett visst tidsintervall för att motverka att en klient skall kunna förberäkna eller återanvända tidigare pussel. Pusslet har även en del som är direkt knuten till en unik användare. Den användarknutna delen existerar för att ge möjligheten att på användarnivå styra svårighetsgrad och därmed motverka attacker (i form av Brute-force, Denial-of-service) mot enskilda användare utan att påverka övriga klienter. Det pussel som skickas ut skiljer sig från tidigare föreslagna lösningar (3.1) på flertalet sätt. Den mest utmärkande skillnaden är att aktuell klients lösenord inkluderas och används som en ingående parameter i den lösning som skall genereras. Att inkludera lösenordet som ingående parameter realiserar kravet på *vetskap om identitet* och realiseras på följande sätt:

Målet för klienten är att försöka lösa ett pussel med en Brute-force metod genom att försöka finna den unika sträng, $X$ som servern efterfrågar. Vid lösning av pusslet inkluderar klienten även en *hashad* form av sitt eget lösenord som kan ses som en *utökad* pusselsträng (jag kommer senare i texten referera till denna ihopslagning som $s\prime$) och konstruerar lösningen utifrån dessa värden. Klientens lösenord används därmed som en del avl pusslet och lösningen som söks baseras på detta. Det som söks efter, genom Brute-force, är därmed strängen $X$ i nedstående formel som genererar en hashrepresentation vars inledande substräng består av $m$ nollor ($h$ syftar på en hashoperation).

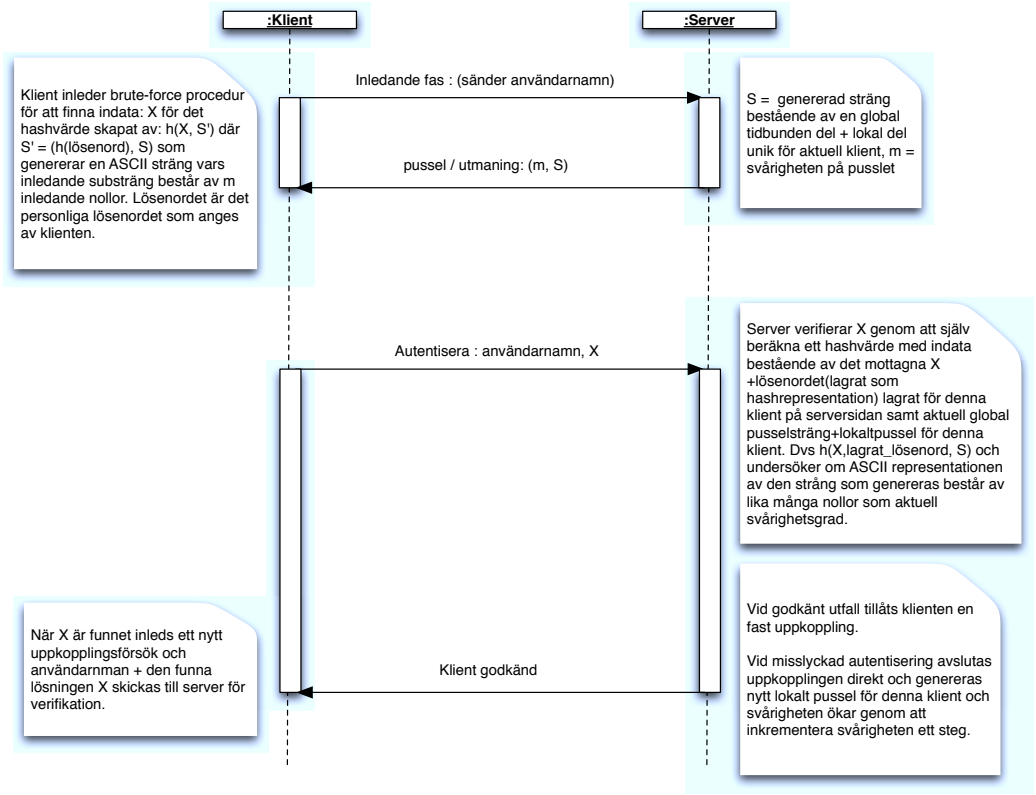
$$h(X, \underbrace{h(lösenord), serversträng}_{s\prime}) = m\ inledande\ nollor?$$

Då en klient funnit ett $X$ som uppfyller kravet på lösning av pussel fortsätter protokollet genom att ett nytt uppkopplingsförsök utförs varpå denna lösnings proposition sänds till servern.

Servern kontrollerar föreslagen lösningen genom att göra en uppslagning och hämta ut användarens lösenord samt unik klientsträng och testar om det är en korrekt lösning i förhållande till tidigare lagrade värden. Om lösningen är korrekt måste det innebära att (1) klienten har utfört ett arbete i och med att korrekt unik sträng har använts, samt (2) korrekt lösenord måste ha blivit angivet. Klienten är därmed autentiserad och tillåts därefter tillgång till

serverns resurser.

Fördelar med denna lösning är att det går att motverka Brute-force och Denial-of-service attacker på användarnivå genom att server kan höja säkerheten (svårigheten på individuellt pussel) vid upptäckt av felaktig lösning på pussel. Med andra ord används PoW som en dynamiskt motåtgärd för nämnda attacker mot klienter vid inloggningsprocedurer: svårigheten ökar med antal felaktiga inloggningsförsök (baserat på klient id). En grafisk representation av protokollet är beskriven nedan.




**Figur 4.1.** Challenge-response + Proof-of-work protokoll

Genom givet protokollförslag sänds aldrig något lösenord över någon osäker kanal, enbart en lösning på ett pussel där en klients lösenord är ett krav för korrekt lösning.

Användning av pussel på detta sätt uppfyller kraven på bra pussel givet i sektion 3.1.1. Att skapa pussel och generera hashvärden kan anses vara billigt för en server. Genom användning av en klientanpassad säkerhetsnivå är det möjligt att dynamiskt variera svårighetsgraden på ett pussel. Med tillägget

av den globala och tidbestämda delen motverkas möjligheten att beräkna lösningar i förväg. Sist men inte minst så kan ett hashpussel lösas oberoende av hårdvara då en hashfunktion kan implementeras helt i mjukvara.

### 4.1.2 Exempel på lyckad autentiseringsprocedur

För att enklare beskriva arbetsflödet i det exempel som följer inför jag några definitioner av värden som kommer att användas.

Initial antas server och klient inneha kunskap om definitionerna enligt vad som framgår av tabell 4.1 och 4.2. Exemplet är menat att visa på hur protokollet fungerar i den prototyp implementation som skapades för denna rapport. Övriga värden som genereras under protokollets livslängd är hypotetiska och specificeras allt eftersom.

| Klient definitioner | |
|---|---|
| Användarnamn | john.doe@runlevel5.se |
| Användares lösenord | “supersecret” |
| Hashvärde för lösenord | b29f7cb1a8aa1d8b12646fbec7f2a489ead88ebf |

**Tabell 4.1.** Information känd av *klient*.

| Server definitioner | |
|---|---|
| AnvändarID | john.doe@runlevel5.se |
| Hashvärde för lösenord | b29f7cb1a8aa1d8b12646fbec7f2a489ead88ebf |
| Global pusselsträng (slumpgenererad för visst tidsintervall) | “dXj” |
| Klientspecifik pusselsträng | “iw3k” |
| Aktuell säkerhetsnivå | 3 |

**Tabell 4.2.** Information känd av *server* (för användare john.doe@runlevel5.se)

Inledningsvis skickar klienten en signal till servern att påbörja inloggningsproceduren genom att skicka sitt eget användarnamn (john.doe@runlevel5.se).

Servern svarar genom att hämta ut information om pussel för denna klient, dvs unik pusselsträng samt aktuell säkerhetsnivå för denna klient, “iw3k” resp. “3”. Den specifika klientdelen av pusslet kombineras med den globala delen, vilket för tillfället är “dXj”. Jag påminner om att den globala delen är tidsbunden och en ny global del genereras med ett visst tidsintervall. Servern

skickar nu ett paket bestående av (3, “iw3kdXj”) till klienten och avslutar därefter uppkopplingen.

Klienten sammanfogar den mottagna *strängdelen* av pusslet med den *hashade* formen av sitt eget lösenord (“supersecret”) för att skapa det modifierade pusslet $s\prime$ och inleder en brute-forcing procedur enligt sektion 3.1.1 för att finna ett tal X som vid konkatenering med den hashade formen av en klients lösenord samt pusselstängen kan generera en hashrepresentation vars inledande *substräng* består av $m$ inledande nollor. I nedstående exempel är $m = 3$ (säkerhetsnivån för pusslet) och $s\prime$ består av klientslösenord(hashad form)+**pussel**.

$$h(X, s\prime) = 3\, inledande\, nollor?$$

$$s\prime = b29f7cb1a8aa1d8b12646fbec7f2a489ead88ebf\mathbf{iw3kdXj}$$

Klienten löser detta pussel med resultatet att om $X = 4711$ (hypotetisk lösning), genererar det en hashsträng givet av formeln: *h(X,s′)* där den inledande substrängen består av 3 inledande nollor. Jag kommer att kalla hashvärdet $X$ för *lösningen* till ett pussel. En ny uppkoppling initieras av klienten med en signal till servern att den nu skall verifiera en lösning. Den information som klienten skickar tillbaka till servern består av användarnamn samt lösning.

Servern konstruerar nu en egen version av lösningen för denna användare genom att generera en hashsträng utifrån bifogad *lösningssträng*: $X$, och en egen versionen av $s\prime$ bestående av lösenordshash lagrat på server sidan för denna klient samt lokalt och globalt pussel som är aktuellt förtillfället. Den server genererade versionen testas ifall den inledande substängen verkligen består av lika många inledande nollor som angivet i säkerhetsnivån för klienten, om så är fallet är lösningen godkänd.

$$h(4711, s\prime) = 3\, inledande\, nollor?$$

Då testet ger ett godkänt utfall, innebär det att klienten har använt samma lösenord som finns lagrat hos servern vid lösning av pusslet och som tidigare påpekat, någon form av arbete har blivit utfört.

## 4.2 Analys av föreslaget protokoll

Designidéen är att en server inte ska utföra något arbete, eller rättare sagt så lite som möjligt, innan en klient är autentiserad. Det leder till att en server bör stänga en uppkoppling så fort information skickats och inte vänta på bekräftelse, om ej nödvändigt, detta för att minska de tillfällen en server kan

fastna i ett väntande stadium då ett sådant tillstånd potentiellt kan utnyttjas vid överbelastningsattacker.

När en server svarar på en förfrågan kan det ske på två legitima sätt. Antingen är det en ny klient som vill skapa en ny anslutning eller så är det en klient som har utfört ett arbete och vill få bekräftelse på detta och sedemera få tillgång till serverns resurser. Servern kan anta att detta är de enda former av uppkoppling som kommer att ske, då det är de enda vi är intresserade av, - och därmed kan övriga typer (potentiella attacker) ignoreras. Eliminationen av tillstånd kan realiseras genom att varje utbyte av information med en server kan ses som en atomisk operation där endast *ett* av något av de två möjliga operationerna *förfrågan om pussel* eller *verifiering av pussel* är godkända.

Tilläggas kan att en server inte behöver hålla reda tillstånd för klienter mellan uppkopplingar, det vill säga, mellan det att ett pussel lämnas ut till dess att en lösning mottas. Den information som används för att auktorisera en klient, nämligen lokal och global pusselsträng, är oberoende av vilket tillstånd en klient förtillfället befinner sig i. Exempelvis kan det ske att en klient som inte orsakar en förändring av sin egen säkerhetsnivå, genom att möjligtvis ange fel lösenord, kan mottaga ett pussel där lokaldelen är samma som vid ett tidigare tillfälle. Dock så påverkas inte pusslet av denna upprepning på grund av av utökningen med en global del.

Det lätt att inbilla sig att sökrymden för möjliga lösningar till ett pussel ökar då antal ingående parametrar ökar. En sådan ökning borde innebära att mer resurser krävs av en angripare att invertera en lösning med målet att få fram lösenordet för denna lösning, detta är tyvärr inte fallet. Då en en viss svårighetsgrad angetts i form av parametern $m$ så innebär det istället en *begränsning* av sökrymden genom att en angripare enbart behöver variera den ingående text representationen av lösenordet och sedan endast intressera sig för resultat som ger en hashrepresentation vars inledande substräng består av $m$ inledande nollor och kasta bort övriga samt att denna form av brute-forcing kan utföras i nedkopplat läge, dvs. inga test måste göras direkt mot en server. Dock ger detta även en fördel. Genom en begränsning av sökrymden för möjliga lösningar till pussel, *ökar* istället antal möjliga kandidater till lösenord. Detta genom definitionen av hur en hashfunktion fungerar, nämligen att olika indata *kan* ge samma hashrepresentation. Genom givna argument är det rimligt att resonera att en ökad svårighetsgrad på pussel minskar antal kandidatlösenord men ökar sökrymden samt omvänt, en minskad svårighetsgrad ökar antal kandidatlösenord men minskar sökrymden.

För att summera resonemanget: en angripare kan i bästa fall lyckas ta reda på en, eller flera, kandidater till korrekt lösenord och dessa kandidater kan lösa ett unikt pussel. För att ta reda på vilken kandidat som är den korrekta (dvs. den som är lagrad hos servern) måste en angripare utföra detta test i

uppkopplat läge, mot en server, och då lösa ett nytt pussel för att säkerställa att ett potentiellt kandidatlösenord verkligen är det korrekta.

Ur detta perspektiv kan lösenordets betydelse i förslagen lösning ses som ett klientgenererat SALT. Alla "test" av lösenord en angripare försöker sig på måste gå via en server där angriparen tvingas lösa ett nytt pussel.

### 4.2.1 Svagheter

Om en genuin användare försöker upprätta en ny förbindelse kan det hända att denna får ett potentiellt svårt pussel till följd av ett tidigare försökt till attack, men vid korrekt lösning så nollställs svårigheten återigen. Resultatet är att det blir ointressant för en angripare att fortsätta attacken då knäckandet av flertalet pussel tar lång tid (pga svårt pussel) och genuin användare inte påverkas nämnvärt (effekten av en angripares inloggningsförsök) då denne enbart behöver lösa *ett* svårt pussel en enda gång.

För att en klient skall kunna lösa ett potentiell väldigt svårt pussel som resultatet efter en attack innebär det att någon form av övre gräns för svårighetsgrad måste identifieras. Användningen av en övre gräns kan utgöra både en fördel eller en nackdel. Nackdelen är att det kan ta väldigt lång tid för en klient att lösa pusslet med resultatet att klienten upplever systemet som långsamt och obrukbart. Fördelen är att svåra pussel uppenbart försvårar ytterligare för angripare. En avvägning mellan de två skulle kunna vara att meddela klienten om att dess konto troligtvis blivit utsatt för en attack vid mottagande att en väldigt svårt pussel och uppmana denna att istället inleda någon form av återställningsprocedur som även innebär att välja ett nytt lösenord.

Då PoW bygger på mängden av beräkningskapacitet hos en klient för att lösa ett pussel kan det vara svårt att avgöra var gränsen går för vad som kan anses vara ett enkelt eller svårt pussel. Med antagandet att beräkningskapaciteten ökar med tiden, exempelvis enligt approximationer såsom *Moores lag*, kan pussel som anses vara svåra i dagsläget möjligtvis ses som lätta i framtiden. Det innebär att nivån på vad som anses vara svårt är starkt knuten till vad som kan uppfattas som normen för beräkningskapacitet och måste således förändras över tiden. En viss implementation av Proof-of-work system kan därmed inte anses vara slutgiltig och statisk utan måste förändras över tiden.

En ständigt överhängande risk vid nätverkskommunikation över icke säkra kanaler är man-in-the-middle attacker. Det existerar en svaghet i föreslaget protokoll där en angripare eventuellt kan snappa upp användarnamn och lösning till ett pussel och använda detta för att autentisera sig för en server. Detta kräver att angriparen både stjäl informationen och hindrar klientens förfrågan

att nå server, alternativt, i sin tur skickar vidare informationen så den når servern innan den ursprungliga klientens information når fram. Servern kan då inte veta vem som skickade vad och tillåter därmed tillgång till den vars information når fram först.

Ytterligare ett krav för genomförbarhet är att en angripare inte enbart stjäl en klients information, utan hela sessionen, vilket ställer andra krav på säkerhet som är utanför ramen för denna text men som exempel kan nämnas att en angripare då behöver genomföra en lyckad *spoofing* attack. Genom de ytterligare externa krav som ställs för denna typ av attack dras slutsatsen att genom användning av Proof-of-work så minskar sannolikheten för lyckad attack genom att ställa ytterligare krav för genomförbarhet. I fallet att återanvända en korrekt lösning av tidigare pussel (inom tidsramen för pusslet) kan det omintetgöras genom att markera lösning som icke längre giltig. Markeringen av använda lösningar uppfylls på samma sätt som tillvägagångssättet för ett misslyckat autentiseringsförsök. Nämligen att låta servern generera en ny pusselsträng. Anledningen till att det räcker med att generera en ny lokalpusselsträng är att då den används tillsammans med den globala delen så är risken för två genererade strängar kommer vara identiska är minimal. Det innebär då att pussel inte kan återanvändas samt om två korrekta lösningar skulle mottas i följd så godkänns enbart den första. Detta bör ske på specifik användare och inte på pussel då flera användare kan, med stor sannolikhet, ha samma lösenord.

Då Challenge-response utmaningen inkluderas i Proof-of-work pusslet och då användningen av PoW innebär att ge en ledtråd till korrekt lösning i form av information om svårighetsgrad innebär det att en viktig egenskap för CR går förlorad. Den information som PoW bidrar med om svårighetsgraden för ett pussel betyder det att en länk skapas mellan lösningen till ett pussel och vilket lösenord som användes för att finna denna lösning. Under förutsättningen att en angripare har lyckats sniffa pusselsträng och svårighetsgrad kan vetskapen om denna länk ses som en *surjektion* (i dess matematiska betydelse inom mängdläran) till vilka lösenord som kan vara möjliga. Det är just existensen av denna surjektion som skapar begränsningen av sökrymden för lösningar, och därmed lösenord, för ett pussel.

# Kapitel 5

# Slutsats

Genom stora likheter är det enkelt att kombinera Proof-of-work med Challenge-response. Den utmaning som en server ställer i Challenge-responseprotokollet kan ses som en specialanpassning av Proof-of-work.

I mitt förslag till autentiseringsprotokoll görs antagandet att säkerhetsnivån ska ökas för varje misslyckat inloggningsförsök. Anledningen till denna ökning är att jag såg ett behov av att på ett enkelt sätt kunna avgöra om ett användarkonto potentiellt var under attack. Det mest troliga beteendet för en användare som glömt bort sitt lösenord är att göra ett par inloggningsförsök, om dessa inte lyckas, gå vidare och försöka använda någon form av återställningsmekanism. En slutsats som då kan dras är att om säkerhetsnivån för ett konto har ökat till en relativt hög nivå har kontot troligtvis varit utsatt för en attack.

Funktionalitet för att upptäcka attacker kunde även ha skapats genom användning av en räknare som inkrementeras. Men genom att istället använda en dynamiskt ökande säkerhetsnivå innebär det att *ju fler* attacker en angripare försöker utföra mot samma konto *desto svårare blir de att slutföra.*

Tanken om att kunna autentisera en klient mot en server utan att låta känslig information gå över någon osäker kanal är intressant. För att möjliggöra detta krävs det att två parter kan bevisa för varandra att de verkligen är vem de utger sig för att vara. Från den undersökning jag genomfört så drar jag slutsatsen att för att lyckas bevisa för en annan part att en klient är av en viss identitet så *måste* någon form av information utbytas som bara den ena parten kan ha kunskap om, samt denna information måste kunna verfieras av den andra parten. Information som utbyts behöver inte innehålla den känsliga informationen i sig, utan istället *bevis av kunskap* dvs att en klient skulle kunna bevisa att den vet något utan att avslöja vad det är den vet om. Föreslaget protokoll är inte lösningen på denna typ av problem då det innehåller brister i form av att det pussel som används innehåller allt för uppenbara surjektioner

mellan en lösning till ett pussel och den känsliga information som användes för att skapa denna lösning. En angripare kan enkelt testa sig fram till ursprungs informationen genom de ledtrådar som protokollet bygger på och gör således att denna form av autentisering inte kan anses vara säker.

Den egentliga enda praktiska nyttan av PoW är att det förhindrar en angripare att genomföra flertalet angrepp i direkt följd, istället så möjliggör PoW en spridning av dessa anrop en aning över tiden (pga. av det pussel som trots allt måste lösas).

Den lilla spridning som kan tänkas vinnas kan dock ge en viss fördel genom att till viss del jämna ut en trafiktopp då oavsett vilken mängd resurser en angripare kan tänkas besitta så kräver ändå lösningen av ett pussel en viss mängd tid, och dessutom ökar tiden att lösa ett pussel med antal felaktiga försök. Proof-of-work bidrar därmed med en viss *load-balancing*. Resonemanget är dock teoretiskt och har inte testats i praktiken.

En prototyp implementation av protokollet skapades i Java i form av en klient/server-modell. Källkoden till prototypen är nästan 900 rader kod stor och anses för stor för att publisera i denna rapport. För den intresserade kan källkoden hämtas på internet på adressen: http://www.runlevel5.se/pow-cr.zip

# Figurer



# Tabeller

# Litteratur


[1] Tuomas Aura, Pekka Nikander och Jussipekka Leiwo. "DOS-Resistant Authentication with Client Puzzles". I: *Security Protocols, 8th International Workshop Cambridge, UK, April 2000, Revised Papers*. Utg. av B. Christianson m. fl. Vol. 2133. Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, 2001, s. 170–177.

[2] Steven M. Bellovin och Michael Merritt. "Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and password file compromise". I: *Computer and Communications Security*. 1993, s. 244–250.

[3] *Brute force attacks*. Besökt: 2011-02-20. URL: `http://www.owasp.org/index.php/Blocking_Brute_Force_Attacks#Finding_Other_Countermeasures`.

[4] Microsoft Safety & Security Center. *Online Privacy & Safety*. Besökt: 2011-03-01. URL: `http://www.microsoft.com/security/online-privacy/passwords-create.aspx`.

[5] *Challenge-response Authentication*. Besökt: 2011-01-14. URL: `http://en.wikipedia.org/wiki/Challenge-response_authentication`.

[6] Cynthia Dwork och Moni Naor. "Pricing via Processing or Combatting Junk Mail". I: *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '92. London, UK: Springer-Verlag, 1993, s. 139–147. ISBN: 3-540-57340-2.

[7] Virgil D. Gligor. "Guaranteeing Access in Spite of Distributed Service-Flooding Attacks". I: *In Proceedings of the Security Protocols Workshop*. 2003.

[8] Mark Handley och Adam Greenhalgh. "Steps towards a DoS-resistant internet architecture". I: *ACM SIGCOMM Conference*. 2004.

[9] Sean Michael Kerner. *Denial of Service Attacks Get more Sophisticated*. Besökt: 2011-02-20. URL: http://www.esecurityplanet.com/trends/article.php/3921156/Denial-of-Service-Attacks-Get-more-Sophisticated.htm.

[10] Ben Laurie och Richard Clayton. ""Proof-of-Work" Proves Not to Work". I: *Proceedings of WEIS*. 2004.

[11] *Proof-of-work system*. Besökt: 2011-01-15. URL: http://en.wikipedia.org/wiki/Proof-of-work_system.

[12] *Rainbow table*. Besökt: 2011-03-20. URL: http://en.wikipedia.org/wiki/Rainbow_table.

[13] Carnegie Mellon University. *CAPTCHA: Telling Humans and Computers Apart Automatically*. Besökt: 2011-03-20. URL: http://www.captcha.net/.

[14] Christoph Wille. *Storing Passwords - done right!* Besökt: 2011-05-01. URL: http://www.aspheute.com/english/20040105.asp.

[15] Thomas Wu. "The secure remote password protocol". I: *In Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*. 1998, s. 97–111.