Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Matematica

Protocolli di autenticazione basati su password: dai primi esempi a quelli a conoscenza zero

Tesi di Laurea in
Algoritmi di Teoria dei numeri e Crittografia

Relatore:
Chiar.mo Prof.
Davide Aliffi

Presentata da:
Leda Vecchi

Sessione Unica
Anno Accademico 2018/2019

"Le forme create dal matematico, come quelle create dal pittore o dal poeta, devono essere belle; le idee, come i colori o le parole, devono legarsi armoniosamente. La bellezza è il requisito fondamentale: al mondo non c'è un posto perenne per la matematica brutta."
(G.H. Hardy)

Ai miei Nonni.

Indice

Elenco delle figure

Elenco dei Protocolli

Introduzione

Scopo di questa tesi è quello di analizzare le varie possibilità, con i relativi limiti e punti di forza, di alcuni protocolli di autenticazione, in particolare di quelli basati sull'uso di una password.

Col termine autenticazione, indichiamo un processo che verifica la veridicità di un'affermazione, in particolare consideriamo l'autenticazione come la verifica dell'identità di un'entità che partecipa alla conversazione.

Infatti lo scenario che andremo a considerare è quello che descriviamo in breve di seguito: ci sarà un utente umano che desidera accedere a dei servizi forniti da un sistema, da un host. Questi servizi prevedono lo scambio di informazioni riservate e sensibili, perciò è necessario che l'utente venga identificato e autenticato dall'host senza che nessuna informazione utile venga fornita a possibili avversari o in generale a terze parti.

In questa tesi, andremo inoltre a considerare il caso di protocolli che permettano questo tipo di comunicazione sicura e in particolare vedremo il caso in cui l'autenticazione dell'utente è realizzata grazie all'uso di una password nota all'utente.[1]

Notiamo infatti subito la differenza tra una password e una chiave. Una chiave è una stringa di bit di lunghezza variabile in base alla tipologia di protocollo grazie alla quale vengono cifrati messaggi e informazioni importanti. Inoltre, poiché le chiavi sono sequenze pseudo-casuali di bit di lunghezza solitamente superiore al centinaio, esse vengono principalmente utilizzate dai server, dai computer e dagli host perché non facilmente memorizzabili. Invece la password è una sequenza di caratteri relativamente corta, affinché possa essere memorizzata facilmente dagli esseri umani, e venga utilizzata come strumento per l'autenticazione.

Nella tesi andremo ad analizzare, dopo un primo capitolo con nozioni utili alla comprensione dell'argomento e degli scenari specifici, i vari protocolli ideati per fornire soluzioni al problema descritto.

Inizieremo vedendo quali sono i protocolli più intuitivi e immediati, che si riveleranno troppo ingenui e quindi difettosi, poiché più o meno facilmente attaccabili. In seguito, andremo a presentare una tipologia di protocollo, detta EKE, più sicura che prevede l'uso della password come chiave di cifratura simmetrica per lo scambio sicuro di una chiave di sessione tra utente e host. Infine, nonostante il protocollo EKE sia sufficien-

[1] Le nozioni di base contenute nell'introduzione rinvia a [1, 2, 3]

temente sicuro, introdurremo un'ultima tipologia di protocollo, detta a conoscenza zero, che semplifica in numero e contenuto le comunicazioni necessarie tra le due parti; inoltre, sottolineeremo l'aspetto più importante di questi protocolli, ossia che non viene inviata sul canale pubblico alcuna informazione utile sulla password segreta, tanto meno la stessa password.
Di quest'ultima tipologia di protocollo, vedremo anche come vengono realizzate alcune applicazioni pratiche nella vita quotidiana.

Capitolo 1

Premesse concettuali e nozioni di base

1.1 Introduzione ai protocolli di autenticazione

Iniziamo la trattazione introducendo il concetto di autenticazione e di protocollo:[1]

Definizione 1.1.1. Chiamiamo **autenticazione** la procedura grazie alla quale un'entità stabilisce la proprietà richiesta da un'altra entità. Per esempio quando un computer, o un server, stabilisce se un utente, che rivendica il diritto all'utilizzo del servizio in questione, ne ha a tutti gli effetti il diritto.

Definizione 1.1.2. Chiamiamo **protocollo** una procedura di comunicazione tra due parti.

Possiamo suddividere il concetto di autenticazione in tre nozioni principali, di cui manteniamo la terminologia inglese:

- ***data-origin authentication***
- ***entity authentication***
- ***authenticated key establishment***

1.1.1 Data-origin authentication

Il *data-origin authentication*, che viene anche chiamato ***message authentication***, è spesso associato al concetto di *data integrity*[2].
Vediamo in cosa consistono entrambe le nozioni e in cosa differiscono.

[1] Le definizioni e i concetti qui riportati in sintesi rinviano alle sezioni 11.1-11.4 di [2], al capitolo 1 di [1] e ai capitoli 2 e 5 di [4].

[2] Definizione tratta dal sito [9] in data 30/08/2019.

Definizione 1.1.3. Definiamo ***data integrity*** (spesso tradotto con l'espressione italiana *integrità dei dati*) la proprietà che permette di verificare se un dato o una informazione siano stati alterati nel loro contenuto durante il processo di trasmissione in una comunicazione o durante la memorizzazione.

Notiamo quindi che il processo di verifica della *data-integrity* non avviene esclusivamente durante una comunicazione, ma può realizzarsi anche in un archivio di dati. Inoltre, questo processo permette di vedere se una terza parte ha messo mano alle informazioni modificandone il contenuto. Infine osserviamo che questo processo non richiede l'identificazione del mittente del messaggio.

Possiamo invece schematizzare il concetto di *data-origin authentication* come segue:

Definizione 1.1.4. Definiamo ***data-origin authentication*** relativa alla trasmissione di un messaggio da una prima parte, il mittente, ad un'altra parte, il ricevente, il quale dovrà validare il messaggio ricevuto. Gli obbiettivi di questo processo e della validazione sono:

1. stabilire l'identità del mittente del messaggio;
2. stabilire il *data integrity* del messaggio, dal momento in cui il messaggio è stato inviato dal mittente;
3. stabilire la *liveness* del mittente del messaggio. *Liveness*, che in inglese significa la condizione di essere vivi, in questo caso viene usata nel senso di essere presenti, attivi. Ovvero un partecipante alla comunicazione è presente se è online, pronto a ricevere messaggi e a rispondere ad eventuali richieste.

In realtà nel punto 2. non solo si cerca di stabilire l'integrità del messaggio, ma se ne determina anche la cosiddetta ***freshness***.

Definizione 1.1.5. Definiamo la ***freshness*** di un messaggio in base a quanto recentemente è stato inviato il messaggio in questione. Ovvero parliamo di messaggi "freschi" quando l'intervallo di tempo che intercorre dal momento il cui il mittente lo ha spedito a quello in cui il ricevente lo ha effettivamente ricevuto è un intervallo di tempo relativamente breve.

La richiesta di ricevere messaggi "freschi" si fonda sul buon senso, perché questa condizione implica che ci sia una buona linea di comunicazione tra i partecipanti alla conversazione ed implica inoltre che ci sia una probabilità più ridotta che il messaggio o il sistema vengano sabotati oppure che uno dei partecipanti venga ingannato.

L'ordine di grandezza dell'intervallo e il suo grado di tolleranza vengono determinati dalle applicazioni, perché ognuna ha un ordine di grandezza appropriato (per esempio, Enigma durante la Seconda Guerra Mondiale cambiava chiavi una volta al giorno, mentre alcune banche considerano valido un assegno fino a 3 mesi dal momento della sua

immissione e, negli schemi di *challenge-response* che vedremo, l'intervallo tollerato è di pochi secondi).

Esistono vari strumenti per verificare la *freshness* di un messaggio. La tecnica che utilizzeremo in questa analisi prevede quasi sempre l'uso di ***nonces***, ovvero un numero o una stringa casuale che viene utilizzato un'unica volta (appunto dall'inglese "*number used once*"). Prima di vedere come vengono usati i *nonces*, vediamo un'altra definizione:

Definizione 1.1.6. Chiamiamo **funzione hash crittografica** o **funzione hash one-way** una funzione $h(x) = y$ tale che l'immagine y sia una stringa più corta della stringa di partenza x. Possiamo quindi vedere le funzioni hash come una funzione che comprime l'argomento dato.

Affinché la funzione sia *one-way* o crittografica, è necessario che la funzione sia resistente alle collisioni, ovvero che sia difficile trovare due elementi x, x' che abbiano la stessa immagine, $h(x) = h(x')$.

Formalizziamo e generalizziamo la definizione di **funzione hash one-way** come segue:

Definizione 1.1.7. Definiamo **famiglia di funzioni hash one-way** una famiglia $H = \{h_i : D_i \to R_i\}$ tale che:

- sia facile campionare l'insieme H e ottenere una funzione h_i
- si abbia una compressione: $|D_i| > |R_i|$
- sia facile calcolare la funzione su un elemento fissato, ovvero sia facile il calcolo $h(x)$
- sia una funzione resistente alle collisioni, ovvero per ogni algoritmo A polinomiale la probabilità che l'algoritmo A, preso il valore y, riesca ad ottenere due elementi distinti x e x' tali che $h(x) = h(x') = y$ è una probabilità trascurabile.

Quindi i *nonces* vengono usati per verificare la *freshness* di un messaggio nel seguente modo: una parte, A, sceglie un numero casuale N_A e lo manda all'altra parte B. Quest'ultima rimanda il valore ad A dopo aver applicato una funzione hash crittografica. Una volta ricevuto il valore, A verificherà se la funzione hash è stata applicata al proprio *nonce* e quindi se il messaggio è "fresh" o no.

1.1.2 Entity authentication

Definizione 1.1.8. Definiamo come ***entity authentication*** un processo attraverso il quale un partecipante stabilisce una comunicazione attiva (*lively*) con un'altra parte la cui identità rispecchia quella ricercata dalla prima.

Ci sono diverse tipologie di scenari possibili che dipendono dalla classificazione dei partecipanti. Vediamo quali sono i principali:

- **host-host**: i due partecipanti alla conversazione sono due computer, detti *nodi* del sistema distribuito[3]. Poiché in questo tipo di sistema sono spesso necessarie cooperazioni tra i diversi nodi, vengono usati i protocolli di autenticazione per stabilire che l'altra parte della comunicazione sia fidata. Di questo scenario fanno parte i casi di *client-server* in cui il primo richiede un certo servizio al secondo.

- **user-host**: un utente, user, ottiene l'accesso ad un sistema di computer accedendo ad uno degli host del sistema. Nei casi più semplici è sufficiente eseguire un protocollo di autenticazione grazie all'uso di password. In altri casi più complessi serve che ci sia autenticazione reciproca.

- **member-club**: questa tipologia è una generalizzazione del caso user-host, perché un membro del club per accedere deve mostrare di possedere le credenziali. In questa tipologia, l'altra parte controlla semplicemente che la prima possieda le credenziali necessarie senza preoccuparsi di quale sia effettivamente la sua identità.

In questa trattazione, vedremo principalmente la seconda tipologia, l'autenticazione tra utente ed host, in particolare nel caso che si basa sull'uso di password.

Vedremo inoltre anche l'ultimo scenario nei protocolli a conoscenza zero.

1.1.3 Authenticated key establishment

Spesso i partecipanti ad un protocollo di autenticazione lo eseguono come mezzo per stabilire comunicazioni future sicure. Infatti vengono usati i protocolli di autenticazione anche come strumento per stabilire una chiave di sessione, ovvero una chiave di cifratura simmetrica temporanea.

I protocolli di autenticazione che prevedono anche come obiettivo quello di stabilire una chiave segreta comune, vengono chiamati anche **authenticated key establishment** o **key exchange**.

Un esempio di questo tipo di protocollo che vedremo successivamente è il protocollo EKE, *Encrypted Key Exchange*.

[3]Possiamo considerare la definizione di *sistema distribuito* quella fornita da Maarten van Steen nel 2016: "Un sistema distribuito è una porzione di software che assicura che un insieme di calcolatori appaiano come un unico sistema coerente agli utenti del sistema stesso". Esistono altre definizioni equivalenti e più approfondite, ma questa è sufficiente per quanto riguarda la comprensione dello scenario. Tratta dal sito [10] in data 30/09/2019

1.1.4 Qualche tipologia di attacco

Poiché l'obbiettivo dei protocolli di autenticazione è quello di verificare la rivendicazione di una determinata proprietà, le tecniche di crittografia vengono utilizzate all'interno di questi protocolli. Ne segue che, parlando di protocolli di autenticazione, ci si riferisca anche ai relativi attacchi possibili.

Un attacco ad un protocollo di autenticazione consiste in uno o più attaccanti che riescono ad ottenere qualche risultato non previsto, per esempio la chiave segreta o riescono ad ingannare una delle due parti.
Consideriamo che il protocollo sia difettoso se alla fine dell'esecuzione di un protocollo, le due parti sono arrivate a conclusioni differenti. Si noterà durante la trattazione che i protocolli non sono sicuri non perché le tecniche crittografiche utilizzate siano difettose, ma principalmente perché un avversario può impedire che l'autenticazione abbia successo senza violare necessariamente gli algoritmi crittografici.

Vedremo diversi attacchi durante la trattazione, che non rappresentano la totalità di quelli possibili. Vediamo quali sono i principali:

- **attacco *eavesdrop*** (dall'inglese "origliare"): è il tipo di attacco di base e prevede che l'avversario riesca a carpire delle informazioni passate nel canale di comunicazione durante l'esecuzione del protocollo. Solitamente questo tipo di attacco viene scongiurato utilizzando delle tecniche di cifratura nei messaggi che è opportuno non vengano intercettati. Viene considerato un attacco di tipo passivo perché non è necessario che l'avversario disturbi la conversazione tra le legittime parti.

- **attacco di "modificazione"**: l'avversario altera le informazioni inviate nel protocollo. Una delle tecniche utilizzate prevede che il messaggio venga suddiviso in più frammenti e assemblato in un messaggio diverso. Le tecniche di cifratura non possono sempre prevenire questo tipo di attacco.

- **attacco replay**: l'avversario registra alcune informazioni carpite dal protocollo e poi le rinvia ad uno delle due parti della conversazione solitamente in un'esecuzione diversa del protocollo. Quindi l'attacco prevede che l'avversario si inserisca nella comunicazione inviando un messaggio o una parte di esso che aveva carpito da una precedente esecuzione del protocollo. Un caso particolare di questo tipo di attacco è l'*attacco di riflessione* in cui l'avversario manda indietro l'ultimo messaggio inviato a colui che lo aveva inviato in primo luogo.

- **attacco dizionario**[4] consiste nel cercar di "rompere" un cifrario o un protocollo di autenticazione tentando solo le chiavi o le password ritenute più probabili. Queste chiavi formano una lista ovvero il cosiddetto dizionario. Questo tipo di attacco funziona in particolare quando si parla di password che vengono scelte dalle persone,

[4]Definizione tratta dal sito [11] in data 31/08/2019.

perché hanno la tendenza ad usare parole della propria lingua, date di nascita o di avvenimenti importanti. In generale si tratta di stringhe non casuali ma con un significato specifico affinché siano più facili da ricordare.

- **attacco *man-in-the-middle***[5] prevede che un avversario si inserisca in una conversazione tra due parti. L'avversario può comportarsi in due modi: in modo passivo, se l'unico atto che compie è ritrasmettere i messaggi che riceve da una parte all'altra e viceversa, come in un attacco di *eavesdrop*; nell'altro caso, quello attivo, altera i messaggi trasmessi durante la conversazione. In generale in questo tipo di attacco, i partecipanti non si accorgono della presenza di una terza parte ma sono convinti di comunicare attraverso un canale sicuro o privato.

- **attacco di forza bruta**: è l'unico tipo di attacco che non possiamo mai escludere, perché consiste nel tentare tutte le possibili chiavi, o password, in maniera esaustiva finché non si trova quella corretta.

1.1.5 Convenzioni

Vediamo una serie di convenzioni su cui fonderemo la trattazione e che riguardano sia il comportamento dei partecipanti sia di eventuali avversari.

- un avversario è sempre in grado di intercettare tutti i messaggi mandati in un protocollo crittografico

- un avversario è in grado di modificare tutti i messaggi inviati in un protocollo crittografico usando le informazioni disponibili. Inoltre, può re-indirizzare un qualsiasi messaggio ad una qualche altra parte e ciò gli permette di generare ed inserire messaggi completamente nuovi nella comunicazione

- l'avversario può essere uno dei partecipanti legittimi alla comunicazione o può essere un elemento esterno o una combinazione dei due

- un avversario è in grado di ottenere il valore della chiave di sessione usata in comunicazioni ed esecuzioni del protocollo sufficientemente vecchie

- un partecipante onesto al protocollo non riesce a comprendere il significato di alcun messaggio del protocollo, prima che un'esecuzione del protocollo venga completata

- un partecipante onesto al protocollo non riesce a riconoscere un messaggio cifrato da una generica stringa casuale, quindi non può scomporla né ricrearla a meno che non conosca la chiave corretta

[5] Definizione tratta dal sito [12] in data 31/08/2019.

- un partecipante onesto non riesce a distinguere un *nonce* o una sequenza di numeri o una chiave di cifratura da una stringa casuale a meno che non sia stata creata da lui stesso nell'esecuzione corrente del protocollo o sia il risultato ottenuto dall'esecuzione del protocollo

- un partecipante onesto non salva alcun messaggio del protocollo se non specificato dal protocollo stesso. Infatti i protocolli di autenticazione sono definiti *stateless* perché non viene richiesto di memorizzare alcuna informazione di stato dopo che la sua esecuzione sia terminata ad eccezione del risultato ottenuto

- un avversario è a conoscenza delle debolezze dei partecipanti e cerca sempre di sfruttarle

1.2 Schemi di *challenge-response*

Abbiamo visto che due caratteristiche importanti relative al *data-origin authentication* sono la *freshness* dei messaggi e la *liveness* dei partecipanti. Sono pertanto necessari meccanismi per verificare tali proprietà e uno di questi è il meccanismo di ***challenge-response*** o **sfida-risposta**.

Questo tipo di schema prevede due partecipanti, Alice e Bob. Supponiamo di essere nel caso in cui Alice voglia autenticarsi a Bob. Quest'ultimo le manderà un messaggio rappresentante una sfida, *challenge*, alla quale Alice dovrà dare una risposta. Se la risposta è corretta, allora Alice viene autenticata e si dà inizio alla comunicazione; altrimenti viene respinta la sua richiesta di comunicazione.
Notiamo che, anche se un avversario intercettasse i messaggi dei vari passaggi di questo schema, non riuscirebbe ad ottenerne informazioni utili perché ad ogni esecuzione del protocollo la sfida cambierebbe e di conseguenza la risposta.

Per esempio quando alcuni siti internet richiedono l'accesso di un utente, spesso per dimostrare di essere umani e non macchine, viene richiesta la risposta ad una sfida che di volta in volta cambia. Solitamente si ha un riquadro con numeri e lettere distorte e si chiede all'utente di digitare ciò che legge. Questo è un esempio di *challenge-response* perché permette al sito di determinare se l'utente è umano o no in base alla risposta data (i computer non sono in grado di riconoscere i caratteri distorti dell'immagine, nonostante questa sia un'operazione facile per l'essere umano); di volta in volta la sfida cambia perciò non è possibile ottenere informazioni utili intercettando la risposta.

Vediamo da cosa è costituito lo schema nel caso di cifrari simmetrici:

Protocollo 1. Schema di *challenge-response*

Premesse: A e B condividono una chiave K_{AB} relativa ad un cifrario simmetrico.
B sceglie un *nonce* N_B.

Obiettivo: A riesce ad autenticarsi con B.

1. B $\rightarrow$ A : N_B
2. A calcola $c = \{N_B\}_{K_{AB}}$
3. A $\rightarrow$ B : c
4. B usa K_{AB} per decifrare c: se trova il valore di N_B, autentica A ed accetta la comunicazione; altrimenti non riconosce A e ne rifiuta la comunicazione.

Questo protocollo realizza gli obiettivi di *freshness* e *liveness* perché B sceglie il proprio *nonce* appena prima di iniziare il protocollo; perciò quando B riceve il messaggio cifrato con K_{AB}, se decifrandolo trova N_B, ne dedurrà che l'operazione sarà stata eseguita dopo la scelta di N_B. Inoltre, B potrà controllare quanto tempo sarà trascorso tra il passo 1. e il passo 4. e potrà così determinare se il messaggio ricevuto sia "fresco" o no.

Se il messaggio che A vuole mandare a B dopo l'autenticazione non contiene informazioni sensibili e può essere mandato in chiaro, il protocollo 1. può essere modificato come segue:

Protocollo 2. Schema di *challenge-response* modificato

Premesse: A e B condividono una chiave K_{AB} relativa ad un cifrario simmetrico.
B sceglie un *nonce* N_B e M è il messaggio che A vuole mandare a B.
Indichiamo con MAC il *message authentication code* ovvero una funzione che prende come input una stringa (per esempio un messaggio) e una chiave; poi trasforma la stringa in una stringa più corta detta *tag*.

Obiettivo: A riesce ad autenticarsi e a comunicare con B.

1. B $\rightarrow$ A : N_B
2. A $\rightarrow$ B : $M, c = MAC\,(K_{AB}, M, N_B)$
3. B calcola a sua volta $MAC\,(K_{AB}, M, N_B)$ e verifica se è uguale a c: se i valori coincidono, allora B autentica A ed accetta il messaggio M ricevuto, altrimenti non riconosce A e ne rifiuta la comunicazione.

Con i protocolli visti finora otteniamo l'autenticazione solo di una delle due parti; per avere autenticazione reciproca esistono algoritmi migliori rispetto alla ripetizione degli algoritmi visti per autenticare anche l'altra parte, di cui non andiamo a vedere il funzionamento perché ininfluenti nella nostra trattazione.

1.3 Problemi computazionalmente difficili

Uno dei punti chiave dei protocolli crittografici in informatica è il concetto di "difficoltà computazionale". Infatti tutta la teoria della crittografia moderna viene realizzata imponendo limiti realistici alle risorse degli avversari, per esempio supponendo che gli avversari siano in grado di eseguire solo algoritmi che hanno un tempo di esecuzione polinomiale.

Vediamo alcune definizioni utili per comprendere cosa significhi che un algoritmo è polinomiale ed efficiente.

Definizione 1.3.1. Un **algoritmo** è una macchina di Turing deterministica che prende in input e restituisce in output delle stringhe di varia lunghezza nell'alfabeto $\Sigma = \{0, 1\}$.

Definizione 1.3.2. Diciamo che un algoritmo $\mathcal{A}$ ha un **tempo d'esecuzione** $T(n)$ se $\forall x \in \{0,1\}^*$ si ha che $\mathcal{A}(x)$ termina dopo $T(|x|)$ passi. Diciamo che l'algoritmo ha **tempo d'esecuzione polinomiale** se esiste $c \in \mathbb{N}$ tale che, se $|x| = n$, allora $T(n) = n^c$.

Definizione 1.3.3. Definiamo come **algoritmo efficiente** un algoritmo che ha un tempo d'esecuzione polinomiale.

In questa sede, è opportuno ricordare che, prima di vedere nello specifico quali siano i problemi irrisolvibili in tempo polinomiale che useremo durante il corso della trattazione, introduciamo altre nozioni che ci aiuteranno a comprendere meglio tali problemi.

Definizione 1.3.4. Diciamo che $\epsilon(n)$ è una **funzione trascurabile** se $\forall c > 0\ \exists\ n_0$ tale che $\forall n > n_0$ si ha che $\epsilon(n) \leq \frac{1}{n^c}$.

1.3.1 Problema della fattorizzazione

Sia $\Pi_n = \{q \text{ tali che } q \text{ primo e } q < 2^n\}$ l'insieme dei numeri primi di n bit, chiamiamo **ipotesi di fattorizzazione** la seguente affermazione: per ogni $\mathcal{A}$ avversario esiste una funzione trascurabile ϵ tale che, presi casualmente due numeri primi, $p, q \in \Pi_n$, e posto $N = pq$, la probabilità che l'avversario preso in input N restituisca uno dei due numeri primi che lo compongono, è inferiore a $\epsilon(n)$. Formalmente diremo che:

$$Pr\,[\, p \leftarrow \Pi_n,\ \ q \leftarrow \Pi_n,\ \ N = pq\ :\ \mathcal{A}(N) \in \{p, q\}\,]\ \leq\ \epsilon(n)$$

Esistono alcuni algoritmi per fattorizzare un numero composto da due primi, ma nessuno di quelli trovati finora ha un tempo d'esecuzione polinomiale, in particolare per valori di p e q sufficientemente grandi.

Notiamo che, grazie all'ipotesi della fattorizzazione, individuiamo la seguente famiglia di funzioni *one-way*:

Teorema/Definizione 1.3.1. Sia $\mathcal{F} = \{\ f_i : D_i \to R_i\ \}_{i \in I}$ con $I = 2\mathbb{N}$, $D_i = \{\ (p, q)$ tale che $p, q \in \Pi_{i/2}$, $R_i = \mathbb{N}$ e $f_i(p, q) = p \cdot q$. Se vale l'ipotesi della fattorizzazione allora $\mathcal{F}$ è una famiglia di funzioni *one-way*.

1.3.2 Problema del logaritmo discreto

Definizione 1.3.5. Sia p un numero primo e siano $g, y \in \mathbb{Z}_p^*$. Se $g^x = y \pmod p$ allora diciamo che x è il **logaritmo discreto** di y.

Definizione 1.3.6. Sia G un gruppo moltiplicativo, diciamo che $g \in G$ è un **generatore del gruppo** G se $\{g^0, g^1, g^2, ...\} = G$. Inoltre indichiamo con Gen_G l'insieme dei generatori del gruppo G.

Notiamo che in alcuni casi il calcolo del logaritmo discreto è facile, tuttavia diventa un problema complesso se g è un generatore del gruppo $\mathbb{Z}_p^*$.

Allora chiamiamo **ipotesi del logaritmo discreto** la seguente affermazione: sia G_q un gruppo ciclico finito di ordine q, con $q \in \Pi_n$, allora per ogni avversario $\mathcal{A}$ esiste una funzione trascurabile ϵ tale che, preso $g \in Gen_{G_q}$ e preso $x \in \mathbb{Z}_q$, allora la probabilità che $\mathcal{A}(g^x) = x$ è inferiore a $\epsilon(n)$. Formalizzando, avremo dunque che:

$$Pr\,[\, q \leftarrow \Pi_n,\ \ g \leftarrow Gen_{G_q},\ \ x \leftarrow \mathbb{Z}_q :\ \mathcal{A}(g^x) = x\,]\ \leq\ \epsilon(n)$$

Notiamo che è importante che q sia primo, perciò solitamente si usano i numeri di Sophie-Germain, ovvero numeri primi p della forma $p = 2q + 1$ con p, q primi, così da poter prendere G_q come il sottogruppo di ordine q dei quadrati del gruppo $\mathbb{Z}_p^*$. In alternativa a questo metodo, per trovare un gruppo ciclico finito di ordine q, possiamo

considerare anche i punti delle curve ellittiche [6].

Analogamente al caso del problema della fattorizzazione, grazie all'ipotesi del logaritmo discreto, è possibile definire una famiglia di funzioni *one-way*:

Teorema/Definizione 1.3.2. Sia $\mathbf{DL} = \{ f_i : D_i \to R_i \}_{i \in I}$ con $I = \{ (q, g) : q \in \Pi_k$ e $g \in GenG_q \}$, $D_i = \{ x : x \in \mathbb{Z}_q \}$, $R_i = G_q$ e $f_i(x) = f_{q,g}(x) = g^x$. Se vale l'ipotesi del logaritmo discreto allora **DL** è una famiglia di funzioni *one-way*.

1.4 Schema di Diffie-Hellman

Lo **schema di Diffie-Hellman** è un protocollo ideato per accordarsi su una chiave segreta comune, attraverso un canale di comunicazione non sicuro. L'aspetto particolare di questo schema è che i due partecipanti riescono a concordare una chiave senza che questa venga mandata nel canale di comunicazione.

Vediamo un'implementazione generale del metodo con i gruppi ciclici:

Protocollo 3. Schema di Diffie-Hellman con gruppi ciclici

Premesse: A e B hanno precedentemente concordato un elemento g che genera un gruppo ciclico moltiplicativo G tale che $q = |G|$, ovvero l'ordine di G sia q, con q numero primo.

Obiettivo: A e B riescono a concordare una chiave segreta senza trasmetterla nel canale insicuro.

1. A sceglie un valore casuale r_A tale che $1 \leq r_A \leq q$ e calcola $t_A = g^{r_A}$
2. B sceglie un valore casuale r_B tale che $1 \leq r_B \leq q$ e calcola $t_B = g^{r_B}$
3. A $\to$ B : t_A
4. B trova la chiave segreta comune come $K_{AB} = t_A^{r_B} = g^{r_A r_B}$
5. B $\to$ A : t_B
6. A trova la chiave segreta comune come $K_{AB} = t_B^{r_A} = g^{r_A r_B}$

---

[6] Nella trattazione non utilizzeremo le curve ellittiche, perciò non approfondiremo l'argomento. Tuttavia nel caso si fosse interessati ad approfondire e capire meglio tale tema è possibile consultare [5, 6]

Originariamente lo schema di Diffie-Hellman utilizzava come G il gruppo $\mathbb{Z}_p^*$, ovvero il gruppo degli interi non nulli modulo p, con p primo sufficientemente grande. In seguito si è preferito prendere G come sottogruppo di $\mathbb{Z}_p^*$ di ordine q, con q primo sufficientemente grande.

Notiamo che un avversario anche intercettando i messaggi del protocollo non riuscirebbe ad ottenere il valore della chiave K_{AB} perché non è possibile ottenere il valore di K_{AB} dai valori passati sul canale t_A e t_B. Questo perché dovrebbe ottenere almeno uno dei valori tra r_A e r_B. Quest'operazione tuttavia corrisponde al calcolo del logaritmo discreto che, come abbiamo in precedenza visto, è computazionalmente difficile.

Lo schema di Diffie-Hellman può essere realizzato con operazioni e gruppi diversi da quelli visti nel protocollo 3. Infatti, questo schema è la base di quello di cifratura a chiave pubblica e può essere realizzato sfruttando il problema del logaritmo discreto, come abbiamo visto nel protocollo 3., ma può essere anche realizzato sfruttando il problema della fattorizzazione [7] o del calcolo della radice quadrata modulo il prodotto di una coppia di primi.

Notiamo che il protocollo di Diffie-Hellman da solo non ci permette di ottenere l'autenticazione di nessuna delle due parti, perciò di per sé è vulnerabile ad attacchi di tipo *man-in-the-middle*. Tuttavia, vedremo in seguito che potrà essere combinato ad altre tecniche per realizzare protocolli di autenticazione efficienti.

[7] Il protocollo basato sul problema della fattorizzazione prende il nome di *protocollo RSA* e usa l'operazione di esponenziale modulo il prodotto di due primi. Il nome RSA è dato dalle tre iniziali di Rivest, Shamir e Adleman che hanno ideato questo protocollo.

Capitolo 2

Primi protocolli con uso di Password

2.1 Introduzione

In crittografia quando si parla di autenticazione è comune richiedere che le parti che desiderano autenticarsi condividano una chiave segreta. Solitamente questa consiste in una stringa molto lunga di bit che viene facilmente gestita dalle macchine.[1]
Tuttavia quando ci troviamo nel caso di un utente (umano) che desidera accedere ad un sistema, ad un'applicazione o ad un servizio (scenario che prende il nome di **user-host authentication**), passiamo all'utilizzo di protocolli di autenticazione basati su password. Col termine password indichiamo una chiave simmetrica sufficientemente corta (non di molto superiore agli 8 caratteri) tale da poter essere facilmente memorizzata dall'utente.

Andremo quindi ad analizzare alcuni protocolli di autenticazione tra utente e host per individuare una possibile maniera di stabilire una comunicazione sicura nonostante i limiti dovuti all'uso di una stringa segreta corta.[2]

2.2 Protocolli "ingenui"

Consideriamo un utente U che desidera accedere ad un sistema attraverso l'host H. In un primo momento H dovrà inizializzare la comunicazione con U salvando in un apposito archivio la coppia (ID_U, P_U), dove ID_U indica l'identità di U (ovvero i dati necessari per identificare U) e P_U la password associata ad U.

In questo caso un primissimo protocollo che permetta ad U di autenticarsi con H grazie una password potrebbe essere il seguente:

[1] Le stringhe che vengono gestite dalle macchine solitamente sono lunghe centinaia, migliaia di bit, in base all'algoritmo utilizzato e al livello di sicurezza richiesto

[2] Alcune delle informazioni contenute in questo lavoro sono tratte dalle opere [1, 2]

Protocollo 4. Approccio base

1. U $\rightarrow$ H : ID_U;
2. H $\rightarrow$ U : "Input Password:";
3. U $\rightarrow$ H : P_U;
4. H cerca la coppia (ID_U, P_U) nell'archivio, se la trova fa accedere U al sistema, altrimenti rifiuta l'accesso.

Questo protocollo era in uso nei primi anni '70, quando, per accedere ad un host, l'utente interagiva con un terminale, il cui collegamento all'host consisteva in un'apposita linea inattaccabile. In quello scenario il protocollo garantiva autenticazione unilaterale di U con H.

Tuttavia attualmente questo protocollo, in uno scenario ben diverso di comunicazioni su una rete aperta e con accesso remoto, non permette di ottenere nessun tipo di autenticazione, né unilaterale né reciproca. Infatti non viene usato alcun tipo di dato che permetta di controllare l'attualità ("*freshness*"[3]) dei messaggi; perciò per H non è possibile effettivamente sapere se la comunicazione con U sia attiva o invece sia in corso un attacco di tipo replay.

Osserviamo che i problemi del protocollo 4. sono essenzialmente i seguenti:

- la vulnerabilità del file di password salvato sull'host. Questo file potrebbe venire letto da un avversario E che potrebbe così ottenere tutti i diritti di accesso di tutti gli utenti. Potrebbe poi utilizzare i dati acquisiti per accedere ad H, immedesimandosi in uno di loro e pertanto recando danni all'utente da lui impersonato o all'intero sistema, senza essere rintracciabile.

- la password viaggia in chiaro sul canale tra U e H. In questo modo la password potrebbe essere intercettata da qualcuno che tiene sotto controllo il canale.

2.2.1 Protocollo di Needham

Per risolvere il problema del protocollo 4. l'informatico inglese Needham propose un metodo semplice ed efficiente che permetteva di salvare in maniera sicura le password in un file sull'host. L'idea di Needham prevedeva che l'host H usasse una funzione one-way

[3]Termine utilizzato in [2]

f per cifrare le password e salvasse sul file la coppia $(ID_U, f(P_U))$ invece di (ID_U, P_U).
In questo modo il protocollo 4. verrebbe modificato in questi termini:

Protocollo 5. Protocollo di autenticazione di Needham

Premesse: L'host H ha salvato la password dell'utente U come $(ID_U, f(P_U))$ dove f è la funzione one-way.
L'utente U memorizza invece la propria password P_U.

Obiettivo: L'utente U riesce ad accedere all'host H autenticandosi con la propria password.

1. U $\rightarrow$ H : ID_U;
2. H $\rightarrow$ U : "Input Password:";
3. U $\rightarrow$ H : P_U;
4. H calcola $f(P_U)$ e cerca la coppia $(ID_U, f(P_U))$ nell'archivio: se la trova garantisce l'accesso ad U, altrimenti nega l'accesso al sistema.

Il protocollo 5. mostra come funziona lo schema di autenticazione tramite password nei sistemi operativi UNIX [4]. Vediamo quindi alcuni dettagli della realizzazione su UNIX del protocollo:

- la funzione f è realizzata tramite l'algoritmo di cifratura DES;[5]
- l'host H salva su un file di password l'identità dell'utente U (che viene indicata con UID) e un messaggio cifrato con DES. Più precisamente l'algoritmo simmetrico viene applicato ad una stringa di 64 zeri e la password P_U viene utilizzata come chiave;

[4] Unix è un sistema operativo per computer sviluppato dai Bell Laboratories alla fine degli anni '60 e utilizzato negli anni '70. Attualmente è alla base di altri sistemi operativi come Linux e MacOS [7, 8]

[5] DES è un algoritmo di cifratura simmetrico con chiave da 64 bit. Venne introdotto come standard nel 1976, ma attualmente viene considerato un algoritmo insicuro per molte applicazioni, principalmente a causa della chiave troppo corta

– la funzione $f(P_U)$ non consiste nell'utilizzo dell'algoritmo puro di DES, ma nella ripetizione dell'algoritmo per 25 volte con l'aggiunta di un metodo detto *"bit-swapping permutation"* che introduce l'uso del **salt**.[6]

Con questi accorgimenti la funzione $f(P_U)$ che sfrutta DES può esser vista come una funzione *hash one-way* cifrata e parametrizzata applicata alla stringa costante 0^{64}, dove la chiave è la password P_U e il parametro è il *salt*.

Grazie all'uso del *salt*, nel file delle password dell'host H, le informazioni per l'accesso degli utenti U sono salvate come (ID_U, *salt*, $f(P_U, salt)$)[7]. Allora il protocollo 5. realizzato per UNIX mette in sicurezza il file di password: infatti se ora un avversario ottiene dal file sull'host la funzione della password, $f(P_U)$, non potrà usarla per accedervi, perché quando verrà proposto ad H $f(P_U)$ nel messaggio 3. del protocollo, al passo successivo, H calcolerà $f(f(P_U))$ e finirà per negare l'accesso all'avversario che si maschera come U poiché non incontrerà la tripla (ID_U, *salt*, $f(f(P_U))$) nel file. Inoltre, la funzione f non è facilmente invertibile, specialmente perché include 25 ripetizioni del *bit-swapping permutation*. E' quindi sufficiente che un utente U scelga una password non facilmente indovinabile affinché diventi molto difficile per un avversario ottenere P_U da $f(P_U)$.

Grazie al protocollo 5. allora, si riesce ad ottenere confidenzialità[8] per il file di password, tuttavia manca la garanzia che il file rimanga integro, cioè che non venga modificato. Resta la possibilità di un attacco online in cui un avversario intercetta la password che viene passata in chiaro sul canale insicuro nel messaggio 3. del protocollo. Per risolvere quest'ultimo problema, allora vediamo un altro tipo di protocollo, detto schema **one-time password**.

2.2.2 Schema one-time password e S/KEY

Il concetto alla base di uno schema *one-time password* è che ogni volta che un utente U voglia autenticarsi con l'host H per accedervi, utilizza una password diversa da quella precedente; inoltre nessuna password usata da U può essere poi riutilizzata. In questo modo, le password, pur essendo legate fra loro computazionalmente, non sono utili in nessun modo in un'altra esecuzione del protocollo per un avversario che abbia intercettato sul canale tra U e H una delle password usate.

Prima di vedere il funzionamento dello schema *one-time password* introduciamo la seguente notazione:

[6] il *bit-swapping permutation* ad ogni ripetizione dell'algoritmo scambia alcuni blocchi di bit del messaggio cifrato, ottenuto dalla chiamata precedente, secondo una stringa di 12 bit casuali detta **salt**. Questa stringa viene salvata anch'essa nel file di password.

[7] Per alleggerire la scrittura useremo la notazione $f(P_U)$ al posto di $f(P_U, salt)$

[8] Definiamo **confidenzialità** uno degli obiettivi dei protocolli crittografici. Questo consiste nel garantire che le informazioni siano disponibili solamente a chi è autorizzato ad ottenerle. Questo aspetto è di solito costruito attraverso tecniche di cifratura.

Notazione 2.2.1. *Sia f una funzione one-way e n un numero intero sufficientemente grande, allora pongo:*

$$f^n(P_U) \stackrel{def}{=} \underbrace{f(\cdots(f}_{n}(P_U))\cdots)$$

Vediamo allora come interagiscono in uno schema *one-time password* un utente e l'host:

Protocollo 6. Schema di autenticazione con *one-time password*

Premesse: L'host H ha salvato la password dell'utente U come (ID_U, $f^n(P_U)$) dove f è la funzione one-way e n è un numero intero sufficientemente grande.
L'utente U memorizza invece la propria password P_U.

Obiettivo: L'utente U riesce ad accedere all'host H usando ogni volta una password diversa, ma inizializzandola un'unica volta.

1. U $\rightarrow$ H : ID_U;
2. H $\rightarrow$ U : "Input Password:";
3. U utilizza un altro dispositivo per calcolare $f^{n-1}(P_U)$, applicando $n-1$ volte la funzione f alla password P_U;
4. U $\rightarrow$ H : $f^{n-1}(P_U)$;
5. H applica f al messaggio ricevuto da U, quindi calcola $f(f^{n-1}(P_U)) = f^n(P_U)$, e controlla la presenza di (ID_U, $f^n(P_U)$) all'interno del file di password: se H trova la coppia nel file, permette a U di accedere aggiornando il file e sostituendo la coppia trovata con (ID_U, $f^{n-1}(P_U)$) per la prossima richiesta d'accesso.

Alla prossima chiamata del protocollo nei punti 3. e 4. U calcolerà e manderà ad H $f^{n-2}(P_U)$; invece nel punto 5. H calcolerà $f(f^{n-2}(P_U))$, che confronterà con $f^{n-1}(P_U)$ e, in caso di autenticazione avvenuta con successo, H sostituirà nel file $f^{n-1}(P_U)$ con $f^{n-2}(P_U)$. Analogamente per le successive chiamate al protocollo.

$\vdots$

Durante il protocollo 6., nel punto 5. della prima chiamata (e nelle successive chiamate con le rispettive potenze di f) se il messaggio mandato da U nel punto 4. supera il test, allora H assume di aver ricevuto tale valore da U, in quanto soltanto U può avere calcolato il valore inviato. Infatti solo l'utente U conosce P_U: l'host H, dopo l'inizializzazione, non ne conserva memoria, perché salva solo $f^n(P_U)$. Inoltre, quando il protocollo viene eseguito più volte e le potenze di f che vengono usate sono la 0-esima e la 1-esima[9], allora U e H devono inizializzare una nuova password P'_U.

Uno dei problemi di questo schema deriva dal fatto che, per funzionare, è necessario che U e H siano sincronizzati rispetto la password da utilizzare; ovvero quando H ha salvato nel file la coppia $(ID_U, f^i(P_U))$, è necessario che U mandi la password $f^{i-1}(P_U)$. Infatti, la sincronizzazione potrebbe andar persa se, per esempio, si verificasse un *crash* del sistema o il canale di comunicazione venisse ritenuto non affidabile per il passaggio di una password. Dato che questi scenari potrebbero essere simulati da un avversario, possono allora costituire la base per un attacco che impedirebbe ad U di autenticarsi ad H.
Per risolvere il problema, si può pensare ad un metodo per ristabilire la sincronizzazione, in caso essa venga persa. Tale metodo necessita però di un'autenticazione reciproca tra H e U. Questa potrebbe essere implementata nella seguente maniera:

> se H è impostato con $f^j(P_U)$ e U invece con $f^k(P_U)$ dove $j \neq k+1$ (ovvero U e H sono desincronizzati)
>
> $\Rightarrow$ il sistema dovrà andare avanti e impostarsi in una situazione dove H ha salvato $f^i(P_U)$ nel file e U è pronto a mandare ad H $f^{i-1}(P_U)$, dove $i \leq min(j,k)$

Il protocollo 6. è alla base di un sistema di *one-time password* noto come **S/KEY**. In questo sistema si cerca di superare il problema di desincronizzazione introducendo un contatore.

Vediamo come funziona il protocollo S/KEY, analizzandone poi i problemi:

[9]Si tratta del caso in cui nel messaggio 3. U "calcola" $f^0(P_U)$ e quindi non esegue alcuna operazione mandando ad H P_U; mentre, nel file, H ha la coppia $(ID_U, f^1(P_U) = f(P_U))$ salvata con cui confrontare il valore mandato da U, dopo l'applicazione della funzione f.

Protocollo 7. S/KEY

Premesse: L'host H ha salvato la password dell'utente U come (ID_U, $f^n(P_U)$, n), dove f è una funzione *hash one-way* e n è un numero intero sufficientemente grande.
L'utente U memorizza invece la propria password P_U.
Al momento di inizio del protocollo, U suppone che in H la sua password sia impostata su (ID_U, $f^c(P_U)$, c) per un qualche c tale che $1 \leq c \leq n$.

Obiettivo: L'utente U riesce ad accedere all'host H usando ogni volta una password diversa, ma inizializzandola un'unica volta.

1. U $\rightarrow$ H : ID_U;
2. H $\rightarrow$ U : c, "Input Password:";
3. U utilizza un altro dispositivo per calcolare $Q = f^{c-1}(P_U)$, applicando $c-1$ volte la funzione f alla password P_U;
4. U $\rightarrow$ H : Q;
5. H trova nell'archivio (ID_U, $f^c(P_U)$, c), se $f(Q) = f^c(P_U)$ allora permette ad U di accedere; in seguito H aggiorna la tripla con (ID_U, $Q = f^{c-1}(P_U)$, $c-1$).

Grazie alle modifiche apportate al protocollo 6., nel protocollo 7. U e H non si desincronizzano. Tuttavia viene apportata una modifica pericolosa, perché viene introdotto l'uso di un contatore che viene mandato in chiaro. Allora questo può essere modificato da un avversario oppure lo stesso avversario potrebbe mascherarsi da host H e riuscire ad ottenere da U la sua password, così da permettergli di autenticarsi con H.

Capitolo 3

EKE: Encrypted Key Exchange

3.1 Introduzione

In questo capitolo, introduciamo una nuova classe di protocolli basati sull'uso di password. Questa nuova tipologia, ideata da Bellovin e Merritt negli anni '90, prende il nome di **Encrypted Key Exchange** (abbreviato con la sigla **EKE**).

Supponiamo di essere sempre nel caso in cui un utente U vuole accedere ad un sistema e deve quindi autenticarsi con un host H. L'idea alla base dei protocolli EKE è la seguente:

- H e U condividono una password P_U, che verrà usata come chiave simmetrica (di lunghezza relativamente breve);
- U, ovvero colui che inizializza il protocollo, sceglie una chiave pubblica temporanea e usa la password P_U come chiave simmetrica per cifrare la propria chiave pubblica;
- H riceve la chiave cifrata, può decifrarla e poi usarla per mandare ad U una chiave di sessione cifrata sia con la chiave pubblica temporanea che con la password P_U.

In questo modo non saranno possibili attacchi di tipo *eavesdrop*, perché un avversario non è in grado di riconoscere quale sia la chiave pubblica che è stata inviata o cosa sia in realtà solo rumore (poiché le chiavi sono semplicemente stringhe di bit, spesso casuali). Anche in caso di successo di un attacco di forza bruta per individuare la password, non è possibile usare la chiave trovata per decifrare il messaggio mandato da H e ottenere la chiave di sessione, perché non si può determinare la chiave privata, necessaria per questa operazione, da quella pubblica.

3.2 Versione di Bellovin e Merritt

Esistono più varianti del protocollo EKE di Bellovin e Merritt e alcuni di questi specificano quale algoritmo di cifratura usare e come trasformare la password da usare

come chiave. Il protocollo originale invece non specifica gli algoritmi da usare. Vedremo prima in generale il funzionamento del protocollo, poi lo vedremo realizzato con lo schema di scambio di chiavi di Diffie-Hellman.

3.2.1 Schema generale

Protocollo 8. Protocollo EKE basico

Premesse: L'utente U e l'host H condividono la password P_U e si sono accordati su quale algoritmo di cifratura simmetrico e quale algoritmo di cifratura asimmetrico utilizzare;
$K()$ indica l'algoritmo di cifratura simmetrico con chiave K;
$\mathcal{E}_U()$ indica la chiave pubblica di U, che viene usata come chiave di cifratura nell'algoritmo asimmetrico.

Obiettivo: L'utente U e l'host H riescono ad autenticarsi reciprocamente e concordare una chiave di sessione segreta.

1. U genera una chiave pubblica casuale $\mathcal{E}_U$;
2. U $\rightarrow$ H : U, $P_U(\mathcal{E}_U)$;
3. H decifra il messaggio ricevuto, usando P_U, e ottenendo $\mathcal{E}_U$, inoltre genera una chiave simmetrica K casuale;
4. H $\rightarrow$ U : $P_U(\mathcal{E}_U(K))$;
5. U decifra due volte il messaggio ricevuto (prima usa la password, poi la sua chiave privata) e genera un nonce N_U;
6. U $\rightarrow$ H : $K(N_U)$;
7. H decifra il messaggio ricevuto ottenendo il valore di N_U e genera un nonce N_H;
8. H $\rightarrow$ U : $K(N_U, N_H)$;
9. U decifra il messaggio ricevuto da H e verifica N_U;
10. U $\rightarrow$ H : $K(N_H)$;

11. se dal punto 5. al punto 10. le verifiche di *challenge-response* hanno successo, U accede ad H e nelle comunicazioni successive tra le due parti verrà usata la chiave K.

L'innovazione del protocollo 8. risiede nei primi passaggi. Infatti, nei passi da 1. a 4. viene usata la password P_U condivisa tra U e H per cifrare la chiave pubblica $\mathcal{E}_U$ e la chiave di sessione K già cifrata con la chiave pubblica. In questo modo, la password, costituita da una stringa di bit corta, è statisticamente indipendente dai due messaggi cifrati: ovvero, intercettando i due messaggi in 2. e 4. non è possibile ottenere nuove informazioni sulla password, perché la chiave pubblica $\mathcal{E}_U$ e la chiave di sessione K sono due stringhe più lunghe e casuali e, alla successiva chiamata del protocollo, vengono cambiate. Quest'ultima caratteristica è molto importante perché diversamente il protocollo, senza questa, fallirebbe: infatti, se $\mathcal{E}_U$ e K non venissero cambiate ogni volta che viene eseguito il protocollo, non si potrebbe escludere la possibilità di attacchi di tipo dizionario offline. Rimangono ancora da considerare gli attacchi attivi, in cui vengono modificati i messaggi passati durante il protocollo. Questo tipo di attacco, grazie all'autenticazione reciproca tra U e H, verrebbe individuato e porterebbe alla conclusione del protocollo.

Inoltre è importante notare che la cifratura della chiave pubblica ci garantisce l'autenticità della chiave e la sua provenienza. Infatti, normalmente questa proprietà verrebbe garantita grazie all'uso dei certificati per le chiavi pubbliche[1]. Tuttavia, poiché $\mathcal{E}_U$ deve essere cambiata di volta in volta, non è possibile affidarsi all'uso dei certificati, perché ne servirebbe uno nuovo per ogni esecuzione del protocollo.

Analogamente a quanto detto per P_U, i nonce N_U e N_H, che sono stringhe abbastanza lunghe di bit e sono generate casualmente, vengono cifrati e inviati nei passi 6. e 8. nascondendo ulteriormente la chiave di sessione K. Inoltre il nonce N_U è cruciale per l'autenticazione di H nei confronti di U: grazie allo scambio del nonce cifrato con la chiave di sessione K, U riesce ad autenticare H nel passo 9., infatti, verificando che N_U sia appunto il nonce da lui inviato, U controlla di non star comunicando con un avversario che si maschera da H.

Osserviamo infine che la chiave pubblica $\mathcal{E}_U$ non è in realtà pubblica, perché nel canale tra U e H viene passata cifrata con P_U e diventa semplicemente la chiave utilizzata in un algoritmo asimmetrico per cifrare messaggi.

[1] In crittografia un **certificato (digitale)** è un documento elettronico che garantisce l'associazione univoca tra una chiave pubblica e l'identità del soggetto che ne rivendica l'utilizzo. [13]

3.2.2 Diffie-Hellman applicato al protocollo EKE

Protocollo 9. Protocollo EKE realizzato con lo schema Diffie-Hellman

Premesse: L'utente U e l'host H condividono la password P_U e si sono accordati su quale algoritmo di cifratura simmetrico e quale funzione usare per ottenere la chiave K, ricavata dalla chiave segreta ottenuta con Diffie-Hellman;
Sia p un numero primo sufficientemente grande, U e H allora si accordano sul gruppo $G = \mathbb{Z}_p^* = \mathbb{Z}_p \setminus [0]$ generato dall'elemento g e tale che $|G| > 2^{64} > 2^{|P_U|}$;
$\{\ \}_K$ indica l'algoritmo di cifratura simmetrico con la chiave K, invece $\mathcal{E}_U$ e $\mathcal{E}_H$ rappresentano la chiavi pubbliche rispettivamente di U e di H utilizzate per determinare la chiave di sessione K grazie allo schema Diffie-Hellman.

Obiettivo: L'utente U e l'host H riescono ad autenticarsi reciprocamente e concordare una chiave di sessione segreta.

1. U sceglie casualmente un elemento di G, x, che diventa la propria chiave privata e calcola la chiave pubblica $\mathcal{E}_U = g^x$;
2. U $\rightarrow$ H : U, $\{\mathcal{E}_U\}_{P_U}$;
3. H sceglie casualmente un elemento di G, y, che diventa la propria chiave privata e calcola la rispettiva chiave pubblica $\mathcal{E}_H$;
4. H calcola la chiave di sessione K dalla chiave segreta $(\mathcal{E}_U)^y = g^{xy}$ e sceglie casualmente un nonce $N_H \in]0, 2^{64}[$;
5. H $\rightarrow$ U : $\{\mathcal{E}_H\}_{P_U}$, $\{N_H\}_K$;
6. U calcola la chiave di sessione K dalla chiave segreta $(\mathcal{E}_H)^x = g^{yx} = g^{xy}$ e sceglie casualmente un nonce $N_U \in]0, 2^{64}[$;
7. U $\rightarrow$ H : $\{N_U, N_H\}_K$;
8. H verifica il proprio nonce N_H;
9. H $\rightarrow$ U : $\{N_U\}_K$;

10. U verifica il proprio nonce N_U e, se tutte le verifiche sono corrette, U accede ad H e nelle successive comunicazioni useranno la chiave K.

Osserviamo che, nell'implementazione del protocollo EKE realizzato con lo scambio di chiavi Diffie-Hellman (protocollo 9.), U e H ottengono entrambi la chiave g^{xy}, ma la funzione utilizzata per ottenere K da questa chiave non è specificata. Non viene specificato neanche l'algoritmo di cifratura simmetrico da utilizzare con la chiave P_U. Originariamente infatti, Bellovin e Merritt ritennero accettabile qualsiasi scelta per l'algoritmo, anche le più deboli. Tuttavia studi successivi mostrarono che l'uso di funzioni di cifratura che non avessero particolari proprietà evitavano che alcune stringhe, introdotte allo scopo di fornire prove di sicurezza al destinatario, venissero ottenute da un avversario, lasciando però in alcuni casi la possibilità di attacchi.
Bellovin e Merritt individuarono infatti un possibile attacco al protocollo 9., che veniva definito come segue:

Definizione 3.2.1 (Attacchi di partizione)**.** Un avversario che tenta di indovinare la password P_U cercherà di decifrare $\{\mathcal{E}_U\}_{P_U}$ e $\{\mathcal{E}_H\}_{P_U}$ e valutare se il risultato è un valore temporaneamente accettabile per lo schema Diffie-Hellman. Se l'ipotesi per P_U non è corretta, il valore testato viene scartato. Allora, grazie a più chiamate del protocollo e a più tentativi di indovinare la password, si otterrebbero le "partizioni", gli insiemi di password valide e no.

Il successo di questo tipo di attacco dipende da due fattori: l'algoritmo di cifratura simmetrico utilizzato nel protocollo e i parametri che definiscono il gruppo G da considerare. Per rendere gli attacchi di partizione più difficili, è stato dimostrato[2] che è opportuno assicurare che valga la relazione: $1 - (p/2^n) < 10^{-4}$ con 2^n che indica la più piccola potenza di 2 maggiore del primo p. Senza approfondire oltre questa relazione, vediamo inoltre che è preferibile scegliere un algoritmo di cifratura simmetrico abbinato al gruppo, in modo da eliminare completamente questo tipo di attacco.

Inizialmente Bellovin e Merritt suggerirono che una delle due cifrature con chiave P_U nei passi 2. e 5. del protocollo 9. potesse essere omessa. Venne poi dimostrato attraverso alcuni controesempi che questa modifica comportava l'introduzione di debolezze nel protocollo rendendolo vulnerabile ad attacchi di tipo dizionario. Ne seguì che entrambe le cifrature erano importanti e non avrebbero dovuto essere omesse.

[2] A condurre l'analisi per trovare la relazione sopraindicata è Barry Jaspan in *Dual-workfactor encrypted key exchange: Efficiently preventing password chaining and dictionary attacks* nel *6th USENIX Security Symposium* a San Jose in California nel Luglio 1996, citato da [1], *op.cit.* pagina 251.

Vediamo gli esempi di possibili attacchi nel caso in cui uno dei due messaggi non venga cifrato:

- Se non viene cifrata la chiave pubblica di U, il messaggio 2. diventa "U, $\mathcal{E}_U = g^x$". Un avversario E può sfruttare questa modifica scegliendo un elemento di G, z, da usare come chiave privata e impostare come chiave pubblica g^z. Poi impersonando U, può mandare ad H in 2. il messaggio "U, $\mathcal{E}_U = g^z$". Ora E può sfruttare le ridondanze nel messaggio $\{N_H\}_K$ in 5. per testare una password $\widetilde{P}_U$. Questo passaggio viene realizzato nel seguente modo: E fa un'ipotesi sulla password, $\widetilde{P}_U$, e la usa per decifrare $\{\mathcal{E}_H\}_{P_U}$, ottenendo $\widetilde{\mathcal{E}}_H$, in seguito calcola la rispettiva chiave $\widetilde{K}$ dalla chiave segreta $(\widetilde{\mathcal{E}}_H)^z$ e controlla le ridondanze del risultato dopo aver decifrato $\{N_H\}_K$ con $\widetilde{K}$.

- Se non viene cifrata la chiave pubblica di H, il messaggio 5. diventa "$\mathcal{E}_H$, $\{N_H\}_K$". Un avversario E può sfruttare la modifica e mascherarsi da H con U. Infatti se $\{N_H\}_K$ non contiene ridondanze[3], E può scegliere casualmente X e z, impostare $\mathcal{E}_H = g^z$ e mandare ad U al passo 5. il messaggio "$\mathcal{E}_H$, X". U accetterà il messaggio e successivamente E potrà tentare una password $\widetilde{P}_U$ decifrando $\{\mathcal{E}_U\}_{P_U}$, ottenendo così $\widetilde{\mathcal{E}}_U$ e la chiave di sessione $\widetilde{K}$ da $(\widetilde{\mathcal{E}}_U)^z$. Ora E potrà usare $\widetilde{K}$ per decifrare il messaggio 7. e verificare se il secondo elemento corrisponde ad X: nel caso ci sia corrispondenza allora si ha che $\widetilde{P}_U = P_U$, altrimenti le password risultano diverse e $\widetilde{P}_U$ viene scartata.

[3]Si può anche modificare l'esempio mostrando che vale anche nel caso di ridondanze limitate.

Capitolo 4

Protocolli a conoscenza zero

4.1 Introduzione

Abbiamo visto come si possa utilizzare la password condivisa tra utente ed host per cifrare le chiavi da cui si ottiene quella di sessione, in modo che non sia possibile attaccare con efficienza il protocollo di autenticazione.
In questo capitolo, vediamo una modalità alternativa per ottenere l'autenticazione tra utente ed host, senza che alcuna informazione segreta, per esempio la password, venga rivelata in nessun momento. In questo modo, essa non è intercettabile perché ogni informazione trasmessa sul canale non è riutilizzabile.
Questo tipo di tecnica prende il nome di tecniche a **conoscenza zero** (*zero knowledge*)

4.2 Scenario fondamentale

Il protocollo fondamentale che sfrutta le tecniche a conoscenza zero è uno schema di *challenge-response* che riusciamo a rappresentare tramite il seguente esempio[1].

Supponiamo di avere una caverna con una galleria, come in Figura 4.1, e che presenti una porta chiusa grazie ad un'informazione segreta.
Supponiamo inoltre che "Peggy" voglia dimostrare a "Victor" di conoscere questa informazione segreta che le permetterebbe di aprire la porta, ma di non rivelargli l'informazione. Per convincere Victor di questa capacità, Peggy entra nella galleria e, al bivio nel punto B, decide casualmente quale direzione prendere. In seguito, Victor entra nella caverna e si ferma davanti al bivio nel punto B. Orienta Peggy dicendole "Sinistra" o "Destra". A questo punto, Peggy ritorna al punto B provenendo dal lato indicato da Victor.

[1] L'esempio viene illustrato da Quisquater, Guillou e Berson in *How to Explain Zero-Knowledge Protocols to Your Children* durante una conferenza sui progressi della crittografia, tenutasi nel 1989 a Santa Barbara, CA (USA) press l'Università della California

Questa sequenza viene ripetuta finché Victor non è convinto che Peggy conosca l'informazione segreta per aprire la porta: infatti, ad ogni esecuzione del protocollo, Peggy sceglie casualmente la direzione, così come anche Victor sceglie in modo del tutto casuale la direzione da indicare a Peggy. Poiché la decisione di Peggy precede quella di Victor, Peggy ha solo il 50% di possibilità di andare dalla stessa parte che poi Victor le indicherà e quindi di non dover aprire la porta.
In questo modo, se il protocollo viene ripetuto 10 volte e in ognuna di queste Peggy è uscita sempre dalla parte indicata da Victor, allora la probabilità che Peggy non sappia aprire la porta ma sia riuscita a raggirare Victor è di $1/2^{10} = 1/1024 \sim 9.8 \cdot 10^{-4}$.
Se Victor registrasse con una telecamera la scena nella caverna e un avversario "Eve" cercasse di estrarre delle informazioni utili dalla registrazione non ne ricaverebbe nulla, né su come aprire la porta, né su come convincere qualcun altro che il segreto è noto, senza nemmeno avere la sicurezza che Peggy sia passata dalla porta perché lei e Victor potrebbero essersi accordati sulla sequenza di direzioni che Victor le ha dato nelle varie esecuzioni del protocollo. Ne deduciamo che Victor non possiede alcuna informazione utile che possa essere trasmessa ad altri.
Notiamo che Victor non ottiene una prova matematica del fatto che Peggy sia in grado di aprire la porta, ma ne ricava un'evidenza sperimentale tramite una serie di *challenge-response*.

Osserviamo inoltre che il termine "a conoscenza zero" non indica semplicemente il fatto che la password, o in generale l'informazione segreta, non venga rivelata in alcun passaggio del protocollo. Infatti questa proprietà viene garantita già dagli schemi di *challenge-response*, quindi il termine "a conoscenza zero" indica qualcosa di molto più forte. I protocolli a conoscenza zero infatti garantiscono che nessuna informazione utile di alcun tipo venga ricavata dall'esecuzione, se non l'identità della parte rappresentata da Peggy. Per questo è importante che l'altra parte, ovvero Victor, scelga i messaggi da inviare a Peggy in maniera casuale, potendo così generare la stessa sequenza con la stessa distribuzione di probabilità.

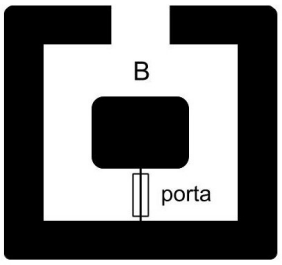

Figura 4.1: Galleria dell'esempio sul protocollo a conoscenza zero

Vediamo ora una delle possibili formalizzazioni matematiche del protocollo descritto:

Protocollo 10. Protocollo a conoscenza zero implementato con radici quadrate

Premesse: Sia $n = pq$ con p, q numeri primi sufficientemente grandi e sia y un residuo quadratico mod n tale che $MCD(y, n) = 1$.
Peggy afferma di conoscere s radice quadrata di y.

Obiettivo: Victor vuole verificare che Peggy conosca una radice quadrata di y.
Peggy vuole convincere Victor senza rivelargli il valore della radice.

1. Peggy sceglie un numero casuale r_1 tale che $MCD(r_1, n) = 1$
2. Peggy calcola $r_2 = sr_1^{-1} \pmod{n}$ così che $r_1 r_2 = s \pmod{n}$. Inoltre calcola $x_1 = r_1^2 \pmod{n}$ e $x_2 = r_2^2 \pmod{n}$
3. Peggy $\rightarrow$ Victor : x_1, x_2
4. Victor verifica che $x_1 x_2 = y \pmod{n}$
5. Victor sceglie casualmente tra x_1 e x_2 e chiede a Peggy di calcolarne la radice quadrata
6. Peggy $\rightarrow$ Victor : r_1 o r_2 (in base alla richiesta di Victor)
7. Victor verifica la risposta di Peggy

Il protocollo viene ripetuto più volte finché Victor non è convinto che Peggy conosca davvero una radice quadrata di y.

È evidente che, se Peggy conosce s, il protocollo viene eseguito facilmente. Invece se Peggy (o un avversario che si mascherasse da Peggy con Victor) cercasse di ingannare Victor e di convincerlo che conosce l'informazione segreta anche se ciò non corrisponde al vero, allora è molto improbabile che abbia successo.
Infatti Peggy potrebbe inviare a Victor due numeri casuali x_1 e x_2 tali che $x_1 x_2 = y$ ma non conoscerebbe una radice di entrambi i numeri, perché, altrimenti, nel caso le conoscesse, sarebbe anche al corrente di una radice di y. Se Peggy riuscisse a prevedere

di quale dei due numeri Victor le chiederà una radice (supponiamo che il numero previsto sia x_2) allora Peggy potrebbe prendere innanzitutto r_2 casuale e tale che $MCD(r_2, n) = 1$, poi calcolare $x_2 = r_2^2 \pmod{n}$ e $x_1 = yx_2^{-1} \pmod{n}$, infine rispondere alla richiesta da parte di Victor di una radice di x_2. Tuttavia, ad ogni esecuzione del protocollo, Peggy avrebbe solo il 50% di possibilità di indovinare quale sarà la richiesta di Victor, quindi probabilmente circa la metà delle volte che verrà eseguito il protocollo, Victor le chiederà di calcolare una radice quadrata che Peggy non conosce e che non potrà fornirgli se non conosce p e q, poiché il calcolo è complicato. D'altro canto, Victor al primo errore concluderà che Peggy non conosce nessuna radice di y.

Notiamo inoltre che, avendo Victor appurato che Peggy conosce una radice quadrata s di y, egli non avrà ottenuto alcuna informazione utile che possa essere utilizzata da un avversario, perché ciò che egli riceverà è soltanto una radice quadrata di un numero casuale e non del numero y. Tuttavia, è importante che Peggy non usi sempre gli stessi numeri durante le repliche del protocollo, perché, ottenendo le radici sia di x_1 sia di x_2, Victor (e chiunque fosse in grado di carpire entrambe le informazioni) otterrebbe anche una radice di y.

4.3 Protocollo di Feige-Fiat-Shamir

Il protocollo 10. richiede che siano scambiati numerosi messaggi tra le due parti. Vediamo allora il **Protocollo di Feige-Fiat-Shamir** che riduce il numero di messaggi necessari.

Protocollo 11. Protocollo di Feige-Fiat-Shamir

Premesse: Sia $n = pq$ con p, q numeri primi sufficientemente grandi.
Supponiamo che Peggy conosca i numeri segreti $s_1, \ldots, s_k$, tali che $MCD(s_i, n) = 1$.
Supponiamo inoltre che Peggy abbia precedentemente inviato a Victor i valori $v_i = s_i^{-2} \pmod{n}$.

Obiettivo: Victor vuole verificare che Peggy conosca i numeri segreti $s_1, \ldots, s_k$.
Peggy vuole convincere Victor, senza rivelare alcun valore segreto.

1. Peggy sceglie un numero intero casuale r e calcola $x = r^2 \pmod{n}$
2. Peggy $\rightarrow$ Victor : x
3. Victor sceglie una sequenza di k bit, $b_1, \ldots, b_k$ tali che $b_i \in \{0, 1\}$
4. Victor $\rightarrow$ Peggy : $b_1, \ldots, b_k$
5. Peggy calcola $y = r s_1^{b_1} s_2^{b_2} \cdots s_k^{b_k} \pmod{n}$
6. Peggy $\rightarrow$ Victor : y
7. Victor verifica se $x = y^2 v_1^{b_1} v_2^{b_2} \cdots v_k^{b_k} \pmod{n}$

Il protocollo viene ripetuto più volte, cambiando di volta in volta il valore del numero casuale r, finché Victor non si è convinto che Peggy conosca davvero i numeri segreti $s_1, \ldots, s_k$.

Vediamo ora come si svolge l'esecuzione del protocollo prima nel caso di $k = 1$ poi di $k > 1$ e quindi perché il protocollo è adatto per l'autenticazione senza che venga riferita alcuna informazione utile a carpirne il segreto.

Se $k = 1$ Victor nel passo 4. può chiedere a Peggy di calcolare $r \cdot 1$ o $r \cdot s_1$ se b_1 è rispettivamente 0 o 1. In questo caso r e rs_1 sono numeri casuali, il cui quoziente è una radice quadrata di v_1, perciò il caso $k = 1$ è in realtà analogo al protocollo 10. con l'operazione di quoziente in luogo del prodotto.

Se $k > 1$, supponiamo Victor mandi a Peggy nel passo 4. la sequenza di bit in cui $b_1 = b_2 = b_4 = 1$ e $b_i = 0$ per ogni altro valore di i. In questo caso Peggy deve calcolare $y = r s_1 s_2 s_4$ che è radice quadrata di $x v_1^{-1} v_2^{-1} v_4^{-1}$.
Allora in generale ad ogni esecuzione del protocollo, Victor chiede a Peggy la radice di un numero del tipo $x v_{i_1}^{-1} v_{i_2}^{-1} \cdots v_{i_j}^{-1}$ e se Peggy conosce i valori $r, s_{i_1}, s_{i_2}, \ldots, s_{i_j}$ allora riesce a rispondere facilmente alla richiesta di Victor, altrimenti il problema diventerebbe insolubile. Infatti, se Peggy non conoscesse almeno uno dei valori segreti $s_1, \ldots, s_k$, l'unica cosa che potrebbe fare sarebbe tentare di indovinare la sequenza di bit che Victor le chiederà al passo 4. e calcolare di conseguenza x. Se Peggy indovinasse la stringa $b_1, \ldots, b_k$, potrebbe prendere un valore casuale per y e calcolare $x = y^2 v_1^{b_1} v_2^{b_2} \cdots v_k^{b_k} \pmod{n}$ così che, mandando a Victor il valore di y, la congruenza modulo n venga verificata e Victor venga ingannato.
Tuttavia, se Peggy non riuscisse ad indovinare la sequenza di bit che Victor le manderà, allora dovrebbe modificare la scelta di y che non potrebbe più essere casuale ma dovrebbe essere calcolato trovando le radici quadrate dei v_i. Infatti se per esempio Peggy avesse supposto che Victor le avrebbe mandato la stringa di bit in cui $b_1 = b_2 = b_4 = 1$ e $b_i = 0$

per ogni altro valore di i, ma in realtà avesse poi ricevuto la stringa con $b_1 = b_3 = 1$ e inoltre tutti gli altri bit fossero nulli, allora in questo caso Peggy sarebbe stata pronta a mandare una radice quadrata di $xv_1^{-1}v_2^{-1}v_4^{-1}$, ma Victor sarebbe in attesa di una radice quadrata di $xv_1^{-1}v_3^{-1}$. Allora Peggy per rispondere correttamente dovrebbe calcolare una radice quadrata di $v_2^{-1}v_3v_4^{-1}$, che non è in grado di calcolare; quindi Peggy ha solo 1 possibilità su 2^k di ingannare Victor.

Possiamo concludere che il protocollo 11. è più efficiente del protocollo 10. in termini di numero di comunicazioni necessarie, perché prendendo $k = 5$ e $t = 4$ (con t che corrisponde al numero di esecuzioni del protocollo) si ricava lo stesso grado di certezza che attraverso 20 iterazioni del protocollo 10.

4.3.1 Applicazione del protocollo nel caso di autenticazione con password

Il protocollo 11. può essere utilizzato per l'autenticazione di un utente U ad un host H tramite l'uso di una o più password, senza che queste vengano passate in alcun modo (in chiaro o cifrate) nel canale di comunicazione. Infatti, se nel protocollo consideriamo i ruoli di U e di H rispettivamente come quelli di Peggy e Victor, allora otteniamo un protocollo di autenticazione.

Se consideriamo una stringa ID_U che contiene tutte le informazioni necessarie per l'identificazione dell'utente U, una funzione hash H pubblica e un'autorità fidata A, possiamo considerare questo scenario: A sceglie un numero $n = pq$ con p, q numeri primi sufficientemente grandi, poi calcola $H(ID_U||j)$[2] per più valori di j. Poiché A conosce sia p che q, può facilmente determinare quali valori di $H(ID_U||j)$ (per j fissato) hanno una radice quadrata modulo n ed inoltre è in grado di calcolarle. Allora A riesce ad ottenere i numeri $v_1 = H(ID_U||j_1), \ldots, v_k = H(ID_U||j_k)$ e le rispettive radici quadrate $s_1, \ldots, s_k$. In seguito, A pubblica i valori $ID_U, n, j_1, \ldots, j_k$, comunica ad U i valori segreti (le password) $s_1, \ldots, s_k$ e poi li scarta insieme ai valori di p e q. Così facendo, anche se il sistema di A venisse violato nessuna informazione sensibile di U verrebbe compromessa. Inoltre, se ogni utente ha un diverso valore di n non è possibile compromettere la sicurezza di più utenti in un'unica volta.

Vediamo un esempio concreto di questa applicazione.
Se per esempio l'utente U si reca ad uno sportello Bancomat (host H), la macchina leggerà l'identificativo di U, ID_U, dalla carta inserita nell'ATM e potrà scaricare dal database i valori $n, j_1, \ldots, j_k$ relativi all'utente U. In seguito calcolerà i valori $v_i = H(ID_U||j_i)$ per $1 \leq i \leq k$ ed eseguirà il protocollo 11 per verificare se U conosce i valori segreti $s_1, \ldots, s_k$. Una volta che il protocollo sarà stato eseguito più volte e la macchina dello sportello si sarà convinta dell'identità di U, allora gli permetterà di svolgere una delle operazioni possibili, come ad esempio il prelievo o l'interrogazione del conto corrente. In

[2]La notazione $ID_U||j$ indica la concatenazione delle stringhe ID_U e j

questo caso, i valori segreti potrebbero essere memorizzati in un microchip all'interno della carta, in modo tale che U non avrebbe bisogno di ricordarsi i diversi valori segreti, ma memorizzare soltanto un PIN.

4.3.2 Schema di identificazione di Schnorr

Un ulteriore caso particolare di applicazione del protocollo 11. è lo **schema di identificazione di Schnorr**.
Vediamo un primo protocollo a conoscenza zero basato sul protocollo 11. da cui otterremo successivamente la procedura dello schema di Schnorr.

Protocollo 12. Protocollo alla base dello schema di identificazione di Schnorr

Premesse: Sia p un numero primo sufficientemente grande, sia α una radice primitiva di $\mathbb{Z}_p^*$, ovvero un generatore del gruppo $\mathbb{Z}_p^*$, e sia $\beta = \alpha^a \pmod{p}$.
Supponiamo che i numeri p, α, β siano pubblici e che Peggy conosca il valore segreto a.

Obiettivo: Victor vuole verificare che Peggy conosca il numero segreto a. Peggy vuole convincere Victor, senza rivelargli il valore segreto.

1. Peggy sceglie un numero intero casuale $r \pmod{p-1}$ e calcola $h_1 = \alpha^r \pmod{p}$ e $h_2 = \alpha^{a-r} \pmod{p}$
2. Peggy $\rightarrow$ Victor : h_1, h_2
3. Victor $\rightarrow$ Peggy : $i = 1$ (oppure $i = 2$)
4. Peggy $\rightarrow$ Victor : $r_i = a \cdot (i-1) - r \pmod{p-1}$
5. Victor verifica se $h_1 h_2 = \beta \pmod{p}$ e se $h_i = \alpha^{r_i} \pmod{p}$

Il protocollo viene ripetuto t volte, cambiando di volta in volta il valore del numero casuale r, finché Victor non è convinto che Peggy conosce il numero segreto a.

Con questa procedura Peggy dimostra a Victor di conoscere un logaritmo discreto, che è soluzione di un problema difficile come la fattorizzazione di un numero composto nel prodotto di una coppia di numeri primi.

Notiamo che se Victor non avesse a disposizione i valori h_1, h_2 non si potrebbe convincere che Peggy conosce effettivamente il valore a, perché la prima verifica non sarebbe più realizzabile e la seconda non avrebbe più significato, quindi Victor non otterrebbe alcuna informazione utile.

Vediamo ora come ottenere lo schema di identificazione di Schnorr dal protocollo 12. e mostrandone poi alcuni utilizzi possibili nella pratica.

Protocollo 13. Schema di identificazione di Schnorr

Premesse: Sia p un numero primo sufficientemente grande, sia α una radice primitiva di $\mathbb{Z}_p^*$ e sia $\beta = \alpha^a \pmod{p}$.
Supponiamo che i numeri p, α, β siano pubblici e che Peggy conosca il valore segreto a.

Obiettivo: Victor vuole verificare che Peggy conosce il numero segreto a. Peggy vuole convincere Victor, senza rivelargli il valore segreto.

1. Peggy sceglie un numero intero casuale k con $1 \leq k < p-1$ e calcola $\gamma = \alpha^k \pmod{p}$
2. Peggy $\rightarrow$ Victor : γ
3. Victor sceglie un numero intero casuale r con $1 \leq r < p-1$
4. Victor $\rightarrow$ Peggy : r
5. Peggy calcola $y = k - ar \pmod{p-1}$
6. Peggy $\rightarrow$ Victor : y
7. Victor verifica se $\gamma = \alpha^y \beta^r \pmod{p}$. Se la verifica ha successo si convince che Peggy conosce il valore segreto a

---

Di solito, per ottenere un utilizzo ottimale del protocollo il valore di p è tale che $p-1$ abbia un fattore primo grande q e α, invece di essere una radice primitiva, è un numero tale che $\alpha^q = 1 \pmod{p}$. Così facendo la congruenza che definisce y è modulo q. Un

altro accorgimento per migliorare il protocollo è prendere r tale che $1 \leq r \leq 2^t$ per un certo valore di t.

Se nel protocollo 13. Victor rappresenta una banca e Peggy un utente, allora il valore segreto a rappresenterebbe il PIN della carta di Peggy. In questo scenario, la banca avrebbe in un file salvato il valore di β e Peggy, per accedere al proprio conto, dovrebbe dimostrare di conoscere a. Tuttavia, se Peggy sta comunicando con la banca, ovvero con Victor, attraverso un canale insicuro, allora Peggy non intende riferire direttamente il codice segreto né svelarne alcuna informazione utile, perciò questo protocollo a conoscenza zero risulta essere una buona soluzione del problema.

Conclusioni

In questo lavoro, abbiamo analizzato diversi protocolli di autenticazione relativi allo scenario user-host con uso di password.

Abbiamo iniziato con una breve analisi dei concetti base, da cui poi siamo partiti per sviluppare l'argomento principale della trattazione del tema. Dopo questa premessa, abbiamo affrontato lo studio dei protocolli di autenticazione partendo dall'analisi dei protocolli più intuitivi e ingenui, quali il protocollo di Needham e i protocolli con uso di *one-time password*. Di questi, abbiamo analizzato i punti di forza e le criticità, in modo da individuare quali aspetti modificare e migliorare per rendere più sicuri i protocolli analizzati.

Abbiamo ulteriormente approfondito l'argomento studiando due tipologie di protocollo di autenticazione considerati sicuri e che affrontano il problema con approcci totalmente diversi: i protocolli EKE e i protocolli a conoscenza zero:

- nel primo caso il protocollo utilizza la password come chiave simmetrica per cifrare altre chiavi, tra cui quella di sessione, e *nonces*. In tal modo, si ottiene l'autenticazione e si può stabilire un'altra chiave per le successive comunicazioni. Osserviamo che in questo tipo di protocollo ogni messaggio passato sul canale è cifrato, perciò un avversario non riuscirà a distinguere i messaggi dal resto rumore del canale, non riuscendo di conseguenza ad interferire sulla comunicazione.

- nel secondo caso invece, la password, ovvero l'informazione segreta, non viene mai rivelata né utilizzata come chiave o come argomento di una qualche funzione hash. Tuttavia si riesce a convincere l'altra parte della propria identità, potendo quindi ottenere l'autenticazione e dimostrando di poter eseguire determinate operazioni possibili solamente se si è in possesso dell'informazione segreta. In questo caso, ogni messaggio inviato attraverso il canale non sicuro è in chiaro, ma nessuno di essi può essere utilizzato da un avversario per evincerne la password.

Queste due tipologie di protocolli vengono frequentemente utilizzate anche in applicazioni quotidiane.

I protocolli EKE, nella versione che abbiamo mostrato che sfrutta lo schema di Diffie-Hellman, ma anche nelle successive versioni modificate che non abbiamo qui contemplato,

sono utilizzati nel cosiddetto EAP[3], sigla per *Extensible Authentication Protocol*. Questo protocollo è solitamente implementato nei meccanismi di autenticazione per gli *access point* e le comunicazioni *point-to-point*. Infatti, nel caso di una rete wireless per esempio, questo protocollo fa sì che l'accesso di un client alla rete non venga autenticato direttamente dall'*access point*, ma da un altro server specifico configurato in particolare per questo scopo.

Invece, per quanto riguarda i protocolli a conoscenza zero, abbiamo visto a conclusione del quarto capitolo alcune applicazioni. In particolare, abbiamo visto come questa tecnica possa essere utilizzata nell'autenticazione di una carta di credito in un ATM.

Concludiamo quindi il nostro percorso di studio, osservando che protocolli studiati sono alla base delle operazioni che svolgiamo quotidianamente e che nella percezione comune e immediata diamo per scontato, o comunque ci auguriamo, siano sicure. Pertanto è importante che lo studio di questi argomenti prosegua con l'obiettivo di migliorare e perfezionare i protocolli già presenti, ma anche nella prospettiva di idearne nuovi, più sofisticati e ancora più sicuri.

[3]Per le informazioni riportate nella trattazione è stato consultato il sito [14] in data 5/09/2019

Bibliografia

[1] Boyd C., Mathuria A., (2003) *Protocols for Authentication and Key Establishment*, Springer-Verlag.

[2] Wenbo Mao,(2003) *Modern Cryptography Theory and Practice*, Prentice Hall PTR.

[3] Trappe W., Washington L.C., (2009) *Crittografia: con elementi di teoria dei codici*, Pearson Paravia Bruno Mondadori.

[4] Pass R., Shelat A., (2010) *A Course in Cryptography.*
Disponibile online sul sito: *https://www.cs.cornell.edu/courses/cs4830/2010fa/lecnotes.pdf.*

[5] Koblitz N., (1987) *A Course in Number Theory and Cryptography*, SPringer-Verlag.

[6] Washington L.C., (2008) *Elliptic Curves: Number Theory and Cryptography*, Taylor & Francis Group.

[7] Fastweb: "Cos'è UNIX e come funziona":
https://www.fastweb.it/web-e-digital/cos-e-unix-e-come-funziona/

[8] Wikipedia: "Unix":
https://it.wikipedia.org/wiki/Unix

[9] Wikipedia: "Data Integrity":
https://en.wikipedia.org/wiki/Data_integrity

[10] Wikipedia: "Sistema distribuito":
https://it.wikipedia.org/wiki/Sistema_distribuito

[11] Wikipedia: "Attacco a dizionario":
https://it.wikipedia.org/wiki/Attacco_a_dizionario

[12] Wikipedia: "Attacco *man-in-the-middle*":
https://it.wikipedia.org/wiki/Attacco_man_in_the_middle

[13] Wikipedia: "Certificato digitale":
https://it.wikipedia.org/wiki/Certificato_digitale

[14] Wikipedia: "Extensible Authentication Protocol":
https://en.wikipedia.org/wiki/Extensible_Authentication_Protocol

Ringraziamenti

Ringrazio innanzitutto il mio relatore, il professor Davide Aliffi, per la sua disponibilità e per aver accettato di seguirmi durante questo percorso.

Ringrazio i miei genitori che amo con tutto il cuore e che mi hanno sostenuto durante questi tre anni (e ovviamente non solo questi ultimi tre anni). Mi hanno aiutato sopportandomi nei miei momenti di ansia pre-esame e supportandomi in ogni modo fosse loro possibile. Mi sono sempre stati accanto e hanno sempre fatto il tifo per me. Spero che continuino così anche in futuro e che io non riesca mai a dargli un motivo per fare altrimenti. Vi ringrazio inoltre per avermi sostenuto ogni volta che dubitavo di me, delle mie capacità e/o delle possibilità che avevo nel superare una difficoltà.
Quello per cui vi ringrazio qua è una briciola di tutto quello per cui vorrei e dovrei ringraziarvi, sono davvero fortunata ad avervi come miei genitori!

Vorrei poi ringraziare mio fratello che, nonostante io non abbia scelto la carriera da odontoiatra come lui si augurava (XD), mi è stato accanto e mi ha aiutato immensamente nel corso dei tre anni, che fosse con un aiuto tecnico, con del supporto morale, con battute o incoraggiandomi a fare del mio meglio.
Inoltre vorrei ringraziare il nuovo membro ufficiale della famiglia Vecchi - Raposo Pereira - Baltazar, Francesca. Solo l'estate scorsa è entrata a fare parte a tutti gli effetti della famiglia, ma in realtà ne faceva già parte da diversi anni. Anche lei devo ringraziare per essere sempre stata presente e disponibile in questi tre anni e per avermi sostenuto già da prima di questi ultimi tre anni.

Ringrazio poi Davide, che è arrivato nella mia vita non poi così tanto per caso come credevo. Grazie di far parte del mio fanclub e di credere tanto in me e nelle mie capacità, più di quanto io non creda in me stessa o a volte addirittura di quanto tu non creda in te stesso. Grazie per avermi cambiato la vita e averla resa migliore.

Grazie a tutti i miei amici e le mie amiche che mi sono stati accanto in questi tre anni. Di questi vorrei ringraziarne in particolare alcuni.
Vorrei ringraziare Elena, la migliore amica al mondo che potessi trovare. Grazie di esserci sempre in tutti i momenti, quelli belli e quelli brutti. Grazie di ascoltare tutte le

mie lamentele e i miei sfoghi e grazie di tutte le chiacchierate (toccasana) che ci facciamo. Ti voglio un mondo di bene e senza te questi tre anni sarebbero stati un'altra storia.

Ai miei amici dell'università grazie di tutti i momenti trascorsi insieme. Vorrei ringraziare in particolare Marina, Alessandro, Angela, Arianna, Elena, Alessandra e Ramona (ce ne sarebbero anche altri da ringraziare e che hanno riempito e vissuto questi tre anni con me, ma non posso nominare tutti, perciò mi limito a citare coloro con cui ho trascorso più tempo dentro e fuori le mura di matematica). A voi, grazie di tutte le risate, i momenti di sfogo e disperazione, dei compleanni a sorpresa che ormai sono una tradizione, dei pranzi per le scale o a Filippo Re, dei caffè post pranzo e di tutto quanto. Per fortuna nessuno di noi andrà da nessun'altra parte!
Ale e Mari, a voi un grazie speciale! Grazie per il legame che si è creato, per tutto l'aiuto e il sostegno che ci siamo dati, non solo per cose riguardanti l'università. Grazie perché so che posso contare sempre su di voi.

Alle mie amiche di danza: Sara, Monia, Giulia, Teresa e Matilde (e anche Clò ovviamente). Mi avete sopportano nonostante vi abbia paccate in continuazione per studiare o per preparare un esame, ma nonostante tutto mi siete rimaste accanto e avete fatto il tifo per me. Grazie per le risate, le serate insieme e ogni singolo momento di follia trascorso insieme.

Infine, ma non per importanza, vorrei ringraziare i miei nonni, a cui è dedicata la tesi. L'amore che mi date è qualcosa di indescrivibile e, che siate a due o a migliaia chilometri di distanza da me, vi porto sempre con me. Magari vi vedo poco e meno di quanto sia io che voi vorremmo, ma vi posso assicurare che il bene che vi voglio è un pozzo senza fondo.
Ringrazio i miei nonni paterni, Peo e Tina, anche voi fate parte del mio fanclub e avete sempre creduto ciecamente in me. Mi avete sostenuto e incoraggiato durante ogni singolo passo di questo cammino. Fate a modo che dobbiamo arrivare tutti insieme e messi bene alla crociera!
Grazie a mia nonna materna, Carlota, che, grazie alla sua passione e curiosità, mi ricorda che la matematica è anche un gioco e può essere divertente.
E grazie a mio nonno materno, Carlos, che non ho mai conosciuto, ma che spero stia seguendo le vicende della mia vita e che sia orgoglioso di chi sono diventata e di chi diventerò in futuro.