Un'analisi dei principali protocolli di autenticazione corredata di un caso di studio - Report finale

Guido Latini
guido.latini@studio.unibo.it

marzo 2022

L'ISO/IEC 27000 definisce l'autenticazione come *"un processo utilizzato per confermare che una caratteristica dichiarata di un'entità sia effettivamente corretta"*.
L'autenticazione consiste, dunque, nel processo di verifica della rivendicazione di una determinata proprietà.

Il progetto intende fornire una *panoramica dei principali protocolli di autenticazione, analizzandone le debolezze ed i punti di forza; come caso di studio di quanto trattato, illustra l'esecuzione di un protocollo di autenticazione basato su password utilizzato da una web application sviluppata*; tale protocollo viene eseguito dai partecipanti in scenari differenti che dipendono dalla configurazione dei sistemi interessati; per ciascun scenario viene effettuata un'autovalutazione della sicurezza dei sistemi, che partecipano al protocollo, sotto forma di vulnerability assessment, mediante l'utilizzo di un sistema terzo con il ruolo di avversario, che effettua attacchi ai danni dei partecipanti allo scopo di individuare le credenziali di autenticazione.

Il progetto è strutturato secondo il seguente schema:

- *presentazione generale con premesse concettuali e nozioni di base* sui protocolli crittografici, la sicurezza computazionale, i protocolli di autenticazione, le tipologie di autenticazione, gli attacchi a tali protocolli e le convenzioni di comportamento dei partecipanti nell'esecuzione;
- *trattazione dei principali protocolli esaminati*, con particolare riferimento a quelli basati su password, su challenge-response ed a conoscenza zero, comprensiva per ciascun protocollo della descrizione delle caratteristiche, debolezze e punti di forza;
- *illustrazione del caso di studio relativo alla web application sviluppata*, che utilizza un protocollo di autenticazione basato su password, comprensivo dei dettagli di implementazione di tale applicazione e della virtualizzazione applicata ai sistemi realizzati che interagiscono con la stessa nell'esecuzione del protocollo, dell'autovalutazione della sicurezza di tali sistemi e di un approfondimento sull'implementazione del SSL/TLS e di ulteriori strategie di remediation alle debolezze del protocollo impiegato.

1 Obiettivo/Requisiti

Come indicato nell'abstract, *l'obiettivo primario del progetto consiste nella trattazione dei principali protocolli di autenticazione comprensiva dell'analisi delle debolezze e dei punti di forza*; *un ulteriore obiettivo consiste nell'evidenziare l'importanza di un'adeguata conoscenza dei protocolli di autenticazione per effettuare la scelta del protocollo più appropriato da utilizzare in un sistema per il raggiungimento degli obiettivi di sicurezza prefissati*; la scelta, chiaramente, deve tener conto, tra gli altri, dello scenario di applicazione, dei servizi forniti, delle applicazioni utilizzate, dei dispositivi coinvolti e delle reti utilizzate dal sistema; il caso di studio trattato, con lo sviluppo della parte pratica del progetto, è mirato proprio al raggiungimento di tale obiettivo evidenziando, con gli attacchi condotti ai danni dei partecipanti al protocollo di autenticazione basato su password, le criticità derivanti dalla debolezza del protocollo scelto sicuramente non adeguato agli scenari proposti.

Con riferimento ai *requisiti*, al fine di agevolare la trattazione dei singoli protocolli di autenticazione esaminati, si rende necessario, come anticipato, introdurre delle *premesse concettuali e delle nozioni di base relative ai protocolli crittografici, alla sicurezza computazionale, ai protocolli di autenticazione, alle tipologie di autenticazione e agli attacchi ai protocolli di autenticazione; si definiscono, infine, le convenzioni di comportamento dei partecipanti nell'esecuzione dei protocolli di autenticazione.*

Per quanto riguarda gli ulteriori requisiti, anche impliciti, necessari per l'implementazione del caso di studio che ha previsto, tra gli altri, lo sviluppo di una web application, che utilizza un protocollo di autenticazione in un ambiente virtuale in cui sono presenti oltre al web server in cui è residente l'applicazione, il sistema da cui un utente effettua l'autenticazione e quello da cui un avversario esegue gli attacchi per impossessarsi delle credenziali, si rimanda all'apposita sezione relativa all'analisi dei requisiti.

1.1 Scenari

Gli scenari di riferimento in cui viene eseguito il protocollo di autenticazione utilizzato dalla web application, sviluppata per la realizzazione del caso di studio, sono i seguenti:

- scenario 1: autenticazione di un utente, che si connette in HTTP tramite browser, ad una web application residente in un web server, realizzato mediante Apache-Tomcat, con un protocollo di autenticazione "ingenuo" basato su password;
- scenario 2: autenticazione di un utente, che si connette in HTTP tramite browser, ad una web application residente in un web server, realizzato mediante Apache configurato come reverse proxy per Tomcat, con il medesimo protocollo di autenticazione "ingenuo" basato su password;
- scenario 3: autenticazione di un utente, che si connette in HTTPS tramite browser, ad una web application residente in un web server, realizzato mediante Apache, configurato come reverse proxy per Tomcat, con l'utilizzo del TLS/SSL e di un certificato digitale self-signed, con lo stesso protocollo di autenticazione "ingenuo" basato su password.

Nella sezione relativa alla trattazione del caso di studio, ciascun scenario è stato illustrato mediante un diagramma di sequenza che evidenzia i protocolli http e https utilizzati nell'interazione tra i partecipanti e l'autenticazione effettuata dalla applicazione sviluppata residente nel web server; a riguardo, si anticipa che, per semplificare l'implementazione, il web server è stato stato installato nello stessa sistema in cui risiede il reverse proxy indicato nell'ultimo scenario; inoltre, per agevolare lo sviluppo della web application, non è stata implementata un'architettura multi-tier in cui il livello dei dati prevede l'utilizzo di un database; dunque la verifica delle credenziali degli utenti che intendono autenticarsi, analogamente alle loro autorizzazioni, viene gestita direttamente dalle classi java dell'applicazione, essendo le stesse memorizzate in chiaro nel codice.
In tutti gli scenari indicati, gli attacchi, eseguiti da un avversario che vuole impossessarsi delle credenziali di autenticazione, non sfruttano tale ulteriore debolezza riferibile al protocollo scelto basato su password.

1.2 Politica di autovalutazione

Con riferimento alla *qualità del progetto* si riportano le seguenti considerazioni.
Nonostante l'organizzazione dello stesso mirata a fornire una panoramica generale dei protocolli di autenticazione, risulta del tutto evidente che, *data la vastità degli argomenti e le tematiche in continua evoluzione, la trattazione non può essere considerata esaustiva e, in alcuni aspetti, risulta lacunosa*; nel progetto, infatti, realizzato a scopo didattico, si approfondiscono alcune delle principali tipologie di protocolli di autenticazione più diffusi a partire dagli anni sessanta; nella sezione lavori futuri sono evidenziati degli esempi di protocolli che non sono stati oggetto di trattazione nel progetto, ognuno dei quali, considerate la diffusione negli attuali scenari di produzione e la complessità di implementazione, meriterebbe un lavoro dedicato. In ogni caso, per maggiori approfondimenti su quanto illustrato, si rimanda alla fonti indicate nella bibliografia.

Per quanto riguarda la *qualità del software prodotto* si evidenzia quanto segue.
Il software sviluppato, ovvero la web application, deve essere valutata tenuto conto esclusivamente della sua funzione didattica per consentire l'esecuzione del protocollo di autenticazione scelto basato su password, non essendo ovviamente applicabile in un ambiente di produzione, considerando anche la semplicità dell'implementazione.
Pur con le semplificazioni indicate, l'applicazione realizzata è comunque significativa per evidenziare le debolezze del protocollo di autenticazione utilizzato mediante gli attacchi condotti da un avversario ai danni dei sistemi coinvolti nell'esecuzione del protocollo.

Per quanto concerne la *valutazione dell'efficacia dei risultati del progetto* valgono le seguenti osservazioni.
Un'apposita sezione è stata dedicata all'autovalutazione dei sistemi che interagiscono con la web application nell'esecuzione del protocollo di autenticazione nell'ambiente virtuale realizzato; *come criterio di valutazione si è scelto la verifica della sicurezza dell'esecuzione del protocollo* e, pertanto, nella valutazione oltre al protocollo di autenticazione utilizzato dalla web application è stato necessario prendere in considerazione i protocolli di comunicazione (HTTP e HTTP over TLS) utilizzati tra i partecipanti oltreché le loro configurazioni sistemistiche.

La valutazione viene effettuata in ciascuno degli scenari indicati nella definizione del caso di studio; a riguardo, per ciascun scenario, sono riportate le fasi del *vulnerability assesment* realizzato ovvero i *test* condotti mediante l'esecuzione degli attacchi come indicato, l'*analisi*, l'*assessmnet* e le possibili soluzioni per la *remediation.*
Le tipologie di attacco, finalizzate a scoprire le credenziali necessarie per effettuare l'autenticazione mediante il web form dell'applicazione, dipendono dai protocolli di comunicazione utilizzati nei differenti scenari e dall'eventuale impiego di algoritmi crittografici; con riferimento a tali scenari, si evidenzia che nell'ambiente di virtualizzazione implementato i sistemi utilizzati sono stati configurati per evidenziare la debolezza del protocollo di autenticazione utilizzato dalla web application e mostrare che, nonostante le ottimizzazioni nelle configurazioni dei web server e l'utilizzo del protocollo SSL, le criticità permangono; dunque *nell'ottica di un'adeguata valutazione della sicurezza dei sistemi che partecipano all'esecuzione del protocollo, oltre alle limitazioni descritte dell'applicazione sviluppata a livello didattico, si deve tener conto del fatto che tali sistemi interagiscono in un ambiente ben differente da un contesto di produzione.*
Tenendo conto del contesto illustrato, gli scenari proposti nel caso di studio permettono, in ogni caso, una valutazione della sicurezza mirata ad evidenziare le possibili criticità a cui possono essere soggetti i sistemi in caso di configurazioni non adeguate e di utilizzo di applicazioni che richiedono protocolli di autenticazione deboli.
Infine, si osserva che la metodologia dell'impiego della virtualizzazione nell'esecuzione del protocollo oggetto di studio, può essere applicata all'analisi di un qualunque protocollo di autenticazione, al fine di valutarne i punti di forza e le debolezze, prima di un impiego in un ambiente di produzione; *il progetto, quindi, con la trattazione dei protocolli esaminati ed il caso di studio illustrato, vuole rappresentare uno punto di partenza per offrire gli strumenti necessari al miglioramento della sicurezza di un sistema, che utilizza un protocollo di autenticazione, mediante l'adozione delle soluzioni più adeguate.*

2 Analisi dei requisiti

Il progetto richiede come *requisiti di base* la conoscenza dei principi dei sistemi distribuiti, con particolare riferimento alle architetture, ai processi, alla comunicazione, al naming ed, ovviamente, alla sicurezza.

Costituiscono dei *requisiti impliciti* le conoscenze matematiche relative all'algebra, alla crittografia ed alla crittoanalisi; per quanto riguarda, invece, l'implementazione degli scenari previsti dal caso di studio, sono richieste conoscenze sistemistiche necessarie all'implementazione delle virtual machine nell'ambiente Oracle Virtual Box utilizzato; ulteriori competenze sono richieste per l'amministrazione dei sistemi Linux installati in tali virtual machine; sono basilari, tra gli altri, le conoscenze delle configurazioni di sicurezza dei web server necessari per l'erogazione del servizio che consente l'esecuzione del protocollo di autenticazione tramite la web application realizzata.
Nello specifico l'applicazione è stata sviluppata utilizzando il linguaggio Java ed impiegando i web server Apache Tomcat, nella forma di contenitore servlet, per implementare le Java Server Pages e le servlet, ed Apache.
A riguardo, per avere una gestione integrata dell'amministrazione di Apache Tomcat e dello sviluppo della web application, è stato utilizzato l'IDE Eclipse.

Considerata la complessità delle numerose configurazioni sistemistiche effettuate nel corso del progetto, illustrate in dettaglio nella sezione dedicata al caso di studio, come anticipato, si è deciso di optare per una *tecnologia basata su un ambiente virtuale "tradizionale" che utilizza virtual machine piuttosto che su microservizi*; la scelta di quest'ultima soluzione, la cui implementazione sicuramente sarebbe stata più flessibile, avrebbe però ulteriormente aggravato i già lunghi tempi di realizzazione del progetto.
Come indicato nella sezione relativa ai futuri lavori, la progettazione della transizione dall'architettura monolitica utilizzata nel progetto ad una a microservizi potrebbe rappresentare un'ulteriore attività per approfondire le tecniche di progettazione Domain Driven Design di microservizi, magari utilizzando servizi in Cloud.
Nell'approccio seguito, dunque, i servizi della web application sono stati impacchettati in un file WAR e distribuiti come una sola unità; tale struttura, seppur caratterizzata da una forte rigidità e, quindi, non adatta alla gestione di applicazioni complesse che si evolvono velocemente, per lo scopo didattico con cui è stata concepita, è idonea ad un rapido utilizzo degli scenari proposti.

Ulteriori requisiti impliciti per lo sviluppo del progetto sono rappresentati, infine, dalle conoscenze e competenze necessarie per effettuare un vulnerability assessment di un sistema distribuito che richiede, tra gli altri, l'utilizzo di appositi tool per l'esecuzione di tutti i test necessari; negli scenari oggetto del caso di studio, come argomentato in dettaglio nell'apposita sezione, si utilizzano in modo avanzato i tool di sniffing e spoofing, in dotazione nella distribuzione linux Kali installata nelle virtual machine dell'ambiente implementato, *Wireshark*, *Ettercap* ed *SSLsplit*; a riguardo, per agevolare la comprensione dei test, ovvero degli attacchi attivi e passivi effettuati con tali tool, considerate le numerose operazioni da svolgere in ciascuno degli scenari indicati, si è optato per dettagliare in modo analitico i passi effettuati al fine agevolarne la comprensione.

3 Progetto

3.1 Premesse concettuali e nozioni di base

Definizione 1 (Sicurezza) *Nel campo delle comunicazioni la sicurezza può essere definita come l'insieme di quelle attività che permettono la protezione dei sistemi e delle reti di elaborazione attraverso procedure specifiche in modo da controllare la fruibilità dei dati e delle applicazioni sulla base di politiche di accesso predefinite.*

Poiché la salvaguardia ed il controllo dell'informazione digitale sulle reti di comunicazione sono sovente realizzate mediante l'impiego di protocolli crittografici si introdurranno delle nozioni di base in merito a tali protocolli che richiedono l'utilizzo di concetti matematici alla base dei sistemi crittografici.

Definizione 2 (Plain text e testo cifrato) *In una comunicazione un messaggio, nella sua forma originaria, è chiamato testo in chiaro (plain text); un mittente, che cripta il messaggio, ottiene un testo cifrato o criptotesto; il destinatario, che riceve il messaggio, decripta il criptotesto. Mittente e destinatario sono anche detti utilizzatori legali o autorizzati del sistema crittografico.*

Definizione 3 (Sistema crittografico) *Un sistema crittografico è costituito da una quintupla (PT,CT,K,E,D) dove*

- *PT è composto da tutti i possibili testi in chiaro, ossia stringhe m su un prefissato alfabeto Σ;*
- *CT è composto dai criptotesti, ossia stringhe c su un prefissato alfabeto $\Sigma^{'}$;*
- *K è l'insieme delle chiavi e D,E due algoritmi:*

$$E : K \times PT \to CT$$
$$D : K \times CT \to PT$$

tali che ogni chiave k determina un metodo di criptazione $E(k, m) = m^{'} \in CT$ ed un metodo di decriptazione $D(k, m^{'}) = D(k, E(k, m)) = m \in PT$.

Coloro che utilizzano un sistema crittografico devono accordarsi sui dettagli degli oggetti descritti ed utilizzarli come convenuto ossia seguire un *protocollo*.
Allo scopo di non far confusione tra i termini *codificare* e *crittografare*, si intenderà per *codifica* la trasformazione di un messaggio da un alfabeto ad un altro *senza lo scopo di renderlo incomprensibile.*

Definizione 4 (Sistemi crittografici block cipher e stream cipher) *I sistemi crittografici possono essere suddivisi in due famiglie:*

- *block chiper nei quali dato un messaggio m (eventualmente già codificato), m viene diviso in blocchi di lunghezza fissata che vengono criptati uno ad una con lo stesso sistema e la stessa chiave k. Ogni blocco può essere anche una lettera;*

- *stream chiper nei quali un messaggio m (eventualmente già codificato) viene diviso in blocchi m_j (solitamente di una lettera), che vengono criptati con una chiave dipendente da j.*

I sistemi che utilizzano il metodo *chiper* sono ritenuti più efficienti di quelli che adottano il *block cipher* perché un eventuale errore di trasmissione sarebbe facilmente rilevabile (per le j) e, quindi, non comprometterebbe tutte le operazioni effettuate.

Definizione 5 (Classificazione dei sistemi in base alla chiavi utilizzate) *Una ulteriore classificazione/definizione relativa ai sistemi crittografici riguarda l'utilizzo delle chiavi; si distinguono, infatti, la seguenti famiglie che si avrà modo di approfondire nel corso della trattazione dei protocolli di autenticazione:*

- *sistemi ad una chiave o simmetrici o a chiave privata che permettono la criptazione e decriptazione di messaggi con la stessa chiave o con chiavi diverse ma facilmente ricavabili l'una dall'altra;*
- *sistemi a due chiavi o asimmetrici o a chiave pubblica, tra i quali i sistemi a chiave pubblica;*
- *sistemi a chiave multipla, usati in genere per i controlli di accesso, per i controlli condivisi e per le firme di gruppo;*
- *sistemi senza chiave nei quali, in genere, si utilizzano funzioni hash one-way.*

3.1.1 Protocolli crittografici

Definizione 6 (Protocollo crittografico) *Un protocollo crittografico, definito anche protocollo di sicurezza o protocollo di crittografia, è un algoritmo distribuito tra n entità, con $n \geq 2$, che utilizzando tecniche crittografiche, specifica le azioni da intraprendere per ottenere un determinato obiettivo di sicurezza.*

Definizione 7 (Verifica di un protocollo crittografico) *La verifica di un protocollo crittografico è la dimostrazione del fatto che il protocollo realizza effettivamente gli obiettivi di sicurezza definiti e perseguiti in fase di progetto.*

La verifica di un protocollo crittografico è un attività non semplice e soggetta ad errori di varia natura.

Definizione 8 (Protocollo crittografico corretto) *Un protocollo crittografico si dice corretto se detto A l'insieme degli obiettivi previsti e B l'insieme degli obiettivi effettivamente raggiunti dal protocollo, risulta $A \subseteq B$.*

Tutte le tecniche di verifica di correttezza di un protocollo, siano esse formali o informali, cercano di esprimere gli *obiettivi di un protocollo in termini di un insieme di*

nozioni o proprietà di base, tra le quali, la *proprietà di autenticazione* e quelle relative ai singoli messaggi trasmessi dal protocollo, quali la *riservatezza* (o *confidenzialità*), l'*autenticità*, l'*unicità* e la *non ripudiabilità*.

Il concetto di *difficoltà computazionale* è uno dei punti chiave dei protocolli crittografici; infatti la crittografia moderna viene realizzata imponendo limiti realistici alle risorse degli avversari che intendono "rompere" gli algoritmi che utilizzano le tecniche di crittografia; una supposizione riguarda, ad esempio, il fatto che tali avversari siano in grado di eseguire solo algoritmi *con un tempo di esecuzione polinomiale*.
Si introdurranno, a riguardo, delle definizioni utili alla comprensione dei cosiddetti *problemi computazionalmente difficili*.

Informalmente con *complessità di un algoritmo* o *efficienza di un algoritmo* si fa riferimento alle risorse di calcolo richieste; i problemi sono, dunque, classificati in differenti *classi di complessità* in base all'efficienza del migliore algoritmo noto in grado di risolvere uno specifico problema.

Un ulteriore distinzione informale riguarda i cosiddetti *problemi "facili"*, di cui si conoscono algoritmi di esecuzione "efficienti" e *problemi "difficili"* di cui gli unici algoritmi noti non sono efficienti; *poiché la crittografia moderna si basa sull'esistenza dei problemi ritenuti "difficili"*, il loro studio ha un'enorme rilevanza per la sicurezza; infatti, qualora si dimostrasse l'esistenza di un algoritmo "*efficiente*" per un problema ritenuto "*difficile*", i sistemi crittografici basati su di esso non sarebbero più sicuri e di, conseguenza, sarebbe compromessa la sicurezza di tutti le componenti che interagiscono con tali sistemi.

Teorema 1 (Tesi di Church-Turing) *La classe delle funzioni calcolabili coincide con quella delle funzioni calcolabili da una macchina di Turing.*

Utilizzando la predetta tesi di Church-Turing, un qualunque modello di calcolo scelto (si può pensare anche ad un computer reale), ai fini della classificazione dei problemi, si comporta dunque come una macchina di Turing; pertanto, il modello della macchina di Turing è utilizzato per misurare l'efficienza di un algoritmo in maniera univoca con una metrica indipendente dalle tecnologie utilizzate.

Definizione 9 (Algoritmo come macchina di Turing deterministica) *In base alla tesi di Church-Turing un algoritmo può essere definito come una macchina di Turing (MdT) deterministica che riceve in input delle stringhe nell'alfabeto* $\Sigma = \{0, 1\}$ *e restituisce delle stringhe nell'alfabeto* Σ.

Definizione 10 (Tempo di esecuzione di un algoritmo) *Si dice che un algoritmo* $\mathcal{A}$ *ha un tempo di esecuzione* $T(n)$ *se* $T(n)$ *è il numero massimo di passi necessari per produrre il risultato su un input* x *di lunghezza* n *e si denota con* $|x| = n$.

Definizione 11 (Tempo di esecuzione polinomiale di un algoritmo) *Si dice che un algoritmo* $\mathcal{A}$ *ha un tempo di esecuzione polinomiale se il suo tempo di esecuzione* $T(n)$ *è limitato superiormente da un'espressione polinomiale nella dimensione dell'input, cioè se* $\exists\, k \in \mathbb{N}^+$ *tale che, utilizzando la notazione* $O(\cdot)$*, O-grande,* $T(n) = O(n^k)$*, formalmente:*

$$T(n) = O(f(n)) \; se \; \exists \, n_0 \in \mathbb{N} \, , \, \exists \, k \in \mathbb{N}^+ \; : \; \forall n > n_0 \, , \, T(n) \leq k f(n)$$

dove $f(n)$ è un espressione polinomiale.

In altre parole, un algoritmo $\mathcal{A}$ è eseguito in tempo polinomiale se esistono due costanti a e k tale che $\mathcal{A}(x)$ termina in $a \cdot |x|^k$ passi.

Definizione 12 (Algoritmo efficiente) *Si definisce un algoritmo efficiente un algoritmo che ha una tempo di esecuzione polinomiale.*

Usare il tempo polinomiale come limite massimo per definire l'efficienza di un algoritmo potrebbe essere soggetto a obiezioni; in effetti, se il grado del polinomio in questione è abbastanza grande, il calcolo non può essere efficiente in pratica; tuttavia l'esperienza suggerisce che gli algoritmi con tempo di esecuzione polinomiale risultano essere efficienti, ovvero "tempo polinomiale" significa quasi sempre "tempo cubico o migliore".

Definizione 13 (Tempo di esecuzione esponenziale di un algoritmo) *Si dice che un algoritmo $\mathcal{A}$ ha un tempo di esecuzione esponenziale se il suo tempo di esecuzione $T(n)$ è limitato superiormente da un'espressione esponenziale del tipo $O(2^{n^k})$, per una qualche costante k, nella dimensione dell'input.*

Gli algoritmi con tempo polinomiale polinomiale sono algoritmi efficienti, a differenza degli algoritmi "esponenziali", in quanto consentono un significativo aumento del tempo di esecuzione all'aumento della "velocità" di calcolo dell'elaboratore.

Il concetto di tempo di esecuzione polinomiale conduce a varie *classi di complessità* nella teoria della complessità computazionale. Una classe di complessità racchiude problemi computazionali con alcune caratteristiche comuni.

Definizione 14 (Macchina di Turing che decide un linguaggio) *Sia $L \subseteq \{0,1\}^*$ un linguaggio, ovvero un insieme arbitrario di stringhe binarie. Si dice che una macchina di Turing M decide un linguaggio L se è in grado si calcolare una funzione $\rho_L : \{0,1\}^* \to \{0,1\}$, tale che $\rho_L(x) = 1 \Leftrightarrow x \in L$. In altri termini, una macchina di Turing M decide $L \Leftrightarrow M(x) = \rho_L(x) \, , \, \forall \, x \in L$.*

Definizione 15 (Funzione complessità di tempo di una MdT) *Sia M una MdT deterministica che si ferma su ogni possibile input. Si definisce complessità di tempo, (o tempo di esecuzione) di M, la funzione $T_M : \mathbb{N} \to \mathbb{N}$, dove $f(n)$ è il numero massimo di passi che M compie su un input di lunghezza n. Se $f(n)$ è la complessità di tempo di M, allora si dice che M lavora in tempo $f(n)$, o che M è una macchina di tempo $f(n)$.*

Definizione 16 (Classe dei linguaggi TEMPO) *Si definisce classe $TEMPO\,(f)$ la classe dei linguaggi f riconoscibili dalla MdT M tali che $T_M \leq f$, dove $T_M(n)$ è il massimo tempo di arresto di una computazione terminante su input di lunghezza n della MdT M che ha costo tempo $T_M : \mathbb{N} \to \mathbb{N}$.*

Definizione 17 (classe dei linguaggi P) *Si definisce classe P la classe dei linguaggi riconosciuti in tempo polinomiale nella lunghezza dell'input; vale a dire*

$$P = \bigcup_{k \in \mathbb{N}} TEMPO\,(n^k)$$

ovvero la classe di complessità dei problemi decisionali che possono essere risolti su una MdT deterministica in tempo polinomiale.

In altri termini la classe P identifica i problemi che possono essere risolti in modo *efficiente*.

Definizione 18 (Problemi trattabili) *I problemi della classe P si definiscono trattabili ovvero i problemi la cui soluzioni possono essere trovate in tempo ragionevole.*

Definizione 19 (Problemi intrattabili) *I problemi per i quali non esiste un algoritmo con complessità polinomiale in grado di risolverli si definiscono intrattabili.*

È errato pensare che un problema intrattabile non abbia soluzione. Il problema intrattabile può avere o meno una soluzione; ciò che distingue il problema intrattabile dal problema trattabile è l'assenza di un algoritmo computazionale in grado di giungere alla soluzione *in tempi di esecuzione accettabili* e *con un consumo di risorse accettabile*. Un problema intrattabile potrebbe, ad esempio, essere risolvibile mediante algoritmi di complessità esponenziale; tuttavia, l'elevato consumo di risorse e/o il tempo di esecuzione proibitivo renderebbe l'algoritmo inutile.

Teorema 2 (Tesi di Church-Turing estesa) *La tesi di Church-Turing estesa afferma che i problemi di decisione trattabili sono quelli risolubili in tempo polinomiale nella lunghezza dell'input.*

Definizione 20 (classe dei linguaggi NTEMPO) *Si definisce $NTEMPO\,(f)$ la classe dei linguaggi verificabili da una MdT con costo in tempo f; vale a dire, $\forall\, w$ input accettato di lunghezza n, $\exists\, w^{'}$ certificato (o prova d'appartenenza) tale che su input w e $w^{'}$ la macchina termina al più in $f(n)$ passi.*

Definizione 21 (Classe dei linguaggi NP) *Si definisce NP la classe dei linguaggi verificabili in tempo polinomiale nella lunghezza dell'input; vale a dire*

$$NP = \bigcup_{k \in \mathbb{N}} NTEMPO\,(n^k)$$

ovvero la classe dei classe di complessità dei problemi decisionali le cui soluzioni positive possono essere verificate in tempo polinomiale avendo le giuste informazioni, o, equivalentemente, la cui soluzione può essere trovata in tempo polinomiale con una macchina di Turing non deterministica.

Definizione 22 (Linguaggio L nella classe NP) *Si dice che un linguaggio $L \in NP$ se $\exists\, M$ MdT con tempo di esecuzione polinomiale, tale che $\forall\, x \in \{0,1\}^*$:*

$$x \in L \Leftrightarrow \exists\, w \in \{0,1\}^{poly(|x|)} \text{ tale che } M(x,w) = 1,$$

dove con $poly(|x|)$ si intende che il numero di passi è $O(|x|^c)$ per una qualche costante $c \in \mathbb{N}$.

Come visto la classe P è data dai problemi che possono essere risolti in un tempo dato da una funzione $f(n) = O(n^k)$ polinomiale nella lunghezza dell'input; tale tipo di problemi non hanno applicazioni di tipo crittografico in quanto *non danno a chi vuole criptare i dati un vantaggio significativo su chi vuole decriptare gli stessi dati.*

La classe dei problemi *NP* è, invece, data dai problemi la cui *soluzione* può essere *verificata* in un tempo dato da una funzione $f(n) = O(n^k)$ polinomiale nella lunghezza n dell'input.

Definizione 23 (Problema fondamentale della complessità strutturale) *Il problema fondamentale della complessità strutturale, uno dei problemi più importanti della matematica contemporanea, riguarda la relazione tra le classi P e NP ovvero se l'inclusione $P \subseteq NP$ sia propria, cioè stabilire:*

$$P \stackrel{?}{=} NP$$

Esistono molti problemi in NP per i quali non si conosce un algoritmo di risoluzione che funzioni in tempo polinomiale e quindi per i quali non si sa se siano in P;

Il problema fondamentale della complessità strutturale rappresenta uno dei sette problemi ancora insoluti del millennio per cui il Clay Mathematics Institute ha messo in palio ben 1.000.000 $!

Finora non è stato dimostrato se $P = NP$ o $P \neq NP$. Considerando il caso estremo della stringa w vuota, $P \subseteq NP$. Se P fosse uguale a NP, allora la capacità di *verificare* la soluzione di un problema in modo efficiente, implicherebbe la capacità di *trovare* la stessa in modo efficiente; poiché tale scenario non è molto plausibile, si assume che $P \neq NP$, tuttavia una dimostrazione di tale problema ancora non esiste.

Dal problema fondamentale della complessità strutturale ne deriva che *la crittografia moderna è basata* sulla ricerca dei problemi che siano in NP ma non in P e, dunque, *sulla fiducia* che $P \neq NP$.

3.1.2 Sicurezza computazionale

Per la *sicurezza delle transazioni nei protocolli di autenticazione* si introducono i seguenti problemi fondamentali $P \neq NP$ usati in crittografia: il *problema della fattorizzazione*, il *problema del logaritmo discreto*, il *problema computazionale di Diffie-Hellmann* ed il problema della *residuosità quadratica.*

Per la formalizzazione dei problemi indicati sono esposti i seguenti concetti relativi alla cosiddetta *sicurezza computazionale.*

Definizione 24 (Calcolo deterministico) *Si dice che un algoritmo $\mathcal{A}$ calcola una funzione $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ se $\forall\, x \in \{0,1\}^*$, $\mathcal{A}$ con input x restituisce come output $f(x)$, ovvero $\mathcal{A}(x) = f(x)$.*

Una naturale estensione della nozione di calcolo deterministico è consentire ad un algoritmo di avere accesso ad *una fonte di generazione di eventi casuali*; quest'ulteriore condizione serve per consentire agli algoritmi una maggiore efficienza nel calcolo di alcuni compiti e sarà necessaria per la sicurezza dei sistemi crittografici presentati in seguito; si estende, quindi, la definizione di calcolo deterministico come segue.

Definizione 25 (Algoritmo deterministico - PPT) *Un algoritmo probabilistico o macchina di Turing probabilistica in tempo polinomiale, Probabilistic Polynomial Tempo (PPT), è una macchina di Turing dotata di un nastro casuale supplementare; ogni bit del nastro casuale viene scelto uniformemente ed in modo indipendente.*

In termini equivalenti, un algoritmo PPT è un macchina di Turing che ha accesso ad un "oracolo casuale" (lancio di moneta) che genera, su richiesta, un bit casuale.

Per definire la nozione di efficienza di calcolo è necessario chiarire il concetto di *tempo di esecuzione di un algoritmo probabilistico.*

Definizione 26 (Tempo di esecuzione di un algoritmo probabilistico) *Una MdT probabilistica $\mathcal{A}$ viene eseguita in tempo $T(n)$ se $\forall x \in \{0,1\}^*$ e per ogni nastro casuale, $\mathcal{A}(x)$ si arresta in un numero di passi minore o uguale a $T(|x|) = n$.*
Diremo che $\mathcal{A}$ è eseguito in tempo polinomiale (oppure $\mathcal{A}$ è un algoritmo probabilistico efficiente se

$$\exists\, a, r \text{ costanti tali che } \mathcal{A}(x) \text{ è eseguito in tempo } T(|x|) = a \cdot |x|^r\,,\ \forall x \in \{0,1\}^*\,.$$

Dalla definizione risulta che il tempo di esecuzione di un algoritmo probabilistico viene definito come il *limite superiore su tutti i nastri casuali*, perché le sequenze casuali di cui è dotato l'algoritmo probabilistico possono far variare il tempo della sua esecuzione.

Con la definizione seguente *si estende la nozione di computazione agli algoritmi probabilistici*; in particolare, una volta che un algoritmo ha un nastro casuale, il suo output diventa una *distribuzione* su qualche insieme. Nel caso del calcolo deterministico l'output è un insieme costituito da un singolo elemento.

Definizione 27 (Calcolo di un algoritmo probabilistico) *Si dice che un algoritmo probabilistico $\mathcal{A}$ calcola una funzione $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ se la probabilità che si verifichi l'evento $\mathcal{A}$, con input x restituisca come output $f(x)$, $\forall\, x \in \{0,1\}^*$, è 1, ovvero $\mathbb{P}[\mathcal{A}(x) = f(x), \forall\, x \in \{0,1\}^*] = 1$.*

In definitiva si utilizza come modello di efficienza di calcolo gli *algoritmi probabilistici in tempo polinomiale*; si fa riferimento a tale classe di algoritmi come *MdT probabilistiche in tempo polinomiale* (PPT) o, indifferentemente, come *algoritmi probabilistici efficienti.*

La nozione di *sicurezza computazionale* prende in considerazione la potenza di calcolo di un ipotetico *avversario* definito come segue; ne deriva che *un sistema di crittografia,*

a prescindere dallo schema utilizzato, deve essere computazionalmente sicuro di fronte ad avversari efficienti.

Definizione 28 (Avversario PPT) *Si dice che un avversario $\mathcal{A}$ è un avversario PPT se usa un certo grado di randomicità come parte della sua logica ovvero $\mathcal{A}$ è probabilistico e $\forall\, x \in \{0,1\}^*$, l'esecuzione di $\mathcal{A}(x)$ termina in un numero di passi t polinomiale nella lunghezza della stringa (al solito, si intende che il numero di passi è $O(|x|^c)$, per una qualche costante $c \in \mathbb{N}$).*

Al fine di dare un significato rigoroso al fatto che un sistema possa essere violato solamente con un probabilità trascurabile si introduce appunto la definizione di funzione trascurabile.

Definizione 29 (Funzione trascurabile) *Una funzione $\varepsilon(n)$, con $\varepsilon : \mathbb{N} \to \mathbb{R}$, è detta trascurabile se $\forall\, c \in \mathbb{N}^+\, \exists\, n_o \in \mathbb{N}^+$ tale che $\forall\, n > n_0$ si ha*

$$\varepsilon(n) < \tfrac{1}{n^c}\,.$$

Un crittosistema computazionalmente sicuro dipende da un *parametro statico di sicurezza* n noto a tutte le parti, attaccanti compresi; la probabilità di successo di un attacco al sistema ed il tempo necessario ad eseguire un attacco sono funzioni di tali parametro; si riporta, quindi, la definizione di sicurezza computazionale ovvero di sistema crittografico sicuro.

Definizione 30 (sistema crittografico sicuro) *Siano $t(n)\,,\, \varepsilon(n)$ costanti positive, con $\varepsilon < 1$, dipendenti da un parametro di sicurezza $n \in \mathbb{N}$. Un sistema crittografico è (t,ε)-sicuro se ogni avversario $\mathcal{A}$ PPT eseguibile in tempo $t(n)$, che tenta di violare il sistema, ha successo con probabilità al più $\varepsilon(n)$, trascurabile in n.*

Dalla definizione precedente, ovviamente, il crittosistema risulta sicuro solo per valori di n "abbastanza" grandi; si supponga, ad esempio, che un dato crittosistema sia computazionalmente sicuro, dove nello specifico un attaccante eseguibile in n^4 minuti è in grado di violare il sistema con probabilità $2^{20} \cdot 2^{-n}$; quando $n \leq 20$, si ha che un avversario eseguibile in tempo 20^4 (circa 15 settimane) riesce a violare la sicurezza con probabilità 1 il che è ovviamente inaccettabile; tuttavia, per $n = 300$ un attaccante che provi ad attaccare il sistema per più di 200 anni ha successo solamente con probabilità di circa 2^{-300}.

Per la realizzazione di gran parte degli schemi crittografici alla base della crittografia moderna, si utilizzano le *funzioni unidirezionali*, cosiddette *One-Way Function (OWF)*; intuitivamente una funzione unidirezionale è una funzione facile da calcolare ma difficile da invertire; il primo requisito è semplice da formalizzare in quanto si richiede che la funzione sia calcolabile in un tempo polinomiale; il secondo requisito, invece, poiché si è interessati alla sicurezza computazionale, richiede che nessun avversario PPT possa invertire la funzione con probabilità non trascurabile.

Definizione 31 (One-Way Function, OWF) *Una funzione* $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ *è detta funzione unidirezionale* (t, ε)*-sicura se soddisfa la seguenti condizioni:*

(i) sia facile da calcolare, cioè esiste un algoritmo polinomiale che riceve in ingresso un valore di $x \in \{0,1\}^*$ *e restituisca* $f(x), \forall x \in \{0,1\}^*$ *;*

(ii) sia difficile da invertire, cioè ogni tentativo efficiente per invertire la funzione f *con input casuale ha successo solo con probabilità trascurabile. Formalmente,* $\forall \mathcal{A}$*, con* $\mathcal{A}$ *avversario efficiente in tempo* t*,* $\exists\, \varepsilon(n)$*, con* $\varepsilon(n)$ *funzione efficiente tale che* $\forall x \in \{0,1\}^n$ *,* x *input di lunghezza* n*, si ha che*

$$\mathbb{P}\left[\mathcal{A}(1^n, y)) = x^{'} \,:\, x \xleftarrow{\$} \{0,1\}^n, y = f(x)\,, f(x^{'}) = y\right] \leq \varepsilon(n)\,,$$

dove la probabilità è calcolata sulla randomicità usata da $\mathcal{A}$ *e con la notazione* $x \xleftarrow{\$} \{0,1\}^n$ *si intende che* x *è scelto casualmente sull'insieme nella distribuzione uniforme* $\{0,1\}^n$*.*

Nella definizione si evidenzia che l'algoritmo $\mathcal{A}$, oltre a y, riceve l'input aggiuntivo 1^n; tale input particolare indica una stringa di n copie di 1 e viene chiamato *parametro di sicurezza*; tale parametro garantisce che il tempo di esecuzione dell'algoritmo sia polinomiale in $|x| = n$.

Definizione 32 (One-Way Function: ulteriore definizione) *In altri termini, una One-Way Function è una funzione* $f : X \to Y$ *iniettiva di cui è computazionalmente trattabile il calcolo di* $f(x)$ *per quasi ogni* $x \in X$ *e computazionalmente intrattabile il calcolo di* $f^{-1}(y)\,, \forall\, y \in f(X)$*.*

Intuitivamente, la definizione è equivalente alla seguente affermazione: per una qualche funzione $f : \{0,1\}^* \longrightarrow \{0,1\}^*$*, l'attaccante* $\mathcal{A}$ *conosce* $y = f(x)$ *e deve calcolare un valore* $x^{'}$ *tale che* $f(x^{'}) = f(x)$*, ovvero una pre-immagine di* y*; dato* $y = f(x)$*, dunque, è sufficiente che l'attaccante* $\mathcal{A}$ *trovi una qualsiasi preimmagine* $x^{'}$ *che soddisfi* $f(x^{'}) = y$*, non importa che* $x = x^{'}$*.*

Con riferimento al problema fondamentale della complessità strutturale, *si congettura l'esistenza delle funzioni unidirezionali*; si può dimostrare che l'esistenza delle funzioni unidirezionali implica $P \neq NP$; d'altra parte, invece, *è ancora un problema aperto dimostrare che se* $P \neq NP$ *implica l'esistenza delle funzioni unidirezionali.*

Mediante le nozioni illustrate è *possibile definire alcune famiglie di OWF* facendo uso dei seguenti concetti algebrici: la *moltiplicazioni di interi* ed il *logaritmo discreto.* Con riferimento alla *moltiplicazione di interi*, il seguente teorema dimostrato da Gauss è alla base della *formulazione dell'ipotesi di fattorizzazione* e della conseguente definizione di famiglia di funzioni OWF.

Teorema 3 (Teorema fondamentale dell'aritmetica) *Ciascun numero naturale è esprimibile in uno ed un solo modo come prodotto di primi.*

Definizione 33 (Ipotesi di fattorizzazione) *Sia* $\Pi_n = \{q:\ q \text{ primo e } q < 2^n\}$ *l'insieme dei numeri primi di* $\boldsymbol{n}$ *bit, si definisce ipotesi di fattorizzazione la seguente affermazione:*
$\forall \mathcal{A}$*, con* $\mathcal{A}$ *avversario efficiente,* $\exists\, \varepsilon$*, con* ε *una funzione trascurabile, tale che, posto* N *un numero fattorizzabile nel prodotto di due numeri primi, la probabilità che, presi casualmente in input* $p \in \Pi_n$ *e* $q \in \Pi_n$*,* $\mathcal{A}$ *restituisca* $N = p \cdot q$ *è inferiore a* $\varepsilon(n)$*. Formalizzando:*

$$\mathbb{P}\left[\mathcal{A}(1^n, p, q) = N\,, con\, N = p \cdot q : p \leftarrow \Pi_n,\, q \leftarrow \Pi_n\right] \leq \varepsilon(n)$$

Sebbene esistano algoritmi per fattorizzare un numero composto da due primi, *nessuno di quelli realizzati finora ha un tempo di esecuzione polinomiale*, in particolare per p e q sufficientemente "grandi", *tenuto conto anche che* p *e* q *abbiano lo stesso numero di bit.*

Dall'ipotesi della fattorizzazione ne deriva che il relativo problema è di tipo NP e che le funzioni ad esso associate sono funzioni OWF; tali funzioni, come visto, risultano fondamentali per la costruzione di crittosistemi. Grazie all'ipotesi di fattorizzazione è possibile, infatti, individuare esempi di famiglie di funzioni one-way. Un esempio di famiglia di funzioni one-way è dato dal seguente teorema:

Teorema 4 (Famiglia di funzioni one-way) *Sia* $\Pi_{\frac{i}{2}}$ *l'insieme dei numeri primi di* $\frac{i}{2}$ *bit, con* i *numero pari, definito come* $\Pi_{\frac{i}{2}} = \{q \text{ tali che } q \text{ primo } q < 2^{\frac{i}{2}}\}$*, con* $i \in I = 2\mathbb{N}$*; sia* $D_i = \{(p, q) \text{ tali che } p, q \in \Pi_{\frac{i}{2}}\}$*,* $R_i = \mathbb{N}$ *e*

$$\mathcal{F} = \{f_i : D_i \to \mathbb{N}\}_{i \in I}$$

una famiglia di funzioni dove $f_i(p, q) = p \cdot q$*. Se vale l'ipotesi di fattorizzazione allora* $\mathcal{F}$ *è una famiglia di funzioni one-way.*

Analogamente a quanto fatto per enunciare l'ipotesi di fattorizzazione, si introducono le seguenti nozioni relative alla teoria dei numeri e all'algebra dei gruppi al fine di formalizzare il *l'ipotesi del logaritmo discreto* grazie alla quale è possibile definire altre famiglie di funzioni one-way.

Definizione 34 (Gruppo) *Si dice gruppo una coppia* $< G, * >$*, dove* G *è un insieme (sostegno del gruppo),* $*$ *è un'operazione interna binaria su* G*, cioè una funzione che ad ogni coppia* (a, b) *di elementi di* G *associa uno ed un solo elemento* $c \in G$ *(denotato* $a * b$*), per la quale valgono le seguenti proprietà:*

- *proprietà associativa:* $\forall\, a, b, c \in G,\ (a * b) * c = a * (b * c)$ *;*
- *esistenza dell'elemento neutro:* $\exists\, e \in G : \forall a \in G\,,\ a * e = e * a = a$ *;*
- *esistenza dell'inverso:* $\forall\, a \in G\,, \exists\, \overline{a} \in G : a * \overline{a} = \overline{a} * a = e$ *.*

Definizione 35 (Ordine di un gruppo) *Dato un gruppo* $< G, * >$*, se l'insieme* G *è finito, si dice il gruppo è finito; il numero degli elementi di un gruppo si chiama ordine del gruppo e si indica con* $|G|$ *.*

Definizione 36 (Gruppo abeliano) *Un gruppo $< G, * >$, per cui vale anche la seguente proprietà:*

- *proprietà commutativa:* $\forall a, b \in G, a * b = b * a$

si dice gruppo abeliano.

E' stato utilizzato il simbolo $*$ per indicare l'operazione binaria sul gruppo, sottolineando il fatto che l'operazione può essere qualunque; nel seguito, se non diversamente specificato, si userà la notazione moltiplicativa, per cui

- l'operazione $*$ sarà chiamata *prodotto* e denotata con $\cdot$ (o con il concatenamento dei due operandi);
- l'elemento neutro sarà chiamato *unità del gruppo* e;
- l'*inverso* di a sarà indicato con a^{-1}.

Si osserva che, grazie alla proprietà associativa, il prodotto di più di due elementi di G può essere definito senza l'utilizzo di parentesi e, dunque, possono essere definite le potenze ad esponente intero di un elemento $a \in G$ ponendo:

- $a^n = \overbrace{a \cdot a \cdots a}^{n \text{ volte}}$, se $n > 0$;
- $a^n = e$, se $n = 0$;
- $a^n = \overbrace{a^{-1} \cdot a^{-1} \cdots a^1}^{-n \text{ volte}}$, se $n < 0$;

la proprietà associativa e le proprietà che definiscono gli elementi neutro e l'inverso garantiscono la validità delle usuali proprietà delle potenze:

- $a^n \cdot a^m = a^{n+m}$
- $(a^n)^m = a^{n\,m}$.

Definizione 37 (Sottogruppo) *Sia $H \subseteq G$. Si definisce $< H, \cdot >$ sottogruppo del gruppo $< G, \cdot >$ se è H a sua volta un gruppo rispetto alla stessa operazione $\cdot$ definita su G.*

Definizione 38 (Sottogruppi banali) *I sottoinsiemi $\{e\}$ e G sono sottogruppi di ogni gruppo $< G, \cdot >$ e sono detti sottogruppi banali.*

Definizione 39 (Sottogruppo ciclico) *Sia $< G, \cdot >$ un gruppo e $g \in G$. L'insieme di tutte le potenze di g è un sottogruppo di G detto sottogruppo ciclico generato da $\boldsymbol{g}$ ed indicato in seguito con $< g >$. Formalmente*

$$< g >= \{g^0, g^1, g^2, ...\}\,.$$

Definizione 40 (Gruppo ciclico) *Un gruppo $< G, \cdot >$ si dice ciclico se $\exists\, g \in G$ tale che $G =< g >$. L'elemento g si chiama generatore di G.*

Definizione 41 (Insieme dei generatori di un gruppo ciclico) *Sia $< G, \cdot >$ un gruppo ciclico. Si definisce l'insieme dei generatori del gruppo G il seguente insieme:*

$$Gen_G = \{g \in G \text{ tale che } < g >= G\}\,.$$

Teorema 5 *Ogni sottogruppo di un gruppo ciclico è ciclico.*

Definizione 42 (Periodo o ordine di un elemento) *Sia $< G, \cdot >$ un gruppo e $g \in G$. Si definisce periodo (o ordine) di g*

- *se esiste, il minimo intero positivo m tale che $g^m = e$ e si indica con $|g|$; in tal caso si dice che l'elemento g è periodico ed ha periodo finito;*
- *se tale m non esiste, cioè se non esiste alcun $n \in \mathbb{N}^+$ tale che $g^n = e$, si dice che g ha periodo 0 (o infinito).*

Teorema 6 *Sia $< G, \cdot >$ un gruppo e $g \in G$. Valgono le seguenti proprietà:*

- *se $|g| = m \Rightarrow < g >= \{g, g^2, \cdots, g^{m-1}, g^m = e\}$;*
- *Sia $|G| = n$. G è ciclico $\Leftrightarrow \exists\, g \in G$ t.c. $|g| = n$;*
- *Se $|G| = p$, con p primo $\Rightarrow G$ è ciclico e $\forall g \in G$, con $g \neq e$, $g \in Gen_G$.*

Un importante gruppo di interesse per le finalità indicate è il *gruppo moltiplicativo $\mathbb{Z}_p^*$ delle classi di resto modulo p*, con *p numero primo*; infatti

- $(\mathbb{Z}_p^*, \cdot)$ è ciclico;
- $(\mathbb{Z}_p^*, \cdot) = \{1, 2, 3, ..., p-1\}$;
- $|\mathbb{Z}_p^*| = p - 1$;
- $Gen_{\mathbb{Z}_p^*} = \{g \in \mathbb{Z}_p^* : (p-1, g) = 1\}$.

Si osservi, per quanto affermato, che $\mathbb{Z}_p^*$ è *ciclico.* poiché p è un numero *primo.*
A scopo esemplificativo si analizza $\mathbb{Z}_5^*$; dalle seguenti operazioni

- $2^1 \equiv_5 2 \quad 2^2 \equiv_5 4 \quad 2^3 \equiv_5 3 \quad 2^4 \equiv_5 1 \Rightarrow 2 \in Gen_{\mathbb{Z}_5^*}$ essendo $< 2 >= \{1, 2, 3, 4\}$;
- $3^1 \equiv_5 3 \quad 3^2 \equiv_5 4 \quad 3^3 \equiv_5 2 \quad 3^4 \equiv_5 1 \Rightarrow 3 \in Gen_{\mathbb{Z}_5^*}$ essendo $< 3 >= \{1, 2, 3, 4\}$;
- $4^1 \equiv_5 4 \quad 4^2 \equiv_5 1 \quad 4^3 \equiv_5 4 \quad 4^4 \equiv_5 1 \Rightarrow 4 \notin Gen_{\mathbb{Z}_5^*}$; essendo $< 4 >= \{1, 4\}$

risulta che $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$ è ciclico, con $Gen_{\mathbb{Z}_5^*} = \{2, 3\}$.

Si osservi che se $\mathbb{Z}_p^*$ è tale che p non sia primo $\not\Rightarrow \mathbb{Z}_p^*$ sia ciclico.
Scegliendo, ad esempio, $\mathbb{Z}_p^*$, con p pari si ha:

- per $p = 4\,, \mathbb{Z}_4^* = \{1, 3\}$ è ciclico, essendo $Gen_{\mathbb{Z}_4^*} = \{3\}$;
- per $p = 6\,, \mathbb{Z}_6^* = \{1, 5\}$ è ciclico, essendo $Gen_{\mathbb{Z}_6^*} = \{5\}$;
- per $p = 8\,, \mathbb{Z}_8^* = \{1, 3, 5, 7\}$ *non* è ciclico, essendo $Gen_{\mathbb{Z}_8^*} = \emptyset$.

Con le nozioni esposte è possibile, finalmente, introdurre il concetto di *logaritmo discreto* che consente di formulare l'*ipotesi del logaritmo discreto* dal quale deriva un altro problema di tipo NP.

Definizione 43 (Logaritmo discreto in $\mathbb{Z}_p^*$) *Dato un gruppo $\mathbb{Z}_p^*$ con p primo, $\mathbb{Z}_p^*$ gruppo ciclico finito, $|\mathbb{Z}_p^*| = p - 1$, si definisce logaritmo discreto di y, $\forall\, y \in \mathbb{Z}_p^*$, con $y \neq e$, in base $g \in Gen_{\mathbb{Z}_p^*}$, l'intero $x \in \mathbb{Z}_p$ tale che*

$$g^x = y \,(mod\, p)$$

ovvero, con la notazione del logaritmo discreto,

$$x = log_g y \,(mod\, p)\,.$$

L'attributo discreto deriva dal fatto che x può assumere valori in un insieme finito.

Consideriamo, ad esempio, $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$. Si è visto che essendo $p = 5$ primo, il gruppo è ciclico. Il logaritmo discreto di $4 \in \mathbb{Z}_5^*$, in base $3 \in Gen_{\mathbb{Z}_5^*}$, modulo 5, è dato da:

$$x = log_3\, 4 \,(mod\, 5) = 2$$

essendo $3^2 \equiv_5 4$.

Teorema 7 (Piccolo teorema di Fermat) *Sia p un primo. Allora $\forall\, a \in \mathbb{Z}_p^*$ si ha $a^{p-1} \equiv 1 \,(mod\, p)$.*

Dal teorema precedente deriva che è possibile sempre aggiungere multipli di $p - 1$ all'esponente, ovvero:

$$y = g^x \equiv g^{x + k\,(p-1)} (mod\, p)\,, \quad \forall\, k \in \mathbb{Z}.$$

Segue che l'esponente $x = log_g(y)$ è sempre un elemento in $\mathbb{Z}_{p-1}$.

Il logaritmo discreto soddisfa le stesse proprietà del logaritmo canonico sui reali; in particolare se $y_1 = g^{x_1}$ e $y_2 = g^{x_2}$, si ha $y_1 \cdot y_2 = g^{x_1 + x_2}$ e quindi

$$log_g(y_1 \cdot y_2) = x_1 + x_2 = log_g(y_1) + log_g(y_2).$$

Siano $h, g \in Gen_{\mathbb{Z}_p^*}$ con $h \neq g$. Allora $g = h^c$ per qualche $c \in \mathbb{Z}_p^*$. Dato un elemento $y = g^x = h^{c\,x}$, si ha:

$$log_h(y) = c\,x = c\, log_g(y) \Rightarrow \frac{log_h(y)}{log_g(y)} = c \in \mathbb{Z}_p^*,$$

ovvero è possibile cambiare la base del logaritmo da g ad h moltiplicando per una costante c che dipende solo da g ed h.

Dati $g \in Gen_{\mathbb{Z}_p^*}$ e $x \in \mathbb{Z}_p^*$, con p *primo*, calcolare $g^x = y\,(mod\,p)$ è computazionalmente facile; inoltre $|\mathbb{Z}_p^*| = p - 1$ risulta pari; pertanto al fine di aumentare la difficoltà computazionale risulta fondamentale utilizzare un *sottogruppo* di ordine *primo* di $\mathbb{Z}_p^*$. Si considera, quindi, $p = r \cdot q + 1$ con p, q **primi** e si utilizza il seguente teorema per provare che $\mathbb{Z}_p^*$ ha un sottogruppo G di ordine q.

Teorema 8 (Sottogruppi di $\mathbb{Z}_p^*$ di ordine q primo) *Sia $p = r \cdot q + 1$, con p, q primi. Allora*

$$G = \{g \in \mathbb{Z}_p^* : [\,g^r (mod\,p)] \in \mathbb{Z}_p^*,\, p = r \cdot q + 1,\, p \text{ e } q \text{ primi } \}$$

è un sottogruppo di $\mathbb{Z}_p^*$ di ordine q.

Per $r = 2$ il gruppo definito nel teorema precedente diventa l'insieme dei "quadrati" modulo p di $\mathbb{Z}_p^*$ ovvero

$$G_q = \{g \in \mathbb{Z}_p^* : [\,g^2 (mod\,p)] \in \mathbb{Z}_p^*,\, p = 2q + 1,\, p \text{ e } q \text{ primi } \}.$$

Come esempio banale si consideri $\mathbb{Z}_p^*$ con $p = 11 = 2 \cdot 5 + 1$ e $q = 5$ primi; ne deriva che

$$G_q = \{4, 9\} \text{ è un sottogruppo di } \mathbb{Z}_{11}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

e che G_q è ciclico essendo $< 9 >= G_q$ dato che $9^1 = 9$ e $9^2 \equiv 4$.

Definizione 44 (Numero primo di Sophie Germain) *Un numero primo p tale che anche $q = 2p + 1$ è primo si dice numero primo di Sophie Germain.*

I numeri primi di Sophie Germain assumono la denominazione di "*sicuri*" in quanto maggiormente resistenti ad un attacco al *protocollo di Diffie-Helmann* che sarà trattato più avanti. Alcuni esempi di numeri primi di Sophie Germain sono i seguenti:

$$2, 3, 5, 11, 23, 29, 41, 53, 89, 113, 173, 178, 191, \cdots$$

Nel febbraio 2016 attraverso il progetto di calcolo distribuito PrimeGrid è stato scoperto il più grande numero di Sophie Germain conosciuto: il numero di 388342 cifre decimali $2618163402417 \cdot 2^{1290000} - 1$ [1].

Per la formulazione dell'ipotesi di logaritmo discreto si fa dunque uso del gruppo $\mathbb{Z}_q^*$, con q *primo* e del sottogruppo G_q di ordine q di $\mathbb{Z}_q^*$, *scelto come l'insieme dei "quadrati" modulo p* di $\mathbb{Z}_p^*$, con p anch'esso *primo*.

[1] `https://it.wikipedia.org/wiki/Numero_primo_di_Sophie_Germain`

Dunque nell'ipotesi in cui:

- $g \in Gen_{\mathbb{Z}_q^*}$;
- $y \in \mathbb{Z}_q^*$;
- $G_q = \{g \in \mathbb{Z}_p^* : [\, g^2 (mod\, p)] \in \mathbb{Z}_p^*,\, p = 2q+1,\, p \text{ e } q \text{ primi } \}$;

determinare $x = log_g y \ (mod\, n)$ è "difficile", poiché dati g e y il numero di tentativi da effettuare per trovare $g^x = y \, (mod\, q)$, calcolando g^x, $\forall\, x \in \mathbb{Z}_q^*$, sarebbe enorme scegliendo un q "grande".

Analogamente a quanto fatto per l'ipotesi di fattorizzazione, con le nozioni introdotte, è possibile finalmente formalizzare l'*ipotesi del logaritmo discreto* e definire un'ulteriore famiglia di funzioni one-way.

Definizione 45 (Ipotesi del logaritmo discreto) *Sia G_q un gruppo ciclico finito di ordine q definito come*

$$G_q = \{g \in \mathbb{Z}_p^* : [\, g^2 (mod\, p)] \in \mathbb{Z}_p^*\,,\, p = 2q+1,\, p \text{ e } q \text{ primi } \},$$

con $q \in \Pi_n = \{q$ tali che q primo e $q < 2^n\}$ l'insieme dei numeri primi di n bit; si definisce ipotesi di logaritmo discreto la seguente affermazione:
$\forall \mathcal{A}$, con $\mathcal{A}$ avversario efficiente, $\exists\, \varepsilon$, con ε una funzione trascurabile, tale che, presi in input $G_q\,,\, q \in \Pi_n\,,\, g \in Gen_{G_q}$ e $x \in G_q$, la probabilità che $\mathcal{A}$ restituisca $y = g^x$ è inferiore a $\varepsilon(n)$. Formalizzando:

$$\mathbb{P}\,[\mathcal{A}(1^n, G_q, q, g, x) = y,\ con\ y = g^x\ : x \xleftarrow{\$} G_q\,, q \leftarrow \Pi_n,\, g \leftarrow Gen_{G_q}] \leq \varepsilon(n)$$

Come anticipato, dall'ipotesi del logaritmo discreto deriva che *il problema del logaritmo discreto è di tipo NP*; dunque, anche tale ipotesi consente di individuare famiglie di funzioni one-way che risultano fondamentali per la costruzione di crittosistemi.

Un esempio di famiglia di funzioni one-way è dato dal seguente teorema.

Teorema 9 (famiglia di funzioni one-way) *Sia $\Pi_n = \{q : q$ primo e $q < 2^n\}$ l'insieme dei numeri primi di n bit e G_q definito come l'insieme dei "quadrati" modulo p di $\mathbb{Z}_p^*$, sottogruppo ciclico di ordine q di $\mathbb{Z}_p^*$, con $p = 2q+1$ primo, ovvero*

$$G_q = \{g \in \mathbb{Z}_p^* : [\, g^2 (mod\, p)] \in \mathbb{Z}_p^*,\, p = 2q+1,\, p \text{ e } q \text{ primi } \}.$$

Sia $I = \{(q, g) : q \in \Pi_n$ e $g \in Gen_{G_q}\}$, $D_i = \{x : x \in \mathbb{Z}_q^*\}, R_i = G_q$ e

$$\mathcal{DL} = \{f_i : D_i \to R_i\,\}_{i \in I}$$

una famiglia di funzioni dove $f_i(x) = f_{q,g}(x) = g^x$.
Se vale l'ipotesi del logaritmo discreto allora $\mathcal{DL}$ è una famiglia di funzioni one-way.

Un ulteriore problema computazionale difficile, che sarà trattato nell'illustrazione del protocollo denominato *Diffie-Hellman key exchange*, è il cosidetto *problema computazionale di Diffie-Helmann (CDH)*; utilizzando le precedenti definizioni, tale problema richiede di calcolare $g^{a\,b} \in G_q$, noti $g^a, g^b \xleftarrow{\$} G_q$; di seguito si riporta la definizione formale dell'ipotesi computazionale da cui deriva.

Definizione 46 (Ipotesi computazionale di Diffie-Helmann) *Sia G_q un gruppo ciclico finito di ordine q definito come*

$$G_q = \{g \in \mathbb{Z}_p^* : [\, g^2 (mod\, p)] \in \mathbb{Z}_p^*\,,\ p = 2q + 1,\ p\ e\ q\ primi\ \},$$

*con $q \in \Pi_n = \{q : q\ primo\ e\ q < 2^n\}$ l'insieme dei numeri **primi di n bit**; si definisce ipotesi computazionale di Diffie-Helmann (CDH) la seguente affermazione:*
$\forall \mathcal{A}$, con $\mathcal{A}$ avversario efficiente, $\exists\, \varepsilon$, con ε una funzione trascurabile, tale che, presi in input $G_q\,,\ q \in \Pi_n\,,\ g^a, g^b \in G_q$, la probabilità che $\mathcal{A}$ restituisca $y = g^{a\,b}$ è inferiore a $\varepsilon(n)$. Formalizzando:

$$\mathbb{P}\,[\mathcal{A}(1^n, G_q, q, g^a, g^b) = y,\ con\ y = g^{a\,b}\ : g^a, g^b \xleftarrow{\$} G_q] \leq \varepsilon(n)$$

Si osserva che se la risoluzione del problema del logaritmo discreto è "facile", scegliendo un gruppo G ciclico di ordine q, allora risulta "facile" anche il problema CDH; infatti dati $\alpha = g^a$ e $\beta = g^b$ è sufficiente calcolare $a = log_g \alpha$ e la soluzione al problema CDH è semplicemente $\beta = g^{a\,b}$. D'altra parte, *non è noto se la difficoltà del problema del logaritmo discreto implichi quella del problema CDH.*

Si conclude la trattazione dei concetti matematici alla base dei sistemi crittografici con l'illustrazione dell'*ipotesi computazionale della residuosità quadratica* che sarà affrontato nell'illustrazione dei protocolli *challenge-response* a *conoscenza zero* (in particolare nei protocolli di *Fiat-Shamir* e di *Feige-Fiat-Shamir*); analogamente a quanto visto per l'ipotesi di fattorizzazione, del logaritmo discreto e del CDH, sarà definita sempre in termini di probabilità.

Definizione 47 (Residuo quadratico) *Dato un gruppo $< G, \cdot >$ moltiplicativo e abeliano, un elemento $g \in G$ si dice residuo quadratico se $g \in Im(f)$ con $f : G \longrightarrow G$ definita da*

$$f(g) = g \cdot g.$$

Definizione 48 (Insieme dei residui quadratici) *Dato un gruppo $< G, \cdot >$ moltiplicativo e abeliano, $g \in G$ residuo quadratico definito da $f(g) = g \cdot g$, si definisce*

$$QR = \{g \in G\ :\ g \in Im(f)\,,\ dove f(g) = g \cdot g\} = Im f.$$

l'insieme dei residui quadratici.

L'applicazione f della definizione precedente è un'*automorfismo* di G, ossia mappa da G in G, compatibile con l'operazione di gruppo, ovvero $\forall\, g\,, h\, \in G$,

$$f(g \cdot h) = f(g) \cdot f(h),$$

da cui segue che l'insieme QR è un sottogruppo di G.

Teorema 10 *Se G è un gruppo di ordine dispari, allora QR coincide con G. Se G è un gruppo ciclico di ordine pari, allora, $\forall g \in Gen_G$, si ha $QR =< g^2 >$.*

Dalla definizione di residuo quadratico deriva la seguente definizione, utilizzando il gruppo moltiplicativo $\mathbb{Z}_p^*$ delle classi resto modulo p.

Definizione 49 (Residuo quadratico modulo p) *Dato $\mathbb{Z}_p^*$, un elemento $a \in \mathbb{Z}_p^*$ si dice residuo quadratico modulo p se $\exists\, x \in \mathbb{Z}_p^*$ tale che*

$$x^2 \equiv a \,mod\, p \text{ (ovvero } x^2 \equiv_p a, \text{ in notazione sintetica)}$$

In caso contrario si dice che a è un ***non residuo modulo p****.*

Definizione 50 (Insieme dei residui quadratici modulo p) *L'insieme dei residui quadratici modulo p si indica con $\mathbb{QR}_p$ ed è definito come segue.*

$$\mathbb{QR}_p = \{a \in \mathbb{Z}_p^* : x^2 \equiv a \,mod\, p, \text{ per qualche } x \in \mathbb{Z}_p^*\}.$$

Definizione 51 (Insieme dei non residui quadratici modulo p) *L'insieme dei non residui quadratici modulo p si indica con $\mathbb{QNR}_p$ ed è definito come segue.*

$$\mathbb{QNR}_p = \{a \in \mathbb{Z}_p^* : a \text{ non è un residuo quadratico } mod\, p\}.$$

Teorema 11 *Sia p un numero primo. Allora*

i. $\mathbb{QR}_p = \{a \in \mathbb{Z}_p^* : x^2 \equiv a \,mod\, p, \text{ con } 0 < x \leq \frac{p-1}{2}\}$;

ii. esistono esattamente $\frac{p-1}{2}$ residui quadratici e $\frac{p-1}{2}$ non residui quadratici $mod\, p$, cioè $\mathbb{Z}_p^*$ è partizionato nei due sottoinsiemi $\mathbb{QR}_p$ e $\mathbb{QNR}_p$, ovvero $\mathbb{QNR}_p = \mathbb{Z}_p^* \setminus \mathbb{QR}_p$. con $|\mathbb{QR}_p| = \frac{p-1}{2} = |\mathbb{QNR}_p|$.

Il seguente teorema fornisce un criterio potente per verificare l'appartenenza di un elemento $a \in \mathbb{Z}_p^*$ in $\mathbb{QR}_p$ o in $\mathbb{QNR}_p$.

Teorema 12 (Criterio di Eulero) *Sia p primo e $a \in \mathbb{Z}_p^*$.*
$a \in \mathbb{QR}_p \Leftrightarrow a^{\frac{p-1}{2}} \equiv_p 1$
$a \in \mathbb{QNR}_p \Leftrightarrow a^{\frac{p-1}{2}} \equiv_p -1$

A scopo esemplificativo si riporta il seguente esempio relativo ai residui quadratici in $\mathbb{Z}_p^*$, con p primo. Si consideri $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$ che ha come insieme dei generatori $Gen_{\mathbb{Z}_5} = \{2, 3\}$. Applicando il criterio di Eulero si ha:

per $a = 1$, $1^{\frac{p-1}{2}} = 1^{\frac{5-1}{2}} = 1^2 \equiv_5 1 \Rightarrow 1 \in \mathbb{QR}_5$

per $a = 2$, $2^{\frac{p-1}{2}} = 2^{\frac{5-1}{2}} = 2^2 \equiv_5 -1 \Rightarrow 2 \in \mathbb{QNR}_5$
per $a = 3$, $3^{\frac{p-1}{2}} = 3^{\frac{5-1}{2}} = 3^2 \equiv_5 -1 \Rightarrow 3 \in \mathbb{QNR}_5$
per $a = 4$, $4^{\frac{p-1}{2}} = 4^{\frac{5-1}{2}} = 4^2 \equiv_5 1 \Rightarrow 4 \in \mathbb{QR}_5$

da cui deriva che $\mathbb{QR}_5 = \{1, 4\}$ e $\mathbb{QNR}_5 = \{2, 3\}$. Si osservi che, per un teorema precedente, essendo $|\mathbb{Z}_5| = 4$ pari, risulta verificato che $\forall\, g \in Gen_{\mathbb{Z}_5}$, $< g^2 >= \mathbb{QR}_p$ e che $\mathbb{QR}_5 \cup \mathbb{QNR}_5 = \mathbb{Z}_5^*$ e $\mathbb{QR}_5 \cap \mathbb{QNR}_5 = \emptyset$.

Dal criterio di Eulero deriva la seguente osservazione.

Sia p primo e $a \in \mathbb{Z}_p^*$; se $a, b \in \mathbb{QR}_p \Rightarrow a \cdot b \in \mathbb{QR}_p$; poiché l'elemento neutro del prodotto è $1 \in \mathbb{QR}_p$ allora l'insieme dei residui quadratici modulo p è un sottogruppo di $(\mathbb{Z}_p^*, \cdot)$.

Le nozioni esposte consentono, finalmente, di introdurre il *problema della residuosità quadratica*, *QRP, modulo p, con p primo*, la cui successiva estensione per $N = p \cdot q$, con p e q primi dispari distinti, è oggetto di interesse dal punto di vista della crittografia perché consente la *formulazione dell'ipotesi computazionale della residuosità quadratica.* Il QRP è un problema "difficile", ben noto nella teoria dei numeri, ed è uno dei problemi algoritmici discussi da Gauss nelle sue *Disquisitiones Arithmeticae*[2]; una sua soluzione efficiente implicherebbe una soluzione efficiente per altri problemi aperti nella teoria dei numeri!

Definizione 52 (Problema della residuosità quadratica modulo p) *Sia p un numero primo e $a \in \mathbb{Z}_p^*$. Per problema della residuosità quadratica modulo p, QRP, si intende stabilire se $a \in \mathbb{QR}_p$.*

Definizione 53 (Problema della residuosità quadratica modulo N) *Sia $N = p \cdot q$, con p e q primi dispari distinti e $x \in \mathbb{Z}_N^*$. Per problema della residuosità quadratica modulo N, QRP, si intende stabilire se $x \in \mathbb{QR}_N$.*

Il seguente teorema fornisce un metodo per la verifica, nota la fattorizzazione di N.

Teorema 13 *Sia N un intero fattorizzabile in $N = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$, con p_i primo per $i = 1, \cdots, n$. Allora $x \in \mathbb{QR}_N \Leftrightarrow x \in \mathbb{QR}_{p_i}$, con p_i primo per $i = 1, \cdots, n$.*

Se la fattorizzazione di N è nota, la residuosità quadratica di x modulo N, può essere stabilita verificando la residuosità di x modulo p_i per ogni primo p_i mediante il criterio di Eulero; tuttavia, se la fattorizzazione di N non è nota, determinare la residuosità quadratica modulo N non è un compito banale.

La versione computazionale del problema della residuosità quadratica, come formalizzato con l'ipotesi seguente, dati $N = p \cdot q$, con p e q primi distinti, ed un elemento $x \in \mathbb{Z}_N^*$, consiste nel calcolare (se esiste) una radice quadrata di x.

[2] `https://it.wikipedia.org/wiki/Disquisitiones_Arithmeticae`

Definizione 54 (Ipotesi computazionale della residuosità quadratica) *Siano $N = p \cdot q$, con p e q primi dispari distinti, $x \in \mathbb{Z}_N^*$; si definisce ipotesi computazionale della residuosità quadratica la seguente affermazione:*
$\forall \mathcal{A}$ avversario efficiente, scelti casualmente due numeri primi dispari distinti p, q e posto $N = p \cdot q$, con $|p| = |q| \approx n$ bit, $\exists\, \varepsilon(n)$, funzione trascurabile, tale che la probabilità che $\mathcal{A}$, presi in input N ed un x tra i residui quadratici modulo N, restituisca una y con $y^2 \equiv x \bmod N$, è inferiore a $\varepsilon(n)$.
Formalizzando, si dice che il problema della residuosità quadratica è $(\boldsymbol{t}, \boldsymbol{\varepsilon})$-difficile in $\mathbb{Z}_N^*$, dove $N = p \cdot q$, con $|p| = |q| \approx n$, con p e q primi dispari distinti, se $\forall$ avversario PPT $\mathcal{A}$, eseguibile in tempo t, $\exists\, \varepsilon(n)$ per cui risulta:

$$\mathbb{P}\left[\mathcal{A}(1^n, N, x) = y,\ con\ y^2 \equiv x \bmod N :\ x \xleftarrow{\$} \mathbb{QR}_N,\ \}\right] \leq \varepsilon(n)$$

Il teorema seguente garantisce che risolvere il problema computazionale della residuosità quadratica è del tutto equivalente a fattorizzare N.

Teorema 14 (Equivalenza tra fattorizzazione e QRP computazionale) *Se il problema QRP computazionale è $(t, 2\varepsilon)$-difficile, non è possibile fattorizzare N in tempo $O(t)$ e con probabilità migliore di ε.*

Dall'equivalenza computazionale tra il problema della fattorizzazione e quello della residuosità quadratica deriva che, con l'ipotesi indicata, *il problema della residuosità quadratica modulo N è di tipo NP* e, quindi, utilizzabile al fine degli aspetti crittografici che interessano la sicurezza dei protocolli di autenticazione.

3.1.3 Protocolli di autenticazione

Definizione 55 (Protocollo di comunicazione) *Un protocollo di comunicazione è un algoritmo che comporta uno scambio di messaggi tra due o più parti, dette principal; esso è definito da una sequenza di passi che specificano in modo univoco le azioni previste dai principal per realizzare determinati obiettivi.*

Nel contesto dei protocolli di comunicazione, *l'obiettivo primario di sicurezza è l'autenticazione dei principal*; tale obiettivo è alla base di tutti gli altri eventuali servizi di sicurezza offerti da un protocollo.

Definizione 56 (Autenticazione) *L'autenticazione è il processo mediante il quale una entità stabilisce la proprietà richiesta da un'altra entità; in altre parole l'autenticazione è l'atto di confermare la verità di un attributo di una singola parte di dato o di una informazione sostenuto vero da un'entità.*

Le entità di un sistema distribuito dimostrano la loro *identità* attraverso il *processo di autenticazione.*

Si deve tener presente che l'*autenticazione* è differente dall'*identificazione* ovvero *la determinazione che una determinata entità sia conosciuta o meno dal sistema* e

dall'*autorizzazione* ovvero *il conferimento del diritto ad accedere ad una specifica risorsa sulla base della sua identità.*

Nella trattazione si vedrà che il processo di autenticazione viene realizzato facendo in modo che ogni partecipante alla comunicazione condivida un *segreto* con un'entità fidata detta *authentication server* o *Trusted Third Party (TTP)*; un'entità riesce a provare la propria identità dimostrando di possedere tale segreto.

Definizione 57 (Protocollo di autenticazione) *Un protocollo di autenticazione è un tipo di protocollo di comunicazione o un protocollo crittografico progettato specificamente per il trasferimento di dati di autenticazione tra due entità.*

Con riferimento ai protocolli di autenticazione, si possono identificare alcune possibili definizioni in accordo con il concetto di *agreement* sviluppato da Lowe: *i requisiti appropriati per i protocolli che realizzano l'autenticazione dipendono dagli scopi precisi di tali protocolli.*
Di seguito si riportano alcune definizioni utili per la caratterizzazione dei protocolli di autenticazione.

Definizione 58 (Aliveness) *Sia P un protocollo di autenticazione, A l'initiator e B un'altra entità. Una volta che A ha effettuato un'esecuzione di P, apparentemente con B, si dice che il protocollo P garantisce ad A la "aliveness" di B se B ha effettivamente eseguito una esecuzione di P.*

Definizione 59 (Weak agreement) *Sia P un protocollo di autenticazione, A l'initiator e B un'altra entità. Si dice che il protocollo P garantisce all'initiator A la "weak agreement" con B se, ogni volta che A completa un'esecuzione di P, apparentemente con B, B ha effettivamente eseguito una esecuzione di P, apparentemente con A.*

Definizione 60 (Non-injective agreement) *Sia P un protocollo di autenticazione, A l'initiator, B un'altra entità e D un insieme di dati. Si dice che il protocollo P garantisce all'initiator A un non-injective agreement con il responder B su D se, ogni volta che A completa un'esecuzione di P, apparentemente con B, allora:*

- *B ha precedentemente eseguito P, apparentemente con A;*
- *B è un responder di tale esecuzione;*
- *le due entità sono d'accordo nei valori di tutte le variabili di B.*

Dalle condizioni indicate nella definizione precedente, risulta evidente che la *non-injective agreement* di un protocollo *non garantisce che ci sia una corrispondenza biunivoca tra le esecuzioni di A e quelle di B.*

Definizione 61 (Injective agreement o full agreement) *Sia P un protocollo di autenticazione, A l'initiator, B un'altra entità e D un insieme di dati. Si dice che il protocollo P garantisce all'initiator A injective agreement o full agreement con un responder B su un insieme di dati D se, ogni volta che A completa un'esecuzione di P, apparentemente con B, allora:*

- *B ha precedentemente eseguito P, apparentemente con A;*
- *B è un responder di tale esecuzione;*
- *le due entità sono d'accordo nei valori di tutte le variabili di B.*
- *ogni suddetta esecuzione di A corrisponde univocamente ad una esecuzione di B.*

Ognuna della precedenti definizioni a partire dalla *aliveness* incorpora la successiva; la *full agreement* viene considerata la più utile poiché le due entità sono d'accordo su tutti i principali aspetti dell'esecuzione del protocollo.

Definizione 62 (Protocollo di mutua autenticazione) *Si definisce protocollo di mutua autenticazione un protocollo tramite il quale ciascuno dei principal nella comunicazione verifica l'identità dell'altro.*

L'obiettivo primario di un protocollo di mutua autenticazione è stabilire l'identità delle parti che partecipano al protocollo, nel senso che *ciascun principal deve svolgere sia il ruolo di soggetto autenticante che di soggetto autenticato*; per tale motivo la mutua interazione risulta più idonea dell'autenticazione unidirezionale per molte applicazioni distribuite; *un obiettivo secondario è la condivisione di una chiave segreta per ulteriori comunicazioni tra i principal*; tale chiave segreta può essere usata per l'autenticità e la segretezza di future comunicazioni. Seguendo l'approccio di Lowe, le due proprietà costituirebbero gli obiettivi da prendere in considerazione per la validazione della correttezza di un protocollo.

Contrariamente a quanto sembrerebbe suggerire l'intuizione, *la verifica di un protocollo di mutua autenticazione non è riconducibile a quella di due protocolli di autenticazione unidirezionali*; in altri termini, per garantire una valida mutua autenticazione tra due principal A e B non basta validare separatamente l'autenticazione di A nei confronti di B e di B nei confronti di A.
La verifica di un protocollo di autenticazione è inoltre influenzata da svariati fattori riconducibili, da un lato, ai principi guida impiegati nella stesura dello stesso protocollo e, dall'altro, al tipo di ragionamenti operativi formali utilizzati per la verifica.

I protocolli crittografici si caratterizzano anche per ulteriori proprietà espresse con le seguenti definizioni.

Definizione 63 (Riservatezza) *Sia P un protocollo di autenticazione, m un messaggio ed S un insieme di entità che partecipano al protocollo. Si dice che P gode della proprietà di riservatezza rispetto al messaggio m ed all'insieme S se nell'ipotesi che il messaggio m sia noto al più a tutte le entità appartenenti ad S prima dell'esecuzione di P, tale ipotesi risulta verificata anche dopo l'esecuzione di P.*

Definizione 64 (Autenticità) *Sia P un protocollo di autenticazione, m un messaggio ed A un'entità che partecipa al protocollo. Si dice che P offre all'entità A garanzia di autenticità rispetto al messaggio m se A è in grado di determinare chi ha generato il messaggio.*

Definizione 65 (Unicità) *Sia P un protocollo di autenticazione, m un messaggio composto del tipo $m = \{m_1, m_2, ..., m_n\}$ e I un insieme di indici tale che $I \subset \{1, \cdots, n\}$. Si dice che P gode della proprietà di unicità rispetto al messaggio S e all'insieme I se, per ogni messaggio composto $m^{'}$, con $m^{'} = \{m_1^{'}, m_2^{'}, \cdots, m_n^{'}\}$, che viene spedito nella rete durante lo svolgimento di una sessione di P, vale il seguente predicato:*

$$(i \in I \Rightarrow m_i = m_i^{'}) \Rightarrow (i \in \{1, \cdots, n\} \Rightarrow m_i = m_i^{'})$$

In altri termini, le componenti indicate dall'insieme I caratterizzano univocamente il messaggio m.
Tale insieme I è di solito costituito da un unico indice che rappresenta un *messaggio atomico*, in grado da solo di identificare il messaggio composto; un requisito di questo genere viene infatti utilizzato per stabilire una corrispondenza tra un messaggio inviato durante una sessione del protocollo e la sessione stessa.

Definizione 66 (Non ripudiabilità) *Sia P un protocollo di autenticazione ed A un'entità che partecipa al protocollo. Si dice che P fornisce garanzia di non ripudiabilità ad A se assicura alla suddetta entità i mezzi per dimostrare che una certa azione è avvenuta.*

Si supponga, ad esempio, di esaminare un protocollo P di autenticazione nella cui esecuzione un'entità A invia un messaggio m ad un'entità B; una *proprietà di non repudabilità* di P potrebbe essere: "*A riesce a dimostrare la ricezione del messaggio m da parte di B*".

Definizione 67 (Imparzialità) *Sia P un protocollo di autenticazione ed A e B due entità che partecipano al protocollo. Si dice che P gode della proprietà di imparzialità se per ogni garanzia G che permetta ad A di dimostrare che B ha eseguito un'azione, viene fornita a B una garanzia complementare.*

Con riferimento all'esempio precedente, in cui la *proprietà di non repudabilità* di P è "*A riesce a dimostrare la ricezione del messaggio m da parte di B*", la proprietà complementare risulterebbe: "*B riesce a dimostrare la spedizione del messaggio m da parte di A*".

I protocolli crittografici sono classificati in base agli obiettivi di sicurezza.

Definizione 68 (Protocolli di prima generazione) *Si definiscono protocolli di prima generazione i protocolli che hanno come unici obiettivi di sicurezza l'autenticazione del principal ed, eventualmente, la riservatezza rispetto al messaggio.*

Definizione 69 (Protocolli di seconda generazione) *Si definiscono protocolli di seconda generazione i protocolli che si basano sull'impiego di protocolli di prima generazione per conseguire ulteriori obiettivi secondari di sicurezza, quali l'anonimato, il non ripudio e la consegna certificata.*

Con riferimento all'autenticazione, risulta fondamentale fornire la nozione di *data origin authentication*, chiamato anche *message authentication*, associata al concetto di *data integrity*.

Definizione 70 (Data integrity) *Si definisce data integrity il processo di verifica dell'integrità di un dato o di un informazione durante una comunicazione o una memorizzazione.*

Il processo di *data integrity* è finalizzato al controllo del contenuto delle informazioni, verificando, in particolare, se nel corso della comunicazione tra due entità una terza parte ne ha modificato il contenuto; da sottolineare che tale processo, *nel caso di verifica dell'integrità di un messaggio, non richiede l'identificazione del mittente.*

Definizione 71 (Data origin authentication) *Si definisce data origin authentication (o message authentication) il processo di validazione di un messaggio inviato da un mittente ad un ricevente in una trasmissione. Gli obiettivi del processo sono:*

1. *stabilire l'identità del mittente del messaggio;*
2. *stabilire il data integrity del messaggio, dal momento in cui lo stesso è stato inviato;*
3. *stabilire la liveness del mittente del messaggio, ovvero il fatto che il mittente sia on-line disponibile a ricevere messaggi ed a rispondere ad eventuali richieste.*

In realtà nel punto 2. della definizione di *data origin authentication* non solo si cerca di stabilire l'integrità del messaggio ma se ne determina anche la *"freshness"* di cui si riportano le seguenti definizioni.

Definizione 72 (Freshness) *Dal punto di vista di un partecipante nell'esecuzione di un protocollo, si definisce freshness un identificatore di "freschezza" (freshness identifier) o un messaggio confermato come nuovo per una particolare esecuzione del protocollo.*

Definizione 73 (Trusted freshness) *Un identificatore di freschezza attendibile, chiamato anche trusted freshness, è un identificatore la cui freshness è stata confermata da una fonte attendibile.*

Definizione 74 (Fresh message) *Dal punto di vista di un partecipante nell'esecuzione di un protocollo, un fresh message è un messaggio ricevuto o inviato che include una trusted freshness.*

Informalmente, nell'esecuzione di un protocollo si può considerare un messaggio "*fresco*" se l'intervallo di tempo che intercorre, dal momento in cui il mittente lo ha spedito a quello in cui il ricevente lo ha effettivamente ricevuto, è "*relativamente breve*".

La richiesta di ricevere *messaggi "freschi"* presuppone che esista un adeguata linea di comunicazione tra i partecipanti alla conversazione ed implica, inoltre, che esista un probabilità ridotta del verificarsi di un alterazione del messaggio, compreso il fatto che uno dei partecipanti possa essere ingannato da un attaccante. L'*ordine di grandezza* ed il suo *grado di tolleranza* vengono determinati dalle applicazioni perché ognuna ha un ordine di grandezza appropriato.

Definizione 75 (Nonce) *In crittografia il termine nonce indica un numero, generalmente casuale o pseudo-casuale, che ha un utilizzo unico; deriva infatti dall'espressione inglese "for the nonce" che significa appunto "per l'occasione".*

Nei protocolli di autenticazione, l'utilizzo dei *nonce* rende difficoltoso il riuso, anche doloso, di dati scambiati nelle tra i partecipanti nella precedenti comunicazioni.

Si conclude la sezione con l'introduzione delle *funzioni hash*, fondamentali per l'utilizzo delle applicazioni crittografiche necessarie per garantire la sicurezza dei protocolli di autenticazione; a riguardo nella definizione di *funzione crittografica di hash* si fa riferimento alle *funzioni One Way* (OWF) trattate nella sezione relativa alla sicurezza computazionale.

Definizione 76 (Funzione hash) *Nel contesto dei protocolli crittografici, si definisce una funzione hash una funzione $h(x)$ che mappa una stringa di lunghezza arbitraria in una stringa di lunghezza predefinita, ovvero*

$$x \overset{h}{\longmapsto} h(x) \text{ con } x \in \{0,1\}^* \ .$$

con le seguenti proprietà:

- *sia computazionalmente intrattabile la ricerca di una stringa in input x che dia un hash $h(x)$ uguale a un dato hash $\bar{y}$ ovvero tale che $h(x) = \bar{y}$ (resistenza alla preimmagine);*
- *sia computazionalmente intrattabile la ricerca di una stringa in input $x^{'}$ che dia un hash $h(x^{'})$ uguale a quello di una data stringa $\bar{x}$ ovvero tale che $h(x^{'}) = h(\bar{x})$ (resistenza alla seconda preimmagine);*
- *sia computazionalmente intrattabile la ricerca di una coppia di stringhe in input $x, x^{'}$ con $x \neq x^{'}$ che diano lo stesso hash cioè tali che $h(x) = h(x^{'})$(resistenza alle collisioni).*

Definizione 77 (Funzione crittografica di hash o hash OWF) *Si definisce una funzione crittografica di hash (o funzione hash one-way) una classe speciale delle funzioni di hash che dispone delle seguenti proprietà fondamentali che la rendono adatta all'uso nella crittografia:*

- *deve identificare univocamente il messaggio (message digest) ovvero non è possibile che due messaggi differenti, pur essendo simili, abbiano lo stesso valore di hash;*
- *deve essere deterministico, ovvero lo stesso messaggio si deve tradurre sempre nello stesso hash;*
- *deve essere semplice e veloce calcolare un valore hash da un qualunque tipo di dato;*
- *deve essere molto difficile o quasi impossibile generare un messaggio dal suo valore hash se non provando tutte le casistiche possibili.*

Poiché le funzione crittografica di hash è progettata per essere unidirezionale (one-way), l'unico modo per ricreare i dati di input a partire dall'output è quello di tentare, come si vedrà nella sezione successiva relativa gli attacchi, una ricerca di *forza-bruta* dei possibili input per verificare se esiste una corrispondenza.

In base a quanto definito risulta evidente l'esigenza dell'utilizzo di funzioni crittografiche di hash in un protocollo di autenticazione; tra i vantaggi a favore dei partecipanti nell'esecuzione del protocollo si evidenzia, in particolare, la *rilevazione delle eventuali manomissioni che possono subire i messaggi nel corso della comunicazione* e l'*adozione delle conseguenti azioni per preservare la sicurezza dei sistemi secondo gli obiettivi prefissati.*

3.1.4 Scenari di autenticazione in base alla tipologia dei partecipanti

Definizione 78 (Autenticazione dell'entità) *L'autenticazione dell'entità è il processo mediante il quale un'entità (il verificatore) ha la certezza dell'identità di una seconda entità (il richiedente) che sta partecipando a un protocollo.*

Tale garanzia si ottiene generalmente richiedendo al richiedente di fornire al verificatore una prova corroborante dell'identità rivendicata; tale identità può essere presentata al verificatore come parte del protocollo o può essere presunta dal contesto. Talvolta come sinonimo di *autenticazione dell'entità* viene utilizzato il termine *identificazione*; tale termine, inoltre, è anche usato per riferirsi semplicemente al *processo di rivendicazione o dichiarazione di un'identità* senza fornire le prove di conferma richieste per l'autenticazione dell'entità; è necessario, dunque, prestare attenzione nell'utilizzo del termine per garantirne una corretta interpretazione.

Definizione 79 (Credenziali di autenticazione) *Si definisce credenziali o codice di autenticazione l'evidenza corroborante richiesta per ottenere l'autenticazione dell'entità.*

La definizione di autenticazione dell'entità (*entity authentication*), intesa come processo attraverso il quale un partecipante stabilisce una comunicazione attiva (*lively*) con un'altra parte la cui identità rispecchia quella cercata dalla prima, conduce all'analisi dei possibili scenari la cui definizione dipende dalla *tipologia dei partecipanti alla comunicazione.* I *principali scenari in base alla tiplogia dei partecipanti* possono essere classificati come segue:

- *host-host*: i partecipanti alla conversazione sono due nodi di un sistema distribuito; poiché in tale 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; a tale scenario appartengono, ad esempio, i casi di client-server in cui il primo richiede un certo servizio al secondo;

- *user-host*: uno user ottiene l'accesso ad un sistema accedendo ad uno degli host del sistema; nei casi più semplici è sufficiente eseguire un protocollo di autenticazione con l'uso di password, in altri casi più complessi è richiesta un'autenticazione reciproca.

- *club membership*: è una generalizzazione del caso *user-host*, poiché un membro del club per accedere deve dimostrare di possedere le credenziali; in tale scenario, l'altra parte controlla semplicemente che la prima possieda le credenziali necessarie senza preoccuparsi di quale sia effettivamente la sua identità.

Lo scenario di riferimento nel quale avviene l'esecuzione del protocollo utilizzato dall'applicazione sviluppata per il progetto, come sarà illustrato in dettaglio nell'apposita sezione, *rientra fondamentalmente nel caso user-host, dovendo l'utente autorizzato autenticarsi ad una web application mediante l'utilizzo di credenziali* e non essendo prevista un'autenticazione, host-host, del client da parte del web server che eroga il servizio.

3.1.5 Attacchi ai protocolli di autenticazione

Nelle sezioni precedenti si è avuto modo di illustrare l'importanza dell'utilizzo delle tecniche di crittografia nell'esecuzione dei protocolli di autenticazione il cui obiettivo è la verifica della rivendicazione di una determinata proprietà; ne deriva necessariamente che nella trattazione si faccia riferimento alla loro "robustezza" ed ai problemi di sicurezza connessi all'utilizzo, dovuti ai possibili attacchi ai quali possono essere sottoposti; risulta, pertanto, fondamentale definire il concetto di *attacco ad un protocollo di autenticazione* e le *principali tipologie di attacco* che un avversario può eseguire ai danni di un protocollo e, quindi, dei suoi partecipanti.

Definizione 80 (Attacco ad un protocollo di autenticazione) *Si definisce attacco ad un protocollo di autenticazione l'esecuzione di una procedura da parte di uno o più attaccanti ai danni degli utilizzatori del protocollo stesso allo scopo di ottenere qualche risultato non previsto.*

Definizione 81 (Attacco passivo) *Si definisce attacco passivo ad un protocollo di autenticazione un attacco che tenta di leggere o utilizzare le informazioni utilizzate dal protocollo senza influenzare i partecipanti, modificare alcuna informazione o arrecare alcun danno.*

Dalla definizione risulta evidente che in un attacco passivo un avversario si limita ad osservare la trasmissione; pertanto *l'attacco passivo rappresenta una minaccia alla riservatezza*; *non essendo l'entità (vittima) a conoscenza dell'attacco*, l'unica strategia di

sicurezza che può attuare consiste nella prevenzione effettuata anche mediante l'utilizzo di tecniche crittografiche.

Definizione 82 (Attacco attivo) *Si definisce un attacco attivo ad un protocollo di autenticazione un attacco che tenta di influire sul suo funzionamento anche modificandone le modalità di interazione tra i partecipanti.*

A differenza del precedente, *in un attacco attivo l'aggressore altera la trasmissione*; pertanto *l'attacco attivo rappresenta una minaccia all'integrità ed alla disponibilità dei dati*; l'entità (vittima), però, attuando opportune strategie, può venire a conoscenza dell'attacco mediante l'utilizzo di strumenti di rilevazione.

Nel contesto degli attacchi spesso si fa riferimento al concetto di protocollo difettoso. Di seguito si formalizza la definizione.

Definizione 83 (Protocollo difettoso) *Si definisce protocollo difettoso un protocollo in cui le parti arrivano a conclusioni differenti al termine della sua esecuzione.*

Nell'esecuzione del protocollo utilizzato dalla web application sviluppata si noterà che *talvolta l'insicurezza dei protocolli non deriva dal fatto che le tecniche crittografiche utilizzate siano difettose*; un avversario, infatti, può impedire che l'autenticazione abbia successo senza violare necessariamente gli algoritmi crittografici.

Di seguito si definiscono i principali tipi di attacchi ai protocolli che saranno trattati nel corso del progetto; ovviamente l'elenco non è esaustivo ma è sufficiente per poter illustrare le debolezze ed i punti di forza dei protocolli che si analizzeranno.

Definizione 84 (Attacco di sniffing) *Si definisce un attacco di sniffing o eavesdropping o snooping un attacco che consente ad un avversario di intercettare le informazioni che sono inviate nel canale di comunicazione durante l'esecuzione del protocollo.*

In genere *un attacco di sniffing viene mitigato utilizzando opportune tecniche di cifratura nei messaggi*; è considerato un attacco di tipo *passivo* in quanto non è necessario che l'avversario disturbi la conversazione tra le legittime parti.

Definizione 85 (Attacco di modificazione) *Si definisce attacco di modificazione un attacco con il quale l'avversario altera le informazioni inviate nel protocollo.*

Una delle tecniche utilizzate in un attacco di modificazione prevede che il messaggio venga suddiviso in più frammenti e assemblato in un messaggio differente; le tecniche di cifratura non sempre possono prevenire tale tipo di attacco.

Definizione 86 (Attacco di replay) *Si definisce attacco di replay un attacco con il quale un avversario registra dapprima alcune informazioni ricavate dal protocollo e, successivamente, le rinvia ad una delle due parti della conversazione solitamente utilizzando una differente esecuzione.*

Una strategia di conduzione di tale 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; successivamente l'avversario cerca di impersonare un altro soggetto utilizzando parti dell'esecuzione del protocollo.

Definizione 87 (Attacco di riflessione) *Si definisce attacco di riflessione o reflection un attacco di tipo replay con il quale l'avversario manda indietro l'ultimo messaggio inviato al partecipante che lo aveva inviato in primo luogo.*

Definizione 88 (Attacco di dizionario) *Si definisce attacco di dizionario un attacco con il quale un avversario cerca di violare un protocollo di autenticazione utilizzando un cosiddetto dizionario composto dalle chiavi/password ritenute più probabili.*

In generale un attacco di dizionario risulta efficace nel caso in cui le password scelte dall'utente non siano delle stringhe casuali ma vengano stabilite con un significato specifico in quanto più facili da ricordare; un dizionario, infatti, è composto, tra gli altri, da stringhe che rappresentano le password ottenute da parole di senso compiuto per ciascuna lingua di utilizzo della vittima, date di nascita, di avvenimenti importanti, ecc; *una difesa a tale attacco consiste nell'impostare, nei sistemi coinvolti nell'esecuzione del protocollo, delle adeguate policy di sicurezza che assicurino l'utilizzo di password "robuste"*; nella sezione relativa ai protocolli basati su password si approfondisce tale strategia.

Definizione 89 (Attacco Man In The Middle) *Si definisce attacco Man In The Middle (MITM) un attacco con il quale un avversario si inserisce in una comunicazione 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 sniffing; nel modo attivo, invece, altera i messaggi trasmessi durante la conversazione.*

In generale nel MITM, i partecipanti non si accorgono della presenza di una terza parte ma sono convinti di comunicare tra loro attraverso un canale sicuro o privato.

Definizione 90 (Attacco di forza bruta) *Si definisce attacco di forza bruta un attacco con il quale un avversario tenta l'utilizzo di tutte le possibili chiavi, o password, in maniera esaustiva fino alla scoperta di quella esatta per la violazione del protocollo.*

Definizione 91 (Attacco di chosen-text) *Si definisce attacco di chosen-text un attacco con il quale un avversario avvia l'esecuzione di un protocollo di autenticazione fornendo dei "challenge" creati appositamente allo scopo di ottenere informazioni sul segreto relativo alla vittima. L'attacco chosen-text è anche detto attacco con testo cifrato scelto o attacco con testo cifrato scelto e testo in chiaro in quanto l'attaccante sfrutta la conoscenza sia del testo in chiaro che del testo cifrato che è riuscito ad ottenere dai messaggi inviati dalla vittima.*

L'attacco di chosen-text è un tipo di attacco crittanalitico mediante il quale un avversario raccoglie informazioni su un sistema crittografico scegliendo un *testo cifrato* ed ottenendo la sua *versione decifrata* con una *chiave non nota*; si basa sul fatto che l'attaccante abbia avuto accesso al protocollo o al meccanismo di decifratura in modo da poter così risalire alle versioni in chiaro dei testi cifrati in suo possesso; *lo scopo dell'attacco è l'ottenimento della chiave utilizzata nei sistemi che eseguono il protocollo per poter decifrare tutte le informazioni protette scambiate tra le parti.*

Definizione 92 (Attacco di interleaving) *Si definisce attacco interleaving un attacco che prevede l'utilizzo di informazioni provenienti da una o più esecuzioni precedenti o simultanee del protocollo anche iniziate dall'avversario stesso.*

Definizione 93 (Attacco di ritardo forzato) *Si definisce attacco di ritardo forzato un attacco con il quale, nell'esecuzione del protocollo, un messaggio intercettato da un attaccante viene inoltrato alla corretta destinazione dopo un determinato intervallo di tempo.*

Definizione 94 (Attacco clogging) *Si definisce attacco clogging un attacco di tipo Denial of Service (DoS) in cui l'attaccante manda una serie di messaggi alla sua vittima allo scopo di generare una chiave di sessione; in tal modo il sistema della vittima viene messo in difficoltà, dal punto di vista computazionale, a seguito del calcolo delle chiavi per rispondere a tutte le richieste pervenute tanto da compromettere la corretta esecuzione del protocollo.*

Nelle sezioni relative alle trattazione dei protocolli, per ciascuna tipologia, sono illustrate le contromisure più efficaci per mitigare i rischi dovuti agli attacchi illustrati;
ad esempio, per difendersi da *attacchi di replay* si usano i *nonces* e si inserisce l'*identità* del *target nelle risposte*; per gli *attacchi di interleaving*, oltre ad inserire l'identificatore del target del response, si usano *messaggi di forme diverse* (evitando simmetrie) e *chiavi unidimensionali*; per i *chosen-text* si inseriscono nelle risposte degli *elementi di disturbo*; negli *attacchi di ritardo forzato* si usano *numeri casuali*, *intervalli di time out brevi* e *marche temporali* unite ad altre tecniche.

In generale, si può affermare che *in un protocollo di autenticazione l'adozione di ogni contromisura al fine di mitigare i possibili rischi derivanti dagli attacchi indicati, se da un parte ne rafforza la sicurezza, dall'altra ne accresce la complessità.*

3.1.6 Convenzioni di comportamento dei partecipanti nell'esecuzione dei protocolli di autenticazione

Con riferimento al comportamento dei partecipanti nell'esecuzione dei protocolli di autenticazione che saranno illustrati e di un eventuale avversario che condurrà gli attacchi ai loro danni, è indispensabile stabilire una serie di convenzioni utili che saranno adottate.

Convenzione 1 (Partecipante onesto ad un protocollo) *Un partecipante ad un protocollo di autenticazione si supporrà onesto se e solo se rispetti tutte le seguenti condizioni:*

- *non riesca a comprendere il significato di alcun messaggio del protocollo prima che un'esecuzione dello stesso sia completata;*
- *non riesca a riconoscere un messaggio cifrato da una generica stringa casuale; quindi non possa scomporla né ricrearla a meno di non conoscere la chiave corretta;*
- *non riesca 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 dello stesso protocollo;*
- *non salvi alcun messaggio del protocollo se non specificato dal protocollo stesso.*

Per quanto riguarda l'ultima condizione, si osserva che i protocolli di autenticazione sono definiti *stateless* poiché, ad eccezione del risultato ottenuto, *non viene richiesto di memorizzare alcuna informazione di stato una volta che l'esecuzione è terminata.*

Convenzione 2 (Avversario che attacca un protocollo) *Considerato un avversario che conduce attacchi ai danni di un protocollo, si supporrà che lo stesso*

- *sia a conoscenza delle debolezze dei partecipanti e cerchi sempre di sfruttarle;*
- *possa essere uno dei partecipanti legittimi alla comunicazione o un elemento esterno o una combinazione dei due;*
- *sia in grado di intercettare e di modificare tutti i messaggi inviati in un protocollo crittografico usando le informazioni disponibili;*
- *sia in grado di inoltrare un qualsiasi messaggio ad un'altra parte e tale inoltro gli permetta di generare ed inserire messaggi completamente nuovi nella comunicazione;*
- *sia in grado di ottenere il valore della chiavi di sessione usate nelle precedenti esecuzioni del protocollo.*

3.2 Principali protocolli di autenticazione esaminati

In accordo con l'obiettivo primario del progetto, in questa sezione vengono analizzate le caratteristiche dei principali protocolli di autenticazione; come anticipato, sono illustrate le seguenti tre classi di protocolli: *protocolli basati su password*, *su challenge-response* ed *a conoscenza zero*.

Nello specifico, per quanto riguarda i *protocolli basati su password*, sono trattati i seguenti: *un protocollo "ingenuo" utilizzato nei terminali sin dagli anni settanta*, il *protocollo di Needham usato nel sistema Unix*, il *Password Autentication Protocol* (PAP), il
Challenge Handshake Protocol (CHAP), il *One Time Password* (OTP), il OTP S/KEY ed il *Diffie-Helmann Key Exchange* (DH-EXE); per l'illustrazione di quest'ultimo protocollo si rende necessaria prima la trattazione dei protocolli *Diffie-Helmann* (DH) *Key Exhcange* e la *Encrypted Key Exchange* (EKE) *versione base del protocollo di Bellovin e Meritt*.

Con riferimento ai protocolli basati su *challenge-response* sono illustrati un *protocollo basato su algoritmi a chiave simmetrica* (*crittografia a chiave privata*), un *protocollo basato su Message Authentication Code* (MAC), un *protocollo basato su algoritmi a chiave asimmetrica* (*crittografia a chiave pubblica*) ed un *protocollo basato su un Key Distribution Center* (KDC).

Infine, per quanto riguarda i *protocolli a conoscenza zero*, sono esaminati il *protocollo Fiat-Shamir* ed il *protocollo Fiege-Fiat-Shamir*.

Prima di passare alla trattazione è necessario osservare che la predetta classificazione dei protocolli non deve essere intesa in senso stretto; infatti, come si avrà modo di evidenziare nella trattazione, i protocolli, in base alla loro progettazione, possono avere caratteristiche riferibili a più di una delle categorie indicate ed essere classificati secondo molteplici criteri.

3.2.1 Protocolli basati su password

Con riferimento agli scenari di autenticazione indicati in base alla tipologia dei partecipanti, si consideri quello di tipo *user-host* nel quale *un utente desidera accedere ad un servizio erogato da un sistema che richiede l'esecuzione di un protocollo di autenticazione basato su password*; l'utente, quindi, per essere autenticato deve necessariamente disporre di una *password/chiave* che deve essere validata dal componente del sistema che effettua l'autenticazione.

Definizione 95 (Password) *Si consideri uno scenario di tipo user-host. Con il termine password si definisce una chiave simmetrica, nota sia allo user che al host; tale chiave fornita dallo user al host, una volta effettuato il processo di verifica, consente in genere allo user l'accesso ad una risorsa, a cui è autorizzato, messa a disposizione dal host.*

Una password per essere considerata "abbastanza" sicura deve soddisfare dei *requisiti di complessità*, in accordo con le politiche di sicurezza imposte nel sistema di utilizzo;

tali requisiti possono, a tutt'oggi, essere individuati almeno nel rispetto delle seguenti condizioni: lunghezza minima della stringa di 10 caratteri, presenza di lettere minuscole (a-z), presenza di lettere maiuscole (A-Z), presenza di numeri arabi (0-9), presenza di caratteri non alfanumerici (ad esempio !, ?, ♯, *); infatti, nel caso in cui un avversario voglia individuare tale chiave con un *attacco di forza bruta*, supponendo che per ciascun carattere esistano 64 possibilità, le combinazioni possibili da tentare sono

$$64^{10} = 1.152.921.504.606.846.976 \ .$$

Dunque tale numero elevato garantisce una adeguata sicurezza nei confronti dell'attacco. La seguente tabella riporta, per ogni lunghezza di password scelta, le combinazioni possibili di caratteri ed il tempo impiegato per la sua decifrazione, considerando un calcolatore capace di processare 63 miliardi di operazioni al secondo (nel 2019)[3].

Lungh.	Comb.	Sec	Min	Giorno	Anno
1	64	> 1			
2	4096	> 1			
3	262144	> 1			
4	16777216	> 1			
5	$1,07 \cdot 10^9$	> 1			
6	$6,87 \cdot 10^{10}$	1,1			
7	$4,40 \cdot 10^{12}$	69,8			
8	$2,81 \cdot 10^{14}$		74,5	0,1	
9	$1,80 \cdot 10^{16}$			3,3	
10	$1,15 \cdot 10^{18}$			211,8	0,6
11	$7,38 \cdot 10^{19}$				37
12	$4,72 \cdot 10^{21}$				2.375
13	$3,02 \cdot 10^{23}$				158.018
14	$1,93 \cdot 10^{25}$				9.729.155

Si può notare che le password da 7 caratteri possono essere recuperate in meno di un minuto mentre quelle di 8 caratteri in due ore e mezza indipendentemente dalla loro complessità; dunque le password di almeno 10 caratteri sono considerate "abbastanza" sicure, occorrendo almeno 200 giorni per verificare tutte le possibilità con tale potenza di calcolo.

Definizione 96 (Protocolli di autenticazione deboli) *I protocolli di autenticazione basati su password si dicono deboli, e dunque insicuri, se hanno almeno una delle seguenti caratteristiche:*

- *prevedono l'uso di password che non possono essere cambiate oppure che non cambiano con una frequenza "sufficiente";*
- *espongono la password ad attacchi di dizionario on-line o off-line;*

[3] `https://www.computersec.it/2019/12/10/password-cracking-ultimate-force/`

- *espongono la password in chiaro nella trasmissione o nell'archiviazione.*

Contrariamente a quanto si possa pensare, a tutt'oggi, i protocolli basati su password sono molto diffusi poiché possono essere usati semplicemente con entità "umane"; in ogni caso va evidenziato che l'adozione di tali protocolli, negli ambienti di produzione, risulta in contrasto con la normativa vigente in materia di privacy e sicurezza informatica; infatti, per le caratteristiche indicate nella definizione precedente, ne deriva che un avversario eseguendo uno uno o più attacchi al sistema, riesce facilmente ad individuare le credenziali valide per l'autenticazione; tali debolezze sono ben evidenziate nelle apposita sezione relativa al caso di studio della web application realizzata che utilizza, appunto, un protocollo debole.

Nel corso della trattazione dei protocolli di autenticazione eseguiti nello scenario di tipo *user-host* sono evidenziate le possibili soluzioni di *remediation*, per poter stabilire una comunicazione sicura, che possano superare, per quanto possibile, le debolezze insite nell'autenticazione basata su password.

3.2.2 Protocollo "ingenuo" basato su password

Protocollo 1 (Protocollo "ingenuo" basato su password) *Sia U un utente che vuole accedere ad un sistema mediante un host H. Si indica con*

- *ID_U l'identità di U ovvero i dati necessari per identificare U;*
- *P_U la password associata ad U;*
- *(ID_U, P_U) la coppia, costituita rispettivamente dall'identità di U e dalla relativa password memorizzata nell'archivio di H per autenticare U.*

Il diagramma di sequenza di un protocollo "ingenuo", che permette ad U di autenticarsi con H tramite la password P_U, è il seguente:

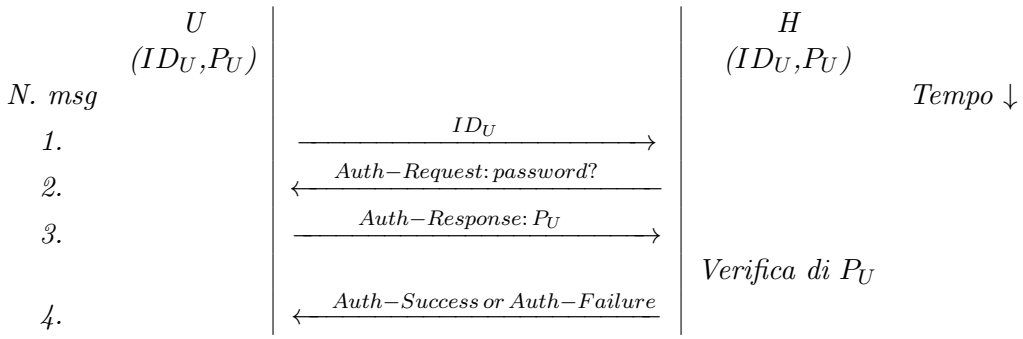


Diagramma 1: Diagramma di sequenza di un semplice protocollo "ingenuo" basato su password.

In dettaglio, l'analisi dei messaggi scambiati nell'esecuzione del protocollo:

- *U invia il proprio ID_U con il messaggio* 1*;*

- *ricevuto ID_U, H chiede ad U la password con il messaggio* 2*;*
- *U invia in risposta la proprio password P_U con il messaggio* 3*;*
- *ricevuta P_U, H ne verifica la corrispondenza con quella memorizzata in (ID_U, P_U) nel suo archivio; con il messaggio* 4*, H autentica U nel caso in cui la condizione sia vera, altrimenti ne rifiuta l'accesso.*

Il protocollo illustrato è stato utilizzato nei terminali sin dagli anni settanta; permetteva, infatti, ad un utente di accedere ad un host con un terminale mediante una linea dedicata considerata inattaccabile.

In base a quanto definito, si evidenzia sin da subito che *il protocollo garantisce solamente un autenticazione unilaterale* di U con H e, quindi, U non ha alcuna assicurazione sull'identità del suo autenticatore; paradossalmente potrebbe essere anche un attaccante che, spacciandosi per H, è così riuscito ad ottenere la password di U.

Osservazione 1 (Debolezze del protocollo "ingenuo" basato su password) *Nello scenario user-host, supponendo che l'utente U utilizzi un accesso remoto tramite una rete pubblica per accedere ad un host H, il protocollo presenta le seguenti debolezze:*

- *la freshness dei messaggi non è verificata;*
- *la coppia (ID_U, P_U) memorizzata in chiaro nell'archivio di H è vulnerabile;*
- *le informazioni di (ID_U, P_U) sono trasmesse in chiaro nel canale tra U e H.*

Analizzando tali le debolezze, si possono dedurre le seguenti considerazioni:

- *in una rete pubblica*, a differenza di quanto avveniva nelle rete dedicata utilizzata dai terminali, il protocollo non garantisce alcun tipo di autenticazione, né unilaterale, né reciproca;
- *dalla mancanza della verifica della freshness dei messaggi* deriva l'impossibilità per H di sapere effettivamente se la comunicazione con U sia attiva o se invece sia in corso un *attacco di tipo replay*;
- *dalla vulnerabilità della coppia (ID_U,P_U) memorizzata in chiaro nell'archivio di H* segue che il file potrebbe essere letto da un avversario A al fine di ottenere le informazioni necessarie per impersonare U od altri utenti memorizzati in tale archivio e, di conseguenza, acquisire diritti di accesso a risorse non autorizzate in pieno anonimato; ne derivano, dunque, anche *problemi di sicurezza nel backup dello stesso archivio di H*;
- infine, *il fatto che le informazioni della coppia (ID_U,P_U) siano inviate in chiaro nel canale tra U e H* permette ad un avversario in ascolto di poter facilmente recuperare le stesse con un semplice *attacco passivo di sniffing.*

3.2.3 Protocollo di Needham usato dal sistema Unix

Per risolvere la debolezza del protocollo precedente, relativa alla memorizzazione in chiaro della password P_U nell'archivio di H, è stato proposto da Needham un metodo semplice ed efficiente per salvare in maniera sicura le password in un file dell'host H; *l'idea prevede l'utilizzo di una funzione one-way per cifrare la password da parte dell'host H e la memorizzazione nello stesso della coppia $(ID_U, f(P_U))$, dove $f(P_U)$ è l'applicazione della funzione one-way f alla password P_U, al posto di (ID_U, P_U)*; con le modifiche apportate da Needham, il protocollo "ingenuo" precedente, risulta così definito.

Protocollo 2 (Protocollo di Needham) *Sia U un utente che vuole accedere ad un sistema mediante un host H. Si indica con*

- *ID_U l'identità di U ovvero i dati necessari per identificare U;*
- *P_U la password associata ad U;*
- *f una funzione one-way applicata in H;*
- *$(ID_U, f(P_U))$ la coppia, costituita dall'identità di U e da $f(P_U)$, immagine della password P_U mediante la funzione one-way f, memorizzata nell'archivio di H per autenticare U.*

Il diagramma di sequenza del protocollo di Needham, che permette ad U di autenticarsi con H tramite password P_U, è il seguente:

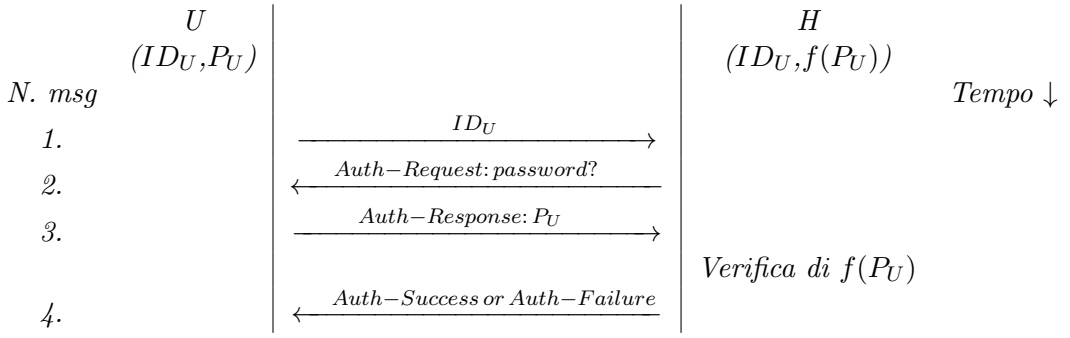


Diagramma 2: Diagramma di sequenza del protocollo di autenticazione di Needham usato dal sistema Unix.

In dettaglio,

- *U invia il proprio ID_U con il messaggio 1;*
- *ricevuto ID_U, H chiede ad U la password con il messaggio 2;*
- *U invia in risposta la propria password P_U con il messaggio 3;*
- *ricevuto P_U, H calcola $f(P_U)$ e verifica se esiste $(ID_U, f(P_U))$ nel suo archivio: in caso affermativo autentica U altrimenti ne rifiuta l'accesso con il messaggio 4.*

Il protocollo di Needham è alla base dello schema di autenticazione utilizzato nei sistemi operativi UNIX fin dagli anni settanta; a riguardo, prima di analizzare alcuni dettagli dell'implementazione su tali sistemi, si definisce sommariamente l'algoritmo di cifratura utilizzato denominato DES.

Definizione 97 (Data Encription Standard (DES)) *Il Data Encription Standard (DES) è un algoritmo di cifratura simmetrico, utilizzato dai sistemi di tipo block chiper, mediante il quale un messaggio m viene diviso in blocchi di* 64 *bit ciascuno, a loro volta manipolati con permutazioni, somme mod* 2 *con funzioni della chiave, ulteriori suddivisioni in sottoblocchi e così via.*

L'utilizzo del DES, introdotto come standard nel 1976*, ha rappresentato la prima standardizzazione dei sistemi crittografici che ha consentito a due utilizzatori di scambiarsi unicamente la chiave del sistema senza la necessità di dover scambiare anche l'algoritmo*; attualmente l'algoritmo viene considerato insicuro principalmente a causa della sua chiave troppo corta (dei 64 bit solamente 56 sono utili poiché gli altri 8 sono di controllo); *dal* 2009*, infatti, una chiave DES può essere forzata in poche ore con un attacco di forza bruta.*
Di seguito si illustra in dettaglio l'implementazione del protocollo di Needham nel sistema UNIX.

Un sistema UNIX per l'implementazione del protocollo di autenticazione di Needham utilizza:

- una *funzione one-way* f realizzata tramite l'algoritmo DES;
- un *file di password* in cui l'host H salva l'identità dell'utente U, indicata con UID, ed un messaggio cifrato con DES; nello specifico l'algoritmo simmetrico viene applicato ad una stringa di 64 zeri e la password P_U viene utilizzata come chiave;
- la funzione $f(P_U)$, anziché applicare l'algoritmo puro di DES, utilizza una funzione *crypt*(), progettata da Morris e Thompson, per aumentare il costo della ricerca della chiave dal punto di vista computazionale.

La funzione *crypt*() utilizza la ripetizione del DES per 25 volte con l'aggiunta di un metodo di "*bit-swap permutation*"; il metodo prevede che ad ogni ripetizione dell'algoritmo siano scambiati alcuni blocchi del messaggio cifrato ottenuto dalla chiamata precedente secondo una stringa di 12 bit casuali, detta *salt*, presi dal clock al tempo della creazione della password (con il comando */bin/passwd*); il salt viene convertito in una stringa di due caratteri e viene memorizzato nel file delle password (*/etc/passwd*) insieme alla "password" crittografata o meglio, si dovrebbe dire, al "blocco di zeri criptato" che viene utilizzato per verificare la password dell'utente; con tali accorgimenti la funzione $f(P_U)$, indicata nel protocollo, che utilizza il DES, può essere vista come una funzione hash one-way applicata alla stringa costante 0^{64}, dove la chiave è la password P_U ed il parametro è il salt a 12 bit. Grazie all'uso del salt, nel file delle password dell'host H le informazioni per l'accesso degli utenti sono salvate come (ID_U, *salt*, $f(P_U$,*salt*)).

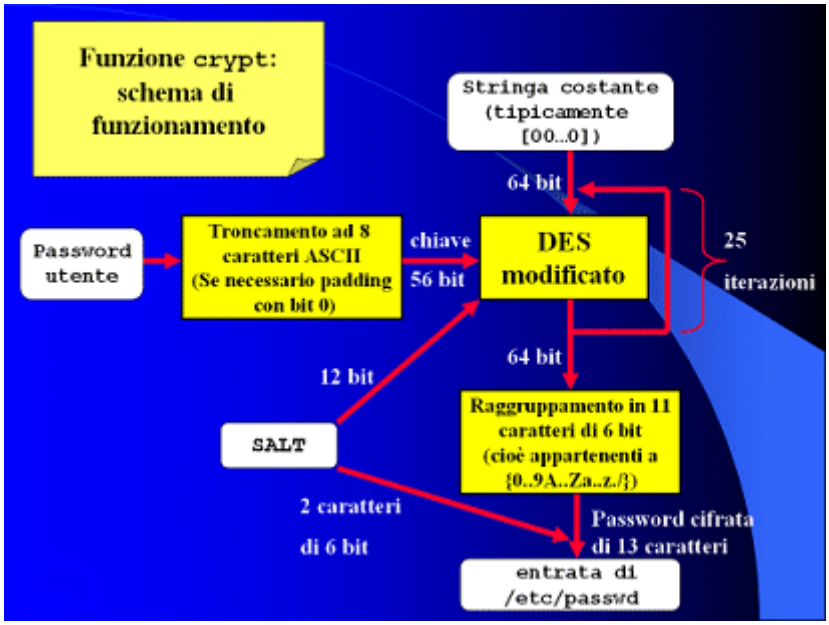


Figura 1: La funzione crypt() in UNIX

Il protocollo di autenticazione di Needham, realizzato per UNIX, migliora la sicurezza delle password rispetto a quanto previsto dal precedente protocollo; infatti nel caso in cui un *avversario* A riuscisse ad ottenere *la funzione della password* dal file dell'host H, $f(P_U, salt)$, che per comodità si indicherà con $P_{UStolen}$, con la modifica introdotta da Morris e Thomson, si avrà:

	A		H
	$P_{UStolen}$		$(ID_U, salt, f(P_U, salt))$
N. msg			*Tempo* $\downarrow$
1.		$\xrightarrow{ID_U}$	
2.		$\xleftarrow{Auth-Request: password?}$	
3.		$\xrightarrow{Authenticate-response: P_{UStolen}}$	
			Verifica di $f(P_{UStolen})$
4.		$\xleftarrow{Auth-Failure}$	

Diagramma 3: Diagramma di sequenza di un tentativo di attacco al protocollo di autenticazione di Needham usato dal sistema Unix.

In dettaglio,

- A, fingendosi U, invia ad H la sua identità, ID_U, con il messaggio 1;
- H invia la richiesta di password per l'autenticazione con il messaggio 2;

- A, tentando di ingannare H, invia la $P_{UStolen} = f(P_U, salt)$, ottenuta in modo fraudolento, con il messaggio 3;
- H calcola, quindi, la trasformazione della password ricevuta $P_{UStolen}$ tramite la funzione f ed il salt relativo a ID_U prelevato da (ID_U, $salt$, $f(P_U, salt)$) ovvero

$$f(P_{UStolen}, salt\,) = f(f(P_U, salt),\, salt\,)$$

  e verifica che la stringa risultante è ben differente da quella corrispondente a ID_U associata a (ID_U, $salt$, $f(P_U, salt)$) che si trova nel file di H; pertanto H rifiuta l'accesso ad A che si finge U.

L'algoritmo *crypt*() di Morris and Thompon è stato utilizzato in sostituzione della funzione f del protocollo dalla v.7 di UNIX; in base alla sua costruzione, *crypt*() offriva una maggiore difesa dagli attacchi di dizionario effettuati con gli hardware che utilizzavano i chip DES disponibili commercialmente a basso costo; le 25 ripetizioni del bit-swapping permutation evitano anche che le stesse password abbiano una cifratura identica in sistemi differenti; tuttavia, a tutt'oggi, lo stesso algoritmo *crypt*() risulta abbastanza insicuro; infatti con hardware dedicati è possibile violare le password, specialmente se hanno una lunghezza inferiore ad 8 caratteri in tempi ragionevoli, considerando tutti i 4096 ($= 2^{12}$) salt possibili, con un attacco a forza bruta.

In base a quanto esposto, seppur il protocollo garantisca una "certa" confidenzialità del file delle password *non assicura la garanzia che tale file rimanga integro*, ovvero che non possa essere modificato; per tale motivo nei sistemi operativi UNIX e UNIX-like si utilizzano, a tutt'oggi, le shadow password.

Definizione 98 (Shadow password) *Le shadow password costituiscono un meccanismo, presente nei sistemi operativi Unix e Unix-like, per memorizzare in forma crittografata le password di autenticazione al sistema in file (ad esempio /etc/shadow) accessibili in lettura e scrittura solo ad utenti privilegiati (ad esempio l'utente root).*

Con l'implementazione delle shadow password in un sistema, le password cifrate non sono più memorizzate in un file (ad esempio */etc/passwd*) che, invece, risulta accessibile in lettura a tutti gli utenti del sistema; in tale file al posto del "blocco di zeri criptato", che è utilizzato per verificare la password di ciascun utente, viene inserita una semplice "x" ad indicare che il valore è memorizzato altrove; *con l'implementazione delle shadow password risulta, dunque, più difficoltoso per un attaccante ottenere i dati necessari per effettuare un attacco a forza bruta.*

Osservazione 2 (Debolezze del protocollo di Needham) *A prescindere dall'implementazione delle shadow password, il protocollo di autenticazione di Needham può essere classificato con un protocollo debole in quanto prevede la comunicazione delle credenziali di autenticazione in chiaro; pertanto, ad un avversario A è sufficiente un attacco di eavesdropping, sfruttando il fatto che il canale utilizzato è insicuro, per ottenere (ID_U, P_U) e autenticarsi all'host H fingendosi la vittima U.*

3.2.4 Password Authentication Protocol (PAP)

Il Password Authentication Protocol (PAP), il cui standard è specificato nella RFC 1334, è un protocollo usato esclusivamente per l'autenticazione nel Point-to-Point Protocol, (PPP), un protocollo di rete di livello di *collegamento dati* del modello ISO/OSI, comunemente usato nello stabilire connessioni dirette tra due nodi, che era stato progettato per lavorare assieme ad altri protocolli di *livello superiore* come l'IP, IPX, e AppleTalk.

Nell'esecuzione del protocollo PAP si presume che il canale di trasmissione tra l'utente e l'host di autenticazione sia *sicuro*.

Protocollo 3 (Password Authentication Protocol (PAP)) *Sia U un utente che vuole accedere ad un sistema mediante un host H; si indica con*

- *ID_U l'identità di U;*
- *P_U la password associata ad U;*
- *f una funzione di tipo Message Detection Code (MDC) per provare l'integrità della password P_U di U da parte di H;*
- *h la trasformazione della password P_U, ottenuta tramite la funzione f ed un salt casuale, ovvero $h = f(P_U, salt)$;*
- *(ID_U, h) la coppia, costituita dall'identità di U e dalla trasformazione della P_U tramite la funzione f, con l'utilizzo del salt, memorizzata nell'archivio di H per autenticare U.*

Lo schema del protocollo PAP, che permette ad U di autenticarsi con H tramite password P_U, è il seguente:

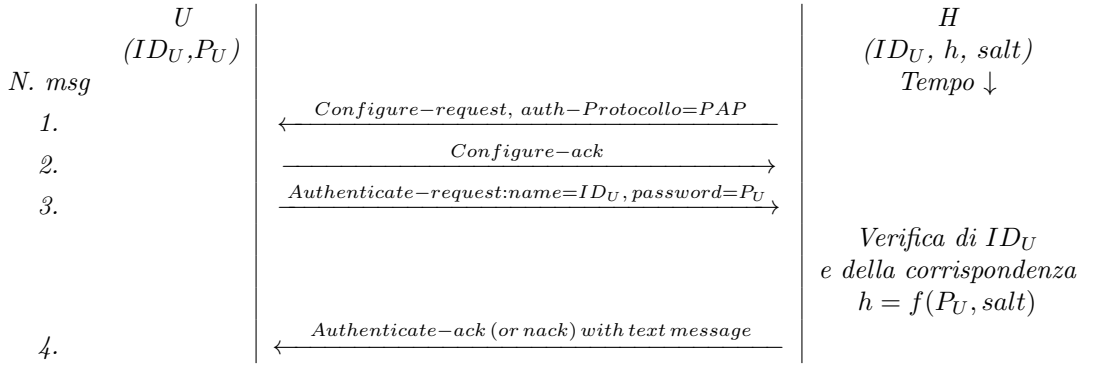


Diagramma 4: Diagramma di sequenza del Password Authentication Protocol (PAP).

Nello specifico, con i messaggi 1 *e* 2 *le parti stabiliscono il collegamento iniziale e si accordano sul protocollo; con il messaggio* 3 *vengono inviate le credenziali e con il messaggio* 4 *viene fornita la risposta in base al risultato della verifica da parte di H; dunque per la verifica dell'identità del client si utilizza un handshake a due vie.*

La verifica della coppia (ID_U, P_U), inviata da U ad H, avviene nel seguente modo:

- *H cerca nel suo archivio ID_U con una funzione di lookup;*
- *se H non trova ID_U l'accesso è rifiutato con un authentication-nack;*
- *se H trova ID_U, ne preleva il valore corrispondente di h ed il salt utilizzato;*
- *H calcola $h' = f(P_U, salt)$, dove P_U è la password comunicata da U;*
- *H confronta i valori di h ed h': se $h = h'$ l'autenticazione ha successo ed è comunicato ad U con un authentication-ack altrimenti l'accesso è rifiutato.*

Analogamente a quanto osservato per il protocollo precedente, il numero casuale del salt rende gli *attacchi a dizionario* più difficili; l'utilizzo del salt garantisce, inoltre, nel caso in cui due utenti abbiano la stessa password, che sia difficilmente probabile il verificarsi dell'evento di avere la stessa trasformazione della password; un'ulteriore analogia con il protocollo precedente riguarda, infine, la confidenzialità dell'archivio delle password che, per quanto esposto, non sono conservate in chiaro.

Osservazione 3 (Debolezze di PAP) *Il protocollo PAP è sicuramente un protocollo debole perché sensibile ai seguenti attacchi:*

- *eavesdropping in quanto le credenziali sono comunicate in chiaro nei messaggi;*
- *man in the midle poiché non è previsto alcuna verifica dell'identità delle parti e, pertanto, un avversario può ingannare la vittima fingendo di essere il server di autenticazione;*
- *replay poiché un avversario può registrare un'autenticazione e, quindi, riprodurla in un secondo momento;*
- *attacchi di dizionario e forza bruta ai danni delle password memorizzate nell'archivio; le password, seppur crittografate, sopratutto nel caso in non siano state scelte secondo adeguati criteri, come visto, possono essere recuperate in un tempo ragionevole.*

Un'ultima considerazione riguarda il canale di comunicazione utilizzato dal protocollo considerato sicuro: in realtà non è detto che lo sia davvero.

Per quanto riguarda possibili le estensioni del protocollo, bisogna sottolineare che la PAP RFC non afferma che la password debba essere fissa o provenire necessariamente dall'utente; la password, infatti, può essere sottoposta ad hashing prima di essere trasmessa; tale miglioramento, in ogni caso, non risolverebbe la debolezza all'*attacco di tipo replay*; un ulteriore misura di sicurezza prevede l'utilizzo di PAP con una *token card*.

In conclusione, in base a quanto esposto, PAP è considerato uno schema di autenticazione debole la cui applicazione andrebbe limitata solo ad alcuni ambienti vincolati; *in generale, quindi, ne risulta decisamente sconsigliato l'utilizzo in ambienti di produzione che fanno uso di canali di trasmissione pubblici a meno dell'utilizzo in un tunnel cifrato per ovviarne almeno le debolezze della trasmissione delle credenziali in chiaro.*

3.2.5 Challenge Handshake Protocol (CHAP)

Il Challenge Handshake Authentication Protocol (CHAP), il cui standard è specificato nella RFC 1994, è un protocollo di autenticazione usato dai server PPP per convalidare l'identità dei client remoti; è una variante del Protocollo PAP.

A differenza del PAP, il protocollo CHAP utilizza un metodo handshake a 3 vie per verificare l'identità del client con l'utilizzo di un "challenge" e può ripetere tale processo più volte nell'arco della connessione.

Protocollo 4 (Challenge Handshake Authentication Protocol (CHAP)) *Sia U un utente che vuole accedere ad un sistema mediante un host H; si indica con*

- *ID_U ed ID_H, rispettivamente, le l'identità di U e di H;*
- *P_U la password associata ad U;*
- *c un numero casuale (salt), generato da H, denominato challenge;*
- *f una funzione hash one-way utilizzata sia da U che da H per le operazioni indicate al punto successivo (nell'implentazione standard del CHAP è utilizzata la funzione MD5, standardizzata con la RFC* 1321*, che prende in input una stringa di lunghezza arbitraria e ne produce in output un'altra a 128 bit);*
- *r la trasformazione della password P_U di U, ottenuta tramite la funzione f, le credenziali ID_H ed il challenge c, generato da H, ovvero $r = f(ID_H, P_U, c)$;*
- *(ID_U, P_U) la coppia, costituita dall'identità di U e dalla sua password associata, memorizzata in H.*

Lo schema del protocollo CHAP, che permette ad U di autenticarsi con H tramite password P_U, è il seguente:

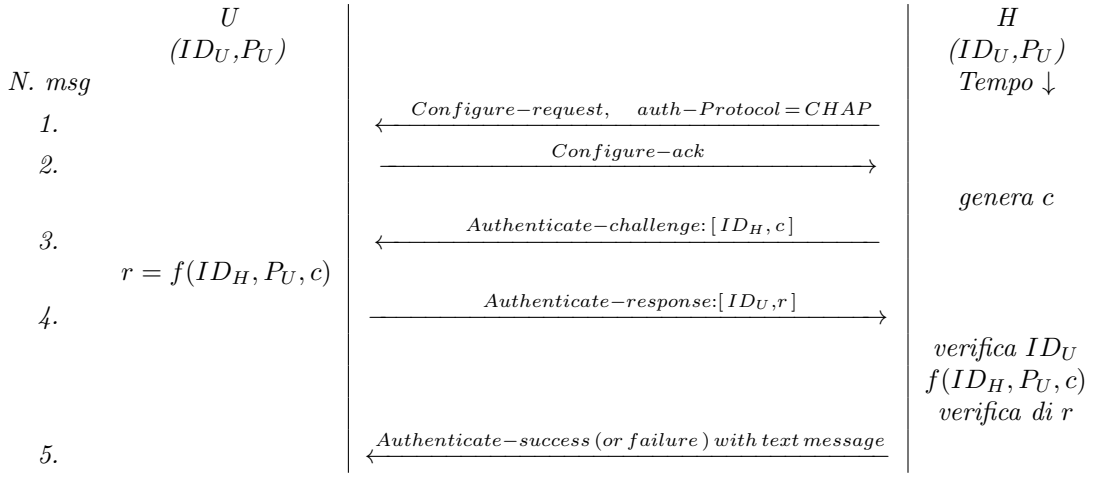


Diagramma 5: Diagramma di sequenza del Challenge Handshake Protocol (CHAP).

In dettaglio, con i messaggi 1 *e* 2 *viene stabilito il collegamento iniziale tra le parti per accordarsi sul protocollo e con quelli* 3, 4 *e* 5 *viene verificata l'identità di U utilizzando un handshake a tre vie.*

La verifica di $[\, ID_U, r \,]$*, inviata da U ad H, ovvero di* $(\, ID_U, f(ID_H, P_U, c) \,)$ *avviene nel seguente modo:*

- *H cerca nel suo archivio* ID_U *con una funzione di lookup;*
- *se H non trova* ID_U *l'accesso è rifiutato con un authentication-nack;*
- *se H trova* ID_U *ne preleva la password corrispondente* P_U*;*
- *H calcola* $r' = f(ID_H, P_U, c)$*, dove* P_U *è la password di U e c il challenge generato e comunicato a U;*
- *H confronta i valori di r ed* r'*: se* $r = r'$ *l'autenticazione ha successo ed è comunicato ad U con un authentication-ack altrimenti l'accesso è rifiutato.*

In sintesi nello schema CHAP una volta stabilito il collegamento, H invia il proprio identificativo ed un challenge; U invia in risposta il suo identificativo ed una stringa calcolata tramite una funzione hash one-way applicata all'identificativo di H, alla sua password ed al challenge; H controlla l'hash generato, anch'esso tramite la stessa funzione one-way applicata al proprio identificativo, la password memorizzata ed il challenge, e se identico autentica U altrimenti termina la connessione.

Nell'esecuzione del protocollo è fondamentale che le parti utilizzino la stessa funzione hash unidirezionale che può essere arbitraria; *in ogni caso affinché l'implementazione di CHAP sia "compiant" alla RFC* 1994*, entrambe le parti devono supportare almeno MD5*; tale RFC afferma, inoltre, che *il challenge casuale deve mostrare unicità globale e temporale*; pertanto, se il protocollo viene utilizzato due volte, per autenticare entrambe le parti, non deve essere utilizzato lo stesso challenge altrimenti la seconda autenticazione potrebbe essere una replica della prima.

Il protocollo CHAP risolve le vulnerabilità di PAP non trasmettendo la password in chiaro e, quindi, evita attacchi di eavesdropping alle chiave; l'utilizzo di un identificatore e del challenge, invece, evitano gli attacchi di replay; tali attacchi, infatti, non sono efficaci se si utilizzano challenge variabili.

Osservazione 4 (Debolezze di CHAP) *Il protocollo CHAP, nonostante il meccanismo di challenge-response, è sicuramente un protocollo debole perché sensibile ai seguenti attacchi:*

- *attacco a dizionario su r;*
- *attacco replay nel caso in cui c non sia davvero scelto in modo casuale;*

Inoltre necessita che l'archivio delle password nell'authentication server sia in chiaro.

Un ulteriore considerazione sulla debolezza di CHAP riguarda l'utilizzo del MD5 come funzione hash crittografica; ad oggi sono disponibili molte risorse online che hanno buone probabilità di riuscire a decriptare parole comuni codificate con MD5; *l'algoritmo non è più considerato sicuro (sebbene il controllo di integrità sui file md5sum sia altamente diffuso) ed è altamente raccomandato l'uso di strumenti di hashing più recenti come il sha-256.*

Esistono delle varianti del protocollo CHAP sviluppate da Microsoft, *MS-CHAP v1* e *MS-CHAP v2*, modificato per integrare la funzionalità di autenticazione basate sulla cifratura tra i client dei sistemi operativi Windows; MS-CHAP eredita dal CHAP il meccanismo di challenge-response per autenticare attraverso una password il client verso il server; *a differenza di quanto previsto nel CHAP, per l'implementazione del PAP il server di autenticazione non ha la necessità di memorizzare la password in formato cifrato*; gli algoritmi utilizzati per il processo di cifratura e per quello di hashing sono rispettivamente il DES (Data Encryption Standard) e l'MD4.

Con il rilascio di Windows Vista nel 2007 il protocollo è stato dichiarato obsoleto ed escluso dai protocolli presenti in tale sistema operativo; *in ogni caso ulteriori problematiche di sicurezza relative a MS-CHAP v2 sono state evidenziate anche nell'implementazione delle stesso protocollo con i successivi sistemi operativi tanto da sconsigliarne l'utilizzo senza incapsulamento insieme al tunnel PPTP per la connettività VPN in quanto la configurazione è potenzialmente non sicura*[4].

3.2.6 One Time Password (OTP)

Definizione 99 (One Time Password - OTP) *Con l'acronimo OTP, ovvero One Time Password, si fa riferimento ai sistemi nei quali viene generata una nuova password ad ogni accesso da parte dell'utente per risolvere il problema dell'intercettazione.*

Uno schema OTP prevede che un utente U, per autenticarsi con un host H, debba ogni volta utilizzare un password diversa da quella precedente e che nessuna delle password usate da U possa essere riutilizzata (cosiddette password "monouso" o "usa e getta"); in tal modo un avversario che abbia scoperto una delle password usate nel canale tra U ed H con un attacco di eavesdropping, non riuscirà ad utilizzarla con successo in un'altra esecuzione del protocollo.

Il sistema di password monouso, basato su una catena di hash, è standardizzato con la RFC 2289.

Definizione 100 (Lamport hash chain) *Sia f una funzione on-way, $n \in \mathbb{N}^+$, con n sufficientemente "grande" e P_U la password dell'utente U; si definisce $f^n(P_U)$ la funzione f eseguita per n volte su P_U:*

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

e

[4] `https://docs.microsoft.com/en-us/security-updates/securityadvisories/2012/2743314`

$$P_i = f^{n-i}(P_U),\ con\ 0 \leq i \leq n-1,$$

l'i-esima one time password ottenuta dalla $(n-i)$*-esima applicazione della funzione* f *a* P_U*.*

Dalla definizione precedente deriva che, *applicando la funzione di hash su* P_U*, si crea una sequenza di n password differenti percorribile solo in un senso*, ovvero:

$$P_{n-1} = f(P_U) \longrightarrow P_{n-2} = f^2(P_U) \longrightarrow \cdots \longrightarrow P_1 = f^{n-1}(P_U) \longrightarrow P_0 = f^n(P_U)$$

dove $P_{n-1} \neq P_{n-2} \neq \cdots \neq P_1 \neq P_0$.

Le password così calcolate dovranno essere utilizzate per l'autenticazione in senso contrario escludendo l'ultima P_0; dunque la successione delle n-1 password calcolate da seguire sarà:

$$P_1 \longrightarrow P_2 \longrightarrow \cdots \longrightarrow P_{n-1}\,.$$

Con tali nozioni è possibile, finalmente, definire un protocollo di autenticazione basato su OTP.

Protocollo 5 (Protocollo di autenticazione basato su OTP) *Sia* U *un utente che vuole accedere ad un sistema mediante un host* H*; si indica con*

- ID_U *l'identità di* U*;*
- P_U *la password associata ad* U*;*
- f *una funzione one-way utilizzata sia da* U *che da* H *per le operazioni illustrate ai punti successivi;*
- n *un numero intero sufficientemente "grande";*
- $(ID_U,\ f^n(P_U))$ *la coppia, costituita dall'identità di* U *e dall'esecuzione* n *volte della funzione* f *su* P_U*, memorizzata nell'archivio di* H *per autenticare* U*.*

Lo schema del protocollo basato su OTP, che permette ad U *di essere autenticato da* H *tramite password OTP* P_i*, con* $1 \leq i \leq n-1$*, generate a partire da* P_U *con una hash chain, una volta stabilito il collegamento iniziale ed accordati sulle regole, è il seguente:*

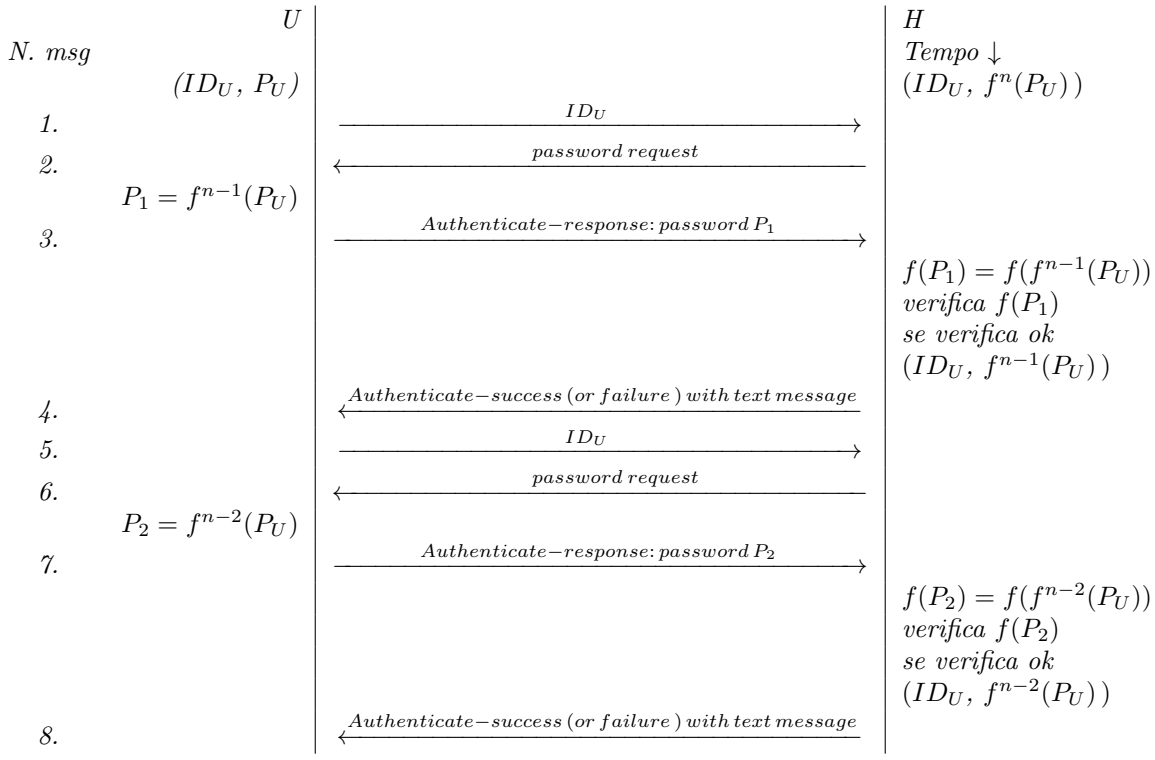


Diagramma 6: Diagramma di sequenza di un protocollo di autenticazione basato su OTP.

Di seguito l'analisi dei messaggi tra U *ed* H*:*

- U *invia la sua identità,* ID_U*, ad* H *con il messaggio* 1*;*
- H*, ricevuta* ID_U*, invia ad* U *la richiesta di password per l'autenticazione con il* 2*;*
- U *calcola* $P_1 = f^{n-1}(P_U)$ *e lo invia ad* H *con il messaggio* 3*;*
- H *calcola, quindi, la trasformazione della password ricevuta,* $P_1 = f^{n-1}(P_U)$*, tramite la funzione* f*, ovvero* $f(P_1) = f(f^{n-1}(P_U)) = f^n(P_U)$*, e verifica se la stringa risultante corrisponde a quella associata ad* ID_U *in* $(ID_U,\ f^n(P_U))$ *nel file delle password; in caso di corrispondenza riuscita,* H *sostituisce l'OTP memorizzato con quello fornito da* U *nel suo archivio che ospiterà dunque* $(ID_U,\ f^{n-1}(P_U))$ *da utilizzare per il prossimo confronto; quindi* H *consente ad* U *di procedere con il login comunicando anche il successo dell'autenticazione con il messaggio* 4 *; in caso di verifica negativa, invece,* H *invia solamente con il messaggio* 4 *la notifica del fallimento.*

Alla prossima richiesta di autenticazione, dopo lo scambio dei messaggi 5 *e* 6 *analoghi, rispettivamente, a* 2 *e* 3*,*

- U *calcola* $P_2 = f^{n-2}(P_U)$ *e lo invia ad* H *con il messaggio* 7*;*

- *H calcola, quindi, la trasformazione della password ricevuta, $P_2 = f^{n-2}(P_U)$, tramite la funzione f ovvero $f(P_2) = f(f^{n-2}(P_U)) = f^{n-1}(P_U)$ e verifica se la stringa risultante corrisponde a quella associata ad ID_U, ovvero $(ID_U,\, f^{n-1}(P_U))$ nel file delle password aggiornato con l'ulteriore computazione; in caso di corrispondenza riuscita, H sostituisce l'OTP memorizzato con quello fornito da U nel suo archivio che ospiterà dunque $(ID_U,\, f^{n-2}(P_U))$ da utilizzare per il prossimo confronto; quindi H consente ad U di procedere con il login comunicando anche il successo dell'autenticazione con il messaggio 8; in caso di verifica negativa, invece, H invia solamente con il messaggio 8 la notifica del fallimento.*

Analogamente U ed H procedono per le successive richieste di autenticazione.

Si osservi che nel corso dell'esecuzione del protocollo nella verifica condotta da H, a seguito del calcolo di f su P_i, ovvero su

- $(ID_U,\, f^n(P_U))$, essendo $f(P_1) = f(f^{n-1}(P_U)) = f^n(P_U)$ nella prima chiamata,
- $(ID_U,\, f^{n-1}(P_U))$, essendo $f(P_2) = f(f^{n-2}(P_U)) = f^{n-1}(P_U)$ nella seconda,
- $\cdots$
- $(ID_U,\, f^2(P_U))$, essendo $f(P_{n-1}) = f(f(P_U)) = f^2(P_U)$ nella $(n-1)$-esima,

H assume di aver ricevuto tali valori da U solamente se le password inviate da U superano le verifiche indicate in quanto condivide la conoscenza di P_U soltanto con U. Inoltre, nel corso dell'esecuzione del protocollo, U invia $P_1,\, P_2, \cdots, P_{n-1}$ con $P_1 \neq P_2 \neq \cdots \neq P_{n-1}$ e non più riutilizzabili.
Ricordando che $0 \leq i \leq n-1$, una volta che U ha consumato tutte le OTP password, calcolate con le computazioni indicate, è necessario che H comunichi ad U una nuova password P_U' su cui costruire una nuova sequenza di $P_i' = f^{n-i}(P_U')$, con $0 \leq i \leq n-1$, con la catena di hash; da notare, infine, che per $i = n-1$ si avrebbe $P_{n-1} = f^0(P_U) = P_U$, ovvero U non effettuerebbe alcuna operazione su P_U e, continuando con l'algoritmo U invierebbe P_U (in chiaro) ad H che dovrebbe confrontarla dopo l'applicazione di f con quella corrispondente alla coppia $(ID_U, f(P_U))$.

Osservazione 5 (Debolezze del protocollo di autenticazione basato su OTP) *Il protocollo OTP descritto, nonostante il meccanismo della catena delle password che limita il rischio di un attacco di eavesdropping essendo le password utilizzate tutte diverse tra loro, è sicuramente un protocollo debole poiché sensibile ai seguenti attacchi:*

- *attacco Man In The Middle (MITM) essendo l'autenticazione non mutua; infatti H non si autentica sul client utilizzato da U e non genera un segreto; dunque la connessione può essere dirottata;*
- *attacchi di modificazione e di replay ai danni del numero di sequenza dei messaggi, per alterare la sincronizzazione tra U ed H sulle password da utilizzare; infatti se le sincronizzazione tra le parti venisse a mancare l'autenticazione fallirebbe;*

ad esempio sia U impostato nell'invio della k-esima password nella forma $P_k = f^{n-k}(P_U)$ ed H nella verifica con $(ID_U, f^j(P_U))$, con $j \neq n-k+1$, ne deriverebbe che $f(P_k) = f(f^{n-k}(P_U)) = f^{n-k+1}(P_U) \neq f^j(P_U)$.

Inoltre, come visto, essendo l'autenticazione permessa solamente per un numero prefissato di volte, esaurito tale numero risulta necessaria una reinizializzazione per reimpostare la catena e, di conseguenza, l'esigenza di una comunicazione della nuova password da parte di H ad U in un canale anche out-of-band e con un metodo sicuro.

Nel protocollo OTP descritto il problema della sincronizzazione può essere risolto con un metodo che permetta di ristabilire la sincronizzazione tra U ed H nel caso fosse persa; tale metodo, però, necessita di un'autenticazione reciproca tra H ed U; l'autenticazione potrebbe essere implementata nel seguente modo:
se U è impostato nell'invio della k-esima password nella forma $P_k = f^{n-k}(P_U)$ ed H nella verifica di $f^j(P_U)$, con $j \neq n-k+1$ allora il sistema dovrà andare avanti e impostarsi nella situazione in cui H dovrà verificare

$$f(P_i) = f(f^{n-i}(P_U))$$

dove $P_i = f^{n-i}(P_U)$ è la password P_i inviata da U, con $i \leq min(\, j,\, k-1\,)$.

Di seguito si riporta lo schema della corrispondenza tra le password inviate da U e le verifiche condotte da H sul file aggiornato ad ogni iterazione con il decremento delle computazioni della funzione hash applicata a P_U; *l'algoritmo di verifica si arresta consumata l'n-esima password a disposizione di U che non prevede alcuna trasformazione di P_U, $P_n = f^0(P_U) = P_U$.*

U		H		
Sequenza password		Sequenza password	Verifica in	Tempo ↓
Password 1: $P_1 = f^{n-1}(P_U)$	$\longrightarrow$	$P_0 = f^n(P_U)$	(ID_U, P_0)	
Password 2: $P_2 = f^{n-2}(P_U)$	$\longrightarrow$	$P_1 = f^{n-1}(P_U)$	(ID_U, P_1)	
Password 3: $P_3 = f^{n-3}(P_U)$	$\longrightarrow$	$P_2 = f^{n-2}(P_U)$	(ID_U, P_2)	
...		...	...	
Password i: $P_i = f^{n-i}(P_U)$	$\longrightarrow$	$P_{i+1} = f^{n-i+1}(P_U)$	(ID_U, P_{n-i+1})	
...		...	...	
Password n-2: $P_{n-2} = f^2(P_U)$	$\longrightarrow$	$P_{n-3} = f^3(P_U)$	(ID_U, P_{n-3})	
Password n-1: $P_{n-1} = f(P_U)$	$\longrightarrow$	$P_{n-2} = f^2(P_U)$	(ID_U, P_{n-2})	

Diagramma 7: Diagramma di sequenza di un protocollo di autenticazione basato su OTP con evidenza delle sequenze delle password utilizzate dalle parti.

3.2.7 OTP S/KEY

Il problema della sincronizzazione esposto è stato risolto da un sistema OTP noto come S/KEY introducendo l'utilizzo di un contatore; il protocollo S/KEY, a volte indicato come *schema di Lamport*, dal nome dell'autore, è un marchio di Telcordia Technologies, precedentemente noto come Bell Communications Research (Bellcore); è stato sviluppato

da Haller, Karn e Walden alla fine degli anni '80 sulla base dello schema dell'hash di Lamport. Di seguito viene illustrato tale protocollo e le sue debolezze.

Protocollo 6 (Protocollo S/KEY basato su OTP) *Sia U un utente che vuole accedere ad un sistema mediante un host H; si indica con*

- *ID_U e P_U, rispettivamente, l'identità di U e la password associata;*
- *f una funzione one-way utilizzata sia da U che da H per le operazioni indicate ai punti successivi;*
- *n un numero intero sufficientemente grande;*
- *c un contatore tale che $1 \leq c \leq n$;*
- *s un seme (seed) generato da H da fornire ad U per la generazione della sequenza di password con l'hash chain, usato come salt nella prima sequenza;*
- *S la concatenazione di P_U con il seed s, ovvero $S = P_U|s$;*
- *$(ID_U,\ f^c(S),\ c)$ la tripla costituita dall'identità di U, dall'esecuzione c volte della funzione f su S e dal contatore c, memorizzata in H per autenticare U.*

Lo schema del protocollo S/KEY basato su OTP, che permette ad U di essere autenticato da H tramite OTP password P_i, con $1 \leq i \leq n-1$, generate a partire da P_U con una hash chain, una volta stabilito il collegamento iniziale ed accordati sulle regole, è il seguente:

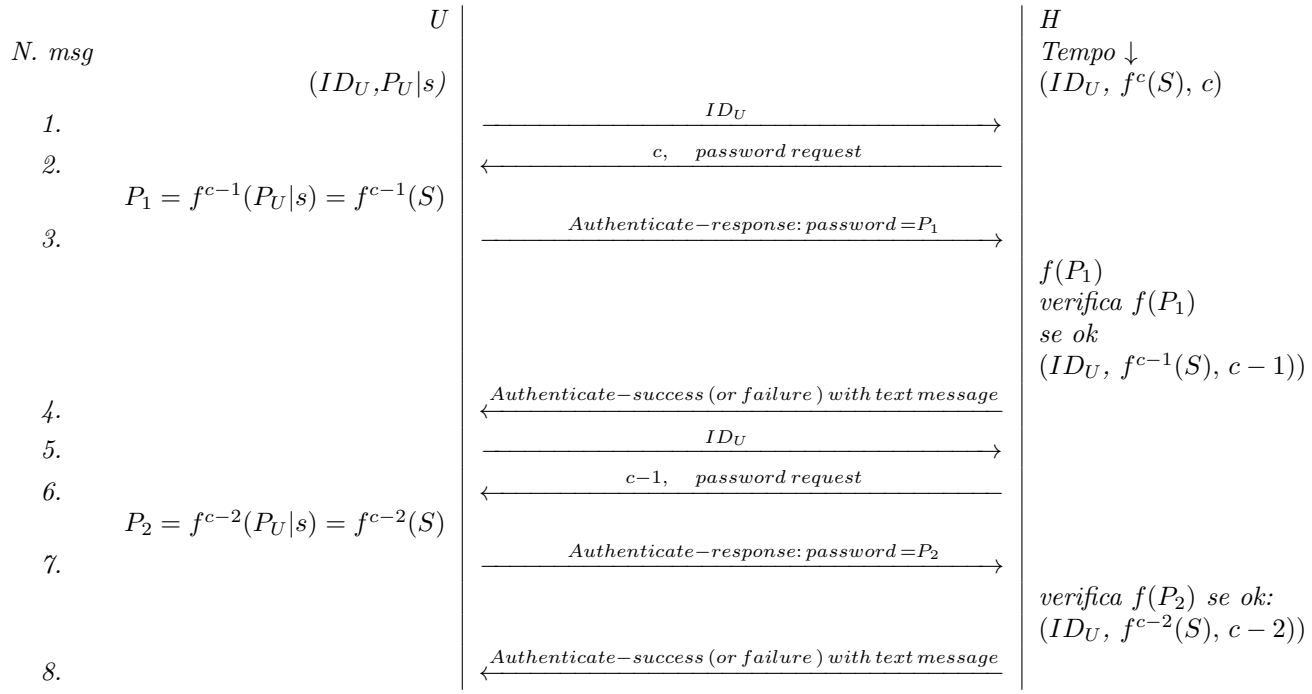


Diagramma 8: Diagramma di sequenza di un protocollo S/KEY basato su OTP .

Di seguito l'analisi dei messaggi tra U ed H:

- *U invia la sua identità, ID_U, ad H con il messaggio 1;*
- *H, ricevuta ID_U, invia ad U il contatore c e la richiesta di password per l'autenticazione con il messaggio 2;*
- *U calcola $P_1 = f^{c-1}(P_U|s) = f^{c-1}(S)$ e lo invia ad H con il messaggio 3;*
- *H calcola, quindi, la trasformazione della password ricevuta $P_1 = f^{c-1}(S)$ tramite la funzione f ovvero $f(P_1) = f(f^{c-1}(S)) = f^c(S)$ e verifica se la stringa risultante corrisponde a quella associata ad ID_U, ovvero $(ID_U,\ f^c(S), c\,)$ nel file delle password; in caso di corrispondenza riuscita, H sostituisce l'OTP memorizzato con quello fornito da U e decrementa il contatore nel suo archivio che ospiterà dunque $(ID_U,\ f^{c-1}(S), c-1\,)$ da utilizzare per il prossimo confronto; quindi H consente ad U di procedere con il login comunicando anche il successo dell'autenticazione con il messaggio 4 ; in caso di verifica negativa, invece, H invia solamente con il messaggio 4 la notifica del fallimento.*

Alla prossima richiesta di autenticazione, dopo lo scambio dei messaggi 5 e 6 analoghi, rispettivamente, ai messaggi 2 e 3, tenendo conto dell'incremento del contatore per la gestione della sincronizzazione,

- *U calcola $P_2 = f^{c-2}(S)$ e lo invia ad H con il messaggio 7;*
- *H calcola, quindi, la trasformazione della password ricevuta $P_2 = f^{c-2}(S)$ tramite la funzione f ovvero $f(P_2) = f(f^{c-2}(S)) = f^{c-1}(S)$ e verifica se la stringa risultante corrisponde a quella associata ad ID_U, ovvero $(ID_U,\ f^{c-1}(S), c-1\,)$ nel file delle password aggiornato con l'ulteriore computazione; in caso di corrispondenza riuscita, H sostituisce l'OTP memorizzato con quello fornito da U e decrementa il contatore nel suo archivio che ospiterà dunque $(ID_U,\ f^{c-2}(S), c-2\,)$ da utilizzare per il prossimo confronto; quindi H consente ad U di procedere con il login comunicando anche il successo dell'autenticazione con il messaggio 8; in caso di verifica negativa, invece, H invia solamente con il messaggio 8 la notifica del fallimento.*

Analogamente U ed H procedono per le successive richieste di autenticazione.

Di seguito lo schema chiarificatore della corrispondenza tra le password inviate da U e le verifiche condotte da H sul file, denominato *skeykeys*, aggiornato ad ogni iterazione con il decremento delle computazioni della funzione hash applicata ad S e del contatore; l'algoritmo di verifica si arresta consumata la $(c-1)$-esima password a disposizione di U in quanto per $c = n$ $P_c = f^0(S) = S$ non prevede alcuna trasformazione di S, e, terminata la verifica di P_{c-2}, il contatore c in H è arrivato ad 1.

U		H		
Sequenza password		Sequenza password	Verifica in	Tempo ↓
Password 1: $P_1 = f^{c-1}(S)$	$\longrightarrow$	$P_0 = f^c(S)$	(ID_U, P_0, c)	
Password 2: $P_2 = f^{c-2}(S)$	$\longrightarrow$	$P_1 = f^{c-1}(S)$	$(ID_U, P_1, c-1)$	
Password 3: $P_3 = f^{c-3}(S)$	$\longrightarrow$	$P_2 = f^{c-2}(S)$	$(ID_U, P_2, c-2)$	
$\dots$		$\dots$	$\dots$	
Password i: $P_i = f^{c-i}(S)$	$\longrightarrow$	$P_{i+1} = f^{c-i+1}(S)$	(ID_U, P_{c-i+1})	
$\dots$		$\dots$	$\dots$	
Password c-2: $P_{c-2} = f^2(S)$	$\longrightarrow$	$P_{c-3} = f^3(S)$	$(ID_U, P_{c-3}, 3)$	
Password c-1: $P_{c-1} = f(P_U)$	$\longrightarrow$	$P_{c-2} = f^2(S)$	$(ID_U, P_{c-2}, 2)$	

Diagramma 9: Diagramma di sequenza del OTP S/KEY con evidenza delle sequenze delle password utilizzate dalle parti.

Il sistema S/KEY è stato sviluppato per l'autenticazione nei sistemi operativi Unix-like; in particolare per i terminali stupidi ed i computer pubblici, considerati non affidabili dal punto di vista della sicurezza, sui quali non si desidera digitare una password a lungo termine; il fatto che la password sia utilizzata una sola volta costituisce un requisito importante per la sicurezza in tali postazioni.

La sicurezza di S/KEY, basata sull'OTP, si basa sulla difficoltà di invertire le funzioni hash crittografiche come illustrato; un avversario che riesca ad entrare in possesso di una password, $P_i = f^{n-i}(S)$, utilizzata per un'autenticazione andata a buon fine, è consapevole che la stessa non può essere utilizzata per un'ulteriore autenticazione e, pertanto, diventa di suo interesse scoprire la successiva $P_{i+1} = f(f^{n-i}(S)) = f^{n-i+1}(S)$ ovvero quella che sarà utilizzata nella prossima iterazione; tuttavia *l'inversione della funzione hash, responsabile dalla catena di produzione delle password, risulta estremamente "difficile" con le attuali funzioni hash crittografiche.*

L'utilizzo del seed s, generato da H e comunicato ad U, in genere con un ulteriore protocollo di sicurezza, permette di riutilizzare la stessa password più volte anche in presenza di utenti che hanno la stessa chiave; infatti tale seed viene concatenato a P_U in modo tale che l'applicazione dell'hash chain avvenga su $S = P_U|s$; *di fatto il seed è utilizzato come salt nella sequenza zero, allo scopo di rafforzare S/KEY contro gli attacchi a dizionario ed impedire, quindi, ad un avversario di ottenere la ripetizione della catena nel caso di riutilizzo della stessa password.*

Osservazione 6 (Debolezze del Protocollo S/KEY basato su OTP) *S/KEY è sicuramente un protocollo debole perché sensibile ai seguenti attacchi:*

- *attacco Man In The Middle (MITM) essendo basato su OTP con autenticazione non mutua in quanto H non si autentica sul client utilizzato da U e non genera un segreto (la connessione può essere dirottata);*
- *attacchi di modificazione e di replay ai danni del message sequence number, condotti, ad esempio, per la disincronizzazione di U ed H sulle password da utilizzare; infatti il contatore, inviato in chiaro, può essere intercettato e modificato*

da un avversario per alterare la sequenza di trasmissione delle credenziali a suo vantaggio;

- *attacco di forza bruta in determinate condizioni di esecuzione; tale attacco viene utilizzato rapidamente una volta che un avversario è riuscito ad individuare i caratteri iniziali di una sequenza di trasmissione considerando le sequenze utilizzate dal protocollo.*

Inoltre, analogamente a quanto visto per l'OTP, l'autenticazione è permessa solamente un numero prefissato di volte; esaurito tale numero, è necessariamente richiesta una reinizializzazione e, quindi, una comunicazione da parte di H della nuova password ad U con un canale ed un metodo sicuro da concordare.

Un ulteriore debolezza di S/KEY deriva dalla seguenti limitazioni:

- *è possibile utilizzare le funzioni hash $MD4$, $MD5$ o SHA-1 ma, in ogni caso, il risultato è ridotto a 64 bit con un operazione XOR;*
- *l'uso del salt in S/KEY è limitato solamente alle sequenza zero.*

Tali limitazioni sono sfruttate da un attacco, chiamato Dungeon, basato su un pratico compromesso spazio-tempo; tale attacco viene realizzato tenendo conto di disporre di uno spazio di memorizzazione e di risorse computazionali ragionevoli tali da poter condurre una serie di operazioni in un tempo inferiore a quello utilizzato da un utente U per consumare tutte le password OTP a disposizione nelle sequenza prevista dal server di autenticazione H. Un avversario memorizza un numero elevato N ma ragionevole a livello computazionale, scegliendo ad esempio, N con$2^{16} \leq N \leq 2^{48}$, delle 2^{64} possibili coppie chiave/valore hash utilizzabili per l'autenticazione allo scopo di condurre un attacco precalcolato; quindi, intercetta una password OTP, $P_j = f^{n-j}(S)$, con $1 \leq j \leq n-1$ inviata da U con un attacco di eavesdropping e ne controlla le presenza negli hash memorizzati nel suo archivio, tenendo conto anche del numero di sequenza nell'intercettazione; in caso di corrispondenza l'attaccante calcola gli hash successivi del valore corrispondente $P_i = f^{n-i}(P_j)$ ed invia la stessa ad H fingendosi U; la richiesta di autenticazione si ripete con $j < i \leq N-1$, finché l'avversario non riesce ad autenticarsi tenendo conto del decremento del contatore e delle possibili autenticazioni di U che "consumano" le OTP password disponibili.

Osservazione 7 (Possibili correzioni per la sicurezza di S/KEY) *Alcune possibili correzioni da apportare al protocollo S/KEY per evitare le gli attacchi descritti possono essere le seguenti:*

- *includere il seed e la sequenza in ogni round;*
- *utilizzare un hash più grande;*
- *utilizzare una Key Derivation Function (KDF) per la sequenza zero:*

S/KEY è supportato in molti sistemi Unix-like essendo possibile utilizzare un'implementazione generica open source; è anche implementato da OpenSSH fin dalla versione 1.2.2 rilasciata nel 1999; con la scadenza dei brevetti di base sulla crittografia a chiave pubblica e l'uso diffuso dei client che eseguono SSH e altri protocolli crittografici in grado di proteggere un'intera sessione, anziché solamente la password, S/KEY sta cadendo in disuso suppur ancora raccomandato nella FreeBSD security guide[5].

Gli attuali sistemi OTP superano le debolezze del protocollo S/KEY, utilizzando anche le cosiddette Time-based One-Time Password (TOTP), standardizzate dal IETF nel 2011 *con la RFC* 6238; *le password TOTP, a differenza di quelle OTP, scadono anche se non vengono utilizzate trascorso per un determinato periodo di tempo; pertanto l'impiego delle TOTP scoraggia gli attacchi di dizionario e a forza bruta non concedendo ad un avversario il tempo materiale per effettuare gli attacchi indicati. Le TOTP sono particolarmente popolari nell'ambito della cosiddetta Multifactor Authentication (MFA) realizzata dall'utente tramite un app o un dispositivo predisposto, detto "token", che fornisce all'utente la password da inserire; poiché chi accede all'app o entra in possesso del token può autenticarsi, per aumentare la sicurezza, è previsto che si debba utilizzare un Personal Identification Number, PIN, ovvero una sequenza di caratteri numerici usata solitamente per verificare l'utilizzatore del dispositivo, combinato con la TOTP*; sono possibili diverse strategie sulla gestione del PIN quali, ad esempio, il PIN accodato alle One Time Password, come password per il token e come parte del segreto base; *in ogni caso va evidenziato che il PIN è pensato come strumento complementare in quanto la sicurezza dell'implementazione del protocollo basato sull'OTP non è affidata totalmente a tale identificativo*; in genere, quindi, il PIN è un numero breve a 5 o 6 cifre.

3.2.8 Diffie-Helmann (DH) Key Exchange

Prima di trattare i successivi protocolli per l'autenticazione basati sulla conoscenza condivisa di una password comune, che fanno uso della condivisione di una chiave crittografica di sessione, è necessario introdurre un protocollo fondamentale per la sicurezza dello scambio della chiave tra le parti, denominato Diffie-Hellman key exchange.

Definizione 101 (Diffie-Hellman key exchange) *Il Diffie-Hellman key exchange è un protocollo crittografico che consente a due entità di stabilire una chiave condivisa e segreta utilizzando un canale di comunicazione insicuro (pubblico) senza la necessità che le due parti si siano scambiate informazioni o si siano incontrate in precedenza.*

Lo schema del protocollo Diffie-Hellman fu pubblicato nel 1976 nell'ambito di una collaborazione tra Whitfield Diffie e Martin Hellman e fu il primo metodo pratico per due interlocutori di accordarsi su un segreto condiviso (la chiave) utilizzando un canale di comunicazione non protetto. Il lavoro di Ralph Merkle, sulla distribuzione delle chiavi pubbliche, ed il suggerimento da parte di John Gill, di utilizzare il problema dei logaritmi discreti come base per la creazione di funzioni unidirezionali, servirono da linee guida per lo sviluppo dell'algoritmo.

[5] `http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/one-time-passwords.html`

Protocollo 7 (Diffie-Helmann (schema originale)) *Siano A e B due utenti che intendono concordare una chiave segreta senza trasmetterla in un canale insicuro. A e B concordano i seguenti valori pubblici:*

- *p un numero primo molto "grande" per considerare il relativo gruppo moltiplicativo degli interi modulo p, $\mathbb{Z}_p^*$, dove $|\,\mathbb{Z}_p^*\,| = p-1$;*
- *g un generatore di $\mathbb{Z}_p^*$ ovvero $< g >= \mathbb{Z}_{\text{|}}^*$ con $1 \leq g \leq p-1$.*

Siano inoltre $a, b \in \mathbb{Z}_p^*$ due valori segreti rispettivamente di A e B tali che

- *a è un numero casuale scelto da A tale che $1 \leq a \leq p-1$;*
- *b è un numero casuale scelto da B tale che $1 \leq b \leq p-1$;*

Lo schema del protocollo Diffie-Helmann (schema originale), con l'utilizzo dei gruppi ciclici, è il seguente:

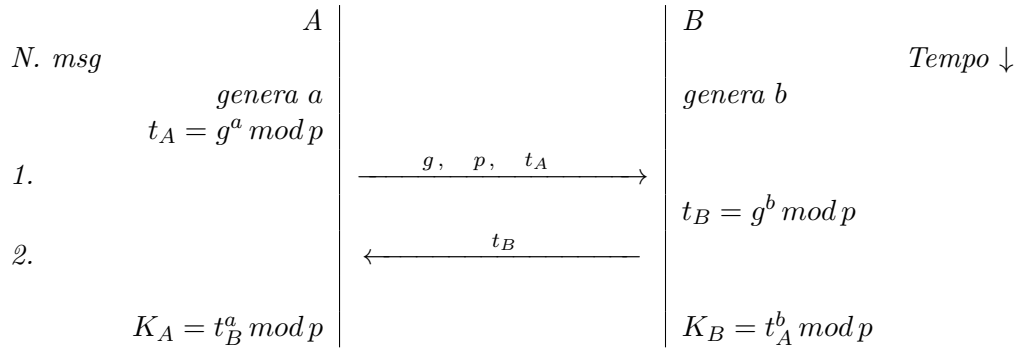


Diagramma 10: Diagramma di sequenza del protocollo Diffie-Helmann (schema originale) con l'utilizzo dei gruppi ciclici.

Di seguito l'analisi dei messaggi tra A ed B:

- *A invia g, p e $t_A = g^a \bmod p$, precedentemente calcolato, a B con il messaggio 1;*
- *B invia $t_B = g^b \bmod p$, precedentemente calcolato, a A con il messaggio 2;*

Ricevuto t_B, A calcola $K_A = t_B^a \bmod p$ ed analogamente fa B, calcolando $K_B = t_A^b \bmod p$. Per la proprietà di G ne deriva che, essendo $g^{b\,a} \bmod p = g^{a\,b} \bmod p$, si ha:

$$K_A = (g^b \bmod p)^a = g^{b\,a} \bmod p = g^{a\,b} \bmod p = (g^a \bmod p)^b = K_B$$

e dunque la chiave segreta concordata tra A e B, per cifrare le successive comunicazioni, sarà $K_{AB} = K_A = K_B$.

Il protocollo Diffie-Helmann è resistente agli attacchi di tipo eavesdropping; infatti se un avversario intercetta g, p, t_A e t_B avrà bisogno di ricavare i *numeri segreti* a e b generati e custoditi rispettivamente da A e B; *l'operazione di calcolo di a e b in*

realtà consiste nel calcolo di $\log t_A \,(\mathrm{mod}\, p)$ *e* $\log t_B \,(\mathrm{mod}\, p)$ *ovvero risolvere l'operazione di logaritmo discreto che come illustrato è un problema computazionalmente difficile.*
Lo schema di Diffie-Helmann può essere realizzato con operazioni e gruppi diversi da quelli previsti dal protocollo originario; può essere eseguito, ad esempio, sfruttando *l'ipotesi della fattorizzazione* o *della radice quadrata modulo il prodotto di una coppia di numeri primi.*

Di seguito viene illustrata una versione più generale del protocollo nella quale A e B concordano, prima degli invii, un sottogruppo G di $\mathbb{Z}_p^*$ di ordine q con q primo molto "grande".

Protocollo 8 (Diffie-Helmann (schema generale)) *Siano A e B due utenti che intendono concordare una chiave segreta senza trasmetterla in un canale insicuro. A e B concordano i seguenti valori pubblici:*

- *q un numero primo molto grande tale che sia l'ordine di un gruppo ciclico G, sottogruppo di $\mathbb{Z}_p^*$, con p primo, ovvero $|\,G\,| = q$;*
- *g un generatore di G, ovvero $< g >= G$, con $1 \leq g \leq q-1$.*

Siano inoltre $a, b \in G$ due valori segreti rispettivamente di A e B tali che

- *a è un numero casuale scelto da A tale che $1 \leq a \leq q$;*
- *b è un numero casuale scelto da B tale che $1 \leq b \leq q$;*

Lo schema del protocollo Diffie-Helmann, schema generale con i gruppi ciclici, che permette ad A e B di concordare una chiave segreta, senza trasmetterla in un canale insicuro, è il seguente:

	A		B
N. msg			*Tempo* $\downarrow$
	$g\,con < g >= G \lhd \mathbb{Z}_p^*$		$g\,con < g >= G \lhd \mathbb{Z}_p^*$
	$p\,primo\,, \lvert G\rvert = q$ *"grande"*		$p\,primo\,, \lvert G\rvert = q$ *"grande"*
	genera a		*genera* b
	$t_A = g^a$		$t_B = g^b$
1.		$\xrightarrow{g,\quad q,\quad t_A}$	
2.		$\xleftarrow{t_B}$	
	$K_A = t_B^a = g^{a\,b}$		$K_B = t_A^b = g^{b\,a}$

Diagramma 11: Diagramma di sequenza del protocollo Diffie-Helmann (schema generale) con l'utilizzo dei gruppi ciclici.

Di seguito l'analisi dei messaggi tra A ed B:

- *A invia t_A, precedentemente calcolato, a B con il messaggio* 1*;*

- *B invia t^B, precedentemente calcolato, a A con il messaggio* 2*;*
- *A calcola $K_A = t_B^a = (g^a)^b = g^{a\,b}$ e B calcola $K_B = t_A^b = (g^b)^a = g^{b\,a}$.*

Per la proprietà di G ne deriva che $K_A = g^{a\,b} = g^{b\,a} = K_B$ e, dunque, la chiave segreta concordata tra A e B, per cifrare le successive comunicazioni, sarà $K_{AB} = K_A = K_B$.

Si noti che *il generatore g del gruppo G, concordato da A e B, non ha importanza che sia "grande" a differenza, invece, del numero primo q e dei numeri segreti a e b che garantiscono la sicurezza del protocollo.*

La chiave ottenuta mediante il Diffie-Hellman key exchange può essere successivamente impiegata per cifrare le comunicazioni successive tramite uno schema di crittografia simmetrica.

Osservazione 8 (Debolezze del protocollo Diffie-Hellman) *Il protocollo Diffie-Hellman key exchange è suscettibile ad attacchi di tipo MITM a causa della mancanza di un meccanismo di autenticazione che permetta di stabilire l'autenticità delle chiavi. Lo schema seguente illustra il funzionamento dell'attacco:*

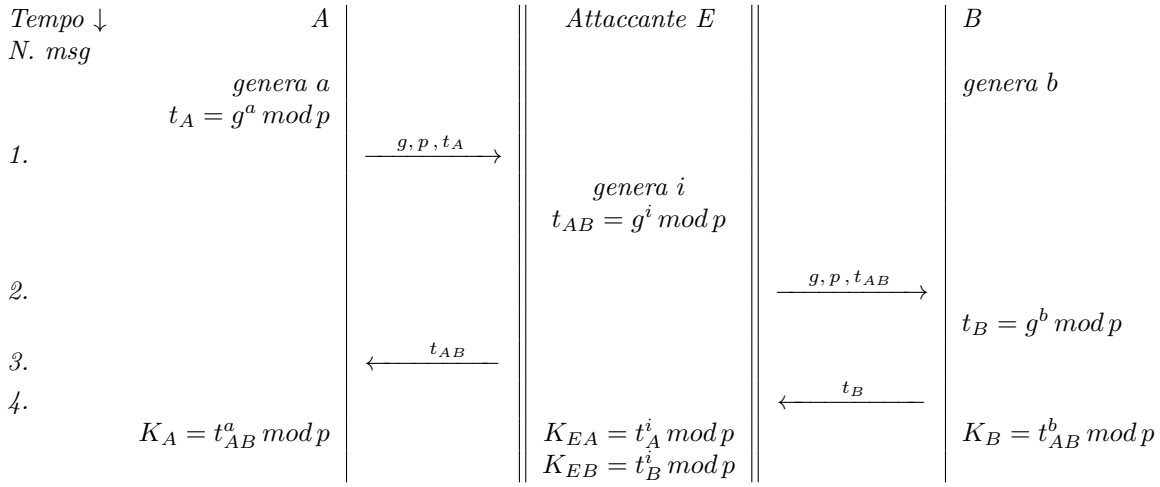


Diagramma 12: Diagramma di sequenza di un attacco al protocollo Diffie-Helmann.

L'attaccante E si interpone nella comunicazione tra A ed B e procede come segue:

- *A credendo di comunicare con B invia g, p e t_A, ad E con il messaggio* 1*;*
- *E, ricevuti g, p e t_A, genera, quindi, il numero casuale i e calcola $t_{AB} = g^i \bmod p$, con $i \leq p - 1$; invia, successivamente, g, p e t_{AB}, in luogo di t_A, a B, fingendosi A, con il messaggio* 2*;*
- *E, analogamente, spacciandosi per B, invia t_{AB} ad A con il messaggio* 3*;*
- *B, credendo di dialogare con A, invierà t_B ad E con il messaggio* 4*.*

A questo punto E è riuscito a derivare le due chiavi distinte K_{EA} e K_{EB} che gli permetteranno di avviare due canali cifrati indipendenti con A e B convinti di comunicare l'uno con l'altro essendo:

$$K_{EA} = t_A^i \, mod \, p = g^{a\,i} \, mod \, p = g^{i\,a} \, mod \, p = t_{AB}^a \, mod \, p = K_A$$
$$K_{EB} = t_B^i \, mod \, p = g^{b\,i} \, mod \, p = g^{i\,b} \, mod \, p = t_{AB}^b \, mod \, p = K_B$$

Altre debolezze dello schema Diffie-Hellman riguardano la vulnerabilità agli:

- *attacchi di tipo clogging*, ovvero ai casi in cui un attaccante sommerga una delle parti con richieste contenti i valori $t_A^{'}$ o $t_B^{'}$ ottenuti da computazioni del logaritmo discreto; in tal modo il ricevente è costretto a rispondere, calcolando la chiave di sessione per ogni richiesta, consumando quindi risorse di calcolo fino ad arrivare ad un disservizio nel protocollo;

- *attacchi di tipo replay* ovvero ai casi in cui un attaccante utilizzi i dati di una precedente comunicazione tra A e B per tentare di impersonare una delle parti.

Il Diffie-Hellman key exchange, seppur non sia un protocollo autenticato e vulnerabile agli attacchi illustrati, può essere combinato con altre tecniche per realizzare protocolli di autenticazione efficienti; per tale motivo è alla base di numerosi protocolli autenticati ed è usato anche in alcune modalità di funzionamento del protocollo TLS in seguito illustrato.

3.2.9 Encrypted Key Exchange (EKE) versione base del protocollo di Bellovin e Merritt

Il protocollo **Diffie-Helmann Encrypted Key Exchange (DH-EKE)** si propone di estendere il protocollo Diffie-Helmann introducendo l'autenticazione sulla chiave che si calcola.

Prima di esporre il protocollo DH-EKE si introducono alcune nozioni sulla classe dei protocolli Encrypted Key Exchange che derivano dalla versione base ideata da Bellovin e Merrit negli anni novanta.

Protocollo 9 (Encrypted Key Echange schema originale) *Siano A e B due utenti che intendono autenticarsi reciprocamente e concordare una chiave segreta di sessione in un canale insicuro. A e B concordano una password, P_{AB}, un algoritmo simmetrico la cui chiave sia $w = f(P_{AB})$, con f funzione hash MDC nota, ed un algoritmo asimmetrico. Siano*

- *(K_{As}^+, K_{As}^-) la coppia di chiavi rispettivamente pubbliche e private di sessione scelte da A per l'applicazione dell'algoritmo asimmetrico;*

- *$E_w(K_{As}^+)$ la cifratura della chiave pubblica di sessione K_{As}^+ di A con l'algoritmo simmetrico utilizzando w in comune con B e $D_w(E_w(K_{As}^+)) = K_{As}^+$ la relativa decifratura;*

- $E_w(E_{K^+_{As}}(K_{Bs}))$ *la cifratura con l'algoritmo simmetrico con* w *di* $E_{K^+_{As}}(K_{Bs})$*, cifratura, a sua volta, con l'algoritmo asimmetrico e* K^+_{As}*, della chiave di sessione di* B*,* K_{Bs}*;*
- $D_w(E_w(E_{K^+_{As}}(K_{Bs}))) = E_{K^+_{As}}(K_{Bs})$ *la decifratura con l'algoritmo simmetrico con* w *della cifratura* $E_w(E_{K^+_{As}}(K_{Bs}))$*;*
- $D_{K^-_{As}}(E_{K^+_{As}}(K_{Bs})) = K_{Bs}$ *la decifratura con l'algorimo asimmetrico e chiave* K^-_{As} *operata da* A *su* $E_{K^+_{As}}(K_{Bs})$*;*
- N_A *e* N_B *i nonce rispettivamente generati da* A *e* B*;*
- $E_{K_{Bs}}(N_A)$ *la cifratura di* N_A *con l'algoritmo simmetrico e chiave* K_{Bs} *e* $D_{K_{Bs}}(E_{K_{Bs}}(N_A)) = N_A$ *la relativa decifratura;*
- $E_{K_{Bs}}(N_A, N_B)$ *la cifratura della coppia di nonce* (N_A, N_B) *con l'algoritmo simmetrico e chiave* K_{Bs} *e* $D_{K_{Bs}}(E_{K_{Bs}}(N_A, N_B)) = (N_A, N_B)$ *la relativa decifratura;*
- $E_{K_{Bs}}(N_B)$ *la cifratura di* N_B *con l'algoritmo simmetrico e chiave* K_{Bs} *e* $D_{K_{Bs}}(E_{K_{Bs}}(N_B)) = N_B$ *la relativa decifratura.*

Lo schema del protocollo Encrypted Key Exchange originale è il seguente:

	A		B
Tempo $\downarrow$			
N. msg	$w = f(P_{AB})$ (K^+_{As}, K^-_{As})		$w = f(P_{AB})$
1.		$\xrightarrow{A\,,\quad E_w(K^+_{As})}$	$D_w(E_w(K^+_{As})) = K^+_{As}$ K_{Bs}
2.	$D_w(E_w(E_{K^+_{As}}(K_{Bs}))) = E_{K^+_{As}}(K_{Bs})$ $D_{K^-_{As}}(E_{K^+_{As}}(K_{Bs})) = K_{Bs}$ N_A	$\xleftarrow{B\,,\, E_w(E_{K^+_{As}}(K_{Bs}))}$	
3.		$\xrightarrow{E_{K_{Bs}}(N_A)}$	$D_{K_{Bs}}(E_{K_{Bs}}(N_A)) = N_A$ N_B
4.	$D_{K_{Bs}}(E_{K_{Bs}}(N_A, N_B)) = (N_A, N_B)$ *verifica* N_A	$\xleftarrow{E_{K_{Bs}}(N_A, N_B)}$	
5.		$\xrightarrow{E_{K_{Bs}}(N_B))}$	$D_{K_{Bs}}(E_{K_{Bs}}(N_B) = N_B$ *verifica* N_B

Diagramma 13: Diagramma di sequenza del protocollo Encrypted Key Exchange (EKE) versione base del protocollo di Bellovin e Merrit.

Di seguito l'analisi dei messaggi tra A ed B:

- *A invia la sua identità a B e* $E_w(K^+_{As})$ *ovvero la chiave pubblica di sessione cifrata, utilizzando l'algoritmo simmetrico e la chiave comune w, con il messaggio* 1*;*
- *B decifra* $E_w(K^+_{As})$*, utilizzando l'algoritmo simmetrico e w, ottenendo* $D_w(E_w(K^+_{As})) = K^+_{As}$ *ovvero la chiave pubblica di sessione di A;*
- *B sceglie una chiave di sessione* K_{Bs} *da utilizzare nell'algoritmo simmetrico in comune con A;*
- *B invia la sua identità a A e* $E_w(E_{K^+A}(K_{Bs})$ *ovvero la chiave di sessione* K_{Bs} *cifrata due volte di cui la prima con l'algoritmo asimmetrico di A e la sua chiave pubblica di sessione, precedentemente ottenuta,* K^+_{As}*, e la seconda con l'algoritmo simmetrico e w, con il messaggio* 2*;*
- *A decifra* $E_w(K^+_{Bs})$ *con l'algoritmo simmetrico e la chiave w ottenendo* $E_{K^+_{As}}(K_{Bs})$ *ovvero la chiave di sessione di* K_{Bs} *cifrata con la sua chiave pubblica di sessione,* K^+_{As}*; A può, quindi, applicare l'algoritmo asimmetrico con la sua chiave privata di sessione* K^-_{As} *per decifrare ulteriormente ottenendo finalmente la chiave di sessione* $D_{K^-_{As}}(E_{K^+_{As}}(K_{Bs})) = K_{Bs}$*;*
- *A genera un nonce* N_A *e lo invia a B cifrato con l'algoritmo simmetrico e la chiave di sessione* K_{Bs}*, ora nota ad entrambi, con il messaggio* 3*;*
- *B decifra il nonce* N_A *con l'algoritmo simmetrico e la chiave di sessione* K_{Bs} *e, a sua volta, genera un nonce* N_B*;*
- *B invia ad A i nonce* N_A *e* N_B *cifrati con l'algoritmo simmetrico e la chiave di sessione* K_{Bs} *tramite il messaggio* 4*;*
- *A decifra con l'algoritmo simmetrico e la chiave di sessione* K_{Bs}*,* $E_{K_{Bs}}(N_A, N_B)$ *ottenendo la coppia costituita dal suo nonce* N_A *inviato a B e quello ricevuto* N_B*;*
- *A verifica, quindi, l'identità di B controllando che il nonce* N_A *inviato a B sia effettivamente lo stesso che B ha inviato insieme al suo nonce* N_B*;*
- *A procede in maniera analoga per far verificare la sua identità a B ovvero invia il nonce* N_B *a B cifrato con con l'algoritmo simmetrico e la chiave di sessione* K_{Bs} *tramite il messaggio* 5*;*
- *B decifra, con l'algoritmo simmetrico e la chiave di sessione* K_{Bs}*,* $E_{K_{Bs}}(N_B)$*, ottenendo il nonce* N_B*, e verifica se effettivamente se tale nonce sia uguale a quello inviato ad A; in tal modo B verifica l'identità di A.*

Se, a partire dalla doppia decifrazione da parte di A, a seguito della ricezione del messaggio 2*, fino alla verifiche reciproche dei nonce da parte di A e B, tali operazioni hanno*

successo allora A e B riescono ad autenticarsi reciprocamente e ad utilizzare la chiave di sessione K_{Bs}.
Per ogni successiva autenticazione il protocollo prevede la ripetizione dei passi indicati ogni volta con una differente coppia di chiavi $(K^+_{As'}, K^-_{As'})$ da utilizzare nell'algoritmo asimmetrico ed, analogamente, una distinta chiave di sessione $K_{Bs'}$ per l'algoritmo simmetrico.

Va specificato che nella versione originale del protocollo di Bellovin e Meritt non sono indicati gli algoritmi di cifratura da utilizzare a differenza, invece, delle varianti.

L'innovazione del protocollo risiede nei primi passaggi che lo rendono resistente a possibili attacchi di tipo eavesdropping; infatti, come visto, nel messaggio 1 viene utilizzata la password condivisa tra A e B, w, per cifrare le chiave pubblica di sessione di K^+_{As} e la chiave di sessione K_{Bs} già cifrata con la chiave pubblica; in tal modo *la password w è statisticamente indipendente dai messaggi cifrati*: se un avversario intercettasse i messaggi 1 e 2 non otterrebbe informazioni sulla password poiché la chiave pubblica K^+_{As} e la chiave di sessione K_{Bs} sono stringhe casuali, di lunghezza maggiore rispetto a quella dalla password, che alla successiva chiamata del protocollo vengono cambiate.

Il cambio delle chiavi (K^+_{As}, K^-_{As}) e K_{Bs} ad ogni sessione del protocollo è fondamentale per evitare attacchi di forza bruta e di dizionario offline finalizzati ad individuare la chiave pubblica utilizzata per la cifratura e la necessaria chiave privata per ottenere w.

La cifratura della chiave pubblica, $E_w(K^+_{As})$, assicura una garanzia dell'autenticità della chiave e della sua provenienza supponendo che solamente A e B siano a conoscenza del segreto comune w; per ottenere la certezza della provenienza, di regola, si ricorre all'utilizzo dei certificati digitali emessi da un Certification Authority (CA) per le chiavi pubbliche. Nel caso del protocollo, poiché K^+_{As} deve essere cambiata ad ogni sessione, tale metodo sarebbe impraticabile; inoltre la chiave K^+_{As}, seppur definita "pubblica" per il suo utilizzo nell'algoritmo asimmetrico, in realtà non è pubblica perché è inviata cifrata a B e, dunque, solamente B può decifrala condividendo con A l'algoritmo simmetrico e la chiave w utilizzata.

L'utilizzo dei nonce N_A e N_B, stringhe casuali abbastanza lunghe di bit, oltre a complicare la decifratura dei messaggi intercettati da un avversario alla ricerca della chiave di sessione, è fondamentale per l'autenticazione reciproca di A e B potendo verificare ciascuno che il suo nonce generato sia lo stesso di quello restituito dall'altro partecipante; in tale modo A e B sono sicuri di poter comunicare tra loro e non con un avversario che si finge una delle parti; quindi eventuali attacchi attivi, con i quali potrebbero essere modificati i messaggi inviati nel corso dell'esecuzione del protocollo, sarebbero individuati da entrambi i partecipanti e porterebbero inevitabilmente alla sua conclusione.

3.2.10 Diffie Helmann - Encrypted Key Exchange (DH-EKE)

Protocollo 10 (Diffie Helmann - Encrypted Key Exchange) *Siano A e B due utenti che intendono autenticarsi reciprocamente e concordare una chiave segreta di sessione in un canale insicuro. A e B concordano una password, P_{AB}, un algoritmo simmetrico la cui chiave sia $w = f(P_{AB})$, con f funzione MDC nota, una funzione da utilizzare per ottenere la chiave K_s di sessione tramite lo schema di Diffie Helmann, e G un gruppo ciclico con g generatore del gruppo, scelto per utilizzare l'ipotesi del logaritmo discreto, tale che $G = \mathbb{Z}_p^*$, con p primo, e $p > 2^{64}$. Siano*

- $(K_{As}^+, K_{As}^-) = (g^a \bmod p, a)$, *con $a \in G$, la coppia di chiavi, rispettivamente pubbliche e private di sessione, scelte da A per l'applicazione dell'algoritmo asimmetrico utilizzando lo schema DH;*
- $E_w(K_{As}^+)$ *la cifratura della chiave pubblica di sessione K_{As}^+ di A con l'algoritmo simmetrico utilizzando la chiave w, in comune con B, e $D_w(E_w(K_{As}^+)) = K_{As}^+$ la relativa decifratura;*
- $(K_{Bs}^+, K_{Bs}^-) = (g^b \bmod p, b)$, *con $b \in G$, la coppia di chiavi, rispettivamente pubbliche e private di sessione, scelte da B per l'applicazione dell'algoritmo asimmetrico;*
- $E_w(K_{Bs}^+)$ *la cifratura della chiave pubblica di sessione K_{Bs}^+ di B con l'algoritmo simmetrico utilizzando la chiave w e $D_w(E_w(K_{Bs}^+)) = K_{Bs}^+$ la relativa decifratura;*
- N_A *e N_B i nonce rispettivamente generati da A e B, con $0 < N_A, N_B < 2^{64}$;*
- $E_{K_s}(N_B)$ *la cifratura del nonce generato da B con l'algoritmo simmetrico utilizzando la chiave di sessione K_s, stabilita con DH, e $D_{K_s}(E_{K_s}(N_B)) = N_B$ la relativa decifratura;*
- $E_{K_s}(N_A N_B)$ *la cifratura della coppia di nonce generati rispettivamente da A e B con l'algoritmo simmetrico utilizzando la chiave di sessione K_s e $D_{K_s}(E_{K_s}(N_A, N_B)) = (N_A, N_B)$ la relativa decifratura;*
- $E_{K_s}(N_A)$ *la cifratura del nonce generato da B con l'algoritmo simmetrico utilizzando la chiave di sessione K_s e $D_{K_s}(E_{K_s}(N_A)) = N_A$ la relativa decifratura;*

Lo schema del protocollo DH-EKE è il seguente:

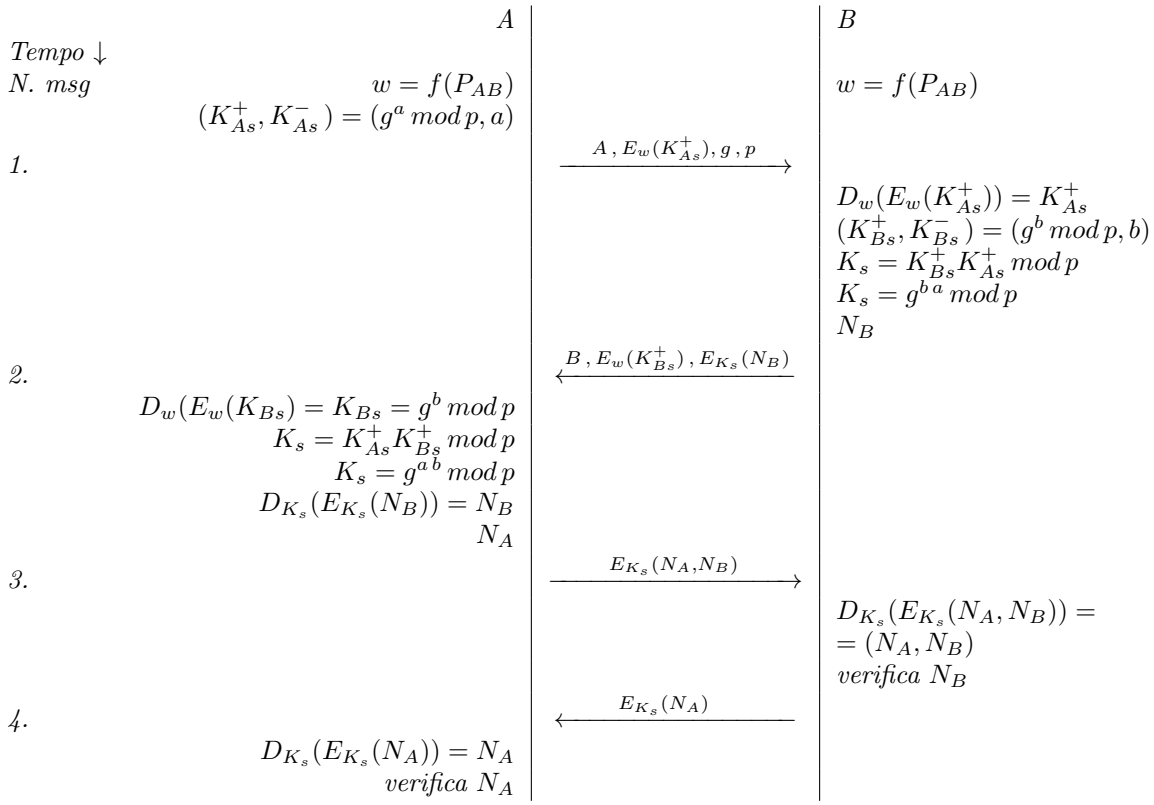


Diagramma 14: Diagramma di sequenza del protocollo Diffie Helmann - Encrypted Key Exchange

Di seguito l'analisi dei messaggi tra A ed B:

- *A, dopo aver generato la coppia di chiavi* (K^+_{As}, K^-_{As}) *scegliendo come chiave privata un elemento* $a \in \mathbb{Z}^*_p$ *e calcolato la sua chiave pubblica come* $g^a \bmod p$*, invia a B tale chiave cifrandola con l'algoritmo simmetrico e la chiave w unitamente al generatore g, il numero primo p scelti e la sua identità, utilizzando il messaggio* 1*;*

- *B decifra* $E_w(K^+_{As})$*, utilizzando l'algoritmo simmetrico e la password in comune w, ottenendo* $D_w(E_w(K^+_{As})) = K^+_{As}$ *ovvero la chiave pubblica di sessione di A;*

  B genera la coppia di chiavi (K^+_{Bs}, K^-_{Bs}) *scegliendo come chiave privata un elemento* $b \in \mathbb{Z}^*_p$ *e calcolando la sua chiave pubblica come* $g^b \bmod p$*; applicando lo schema DH, B è quindi in grado di calcolare la chiave di sessione comune,* $K_s = K^+_{Bs} K^+_{As} \bmod p$*, da utilizzare per lo scambio cifrato con A; B invia ad A la sua chiave pubblica,* K^+_{Bs}*, cifrata con l'algoritmo simmetrico e la chiave w, unitamente al nonce* N_B *generato, cifrato con la chiave di sessione* K_s*, ed alla sua identità utilizzando il messaggio* 2*;*

- *A decifra* $E_w(K^+_{Bs})$*, con l'algoritmo simmetrico e la chiave w ottenendo la chiave pubblica di sessione di B,* K^+_{Bs}*; utilizzando lo schema DH può quindi calcolare*

la chiave di sessione comune $K_s = K^+_{As} K^+_{Bs}$ *ed utilizzarla per decifrare* $E_{K_s}(N_B)$ *ottenendo il nonce* N_B *inviato da* B*; A, a sua volta, genera un nonce* N_A *ed invia la coppia* (N_A, N_B) *cifrata con l'algoritmo simmetrico e la chiave di sessione* K_s*, ora nota ad entrambi, con il messaggio* 3*;*

- *B decifra* $E_{K_s}(N_A, N_B)$ *con l'algoritmo simmetrico e la chiave di sessione* K_s *ottenendo la coppia* (N_A, N_B) *e verifica, quindi, l'identità di A controllando che il nonce* N_B *inviato a A sia effettivamente lo stesso di quello ricevuto insieme a* N_A*; infine B invia tale nonce ad A, cifrandolo al solito con l'algoritmo simmetrico e la chiave di sessione* K_s*, per l'analoga verifica, con il messaggio* 4*.*

- *A decifra con l'algoritmo simmetrico e la chiave di sessione* K_{Bs}*,* $E_{K_s}(N_A)$ *ottenendo il nonce* N_A *e verifica se effettivamente corrisponde a quello inviato ad A; in tal modo A verifica l'identità di B.*

Se a partire dalla decifrazione da parte di A ed al calcolo della chiave comune di sessione, K_s*, secondo lo schema DH, a seguito della ricezione del messaggio* 2*, fino alla verifiche reciproche dei nonce da parte di A e B, tali operazioni hanno successo allora A e B riescono ad autenticarsi reciprocamente ed ad utilizzare la chiave di sessione.*
Per ogni successiva autenticazione il protocollo si ripete con la scelta di una diversa coppia di chiavi, $(K^+_{As'}, K^-_{As'}) = (g^{a'} \bmod p, a')$ *e* $(K^+_{Bs'}, K^-_{Bs'}) = (g^{b'} \bmod p, b')$*, rispettivamente per A e B, da utilizzare nell'algoritmo simmetrico e nello schema di DH per ottenere una differente chiave di sessione* $K_{s'}$*.*

Osservazione 9 (Debolezze del protocollo DH-EKE) *Nell'implementazione del protocollo DH-EKE non è specificato l'algoritmo di cifratura simmetrico da usare con la chiave* $K_{s'}$*; originariamente, infatti, Bellovin e Meritt ritennero accettabile qualunque scelta dell'algoritmo; tuttavia gli stessi autori evidenziarono che il protocollo soffriva delle debolezze relative ai seguenti attacchi:*

- *attacchi di dizionario;*

- *attacchi di password chaining.*

Gli attacchi di password chaining prevedono che un avversario utilizzi la conoscenza della password corrente di un utente per apprendere le password future; di conseguenza, l'esposizione di una singola password compromette efficacemente tutte le succesive comunicazioni da parte di quell'utente.

Una soluzione per mettere in sicurezza il protocollo è stata proposta da Barry Jaspan nell'articolo *Dual-workfactor encrypted key exchange: efficiently preventing password chaining and dictionary attacks: Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6 July 1996*[6], con il quale si evidenzia che *il successo di determinare le password idonee per cercare di decifrare* $E_w(K^+_{As})$ *e* $E_w(K^+_{Bs})$*, inviate rispettivamente da A e B con i messaggi* 1 *e* 2 *e*

[6] `https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.28.3006&rep=rep1&type=pdf`

verificare se il risultato è un valore temporaneamente accettabile per lo schema Diffie-Helmann, dipende dall'algoritmo di cifratura utilizzato dal protocollo e dai parametri che definiscono il gruppo G da considerare.

Le cifrature previste nei messaggi 1 *e* 2 *sono fondamentali per la sicurezza del protocollo DH-EKE e non possono essere omesse come inizialmente suggerito dagli stessi Bellovin e Meritt; infatti, nel caso in cui non fossero applicate, il protocollo sarebbe vulnerabile ad attacchi di dizionario.*
A dimostrazione di quanto affermato, nel primo caso si consideri lo schema seguente:

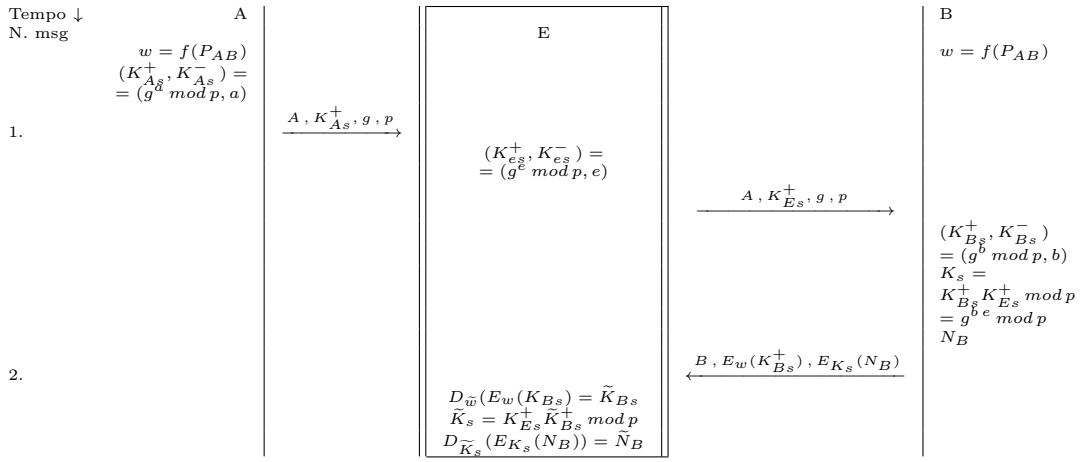


Diagramma 15: Diagramma di sequenza dei primi messaggi del DH-EXE, eseguito senza la cifratura della chiave pubblica di A con un avversario che si interpone.

Nel messaggio 1 non essendo cifrata la chiave pubblica K^+_{As}, un avversario E la intercetta con un attacco Man In The Midle (MITM) e, fingendosi A, genera la sua coppia di chiavi $(K^+_{es}, K^-_{es}) = (g^e \bmod p, e)$ ed invia K^+_{es} a B insieme a A, g e p. B ottiene, quindi, la chiave pubblica di sessione di E, a sua volta inviata in chiaro, convinto che sia di A; B genera la coppia di chiavi (K^+_{Bs}, K^-_{Bs}) scegliendo come chiave privata un elemento $b \in \mathbb{Z}^*_p$ e calcolando la sua chiave pubblica come $g^b \bmod p$; applicando lo schema DH, B è quindi in grado di calcolare la chiave di sessione comune, $K_s = K^+_{Bs} K^+_{Es} \bmod p$, da utilizzare per lo scambio cifrato con A ma in realtà con E; B invia ad A la sua chiave pubblica, K^+_{Bs}, cifrata con l'algoritmo simmetrico e la chiave w, unitamente al nonce N_B generato cifrato con la chiave di sessione compromessa da E, K_s, ed alla sua identità utilizzando il messaggio 2; E intercetta, quindi, il messaggio 2 diretto ad A e viene a conoscenza di $E_w(K^+_{Bs})$ e $E_{K_s}(N_B)$. Ora *l'obiettivo di E è decifrare entrambe le cifrature di B non conoscendo la password $w = f(P_{AB})$ ma potendo sfruttare le ridondanze delle informazioni cifrate ricevute*; E conduce, quindi, un *attacco di dizionario* su tali cifrature ipotizzando una password $\widetilde{w}$ per cui ottenere $D_{\widetilde{w}}(E_w(K_{Bs})) = \widetilde{K}_{Bs}$ e $D_{\widetilde{K}_s}(E_{K_s}(N_B)) = \widetilde{N}_B$ fino ad individuare un $\widetilde{w}$ tale che $\widetilde{K}_{Bs} = K_{Bs}$ e $\widetilde{N}_B = N_B$ ovvero la chiave pubblica di sessione di B ed il nonce spedito per la verifica dell'identità; *se l'attacco va a buon fine, E ha finalmente tutte le informazioni necessarie per continuare con successo il protocollo*

con B riuscendo a stabilire la chiave comune ed autenticarsi fingendosi A e, allo stesso tempo, ad effettuare lo stesso attacco ai danni di A spacciandosi per B.

Nel secondo caso, se non viene cifrata la chiave pubblica di B, K^+_{Bs}, nel messaggio 2 si consideri lo schema successivo:

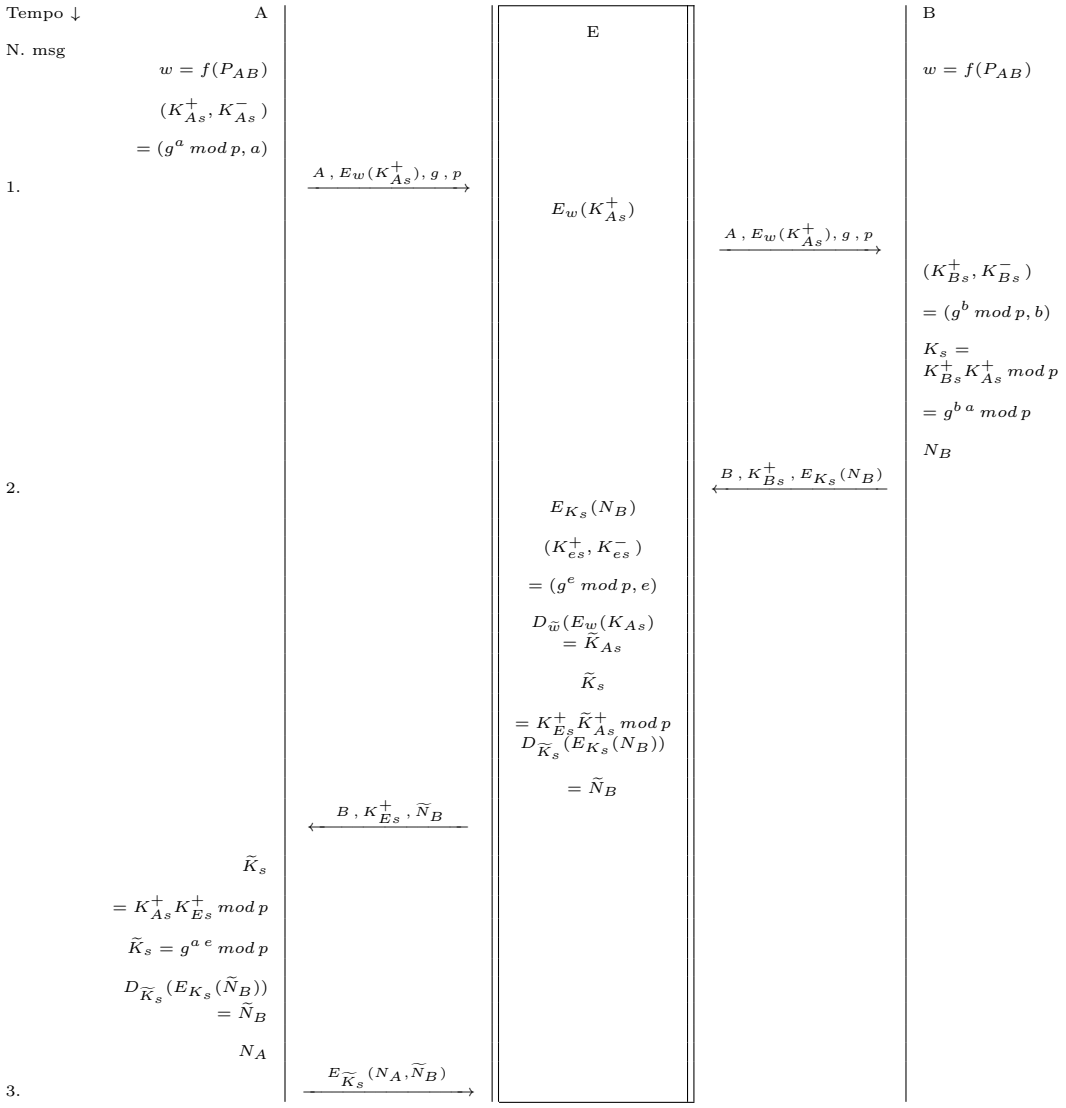


Diagramma 16: Diagramma di sequenza dei primi messaggi del DH-EXE, eseguito senza la cifratura della chiave pubblica di B con un avversario che si interpone.

Analogamente a quanto visto nel caso precedente, l'avversario E, che si interpone tra le parti, intercetta il messaggio 2 con la chiave pubblica K^+_{Bs} con un *attacco MITM* fingendosi A. Analizzando $E_{K_s}(N_B)$, ricevuto con lo stesso messaggio, cifrato con l'algoritmo simmetrico e la chiave di sessione K_s secondo lo schema DH, se lo stesso non contiene ridondanze, E può sfruttare tale debolezza per ingannare A simulando di essere

B; E genera casualmente una coppia di chiavi $(K^+_{Es}, K^-_{Es}) = (g^e \, mod \, p, e)$ e conduce un *attacco a dizionario* su $E_w(K^+_{As})$ ipotizzando una password $\widetilde{w}$ per cui ottenere $D_{\widetilde{w}}(E_w(K^+_{As})) = \widetilde{K}_{As}$ tale che $\widetilde{K}_{As} = K_{As}$ e, di conseguenza, poter ricavare la chiave di sessione $K_s = \widetilde{K}^+_{As} K^+_{Bs} \, mod \, p$ necessaria per decifrare $E_{K_s}(N_B)$ e conoscere N_B.
E genera un nonce $\widetilde{N}_B$ tale che $\widetilde{N}_B = E_{\widetilde{K}_s}(N_B)$ ed invia lo stesso ad A insieme alla sua chiave pubblica, K^+_{Es}, ed all'identità di B spacciandosi appunto per B.
Se il tentativo effettuato dall'attaccante va a buon fine, A calcola la chiave di sessione con la chiave pubblica da lui ricevuta, ottenendo $\widetilde{K}_s = g^{a\,e} \, mod \, p$, riuscendo così a decifrare $\widetilde{N}_B$; A invia, quindi, $\widetilde{N}_B$ cifrato con la chiave di sessione insieme al suo nonce N_A a E, credendo di comunicare con B con il messaggio 3; E, decifrando $E_{\widetilde{K}_s}(N_A, \widetilde{N}_B)$, ricevuto da A con la chiave di sessione, ha finalmente conferma della correttezza della password, $\widetilde{w}$, ipotizzata e, finalmente, ha tutte le informazioni necessarie per proseguire l'esecuzione del protocollo con successo, riuscendo a stabilire la chiave comune con A ed a autenticarsi fingendosi B.
Nel caso in cui, invece, la scelta di $\widetilde{w}$, tentata a seguito dell'attacco di dizionario non sia corretta, A fallirebbe la decifratura del nonce ricevuto da E a seguito dell'errata chiave di sessione ottenuta e, di conseguenza, l'esecuzione del protocollo sarebbe interrotta.

Dunque, in base a quanto approfonditamente argomentato, se un attaccante riesce ad indovinare la password comune utilizzata dalle parti nel protocollo DH-EKE, lo stesso è in grado di sostituirsi ad una delle due con un attacco attivo.

3.2.11 Protocolli basati su challenge-response

Si è visto che la prima fase di creazione di un canale sicuro fra due parti è generalmente quella dell'autenticazione ovvero la procedura che consente ad un'entità non solo di comunicare la propria identità all'altra ma di fornire una qualche prova che dimostri la correttezza dell'identità stessa. In breve, date le due parti in causa A e B, l'obiettivo del protocollo di autenticazione è quello di terminare con l'accettazione dell'identità di A da parte di B oppure con il rifiuto della stessa; *i protocolli di tipo challenge-response prevedono anche l'autenticazione mutua.* Di seguito si riporta una definizione.

Definizione 102 (Protocollo challenge-response) *Un protocollo crittografico di autenticazione si definisce di tipo challenge-response (autenticazione forte) se realizza i seguenti obiettivi:*

1. *date due entità oneste A e B, la prima deve riuscire ad autenticarsi alla seconda cioè B deve completare il protocollo con l'accettazione dell'identità di A;*

2. *B non deve poter usare le informazioni di identificazione di A per impersonarla di fronte ad una terza entità C;*

3. *la probabilità che una terza parte C, che esegua il protocollo usando il ruolo di A, riesca a farsi accettare da B come A, deve essere trascurabile (il significato di trascurabile dipende dall'applicazione);*

4. *l'obiettivo precedente deve essere realizzato anche nel caso in cui siano state osservate molte esecuzioni del protocollo da parte di un avversario C e siano possibili istanze simultanee del protocollo stesso.*

Dagli obiettivi stabiliti dalla definizione dei protocolli challenge-response ne deriva che l'idea alla base di tali protocolli è la dimostrazione da parte del soggetto che si vuole autenticare (claimant) di essere a conoscenza di una quantità segreta associata al soggetto stesso, senza però rivelarla al verificatore (verifier) durante l'esecuzione.
In pratica una delle due parti sceglie ed invia una quantità tempo-variante, il challenge, e l'altra fornisce come risposta un valore che dipende sia dal challenge ricevuto sia dal segreto associato alla propria identità, il response; in tal modo, anche se il mezzo di comunicazione è monitorato da un avversario, lo stesso non sarà in grado si ottenere informazioni utili per tentare di impersonare il soggetto, visto che il challenge varia da un'esecuzione all'altra.

In fase di progettazione di un protocollo crittografico, per ottenere gli obiettivi indicati dal challenge-response, bisogna considerare i possibili attacchi di tipo replay, interleaving, reflection, ritardo forzato e chosen-text che un avversario può operare; per difendersi da ciascun attacco sono previste delle contromisure che si possono riassumere nella tabella seguente:

Tipo di attacco	**Contromisura**
replay	*usare nonces; inserire l'identità del target nelle risposte*
interleaving	*collegare logicamente i messaggi di una stessa esecuzione del protocollo*
reflection	*inserire l'identificatore del target nelle response*; *usare messaggi di forme diverse (evitare simmetrie)*; *usare chiavi unidirezionali*
chosen-text	*inserire nelle rispote delle quantità di disturbo*
ritardo forzato	*inserire numeri casuali ed intervalli di time out brevi*; *usare marche temporali unite ad altre tecniche*

Dalla tabella si evidenzia che un protocollo, per difendersi dagli *attacchi di replay e interleaving*, deve poter distinguere un'esecuzione dalle altre; per tale ragione, come già visto nell'esposizione di alcuni protocolli trattati, si fa uso dell'inserimento dei nonces all'interno dei messaggi ovvero dei valori mai utilizzati in istanze precedenti.

Con la tabella successiva, in particolare, si evidenziano le varie tipologie di *nonce* e, per ognuna, le principali controindicazioni nell'utilizzo.

Tipo di nonce	**Controindicazione**
numero random	*il protocollo spesso deve essere composto da un messaggio in più*
numero di sequenza	*necessario memorizzare delle informazioni di stato che permettano di ricavare quali sono i numeri di sequenza accettabili*
timestamp	*necessità di avere orologi che non possano essere manomessi da un avversario e che siano sincronizzati in maniera sicura*

In merito ai protocolli *challenge-response*, saranno illustrati i protocolli con autenticazione forte basati su *algoritmi a chiave simmetrica (crittografia a chiave privata)*, su *Message Authentication Code (MAC)*, su *algoritmi a chiave asimmetrica (crittografia a chiave pubblica)* e su un *Key Distribution Center (KDC)*.

3.2.12 Protocollo challenge-response basato su algoritmi a chiave simmetrica (crittografia a chiave privata)

Tra i possibili protocolli challenge-response quelli basati su algoritmi crittografici a chiave simmetrica sono spesso usati nei sistemi che hanno delle limitate capacità di calcolo in quanto non efficienti dal punto di vista computazionale e anche perché necessitano generalmente di chiavi di dimensione limitata. Nel caso di un numero limitato di utenti, le chiavi possono essere precaricate o, altrimenti, distribuite da un server fidato sempre attivo.

Di seguito si illustrano tre semplici protocolli di autenticazione challenge-response basati su crittografia a chiave simmetrica in base ai quali si possono costruire protocolli più complessi. I protocolli rispettano lo standard ISO/IEC 9798-2.

Nella trattazione si continuerà ad utilizzare la notazione E_K per indicare la crittografia a chiave simmetrica con la chiave K condivisa tra le entità A e B.

Protocollo 11 (Challenge-response con autenticazione unilaterale e timestamp) *Siano A e B due entità che condividono una chiave K ed un algoritmo simmetrico. Siano inoltre t_A un timestamp generato da A e $E_K(t_A, B)$ la cifratura di t_A e B con tale algoritmo con la chiave K operata da A.*

Lo schema challenge-respose che prevede l'autenticazione unilaterale con il timestamp, è il seguente:

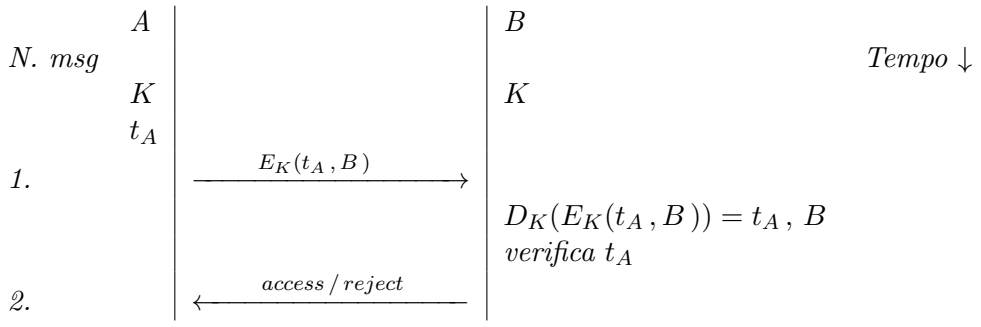


Diagramma 17: Diagramma di sequenza di un protocollo challenge response con autenticazione unilaterale e timestamp.

Di seguito l'analisi dei messaggi tra A ed B:

- *A cifra con l'algoritmo simmetrico e la chiave K il timestamp t_A insieme all'identificatore B, necessario per evitare che un avversario lo riutilizzi immediatamente su A con una reflection, e lo invia a B con il messaggio* 1*;*

- *B decifra* $E_K(t_A\,,B\,)$ *con l'algoritmo simmetrico e la chiave* K*, e verifica la freshness del messaggio con il timestamp* t_A *ricevuto; in base all'esito del controllo del timestamp* B *accetta o rifiuta l'autenticazione di* A *con il messaggio* 2*.*

Protocollo 12 (Challenge-response con autenticazione unilaterale e nonce) *Siano* A *e* B *due entità che condividono una chiave* K *ed algoritmo simmetrico. Siano inoltre* r_B *un nonce generato da* B *e* $E_K(r_B\,,B\,)$ *la cifratura di* r_B *e* B *con tale algoritmo con la chiave* K *operata da* A*.*

Lo schema challenge-respose che prevede l'autenticazione unilaterale con il nonce, è il seguente:

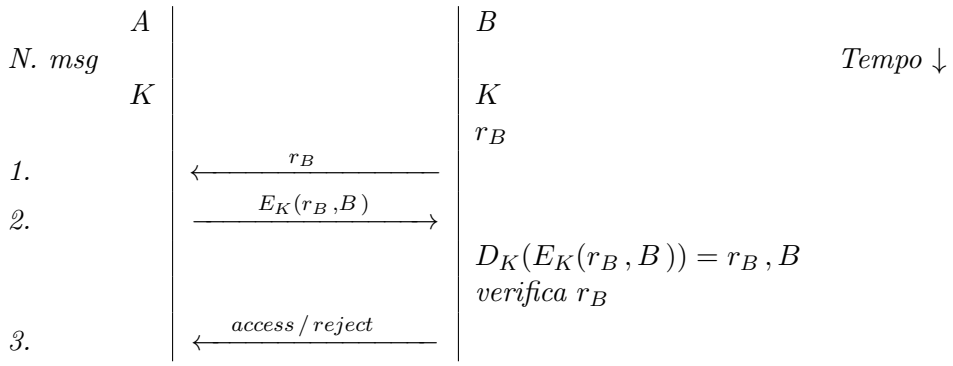


Diagramma 18: Diagramma di sequenza di un protocollo challenge response con autenticazione unilaterale e nonce.

Di seguito l'analisi dei messaggi tra A *ed* B*:*

- B *genera un nonce* r_B *e lo invia a* B *con il messaggio* 1*;*
- A *cifra, con l'algoritmo simmetrico e la chiave* K*, il nonce ricevuto e l'identificatore di* B *e li invia ad* A *con il messaggio* 2*;*
- B *decifra* $E_K(r_B\,,B\,)$*, con l'algoritmo simmetrico e la chiave* K*, e verifica il nonce* r_B *ricevuto; in base all'esito del controllo del nonce* B*, verificando anche che l'identificatore contenuto sia il suo per evitare attacchi di reflection, accetta o rifiuta l'autenticazione di* A *con il messaggio* 3*.*

Protocollo 13 (Challenge-response con mutua autenticazione e nonce) *Siano* A *e* B *due entità che condividono una chiave* K *ed algoritmo simmetrico. Siano inoltre* r_A *e* r_B *i nonce generati rispettivamente da* A *e* B*, e* $E_K(r_A\,,r_B\,,B\,)$ *la cifratura di* r_A*,* r_B *e* B *con tale algoritmo mediante la chiave* K *operata da* A *e* $E_K(r_B\,,r_A\,)$ *la cifratura di* r_B *e* r_A *effettuata da* B*.*

Lo schema challenge-respose che prevede l'autenticazione mutua con i nonce, è il seguente:

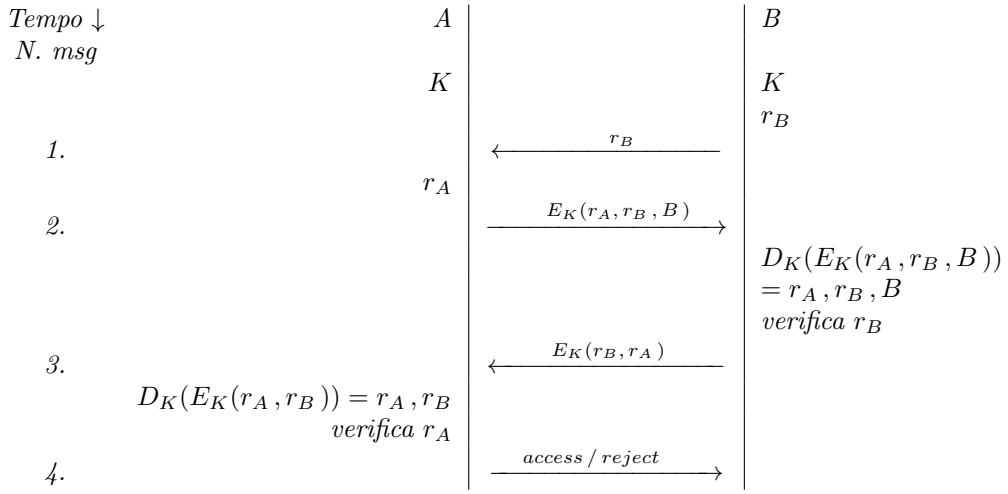


Diagramma 19: Diagramma di sequenza del protocollo challenge-response con autenticazione mutua e nonce.

Di seguito l'analisi dei messaggi tra A ed B:

- *B genera un nonce r_B e lo invia a A con il messaggio 1;*
- *A cifra, con l'algoritmo simmetrico e la chiave K, il nonce ricevuto r_B, il suo nonce r_A ed l'identificatore di B e li invia a B con il messaggio 2;*
- *B decifra $E_K(r_A, r_B, B)$, con l'algoritmo simmetrico e la chiave K, e verifica il nonce r_B ricevuto; in base all'esito del controllo del nonce B, assicurandosi anche che l'identificatore contenuto sia il suo per evitare attacchi di reflection, verifica l'autenticazione di A; in caso affermativo genera un nonce r_A e lo invia cifrato insieme a quello ricevuto e verificato, r_B, a B con il messaggio 3 per consentire la sua autenticazione ad A;*
- *A decifra $E_K(r_B, r_A)$, con l'algoritmo simmetrico e la chiave K, e verifica il nonce r_A ricevuto; in base all'esito del controllo del nonce A può accettare o rifiutare l'autenticazione di B con il messaggio 4.*

Si fa notare che la mutua autenticazione prevista dal protocollo challenge-response con l'utilizzo dei nonce è più conveniente rispetto all'esecuzione di due singole istanze, in quanto le identificazione sono collegabili ad una singola esecuzione.

Protocollo 14 (Challenge-response a due vie con mutua autenticazione) *Siano A e B due entità che condividono una chiave K ed algoritmo simmetrico. Siano inoltre r_A e r_B i nonce generati rispettivamente da A e B, $E_K(r_B)$ la cifratura di r_B con tale algoritmo mediante la chiave K operata da A e $E_K(r_A)$ la cifratura di r_A effettuata da B in modo analogo.*

Lo schema challenge-respose a due vie che prevede l'autenticazione mutua con i nonce, è il seguente:

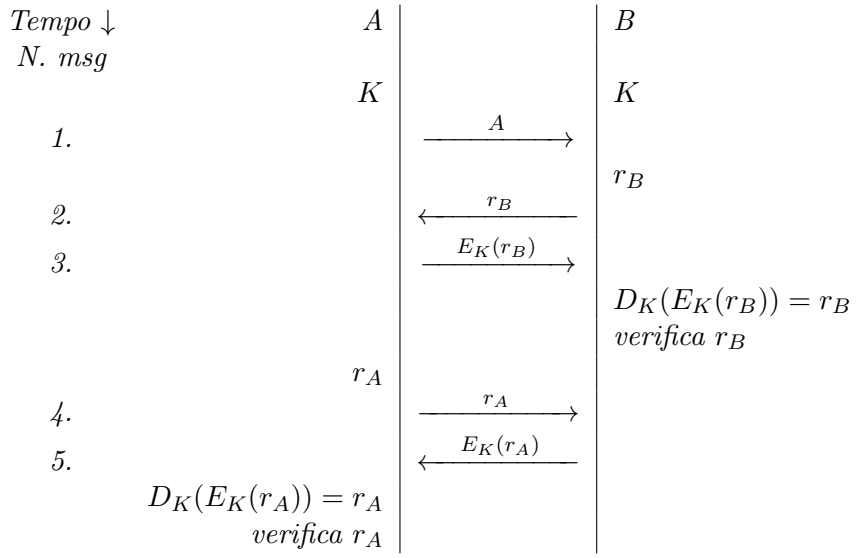


Diagramma 20: Diagramma di sequenza di un protocollo challenge-response a due vie con mutua autenticazione e nonce basato su algoritmo a chiave simmetrica.

Di seguito l'analisi dei messaggi tra A ed B:

- *A invia la sua identità a B con il messaggio* 1*;*
- *B invia un challenge r_B ad A con il messaggio* 2*;*
- *A cifra il challenge ricevuto, con l'algoritmo simmetrico e la chiave K, e invia $E_K(r_B)$ a B con il messaggio* 3*;*
- *B decifra $E_K(r_B)$, con l'algoritmo simmetrico e la chiave K, e verifica se il nonce r_B ricevuto corrisponde a quello inviato; in base all'esito del controllo del nonce B può autenticare A.*
- *A invia, a volta, un challenge r_A ad B con il messaggio* 4*;*
- *A decifra $E_K(r_A)$, con l'algoritmo simmetrico e la chiave K, e verifica se il nonce r_A ricevuto corrisponde a quello inviato; in base all'esito del controllo del nonce anche A può autenticare B.*

Allo scopo di evidenziare la difficoltà nella progettazione di protocolli di autenticazione challenge-response che realizzino realmente gli scopi previsti da un'autenticazione forte si illustra il seguente protocollo che rappresenta una modifica del precedente realizzata con l'uso di soli tre messaggi anziché di cinque; si vedrà che *l'effetto dell'ottimizzazione, in realtà, espone il protocollo al grave rischio di un attacco basato sulla riflessività.*

Protocollo 15 (Challenge-response semplificato con mutua autenticazione) *Siano A e B due entità che condividono una chiave K ed algoritmo simmetrico. Siano inoltre r_A e r_B i nonce generati rispettivamente da A e B, $E_K(r_B)$ la cifratura di r_B con tale algoritmo mediante la chiave K operata da A e $E_K(r_A)$ la cifratura di r_A effettuata da B in modo analogo.*

Lo schema challenge-response del protocollo, semplificato con l'uso di soli 3 messaggi, che prevede l'autenticazione mutua con i nonce, è il seguente:

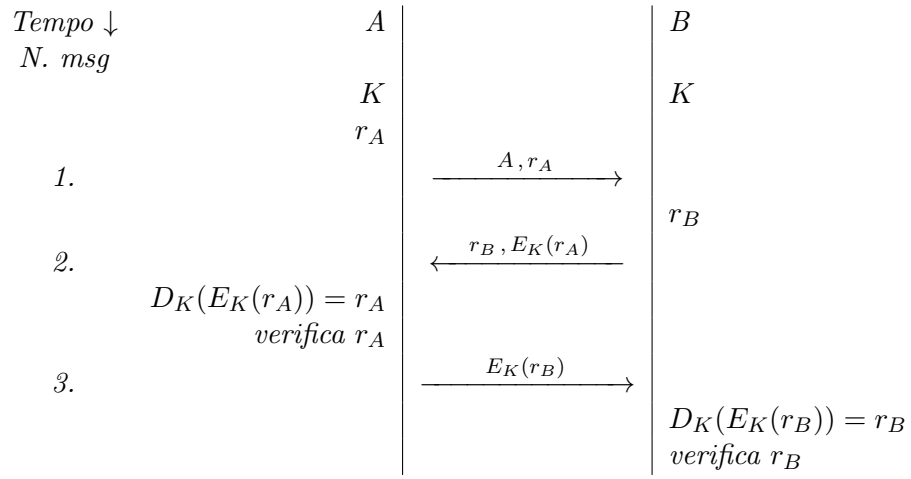


Diagramma 21: Diagramma di sequenza di un protocollo challenge-response semplificato con l'uso di soli 3 messaggi, che prevede l'autenticazione mutua con i nonce, basato su algoritmi a chiave simmetrica.

Di seguito l'analisi dei messaggi tra A ed B:

- *A genera il challenge r_A e lo invia con la sua identità a B tramite il messaggio* 1*;*
- *B, a sua volta, genera un challenge r_B e lo invia a B in chiaro insieme al challenge r_A, cifrato con l'algoritmo simmetrico e la chiave K, con il messaggio* 2*;*
- *A decifra il challenge ricevuto, con l'algoritmo simmetrico e la chiave K, e verifica se il nonce r_A corrisponde a quello inviato; in base all'esito del controllo B può autenticare A.*
- *A invia, a volta, a B il challenge r_B, cifrandolo con l'algoritmo simmetrico e la chiave K, con il messaggio* 3*;*
- *A decifra $E_K(r_B)$, con l'algoritmo simmetrico e la chiave K, e verifica se il nonce r_B ricevuto corrisponde a quello inviato; in base all'esito del controllo del nonce anche B può autenticare A.*

Osservazione 10 (Debolezze del protocollo challenge-response semplificato) *Come anticipato il protocollo precedente è affetto dalla grave debolezza dovuta all'essere vulnerabile ad un attacco di riflessione; infatti sia C un avversario che ha l'obiettivo di*

instaurare un canale di comunicazione tra A e B in modo tale che A creda di comunicare con B; C può realizzare l'obiettivo se risponde correttamente ad un challenge inviato da B restituendo la versione codificata di un nonce inviato da B. L'inganno ad opera di C viene effettuato senza che lo stesso avversario sia a conoscenza della chiave K, condivisa tra A e B.

Lo schema dell'attacco di riflessione condotto dall'avversario C ai danni di B è il seguente.

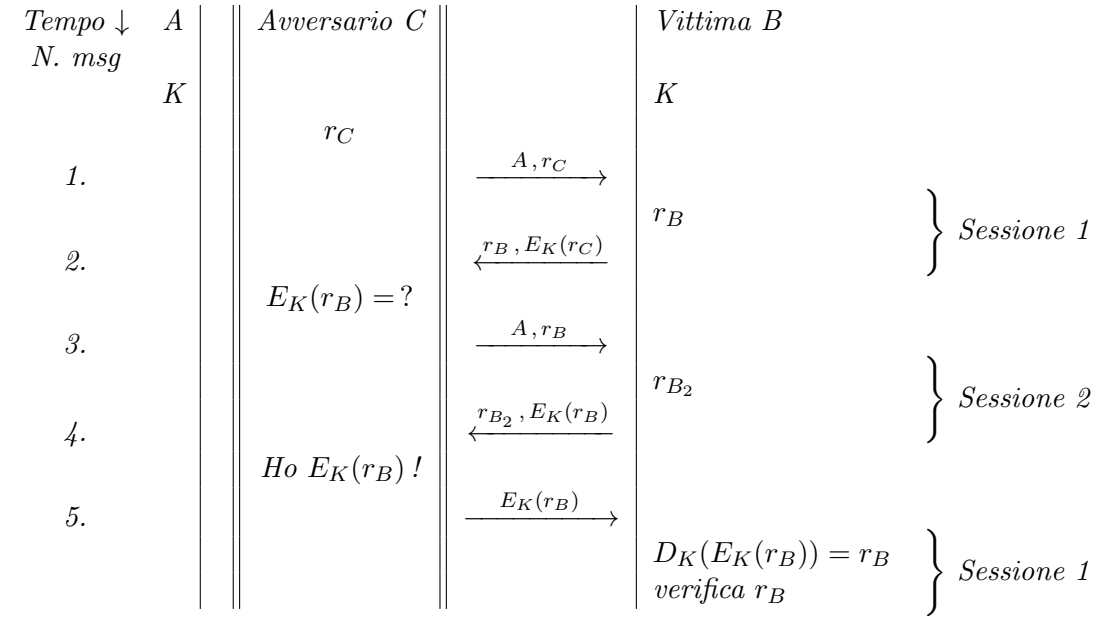


Diagramma 22: Diagramma di sequenza di un attacco di riflessione ai danni di un partecipante che esegue un protocollo challenge-response semplificato con l'uso di soli 3 messaggi basato su algoritmi a chiave simmetrica.

Di seguito l'analisi dei messaggi tra l'attaccante C e la vittima B:

- *C, spacciandosi per A, invia a B l'identità di A ed il nonce r_C con il messaggio* 1*;*
- *B genera il nonce r_B ed invia il nonce ricevuto, r_C, cifrato con l'algoritmo simmetrico e la chiave K a C, all'attaccante convinto di comunicare con A, con il messaggio* 2*;*
- *C, ricevuto r_c, dovrebbe dimostrare a B di saper cifrare restituendo $E_k(r_B)$ ma C non è a conoscenza della chiave K; pertanto, per avviare un attacco di riflessione, C inizia una seconda sessione, inviando a B l'identità di A ed il nonce r_B con il messaggio* 3*;*
- *B, non rendendosi conto di avere usato lui stesso in precedenza r_B come challenge, invia a C, sempre convinto di comunicare con A, $E_K(r_B)$ ed un altro challenge r_{B_2}, con il messaggio* 4*;*

- *C ha, quindi, ottenuto $E_K(r_B)$ e può inviare il response richiesto da B nella prima sessione restituendo $E_K(r_B)$ a B, fornito dalla stessa vittima, con il messaggio 5;*

- *B è quindi in grado di decifrare il challenge cifrato e da lui stesso fornito all'avversario, credendo che provenga da A; di conseguenza, la vittima autentica C, che si è spacciato per A, essendo l'attacante riuscito a concludere con successo il challenge-response con l'attacco di riflessione pur non conoscendo la chiave K che B condivide con A.*

L'attacco di riflessione condotto ai danni del protocollo illustrato ha avuto successo perché una delle due parti ha usato lo stesso challenge in due diverse esecuzioni del protocollo. Una possibile contromisura alla debolezza del protocollo consiste nell'usare sempre challenge differenti per l'iniziatore ed il risponditore.
Ad esempio, se A avesse usato sempre un numero pari e B uno dispari, B si sarebbe accorto che stava accedendo qualcosa di sospetto nel ricevere r_B nel messaggio 3; in ogni caso anche tale soluzione sarebbe soggetta ad altri attacchi quali il più volte evidenziato MITM.

In generale, per limitare le debolezze nella progettazione di un protocollo, è preferibile che non vengano eseguite operazioni identiche dalle due parti che stanno instaurando un canale sicuro.

Un altro principio violato nel protocollo semplificato riguarda l'invio della preziosa informazione da parte di B, ovvero il response al challenge $E_K(r_c)$, senza avere autenticato il suo interlocutore; tale principio, invece, non era violato nel protocollo originale in cui è prevista la dimostrazione dell'identità di A prima dell'invio di operazioni cifrate.

Osservazione 11 (Debolezze del protocollo challenge-response a due vie)
Anche la versione del protocollo descritto a due vie che utilizza cinque messaggi è vulnerabile ad un attacco di riflessione; infatti sia C un avversario che ha l'obiettivo di instaurare un canale di comunicazione tra A e B in modo tale che A creda di comunicare con B.
Lo schema dell'attacco di riflessione condotto da C ai danni di A instaurando, come in precedenza, due sessioni è il seguente.

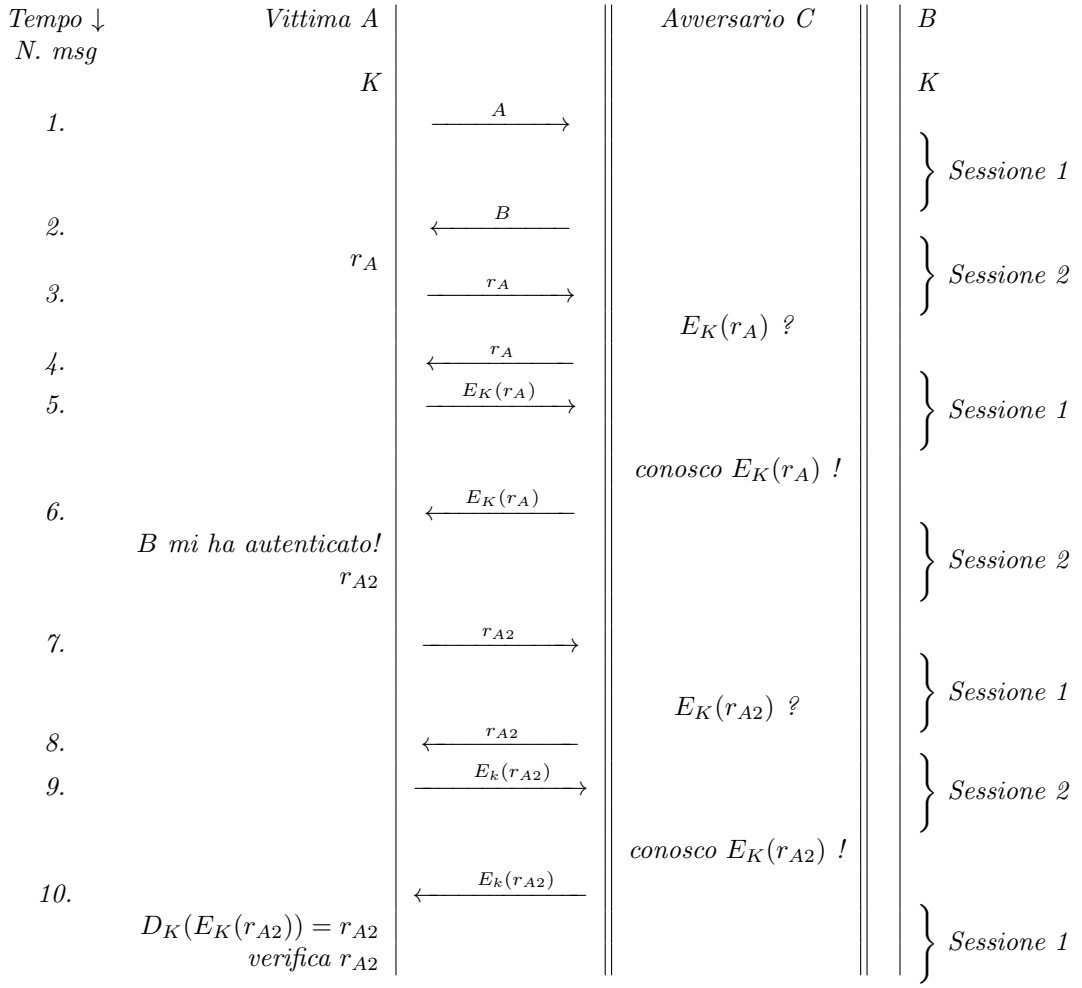


Diagramma 23: Diagramma di sequenza di un attacco di riflessione ai danni di un partecipante che esegue un protocollo challenge-response a due vie che utilizza cinque messaggi basato su algoritmi a chiave simmetrica.

Di seguito l'analisi dei messaggi tra la vittima A e l'attaccante C:

- *A invia a C, che si finge B, la sua l'identità di A con il messaggio* 1*;*
- *C inizia una seconda sessione inviando ad A l'identità di B con il messaggio* 2*;*
- *A, in conseguenza della seconda sessione iniziata da C, genera un challenge r_A e lo invia a C, credendo di comunicare con B, con il messaggio* 3*;*
- *C approfitta del r_A, ricevuto nella seconda sessione, per reinviarlo ad A, continuando la prima sessione con il messaggio* 4*;*
- *A, seguito dell'attacco di riflessione condotto da C, cifra, con l'algoritmo simmetrico e la chiave K, il challenge da lui stesso inviato e lo invia a C, continuando la prima sessione, con il messaggio* 5*;*

- *C ha così ottenuto $E_K(r_A)$ da A stesso senza conoscere la chiave K; può così inviare ad A il response della seconda sessione, $E_K(r_A)$, con il messaggio* 6*;*
- *A, credendo di essere autenticato da B, genera il challenge r_{A2} per permettere all'altro partecipante di completare l'autenticazione; A, quindi, invia all'altra parte, ignaro che sia l'attaccante, il challenge per il proseguo della prima sessione, con il messaggio* 7*;*
- *C utilizza la seconda sessione per ingannare nuovamente A inviando il challenge ricevuto nella prima sessione, r_{A2}, con il messaggio* 8*;*
- *A cifra, con l'algoritmo simmetrico e la chiave K, il challenge da lui stesso inviato e lo invia a C completando la seconda sessione, con il messaggio* 9*;*
- *C ricevuto $E_K(r_{A2})$ al termine della seconda sessione può, quindi, completare la prima inviando lo stesso ad A con il messaggio* 10*.*

  L'avversario è, dunque, riuscito a portare a termine l'attacco di riflessione: ha completato la prima sessione del protocollo senza conoscere la chiave K facendosi autenticare dalla vittima A, convinta di aver verificato il challenge inviato e di aver instaurato una comunicazione con B.

Generalizzando, in un protocollo di tipo challenge-response basato su un algoritmo a chiave simmetrica, per evitare gli attacchi descritti, si devono rispettare le seguenti regole generali:

1. *un partecipante che inizia l'autenticazione deve provare l'identità dell'altra parte prima di fornire informazioni che possono compromettere la sua sicurezza*;
2. *i partecipanti devono utilizzare per ogni sessione chiavi differenti*;
3. *i partecipanti devono utilizzare come challenge numeri scelti da insiemi diversi*;
4. *i partecipanti non possono riutilizzare le stesse informazioni, ottenute nel corso di una sessione, in una ulteriore sessione parallela per la prosecuzione del protocollo.*

3.2.13 Protocollo challenge-response basato su Message Authentication Code (MAC)

Un'altra tipologia di protocolli di autenticazione di tipo challenge-response è quella basata su Message Authentication Code, MAC, standardizzata dall'ISO/IEC 9798-4. Con riferimento a tale tipologia si tratterà un protocollo che fa uso di HMAC (Hash-based Message Authentication Code) ovvero una modalità per l'autenticazione dei messaggi MAC basata su una funzione di hash che richiede anche una chiave segreta; tramite HMAC è infatti possibile garantire sia l'integrità che l'autenticità di un messaggio.

Come si evince dalla seguente definizione, *HMAC utilizza una combinazione del messaggio originale ed una chiave segreta per la generazione del codice; inoltre non è legata ad alcuna funzione di hash e, pertanto, tale caratteristica rende possibile la sostituzione della funzione utilizzata nel caso in cui la stessa non venga ritenuta più sicura.*

Definizione 103 (Funzione HMAC) *Siano m il messaggio da autenticare, H una funzione hash iterata che usa una funzione di compressione che opera su blocchi di B byte ($B = 64$ per la maggior parte delle funzioni hash), i cosiddetti initialization vector, K la chiave segreta e condivisa dell'autenticazione del messaggio delle due parti e $K^{'}$ la chiave derivata da K così costruita:*

- *se K ha lunghezza uguale a B byte coincide con K;*
- *se K ha lunghezza minore di B byte sono aggiunti degli zeri per rendere la lunghezza di esattamente B byte;*
- *se K ha lunghezza maggiore di B byte si calcola il valore hash di K, ovvero $H(K)$, per ottenere una nuova chiave $K^{'} = H(K)$ di L byte (dove L è la lunghezza dell'output della funzione hash utilizzata).*

Formalizzando:

$$K^{'} = \begin{cases} K & se\,|K| = B \\ K + padding\ di\ zeri & se\,|K| < B \\ H(K) & se\,|K| > B \end{cases}$$

Siano, inoltre, "ipad" e "opad" due differenti stringhe di B byte definite come segue:

$$ipad = il\ byte\ 0x36\ ripetuto\ B\ volte$$
$$opad = il\ byte\ 0x5c\ ripetuto\ B\ volte$$

Si definisce una funzione $\mathrm{HMAC}(K, m)$*, applicata alla chiave K ed al messaggio m, la seguente trasformazione operata dalla funzione hash H*

$$\mathrm{HMAC}(K, m) = H\Big((K' \oplus \mathrm{opad}) \parallel H\Big((K' \oplus \mathrm{ipad}) \| m\Big)\Big)$$

dove $\|$ indica la concatenazione posizionale, $(K' \oplus \mathrm{opad}) \| H\Big((K' \oplus \mathrm{ipad}) \| m\Big)$ *equivale a* $H\Big((K' \oplus \mathrm{ipad}) \| m\Big)\Big)$ *accodato a* $(K' \oplus \mathrm{opad})$ *e $\oplus$ denota l'operatore XOR.*

Alla base della costruzione di una funzione HMAC sussistono i seguenti obiettivi:

- l'utilizzo di funzioni hash disponibili senza effettuare modifiche, con particolare riferimento a quelle il cui codice è distribuito gratuitamente;
- la conservazione delle prestazioni originali della funzione hash;
- l'utilizzo della chiave in modo semplice;
- la disponibilità di una buona e comprensibile analisi crittografica della forza del meccanismo di autenticazione basata su ragionevoli assunzioni che sono alla base della funzione hash utilizzata;
- la facilità della sostituzione della funzione hash utilizzata nell'eventualità che sia disponibile un'altra più sicura e che garantisca maggiori prestazioni.

Protocollo 16 (Protocollo challenge-response basato su HMAC) *Siano A e B due entità che condividono una chiave K ed una funzione HMAC. Siano inoltre r_A e r_B i nonce generati rispettivamente da A e B, $HMAC(r_A, r_B, A, B, K)$ la cifratura ottenuta applicando la funzione HMAC a r_A, r_B, A e B con la chiave K operata da A e $HMAC(r_A, r_B, A, B, K)$ la cifratura ottenuta applicando la stessa funzione a r_A, r_B e la chiave K da parte di B.*
Lo schema di un protocollo challenge-response basato su HMAC che prevede l'autenticazione mutua con i nonce e la funzione HMAC, è il seguente:

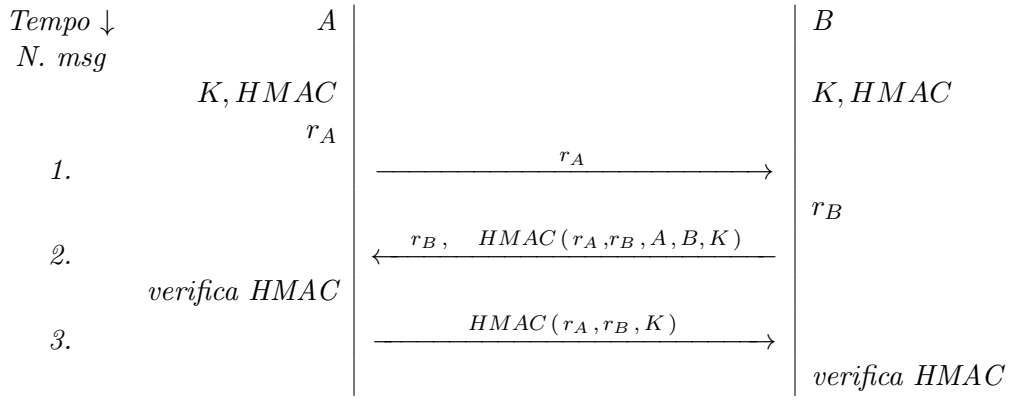


Diagramma 24: Diagramma di sequenza di un protocollo challenge-response basato su HMAC.

Di seguito l'analisi dei messaggi tra A ed B:

- *A genera il challenge r_A e lo invia in chiaro a B con il messaggio* 1*;*
- *B, a sua volta, genera un challenge r_B e lo invia in chiaro ad A insieme alla cifratura ottenuta applicando la funzione HMAC a r_A, r_B e B con la chiave K, tramite il messaggio* 2*;*
- *A, ricevuti r_B e la trasformazione HMAC, condividendo la chiave K con B, può calcolare a sua volta $HMAC(r_A, r_B, A, B, K)$ e verificare se il valore è identico a quello ottenuto da B; in base all'esito del controllo A può autenticare B.*
- *Verificata l'identità di B, A invia a B la cifratura ottenuta applicando la funzione HMAC a r_A, r_B con la chiave K, con il messaggio* 3*;*
- *B, ricevuto $HMAC(r_A, r_B, K)$, condividendo la chiave K e la trasformazione HMAC con A, calcola a sua volta la stessa trasformazione e verifica se corrisponde a quella ottenuta da A; in base all'esito del controllo anche B può, quindi, autenticare A.*

Negli argomenti dalla funzione HMAC utilizzata nei messaggi 2 e 3 previsti dal protocollo sono inclusi dei valori scelti dalla destinazione, ovvero r_A, fornito da A a B con il messaggio 1, nel primo caso e r_B, fornito da B ad A con il messaggio 2, nel secondo

caso; in tal modo un avversario C non ha la possibilità di ingannare una delle due parti fingendosi l'altra poiché la verifica della corrispondenza dei MAC calcolati sia da A che da B non darebbe esito positivo; dunque *il protocollo, così progettato, risolve il problema dell'attacco di riflessione applicato all'autenticazione a due vie.*

A differenza della firma digitale che utilizza un protocollo a chiave pubblica, HMAC usa una chiave privata condivisa; dunque nel protocollo di autenticazione illustrato la conoscenza di tale chiave è fondamentale per la generazione del messaggio in HMAC da inviare all'altro partecipante e, di conseguenza, per la sua autenticazione; *l'utilizzo della chiave condivisa nel protocollo, però, non assicura una forma di autenticazione abbastanza forte da garantire il non ripudio*; infatti di fronte ad una terza entità, entrambe le parti che condividono la chiave possono ripudiare il messaggio ed accusare l'altro di esserne l'autore.

Da quanto esposto è evidente che *la sicurezza del protocollo dipende dalla funzione HMAC*; per quanto riguarda la scelta della chiave comune da applicare nella trasformazione si possono fare le seguenti considerazioni:

- l'utilizzo di una una chiave con una lunghezza inferiore a L byte (dove L è la lunghezza dell'output della funzione hash utilizzata) è sconsigliato poiché diminuirebbe la sicurezza della funzione hash;
- la scelta, invece, di una chiave di lunghezza maggiore di L byte non incrementa in modo significativo la robustezza della funzione.

Ovviamente, come per gli altri protocolli, la chiave deve essere scelta casualmente, tenuta segreta e cambiata periodicamente da entrambi gli utenti che la condividono.
Poiché solo i primi t byte del risultato di HMAC sono utilizzati come MAC, la lunghezza, come indicato, costituisce un elemento di sicurezza da tenere in considerazione.

Per indicare la realizzazione di HMAC utilizzando la funzione hash H e l'output di t-bit si utilizza la seguente notazione

HMAC-H-t

Ad esempio,

- HMAC-SHA1-96 indica un HMAC realizzato utilizzando SHA-1 con l'output troncato a 96 bit;
- HMAC-MD5 utilizza tutti i 128 bit.

Per garantire un'adeguata sicurezza del HMAC, la lunghezza di t non dovrebbe essere minore di $L/2$; la RFC 2104 *raccomanda* $t \geq 80$.
In ogni caso va detto che se da una parte il troncamento ha il vantaggio di offrire una minore informazione ad un avversario, dall'altra, invece, permette un numero minore di byte da predire.

Come già evidenziato, è necessario essere particolarmente attenti nella scelta di una funzione hash H da usare nel HMAC in quanto dalla sua sicurezza dipende la sicurezza dell'HMAC e dello stesso protocollo nel quale viene utilizzata; dunque, nel caso in cui siano note delle debolezze, la funzione hash va immediatamente sostituita.

Osservazione 12 (Attacchi alle funzioni hash) *Data una funzione di hash H, gli attacchi possono essere classificati, fondamentalmente, nelle seguenti tipologie:*

- *attacchi alla preimmagine: dato un hash h, trovare il messaggio m tale che $H(m) = h$ (violazione del principio di non invertibilità);*
- *attacchi alla seconda preimmagine: dato un messaggio m_1, trovare un messaggio m_2 tale che $H(m_1) = H(m_2)$ (violazione del principio di resistenza debole delle collisioni);*
- *rilevamento delle collisioni: trovare due messaggi m_1, m_2 tali che $H(m_1) = H(m_2)$ (violazione del principio di resistenza forte alle collisioni).*

L'analisi di sicurezza degli algoritmi hash e, di conseguenza, i gli attacchi, si sono concentrati soprattutto sulle *tecniche di rilevamento delle collisioni.*
Partendo dalla considerazione che se una funzione hash ha n bit di output esistono 2^n hash possibili e, per il *principio della piccionaia*, trovare una collisione richiede di valutare al massimo $2^n + 1$ messaggi, utilizzando il seguente attacco *risulta possibile scoprire una collisione in modo più semplice rispetto a quanto si può ipotizzare ovvero, in media, effettuare metà dei tentativi possibili, cioè* $2^n/2 = 2^{n-1}$.
L'attacco che consente di ridurre i tentativi è denominato *attacco del compleanno* ed è basato sul cosiddetto *teorema del compleanno* definito da Richard Von Mises nel 1939.

Teorema 15 (Paradosso del compleanno) *Dato un gruppo di p persone, il verificarsi dell'evento*
A_p={almeno due persone compiono gli anni nello stesso giorno nel gruppo di p persone} ha la seguente probabilità:

$$P(A_p) = 1 - \underbrace{\frac{364}{365} \cdot \frac{363}{365} \cdot \ldots \cdot \frac{365 - p + 1}{365}}_{p-1}$$

Il teorema del compleanno viene anche definito *paradosso del compleanno* in quanto i calcoli che derivano dalla probabilità stabilita sono largamente superiori all'intuito; infatti per un gruppo di $n = 23$ persone la probabilità risulta leggermente superiore al 50%; con 30 persone supera il 70%, con 50 persone arriva al 97% e con almeno 366 persone (367 se si considera l'anno bisestile) si arriva all'evento certo.
Per effettuare il calcolo, si ricorre alla formula della probabilità condizionata con l'assunzione, per semplificare, che gli anni siano tutti composti da 365 giorni: aggiungere il giorno bisestile peggiora leggermente la probabilità, ma aumenta il fatto che i compleanni non siano equiprobabili. Se si considera l'evento A_p={*almeno due persone compiono gli anni nello stesso giorno nel gruppo di p persone*} ed il suo complementare A_p^C={*nessuno compie gli anni lo stesso giorno nel gruppo di p persone*} ne deriva che

$$P(A_p^C) = \underbrace{\frac{364}{365} \cdot \frac{363}{365} \cdot \ldots \cdot \frac{365 - p + 1}{365}}_{p-1}$$

e dunque la probabilità che si verifichi A_p, ovvero che esistano almeno due compleanni uguali, è

$$P(A_p) = 1 - P(A_p^C) = 1 - \underbrace{\frac{364}{365} \cdot \frac{363}{365} \cdot \ldots \cdot \frac{365-p+1}{365}}_{p-1}$$

Il paradosso del compleanno ha importanti ricadute nella sicurezza della funzione hash utilizzata in HMAC ed, in particolare, nella proprietà di "resistenza forte alle collisioni"; con un ragionamento analogo, infatti, si può dimostrare la validità del *paradosso formulato in termini di collisioni.*

Teorema 16 (Paradosso del compleanno - in termini di collisioni) *Sia A un insieme con p elementi ovvero $|A| = p$. Se gli elementi sono prelevati da A a caso, in media si troverà una collisione con probabilità migliore di $\frac{1}{2}$ dopo aver selezionato un numero $Q \approx 1.17\sqrt{p}$ elementi.*

Dal teorema precedente risulta chiara la definizione dell'attacco di compleanno ai danni di una funzione hash.

Definizione 104 (Attacco del compleanno) *L'attacco del compleanno ai danni di una funzione hash H è un attacco a forza bruta che prevede l'analisi esaustiva dei possibili messaggi casuali alla ricerca di una collisione con una probabilità migliore del* 50% *sfruttando i principi matematici del paradosso del compleanno nella teoria delle probabilità.*

In base a quanto illustrato, *se un attaccante riusce a condurre con successo un attacco ad HMAC, lo stesso può computare l'output della funzione di compressione e le collisioni nella funzione hash anche quando l'initialization vector è casuale e sconosciuto.*
Va, infine, osservato che *anche i risultati intermedi ($K^{'} \oplus ipad$) e ($K^{'} \oplus opad$) della trasformazione $HMAC(K, m)$ possono essere precomputati; tali risultati possono essere conservati e usati per inizializzare H ogni volta che bisogna autenticare un messaggio con la chiave K; pertanto vanno protetti alla stessa stregua del valore della chiave segreta.*

3.2.14 Protocollo challenge-response basato su algoritmi a chiave asimmetrica (crittografia a chiave pubblica)

I protocolli di autenticazione challenge-response basati su algoritmi asimmetrici prevedono che ciascun partecipante possieda una coppia di chiavi, composta da una chiave pubblica ed una privata, che si rappresenteranno con la consueta notazione (K^+, K^-); quindi *per l'autenticazione mutua è necessaria una doppia coppia di chiavi*; ovviamente solo la chiave pubblica, K^+, viene condivisa tra le parti.
In alternativa alle chiavi possono essere usati dei *certificati digitali* generati da una terza parte fidata, detta *Certification Authority* (CA), a ciascun partecipante.

Gli algoritmi utilizzati in tali protocolli prevedono la cifratura/decifratura con le rispettive *chiavi pubbliche* e *firme digitali* dei partecipanti ed, inoltre, la dimostrazione

della conoscenza da parte del claimant della chiave segreta decifrando un challenge cifrato con la chiave pubblica o firmando un challenge con la chiave privata.

Di seguito si illustra un protocollo challenge-response che utilizza algoritmi asimmetrici con coppia di chiavi (K^+, K^-).

Protocollo 17 (Protocollo challenge-response basato su chiave asimmetrica) *Siano A e B due partecipanti che intendono autenticarsi reciprocamente e concordare una chiave segreta di sessione in un canale insicuro. Siano*

- (K_A^+, K_A^-) *e* (K_B^+, K_B^-) *le coppie di chiavi pubbliche e private scelte rispettivamente da A e B per l'applicazione dell'algoritmo asimmetrico;*
- r_A *e* r_B *i challenge rispettivamente generati da A e B;*
- $E_{K_B^+}(A, r_A)$ *la cifratura dell'identità di A e del challenge r_A, generato da A, con la chiave pubblica di B, K_{Bs}^+, utilizzando l'algoritmo asimmetrico;*
- $E_{K_A^+}(r_A, r_B, K_{ABs})$ *la cifratura del challenge r_A, ricevuto da A, del challenge r_B, generato da B, e della chiave di sessione, K_{ABs}, generata da B, utilizzando l'algoritmo asimmetrico;*
- $E_{K_{ABs}}(r_B)$ *la cifratura del challenge r_B, con la chiave comune di sessione K_{ABs};*

Lo schema del protocollo challenge-response basato su algoritmi a chiave asimmetrica è il seguente:

Tempo ↓	A		B
N. msg			
	(K_A^+, K_A^-)		(K_B^+, K_B^-)
	r_A		
		$\xrightarrow{E_{K_B^+}(A, r_A)}$	
1.			$D_{K_B^-}(E_{K_B^+}(A, r_A))$
			$= (A,\ r_A)$
			r_B
			K_{ABs}
		$\xleftarrow{E_{K_A^+}(r_A, r_B, K_{ABs})}$	
2.			
	$D_{K_A^-}(E_{K_A^+}(r_A, r_B, K_{ABs})$		
	$= (r_A, r_B, K_{ABs})$		
	verifica r_A		
		$\xrightarrow{E_{K_{ABs}}(r_B)}$	
3.			
			$D_{K_{ABs}}(E_{K_{ABs}}(r_B)) = r_B$
			verifica r_B

Diagramma 25: Diagramma di sequenza di un protocollo challenge-response basato su algoritmi a chiave asimmetrica.

Di seguito l'analisi dei messaggi tra A ed B:

- *A genera un challenge r_A e lo invia a B insieme alla sua identità, cifrandolo utilizzando l'algoritmo asimmetrico e la chiave pubblica dello stesso B, K_B^+, con il messaggio* 1*;*
- *B decifra $E_{K_B^+}(A, r_A)$, utilizzando l'algoritmo asimmetrico e la sua chiave privata K_B^-, ottenendo così il challenge r_A e l'identità di A; B, a sua volta, genera un challenge r_B ed una chiave di sessione K_{ABs} che invia cifrate insieme al challenge r_A, ad A usando la sua chiave pubblica, K_A^+, con il messaggio* 2*;*
- *A decifra con la sua chiave privata $E_{K_B^+}(A, r_A)$ ottenendo il challenge r_A precedentemente inviato, il challenge r_B e la chiave di sessione, K_{ABs}, da utilizzare per l'invio successivo come chiave privata di un algoritmo condiviso con B; A verifica, quindi, l'identità di B controllando che il challenge r_A, inviato a B, sia effettivamente lo stesso che B ha restituito insieme al suo r_B e la chiave di sessione; se la verifica del challenge va a buon fine, A può autenticare B e procedere alla response del challenge r_B a B, cifrandolo con la chiave di sessione K_{ABs} precedentemente ottenuta, tramite il messaggio* 3*;*
- *B decifra $E_{K_{ABs}}(r_B)$ con la chiave di sessione K_{Bs}, ottenendo il response al suo challenge e verifica se r_B effettivamente corrisponde a quello inviato ad A; in caso affermativo B verifica l'identità di A completando la mutua autenticazione tra le parti.*

Se a partire dalla decifrazione da parte di B, a seguito della ricezione del messaggio 1*, le verifiche reciproche dei challenge condotte dai partecipanti avvengono con successo allora gli stessi riescono ad autenticarsi reciprocamente ed ad utilizzare la chiave di sessione K_{ABs}. Per ogni successiva autenticazione il protocollo si ripete con una differente chiave di sessione K_{ABs} generata ed inviata cifrata secondo le modalità indicate.*

Osservazione 13 (Debolezze del challenge-response a chiave pubblica) *Il protocollo illustrato, oltre a possibili problematiche di tipo prestazionale, dovendo entrambe le parti continuamente utilizzare l'algoritmo asimmetrico con la coppia di chiavi per cifrare e decifrare i messaggi, ha tra le varie debolezze di sicurezza la criticità della scambio delle chiavi pubbliche; infatti, per quanto riguarda tale aspetto, le parti non hanno alcuna garanzia sul fatto che le chiavi pubbliche utilizzate per cifrare appartengano effettivamente ai legittimi proprietari, che le hanno generate, con i quali si intende comunicare; si vedrà che una soluzione a tale problema è data dall'utilizzo di un Key Distribution Center (KDC) fidato.*

Il protocollo, inoltre, può essere soggetto ad attacchi di tipo chosen text.
Un avversario, spacciandosi per la parte nota, invia all'altra un challenge cifrato con la sua chiave pubblica allo scopo di spingere l'altro partecipante a decifrare il messaggio per tentare, quindi, di ricavare informazioni sulla sua chiave segreta.

Per difendersi da tale attacco è possibile adottare nel protocollo le seguenti contromisure:

- *introdurre quantità tempo-varianti scelte da ciascun partecipante;*
- *legare le quantità usate alle identità dei partecipanti;*
- *includere nelle risposte ai challenge dei dati casuali;*
- *utilizzare chiavi differenti per funzionalità diverse.*

3.2.15 Protocollo challenge-response basato su un Key Distribution Center (KDC)

Si è visto che nel protocollo precedente basato su algoritmi a chiave asimmetrica una criticità riguarda la scalabilità della chiave di sessione condivisa; infatti se un sistema distribuito ha N host e ciascun host deve condividere una chiave segreta con ognuno degli altri $N-1$ host, ne deriva che il sistema deve gestire $\frac{N(N-1)}{2}$ chiavi mentre ogni host $N-1$ chiavi; chiaramente se N è molto "grande" l'esecuzione del protocollo può risultare un problema; pertanto, come anticipato, *è possibile utilizzare un approccio centralizzato per mezzo di un Key Distribution Center (KDC) che condivide una chiave segreta con ciascun host senza consentire la condivisione della chiave per alcuna coppia*; in tal modo l'uso di un KDC comporta la gestione di N chiavi anziché di $\frac{N(N-1)}{2}$.

Protocollo 18 (Principio di utilizzo di un KDC) *Siano A e B due utenti che intendono autenticarsi reciprocamente e concordare una chiave segreta di sessione in un canale insicuro mediante un Key Distribution Center (KDC) fidato (trusted). Siano*

- *$K_{A,KDC}$ e $K_{B,KDC}$ la chiavi condivise, rispettivamente, tra A e il KDC e B e il KDC;*
- *K_{AB} la chiave generata dal KDC per la comunicazione tra A e B;*
- *$E_{K_{A,KDC}}(K_{AB})$ la cifratura della chiave K_{AB} per la comunicazione tra A e B con la chiave $K_{A,KDC}$;*
- *$E_{K_{B,KDC}}(K_{AB})$ la cifratura della chiave K_{AB} per la comunicazione tra A e B con la chiave $K_{B,KDC}$.*

Lo schema del principio di utilizzo del KDC è il seguente:

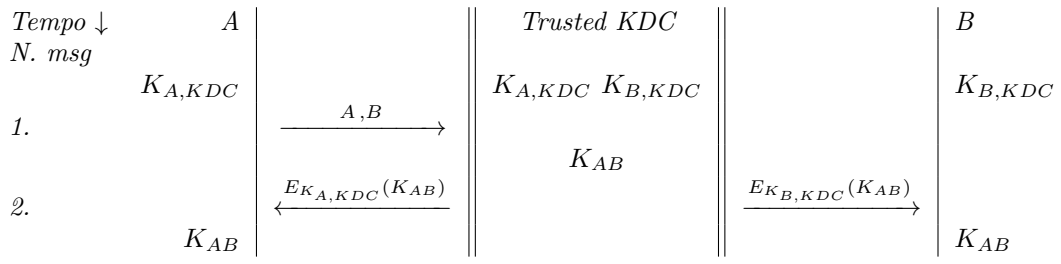


Diagramma 26: Diagramma di sequenza di un principio di utilizzo di un Key Distribution Center (KDC)

Di seguito l'analisi dei messaggi tra il KDC, A e B:

- *A invia al KDC fidato la sua identità e quella di B per poter instaurare con l'altro partecipante una comunicazione sicura, con il messaggio* 1*;*
- *ricevuto il messaggio di A, il KDC genera, quindi, una chiave da condividere tra parti, K_{AB}, e la invia sia ad A che a B cifrandola rispettivamente con la chiave $K_{A,KDC}$, condivisa con A, e $K_{B,KDC}$, condivisa con B, tramite il messaggio* 2 *(un messaggio per ciascun partecipante);*
- *A e B, ricevuti rispettivamente $E_{K_{A,KDC}}(K_{AB})$ e $E_{K_{A,KDC}}(K_{AB})$, possono decifrare i messaggi, ciascuno con la propria chiave condivisa con il KDC, ed ottenere la chiave comune K_{AB} per instaurare un canale sicuro.*

Osservazione 14 (Debolezze del principio di utilizzo di un KDC) *Il protocollo illustrato per l'utilizzo di un KDC da parte di A e B ha numerose debolezze; lo svantaggio principale consiste nel fatto che A potrebbe iniziare a comunicare con B prima che B stesso abbia ricevuto la chiave condivisa dal KDC ed un attaccante potrebbe sfruttare tale vulnerabilità; inoltre di fronte alla richiesta di A al KDC, per ottenere la chiave condivisa per comunicare con B, il KDC deve necessariamente coinvolgere B per inviargli la chiave.*

I problemi evidenziati possono essere evitati facendo restituire al KDC un ticket, ovvero $K_{B,KDC}(K_{AB})$, per A; una volta ricevuto il ticket dal KDC, A provvederà ad inviarlo a B per instaurare una comunicazione sicura; in tal modo solamente B sarà in grado di utilizzare il ticket in quanto è l'unico, oltre al KDC, in grado di decifrare l'informazione che contiene.

Di seguito lo schema del protocollo precedente con l'utilizzo del ticket:

Protocollo 19 (Principio di utilizzo di un KDC con ticket) *Siano A e B due utenti che intendono autenticarsi reciprocamente e concordare una chiave segreta di sessione in un canale insicuro mediante un KDC fidato. Siano*

- *$K_{A,KDC}$ la chiave condivisa tra A e il KDC;*
- *$K_{B,KDC}$ la chiave condivisa tra B e il KDC;*
- *K_{AB} la chiave generata dal KDC per la comunicazione tra A e B;*
- *$E_{K_{A,KDC}}(K_{AB})$ la cifratura della chiave K_{AB} per la comunicazione tra A e B con la chiave $K_{A,KDC}$;*
- *$E_{K_{B,KDC}}(K_{AB})$ la cifratura della chiave K_{AB} per la comunicazione tra A e B con la chiave $K_{B,KDC}$.*

Lo schema del principio di utilizzo di un KDC con ticket è il seguente:

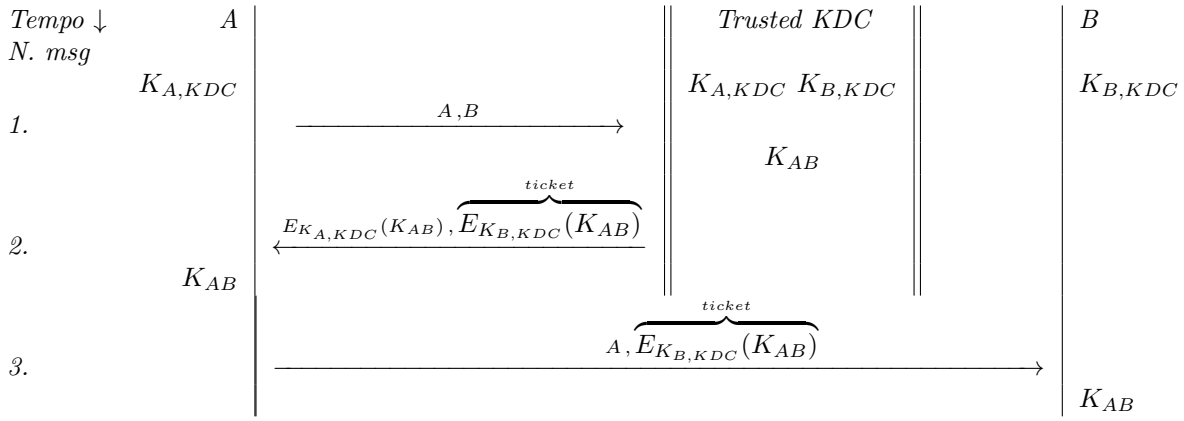


Diagramma 27: Diagramma di sequenza del principio di utilizzo di un KDC con ticket.

Di seguito l'analisi dei messaggi tra il KDC, A e B:

- *A invia al KDC fidato la sua identità e quella di B per poter instaurare una comunicazione sicura con l'altro partecipante, con il messaggio* 1*;*
- *ricevuto il messaggio di A, il KDC genera, quindi, una chiave da condividere tra le parti,* K_{AB}*, ed la invia ad A cifrandola con la chiave condivisa* $K_{A,KDC}$ *insieme alla stessa chiave cifrata però con la la chiave condivisa con B,* $K_{B,KDC}$*, ovvero il cosiddetto ticket* $E_{K_{B,KDC}}(K_{AB})$ *per poter contattare direttamente B, con il messaggio* 2*;*
- *A invia, quindi, la sua identità ed il ticket ricevuto dal KDC,* $E_{K_{B,KDC}}(K_{AB})$*, a B con il messaggio* 3*;*
- *B, ricevuto il ticket, può decifralo con la sua chiave condivisa con il KDC,* $K_{B,KDC}$*, ottenendo così la chiave condivisa con A,* K_{AB}*, per poter finalmente instaurare un canale sicuro con lo stesso partecipante.*

Osservazione 15 (Debolezze dell'utilizzo di un KDC con ticket) *Lo schema illustrato rappresenta una semplice variante del protocollo di Needham-Schroeder che si tratterà successivamente; in base alle considerazioni esposte sulla sicurezza dei protocolli di challenge-response, non prevedendo l'uso dei nonce, tale protocollo risulta vulnerabile agli attacchi di tipo replay, interleaving e reflection.*

Si consideri, ad esempio, *il caso in cui un attaccante C che sia riuscito ad ottenere una delle vecchie chiave utilizzate da B, indicata con* $K_{B,KDC}$*, con il quale decifrava i ticket, generati dal KDC, utilizzata nel corso di una precedente sessione per instaurare un canale sicuro con A.*
Lo schema dell'attacco condotto da un avversario ai danni dei partecipanti che utilizzano un KDC fidato eseguendo il protocollo indicato è il seguente.

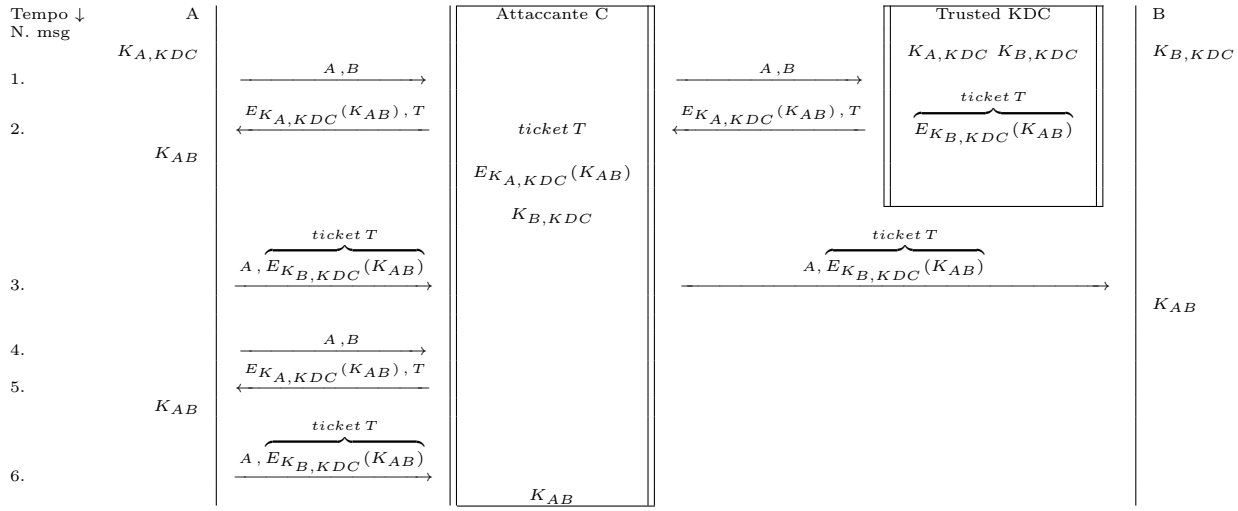


Diagramma 28: Diagramma di sequenza di un attacco del protocollo di Needham-Schroeder, che fa uso di un KDC per il rilascio di ticket, mediante il riuso di una vecchia chiave utilizzata da un partecipante.

Di seguito si riportano i passi seguiti dall'attaccante C, per autenticarsi con A, spacciandosi per B, a partire dal messaggio 4*, a seguito dello sniffing compiuto ai danni dei partecipanti al protocollo mediante il quale sono riusciti, utilizzando il KDC fidato, a scambiarsi la chiave di sessione K_{AB} utilizzando i primi 3 messaggi.*
Nel corso di tale sessione si suppone che l'attaccante sia riuscito ad intercettare il ticket generato dal KDC per A da inviare a B, $E_{K_{B,KDC}}(K_{AB})$, e sia entrato in possesso anche della chiave in comune tra B e lo stesso KDC ovvero $E_{K_{B,KDC}}$.

- *A vuole instaurare un nuovo canale sicuro con B e, analogamente a quanto fatto nella sessione precedente, invia al KDC la sua identità unitamente a quella di B, con il messaggio* 4;

- *l'attaccante C, in ascolto nella comunicazione, spacciandosi per il KDC, invia ad A una vecchia cifratura, spedita dal KDC allo stesso A ed intercettata in una sessione precedente, di una chiave comune da utilizzare con B, ovvero $E_{K_{A,KDC}}(K_{AB})$, unitamente al ticket, $E_{K_{B,KDC}}(K_{AB})$, anch'esso catturato in una sessione precedente, che A dovrà inviare a B; il ticket inviato dall'attaccante consentirà allo stesso di instaurare, all'insaputa della vittima A, una comunicazione sicura poiché non appena A lo rinvierà allo stesso attaccante, convinto di comunicare con B, permetterà allo stesso C di ottenere la chiave comune K_{AB} essendo in grado di effettuare la decifrazione con la precedente chiave di B, $K_{B,KDC}$, che è riuscito ad impossessarsi; le cifrature indicate sono inviate da C ad A tramite il messaggio* 5;

- *A, riuscendo a decifrare $E_{K_{A,KDC}}(K_{AB})$, è convinto dell'identità del KDC e, di conseguenza, invia, la sua identità insieme al ticket inoltrato dall'attaccante all'altro partecipante, con il messaggio* 6;

In tal modo, come anticipato, l'attaccante è così riuscito, riutilizzando la vecchia chiave di B, $K_{B,KDC}$, e rinviando un messaggio intercettato ad A in una sessione precedente, $E_{K_{A,KDC}}(K_{AB})$, ad ingannare lo stesso partecipante al protocollo, spacciandosi per il KDC; C è, quindi, in grado di decifrare il contenuto del messaggio 6, $E_{K_{B,KDC}}(K_{AB})$, con la chiave $K_{B,KDC}$ ed ottenere, finalmente, la chiave di sessione comune, $K_{K_{AB}}$, già utilizzata dai partecipanti al protocollo; ora C può instaurare un canale sicuro con la vittima A spacciandosi, a sua insaputa, per B.

Il protocollo di Needham-Schroeder dai nomi dei suoi inventori (Needham e Schroeder, 1978) migliora le debolezze dello schema precedentemente illustrato con l'introduzione dei nonce e di altri accorgimenti di sicurezza; in base alla sua implementazione si classifica come un *protocollo challenge-response a più vie.*

Protocollo 20 (Protocollo di Needham-Schroeder) *Siano A e B due utenti che intendono autenticarsi reciprocamente e concordare una chiave segreta di sessione in un canale insicuro mediante un KDC fidato. Siano*

- $K_{A,KDC}$ *la chiave condivisa tra A e il KDC;*
- $K_{B,KDC}$ *la chiave condivisa tra B e il KDC;*
- K_{AB} *la chiave generata dal KDC per la comunicazione tra A e B;*
- $E_{K_{B,KDC}}(A, K_{AB})$ *la cifratura dell'identità di A e della chiave K_{AB} con la chiave $K_{B,KDC}$ ovvero il ticket emesso dal KDC per la comunicazione tra A e B;*
- $E_{K_{A,KDC}}\Big(r_{A1},\, B\,, K_{AB}\,,\; E_{K_{B,KDC}}(A\,,\, K_{AB})\Big)$ *la cifratura del nonce r_{A1}, generato da A, l'identità di B e del ticket per la comunicazione tra A e B con la chiave $K_{A,KDC}$ operata dal KDC;*
- $K_{AB}(r_{A2})$ *la cifratura del nonce r_{A2}, generato da A, con la chiave K_{AB};*
- $E_{K_{AB}}(r_{A2} - 1\,,\, r_B)$ *la cifratura del nonce $r_{A2} - 1$, generato da A, e del nonce r_B con la chiave K_{AB};*
- $E_{K_{AB}}(r_B - 1)$ *la cifratura del nonce $r_B - 1$ con la chiave K_{AB}.*

Lo schema del protocollo di Needham-Schroeder è il seguente:

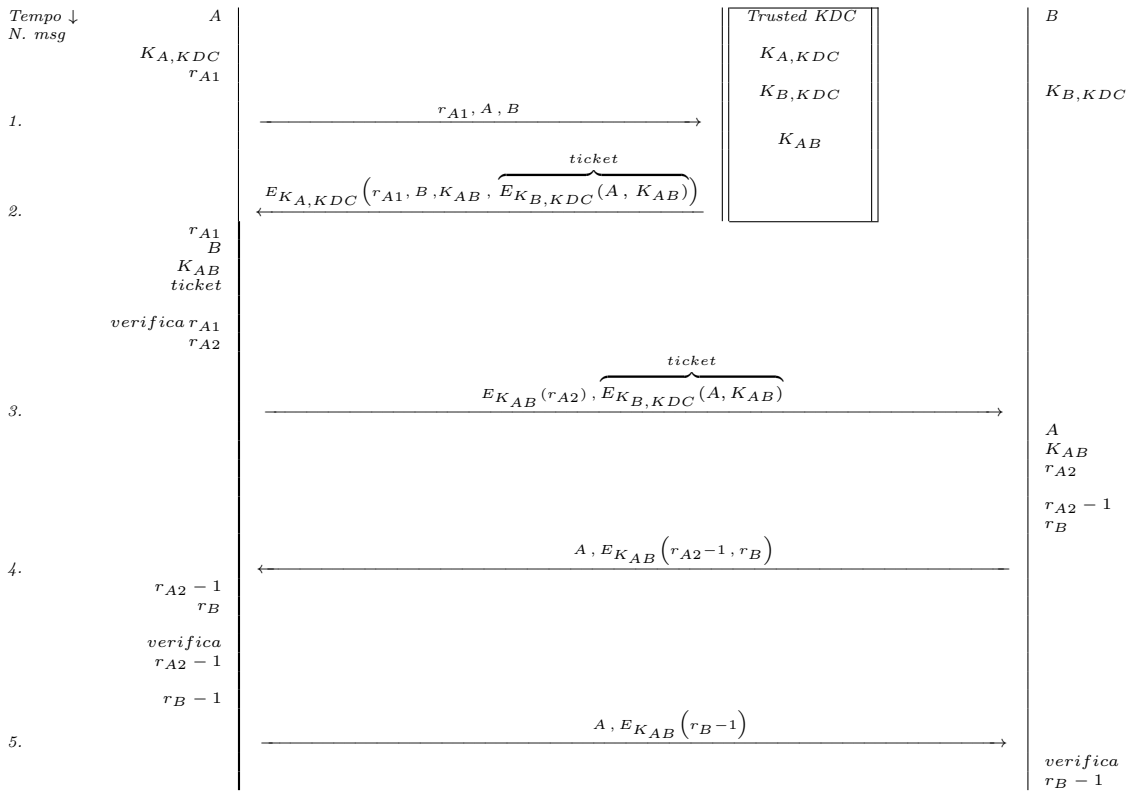


Diagramma 29: Diagramma di sequenza del protocollo di Needham-Schroeder.

Di seguito l'analisi dei messaggi tra il KDC, A e B che realizzano lo scambio della chiave comune generata dal KDC e la mutua autenticazione tra A e B.

- *A, volendo instaurare un canale sicuro con B, invia una richiesta al KDC contenente il challenge r_{A1}, la sua identità e quella di B, con il messaggio* 1*;*
- *il KDC risponde ad A inviando, con il messaggio* 2*, il ticket $E_{K_{B,KDC}}(A, K_{AB})$, ovvero la cifratura dell'identità di A e della chiave K_{AB} con la chiave $K_{B,KDC}$, che condivide con B, a sua volta cifrato insieme al challenge ricevuto, l'identità di B e la chiave comune K_{AB}, cifrata con la chiave $K_{A,KDC}$ che condivide con A;*
- *A, in possesso della chiave $K_{A,KDC}$ che condivide con il KDC, può quindi decifrare l'informazione ricevuta con il messaggio* 2*, ottenendo così il nonce r_{A1}, l'identità di B, la chiave K_{AB} da utilizzare per comunicare con B ed il ticket da inviare allo stesso B; tramite la verifica del challenge ottenuto, A può assicurarsi che la risposta provenga dal KDC e generare, quindi, un ulteriore challenge r_{A2} ed inviare a B, cifrato con la chiave comune K_{AB} insieme al ticket (che solo B può decifrare) con il messaggio* 3*;*
- *B decifra il ticket ricevuto da A con la chiave $K_{B,KDC}$, ottenendo così l'identità di A, da verificare, e la chiave comune K_{AB}, generata dal KDC, necessaria per*

decifrare $E_{K_{AB}}(r_{A2})$ *e ricavare il nonce inviato da A; B può dunque rispondere al challenge di A inviando allo stesso la sua identità e il nonce diminuito di un'unità, per dimostrare che ha saputo decifrare il messaggio precedente ed un suo challenge, cifrati con la chiave comune mediante il messaggio* 4*;*

- *A decifra* $E_{K_{AB}}(r_{A2} - 1)$ *con la chiave comune, ottenendo* $r_{A2} - 1$*; può, quindi, verificare l'identità di B in base al confronto tra il challenge inviato e la risposta ricevuta; A, infine, invia a B la sua identità e la risposta al challenge di B cifrando il nonce ricevuto* r_B *diminuito di un'unità con la chiave comune* K_{AB}*, con il messaggio* 5*;*

- *B finalmente può decifrare la risposta al challenge ricevuto da A e verificarne la correttezza per completare la mutua autenticazione.*

Il protocollo di Needham-Schroeder, a differenza del precedente illustrato, con l'utilizzo dei challenge non è vulnerabile agli attacchi basati sul riutilizzo di vecchi messaggi intercettati e sul replay con l'inganno dell'identità.
Si supponga, infatti, che un attaccante C sia entrato in possesso di una delle vecchie chiavi di B, $K^{old}_{B,KDC}$, ed abbia intercettato una precedente risposta,
del tipo $E_{K_{A,KDC}}\Big(r_{Aold},\, B\,, K_{AB}\,,\ E_{K^{old}_{B,KDC}}(A\,,\, K_{AB})\Big)$, che il KDC aveva restituito a fronte di una passata richiesta di A di comunicare con B. Nel frattempo B ha negoziato una nuova chiave condivisa con il KDC. C rimane in attesa che A richieda di nuovo di instaurare un canale sicuro verso B; a tal punto C risponde con la vecchia risposta, tentando di ingannare A facendogli credere di comunicare con B in quanto può decifrare il ticket e dimostrare che conosce la chiave segreta condivisa K_{AB}.
L'inclusione del nonce prevista dal protocollo, invece, vanifica l'attacco basato sul riutilizzo del vecchio messaggio in quanto il nonce nel messaggio della risposta r_{Aold} non corrisponderà al nonce, r_{Anew}, nella richiesta originale.

Va evidenziato, inoltre, che il messaggio 2 contiene anche l'identità di B; tale inclusione ad opera del KDC protegge A dal seguente *attacco di tipo replay.*
L'attaccante C modifica il messaggio 1 sostituendo la sua identità con quella di B allo scopo di ingannare il KDC facendogli credere che A voglia instaurare un canale sicuro con lui; C, quindi, prova ad ingannare ad A spacciandosi per il KDC ed inviandogli un fake ticket per comunicare con B in realtà, invece, per instaurare un canale con la vittima; decifrando però il messaggio ricevuto dall'attaccante, in luogo del vero KDC, ovvero $E_{K_{A,KDC}}\Big(r_{A1},\, C\,, K_{AC}\,,\ E_{K_{C,KDC}}(A\,,\, K_{AC})\Big)$, *A può verificare che, in realtà, il messaggio* 1 *inviato al KDC è stato modificato essendo incluso nella risposta di quest'ultimo l'identità di C in luogo di quella di B.*

La protezione dal riutilizzo di precedenti messaggi, garantita con i messaggi 1 *e* 2*, è assicurata anche dai messaggi* 3 *e* 4.
Infatti dopo aver ricevuto e decifrato la chiave comune di sessione K_{AB} dal KDC, A cifra con tale chiave il nuovo challenge r_{A2} generato, dunque successivo a r_{A1}, per l'invio a B insieme al ticket ricevuto con il messaggio 3; B, quindi, decodifica il ticket per ottenere la chiave condivisa e restituisce la risposta $E_{K_{AB}}(r_{A2} - 1\,,\, r_B)$, contenente anche il suo

challenge r_B, cifrata con tale chiave; *restituendo* $r_{A2}-1$ *e non* r_{A2}*, B non solo dimostra ad A di conoscere la chiave comune* K_{AB} *ma anche di aver realmente decodificato il challenge; tale prova fornita, dunque, lega il messaggio* 4 *al messaggio* 3 *nello stesso modo in cui il nonce* r_{A1} *legava il messaggio* 2 *al messaggio* 1*; il protocollo risulta così protetto dal riuso doloso dei messaggi.*
In tale caso particolare sarebbe stato sufficiente che B avesse restituito $E_{K_{AB}}(r_{A2}\,,\,r_B)$, per la semplice ragione che il messaggio non è stato ancora utilizzato nel protocollo; $E_{K_{AB}}(r_{A2}-1\,,\,r_B)$, infatti, dimostra già che B è stato in grado di decifrare il challenge inviato nel messaggio 3: il contenuto del messaggio 4, previsto nel protocollo di Needham-Schroeder, è dovuto a ragioni storiche.

Osservazione 16 (Debolezze del protocollo di Needham-Schroeder) *Una vulnerabilità del protocollo di Needham-Schroeder è basata sul fatto che l'autenticazione di A da parte di B avviene solamente dopo la verifica del nonce a seguito della ricezione del messaggio* 5 *e, pertanto, B non è sicuro dell'identità di A nell'invio del messaggio* 4*; tale vulnerabilità viene sfruttata dal seguente attacco che consiste nel riuso di una vecchia chiave di sessione,* K_{AB}*, intercettata e decodificata da un attaccante C che si finge A.*

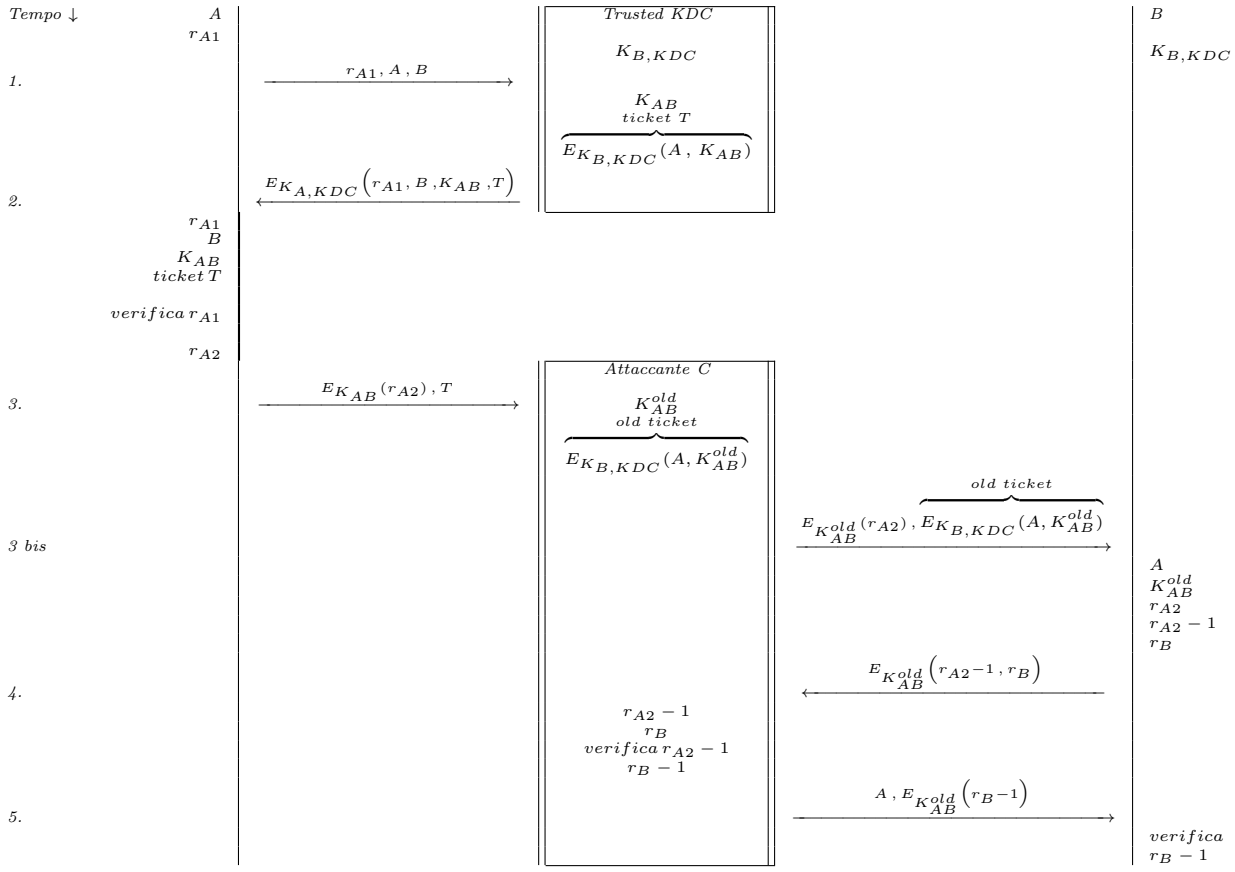


Diagramma 30: Diagramma di sequenza di un attacco al protocollo di Needham-Schroeder che prevede il riuso di una vecchia chiave di sessione.

I passi seguiti dall'attaccante C per autenticarsi con B, spacciandosi per A, a partire dal messaggio 3, sono i seguenti:

- *C spacciandosi per A, invia a B un vecchio messaggio 3 del protocollo composto dalla cifratura del nonce r_{A2} mediante la chiave di sessione comune di cui è entrato in possesso, ovvero $E_{K_{AB}^{old}}(r_{A2})$ e da un precedente ticket che contiene tale chiave $E_{K_{B,KDC}}(A, K_{AB}^{old})$. Nel diagramma di sequenza dell'attacco il messaggio è il 3 bis;*
- *B riceve il messaggio riutilizzato da C, convinto che provenga da A, e ne decifra il contenuto con la sua chiave $E_{K_{B,KDC}}$ ottenendo l'identità di A, la vecchia chiave k_{AB}^{old}, di cui C è entrato in possesso, ed il nonce che era cifrato con tale chiave;*

  B calcola, quindi, $r_{A2} - 1$ e genera il nonce r_B per la verifica dell'identità di A; B invia, quindi, tali nonce cifrati con la chiave k_{AB}^{old} ad A con il messaggio 4;
- *C intecetta il messaggio 4 diretto ad A e lo decifra con la vecchia chiave k_{AB}^{old} in possesso inviata in modo fraudolento a B; l'attaccante, sostituendosi ad A, riesce quindi verificare l'identità di B con il nonce r_{A2} e, a sua volta, calcolare $r_B - 1$ da inviare alla vittima insieme all'identità di A con il messaggio 5;*
- *B, decifrando il messaggio 5 e controllando la validità del nonce $r_B - 1$, valida la falsa identità di A generata da C e, pertanto, si convince di aver instaurato un canale sicuro con A ignaro, in realtà, di averlo stabilito con l'attaccante.*

Di seguito si illustrano una serie di *estensioni delle versione del protocollo di Needham-Schroeder che prevedono l'utilizzo di ulteriori nonces, timestamp e di entrambe le contromisure combinate a protezione dall'attacco basato sul riuso doloso di precedenti chiavi di sessione decifrate.*

Protocollo 21 (Protocollo di Needham-Schroeder esteso con nonce) *Siano A e B due utenti che intendono autenticarsi reciprocamente e concordare una chiave segreta di sessione in un canale insicuro mediante un KDC fidato. Siano*

- *$K_{A,KDC}$ la chiave condivisa tra A e il KDC;*
- *$K_{B,KDC}$ la chiave condivisa tra B e il KDC;*
- *K_{AB} la chiave generata dal KDC per la comunicazione tra A e B;*
- *$E_{K_{B,KDC}}(r_{B1})$ la cifratura del nonce r_{B1} emesso da B con con la chiave $K_{B,KDC}$;*
- *$E_{K_{B,KDC}}(A, K_{AB}, r_{B1})$ la cifratura dell'identità di A, della chiave K_{AB} e del nonce r_{B1} con la chiave $K_{B,KDC}$ ovvero il ticket emesso dal KDC per la comunicazione tra A e B;*

- $E_{K_{A,KDC}}\Big(r_{A1},\, B\,, K_{AB}\,,\; E_{K_{B,KDC}}(A\,,\, K_{AB}, r_{B1})\Big)$ *la cifratura del nonce* r_{A1}*, generato da* A*, l'identità di* B*, la chiave comune* K_{AB} *e del ticket emesso per la comunicazione tra* A *e* B *con la chiave* $K_{A,KDC}$ *operata dal KDC;*
- $E_{K_{AB}}(r_{A2})$ *la cifratura del nonce,* r_{A2}*, generato da* A *con la chiave* K_{AB}*;*
- $E_{K_{AB}}(r_{A2}-1\,,\, r_{B2})$ *la cifratura dei nonce* $r_{A2}-1$ *e* r_{B2} *con la chiave* K_{AB}*;*
- $E_{K_{AB}}(r_{B2}-1)$ *la cifratura del nonce* $r_{B2}-1$ *con la chiave* K_{AB}*.*

Lo schema del protocollo di Needham-Schroeder esteso con nonce è il seguente:

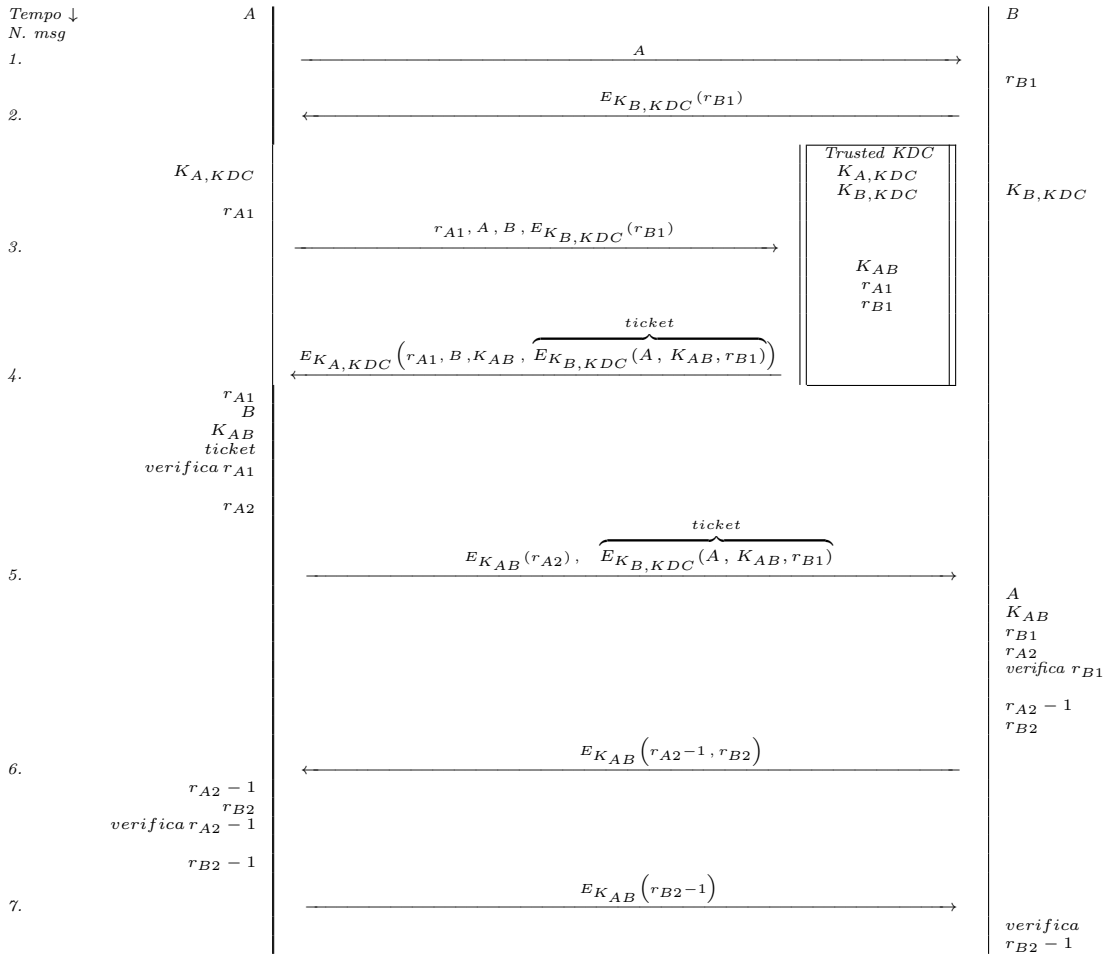


Diagramma 31: Diagramma di sequenza del protocollo di Needham-Schroeder esteso con nonce.

Di seguito l'analisi dei messaggi tra il KDC, A *e* B *che realizzano lo scambio della chiave comune generata dal KDC e la mutua autenticazione tra* A *e* B*.*

- A*, volendo instaurare un canale sicuro con* B*, invia la sua identità a* B *con il messaggio* 1*;*

- *B risponde ad A inviando un nonce, r_{B1}, cifrato con la chiave segreta che condivide con il KDC (che A non conosce), con il messaggio* 2*;*

- *A, a sua volta, genera un nonce, r_{A1}, e lo invia al KDC insieme alla sua identità, quella di B e la cifratura ricevuta dallo stesso B, $E_{K_{B,KDC}}(r_{B1})$, tramite il messaggio* 3*;*

- *il KDC decifra $E_{K_{B,KDC}}(r_{B1})$, con la chiave $K_{B,KDC}$ che condivide con B, e ricava il nonce r_{B1} generato da B ed inoltrato da A; può, quindi, inviare ad A il nonce r_{A1} ricevuto, l'identità di B, la chiave segreta di sessione K_{AB}, per la comunicazione tra A e B, ed il ticket $E_{K_{B,KDC}}(A, K_{AB}, r_{B1})$, tutti cifrati con la chiave condivisa con A, $K_{A,KDC}$, con il messaggio* 4*;*

- *A, in possesso della chiave $K_{A,KDC}$, che condivide con il KDC, può decifrare l'informazione ricevuta con il messaggio precedente, ottenendo il nonce r_{A1}, l'identità di B, la chiave di sessione K_{AB}, da utilizzare per comunicare con B, ed il ticket da inviare allo stesso B; tramite la verifica del challenge ottenuto, A può assicurarsi che la risposta provenga dal KDC; A genera, quindi, un ulteriore challenge, r_{A2}, e lo invia a B, cifrandolo con la chiave comune, K_{AB}, unitamente al ticket ricevuto (che solo B può decifrare), con il messaggio* 5*;*

- *B decifra il ticket ricevuto da A con la chiave $K_{B,KDC}$ ottenendo così l'identità di A, da verificare, e la chiave comune, K_{AB}, necessaria per decifrare $E_{K_{AB}}(r_{A2})$ e ricavare il nonce inviato da A; B può dunque rispondere al challenge di A, r_{A2}, inviando allo stesso il nonce diminuito di un'unità, per dimostrare che ha saputo decifrare il messaggio precedente ed un suo challenge, r_{B2}, cifrati con la chiave comune, mediante il messaggio* 6*;*

- *A decifra il testo del messaggio* 6 *con la chiave comune, ottenendo $r_{A2} - 1$ e r_{B2}; può, quindi, verificare l'identità di B in base al confronto tra il challenge inviato, r_{A2}, e la risposta ricevuta $r_{A2} - 1$; A, infine, invia a B la risposta al suo challenge, cifrando il nonce ricevuto r_{B2} diminuito di un'unità, cifrato con la chiave comune, con il messaggio* 7*;*

- *B, finalmente, può decifrare la risposta al challenge ricevuto da A, $r_{B2} - 1$, e verificarne la correttezza per completare la mutua autenticazione.*

La protezione dal riuso doloso delle vecchie chiavi di sessione decifrate viene attuata nel protocollo precedente legando la successione dei messaggi con i nonce; ciascun partecipante, infatti, effettua due verifiche consecutive che consentono il riconoscimento di eventuali manipolazione dei messaggi da parte di un'attaccante che si intromette nella comunicazione.
La richiesta inviata da A al KDC nel messaggio 3 dipende dal nonce inviato da B ad A con il messaggio 2; tale fatto garantisce a B la certezza che chiunque voglia approntare un canale sicuro verso di lui deve ottenere l'informazione appropriata da parte del KDC tramite il messaggio 4 dipendente, quindi, del messaggio 3. Il messaggio 5, inviato da

A a B, dipende dal precedente in quanto contiene il ticket per B, emesso dal KDC e decifrato da A, ed un ulteriore challenge per B; il messaggio 6 riporta la risposta di B al challenge ricevuto con il messaggio precedente ed un ulteriore challenge che servirà a B per verificare l'identità di A. Infine il messaggio 7, legato al messaggio 6, consente ad A di inviare a B il response al secondo challenge, completando la mutua autenticazione con la verifica dell'ultimo nonce ricevuto.

Protocollo 22 (Protocollo di Needham-Schroeder esteso con timestamp) *Siano A e B due utenti che intendono autenticarsi reciprocamente e concordare una chiave segreta di sessione in un canale insicuro mediante un KDC fidato. Assumendo un tempo universale, siano*

- $K_{A,KDC}$ *la chiave condivisa tra A e il KDC;*
- $K_{B,KDC}$ *la chiave condivisa tra B e il KDC;*
- K_{AB} *la chiave generata dal KDC per la comunicazione tra A e B;*
- $t_{K_{AB}}$ *il timestamp emesso dal KDC che indica il periodo di validità della chiave K_{AB};*
- $E_{K_{B,KDC}}(A\,,\,K_{AB}, t_{K_{AB}})$ *la cifratura dell'identità di A, della chiave K_{AB} e del timestamp $t_{K_{AB}}$ con la chiave $K_{B,KDC}$ ovvero il ticket emesso dal KDC per la comunicazione tra A e B;*
- $E_{K_{A,KDC}}\Big(r_{A1},\, B\,, K_{AB}\,,\; E_{K_{B,KDC}}(A\,,\,K_{AB}, t_{K_{AB}})\Big)$ *la cifratura del nonce r_{A1}, generato da A, dell'identità di B, della chiave K_{AB} e del ticket per la comunicazione tra A e B cifrato con la chiave $K_{A,KDC}$ ad opera dal KDC;*
- T_{auth} *il tempo in cui avviene l'autentica;*
- $K_{AB}(T_{auth})$ *la cifratura di T_{auth} con la chiave K_{AB};*
- $K_{AB}(T_{auth}+1)$ *la cifratura di $T_{auth}+1$ con la chiave K_{AB}.*

Lo schema del protocollo Needham-Schroeder esteso con timestamp è il seguente:

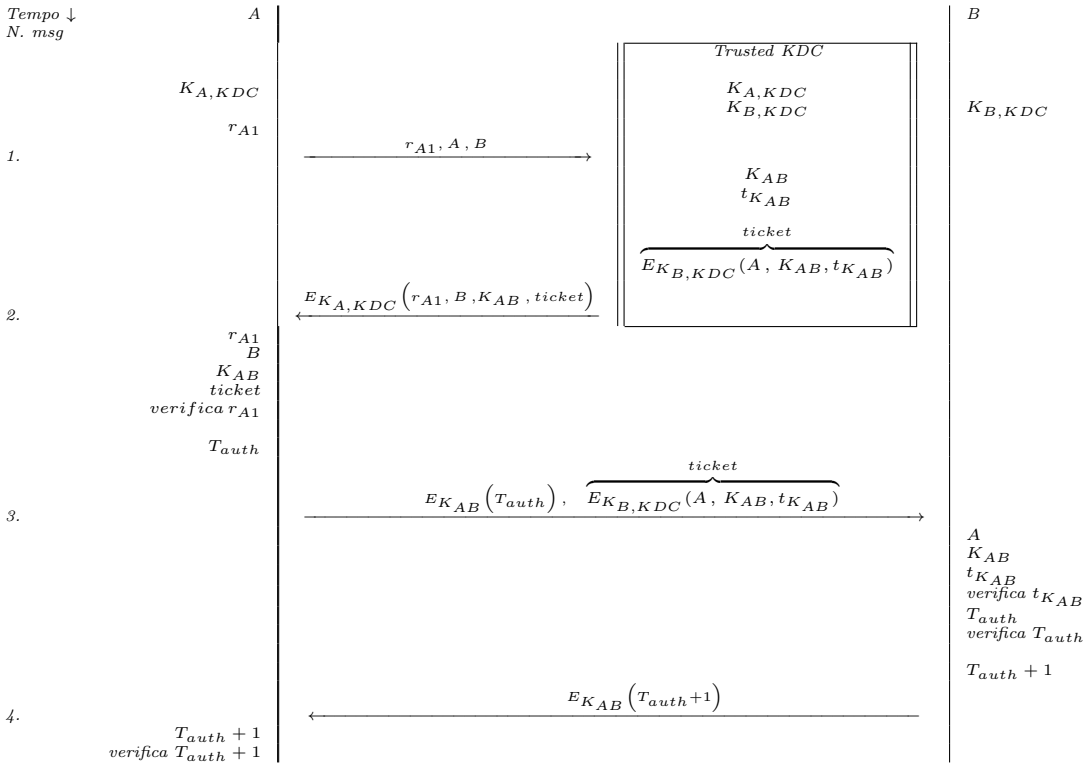


Diagramma 32: Diagramma di sequenza del protocollo di Needham-Schroeder esteso con timestamp.

Di seguito l'analisi dei messaggi tra il KDC, A e B che realizzano lo scambio della chiave comune generata dal KDC e la mutua autenticazione tra A e B.

- *A, volendo instaurare un canale sicuro con B, invia una richiesta al KDC contenente il challenge r_{A1}, la sua identità e quella di B, con il messaggio* 1*;*
- *il KDC, ricevuta la richiesta di A, emette a favore dello stesso un ticket cifrato con la chiave $E_{K_{B,KDC}}$ che, oltre all'identità di A e la chiave segreta da utilizzare tra le parti, K_{AB}, contiene un timestamp per la verifica della validità di tale chiave; il ticket, che solo B può decifrare, è a sua volta cifrato con la chiave condivisa con A, $E_{K_{A,KDC}}$, insieme al nonce ricevuto da A, r_{A1}, l'identità di B e la chiave segreta di sessione da utilizzare con B ed inviato ad A con il messaggio* 2*;*
- *A decifra il contenuto del messaggio* 2 *con la chiave che condivide con il KDC, verifica l'identità del KDC tramite il response ricevuto al challenge, r_{A1}, inviato con il messaggio* 1 *e ricava la chiave K_{AB} da utilizzare nella comunicazione con B; A cifra, quindi, con la chiave comune con B, il tempo in cui avviene l'autentica, T_{auth} e lo invia a B insieme al ticket ricevuto dal KDC con il messaggio* 3*;*
- *B decifra il ticket ricevuto da A con la chiave $K_{B,KDC}$ ottenendo così l'identità di A, da verificare, la chiave comune K_{AB}, necessaria per decifrare $E_{K_{AB}}(T_{auth})$, e il*

timestamp $t_{K_{AB}}$ inviati da A; B può dunque verificare la correttezza del periodo di validità della chiave K_{AB} e l'identità di A; B, una volta effettuata l'autenticazione di A, incrementa di uno T_{auth} e lo cifra con la chiave K_{AB} inviandolo a A con il messaggio 4;

- *A decifra $E_{K_{AB}}(T_{auth}+1)$, con la chiave, K_{AB}, in comune con B, e verifica l'identità di B mediante la validità del tempo di autenticazione incrementato dallo stesso B; in tal modo la mutua autenticazione è completa.*

Protocollo 23 (Protocollo di Needham-Schroeder con timestamp e nonce) *Siano A e B due utenti che intendono autenticarsi reciprocamente e concordare una chiave segreta di sessione in un canale insicuro mediante un KDC fidato con timestamp e nonce. Si suppone che A e B utilizzino clock sincroni per la validazione dei timestamp. Siano*

- *$K_{A,KDC}$ la chiave condivisa tra A e il KDC;*
- *$K_{B,KDC}$ la chiave condivisa tra B e il KDC;*
- *K_{AB} la chiave generata dal KDC per la comunicazione tra A e B;*
- *$t_{K_{AB}}$ il timestamp, emesso dal KDC, che indica il periodo di validità della chiave K_{AB};*
- *$E_{K_{B,KDC}}(A, K_{AB}, t_{K_{AB}})$ la cifratura dell'identità di A, della chiave K_{AB} e del timestamp $t_{K_{AB}}$ con la chiave $K_{B,KDC}$ ovvero il ticket emesso dal KDC per la comunicazione tra A e B;*
- *$E_{K_{A,KDC}}\Big(r_{A1},\, B\,, K_{AB}\,, t_{K_{AB}}\,, E_{K_{B,KDC}}(A\,,\, K_{AB}, t_{K_{AB}})\Big)$ la cifratura del nonce r_{A1}, generato da A, dell'identità di B, della chiave K_{AB}, del timestamp $t_{K_{AB}}$ e del ticket per la comunicazione tra A e B con la chiave $K_{A,KDC}$ operata dal KDC;*
- *$E_{K_{AB}}(B\,, r_{B1})$ la cifratura di B e del nonce r_{B1}, generato da A, con la chiave K_{AB};*
- *$E_{K_{AB}}(A\,, r_{B1}-1)$ la cifratura di A e del nonce $r_B - 1$, generato da B, con la chiave K_{AB}.*

Lo schema del protocollo di Needham-Schroeder con l'utilizzo di timestamp e nonce è il seguente:

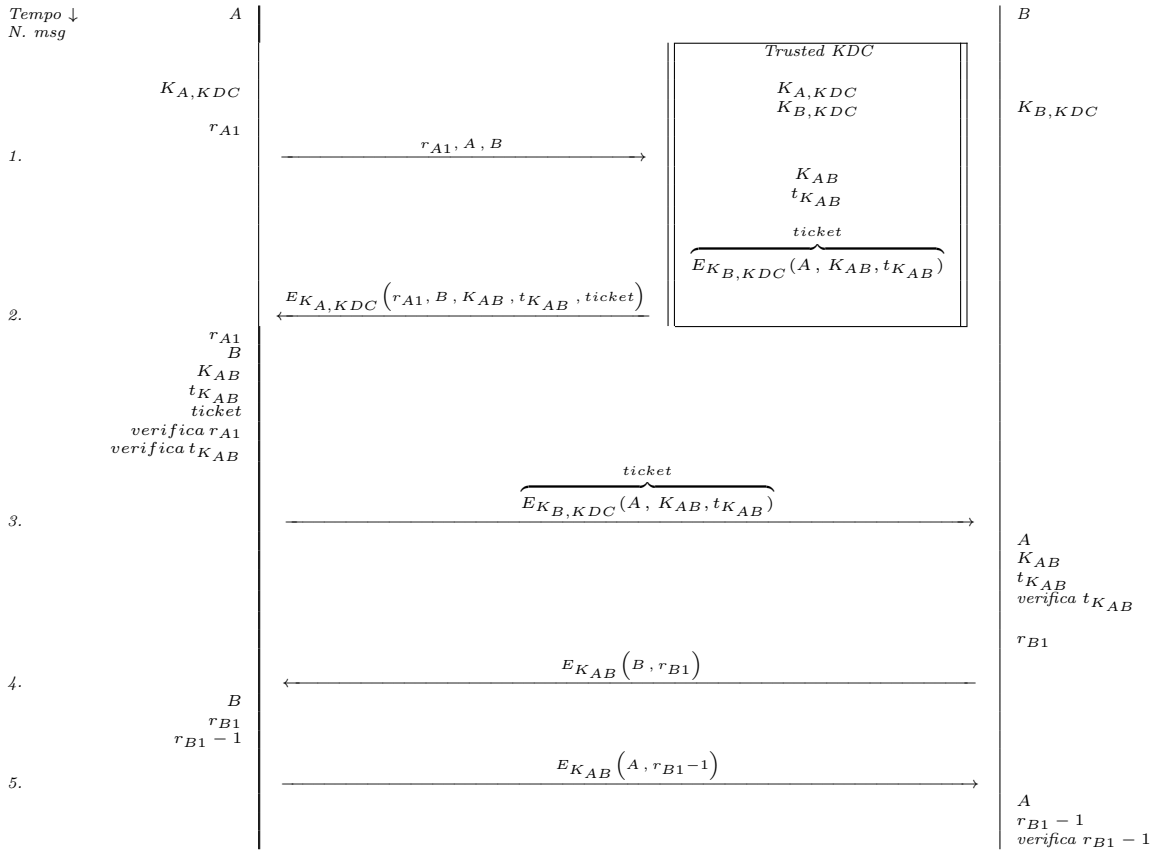


Diagramma 33: Diagramma di sequenza del protocollo di Needham-Schroeder con timestamp e nonce.

Di seguito l'analisi dei messaggi tra il KDC, A e B che realizzano lo scambio della chiave comune generata dal KDC e la mutua autenticazione tra A e B.

- *A, volendo instaurare un canale sicuro con B, invia una richiesta al KDC contenente il challenge r_{A1}, la sua identità e quella di B con il messaggio* 1*;*
- *il KDC, ricevuta la richiesta di A, emette a favore dello stesso un ticket cifrato con la chiave $E_{K_{B,KDC}}$ che contiene l'identità di A, la chiave segreta da utilizzare tra le parti, K_{AB}, ed un timestamp per la verifica della validità di tale chiave; il ticket, che solo B può decifrare, è a sua volta cifrato con la chiave condivisa con A, $E_{K_{A,KDC}}$, insieme al nonce ricevuto da A, r_{A1}, l'identità di B, la chiave segreta di sessione da utilizzare con B e lo stesso timestamp inserito nel ticket, ed inviato ad A con il messaggio* 2*;*
- *A decifra il contenuto del messaggio* 2 *con la chiave che condivide con il KDC, ricavando la risposta del KDC al challenge inviato con il messaggio* 1*, il timestamp generato dallo stesso KDC, l'identità di B e la chiave K_{AB} da utilizzare nella*

comunicazione con B; A verifica, quindi, le validità della risposta al challenge, r_{A1}, del timestamp relativo alla chiave ricevuto, $t_{K_{AB}}$, essendo i clock tra la parti sincroni; in tal modo A valida l'identità di B, tramite il KDC e, di conseguenza, prosegue l'esecuzione del protocollo inviando a B il ticket ricevuto dal KDC con il messaggio 3*;*

- *B decifra il ticket ricevuto da A, con la chiave $K_{B,KDC}$, ottenendo l'identità di A, ancora da verificare, la chiave comune K_{AB}, generata dal KDC, e il timestamp $t_{K_{AB}}$ necessario per verificare la correttezza del periodo di validità della chiave K_{AB}; B, una volta effettuata la validazione temporale del timestamp, genera il challenge r_{B1} e lo invia insieme alla sua identità, cifrato con la chiave K_{AB}, ad A con il messaggio* 4*;*

- *A decifra il challenge con la chiave in comune con B, K_{AB}, ottenendo r_{B1}, lo decrementa di un'unità e lo invia cifrato insieme alla sua identità, con la stessa chiave, a B con il messaggio* 5*;*

- *B decifra la risposta al challenge ricevuta da A con la chiave K_{AB}; B può, quindi, verificare l'identità di A mediante la validazione di $r_{B1} - 1$; in tal modo la mutua autenticazione è completa.*

3.2.16 Protocolli a conoscenza zero

Si conclude la trattazione dei protocolli di autenticazione con l'illustrazione dei *protocolli a conoscenza zero, Zero Knowledge Proof*, (*ZKP*).
Prima di entrare nel vivo dell'argomentazione dei protocolli ZKP, allo scopo di evidenziare le ragioni che hanno portato alla progettazione e le peculiarità che li caratterizzano, risulta utile riepilogare alcune problematiche di sicurezza emerse finora in merito all'autenticazione.

Dai protocolli di autenticazione esaminati si è visto che *il problema dell'autenticazione è stato risolto fornendo una prova dell'identità da un partecipante al protocollo che vuole accedere ad un sistema per ottenere informazioni private; tale prova può essere costituita da nome utente e password, da un PIN o da altri tipi di informazioni richieste dallo stesso sistema.*

Si è più volte evidenziato che, in caso di canale non sicuro, un attaccante in ascolto può scoprire le informazioni trasmesse ed usarle in un secondo momento per accedere alle informazioni private pur non avendo le necessarie autorizzazioni; anche nell'ipotesi migliore in cui il canale utilizzato per lo scambio delle informazioni riservate sia stato reso sicuro, rimane, in ogni caso, il problema della fiducia del sistema che richiede l'autenticazione; infatti nel momento in cui la parte, che deve essere autenticata, fornisce le proprie credenziali all'altro partecipante, comunica informazioni che potrebbero esserle usate contro in un secondo momento.
Il problema intrinseco di un tale schema di autenticazione sta nel fatto che sono fornite più informazioni rispetto a quanto richiesto; infatti, come visto, *lo scopo di un protocollo di autenticazione non è l'acquisizione delle credenziali ma piuttosto la verifica*

della correttezza di una caratteristica dichiarata da un'entità; pertanto, un protocollo che necessita di più informazioni rispetto a quelle necessarie, non rispetta agli standard di sicurezza previsti e non risolve il problema dell'autenticazione in modo ottimale, esponendo i sistemi che lo utilizzano a potenziali rischi nei confronti di un attaccante che intende sfruttare le informazioni aggiuntive raccolte nel corso dello scambio dei messaggi. La progettazione dei protocolli ZKP nasce proprio per ovviare a tali problematiche.

Definizione 105 (Protocollo a conoscenza zero - ZKP) *In crittografia, si definisce un protocollo a conoscenza zero o Zero Knowledge Proof (ZKP) un metodo interattivo mediante il quale una partecipante, il prover, può dimostrare all'altro, il verifier, che una affermazione (solitamente matematica) è vera, senza rivelare niente altro oltre alla veridicità della stessa.*

L'essenza di una dimostrazione a conoscenza zero risiede nel fatto che è banale dimostrare di possedere la conoscenza di determinate informazioni semplicemente rivelandole; la sfida, invece, consiste nel provare tale possesso senza rivelare l'informazione stessa o alcuna informazione aggiuntiva.

Si supponga, ad esempio, di voler realizzare un sistema di autenticazione basato sulla crittografia a chiave pubblica; utilizzando la ormai consueta notazione, ogni utente avrà una chiave pubblica K^+ ed una chiave privata segreta K^- tali che $D_{K^-}(E_{K^+}(M)) = M$. Se il sistema crittografico è sicuro, riuscire a decriptare il messaggio cifrato $E_{K^+}(M)$ risulta un problema intrattabile a meno di conoscere la chiave privata K^-; dunque è ragionevole supporre che chi sia in grado di decriptare $E_{K^+}(M)$ conosca K^-.
Un primo semplice esempio di protocollo ZKP che si introduce è il seguente.

Protocollo 24 (Un tentativo di protocollo ZKP) *Siano A e B due partecipanti che intendono utilizzare un protocollo ZKP; sia A il prover e B il verifier che deve verificare l'identità di A. A vuole dimostrare di possedere la chiave privata, K_A^-, di un algoritmo a chiave pubblica, senza ovviamente rivelarla, a B . Sia (K_A^+, K_A^-) la coppia di chiavi rispettivamente pubblica e privata di A, M un messaggio scelto da B e $E_{K_A^+}(M)$ la cifratura del messaggio M scelto da B con la chiave pubblica, K_A^+, di A.*
Lo schema di un tentativo di protocollo ZKP il seguente:

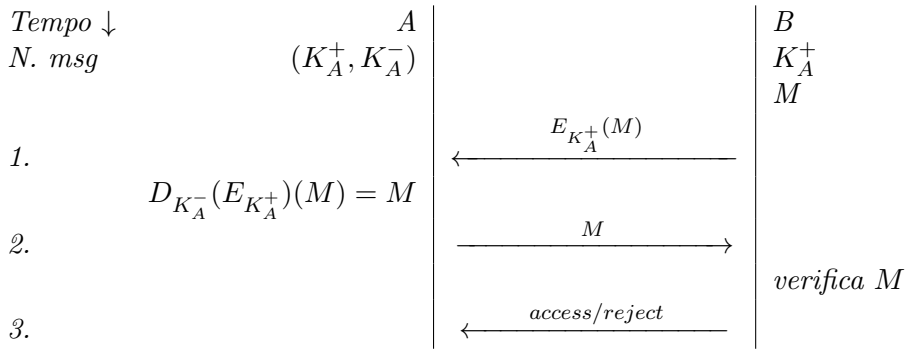


Diagramma 34: Diagramma di sequenza di un tentativo di protocollo ZKP.

Di seguito l'analisi dei messaggi tra A ed B:

- *B sceglie un messaggio M ed invia lo stesso ad A cifrato con la chiave pubblica del prover, $E_{K_A^+}(M)$, con il messaggio* 1*;*
- *A decifra $E_{K_A^+}(M)$, usando la sua chiave privata K_A^-, ed invia il risultato, M, al verifier B con il messaggio* 2*;*
- *B verifica il risultato ricevuto da A ed in base all'esito accetta o rifiuta l'identità di A con il messaggio* 3*.*

Osservazione 17 (Debolezza del tentativo di protocollo ZKP) *Nel semplice protocollo illustrato, B, il verifier, può essere certamente soddisfatto dell'esecuzione: A ha, infatti, provato la sua identità dimostrando di essere capace di decifrare C; se il sistema crittografico è sufficientemente sicuro, solamente A conosce K_A^- e dunque il protocollo è corretto; inoltre se A possiede K_A^+ riesce sicuramente a farsi autenticare e dunque il protocollo risulta che completo.*

Dal punto di vista del prover, invece, bisogna chiedersi se A sia soddisfatto dell'esecuzione del protocollo; se B è onesto sicuramente A è soddisfatto ma se B tenta di barare, ad esempio, inviando ad A dei messaggi cifrati C di cui in realtà non conosce la decodifica per sfruttare A stesso ad indurlo a scoprire il messaggio originale, il protocollo presenta un problema; dunque non c'è la sicurezza che B conosca il testo originale M nel momento in cui lo invia; d'altro canto, però, non si può costringere B a rivelare M insieme a $E_{K_A^+}(M)$, altrimenti A potrebbe rispondere utilizzando tale M.

Il problema esposto nell'osservazione può essere risolto utilizzando un *commitment* nel protocollo che assumerà la seguente nuova formulazione.

Protocollo 25 (Un tentativo di protocollo ZKP con commitment) *Siano A e B due partecipanti che intendono utilizzare un protocollo ZKP; sia A il prover e B il verifier che deve verificare l'identità di A. A vuole dimostrare di possedere la chiave privata K_A^- , senza ovviamente rivelarla, di un algoritmo a chiave pubblica, a B .*
Sia (K_A^+, K_A^-) la coppia di chiavi rispettivamente pubblica e privata di A, M un messaggio scelto da B e $E_{K_A^+}(M)$ la cifratura del messaggio M scelto da B con la chiave pubblica, K_A^+, di A.

Lo schema di un tentativo di protocollo di Zero-Knowledge con il commitment è il seguente:

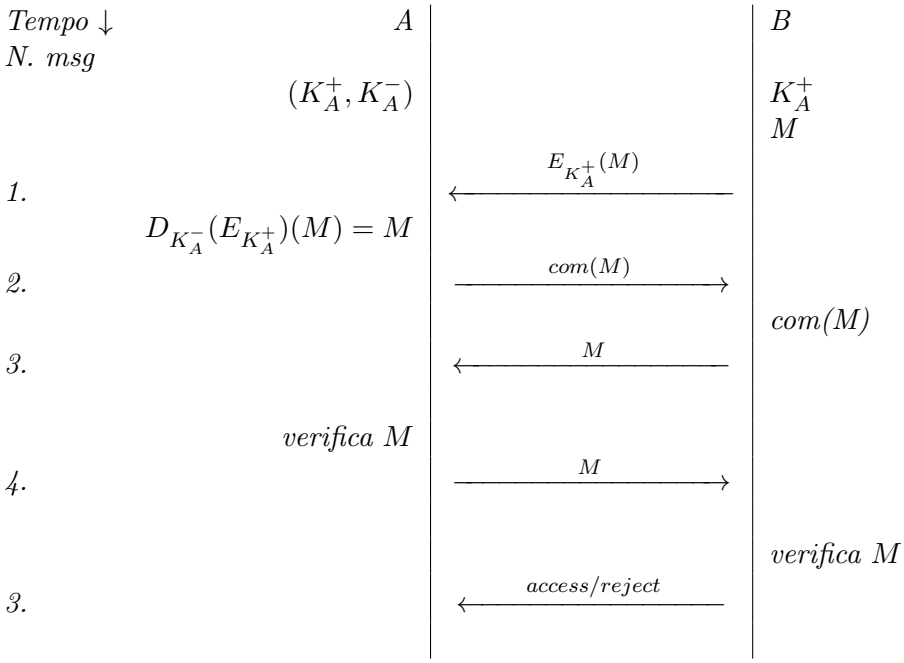


Diagramma 35: Diagramma di sequenza di un tentativo di protocollo ZKP con commitment.

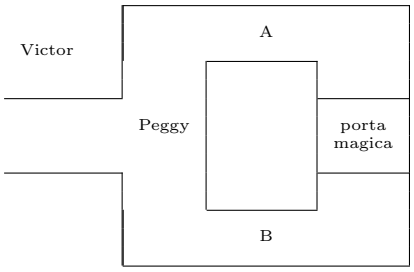
Di seguito l'analisi dei messaggi tra A ed B:

- *B sceglie un messaggio M ed invia lo stesso ad A cifrato con la chiave pubblica del prover, $E_{K_A^+}(M)$, con il messaggio* 1*;*
- *A decifra $E_{K_A^+}(M)$, usando la sua chiave privata, K_A^-, ed invia a B il commitment del risultato, com(M), con il messaggio* 2*;*
- *Ricevuto il commitment, B rivela M ad A con il messaggio* 3*;*
- *A verifica la correttezza di M; se l'esito è negativo interrompe l'esecuzione del protocollo poiché significherebbe che B ha barato, altrimenti rileva il contenuto del commitment a B con il messaggio* 4*;*
- *B verifica il messaggio M ricevuta e se l'esito è positivo accetta l'identità di A con il messaggio* 5*.*

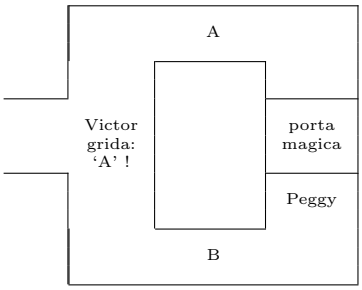
La tecnica impiegata nello schema illustrato è molto utilizzata nella progettazione dei protocolli ZKP; infatti, spesso, il prover A dimostra al verifier B qualcosa che B già conosce e, per tanto, lo stesso verifier non guadagna alcuna informazione.

Nel famoso articolo "*How to explain Zero-Knowledge Protocollos to Your Children*" illustrato da Quisquater, Guillou e Berson alla conferenza sui progressi della crittografia, tenutasi nel 1989 a Santa Barbara presso l'Università della California, viene utilizzata

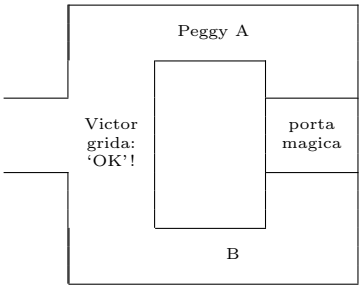
la *nota storia della caverna di Ali Babà per presentare alcune idee alla base dei ZKP basate su uno schema di challenge-response.*
Nelle dimostrazione a conoscenza zero è uso comune utilizzare i seguenti nomi per indicare le parti: *Peggy*, il prover, e *Victor*, il *verifier*.
La storia narra che Peggy ha scoperto la parola segrata per aprire la porta magica della caverna di Ali Babà; tale caverna, rappresentata nella figura seguente, ha l'entrata su un lato e la porta magica che blocca il percorso circolare nell'altro lato.
Victor dice a Peggy che è disposto a pagarla per frasi rivelare il segreto ma non prima di essersi assicurato che ne sia effettivamente a conoscenza; Peggy, d'altro canto, è d'accordo a rivelare il segreto a Victor ma non prima di aver ricevuto il pagamento. Peggy e Victor, quindi, per soddisfare entrambe le esigenze, pianificano uno schema con il quale Peggy può dare prova a Victor di conoscere la parola magica senza però rivelarla.
Dapprima Victor aspetta fuori della caverna mentre Peggy entra. Peggy ha a disposizione due percorsi: quello alla sua sinistra, indicato con A, e quello alla sua destra, indicato con B, come illustrato.



Peggy sceglie a caso uno dei due percorsi; quindi Victor entra nella caverna e grida il nome del percorso che Peggy dovrà utilizzare per tornare indietro tra A e B presi a caso; se si ipotizza che Peggy conosca veramente la parola magica per aprire la porta, il ritorno indietro risulta facile; infatti Peggy apre la porta se necessario e ritorna mediante la strada desiderata. Nella figura seguente, ad esempio, Victor entra nella caverna e chiede a Peggy, che ha scelto il percorso B, di andare in A.



Peggy, quindi, pronuncia la *parola magica* per aprire la porta necessaria per arrivare al percorso A indicato da Victor; lo stesso verifica, dunque, che Peggy è riuscita ad arrivare al percorso richiesto, come nella rappresentato nella figura seguente.



La sequenza delle richieste viene ripetuta finché Victor, che non conosce il percorso della caverna dal quale Peggy è entrata, non si convince del fatto che Peggy conosca l'informazione segreta per aprire la porta; infatti, ad ogni esecuzione del protocollo, Peggy sceglie casualmente la direzione ed analogamente, in modo del tutto casuale, Victor indica la direzione da seguire 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 aprire la porta; in tal modo, se il protocollo venisse ripetuto 10 volte e se in ognuna delle esecuzioni Peggy fosse uscita sempre dalla parte indicata da Victor, allora la probabilità che si verifichi l'evento $E = \{Peggy\,non\,sa\,aprire\,la\,porta\}$, e, dunque, che Peggy riesca a raggirare Victor, considerando che lo stesso verifier ha scelto a caso il percorso tra l'insieme delle direzioni possibili $\{A,B\}$, è $P(E) = (\frac{1}{2})^{10} = \frac{1}{1024} \sim 9.8 \cdot 10^{-4}$;
perciò *se Peggy appare in modo 'affidabile' all'uscita secondo le indicazioni di Victor, lo stesso può concludere che molto probabilmente conosca davvero la parola segreta.*

Si osservi che dall'esecuzione del protocollo il verifier, Victor, non ottiene una prova matematica del fatto che il prover, Peggy, sia in grado di aprire la porta ma ne ricava un'evidenza sperimentale tramite una serie di challenge-response.

Un'ulteriore considerazione riguarda il fatto che se Victor registrasse con una telecamera la scena della caverna ed un avversario cercasse di estrarre informazioni utili dalla registrazione, di fatto l'attaccante non ricaverebbe alcuna informazione significativa; l'avversario, infatti, non saprebbe, come aprire la porta, come convincere qualcun altro che il segreto sia noto e non avrebbe la sicurezza che Peggy sia riuscita realmente ad aprire la porta perché potrebbe ipotizzare che sia accordata con Victor sulla sequenza delle direzioni da scegliere nell'esecuzione del protocollo.

Nei protocolli a conoscenza zero il termine 'Zero Knowledge' indica 'qualcosa di più forte' rispetto al fatto che la password, o in generale, l'informazione segreta, non venga rilevata in alcun passaggio; infatti tale proprietà viene già garantita dagli schemi di challenge-response.

Tornando all'esempio della caverna, il protocollo ZKP garantisce che nessuna informazione utile di alcun tipo venga ricavata dall'esecuzione se non l'identità della parte rappresentata dal prover Peggy; per tale motivo è fondamentale che il verifier Victor scelga i messaggi da inviare in modo casuale, potendo così generare la stessa sequenza con la medesima distribuzione di probabilità.

Le dimostrazioni a conoscenza zero sono utilizzate per dimostrare di saper risolvere i problemi NP; nella sezione dedicata alla trattazione di tali problemi è stato evidenziato

che *per i problemi NP non è possibile trovare una soluzione in tempi ragionevoli seppur sia possibile dimostrare polinomialmente la correttezza di una soluzione proposta*;
a riguardo la sicurezza di molti algoritmi di cifratura è basata proprio sui problemi NP e, pertanto, è ovvio che chiunque riesca a trovare un algoritmo polinomiale che li risolva sia interessato a mantenere segreta la strategia utilizzata.
Le dimostrazioni a conoscenza zero, dunque, permettono di dimostrare di aver trovato una soluzione effettivamente valida ed efficace senza rilevare alcuna informazione che possa permettere ad altri la strategia utilizzata.

Come esempi di dimostrazioni a conoscenza zero dei problemi trattati, prima di introdurre i *protocolli di Fiat-Shamir* e di *Fiege-Fiat-Shamir*, è indispensabile definire le *proprietà che caratterizzano un protocollo ZKP* ed evidenziare come tali proprietà siano soddisfatti nei seguenti protocolli che utilizzano i noti problemi del logaritmo discreto e della radice quadrata modulo N.

Definizione 106 (Proprietà di un protocollo ZKP) *Siano P e V, rispettivamente prover e verifier, due interlocutori che intendono utilizzare un protocollo ZKP.*
I due interlocutori possono essere onesti, cioè sia P che V rispettano esattamente il protocollo concordato e P possiede effettivamente la conoscenza affermata, o disonesti nel caso in cui alcune delle caratteristiche indicate non siano rispettate.
Affinché il protocollo eseguito da P e V si possa definire ZKP debbono essere rispettare le seguenti tre proprietà:

- ***Completezza**: se l'affermazione di P è vera, V ne accetterà sempre la dimostrazione ovvero un prover onesto potrà sempre convincere del fatto un verifier onesto;*
- ***Correttezza**: se l'affermazione di P è falsa, ovvero P è disonesto, V può essere convinto della veridicità dell'affermazione solo con "bassa" probabilità, cioè con una probabilità $\leq \frac{1}{2^k}$ per un valore arbitrario k scelto da lui stesso; alternativamente la correttezza può essere riformulata affermando che se P è onesto, P sarà in grado di convincere V con probabilità $\geq 1 - \frac{1}{2^k}$;*
- ***Conoscenza zero**: se l'affermazione di P è vera, nessun verificatore V, anche se disonesto nell'uso del protocollo, può acquisire alcuna informazione su tale fatto salvo la sua veridicità.*

Protocollo 26 (Un protocollo ZKP basato sul logaritmo discreto) *Siano P e V due utenti che intendono utilizzare un protocollo ZKP; sia P il prover e V il verifier che deve verificare l'identità di P. P vuole dimostrare di saper calcolare il logaritmo discreto di un dato numero ovvero sia p un numero primo sufficientemente elevato, $g \in Gen_{\mathbb{Z}_p^*}$, e $b \in \mathbb{Z}_p^*$, P vuole dimostrare di saper calcolare $g^x = b \bmod p$, con $x \in \mathbb{Z}_p$.*
P e V conoscono entrambi il generatore g e l'elemento b del gruppo generato.

Lo schema del protocollo ZKP basato sul logaritmo discreto è il seguente:

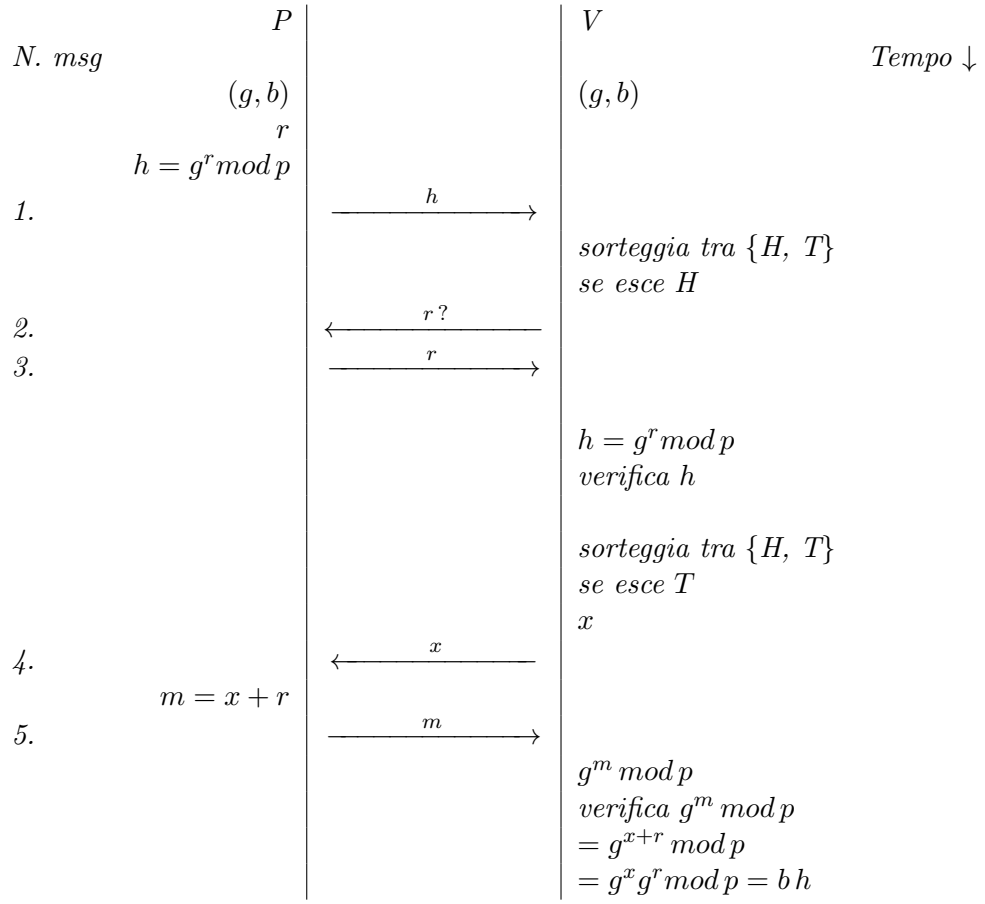


Diagramma 36: Diagramma di sequenza di un protocollo ZKP basato sul logaritmo discreto.

Di seguito l'analisi dei messaggi tra P ed V:

- *P genera un elemento r in modo casuale, calcola $h = g^r mod\, p$ ed invia h a V con il messaggio* 1*;*
- *V sceglie in maniera casuale una tra due possibili opzioni, indicate con H (Head) e T (Tails) ovvero come l'operazione di lancio di una moneta in cui i due eventi possibili sono equiprobabili; se esce Head (H), V invia a P la richiesta di r con il messaggio* 2*;*
- *P risponde al challenge di V inviando r con il messaggio* 3*;*
- *V, ricevuto r, può, quindi, calcolare $h = g^r mod\, p$, conoscendo g e r, e verificare se effettivamente h, precedentemente ricevuto da P, corrisponde al calcolo di $g^r mod\, p$ ovvero se P sa calcolare il logaritmo discreto;*
- *Se, invece, esce Tails (T), V genera un elemento casuale x e lo invia a P con il messaggio* 4*;*

- *P risponde al challenge di V calcolando $m = x + r$ ed inviandolo a P con il messaggio 5;*
- *V, ricevuto m, può, quindi, controllare la correttezza della risposta di P, conoscendo b ed h, dove $b = g^x \, mod \, p$ e $h = g^r \, mod \, p$; infatti*

  $g^m \, mod \, p = g^{x+r} \, mod \, p = g^x g^r mod \, p = b \, h$;
- *i passi a partire dalla generazione del numero casuale r, da parte di P, alle verifiche da parte di V, a seguito delle risposte di P, a seconda del verificarsi dei due eventi casuali indicati, vengono ripetuti finché V non è convinto del fatto che P conosca il logaritmo discreto x; la convinzione di V è, dunque, basata sulle probabilità che si verifichi l'evento E= $\{P$ conosce il logaritmo discreto di $x\}$ con*

$$\mathbb{P}(E) = 1 - \frac{1}{2^k} \, ,$$

  dove k è il numero di volte per cui viene ripetuto l'algoritmo.

Si osservi che nel protocollo precedente, basato sul logaritmo discreto, una sola interazione non è sufficiente a convincere il verifier V dell'onestà del prover P; infatti ogni volta che esce H (Head), P non dimostra nulla di significativo a V; d'altra parte se esce T (Tails), P potrebbe tentare di imbrogliare; dunque l'esecuzione di una sola iterazione non assicura a V l'onestà di P.

La sicurezza del protocollo, come evidenziato dalla formula relativa alla probabilità, che P conosca x, migliora con l'aumentare delle iterazioni tenendo però conto della seguente considerazione; se da una parte, scegliendo un K elevato, V ha maggiori garanzie sull'onestà di P, dall'altra V aumenta il costo computazionale ed il tempo necessario per l'esecuzione dell'algoritmo.

Dall'osservazione precedente ne deriva che una prova a conoscenza zero one-round che dimostrasse la validità dell'algoritmo polinomiale in un'unica iterazione, risulterebbe estremamente vantaggiosa; infatti non solo ridurrebbe il costo computazionale ma anche il tempo necessario all'esecuzione della prova permettendo di limitare l'impiego di risorse.

Un protocollo di tipo challenge-respose che dimostra l'onestà del prover P in una sola iterazione è stato proposto da Almuhammadi e Neumann della University of Southtern California.

Protocollo 27 (Protocollo ZKP di Almuhammadi-Neumann) *Siano P e V due utenti che intendono utilizzare un protocollo ZKP one-round; sia P il prover e V il verifier che deve verificare l'identità di P. P vuole dimostrare di saper calcolare il logaritmo discreto di un dato numero ovvero sia p un numero primo sufficientemente elevato, $g \in Gen_{\mathbb{Z}_p^*}$, e $b \in \mathbb{Z}_p^*$, P vuole dimostrare di saper calcolare $g^x = b \, mod \, p$, con $x \in \mathbb{Z}_p$. P e V conoscono entrambi il generatore g e l'elemento b del gruppo generato.*

Lo schema del protocollo ZKP one-round di Almuhammadi-Neuman basato sul logaritmo discreto è il seguente:

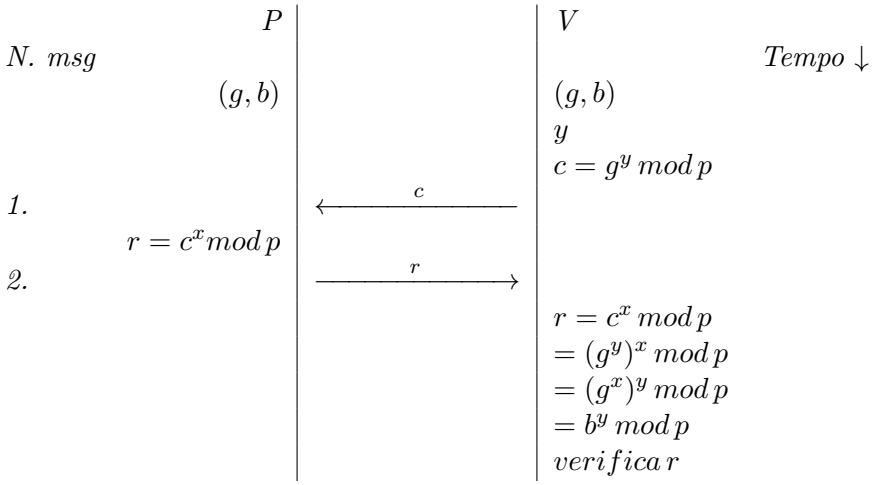


Diagramma 37: Diagramma di sequenza del protocollo ZKP one-round di Almuhammadi-Neuman basato sul logaritmo discreto.

Di seguito l'analisi dei messaggi tra P ed V:

- *V genera un elemento y in modo casuale, calcola $c = g^y mod\, p$ ed invia c a P con il messaggio* 1*;*
- *P risponde al challenge calcolando $r = c^x mod\, p$ ed inviando il risultato a V con il messaggio* 2*;*
- *V verifica che effettivamente $r = c^x mod\, p$ conoscendo b ed y ed essendo*

$$r = c^x \, mod\, p = (g^y)^x \, mod\, p = (g^x)^y \, mod\, p = b^y \, mod\, p.$$

Il protocollo proposto da Almuhammadi-Neumann basato sul logaritmo discreto è di tipo ZKP e soddisfa le proprietà di completezza, correttezza e zero-knowledge.

Se P conosce il segreto x può facilmente calcolare $r = c^x \, mod\, p$ ed inviare il risultato a V che ne verifica, come visto, la validità calcolando $b^y \, mod\, p$; tale verifica è sufficiente a convincere V del fatto che P sa effettivamente calcolare il logaritmo discreto.
Se, invece, P non conosce x non ha alcuna possibilità di calcolare in tempo polinomiale $(g^y)^x \, mod\, p$ ed ingannare così V, in accordo con il problema di Diffie-Helmann (PDH) che afferma l'impossibilità di calcolare in tempo polinomiale $(g^y)^x \, mod\, p$ conoscendo solamente $g^x \, mod\, p$ e $g^y \, mod\, p$.
A riguardo si osserva che se è vero che risolvere il problema del logaritmo discreto (PDL) permette di risolvere il problema di Diffie-Helmann (PDH), il contrario, ovvero risolvere il PDH permette di risolvere il PDL, è stato solo ipotizzato da Diffie-Helmann ma tale congettura non è stata ancora provata.

Dunque le proprietà di completezza e correttezza sono soddisfatte dal protocollo di Almuhammadi-Neumann grazie al problema di Diffie-Helmann: l'impossibilità di calco-

lare $r = c^x \, mod \, p$ conoscendo solo $b = g^x \, mod p$ e $c = g^y \, mod p$ assicura che la dimostrazione venga sempre accettata (*completezza*) e garantisce l'impossibilità di dare false dimostrazioni (*correttezza*).
Infine anche *la proprietà di conoscenza zero risulta soddisfatta non essendo il segreto x mai rivelato e tenendo conto che ricavare x da r, inviato da P a V, è proprio l'operazione di calcolo del logaritmo discreto che V non sa risolvere.*

Protocollo 28 (Un protocollo ZKP basato sulle radici quadrate mod p) *Siano P e V due utenti che intendono utilizzare un protocollo ZKP; sia P il prover e V il verifier che deve verificare l'identità di A. P vuole dimostrare di saper calcolare le radici quadrate modulo p di un dato numero a ovvero sia p un numero primo sufficientemente elevato e $a \in \mathbb{Z}_p^*$, P vuole dimostrare di saper calcolare x tale che $x^2 = a \, mod \, p$, con $x \in \mathbb{Z}_p^*$ ovvero a è un residuo quadratico modulo p.*
P e V stabiliscono a e p ed eseguono i passi indicati nel seguente schema.
Lo schema del protocollo ZKP basato sulle radici quadrate mod p è il seguente:

	P		V
N. msg			*Tempo* $\downarrow$
	(p, a)		(p, a)
	r		
	$s = r^2 mod \, p$		
1.		$\xrightarrow{s}$	
			sorteggia tra $\{H, \, T\}$
			se esce H
2.		$\xleftarrow{r \, ?}$	
3.		$\xrightarrow{r}$	
			$r^2 = s \, mod \, p$
			verifica r^2
			sorteggia tra $\{H, \, T\}$
			se esce T
			x
4.		$\xleftarrow{x}$	
	$m = r \cdot x$		
5.		$\xrightarrow{m}$	
			$m^2 \, mod \, p$
			$= (r \cdot x)^2 \, mod \, p$
			$= r^2 \cdot x^2 \, mod \, p = s \cdot a \, mod \, p$
			verifica m^2

Diagramma 38: Diagramma di sequenza di un protocollo ZKP basato sulle radici quadrate mod p .

Di seguito l'analisi dei messaggi tra P ed V:

- *P genera un elemento r in modo casuale, calcola $s = r^2 mod\, p$ ed invia s a V con il messaggio* 1*;*
- *V sceglie in maniera casuale una tra due possibili opzioni, indicate con H (Head) e T (Tails) ovvero come l'operazione di lancio di una moneta in cui i due eventi possibili sono equiprobabili; se esce Head (H), V invia a P la richiesta di r con il messaggio* 2*;*
- *P risponde al challenge di V inviando r con il messaggio* 3*;*
- *V, ricevuto r, può calcolare $r^2 = s \,mod\, p$, conoscendo s e p, e verificare se effettivamente s, ricevuto da P, è il residuo quadratico di $r^2 = s \,mod\, p$ ovvero se P sa calcolare le radici quadrate di s modulo p;*
- *Se, invece, esce Tails (T), V invia a P un elemento casuale x con il messaggio* 4*;*
- *P risponde al challenge di V inviando m calcolato come $r \cdot x$ con il messaggio* 5*;*
- *V, ricevuto m, può, quindi, controllare la correttezza della risposta di P, conoscendo s ed a; infatti $m^2 \,mod\, p = (r \cdot x)^2 \,mod\, p = r^2 \cdot x^2 \,mod\, p = s \cdot a \,mod\, p$*
- *i passi a partire dalla generazione del numero casuale r, da parte di P, alle verifiche da parte di V, a seguito delle risposte di P, a seconda del verificarsi dei due eventi casuali indicati, vengono ripetuti finché V non è convinto del fatto che P sappia calcolare le radici quadrate modulo p di un dato numero a; la convinzione di V è, dunque, basata sulle probabilità che si verifichi l'evento E= $\{P$ conosce le radici quadrate mod p di un dato numero $a\}$ è*

$$\mathbb{P}(E) = 1 - \frac{1}{2^k}\,,$$

dove k è il numero di volte per cui viene ripetuto l'algoritmo.

Con un ragionamento analogo a quello seguito per il protocollo di Almuhammadi-Neumann basato sul logaritmo discreto, si dimostra anche il protocollo basato sulle radici quadrate mod p è di tipo ZKP e soddisfa le proprietà di completezza, correttezza e conoscenza zero.
Se P conosce il segreto r può facilmente calcolare $s = r^2 \,mod\, p$ ed inviare il risultato a V che ne verifica, come visto, la validità calcolando $s \,mod\, p$; tale verifica è sufficiente a convincere V del fatto che P sa effettivamente calcolare le radici quadrate $mod\, p$.
Se, invece, P non conosce r non ha alcuna possibilità di calcolare in tempo polinomiale $r^2 \cdot x^2 \,mod\, p$ ed ingannare così V, *in accordo con il teorema dell'equivalenza tra fattorizzazione e QRP computazionale.*
Ne deriva che l'impossibilità di calcolare $m = r \cdot x \,mod\, p$, conoscendo solo x in un tempo polinomiale, assicura che la dimostrazione venga sempre accettata (*completezza*) e garantisce l'impossibilità di dare false dimostrazioni (*correttezza*).
Infine anche la proprietà di *conoscenza zero* risulta soddisfatta non essendo il segreto r mai rivelato e tenendo conto che ricavare r da m, inviato da P a V, è proprio l'operazione di calcolo della radice quadrata $mod\, p$ che V non sa risolvere.

3.2.17 Protocollo Fiat-Shamir

Il primo esempio noto di protocollo ZKP venne ideato da Fiat e Shamir nel 1988 allo scopo di consentire un'autenticazione sicura all'utente senza fornire alcun'altra informazione utile oltre a quella strettamente richiesta.
La sicurezza del protocollo di Fiat-Shamir è basata sulla difficoltà computazionale del calcolo della radice quadrata modulo n di un intero che, come illustrato, risulta computazionalmente equivalente alla ricerca della fattorizzazione di un numero n.

Protocollo 29 (Protocollo Fiat-Shamir) *Siano A e B due utenti che intendono utilizzare un protocollo ZKP; sia A il prover e B il verifier che deve verificare l'identità di B. Il protocollo Fiat-Shamir si implementa con i seguenti passi.*
Setup:

- *A sceglie due numeri primi p, q primi, che mantiene segreti, e calcola $n = p \cdot q$;*
- *A genera un numero casuale $s \in \mathbb{Z}_p^*$ tale $(s, n) = 1$, e $1 \leq s \leq n-1$ e lo tiene segreto; tale numero è la sua chiave privata, ovvero $K_A^- = s$;*
- *A calcola $v = s^2 \bmod n$ e lo pubblica insieme ad n, ovvero $K_A^+ = (n, v)$ è la chiave pubblica di A.*

Lo schema del protocollo Fiat-Shamir è il seguente:

N. msg	*A*		*B*	*Tempo* ↓
	$K_A^- = s$			
	$K_A^+ = (n, v)$		$K_A^+ = (n, v)$	
	$r \in \{1, n-1\}$			
	$x = r^2 mod\, n$			
1.		$\xrightarrow{x}$		
			sorteggia e in $\{0, 1\}$	
			se $e = 0$	
2.		$\xleftarrow{e}$		
	$y = r \bmod n$			
3.		$\xrightarrow{y}$		
			y^2	
			verifica di y considerando che	
			$y^2 = r^2 \bmod n = x \bmod n$	
			sorteggia e in $\{0, 1\}$	
			se $e = 1$	
4.		$\xleftarrow{e}$		
	$y = r \cdot s \bmod n$			
5.		$\xrightarrow{y}$		
			y^2	
			verifica di y considerando che	
			$y^2 = (r \cdot s)^2 \bmod n$	
			$= r^2 \cdot s^2 \bmod n = x \cdot v \bmod n$	

Diagramma 39: Diagramma di sequenza del protocollo Fiat-Shamir.

Di seguito l'analisi dei messaggi tra A ed B:

- *A genera un numero casuale r, con $1 \leq r \leq n-1$, calcola $x = r^2 \bmod n$ ed invia x a B con il messaggio 1;*
- *B sceglie in maniera casuale un bit e (challenge) nell'insieme $\{0,1\}$; se si verifica che $e = 0$, B invia ad A tale bit, che presuppone la richiesta di $y = r \bmod n$, con il messaggio 2;*
- *A risponde al challenge di B, $e = 0$, calcolando $y = r \bmod n$ ed inviando y con il messaggio 3;*
- *B, ricevuto y, conoscendo x, può, quindi, calcolare $y^2 = r^2 \bmod n = x \bmod n$, e verificare la correttezza del response di A;*
- *Se, invece, si verifica che $e = 1$, inviando tale challenge, B presuppone la richiesta di $y = r \cdot s \bmod n$ con il messaggio 4 ;*
- *A, ricevuto il challenge $e = 1$, calcola $y = r \cdot s \bmod n$ e invia tale response a B con il messaggio 5;*
- *B, ricevuto y, può controllare la correttezza della risposta di A, conoscendo x ed v; infatti $y^2 = (r \cdot s)^2 \bmod n = r^2 \cdot s^2 \bmod n = x \cdot v \bmod n$;*
- *i passi a partire dalla generazione del numero casuale r, da parte di A, alle verifiche da parte di B, a seguito delle risposte di A, a seconda del verificarsi di uno dei due eventi casuali indicati, vengono ripetuti finché B non è convinto del fatto che A conosca la chiave privata, $K_A^- = s$, corrispondente alla chiave pubblica, $K_A^+ = (n, v)$, a lui nota; la convinzione di B è, dunque, basata sulle probabilità che si verifichi l'evento E= $\{A$ conosce K_A^- corrispondente a $K_A^+\}$ è*

$$\mathbb{P}(E) = 1 - \frac{1}{2^k},$$

dove k è il numero di volte per cui viene ripetuto l'algoritmo.

Osservazione 18 (Debolezze del Fiat-Shamir con una sola iterazione) *Bisogna osservare che una sola iterazione del protocollo Fiat-Shamir non dà alcuna garanzia a B dell'identità di A in base a quanto A dichiara di essere; infatti un avversario E può tentare di impersonare A, pur non conoscendo la chiave segreta s, con due possibili tipi di attacco. Un attacco prevede la generazione del numero casuale r, l'altro, invece, la scoperta di x da parte di E.*
Di seguito, per entrambi i casi, si illustrano le modalità di esecuzione del protocollo con le quali l'attaccante riesce ad ingannare il prover.

Attacco al protocollo Fiat-Shamir con una sola iterazione: caso in cui E genera il numero casuale r.

Sia E l'attaccante che intende ingannare il verifier B nella verifica della sua identità utilizzando il protocollo Fiat-Shamir. Lo schema dell'attacco al protocollo, con la generazione del numero casuale r da parte di E, è il seguente:

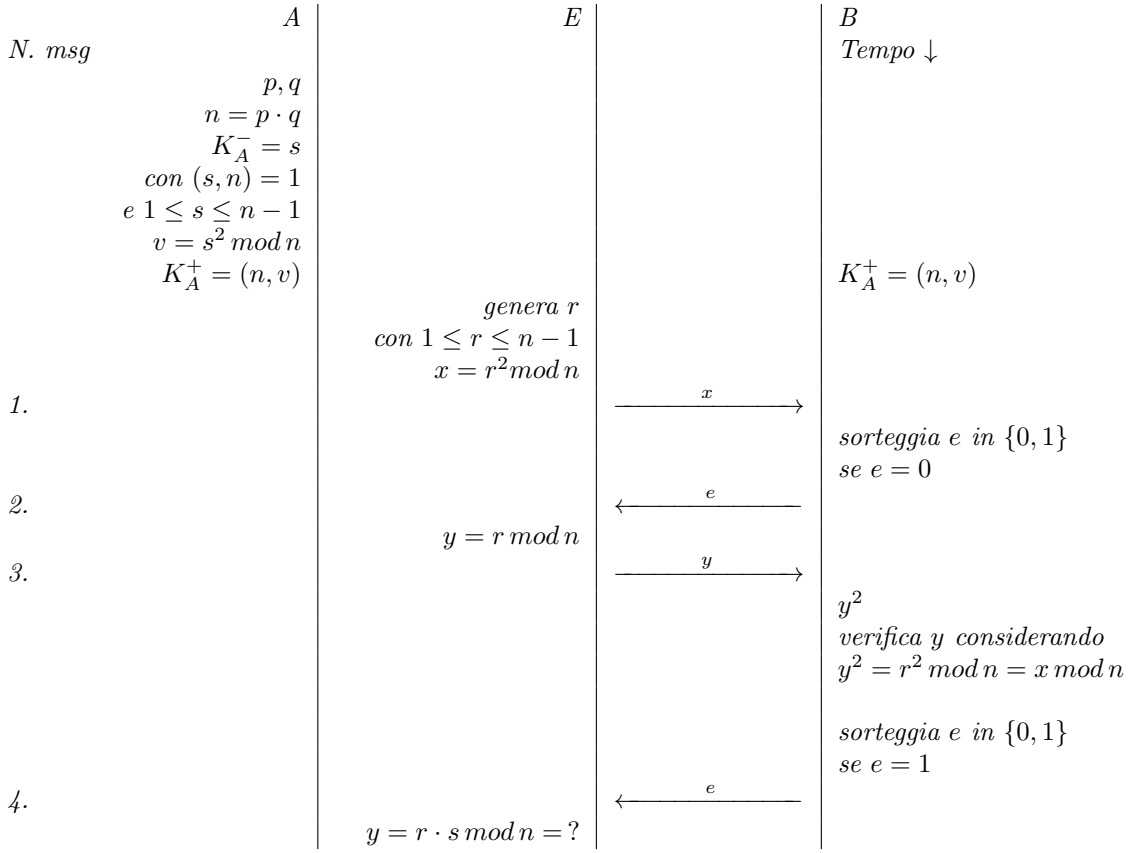


Diagramma 40: Diagramma di sequenza dell'attacco al protocollo di Fiat-Shamir nel caso in cui un avversario genera il numero casuale r con una sola iterazione.

Di seguito l'analisi dei messaggi tra E ed B:

- *E, spacciandosi per il prover A, genera un numero casuale r, con $1 \leq r \leq n-1$, calcola $x = r^2 mod\, n$ ed invia x a B con il messaggio 1;*
- *B sceglie in maniera casuale un bit e (challenge) con $e \in \{0, 1\}$; se $e = 0$, B invia al prover, in realtà ad E, tale challenge, che presuppone la richiesta di $y = r\, mod\, n$, con il messaggio 2;*
- *E, non avendo bisogno della chiave segreta di A, risponde al challenge di B calcolando $y = r\, mod\, n$ ed inviando r con il messaggio 3;*

- *B, ricevuto y, conoscendo x, può, quindi, calcolare* $y^2 = r^2 \bmod n = x \bmod n$*, e verificare la correttezza del response di E che riesce ad ingannare il verifier.*
- *Nel caso in cui, invece, B sorteggia il bit* $e = 1$*, tale challenge presuppone la richiesta di* $y = r \cdot s \bmod n$*; il challenge viene inviato al prover con il messaggio* 4*;*
- *E, ricevuto il bit* $e = 1$*, non sa calcolare* $y = r \cdot s \bmod n$ *e, quindi, non può rispondere al challenge di B che non riuscirà a completare il processo di identificazione, realizzando che E non è realmente chi dichiara di essere.*

Attacco al protocollo Fiat-Shamir con una sola iterazione: caso in cui** E **scopre** x**.
Sia E l'attaccante che intende ingannare il verifier B nella verifica della sua identità utilizzando il protocollo Fiat-Shamir. Lo schema dell'attacco al protocollo con la scoperta di x da parte di E è il seguente:

N. msg	A	E		B
Tempo $\downarrow$	p, q			
	$n = p \cdot q$			
	$K_A^- = s$			
	con $(s, n) = 1$			
	e $1 \leq s \leq n-1$			
	$v = s^2 \bmod n$			
	$K_A^+ = (n, v)$			$K_A^+ = (n, v)$
		conosce $x = r^2 \bmod n$		
		$\frac{r^2}{v}$		
1.			$\xrightarrow{x = r^2/v}$	
				sorteggia e in $\{0, 1\}$
				se $e = 1$
2.			$\xleftarrow{e}$	
		$y = r \bmod n$		
3.			$\xrightarrow{y}$	
				y^2
				verifica y considerando
				$y^2 = x \cdot v \bmod n =$
				$\frac{r^2}{v} v \bmod n = r^2 \bmod n$
				sorteggia e in $\{0, 1\}$
				se $e = 0$
4.			$\xleftarrow{e}$	
		$y^2 = \frac{r^2}{s^2} \bmod n$		
		$y = \sqrt{\frac{r^2}{v}} \bmod n = ?$		

Diagramma 41: Diagramma di sequenza dell'attacco al protocollo di Fiat-Shamir, con una sola iterazione, nel caso in cui un avversario scopre x.

Di seguito l'analisi dei messaggi tra E ed B:

- *E, avendo scoperto x, dove $1 \leq x \leq n-1$, calcola $\frac{r^2}{v} \bmod n$, considerato che $x = r^2 \bmod n$ e v è nota, essendo $K_A^+ = (n, v)$, ed invia tale valore come x a B con il messaggio 1;*
- *B sceglie in maniera casuale un bit e, challenge, con $e \in \{0, 1\}$; se $e = 0$, B invia ad A tale bit, che presuppone la richiesta di $y = r \bmod n$, con il messaggio 2;*
- *E, non avendo bisogno della chiave segreta di A, risponde al challenge di B calcolando $y = r \bmod n$ ed inviandolo con il messaggio 3;*
- *B, ricevuto y, può, quindi, calcolare $y^2 = r^2 \bmod n = \frac{r^2}{v} v \bmod n = x \cdot v \bmod n$ e verificare la correttezza del response di E che riesce, quindi, ad ingannare il verifier;*
- *Se, invece, B sceglie in modo casuale il bit $e = 1$, inviando tale challenge con il messaggio 4, presuppone la richiesta di $y = r \cdot s \bmod n$;*
- *E, ricevuto il bit $e = 1$, essendo $y^2 = \frac{r^2}{s^2} \bmod n = \frac{r^2}{v} \bmod n$, per rispondere al challenge di B di E dovrebbe calcolare $y = \sqrt{\frac{r^2}{v}} \bmod n$ ma la radice quadrata $\bmod n$ è proprio l'operazione computazionalmente complessa che non sa risolvere, non conoscendo la chiave segreta s; l'attaccante, dunque, non riesce a completare il processo di identificazione e, di conseguenza, il verifier B realizza che E non è realmente chi dichiara di essere.*

Dagli attacchi illustrati si evidenzia che se il protocollo di Fiat-Shamir viene eseguito una sola volta, la probabilità che un avversario E riesca ad identificarsi al posto di A, senza conoscere la chiave privata, è di $\frac{1}{2}$, potendo rispondere solo ad uno dei due challenge di B; dunque, come anticipato, una sola iterazione non è sufficiente per garantire la sicurezza del protocollo che, invece, migliora all'aumentare delle esecuzioni; ad ogni esecuzione, infatti, la probabilità che l'avversario E riesca ad identificarsi si dimezza e, quindi, dopo k iterazioni, diventa $\frac{1}{2^k}$.
In analogia con quanto visto finora per i precedenti protocolli ZKP, per ottenere un'adeguata sicurezza nell'esecuzione del Fiat-Shamir, il numero delle iterazioni k deve essere elevato con conseguente aumento del costo computazionale e del tempo necessario al completamento dell'autenticazione del prover.

Al fine di migliorare le prestazioni del protocollo Fiat-Shamir sono stati sviluppati diversi algoritmi; a riguardo, si conclude la trattazione dei protocolli ZKP con la trattazione del protocollo di Feige-Fiat-Shamir (FFS), illustrato sia nella versione base che nella versione revisionata; analogamente al Feige-Fiat, il protocollo FFS utilizza una

coppia di chiavi pubblica-privata; il FFS introduce un miglioramento da un punto di vista computazionale rispetto al Fiat-Shamir in quanto velocizza il processo di verifica tra i partecipanti finalizzato all'acquisizione della fiducia da parte del prover.

3.2.18 Protocollo Feige-Fiat-Shamir

Protocollo 30 (Protocollo Feige-Fiat-Shamir (originale)) *Siano A e B due partecipanti che intendono utilizzare un protocollo ZKP; sia A il prover e B il verifier che deve verificare l'identità di B. Il protocollo Feige-Fiat-Shamir si implementa con i seguenti passi. Setup:*

- *A sceglie $n = p \cdot q$ con p, q numeri primi, che mantiene segreti, tali che $p \equiv 3 \, mod \, 4$ e $q \equiv 3 \, mod \, 4$ (in tal modo p e q sono due interi di Blum) e pubblica n;*
- *A sceglie due interi k e t come parametri di sicurezza;*
- *Usando n, A sceglie un vettore segreto $s = \{s_1, s_2, \cdots, s_k\}$ con $1 \leq s_i \leq n-1$ e $(s_i, n) = 1$ da utilizzare come chiave privata, ovvero $K_A^- = s$, ed un vettore di c k random bit $c = \{c_1, c_2, \cdots, c_k \, , \, c_i \in \{0,1\} \, , \, i = 1, \cdots, k\}$;*
- *A computa il vettore $v = \{v_1, v_2 \cdots, v_k : v_i \equiv s_i^2 \, mod \, n \, , \, i = 1, \cdots, k\}$; utilizza quindi come chiave pubblica $K_A^+ = (v, n)$;*

Lo schema del protocollo Feige-Fiat-Shamir (originale) è il seguente:

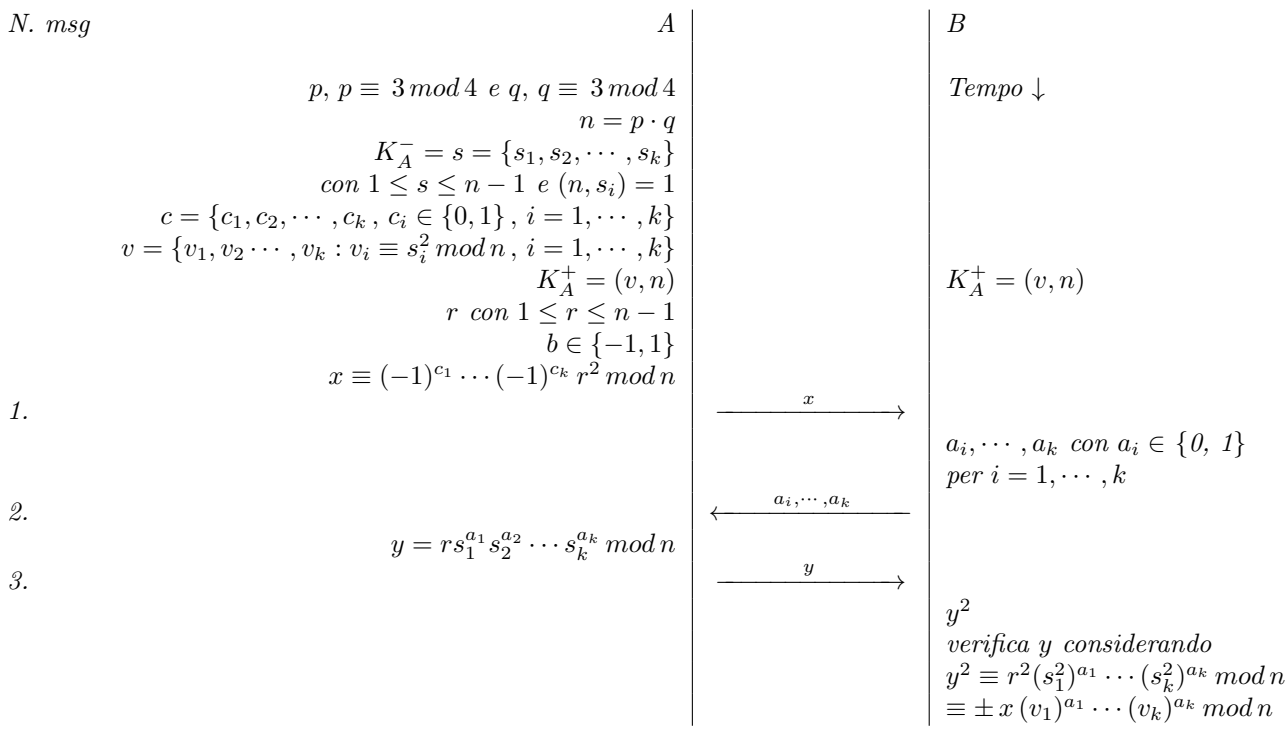


Diagramma 42: Diagramma di sequenza del protocollo Feige-Fiat-Shamir (originale)

Di seguito l'analisi dei messaggi tra A ed B:

- *A genera un numero casuale r, con $1 \leq r \leq n-1$, un segno casuale $b \in \{-1, 1\}$, computa $x \equiv (-1)^{c_1} \cdots (-1)^{c_k} r^2 \, mod \, n$ ed invia x a B con il messaggio 1;*
- *B sceglie in maniera casuale un challenge di bit $a_i, \cdots, a_k$ con $a_i \in \{0, 1\}$, con $i = 1, \cdots, k$, ed invia ad A tali bit con il messaggio 2;*
- *A risponde al challenge di B calcolando $y = r s_1^{a_1} s_2^{a_2} \cdots s_k^{a_k} \, mod \, n$ ed inviando y con il messaggio 3;*
- *B, ricevuto y può calcolare $y^2 \equiv r^2 (s_1^2)^{a_1} \cdots (s_k^2)^{a_k} \, mod \, n$ e, tenendo conto che $x \equiv (-1)^{c_1} \cdots (-1)^{c_k} r^2 \, mod \, n$, diventa $y^2 \equiv \pm x \, (v_1)^{a_1} \cdots (v_k)^{a_k} \, mod \, n$ e, dunque, conoscendo x e $K_A^+ = (v, n)$, può verificare la correttezza del response di A;*
- *La procedura è ripetuta con differenti valori di r e a_i, con $i = 1, \cdots, k$, finché il verifier B non è convinto che il prover B conosca le radici quadrate $s_1, \cdots, s_k$, modulo n dei suoi numeri $v_1, \cdots, v_k$.*

Osservazione 19 ("Problema del ping-pong" nel Feige-Fiat-Shamir) *Sebbene lo schema di identificazione del protocollo Feige-Fiat-Shamir sia il più utilizzato tra quelli ZKP che utilizzano l'aritmetica modulare, esso soffre del cosiddetto "problema del ping-pong"; tale problema, come indica il nome, deriva dai ripetuti scambi di tipo challenge-response tra il verifier e il prover dovuti al fatto che il verifier cerca di ottenere quante più informazioni possibili dal prover per completare il processo di autenticazione nel più breve possibile possibile; d'altro canto, per ragioni di sicurezza, il prover fornisce le informazioni necessarie al processo di autenticazione senza rivelare la sua identità.*

Il principale effetto del "problema del ping-pong" è il consumo di risorse dei sistemi coinvolti nell'esecuzione del protocollo; inoltre, in alcuni ambienti "time critical", i tempi di esecuzione del protocollo Feige-Fiat-Shamir, necessari per ottenere la fiducia da parte del verifier, non sempre sono compatibili con le esigenze dei sistemi; d'altro canto una riduzione dei tempi del "ping-pong" comporterebbe inevitabilmente una minor fiducia, e di conseguenza una insicurezza nel processo di identificazione.

A riguardo si illustra la seguente revisione del protocollo Feige-Fiat-Shamir mirata ad accelerare il processo originale per instaurare rapidamente la fiducia del verifier nei confronti del prover; il protocollo prevede che il verifer autentichi il prover, sulla base di un suo token segreto, attraverso una serie limitata di challange senza che tale token sia mai rivelato al verifier.

Protocollo 31 (Protocollo Feige-Fiat-Shamir (revisionato)) *Siano A e B due utenti che intendono utilizzare un protocollo ZKP; sia A il prover e B il verifier che deve verificare l'identità di B. Il protocollo Feige-Fiat-Shamir (revisionato) si articola nei seguenti passi.*

Setup:

- *A sceglie* $n = p \cdot q$ *con* p, q *numeri primi, che mantiene segreti, tali che* $p \equiv 3 \, mod \, 4$ *e* $q \equiv 3 \, mod \, 4$ *(in tal modo* p *e* q *sono due interi di Blum) e pubblica* n*;*
- *A sceglie due interi* k *e* t *come parametri di sicurezza;*
- *Usando* n*, A sceglie un vettore segreto* $s = \{s_1, s_2, \cdots, s_k\}$ *con* $1 \leq s_i \leq n-1$ *e* $(s_i, n) = 1$ *da utilizzare come chiave privata, ovvero* $K_A^- = s$*, ed un vettore di* c k *random bit* $c = \{c_1, c_2, \cdots, c_k \, , \, c_i \in \{0,1\} \, , \, i = 1, \cdots, k\}$*;*
- *A computa il vettore* $v = \{v_1, v_2 \cdots, v_k : v_i \equiv s_i^2 \, mod \, n \, , \, i = 1, \cdots, k\}$*; utilizza quindi come chiave pubblica* $K_A^+ = (v, n)$*;*

Lo schema del protocollo Feige-Fiat-Shamir (revisionato) è il seguente:

N. msg	*A*		*B*
	p, $p \equiv 3 \, mod \, 4$ *e* q, $q \equiv 3 \, mod \, 4$		*Tempo* $\downarrow$
	$n = p \cdot q$		
	$K_A^- = s = \{s_1, s_2, \cdots, s_k\}$		
	con $1 \leq s \leq n-1$ *e* $(n, s_i) = 1$		
	$c = \{c_1, c_2, \cdots, c_k \, , \, c_i \in \{0,1\} \, , \, i = 1, \cdots, k\}$		
	$v = \{v_1, v_2 \cdots, v_k : v_i \equiv s_i^2 \, mod \, n \, , \, i = 1, \cdots, k\}$		
	$K_A^+ = (v, n)$		$K_A^+ = (v, n)$
	r *con* $1 \leq r \leq n-1$		
	$b \in \{-1, 1\}$		
	$x \equiv (-1)^{c_1} \cdots (-1)^{c_k} \, r^2 \, mod \, n$		
1.		$\xrightarrow{x}$	
			$a_i, \cdots, a_k$ *con* $a_i \in \{0, 1\}$ *per* $i = 1, \cdots, k$ $D = \{(a_{ij}), i = 1, \cdots, k, \; j = 1, \cdots, f, \, con \, f < k\}$ *sottomatrice di* $M = \{(a_{ij}), i = 1, \cdots, K \; j = 1, \cdots, f, \, con \, f < k\}$
2.		$\xleftarrow{D}$	
	$y = \{y_1, y_2, \cdots y_f : y_i = r \, \Pi \, (s_i)^{a_{ij}} \, mod \, n \, ,$ *con* $i = 1, \cdots, k$, $j = 1, \cdots, f\}$		
3.		$\xrightarrow{y}$	
			y^2
			verifica $y = \{y_1, y_2, \cdots y_f\}$ *con* $y_i^2 \equiv r^2 \Pi \, (s_i^2)^{a_{ij}} \, mod \, n$ $\equiv \pm \, x \, \Pi \, (v_i)^{a_{ij}} \, mod \, n$, *con* $i = 1, \cdots, k \, , \, j = 1, \cdots, f$

Diagramma 43: Diagramma di sequenza del protocollo Feige-Fiat-Shamir (revisionato)

Di seguito l'analisi dei messaggi tra A ed B:

- *A genera un numero casuale r, con* $1 \leq r \leq n-1$*, un segno casuale* $b \in \{-1, 1\}$*, computa* $x \equiv (-1)^{c_1} \cdots (-1)^{c_k} r^2 \, mod \, n$ *ed invia x a B con il messaggio* 1*;*

- *ogni volta che B sceglie in modo casuale un challenge esistono* $(c_i)^k$*, con* $i = 1, \cdots, k$*, possibili modo di scegliere il vettore* $a = \{a_1, \cdots, a_k\}$ *con* $a_i \in \{0, 1\}$*; tali scelte sono rappresentate dalla matrice* $M = \{(a_{ij}),\ i = 1, \cdots, k,\ j = 1, \cdots, f\}$ *composta di* 0 *e* 1*; B sceglie, in modo casuale, un numero di righe della matrice M creando una sottomatrice* $D = \{(a_{ij}),\ i = 1, \cdots, k,\ j = 1, \cdots, f, \, con \, f < k)\}$ *di M ed invia simultaneamente tale sottomatrice, con i bit di challenge, ad A con il messaggio* 2*;*

- *A, invece di computare il solo numero y, a differenza di quanto avviene nel protocollo originale, calcola* $y = \{y_1, y_2, \cdots, y_f : y_i = r \, \Pi \, (s_i)^{a_{ij}} \, mod \, n,\ i = 1, \cdots, k,$ $j = 1, \cdots, f\}$ *e risponde al challenge di B inviando il vettore con il messaggio* 3*;*

- *B, ricevuto il vettore y, può, quindi, verificare la correttezza del response di A ovvero che ogni componente* y_i *soddisfi* $y_i \equiv \pm \, x \, \Pi \, (v_i)^{a_{ij}} \, mod \, n,\ i = 1, \cdots, k,$ $j = 1, \cdots, f$*, conoscendo x e* $K_A^+ = (v, n)$ *ed essendo* $y_i^2 \equiv r^2 \Pi \, (s_i^2)^{a_{ij}} \, mod \, n \equiv \pm \, x \, \Pi \, (v_i)^{a_{ij}} \, mod \, n, i = 1, \cdots k, j = 1, \cdots f.$

- *La procedura è ripetuta con differenti valori di r e sottomatrici* $D = \{(a_{ij}),\ i = 1, \cdots, k,\ j = 1, \cdots, f, \, con \, f < k)\}$ *finché il verifier B non è soddisfatto che il prover B conosca le radici quadrate* $s_1 \cdots s_k$ *modulo n dei suoi numeri* $v_1 \cdots v_k$*.*

Lo schema del protocollo FFS revisionato consente di ottenere enormi risparmi di tempo nell'ottenimento della fiducia da parte del prover nel rispetto della sicurezza.
Nell'esecuzione del protocollo, essendo basato sullo schema del FFS, il prover non fornisce alcuna informazione utile al verifier durante l'intero processo; pertanto, nel caso in cui si verificassero eventuali intercettazioni nella comunicazione, l'attaccante non riuscirebbe a ricavare alcuna informazione utile dai messaggi inviati dal prover; il nuovo schema del FFS, infatti, non aggiunge alcun elemento al FFS originale che potrebbe compromettere il mantenimento del segreto da parte del prover; quindi la sicurezza del protocollo è assicurata.

Come detto, il maggiore contributo apportato dallo schema del protocollo revisionato consiste nella rapida costruzione della fiducia da parte del verifier e della confidenza per il prover. Il ragionamento seguente illustra un esempio di calcolo in termini probabilistici di tale fiducia evidenziandone i conseguenti benefici.
Si supponga che gli interi di sicurezza scelti dal prover siano k e 1; dunque il verifier ha $k!$ modi di scegliere un vettore di 0 e 1 di lunghezza k; a riguardo si consideri la matrice M definita nello schema. Si ipotizzi inoltre che, nel rispetto del protocollo, il verifier decida di inviare al prover parecchi di tali vettori, in una sola volta, utilizzando una sottomatrice D di M di tali scelte; il prover, di conseguenza, risponde con l'invio del vettore $y = \{y_1, y_2, \cdots, y_f\}$. Per ogni invio parallelo nella matrice D di f righe, con

k bit ciascuna, che il prover restituisce correttamente, la fiducia del verifier cresce con la probabilità:

$$1 - ((\tfrac{1}{2})^f)^k;$$

in tal modo per t invii, il contributo alla fiducia aumenta con la probabilità:

$$1 - (((\tfrac{1}{2})^f)^k)^t.$$

Risulta evidente, quindi, che con tale accelerazione il processo di autenticazione richieda meno tempo.

Le esecuzioni parallele, proposte del protocollo Feige-Fiat-Shamir revisionato, sono certamente attraenti in quanto riducono il numero di messaggi di "ping-pong" tra il prover ed il verifier; tuttavia è stato dimostrato che tali esecuzioni potrebbero non essere completamente a conoscenza zero (Joe Kilian, Eriz Petrank and Ransom Richardson [2001], M. Konidala Dviyan [2003]); *in ogni caso sono state sviluppate tecniche zero-knowledge concorrenti che ne preservano le proprietà* (Li Lu, Jinsong Han, Yunhao Liu, Lei hu, Jinpeng Huai Lionel Ni and Jian Ma [2006])[7].

Un esempio di applicazione del protocollo Feige-Fiat-Shamir riguarda l'autenticazione di un utente U ad un host H mediante l'utilizzo di una o più password senza che queste vengano in alcun modo inviate (in chiaro o cifrate) nel canale di comunicazione.
Se si analizzano i ruoli di U e H nell'ottica di uno schema ZKP, essi sono rispettivamente quelli del prover e del verifier nel protocollo FFS.
Considerate una stringa, ID_U, che contiene tutte le informazioni necessarie per l'identificazione dell'utente U, una funzione hash f pubblica ed un'autorità fidata A, si può analizzare il seguente scenario:

- A sceglie un numero $n = p \cdot q$ con p, q numeri primi "sufficientemente grandi";
- A calcola $f(ID_U||j_i)^2$ per più valori di j_i con $i = 1, 1 \cdots n$;
- poiché A conosce sia p che q, può facilmente determinare quali valori di $f(ID_U||j_i)$, per j_i fissato, hanno una radice quadrata modulo n, ovvero $f(ID_U||j) \equiv s^2 \, mod \, n$, ed inoltre è in grado di calcolarli; dunque A riesce ad ottenere i numeri
  $v_1 = f(ID_U||j_1), \cdots, v_k = f(ID_U||j_k)$ e le rispettive radici quadrate $s_1, \cdots s_k \, mod \, n$;
- A pubblica i valori di $ID_U, (j_1, \cdots, j_k, n)$, ovvero $K_U^+ = (j_1, \cdots, j_k, n)$;
- A comunica ad U i valori segreti (le password), ovvero $K_U^- = (s_1, \cdots, s_k)$;
- A scarta $(s_1, \cdots, s_k)$, p e q in modo tale che se il suo sistema venisse violato nessuna informazione sensibile di U sarebbe compromessa; inoltre, considerato che ogni utente ha un differente valore di n, non è possibile compromettere la sicurezza di più utenti in un'unica volta.

[7] `https://www.researchgate.net/publication/228974847_Feige-Fiat-Shamir_ZKP_Scheme_Revisited`

Come esempio concreto in tale scenario, si consideri l'utente U che utilizza un ATM o un POS (l'host H). I passi eseguiti sono i seguenti:

- Il sistema legge la carta inserita nel dispositivo e scarica dal database $K_U^+ = (j_1, \cdots, j_k, n)$ relativa all'utente U;
- H calcola, quindi, i valori $v_1 = f(ID_U||j_1), \cdots, v_k = f(ID_U||j_k)$ ed esegue il protocollo FFS per verificare se U conosce $K_U^- = (s_1, \cdots, s_k)$;
- il protocollo viene eseguito più volte finché il sistema del dispositivo non si convince dell'identità di U;
- una volta che il sistema si è convinto dell'identità di U, permette allo stesso di svolgere una delle operazioni consentite quali, ad esempio, un pagamento.

Nell'esempio illustrato i valori segreti, che costituiscono la chiave privata di U, sono memorizzati in un microchip all'interno della card utilizzata da U in modo tale che l'utente necessita solamente di ricordare un PIN per accedere al servizio.

Dalla trattazione dei protocolli ZKP si è avuto modo di evidenziare che tali protocolli offrono alti livelli di sicurezza poiché non richiedono la divulgazione di alcun segreto tra i partecipanti; inoltre la logica che utilizzano non richiede la creazione di nuovi sistemi crittografici per la loro applicabilità; molti protocolli ZKP, infatti, aboliscono l'utilizzo della crittografia abbassando i tempi di esecuzione. Anche la riusabilità non costituisce una minaccia alla sicurezza in quanto nelle successive esecuzioni non sono mai rilasciate informazioni utili che potrebbero essere sfruttate da un avversario o da un partecipante mal intenzionato.

Al di là dei punti di forza illustrati, nell'implementazione dei protocolli ZKP si deve tener presente anche dei seguenti svantaggi.

Osservazione 20 (Svantaggi dei protocolli ZKP) *I protocolli ZKP*

- *richiedono un'alta computazionabilità: l'efficienza assicurata è, infatti, pagata, nei sistemi in cui sono eseguiti, in termini di alti costi di esecuzione e di utilizzo della memoria; pertanto, se paragonati ad altri protocolli che utilizzano sistemi di crittografia, da un punto di vista computazionale, i risultati conseguiti risultano poco convenienti;*
- *non risolvono il problema della sicurezza della trasmissione dell'informazione; seppur un avversario non riesca a ricavare informazioni utili dall'intercettazione dei messaggi, in ogni caso, lo stesso può effettuare una serie di attacchi mirati ad alternarne il contenuto a discapito dell'esecuzione del protocollo e dei partecipanti;*
- *richiedono una complessa implementazione e revisione algoritmica; tale complessità ne costituisce un limite dal punto di vista dello sviluppo e delle problematiche legate al troubleshooting.*

3.3 Caso di studio: implementazione di una web application che utilizza un protocollo di autenticazione "ingenuo" basato su password

Come anticipato nella descrizione del progetto, per quanto riguarda l'aspetto pratico, si è sviluppata *una web application in Java che utilizza un protocollo di autenticazione "ingenuo" basato su password*; si è anche indicato che per l'esecuzione del protocollo avviene mediante l'interazione di virtual machine (VM) nell'ambiente di test implementato; il protocollo prevede che dalla VM con il ruolo di client, denominata *victim01* a seguito degli attacchi a suoi danni di seguito illustrati, un utente tramite il web browser si connette in http o in https, a seconda dello scenario, alla web application per autenticarsi ed accedere a delle pagine protette differenti a seconda dalla sua autorizzazione.
La web application è residente in una VM con il ruolo di server, *server01*, che eroga i servizi http e http; nell'ambiente virtuale è prevista anche una terza VM, *hack01*, dedicata al sistema dell'avversario; da tale VM, sfruttando le debolezze del protocollo mediante una serie di attacchi mirati, l'attaccante riesce ad autenticarsi alla web application utilizzando le credenziali della vittima ed, addirittura ad implmentare un proxy dello stesso servizio erogato dal web server.

Di seguito si riepilogano i tre scenari in cui viene eseguito il protocollo di autenticazione; per ciascuno di essi si riporta un semplice diagramma di sequenza che evidenzia i protocolli http e https utilizzati nell'interazione tra i partecipanti e l'autenticazione effettuata dalla web application residente nella VM server01.

Si sottolinea che, per semplificare l'implementazione dei sistemi, alla VM server01 sono attribuiti sia il ruolo di web server che di reverse proxy; inoltre, per facilitare lo sviluppo della web application, con riferimento ad un'architettura a livelli, il livello dei dati non prevede l'utilizzo di un database; dunque le credenziali degli utenti che intendono autenticarsi alla web appplication, analogamente alle loro autorizzazioni, sono memorizzate direttamente in chiaro nel codice delle classe java responsabile della verifica della loro correttezza; in ogni caso, nell'esecuzione degli attacchi, l'avversario non sfrutta tale ulteriore debolezza del protocollo.

- Scenario 1: autenticazione di un utente, che si connette in HTTP tramite browser, ad una web application residente in un web server, realizzato mediante Apache-Tomcat, con il protocollo di autenticazione "ingenuo" basato su password;

- Scenario 2: autenticazione di un utente, che si connette in HTTP tramite browser, ad una web application residente in un web server, realizzato mediante Apache configurato come reverse proxy per Tomcat, con il protocollo di autenticazione "ingenuo" basato su password;

- Scenario 3: autenticazione di un utente, che si connette in HTTPS (HTTP over TLS) tramite il browser, ad una web application residente in un web server, realizzato mediante Apache, configurato come reverse proxy per Tomcat, con l'utilizzo del TLS/SSL e di un certificato digitale self-signed, con il protocollo di autenticazione "ingenuo" basato su password.

Gli scenari di riferimento in cui viene eseguito il protocollo di autenticazione utilizzato dalla web application, sviluppati per la realizzazione del caso di studio, già introdotti nella descrizione della parte generale del progetto, sono illustrati con le seguenti figure.

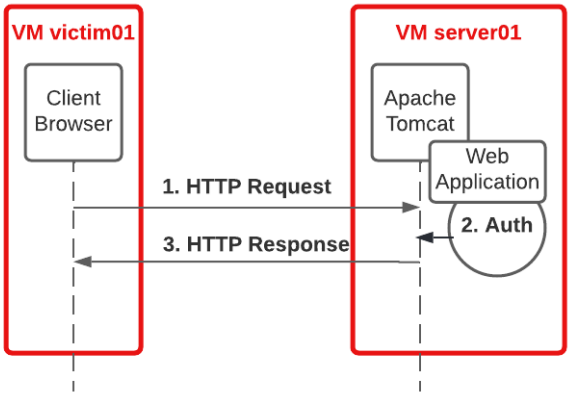


Figura 2: Scenario 1: autenticazione di un utente, che si connette in HTTP tramite browser, ad una web application residente in un web server, realizzato mediante Apache-Tomcat, con il protocollo di autenticazione "ingenuo" basato su password.

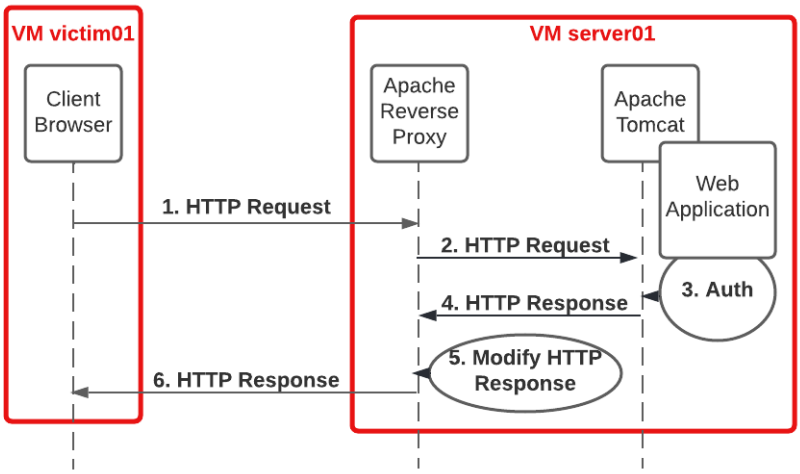


Figura 3: Scenario 2: autenticazione di un utente, che si connette in HTTP tramite browser, ad una web application residente in un web server, realizzato mediante Apache configurato come reverse proxy per Tomcat, con il protocollo di autenticazione "ingenuo" basato su password.

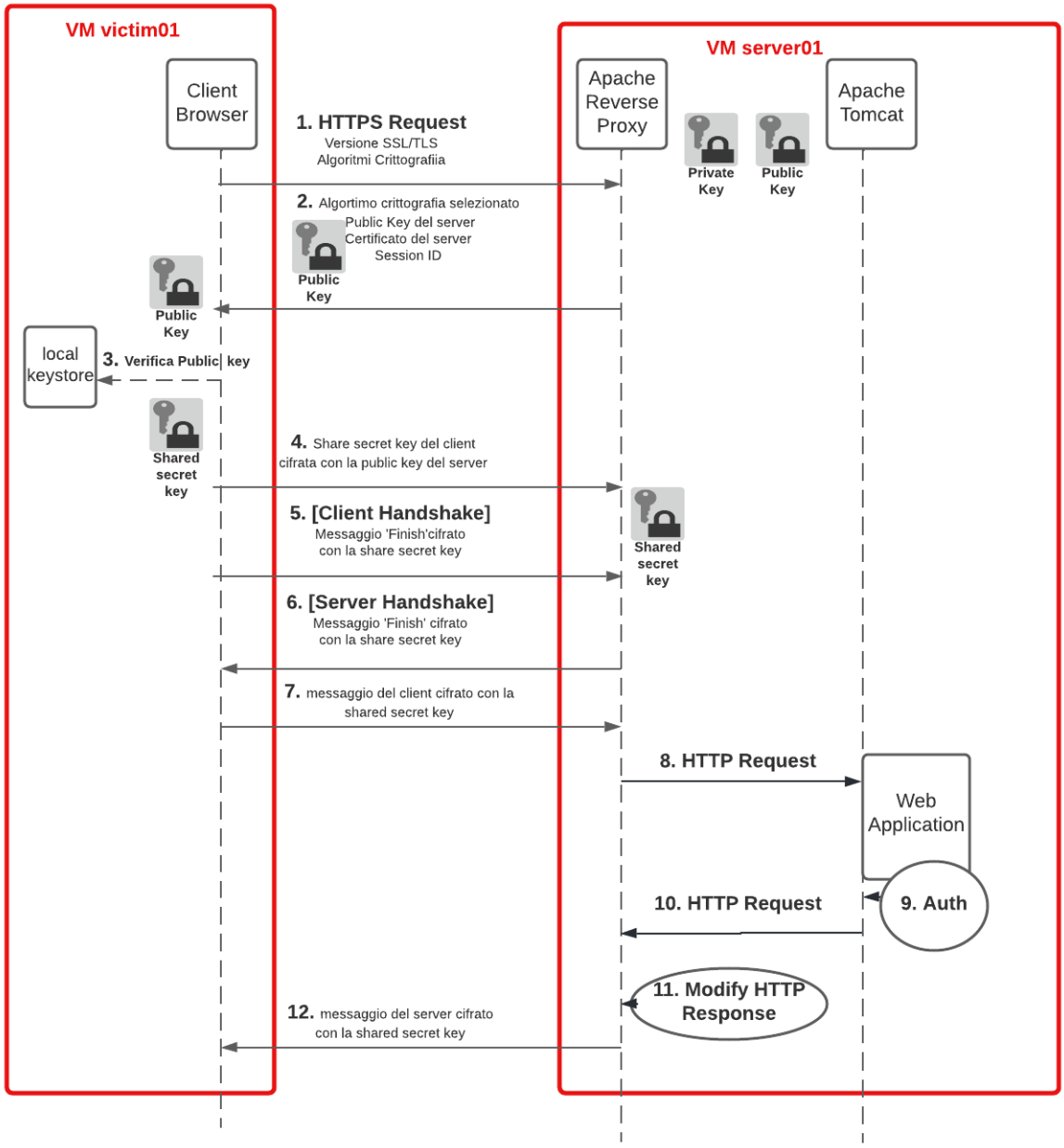


Figura 4: Scenario 3: autenticazione di un utente, che si connette in HTTPS (HTTP over TLS) tramite browser, ad una web application residente in un web server, realizzato mediante Apache, configurato come reverse proxy per Tomcat, con l'utilizzo del TLS/SSL e di un certificato self-signed, con il protocollo di autenticazione "ingenuo" basato su password.

3.3.1 Il protocollo "ingenuo" di autenticazione basato su password utilizzato dalla web application AuthWebApp

I tre scenari indicati, a prescindere dall'utilizzo dei protocolli HTTP e HTTPS e dall'impiego dell'Apache come reverse proxy per Tomcat, hanno in comune l'utilizzo del protocollo "ingenuo" di autenticazione basato su password utilizzato dalla web application denominata *AuthWebApp*.
Il protocollo di autenticazione è analogo al primo trattato nella sezione dei protocolli basati su password; di seguito si fornisce la descrizione e si evidenziano le già note debolezze.

Protocollo 32 (Protocollo di autenticazione basato su password di AuthWebApp) *Sia U un utente che vuole accedere tramite browser da un client, indicato con client01, ad una web application, pubblicata in un server indicato con server01.*
Sia (admin01, Password01) la coppia, costituita rispettivamente dallo username di U e dalla relativa password, memorizzata nel codice della web application pubblicata in server01, per autenticare U.

Il diagramma di sequenza del protocollo "ingenuo", che permette ad U di autenticarsi, per accedere alle pagine protette della web application, in server01, tramite username e password è il seguente:

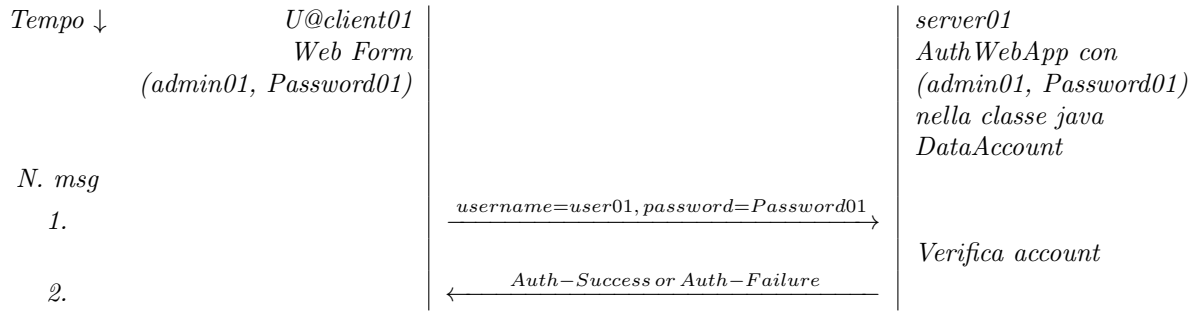


Diagramma 44: Diagramma di sequenza del semplice protocollo di autenticazione basato su password utilizzato dalla web application AuthWebApp.

In dettaglio, l'analisi dei messaggi scambiati nell'esecuzione del protocollo:

- *L'utente U invia, tramite la web form dell'applicazione AuthWebApp, le proprie credenziali costituite dalla coppia username e password con il messaggio* 1*;*
- *La web application confronta le credenziali ricevute da U con quelli memorizzate nel codice della classe dedicata all'autenticazione; in base all'esito della verifica AuthWebApp propone al client01, dal quale si è connesso l'utente, la pagina di benvenuto o rimanda di nuovo alla form di login con il messaggio* 2*.*

In base a quanto definito, si evidenzia sin da subito che *il protocollo garantisce solamente un autenticazione unilaterale; dunque l'utente U non ha alcuna assicurazione*

sull'identità del suo autenticatore; nella sessione dedicata agli attacchi al protocollo si evidenzieranno, a riguardo, le tecniche utilizzate da un attaccante per impossessarsi delle credenziali dell'utente e, addirittura, per assumere il ruolo dell'autenticatore ai danni dell'inconsapevole vittima.

Osservazione 21 (Debolezze del protocollo di autenticazione di AuthWebApp) *Con riferimento all'autenticazione utilizzato da AuthWebApp nei tre scenari indicati, il protocollo soffre, tra gli altri, delle seguenti debolezze:*

- *la coppia (username, password) è memorizzata in chiaro nel codice della web application è vulnerabile; dunque il codice potrebbe essere letto da un avversario al fine di ottenere le informazioni necessarie per impersonare l'utente od altri utenti memorizzati in tale file e, di conseguenza, acquisire diritti di accesso a risorse non autorizzate in pieno anonimato; ne derivano, inoltre, anche problemi di sicurezza nel backup della stessa AuthWebApp residente nel server01;*
- *non essendo previsto l'utilizzo di algoritmi di crittografia, le credenziali di autenticazione sono trasmesse in chiaro mediante il protocollo HTTP, nel canale tra il client01 ed il server01, e dunque è vulnerabile ad attacchi di sniffing;*
- *non utilizzando un'autenticazione forte con meccanismi di challenge-response che prevedono tra gli altri l'utilizzo di nonce e timestamp, è vulnerabile a tutte le tipologie di attacco illustrate a riguardo.*

Va osservato che nel terzo scenario, benché sia previsto l'utilizzo del SSL/TLS a supporto del protocollo HTTP per garantire la riservatezza dei messaggi coinvolti nell'autenticazione, non utilizzando l'autenticatore un dominio validato da una Certification Authority (CA) ed associato ad un certificato firmato da tale (CA), il protocollo non garantisce alcun tipo di autenticazione, né unilaterale, né reciproca tra i partecipanti.

3.3.2 Dettagli dell'implementazione di AuthWebApp

La web application *AuthWebApp*, implementata a scopo didattico per l'esecuzione del protocollo di autenticazione "ingenuo" nei tre scenari illustrati, è stata sviluppata in Java mediante l'utilizzo dell'IDE Eclipse version 2021-09 (4.21.0), con la versione jdk-1.15.0.1, in ambiente Windows 10; nello stesso IDE è stato installata e configurata la versione 8.5.64 di Apache Tomcat.

L'applicazione *AuthWebApp* prevede nella home page una form di login tramite la quale possono autenticarsi differenti tipologie di utenza in base ai ruoli assegnati; a scopo esemplificativo, sono stati configurati due utenti: uno con ruolo di "User" e l'altro con i ruoli di "Admin" e di "User"; gli account sono i seguenti:

- Account user: **(user01, Password01)**, ruolo: "User";
- Account admin: **(admin01, Password01)**, ruoli: "Admin" e "User"

Come indicato, il ruolo dell'utente discrimina la visualizzazione delle pagine protette dell'applicazione; all'utente con il ruolo di "User" è permesso l'accesso al suo pannello utente, *User Panel*, ed alla pagina informativa, *User Info*, relativa al suo account; l'utente con l'account admin, avendo anche il ruolo di "User", ha accesso a tali pagine protette e, grazie al ruolo di "admin", può accedere al pannello amministrativo, *Admin Panel*.

Al fine di agevolare la descrizione dell'organizzazione delle pagine pubbliche e private fruibili tramite la web application, si riporta un diagramma della site map di *AuthWebApp* in cui sono evidenziati i pattern consentiti all'utente a seconda della correttezza dell'autenticazione, delle autorizzazioni (ruoli) e della sessione instaurata come di seguito, in dettaglio, argomentato.

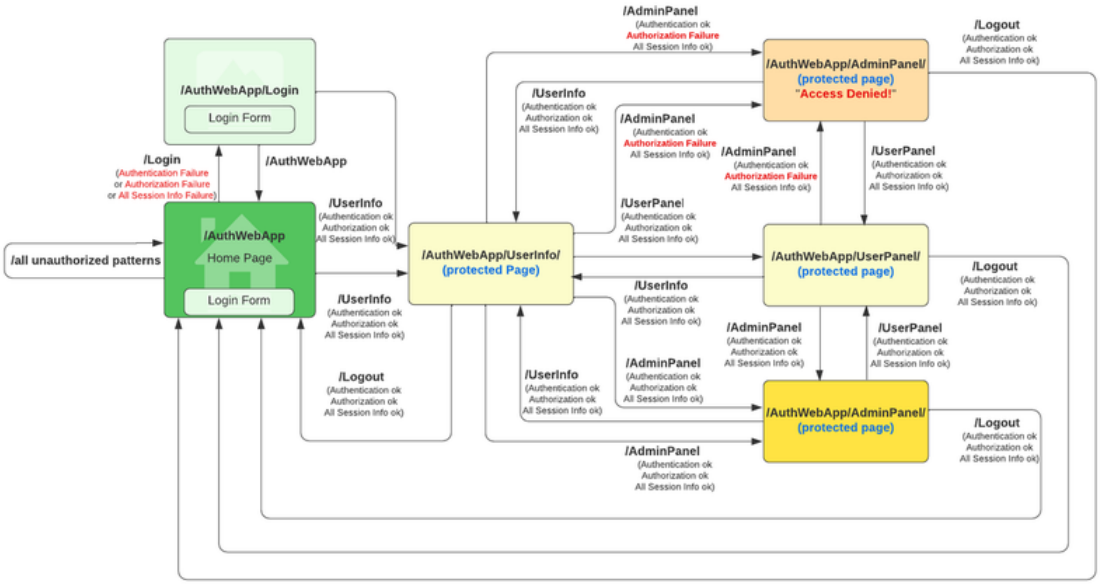


Figura 5: Site map di AuthWebApp con evidenza dei pattern consentiti all'utente in base alla correttezza dell'autenticazione, del ruolo e della sessione instaurata.

Una volta caricata nel browser la pagina iniziale di *AuthWebApp*, nell'ambiente di sviluppo fruibile comodamente mediante il browser integrato in Eclipse ed il server Apache Tomcat configurato all'URL *http://localhost:8080/AuthWebApp/*, l'utente trova direttamente nella home page il form per l'autenticazione.
Per l'accesso alla pagine protette, oltre alla validazione delle credenziali in base al protocollo di autenticazione illustrato, sono necessarie anche le verifiche sulla correttezza del ruolo dell'utente e della sessione instaurata; tale accesso è gestito da un *security filter* realizzato mediante un *servlet filter*; il security filter ha il compito di testare l'accesso dell'utente ad una pagina protetta; nel caso in cui l'utente non sia loggato, la richiesta

viene, infatti, rediretta alla home page con la form di login; se, invece, l'autenticazione non va a buon fine, ad esempio per l'immissione di credenziali errate, l'applicazione rimanda al pattern */Login* che fa riferimento ad una pagina pubblica con una web form per accedere nell'area riservata. In caso di login effettuato con successo, per consentire l'accesso ad una pagina protetta, il security filter verifica l'adeguatezza dei ruoli dell'utente.
In dettaglio, una volta effettuata correttamente l'autenticazione, un utente, sia con il ruolo "User" che con quello di "Admin", viene rediretto nella pagina con il pattern */UserInfo*; da tale pagina, mediante un apposito menù, può accedere a quelle riferibili ai seguenti pattern: */UserPanel*, */AdminPanel* e */Logout*.
La pagina corrispondente al pattern */UserInfo* presenta le informazioni relative allo username ed al ruolo; quella relativa a */UserPanel*, invece, mostra le seguenti informazioni: il *Remote IP*, la *Remote Port*, il *Server Name*, il *Server IP*, la *Server Port*, i *Server Network Adapter ed i corrispondenti Mac Address*; il patter */Logout* consente la disconnessione e rimanda alla home page.
La pagina riferibile al pattern */AdminPanel*, invece, presenta contenuti differenti in base alle autorizzazione dell'utente autenticato; l'utente con il ruolo di "User" che accede al pannello amministrativo, per il quale si richiede il ruolo si "Admin", riceve il messaggio "*Access Denied! Login as a right role account to access admin panel*" a seguito dell'azione del security filter che redirige alla pagina responsabile dell'informativa dell'accesso negato; l'utente con ruolo di "Admin", invece, ha accesso a tale pannello nel quale sono pubblicate, oltre a quelle indicate riguardo alla pagina del */UserPanel*, le seguenti informazioni: *Protocol*, *Method*, *Request URL*, *Request URI*, *Context Path*, *Servlet Path*, *All User Roles*, *URL Pattern request*, *URL Patterns for Role Admin*, *URL Patterns for Role User*, *Cookies*, *Request Header Names*, *Request Session Id*, *Session*, *Logined User Session* e *Principal*; tali informazioni, come si avrà modo di verificare nella sessione relativa all'esecuzione del protocollo con la presenza di un avversario, saranno utili per verificare gli effetti degli attacchi attivi e passivi ai danni dei sistemi partecipanti.

Con riferimento al security filter, responsabile dell'autorizzazione dei pattern riferibili alle pagine protette, ovvero */UserInfo* e */UserPanel* per l'utente con il ruolo di "User" e */UserInfo* e */AdminPanel* per quello con il ruolo di "Admin", fatta eccezione per */Logout* che comporta la distruzione della sessione dell'utente autenticato, va aggiunto che l'utilizzo di pattern differenti dai predetti, a prescindere dall'autenticazione come illustrato nel diagramma del site map, implica la redirezione al home page.

Analizzando in dettaglio il progetto *AuthWebApp*, le classi sono organizzate mediante i seguenti packages. *authwebapp* è il root package; *authwebapp.account* ospita le classi DataAccount e UserRoleAccount; *authwebapp.security* ospita le classi ConfigSecurity, FilterSecurity, RequestSecurity, RequestWrapperSecurity, SessionSecurity e UrlPatternSecurity; *authweb.servlet*, infine, ospita le classi AdminTaskServlet, HomeServlet, LoginServlet, LogoutServlet, UserInfoServlet e UserTaskServlet.
Ciascun package ospita inoltre il package-info, creato di default dall'IDE, utile per la documentazione del codice sorgente realizzato con Javadoc; per la generazione del Javadoc in Eclipse, con riferimento alla JRE source compatibility, si è utilizzata la versione 15.

Per ciascuna delle classi indicate si evidenziano, di seguito, gli aspetti peculiari riportati nella Javadoc.

- *DataAccount*; la classe, utilizzando una HashMap, gestisce gli account degli utenti che possono autenticarsi all'applicazione mediante la coppia username e password e le loro autorizzazioni stabilite nei ruoli di "User" e di "Admin"; il metodo *findUser* della classe consente di verificare la correttezza delle credenziali immesse dall'utente per autenticarsi;

- *UserRoleAccount*; la classe rappresenta l'utente, che si autentica mediante la coppia username e password, e prevede la gestione dei ruoli che lo stesso può assumere;

- *ConfigSecurity*; la classe permette di configurare i ruoli e i pattern a cui è consentito accedere in base al ruolo; si è già osservato che, al ruolo di "User" sono consentiti i pattern */UserInfo* e */UserPanel* mentre al ruolo di "Admin" i pattern */UserInfo* e */AdminPanel*;

- *FilterSecurity*; la classe gestisce le configurazioni di sicurezza del Servlet Filter che ha il compito di controllare le richieste prima di consentire l'accesso alle pagine protette; è anche responsabile dei reindirizzamenti dalle pagine protette alla home page in caso di mancata autenticazione e dalle pagine protette alla pagina di "Access Denied" in caso di autorizzazione non corretta (l'utente con il ruolo "User" che accede alla pagina riferibile al pattern */AdminPanel* permessa solamente al ruolo di "Admin");

- *RequestSecurity*; la classe permette di gestire la sicurezza delle request verificando la necessità dell'autenticazione e la validità del ruolo; il primo controllo è effettuato dal metodo *isSecurityPage* mentre il secondo dal metodo *hasPermission*;

- *RequestWrapperSecurity*; la classe consente un'estensione per la *HTTPServletRequest* che sovrascrive i metodi *getUserPrincipal*, per ottenere l'utente principal associato al messaggio di richiesta HTTP specificato, e *isUserInRole*, per ottenere un valore che indica se l'utente attualmente connesso al sistema appartiene al ruolo specificato; è responsabile del corretto funzionamento del FilterSecurity effettuando il "wrap" di una precedente richiesta con una nuova ed evitando così che il FilterSecurity effettui reindirizzamenti errati in base a precedenti informazioni memorizzate relative all'utente ed al suo ruolo;

- *SessionSecurity*; la classe permette la memorizzazione e la gestione dello informazioni della sessione utente mediante i metodi *storeLoginedUser* e *getLoginedUser*; fa uso delle HashMap per la gestione della raccolta delle coppie chiave-valore, definite come *id_uri_map* e *uri_id_map*; tali HashMap sono utilizzate dai metodi *storeRedirectAfterLoginUrl* e *getRedirectAfterLoginUrl* rispettivamente per la memorizzazione dell'identificativo ricavato dalla requestURI e dalla sessione Http e per l'acquisizione dell'URL dopo l'autenticazione mediante la sessione Http e l'id associato al redirect a seguito dell'autenticazione;

- *UrlPatternSecurity*; la classe, utilizzando l'HashMap, permette il controllo dell'url pattern mediante l'acquisizione del servlet context e dell'url pattern;
- *HomeServlet*; la classe è l'implementazione servlet che consente la gestione della home page dell'applicazione mediante l'utilizzo della java server page *index.jsp* che contiene la descrizione dell'applicazione ed il form di login per l'accesso al pannello utente ed amministrativo;
- *LoginServlet*; la classe è l'implementazione servlet che permette la gestione della pagina di login dell'applicazione mediante l'utilizzo della java server page *login.jsp* che ospita il web form per l'autenticazione tramite utente e password; è responsabile del controllo delle credenziali inviate dall'utente mediante il confronto con quelle registrate nella classe *DataAccount*; in caso di correttezza delle credenziali la classe gestisce il redirect alla pagina protetta con pattern */UserInfo*, altrimenti rinvia alla form della pagina *login.jsp* per una nuova autenticazione;
- *LogoutServlet*; la classe è l'implementazione servlet che consente la gestione della pagina prottetta di logout con pattern */Logout*; distrugge la sessione instaurata dall'utente con un corretto login ed effettua un reindirizzamento alla home page;
- *UserInfoServlet*; la classe è l'implementazione servlet che permette la gestione della pagina informativa protetta dell'applicazione, con pattern */UserInfo*, mediante l'utilizzo della java server page *userinfo.jsp*; tale pagina mostra all'utente, che si è autenticato, le informazioni relative allo username ed al suo ruolo;
- *UserTaskServlet*; la classe è l'implementazione servlet che consente la gestione della pagina protetta dell'applicazione, con pattern */UserPanel*, relativa al pannello utente mediante l'utilizzo della java server page *userpanel.jsp*; tale pagina, come anticipato nella descrizione dell'applicazione, mostra il *Remote IP*, la *Remote Port*, il *Server Name*, il *Server IP*, la *Server Port* ed il *MAC Address associato a ciascun network adapter del server*;
- *AdminTaskServlet*; la classe è l'implementazione servlet che consente la gestione della pagina protetta relativa al pannello amministrativo con l'utilizzo della java server page *adminpanel.jsp*; è già stato sottolineato che tale pagina, accessibile solo dall'utente con il ruolo di "Admin", è riferibile al pattern */AdminPanel*, e, oltre alle informazioni pubblicate nel pannello utente, mostra le seguenti: *Protocol*, *Method*, *Request URL*, *Request URI*, *Context Path*, *Servlet Path*, *All User Roles*, *URL Pattern request*, *URL Patterns for Role Admin*, *URL Patterns for Role User*, *Cookies*, *Request Header Names*, *Request Session Id*, *Session*, *Logined User Session* e *Principal*; tali informazioni saranno utili per l'analisi degli attacchi, successivamente descritti, ai danni del protocollo di autenticazione, effettuati mediante i tool di sniffing e spoofing della distribuzione Kali Linux.

A conclusione dell'illustrazione dei dettagli dell'implementazione della web application *AuthWebApp*, si descrivono sinteticamente i compiti delle java server pages utilizzate nel progetto, già indicati nell'illustrazione delle servlet.

- *index.jsp*; è l'home page dell'applicazione definita nel web.xml del server Apache Tomcat; mostra la descrizione del progetto ed ospita il web form per l'autenticazione necessaria per accedere alle pagine protette; il form, essendo utilizzato anche da login.jsp, è stato sviluppato in un apposito file che viene incluso sia nella home page che nella predetta login.jsp;

- *_loginform.jsp*; ospita il web form indicato che richiama l'url pattern autorizzato /Login per poter accedere alle pagine protette a seguito di un'autenticazione andata a buon fine;

- *login.jsp*; viene caricata se l'utente non effettua una corretta autenticazione dalla home page; a riguardo presenta nuovamente il web form; la stessa pagina consente mediante il link "Home" di tornare alla pagina iniziale;

- *userinfo.jsp*; gestisce la pagina protetta che corrisponde al pattern autorizzato */UserInfo*, alla quale è reindirizzato l'utente a seguito dell'avvenuta autenticazione, indipendentemente dal ruolo autorizzato; come già indicato, la pagina mostra lo username ed il ruolo dell'utente autenticato; analogamente a tutte le altre pagine protette, include la *_loggedusermenu.jsp* per la gestione del menù per la navigazione a disposizione dell'utente;

- *_loggedusermenu.jsp*; come anticipato, è inclusa in tutte le pagine jsp protette in quanto mostra lo username dell'utente autenticato e gestisce il menù di navigazione costituito dei pattern autorizzati */UserInfo*, */UserPanel*, */AdminPanel* e */Logout*;

- *userpanel.jsp*; gestisce la pagina protetta che, come illustrato, visualizza l'IP ed il Mac Address del dispositivo utilizzato dall'utente per la connessione; richiama la *_parampanel.jsp* per la gestione in java della visualizzazione degli indirizzi indicati;

- *_parampanel.jsp*; come anticipato, consente la visualizzazione del Remote IP, del Server Name, del Server IP, della Server Port e del Mac Address associato a ciascun network adapter del server; è inclusa anche nell'*adminpanel.jsp* per ottenere, tra gli altri, i predetti parametri di connessione;

- *accessdenied.jsp*; gestisce la pagina protetta che viene caricata in caso di accesso al pannello amministrativo (*/AdminPanel*) con un ruolo non appropriato (di "User"); la pagina visualizza il messaggio di accesso negato ed invita l'utente ad autenticarsi con un account che ha un ruolo adeguato per accedere a tale pannello;

- *adminpanel.jsp*; gestisce la pagina protetta che viene caricata in caso di accesso al pannello amministrativo (*/AdminPanel*) con il ruolo appropriato (si "Admin"); include il *_parampanel.jsp* per mostrare le informazioni illustrate nella descrizione di tale jsp e gli ulteriori parametri già indicati nella trattazione della classe *AdminTaskServlet*.

3.3.3 Implementazione della virtualizzazione applicata ai sistemi che interagiscono con AuthWebApp nell'esecuzione del protocollo di autenticazione

Considerata la complessità delle numerose configurazioni sistemistiche effettuate nell'implementazione del progetto, successivamente illustrate in dettaglio, si è deciso di optare per *la scelta di un ambiente "tradizionale" basato su un'architettura che utilizza virtual machine* (VM) piuttosto che su microservizi; la scelta di quest'ultima soluzione, la cui implementazione sarebbe stata sicuramente più flessibile, avrebbe però ulteriormente aggravato i già lunghi tempi di realizzazione.

Come indicato nella sezione relativa ai lavori futuri, *la progettazione della transizione dall'architettura monolitica del progetto ad una a microservizi* potrebbe rappresentare un'ulteriore attività per approfondire le tecniche di progettazione Domain-Driven Design (DDD) di microservizi magari utilizzando servizi disponibili in Cloud.

Nell'approccio seguito, il progetto *AuthWebApp* è stato impacchettato in un file WAR per essere distribuito come una singola unità; tale struttura, seppur caratterizzata da una forte rigidità e, quindi, non adatta alla gestione di applicazioni complesse che si evolvono velocemente, risulta idonea allo scopo didattico per il quale è stata concepita; a riguardo, nella sezione successiva relativa all'autovalutazione dei sistemi che interagiscono con il protocollo di autenticazione utilizzato dalla web application, si illustrano in dettaglio l'esecuzione del protocollo nei tre scenari descritti nell'ambiente virtuale implementato.

Gli aspetti negativi dell'architettura monolitica utilizzata riguardano principalmente l'occupazione dello spazio disco delle VM impiegate (ciascuna della 3 VM occupa 20 GB) ed i conseguenti tempi per il deploy in un nuovo ambiente virtuale; a riguardo si è concordato di salvare le VM nello spazio personale offerto da Microsoft OneDrive, fruibile con l'account istituzionale in Microsoft 365, e di renderle fruibili tramite i servizi di condivisione della stessa suite; in ogni caso il file war del progetto, i sorgenti e la documentazione realizzata con Javadoc sono disponibili nel repository all'url `https://gitlab.com/pika-lab/courses/ds/projects/ds-project-latini-ay2021/-/tree/main` .

Per la realizzazione del progetto è stato utilizzata la versione 6.1.28 dell'Oracle VM Virtual BOX in ambiente Windows 10 il cui user manual è consultabile all'url `https://download.virtualbox.org/virtualbox/6.1.28/UserManual.pdf`.
Il download è disponibile all'url: `https://www.virtualbox.org/wiki/Downloads`.
In ogni caso è consigliato installare l'ultima versione disponibile per garantire stabilità e sicurezza all'ambiente e convertire le VM importate al formato compatibile previsto.
Si osserva che, nel host utilizzato, Windows 10, a seguito della presenza del ruolo di Hyper-V, è stato necessario procedere alla disattivazione di tale ruolo in quanto il Microsoft hypervisor attivo risultava in conflitto con l'Oracle VM Virtual BOX, precludendo il corretto avvio delle VM; per effettuare tale operazione dalla shell, con un account amministrativo, si esegue il comando *bcdedit.exe /set hypervisorlauchtype off* ed, infine, si riavvia il sistema.

Come anticipato, nell'Oracle VM Virtual BOX sono state create le 3 VM indicate con OS VM guest Kali GNU/Linux versione 2021.3 (Debian 64-bit) delle quali si evidenziano le seguenti caratteristiche.

Ogni VM ha 2 CPU, 2 GB di RAM, 20 GB di spazio disco e 2 schede di rete così configurate per ragioni di sicurezza, al fine di limitare le interazioni con il server Host:

- scheda 1 (eth0): è connessa con NAT; alla scheda è stata negata la modalità promiscua; ottiene un IP dinamico nella subnet 10.0.2.15/24 dall'Oracle VM VirtualBox DHCP server; è stato abilitato il port-forwarding tra l'host ed la guest VM per permettere il traffico di rete con i seguenti protocolli utili per l'amministrazione e l'aggiornamento: ssh (TCP 22), http (TCP 80), https (TCP 443);

- scheda 2 (eth1): ha un IP statico assegnato nella subnet 192.168.100.0/24; alla scheda ed è stata permessa la modalità promiscua per un utilizzo efficace dei tool offensivi della distro Kali nei confronti della vittima che eseguirà il protocollo di autenticazione previsto dall'applicazione *AuthWebApp*; *la subnet assegnata a tale scheda (eth1) consente l'esecuzione di attacchi ai danni delle VM partecipanti al protocollo di autenticazione senza interagire con i dispositivi i cui indirizzi IP appartengono alle altre subnet utilizzate dal sistema Host.*

Nella tabella seguente, per ciascuna VM, si riepilogano gli IP assegnati alle due schede di rete.

VM	**IP Scheda 1 (eth0)**	**IP Scheda 2 (eth1)**
server01	Dinamic	192.168.100.1
victim01	Dinamic	192.168.100.2
hack01	Dinamic	192.168.100.3

Non essendo stato utilizzato un DNS nel virtual enviroment, la risoluzione dei nomi in ciascuna VM è gestita localmente con il file */etc/hosts* che contiene l'associazione nome - IP della eth1; a riguardo, si è modificato anche il file */etc/hostname* sostituendo il nome predefinito, Kali, con quello indicato nella tabella precedente. Ogni VM ha, inoltre, una cartella condivisa con il sistema host, accessibile in sola lettura con i privilegi di root; nel caso del sistema Windows utilizzato, tale directory è montata in automatico all'avvio in */myshare* e permette l'accesso al percorso c:\temp del sistema Windows.
A prescindere dalle configurazioni descritte, terminata l'importazione delle VM così implementate nell'Oracle VM Virtual BOX, è fortemente raccomandato assicurarsi che le configurazioni indicate siano mantenute e che le VM non abbiano alcuna interazione con i sistemi connessi all'Host per evitare potenziali disservizi e problemi di sicurezza, considerata anche l'esecuzione dei tool offensivi della distribuzione Kali.

A completamento della descrizione delle caratteristiche della VM, per ciascun sistema si riporta anche le principali configurazioni sistemistiche; nell'esecuzione del protocollo utilizzato da *AuthWebApp* alcune di queste saranno oggetto di modifica in accordo con gli scenari indicati.
Nella VM server01 sono stati installati i seguenti software:

- Java version 15 (build 15+36-1562): la stessa versione di Java utilizzata nel host Windows in cui è stato sviluppato il progetto con Eclipse; tale versione assicura il corretto deploy in Apache Tomcat del file WAR del progetto esportato dall'IDE. Il percorso di JAVA_HOME è stato impostato nella dir */opt/jdk-15*.

- Apache Tomcat 8.5.71. La configurazione si trova nel percorso:

  /opt/tomcat/apache-tomcat-8.5.71.

  In particolare sono state impostate le seguenti credenziali dell'utente con il ruolo di manager-gui: (admin01, password01) nel file

  /opt/tomcat/apache-tomcat-8.5.71/conf/tomcat-users.xml.

  Con tali credenziali si è effettuato il login da *http://127.0.0.1:8080/manager/html*, Manager App di Apache Tomcat, per gestire il deploy del file, *AuthWebApp.war*, esportato da Eclipse, in precedenza copiato dalla dir */myshare*, accessibile dal sistema host, a */opt/tomcat/apache-tomcat-8.5.71/webapps*. Inoltre, per consentire in Kali l'avvio in automatico di Apache Tomcat con la Java Version 15, è stata creata una configurazione personalizzata del file */lib/systemd/system/tomcat.service*. L'avvio e l'arresto del servizio è stato facilitato mediante la creazione degli script */usr/bin/tomcatup* e */usr/bin/tomcatdown* che fanno rispettivamente riferimento a */opt/tomcat/apache-tomcat-8.5.71/bin/startup.sh* e *shutdown.sh* nello stesso percorso.

- OpenSSL 1.1.1l. La configurazione si trova nel percorso di default: */etc/ssl*.

  Il package consente la creazione dei certificati digitali utilizzati da Apache e l'utilizzo del SSL a supporto del protocollo http; il certificato pubblico utilizzato dal web server si trova in */etc/ssl/certs*, la chiave privata in */etc/ssl/private/server.crt*.

- Apache/2.4.51. La configurazione si trova nel percorso: */etc/apache2*.

  I moduli si trovano in */usr/lib/apache2/modules/* e la configurazione di quelli abilitati in */etc/apache2/mods-enabled/*; nello specifico, la configurazione relativa al ruolo di reverse proxy si trova in */etc/apache2/mods-enabled/proxy.load*.

  Il virtual host relativo alla web application *AuthWebApp* è gestito con il file

  /etc/apache2/sites-available/AuthwebApp.conf; in tale file è specificata l'abilitazione del SSL, il percorso delle chiavi utilizzate, dei log e degli URL da utilizzare per effettuare il *proxy pass* ed il *proxy pass reverse* del pattern */AuthWebApp*.

Nella VM hack01, utilizzata dall'avversario, sono stati installati i seguenti software:

- OpenSSL 1.1.1l. La configurazione si trova nel percorso di default: */etc/ssl*.

  Il package consente la creazione dei certificati digitali "fake" utilizzati dall'avversario nell'attacco MITM ed il conseguente utilizzo del SSL a supporto del protocollo http;

- SSLsplit 0.5.5 (built 2021-10-26). L'eseguibile si trova nel percorso: */usr/bin*.

  Il certificato utilizzato per l'attacco e la sua chiave privata, per ragioni di sicurezza, sono posizionati nella dir */root*.

Nella VM victim01, a differenza delle altre, non sono stati installati pacchetti aggiuntivi poiché per l'esecuzione del protocollo è necessario solamente l'utilizzo del browser Firefox già in dotazione dell'ambiente grafico della distribuzione Kali.

3.3.4 Autovalutazione dei sistemi che interagiscono con il protocollo di autenticazione utilizzato da AuthWebApp nell'ambiente virtuale realizzato

Con la presente sezione si intende valutare, nell'ambiente virtuale realizzato, i sistemi che interagiscono con *AuthWebApp* nell'esecuzione del protocollo di autenticazione basato su password; *come criterio di valutazione si è scelto la verifica della sicurezza dell'esecuzione del protocollo* e, pertanto, nella valutazione oltre al protocollo di autenticazione utilizzato dalla web application è necessario prendere in considerazione i protocolli di comunicazione (HTTP e HTTP over TLS) utilizzati dai partecipanti oltreché le loro configurazioni sistemistiche.

Gli scenari di riferimento in cui avviene la valutazione sono i tre indicati nella definizione del caso di studio; in ciascun scenario, il metodo seguito consiste nell'effettuare uno o più attacchi ai danni dei sistemi coinvolti nell'esecuzione del protocollo di autenticazione al fine di individuare le credenziali utilizzate da un utente, che accede alle pagine protette tramite il web form di *AuthWebApp* per autenticarsi al posto della vittima inconsapevole. Le tecniche di attacco individuate dipendono dai protocolli di comunicazione utilizzati e dall'eventuale impiego di algoritmi crittografici.

Chiaramente gli scenari proposti sono stati concepiti per evidenziare la debolezza del protocollo di autenticazione utilizzato dalla web application e mostrare che, nonostante le ottimizzazioni nelle configurazioni dei web server e l'utilizzo del protocollo SSL, le criticità permangono; dunque nell'ottica di un'adeguata valutazione della sicurezza dei sistemi che partecipano all'esecuzione del protocollo, si deve tener conto del fatto che l'applicazione *AuthWebApp* è stata sviluppata a livello didattico con le note limitazioni descritte (tra le quali le credenziali di autenticazione, utilizzate per la verifica, memorizzate in chiaro direttamente nel codice), dell'ingenuità dello stesso protocollo di autenticazione e del fatto che le VM interessate interagiscono in un ambiente virtuale ben differente da un contesto di produzione; in ogni caso, pur con tutte le limitazioni indicate, *gli scenari proposti sono significativi per effettuare una valutazione della sicurezza mirata ad evidenziare le possibili criticità a cui possono essere soggetti i sistemi in caso di configurazioni non adeguate e di utilizzo di applicazioni che richiedono protocolli di autenticazione deboli.*

Grazie alle configurazioni illustrate nella sezione relativa all'implementazione della virtualizzazione dei sistemi, i differenti scenari, in cui si effettuano i test, si ottengono rapidamente con l'esecuzione di alcune modifiche da apportare alla configurazione dei web server utilizzati; inoltre, fatta eccezione per la personalizzazione effettuata per il tool SSLsplit, i software di sniffing e spoofing utilizzati per gli attacchi sono disponibili nella distribuzione Kali e, pertanto, non sono richieste installazioni e configurazioni aggiuntive.

Le istruzioni riportate in dettaglio tengono conto del fatto che i test siano eseguiti nell'ordine stabilito per agevolare le modifiche delle configurazione dei web server Apache-Tomcat ed Apache; in ogni caso gli attacchi possono essere riprodotti in un ordine qualunque o anche in un singolo scenario; ovviamente la migliore configurazione dei due web server, che più si adatta ad una situazione reale, è quella indicata nel terzo scenario, seppur si faccia uso di certificati self-signed decisamente sconsigliati in un ambiente di produzione.

In accordo con quanto premesso, per ciascun scenario indicato, si riportano i test, ovvero gli attacchi effettuati, comprensivi anche delle successive fasi del *vulnerability assesment* realizzato in accordo con le "*linee guida per la modellazione delle minacce ed individuazione della azioni di mitigazione conformi ai principi del secure/privacy by design" dell'Agenzia per l'Italia Digitale (AgID)*" [8].

Scenario 1: autenticazione di un utente, che si connette tramite browser in HTTP, alla web application, AuthWebApp, residente nel web server, server01, realizzato mediante Apache-Tomcat, in ascolto nella porta TCP 8080, con il protocollo di autenticazione "ingenuo" basato su password.

Test effettuato: attacco ai sistemi con Wireshark ed Ettercap.

Nell'Oracle VM VirtualBox gestore, accese le VM server01, victim01 e hack01, , si effettua l'autenticazione in ciascuna di esse nell'ambiente grafico predefinito XFCE con utente kali (password kali) e si eseguono le operazioni indicate.
Nella VM server01

- si apre un terminale di root (o si esegue *sudo su* dall'utente kali con password kali);
- si edita il file di configurazione di Apache Tomcat */opt/tomcat/apache-tomcat-8.5.71/conf/server.xml*,
- si modifica il settaggio *address="127.0.0.1"*, sostituendo l'IP di localhost con l'IP *192.168.100.1* e si salva il file; in tal modo il web server risponderà all'IP assegnato alla VM server01 e potrà essere contattato dalle VM victim01 e hack01 i cui IP della scheda eth1 appartengono alla stessa subnet;
- si riavvia il servizio, per rendere operativa la nuova configurazione, con il comando *systemctl restart tomcat* ;
- si verifica, quindi, che Tomcat sia in ascolto nella porta TCP 8080 con il nuovo indirizzo tramite il comando *telnet server01 8080* .

Nella VM hack01

- dalla voce di menù "*09-Sniffing & Spoofing*", si esegue *wireshark*;
- dal menu di wireshark si seleziona *Caption*, si esegue *Refresh interface* e si seleziona l'interfaccia *eth1* (che ha come IP 192.168.100.3);
- si avvia lo sniffing con l'apposito bottone (o da Capture selezionando *Start*), verificando che nella barra inferiore di wireshark sia attivo "*eth1:<il live capture in progress>*";

[8] `https://www.agid.gov.it/sites/default/files/repository_files/allegato_4_-_linee_guida_per_la_modellazione_delle_minacce-dlt.pdf`

- dalla stessa voce di menù "*09-Sniffing & Spoofing*", si esegue *ettercap-graphical*;
- si esegue l'autenticazione (con password kali) per assicurare gli opportuni privilegi amministrativi all'applicazione;
- nel setup di ettercap si seleziona *eth1* come Primary Interface, si abilita il "*Bridged Sniffing*" nell'interfaccia *eth1*, si deseleziona "*Sniffing at startup*" e si conferma con il bottone "*Accept*";
- si avvia la cattura dei pacchetti dal bottone "*Start/Stop Sniffing*" verificando che compaia il messaggio "Starting Bridged sniffing.." nella finestra dell'output;
- si imposta la modalità di visualizzazione delle connessioni da "*Connections*" dal menù "*View*".

Nella VM victim01

- si apre il browser Mozilla e si digita l'url *http://server01:8080/AuthWebApp/* per utilizzare la web application (l'indirizzo è già memorizzato, per comodità, nella cartella AuthWebApp dei bookmarks);
- si effettua l'autenticazione tramite la web form dell'applicazione utilizzando uno dei 2 utenti configurati: ad esempio con admin01 (Password01);
- si accede, quindi, allo User Panel per la verifica dei parametri della connessione;

Nella VM hack01

- da *Wireshark* si arresta la cattura dei pacchetti con il bottone "*Stop capturing filter*";
- nella barra dei filtri si imposta il seguente:

  ip.src==192.168.100.2 and ip.dst==192.168.100.1 and

  http.request.method=="POST"

  per catturare il pacchetto relativo all'autenticazione dell'utente dalla VM victim01 alla VM server01.

Il filtro applicato ai pacchetti catturati da Wireshark selezionerà il pacchetto che ha come *Info* "*POST /AuthWebApp/Login HTTP/1.1*" e mosterà nel dettaglio nella sezione "*HTML Form URL Encoded: application/x-www-form-urlencoded*" le credenziali di autenticazione nel seguente modo:
Form item: "username" = "admin01"
Form item: "password" = "Password01".
Si osservi, a riguardo, lo snapshot seguente relativo a tale cattura.

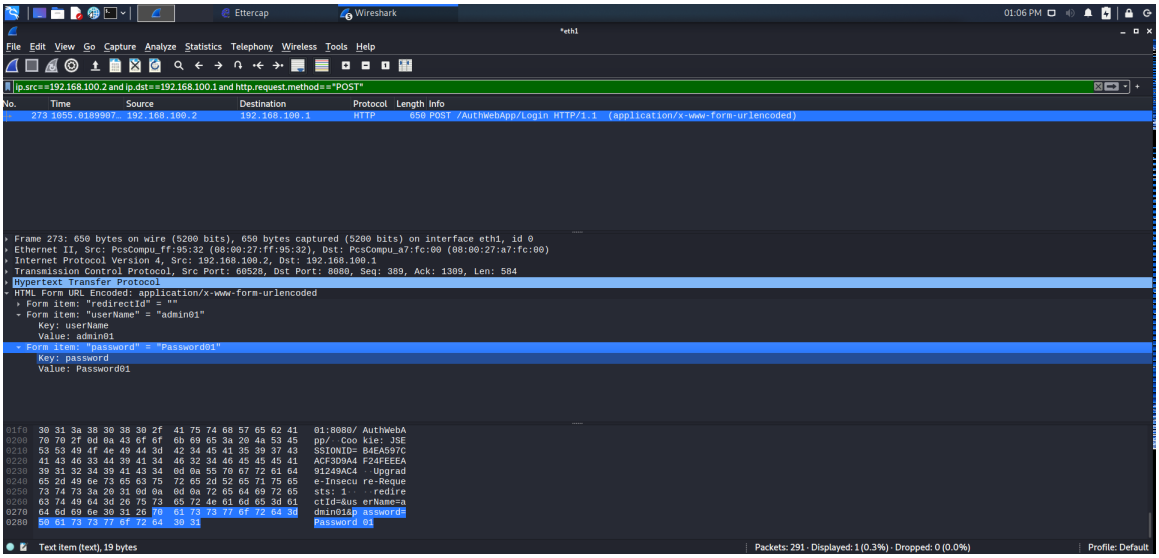


Figura 6: Credenziali di autenticazione di AuthWebApp, che utilizza l'http con Apache-Tomcat in ascolto nella porta TCP 8080, catturate da Wireshark.

Anche da *Ettercap* si arresti la cattura dei pacchetti con il bottone "*Start/Stop Sniffing*" per visualizzare le credenziali di autenticazione espresse nel seguente modo:

HTTP:192.168.100.1:8080− >USER:admin01 PASS:Password01

INFO:http://server01:8080/AuthWebApp

CONTENT: redirectid=&userName=admin01&password=Password01

Di seguito si riporta lo snapshot relativo ad Ettercap.

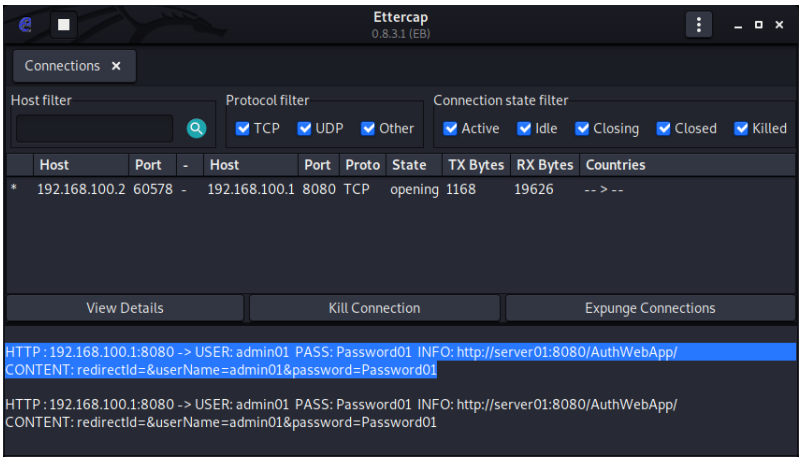


Figura 7: Credenziali di autenticazione di AuthWebApp, che utilizza l'http con Apache-Tomcat in ascolto nella porta TCP 8080, catturate da Ettercap.

Sulla base dei risultati ottenuti dall'attacco effettuato, si riportano le fasi del *vulnerability assesment* relative allo scenario 1.
Test: attacco ai sistemi con Wireshark ed Ettercap.
Analisi: le origini delle vulnerabilità sono da imputare alla

- debolezza del protocollo di autenticazione utilizzato da AuthWebApp;
- debolezza del protocollo di comunicazione (HTTP) utilizzato dal web server Apache Tomcat.

Entrambi i protocolli indicati, infatti, non utilizzano la crittografia.
Assessment con ranking basato su una classificazione con tre livelli di gravità crescente ("lieve","media","grave") di non conformità alle indicazioni dell'AgID":

- protocollo di autenticazione utilizzato da AuthWebApp: vulnerabilità grave;
- configurazione dal web server Apache Tomcat: vulnerabilità grave.

Remediation:

- sostituzione del protocollo di autenticazione utilizzato da AuthWebApp con un protocollo di autenticazione forte;
- modifica della configurazione dal web server Apache Tomcat con utilizzo del SSL.

Scenario 2: autenticazione di un utente, che si connette tramite web browser in HTTP, alla web applicazion, AuthwebApp, residente nel web server, server01, che utilizza Apache, configurato come Reverse Proxy per Apache-Tomcat che risponde in localhost, in ascolto nella porta TCP 80, con il protocollo di autenticazione "ingenuo" basato si password.

Test effettuato: attacco ai sistemi con Wireshark ed Ettercap.

Nella VM server01

- si utilizza il terminale di root aperto in precedenza per editare il file di configurazione di Apache Tomcat */opt/tomcat/apache-tomcat-8.5.71/conf/server.xml*;
- si ripristina il settaggio *address="127.0.0.1"* sostituendo l'IP *192.168.100.1*, in precedenza modificato, con quello di *localhost* e si salva il file; in tal modo il web server sarà accessibile solamente dalla stessa VM server01 in ascolto nella porta TCP 8080 e non potrà più essere contattato dalle VM victim01 e hack01;
- si riavvia il servizio, per rendere operativa la nuova configurazione, con il comando: *systemctl restart tomcat*;

- si verifica, quindi, che Tomcat sia in ascolto nella porta TCP 8080 con il nuovo indirizzo tramite il comando: *telnet 127.0.0.1 8080*;
- si edita il file di configurazione del virtual host di Apache relativo a AuthWebApp */etc/apache2/sites-available/AuthwebApp.conf* e si commentano le seguenti righe:

  SSLEngine on

  SSLCertificateFile /etc/ssl/certs/server.crt

  SSLCertificateKeyFile /etc/ssl/private/server.crt

  per disabilitare il modulo SSL necessario per l'https (impostato per lo scenario 3);
- si controlla la correttezza della sintassi, per rendere operativa la nuova configurazione, con il comando: *apachectl configtest*;
- si riavvia il servizio, per rendere operativa la nuova configurazione, con il comando: *systemctl restart apache2*;
- si verifica, quindi, che Apache sia in ascolto sulla porta TCP 80 tramite il comando: *telnet server01 80*.

Nella VM hack01

- dal menu di *Wireshark* si avvia lo sniffing nell'interfaccia eth1, attivata nel precedente attacco, con l'apposito bottone (o da Capture selezionando *Start*);
- alla richiesta di salvataggio dei pacchetti, catturati nell'esempio precedente, si risponde con la scelta di *"Continue without Saving"*;
- si verifica che nella barra inferiore di wireshark sia attivo "*eth1:<il live capture in progress>*";
- dall'interfaccia di *Ettercap* si avvia la cattura dei pacchetti, con il "*Bridged Sniffing*" nell'interfaccia *eth1* abilitato dall'esempio precedente, mediante il bottone "*Start/Stop Sniffing*", verificando che nella finestra dell'output compaia il messaggio "Starting Bridged sniffing..";
- si imposta la modalità di visualizzazione delle connessioni da "*Connections*" dal menù "*View*".

Nella VM victim01

- si apre il web browser e si digita l'url *http://server01/AuthWebApp/* per utilizzare l'applicazione (l'indirizzo è già memorizzato, per comodità, nella cartella AuthWebApp dei bookmarks);
- si effettua l'autenticazione nell'applicazione utilizzando uno dei 2 utenti configurati: ad esempio con admin01 (Password01);

- si accede allo User Panel per la verifica dei parametri della connessione.

Nella VM hack01

- dal menu di *Wireshark* si avvia lo sniffing nell'interfaccia eth1, attivata nel precedente attacco, con l'apposito bottone (o da Capture selezionando *Start*);
- alla richiesta di salvataggio dei pacchetti, catturati nell'esempio precedente, si risponde con la scelta di *"Continue without Saving"*;
- si verifica che, nella barra inferiore di wireshark, sia attivo "*eth1:<il live capture in progress>*";
- da *Wireshark* si arresta la cattura dei pacchetti con il bottone *"Stop capturing filter"*;
- nella barra dei filtri si imposta lo stesso filtro utilizzato nell'esempio precedente:

  ip.src==192.168.100.2 and ip.dst==192.168.100.1 and

  http.request.method=="POST"

  per catturare il pacchetto relativo all'autenticazione dell'utente dalla VM victim01 alla VM server01.

Si osserva, a riguardo, lo snapshot relativo a tale cattura.

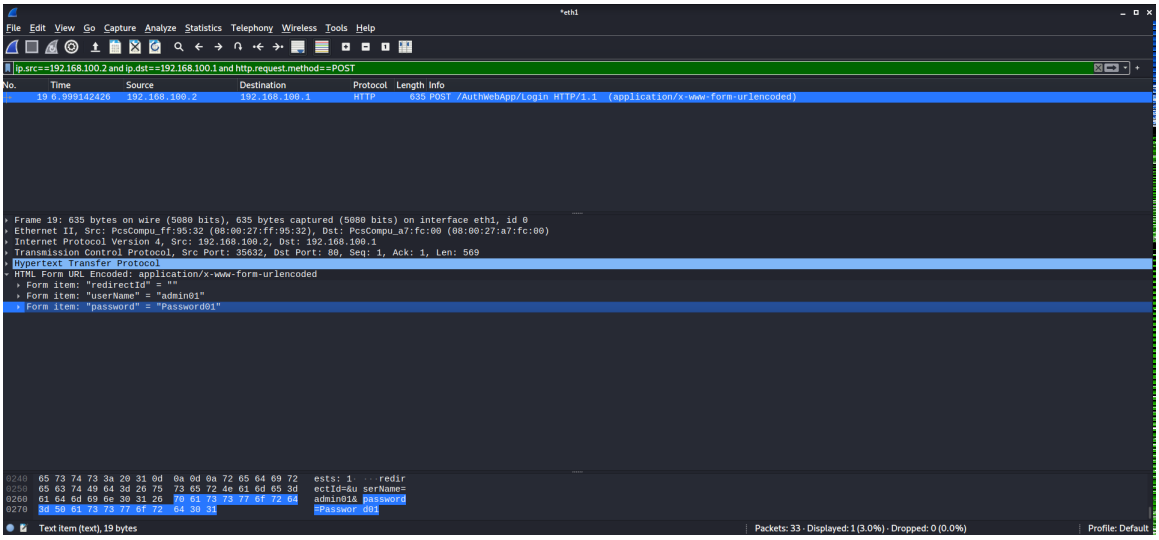


Figura 8: Credenziali di autenticazione di AuthWebApp, che utilizza l'http con Apache, configurato come reverse proxy per Tomcat, in ascolto nella porta TCP 80, catturate da Wireshark.

Nello snapshot si evidenzia, dal risultato del filtro, la selezione del pacchetto che ha come Info "*POST /AuthWebApp/Login HTTP/1.1*"; l'analisi della sezione relativa al

Hypertext Trasfer Protocol di tale pacchetto mostra, analogamente a quanto avvenuto nello scenario predente, le credenziali di autenticazione, inviate in chiaro al web server Apache, configurato come Reverse Proxy per Tomcat, che risponde nel server01 con l'indirizzo di localhost, nel seguente modo:
Form item: "username" = "admin01"
Form item: "password" = "Password01".

Anche da *Ettercap* si arresta la cattura dei pacchetti con il bottone "*Start/Stop Sniffing*" per visualizzare le credenziali di autenticazione espresse nel seguente modo:
HTTP:192.168.100.1:8080− >USER:admin01 PASS:Password01
INFO:http://server01:8080/AuthWebApp
CONTENT: redirectid=&userName=admin01&password=Password01
Di seguito si riporta lo snapshot relativo ad Ettercap.

Figura 9: Credenziali di autenticazione di AuthWebApp, che utilizza l'http con Apache, configurato come reverse proxy per Tomcat, in ascolto nella porta TCP 80, catturate da Ettercap.

Sulla base dei risultati ottenuti dall'attacco effettuato, si riportano le fasi del *vulnerability assesment* relative allo scenario 2.
Test: attacco ai sistemi con Wireshark ed Ettercap.
Analisi: le origini delle vulnerabilità sono da imputare alla

- debolezza del protocollo di autenticazione utilizzato da AuthWebApp;
- debolezza del protocollo di comunicazione (HTTP) utilizzato dal web server Apache configurato come reverse proxy per Tomcat.

Entrambi i protocolli indicati, analogamente al primo scenario, infatti non utilizzano la crittografia.

Assessment con ranking basato su una classificazione con tre livelli di gravità crescente ("lieve","media","grave") di non conformità alle indicazioni dell'AgID":

- protocollo di autenticazione utilizzato da AuthWebApp: vulnerabilità grave,
- configurazione dal web server Apache: vulnerabilità grave,

Remediation:

- sostituzione del protocollo di autenticazione utilizzato da AuthWebApp con un protocollo di autenticazione forte;
- modifica della configurazione dal web server Apache, configurato come reverse proxy per Tomcat con utilizzo del SSL.

Scenario 3: autenticazione di un utente, che si connette tramite web browser in HTTPS alla web application, AuthWebApp, residente nel web server, server01, che utilizza Apache, configurato come reverse proxy per Apache-Tomcat che risponde in localhost, in ascolto nella porta TCP 443 con l'utilizzo di un certificato self-signed, con il protocollo di autenticazione "ingenuo" basato su password.
Test effettuato: attacchi ai sistemi con Ettercap, mediante Man In The Midle (MITM) con arp-poisoning, e SSLsplit.

Nella VM server01

- si utilizza il terminale di root, aperto in precedenza, per editare nuovamente il file di configurazione del virtual host di Apache relativo a AuthWebApp

  /etc/apache2/sites-available/AuthwebApp.conf

  e si eliminano i commenti in precedenza all'inizio delle seguenti direttive:

  SSLEngine on

  SSLCertificateFile /etc/ssl/certs/server.crt

  SSLCertificateKeyFile /etc/ssl/private/server.crt

  per riabilitare il modulo SSL necessario al servizio https;

- si controlla la correttezza della sintassi, per rendere operativa la nuova configurazione, con il comando: *apachectl configtest*;
- si effettua il riavvio del servizio, per rendere operativa la nuova configurazione, con il comando: *systemctl restart apache2*;
- si verifica, quindi, che Apache sia in ascolto nella porta TCP 443 tramite il comando: *telnet server01 443.*

Nella VM hack01

- dal menu di *Wireshark* si avvia lo sniffing nell'interfaccia eth1, attivata nel precedente attacco, con l'apposito bottone (o da Capture selezionando *Start*);
- alla richiesta di salvataggio dei pacchetti, catturati nell'esempio precedente, si risponde con la scelta di *"Continue without Saving"*;
- si verifica che nella barra inferiore di Wireshark sia attivo "*eth1:<il live capture in progress>*";
- *si chiude Ettercap* lanciato nel test relativo allo scenario precedente;
- si apre un terminale come root ed si esegue il seguente comando

  iptables –list -t nat

  per verificare che le regole di nat (policy ACCEPT) del firewall iptables relative alle catene di PREROUTING, di INPUT, di OUTPUT e di POSTROUTING della VM siano vuote;
- *si esegue di nuovo Ettercap* in modalità grafica e si effettua l'autenticazione per permettere l'esecuzione di operazioni che richiedono i privilegi amministrativi.

  Poiché nello scenario considerato la web app, AuthWebApp, utilizza il protocollo https, http over SSL, per la comunicazione sicura, al fine di effettuare un'intercettazione delle credenziali inviate nel protocollo di autenticazione, nonostante l'utilizzo della crittografia, si rendono necessarie le seguenti configurazioni che permettono un attacco attivo di tipo MITM utilizzando l'arp poisoning.
- *Si disabilita in ettercap* l'opzione di "*Sniffing at Startup*", si seleziona *eth1* come "Primary Interface" e si conferma con il botton "Accept";
- dal terminale di root aperto in precedenza si esegue di nuovo il seguente comando

  iptables –list -t nat

  per verificare che siano state aggiunte le seguenti regole di nat (policy ACCEPT) del firewall iptables relative alla catena di PREROUTING;

  Chain PREROUTING (policy ACCEPT)

  target prot opt source destination

  REDIRECT tcp – anywhere anywhere tcp dpt:telnets redir ports 59263

  REDIRECT tcp – anywhere anywhere tcp dpt:submissions redir ports 59264

  REDIRECT tcp – anywhere anywhere tcp dpt:pop3s redir ports 59265

  REDIRECT tcp – anywhere anywhere tcp dpt:nntps redir ports 59266

  REDIRECT tcp – anywhere anywhere tcp dpt:ldaps redir ports 59267

  REDIRECT tcp – anywhere anywhere tcp dpt:994 redir ports 59268

REDIRECT tcp – anywhere anywhere tcp dpt:imaps redir ports 59269

REDIRECT tcp – anywhere anywhere tcp dpt:http-alt redir ports 59270

REDIRECT tcp – anywhere anywhere tcp dpt:https redir ports 59271

REDIRECT tcp – anywhere anywhere tcp dpt:ftps redir ports 59272

Come si evince dalla sintassi, tali regole permettono la redirezione del traffico tcp proveniente da qualunque host verso la stessa VM hack01 su una particolare porta per ciascun protocollo specificato; *la regola che sarà utilizzata nell'attacco è la penultima in quanto è responsabile della redirezione del traffico https verso la porta TCP 59271.*

- da Ettercap si avvia la scansione degli host mediante il bottone "*Scan for hosts*" e si verifica che compaia l'output "2 hosts added to the host list";
- dal menù *Host* si seleziona *Hosts list* per verificare la rilevazione degli IP 192.168.100.1 e 192.168.100.2 associati rispettivamente alle VM server01 e victim01; si noti in wireshark i pacchetti generati da Ettercap per effettuare le ARP query, del tipo "Who has 192.168.100.* ? Tell 192.168.100.3", necessarie per la rilevazione degli host nella subnet 192.168.100.1/24 associata all'interfaccia eth1;
- una volta terminata la rilevazione degli host, si seleziona prima l'IP 192.168.100.1, e si clicca sul bottone *Add to Target1*, poi l'IP 192.168.100.2 e si clicca sul bottone *Add to Target2* per aggiungere gli IP, rispettivamente corrispondenti a server01 e victim01, ai target di Ettercap;
- dalle voce di menu *Target* si seleziona *Current Target* per avere la visualizzazione degli IP assegnati come target;
- si utilizza il bottone *MITM menù*, selezionando come tipo di attacco l'*ARP poisoning* e si conferma il parametro opzionale selezionato "*Sniff remote connections*", verificando che nell'output compaiano i seguenti dettagli dell'attacco:

  "ARP poisoning victims:

  GROUP1:192.168.100.1 08:00:27:A7:FC:00

  GROUP2:192.168.100.2 08:00:27:FF:95:32";
- si avvia la cattura dei pacchetti dal bottone *Start/Stop Sniffing* verificando che nell'output compaia il messaggio "Starting Unified sniffing...".
- si verifica in Wireshark le query ARP inviate a tutti gli IP della subnet da parte della VM hack01 per la scansione degli host, come evidenziato nello snapshot successivo;
- si verifica, inoltre, in Wireshark l'attacco di tipo ARP poisoning eseguito dalla VM hack01 ai danni di 192.168.100.1 (server01) e 192.168.100.2 (victim01) andato a buon fine; ad entrambe le VM è stato, infatti, fornito il mac address

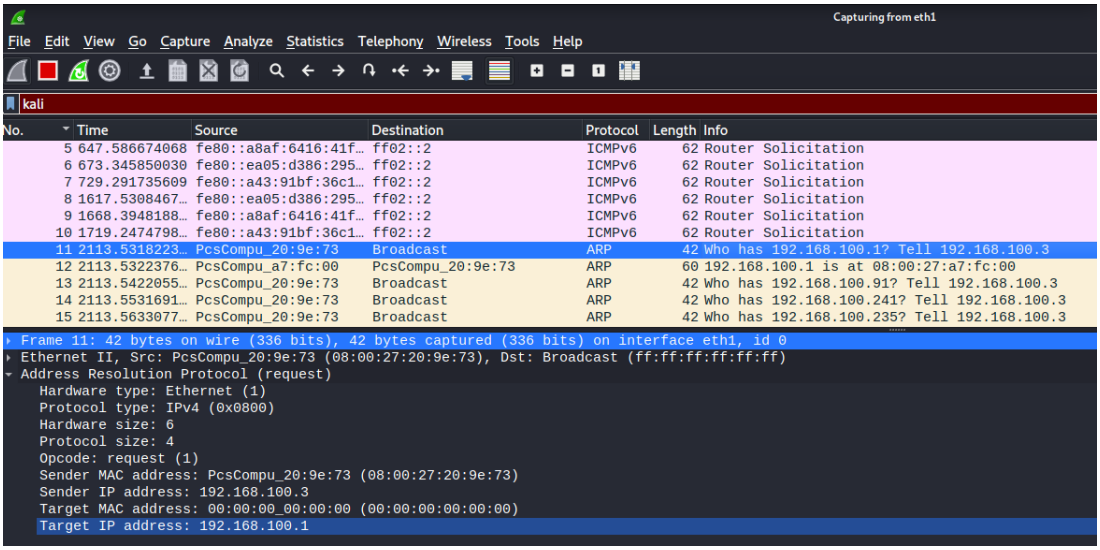


Figura 10: Cattura in Wireshark delle ARP query generate da Ettercap, in esecuzione nella VM hack01, finalizzate al discovery di tutti gli host della sua subnet.

08:00:27:20:9e:73 corrispondente alla eth1 di hack01 in luogo di quelli effettivi; in tal modo server01 e victim01 ora trasmettono i loro messaggi a hack01 (in the midle) convinti, invece, di dialogare direttamente l'uno con l'altro!

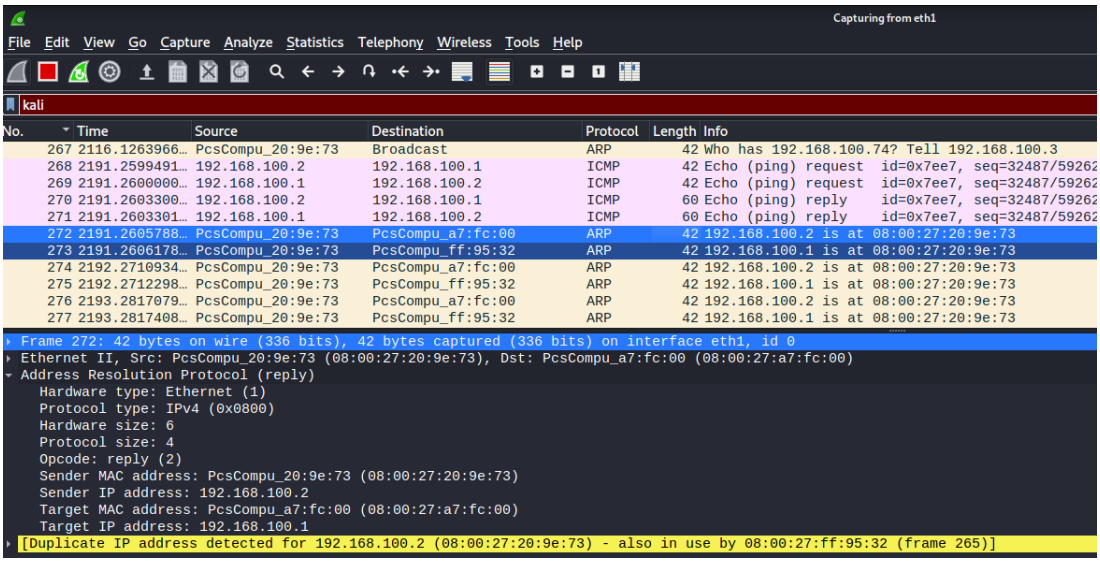


Figura 11: Cattura in Wireshark dei pacchetti relativi all'ARP poisoning, generato da Ettercap, ai danni delle VM 192.168.100.1 e 192.168.100.2 che ottengono come rispettivo mac address 08:00:27:20:9e:73 ovvero quello dell'attaccante.

Nella VM victim01

- si apre il web browser e si digita l'url *https://server01/AuthWebApp/* per utilizzare la web application (l'indirizzo è già memorizzato, per comodità nella cartella AuthWebApp dei bookmarks);
- alla comparsa dell'avviso di sicurezza del browser "*Warning Potential Security Risk Ahead*", dovuto al certificato self-signed utilizzato dal server, si clicca sul bottone Advanced e si visualizza il certificato;

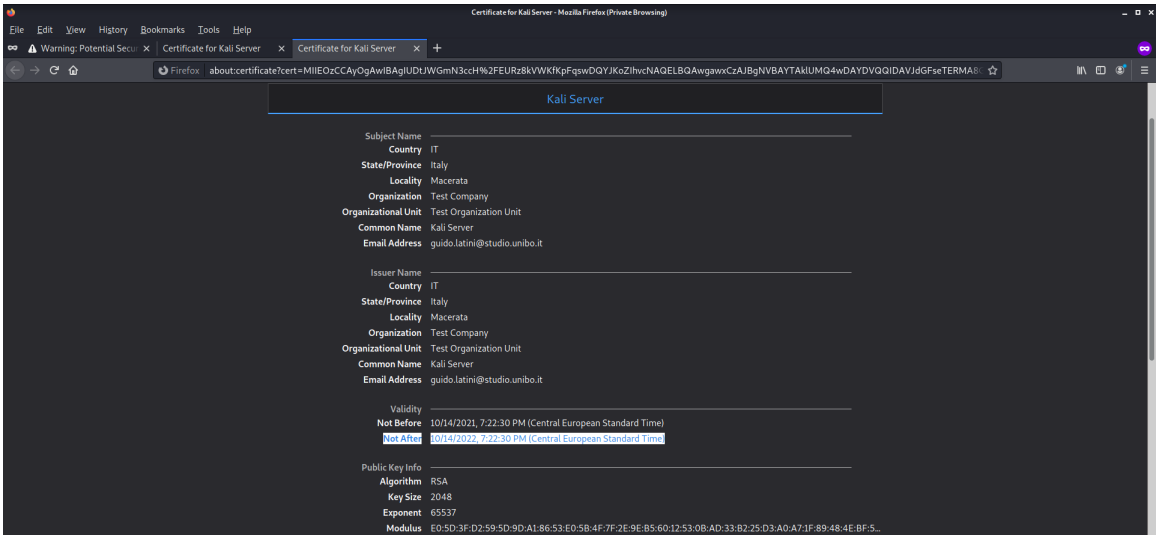


Figura 12: Certificato self-signed utilizzato dall'Apache della VM server01 per l'implementazione del protocollo https.

- si chiude, quindi, il tab del certificato e si clicca sul bottone "*Accept the Risk and Continue*", non essendo presente nella "Gestione certificati del browser" e non verificabile da una Certification Authority fidata, per accedere alla home page di AuthWebApp;
- si effettua l'autenticazione tramite la web form di AuthWebApp utilizzando uno dei 2 utenti configurati: ad esempio admin01 (Password01);
- si accede all'Admin Panel per la verifica dei parametri della connessione.

  Come evidenziato nello snaphost successivo, si osserva che l'Admin Panel rileva il valore 08:00:27:A7:FC:00, come mac address associato all'interfaccia eth1 della VM server01, e l'IP 192.168.100.3 ovvero l'IP della VM hack01, come indirizzo del client, anziché l'IP 192.168.100.2 della VM victim01; si deduce, quindi, che l'attacco MITM con l'arp poisoning è andato a buon fine.

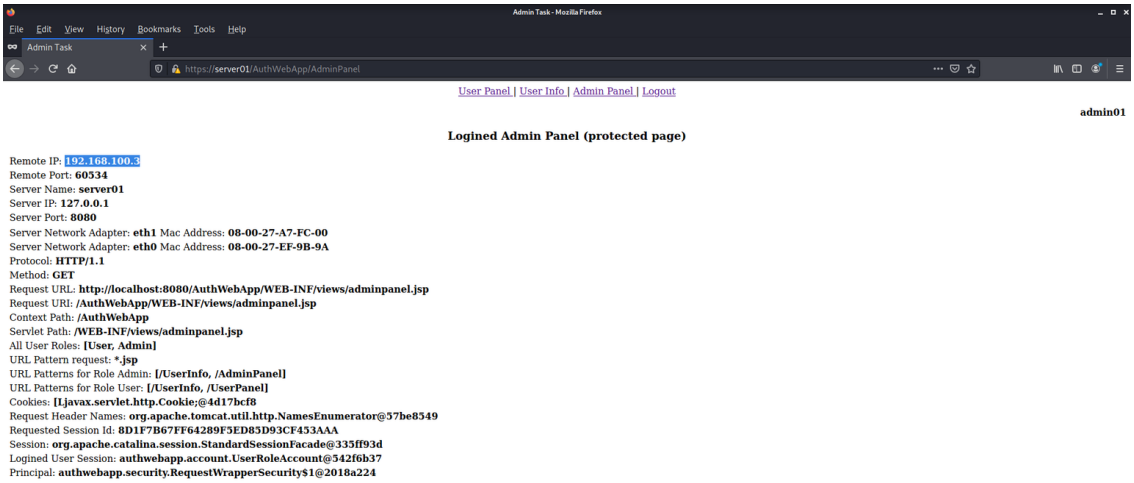


Figura 13: Admin Panel di AuthWebApp eseguito dal browser della VM victim01, di IP 192.168.100.2, in cui si evidenzia, la rilevazione da parte di server01 dell'IP 192.168.100.3 della VM hack01 a seguito del riuscito attacco MITM.

Nella VM hack01

- si utilizza il terminale di root aperto per visualizzare il mac address della VM associato all'interfaccia eth1 con il comando *ip a*
- si verifica che il mac address è il seguente 08:00:27:20:9e:73.

Nella VM server01

- si utilizza il terminale di root, aperto in precedenza, per avere un'ulteriore conferma della riuscita dell'arp poisoning per realizzare l'attacco MITM con il comando:

  arp -a;
- come illustrato nello snapshot seguente, dalla visualizzazione della ARP table della VM server01 si verifica che all'IP 192.168.100.2 della VM victim01 è associato lo stesso mac address della VM hack01, ovvero 08:00:27:20:9e:73.

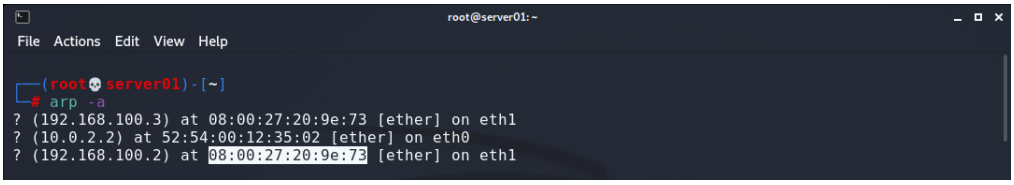


Figura 14: Arp table della VM server01 dopo l'arp poisoning effettuato dalla VM hack01.

Nella VM victim01

- si utilizza allo stesso modo un terminale di root per verificare la riuscita dell'arp poisoning sempre con il comando:

  arp -a;

- come illustrato nello snapshot seguente, analogamente a quanto rilevato nella VM server01, dalla visualizzazione della ARP table della VM victim01, si verifica che all'IP 192.168.100.1 della VM server01 è associato lo stesso mac address della VM hack01, ovvero 08:00:27:20:9e:73.

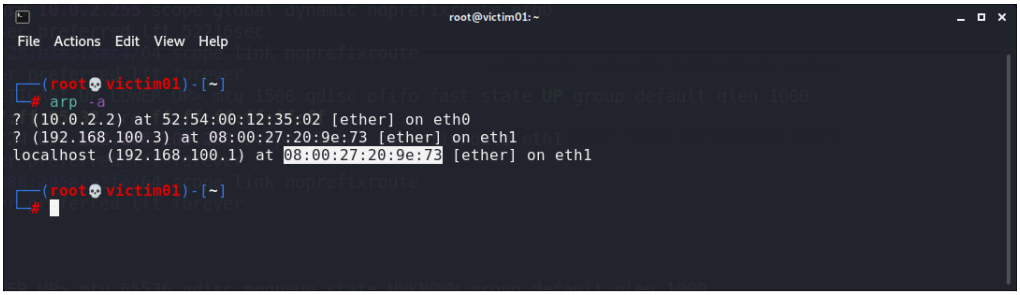


Figura 15: Arp table della VM victim01 dopo l'arp poisoning effettuato dalla VM hack01.

Nella VM hack01

- da *Ettercap* si arresta la cattura dei pacchetti con il bottone "*Start/Stop Sniffing*" e si visualizzano, quindi, le credenziali di autenticazione utilizzate come indicate nello snapshot successivo.

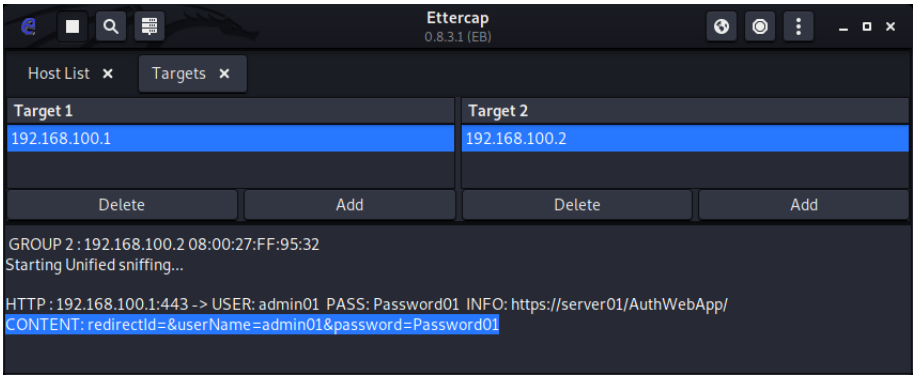


Figura 16: Credenziali di autenticazione di AuthWebApp, che utilizza l'https con Apache, configurato come reverse proxy per Tomcat, in ascolto nella porta TCP 443, catturate da Ettercap.

Dallo snapshot risulta, dunque, che:
HTTP:192.168.100.1:443− >USER:admin01 PASS:Password01
INFO:https://server01/AuthWebApp
CONTENT: redirectid=&userName=admin01&password=Password01

Si conclude il test relativo all'attacco MITM ai danni dei partecipanti al protocollo, utilizzato dall'applicazione AuthWebApp, che sfruttano la comunicazione crittografata con SSL/TLS, con l'esecuzione di un ulteriore tool spesso utilizzato anche per l'analisi forense: *SSLsplit*.

SSLsplit realizza l'attacchi di tipo MITM nel seguente modo: le connessioni sono intercettate in modo trasparente tramite un network address translation engine e reindirizzate allo stesso SSLsplit; il tool agisce terminando la connessione SSL/TLS ed avviandone una nuova all'indirizzo di destinazione originale, registrando anche tutti i dati trasmessi. Di seguito le istruzioni per l'esecuzione dell'ulteriore attacco di tipo MITM, nello scenario preso in esame, con SSLsplit.
Nella VM hack01

- dal terminale di root aperto si esegue il seguente comando per attivare la redirezione del SSL dalla VM server01 a quella dell'attaccante con SSLplit:

  sslsplit -D -l connections.log -j /tmp -S /tmp -k ca.key -c ca.crt ssl 192.268.100.103 443 server01 443;

il significato delle opzioni utilizzate è il seguente:

- *-D* attiva il debub mode;
- *-l connection.log* permette il log delle connessioni nel file indicato;
- *-j /tmp* imposta il chroot jail, per isolare il processo ed i suoi figli dal resto del sistema, nella dir indicata;
- *-S /tmp* salva i file nella dir indicata;
- *-k ca.key* specifica il file della chiave privata relativa al certificato da utilizzare;
- *-c ca.crt* specifica il file del certificato;
- *ssl* indica il protocollo crittografico da utilizzare;
- *192.168.100.3 443* indicano, rispettivamente, l'IP della VM hack01, che funge da proxy, e la porta TCP 443 da utilizzare;
- *server01 443* indicano, rispettivamente, l'host e la porta TCP oggetto del reindirizzamento; si ricorda, a riguardo, che nelle VM la risoluzione dei nomi viene realizzata tramite il file /etc/hosts.

Dallo snaphost seguente si noti il certificato self-signed, generato dall'attaccante, utilizzato da SSLsplit per reindirizzare il servizio SSL dall'IP della VM server01, porta TCP 443, all'IP 192.168.100.103 della VM hack01 sulla stessa porta TCP 443

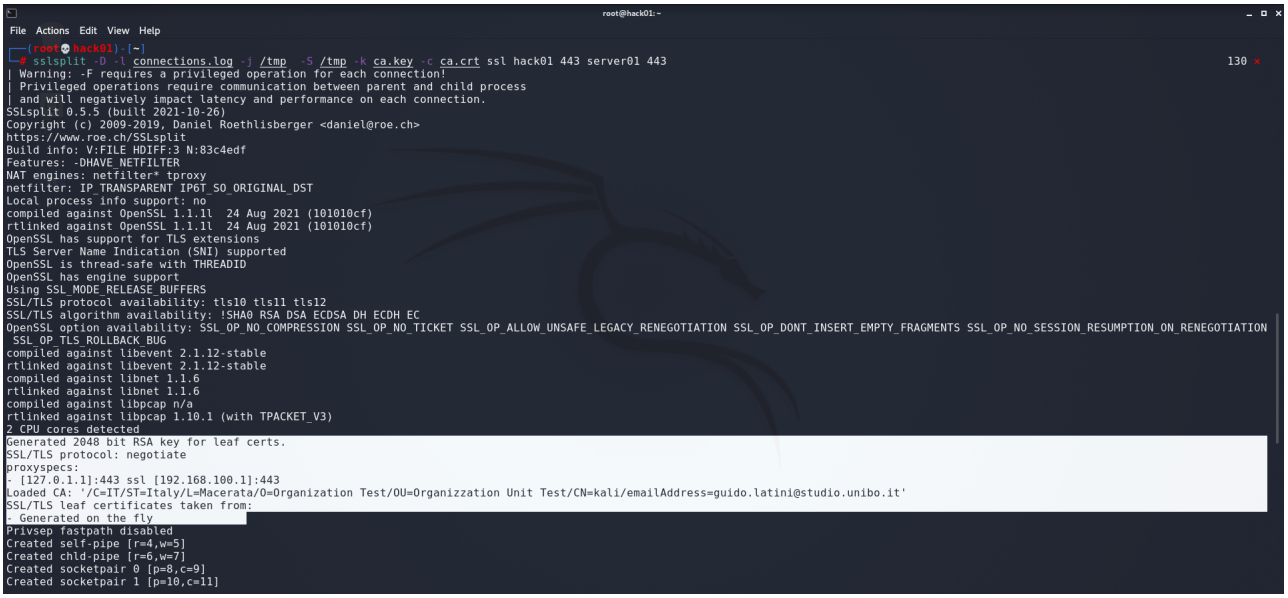


Figura 17: Esecuzione in debug mode di SSLsplit per l'attacco MITM ai danni dei partecipanti al protocollo di autenticazione che utilizza l'https mediante Apache, configurato come reverse proxy per Tomcat, in ascolto nella porta TCP 443, per consentire la fruizione di AuthWebApp.

Nella VM victim01

- si apre il browser e si digita l'url *https://192.168.100.3/AuthWebApp*, dove si ricorda che 192.168.100.3 è l'IP della VM hack01 (l'indirizzo è già memorizzato, per comodità nella cartella AuthWebApp dei bookmarks);
- alla comparsa dell'avviso di sicurezza del browser "*Warning Potential Security Risk Ahead*", dovuto al certificato self-signed utilizzato da hack01, si clicca sul bottone *Advanced* e si visualizza il certificato dal link "*View Certificate*";
- si verifica, in modo consapevole, che il tool SSLsplit eroga il servizio proxy per conto della VM server01 tramite la VM hack01; infatti il certificato proposto al browser è differente da quello utilizzato dalla VM server01: tra le varie differenze, si nota che il certificato delle VM hack01 scade il 14/12/2022 mentre quello delle VM server01, precedentemente illustrato, scade il 14/10/2022;
- si chiude, quindi, il tab del certificato e si clicca sul bottone "*Accept the Risk and Continue*" per accedere alla home page di AuthWebApp; in tal modo si è accettata la redirezione e, di conseguenza, d'ora in poi l'autenticazione prevista da AuthWebApp può essere gestita in modo fraudolento dall'attaccante;
- si apre il *Certificate manager* del browser, accessibile seguendo il seguente percorso *Edit* → *Preferences* → *Privacy & Security* → *View Certificates*;

- si verifica, come mostrato nello snapshot successivo, che nel *Certificate manager* del browser della VM victim01 sono stati memorizzati i due differenti certificati indicati: uno relativo alla VM server01 e l'altro "fake" rilasciato dalla VM hack01;

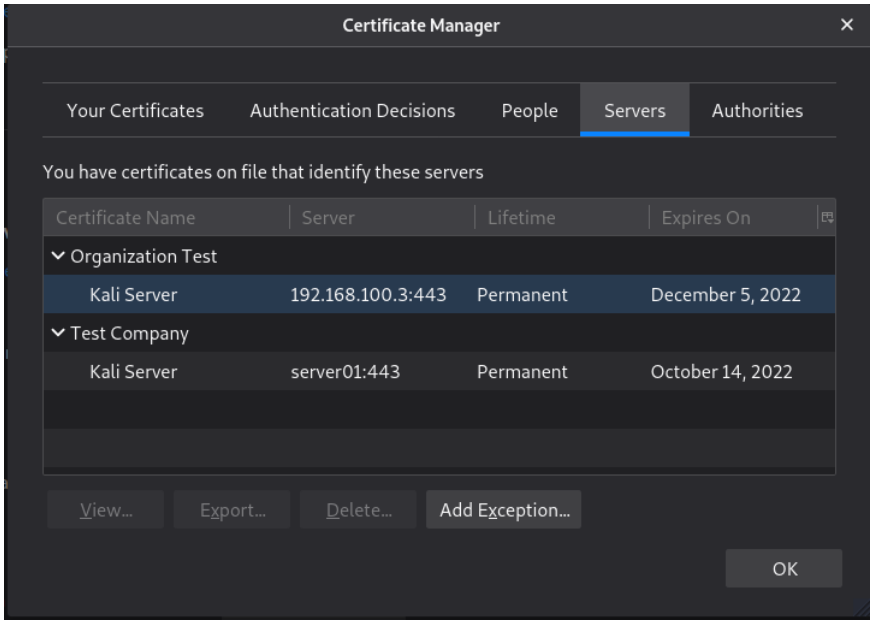


Figura 18: Certificate manager del browser della VM victim01 in cui si evidenziano il certificato rilasciato dalla VM server01 e quello fake di hack01, a seguito dell'attacco MITM condotto tramite SSLsplit, con il quale la VM hack01 funge in modo fraudolento da proxy per server01 utilizzando la stessa porta TCP 443, per la fruizione di AuthWebApp.

- si effettua l'autenticazione tramite la web form di AuthWebApp utilizzando l'utente con il ruolo di amministratore (admin01);
- si accede all'Admin Panel di AuthWebApp per la verifica dei parametri della connessione.

  Come evidenziato nello snaphost successivo, si osserva che

  Remote IP: 192.168.100.3

  ovvero l'indirizzo, rilevato dall'applicazione residente nella VM server01, corrisponde all'IP della VM hack01 che funge da proxy mediante l'attacco MITM con SSLSplit, e non all'IP della VM victim01 192.168.100.2.

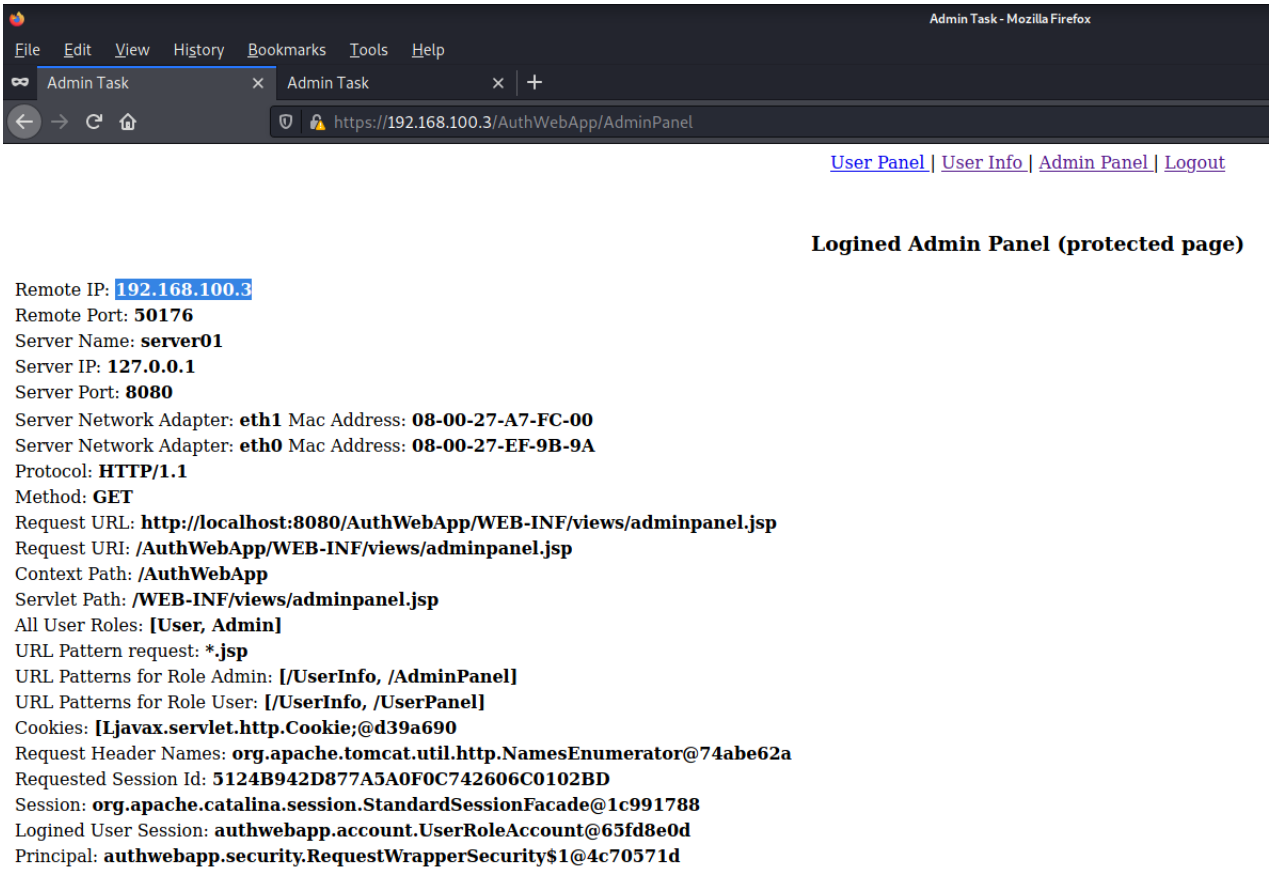


Figura 19: Admin Panel dell'applicazione AuthWebApp eseguita dal browser della VM victim01, di IP 192.168.100.2, in cui si evidenzia la rilevazione da parte di server01 dell'IP 192.168.100.3 della VM hack01 a seguito dell'attacco MITM con SSLSplit.

Nella VM hack01,

- dal terminale di root in cui è in esecuzione SSLsplit in modalità debug, si arresta il tool digitando $[CTRL] + [C]$;
- si apre, quindi, il file di log prodotto da SSLsplit nella dir */tmp* per visualizzare, tra gli altri, le credenziali intercettate che sono state utilizzate per l'autenticazione in AuthWebApp.

Nello snapshot seguente, relativo al file di log

20211206T115223Z-192.168.100.2,43492-192.168.100.1,443.log,

tali credenziali sono state evidenziate.

Figura 20: Visualizzazione parziale del file di log prodotto da SSLsplit, a seguito dell'attacco MITM effettuato dalla VM hack01 di IP 192.168.100.3, in cui si evidenziano le credenziali intercettate ad un utente che si è autenticato in AuthWebApp dalla VM victim01 di IP 192.168.100.2.

Sulla base dei risultati ottenuti dagli attacchi effettuati, si riportano le fasi del *vulnerability assesment* relative allo scenario esaminato.

Test: attacchi ai sistemi con Ettercap, mediante Man In The Midle (MITM) con arp-poisoning e SSLsplit.
Analisi: le origini delle vulnerabilità sono da imputare alla

- debolezza del protocollo di autenticazione che, tra gli altri, non utilizza la crittografia;
- configurazione del web server Apache che non prevede l'utilizzo di un dominio validato da un'autorità di certificazione affidabile e di un certificato digitale, valido per tale dominio, firmato dalla stessa autorità,
- mancanza di dispositivi di protezione di packet filtering ed egress filtering nei sistemi che eseguono il protocollo di autenticazione previsto da AuthWebApp.

Assessment con ranking basato su una classificazione con tre livelli di gravità crescente ("lieve","media","grave") di non conformità alle indicazioni dell'AgID":

- protocollo di autenticazione utilizzato da AuthWebApp: vulnerabilità grave;
- configurazione del web server Apache che non prevede l'utilizzo di un dominio validato da un'autorità di certificazione affidabile: vulnerabilità grave;

- configurazione del web server Apache che non prevede l'utilizzo di un certificato, valido per il dominio utilizzato, firmato da una CA attendibile: vulnerabilità grave;
- mancanza di dispositivi di protezione di packet filtering ed egress filtering nei sistemi che eseguono il protocollo di autenticazione previsto da AuthWebApp: vulnerabilità grave.

Remediation:

- sostituzione del protocollo di autenticazione utilizzato da AuthWebApp con un protocollo di autenticazione forte;
- nell'ottica del rispetto delle indicazioni dell'AgID, considerate le criticità riscontrate, si è ritenuto opportuno dedicare alle ulteriori strategie di remediation una sezione dedicata nella quale sono approfonditi l'implementazione del SSL/TLS e le misure più idonee da adottare in tutti gli scenari indicati.

3.3.5 Approfondimento sull'implementazione del SSL/TLS come una possibile strategia di remediation alle debolezze del protocollo di autenticazione utilizzato da AuthWebApp

Come sottolineato dai risultati dell'autovalutazione eseguita nei tre scenari descritti, emerge, oltre alla necessità di una sostituzione del protocollo di autenticazione utilizzato da AuthWebApp per le debolezze indicate, l'esigenza di utilizzo del protocollo SSL.

In ogni caso va evidenziato che, dei tre scenari esaminati, solamente l'ultimo, ovvero il protocollo di autenticazione di AuthWebApp che utilizza l'HTTPS, con Apache configurato come Reverse Proxy per Tomcat, in ascolto nella porta TCP 443, è consigliabile in un ambiente di produzione reale, seppur con le indicazioni di seguito descritte che possono rappresentare una strategia di remediation alle debolezze del protocollo di autenticazione implementato; a riguardo, considerato che il protocollo HTTPS integra l'interazione del protocollo HTTP attraverso un meccanismo di crittografia; si fornisce una descrizione del protocollo SSL/TLS e, a seguire, si approfondiscono le strategie di remediation da attuare utilizzando anche tale protocollo al fine di evitare che attacchi quali il MITM, condotti con Ettercap e SSLStip, vadano a buon fine.

Come noto il protocollo HTTPS è stato progettato per resistere a tali attacchi (con eccezione delle versioni più obsolete e deprecate del Protocollo SSL); l'HTTPS, infatti, a differenza del protocollo HTTP che non è criptato, e quindi vulnerabile alle intercettazioni e ad attacchi di tipo MITM (come illustrato nei primi 2 casi d'uso), offre protezione da tali attacchi se sia nel client che nel server sono adottate le corrette configurazioni.

Con riferimento al modello TCP/IP, il protocollo HTTPS opera al livello più alto, ovvero al livello di applicazione mentre il TLS ad un sotto-strato più basso dello stesso livello; in pratica, tra il protocollo TCP e HTTP si interpone, trasparentemente all'utente, un livello di crittografia/autenticazione SSL/TLS, che cripta il messaggio HTTP prima della trasmissione e decripta lo stesso una volta giunto a destinazione; la figura successiva evidenzia il layer SSL ed il suo rapporto con l'HTTP.

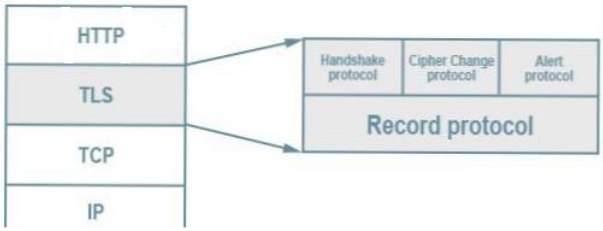


Figura 21: I layers del protocollo TLS/SSL in cui si evidenziano i protocolli di livello superiore che si interfacciano con il protocollo HTTP ed il record protocol che, invece, si interfaccia con il protocollo di trasporto affidabile TCP.

Nel HTTPS, a riguardo, il canale di comunicazione criptato tra client e server viene creato attraverso uno scambio di certificati; una volta stabilito il canale, al suo interno viene utilizzato il protocollo HTTP per la comunicazione; tale comunicazione garantisce che solamente il client e il server siano in grado di conoscere il contenuto dei messaggi scambiati; infatti nel messaggio HTTPS qualunque informazione è cifrata, inclusi gli header e il carico di richiesta/risposta del messaggio; pertanto, in una comunicazione HTTPS, in cui il client ed il server adottano opportune misure di sicurezza, un attaccante che prova ad intercettare i messaggi riesce solamente a conoscere i nomi di dominio, gli indirizzi IP delle parti ed a sapere che è avvenuta una connessione; tali caratteristiche hanno contribuito alla diffusione del TLS/SSL con particolare riferimento alla protezione delle form (è il caso di AuthWebApp) che richiedono, ad esempio, l'invio di credenziali, di dati riservati per una particolare autorizzazione, di ordini e/o pagamenti.

Il *protocollo SSL (Secure Socket Layer)*, di cui il *TLS (Trasport Layer Security)*, è il successore, uno standard aperto e non proprietario, è nato al fine di garantire la privacy delle comunicazioni su Internet; infatti permette alle applicazioni client/server di comunicare in modo da prevenire le intrusioni, le manomissioni e le falsificazioni dei messaggi. Il protocollo garantisce la sicurezza del collegamento mediante tre funzionalità fondamentali:

- *la riservatezza del collegamento*; ganantisce un collegamento sicuro tra due utenti coinvolti in una comunicazione: i dati vengono protetti utilizzando algoritmi di crittografia a chiave simmetrica;
- *l'autenticazione*; l'autenticazione dell'identità nelle connessioni può essere eseguita usando la crittografia a chiave pubblica; in tal modo i client sono sicuri di comunicare con il server corretto, prevenendo eventuali interposizioni; inoltre è prevista la certificazione sia del server che del client;
- *l'affidabilità*; il livello di trasporto include un controllo sull'integrità del messaggio basato su un apposito MAC che utilizza funzioni hash sicure; in tal modo è possibile verificare che i dati spediti tra client e server non siano stati alterati durante la trasmissione.

Gli scopi del protocollo SSL possono essere sintetizzati nella

- *sicurezza del collegamento*; stabilisce un collegamento sicuro tra sistemi;
- *interoperabilità*; consente agli sviluppatori di accordarsi sui parametri utilizzati dagli algoritmi di crittografia senza che l'uno conosca il codice l'uno dell'altro;
- *ampliamento*; fornisce una struttura all'interno della quale i futuri metodi di crittografia a chiave pubblica e chiave simmetrica possano essere incorporati senza la necessità di creazione di un nuovo protocollo;
- *efficienza*; incorpora uno schema di session caching opzionale per ridurre il numero di collegamenti che necessitano di essere stabili ex novo, riducendo così le attività sulla rete e quelle delle CPU per la gestione delle operazioni di crittografia.

Il protocollo SSL, come illustrato nella figura precedente, utilizza i seguenti componenti:

- *protocollo SSL Handshake*; permette alle parti di autenticarsi a vicenda e di negoziare un algoritmo di crittografia e le relative chiavi prima che il livello di applicazione trasmetta o riceva il suo primo byte. Un vantaggio del SSL Handhake è l'indipendenza dal protocollo di applicazione: in tal modo un protocollo di livello più alto (es, l'HTTP) può interfacciarsi con il SSL in modo trasparente;
- *protocollo SSL Record*; responsabile della compressione, dell'integrità e della cifratura, è usato per l'incapsulamento dei dati proveniente dai protocolli superiori;
- *protocollo di Alert*; notifica eccezioni che possono avvenire nella comunicazione con appositi messaggi che contengono il livello di severità ed una descrizione dell'evento che si è verificato; ad esempio, un messaggio con livello "fatal" si risolve con l'immediata terminazione della sessione; i messaggi di alert, analogamente agli altri, sono cifrati e compressi;
- *protocollo Change CipherSpec*; comunica i cambiamenti nella strategia di cifratura; è comunemente usato come parte del metodo handshake per negoziare le modifiche dei parametri di sicurezza.

Il protocollo SSL Handshake utilizza una combinazione di chiavi pubbliche e chiavi simmetriche; a riguardo si faccia riferimento alla figura relativa al terzo scenario riportato nella sezione dedicata al caso di studio.
Una sessione SSL inizia sempre con uno scambio di messaggi di SSL handshake; l'handshake consente al server di autenticarsi al client utilizzando una tecnica a chiave pubblica, quindi permette al client ed al server di cooperare per la creazione delle chiavi simmetriche usate per una veloce cifratura, decifratura e controllo delle intrusioni durante la sessione avviata; eventualmente l'handshake permette anche al client di autenticarsi al server.
Nel protocollo una nuova sessione può essere avviata sia dal client che dal server; nel primo caso, il client invierà un messaggio di *client hello*, iniziando così la fase di hello, e

si porrà in attesa della risposta del server che avverrà con un messaggio di *server hello*; nel secondo caso, invece, il server inizierà la connessione inviando un messaggio di *hello request* per chiedere al client di iniziare la fase di hello. Con lo scambio dei messaggi indicati, il client ed il server si accordano sugli algoritmi da usare per la generazione delle nuove chiavi; in particolare il client ne propone una lista al server che server sceglie quale utilizzare.
Dopo la negoziazione dell'algoritmo, a seconda del metodo di autenticazione impiegato (autenticazione di entrambe le parti, del server con client non autenticato, totale anonimato) può iniziare o meno uno scambio di certificati tra client e server.
L'autenticazione consente di provare l'identità di un client o di un server; un client abilitato può controllare che il certificato del server sia valido e che sia firmato da una Certificate Authority (CA) fidata; *pertanto, con riferimento allo scenario analizzato, in cui il client della VM victim01 si collega alla VM server01 per utilizzare l'applicazione AuthWebApp, tale verifica è fondamentale per avere la garanzia dell'effettiva identità del server e per evitare gli attacchi di tipo MITM descritti*; allo stesso modo un client può essere autenticato e, dunque, *sempre per quanto concerne il caso di studio, in tale contesto, il client della VM hack01 non sarebbe in grado di collegarsi in modo fraudolento alla VM server01 spacciandosi per quello della VM victim01.*

E' importante sottolineare che l'autenticazione del client, analogamente a quella del server, implica la cifratura di alcuni dati condivisi con una chiave pubblica o privata e la decifratura con la chiave corrispondente.
Nel caso dell'*autenticazione del server*, il client deve cifrare i dati segreti con la chiave pubblica del server; il fatto che solamente il server possieda la chiave privata per la decifrazione del segreto, permette allo stesso client di avere le dovute assicurazioni sulla reale identità del server; infatti, in caso di intromissione di un avversario che intende spacciarsi per il server, lo stesso, non disponendo della chiave privata, non riesce a generare le chiavi simmetriche richieste per la sessione che, di conseguenza, viene terminata (si faccia riferimento, al solito, alle figura relativa allo scenario in esame).
In caso di *autenticazione del client*, invece, lo stesso deve cifrare alcuni valori casuali condivisi con la sua chiave privata, creando una firma; la chiave pubblica nel certificato del client può facilmente convalidare tale firma se il certificato è autentico, in caso contrario la sessione sarà terminata.

Avvenuta l'autenticazione, si procede con la generazione della chiavi per la cifratura e per l'autenticazione dei dati provenienti dal livello applicazione, attraverso i messaggi di *server key exchange* e *client key exchange*; terminata tale operazione, il server annuncia la fine della fase di Hello al client, con l'invio di un messaggio di *server hello done*, dopodiché, con l'invio di un messaggio di *change ChiperSpec*, entrambi controllano la correttezza dei dati ricevuti; se tutto è avvenuto in modo corretto, client e server utilizzano gli algoritmi di sicurezza concordati. La fase di handshake termina con l'invio, da entrambe le parti, di un messaggio di *finished* che è il primo dato ad essere cifrato.

Di seguito vengono illustrate, in dettaglio, le 4 fasi che caratterizzano il *protocollo SSL Handshake.*

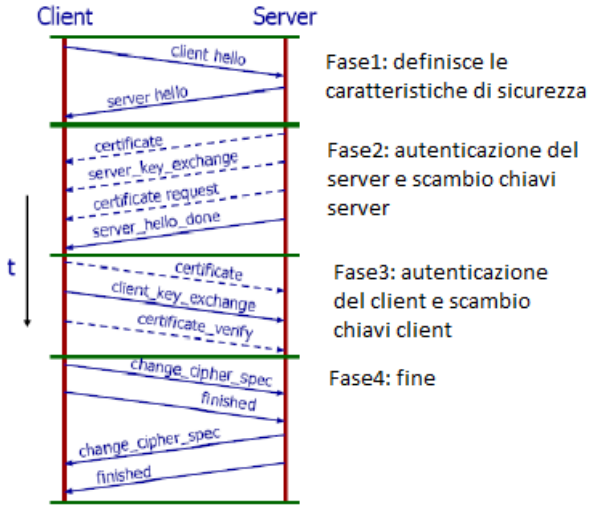


Figura 22: Le 4 fasi del protocollo SSL Handshake. I messaggi contrassegnati con la linea tratteggiata non sono obbligatori; il messaggio *change ChiperSpec* non fa parte del protocollo ma viene inviato in quella posizione.

Si deve osservare che il client ed il server possono decidere di riabilitare una precedente sessione o di duplicarne una esistente invece di negoziare di nuovo i parametri di sicurezza; in tal caso di parla di *riesumazioni di sessioni*; nella figura seguente di illustrano le fasi del protocollo SSL Handshake con riesumazione.

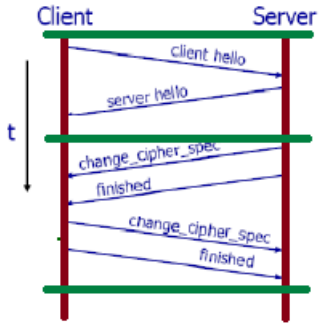


Figura 23: Le fasi del protocollo SSL Handshake con riesumazione.

Il client invia un *client hello* usando il Session ID della sessione da riesumare; il server allora controlla la sua session cache per trovare una corrispondenza; se la corrispondenza è trovata il server manda un *server hello* con lo stesso Session ID; a tal punto sia il client che il server devono mandare un messaggio di *change CipherSpec* e procedere direttamente fino al messaggio di *finished*; nel caso in cui il server non trovi la corrispondenza del Session ID nella propria session cache, genererà un nuovo Session ID e sarà eseguito un handshake completo.

Entrando nel dettaglio dei messaggi secondo l'ordine di spedizione previsto dalle 4 fasi del protocollo SSL handshake, nella *fase 1, che definisce le caratteristiche di sicurezza*, i messaggi di Hello vengono utilizzati per avviare una nuova connessione; l'inizio della connessione può avvenire in 2 modi, da parte del client, con l'invio del messaggio di:

- *hello request* da parte del server; tale messaggio può essere inviato dal server in ogni momento ma viene ignorato dal client se il protocollo di handshake è iniziato;
- *client hello* da parte del client; al quale il server deve rispondere con un messaggio di *server hello*; qualora il server non dovesse rispondere si verifica un errore fatale e la connessione fallisce.

Il messaggio di *client hello* è utilizzato da un client per avviare una nuova connessione con un server, rispondere ad un *hello request* e rinegoziare i parametri di sicurezza di una connessione esistente; ha i seguenti campi con indicazioni relative al:

- *protocol version*: 2 byte utilizzati per identificare la versione di SSL in uso;
- *random byte*: composto da un timestamp di 32 bit ed il nonce di 38 bit random;
- *session identifier*: costituto da 32 byte contenenti l'identificativo di una precedente sessione di una connessione che poterebbe essere riesumata; se i byte sono tutti zero indicano che si tratta di una nuova sessione;
- *lista delle CipherSuite*: l'elenco contenente le combinazioni degli algoritmi di crittografia supportati dal client in ordine di preferenza;
- *lista dei compression method*: un elenco di algoritmi di compressione, in ordine di preferenza, supportati dal client; se il server non sopporta nessuno di quelli specificati dal client, la sessione fallisce.

Dopo aver inviato un *client hello*, il client aspetta un messaggio di *server hello*; qualunque altro messaggio di handshake, fatta eccezione per un *hello request* che sarebbe ignorato, provocherebbe un fatal error. In risposta al messaggio di *client hello*, il server invia un messaggio di *server hello* composto da:

- *protocollo version*: una coppia di byte che rappresentala versione del Protocollo scelto; tale valore corrisponde al minimo tra la versione del protocollo proposta dal client e la massima supportata dal server;
- *random byte*: byte casuali generati dal server indipendenti dai parametri omologhi forniti dal client;
- *session identifier*: identifica la sessione che deve essere avviata; se l'identificativo di sessione inviato dal client è diverso dal valore zero, il server cerca nella sua session cache un valore di riscontro: nel caso in cui lo trovi risponde con lo stesso valore indicato nel client hello e la vecchia sessione viene riesumata; in caso contrario, il server restituisce un identificativo vuoto per indicare che la sessione non era

memorizzata e quindi non può essere ripresa. Nel caso in cui l'identificativo nel *client hello* è pari a zero il server imposta un nuovo identificativo per la sessione da avviare;

- *CipherSuite*: una coppia di byte che rappresenta la famiglia di algoritmi di crittografia scelta dal server tra quelli proposti dal client;
- *compression method*: metodo di compressione scelto dal server tra quelli proposti dal client.

Per le sessione riesumate i precedenti due campi corrispondono a quelli della sessione ripresa.

Dopo lo scambio dei messaggi di hello, tra client e server può iniziare *la fase di autenticazione tramite lo scambio di un certificato ovvero una struttura dati composta, tra gli altri, da una chiave pubblica, da una stringa identificativa e dalla firma di un'autorità che lega la chiave all'identità; il tipo di certificato deve essere appropriato all'algoritmo selezionato nella CipherSuite per lo scambio di chiavi*; dunque nel certificato sono presenti le informazioni utili all'esecuzione dell'algoritmo; generalmente si utilizza un *certificato X.509 v.3*; tra i campi del certificato si evidenziano i seguenti: *version, serial number, signature algorithm ID, issuer name, validity period, subject name, subject public key information, issuer unique identifier, subject unique identifier, extensions* e la *firma* dei campi precedenti.
Dunque, nella *fase 2*, il server invia al client il messaggio di *server hello* con indicazioni relative al:

- *server certificate*; il certificato è richiesto in tutti i casi fatta eccezione per il Diffie-Helmann anonimo;
- *server key exchange*; è richiesto in tutti i casi fatta eccezione per l'RSA ed il Diffie-Helmann anonimo;
- *certificate request* ovvero la richiesta inviata dal server al client per ottenere il certificato utilizzato per fornire le informazioni necessarie all'algoritmo di scambio di chiavi; in particolare, il messaggio contiene una lista di tipi di certificati, ordinati secondo le preferenze del server, ed una lista di nomi di autorità fidate che emettono certificati; nel caso in cui un server anonimo richieda l'identificazione del client si verifica un fatal error;
- *server hello done*; indica la fine dei messaggi del server iniziati con il messaggio di *server hello*;

Nelle *fase 3*, che definisce l'autenticazione del client e lo scambio delle chiavi client, il client verifica il certificato ed i parametri del server; nel caso in cui sia tutto soddisfacente, trasmette al server i seguenti 3 messaggi:

- *client certificate*; è il primo messaggio che il client può inviare dopo aver ricevuto il messaggio *server hello done* e la richiesta di un certificato da parte del server;

se non è disponibile un certificato, il client lo segnalerà per mezzo di un messaggio di alert, al quale il server potrebbe rispondere con un fatal error se è necessaria l'autenticazione del client;

- *client key exchange*; è un messaggio che deve essere inviato obbligatoriamente dal client al fine di fissare il "*pre master secret*" necessario per calcolare il "*master secret*", ovvero un valore condiviso da client e server necessario alla generazione di chiavi MAC, chiavi simmetriche e vettori di inizializzazione; il contenuto del messaggio dipende dall'algoritmo per lo scambio di chiavi specificate nella *CipherSuite*;

- *certificate verify*; è un messaggio che fornisce una verifica esplicita del suo certificato; ovvero il client invia un hash dei messaggi di handshake, scambiati a partire da *client hello* fino a questo punto, e della "*master secret*", in modo che il server possa verificarne l'autenticità.

Nella *fase 4* il client invia i messaggi di

- *Change ChiperSpec*; il messaggio ha lo scopo di segnalare i cambiamenti delle strategie di cifratura; è inviato dal client per comunicare alla parte ricevente che i dati che seguono saranno protetti dalle chiavi e dal *CipherSpec* appena negoziato; il client invia un messaggio di *change CipherSpec* dopo l'invio dei messaggi di *key exchange* e *certificate verify* della fase di handshake; quando si riprende una vecchia sessione il messaggio di *change CipherSpec* è inviato dopo i messaggi di *hello*;

- *Finished*; il messaggio è sempre inviato dopo un messaggio di *change CipherSpec* per verificare che i processi di scambio di chiavi e autenticazione abbiano avuto successo; se non è preceduto da tale messaggio si verifica un fatal error; è il primo messaggio protetto con l'algoritmo negoziato.

Il server risponde inviando i messaggi di:

- *Change ChiperSpec*; analogamente a quanto compiuto dal client, il messaggio è inviato dal server per comunicare alla parte ricevente che i dati seguenti saranno protetti dalle chiavi e dal *CipherSpec* appena negoziato; il server invia il *Change ChiperSpec* dopo che il messaggio di *key exchange* è stato ricevuto dal client con successo;

- *Finished*; comunica il successo del processo; la fase di handshake è, quindi, completata ed client ed server possono iniziare a scambiare i dati del livello applicazione.

Il *protocollo record* ha il compito di gestire i messaggi che devono essere trasmessi, della frammentazione in blocchi di dati (record di 214 byte o meno), della compressione, dell'applicazione di un MAC (che viene utilizzato per il protocollo dell'integrità dei dati), delle cifratura, dell'aggiunta di un header (con le informazioni relative al protocollo, la dimensione ed il tipo di record) e della trasmissione del risultato; i dati ricevuti, una volta che sono stati decifrati, verificati, decompressi e riassemblati, vengono, quindi,

trasmessi al livello più alto; dunque il protocollo record, come illustrato nella figura relativa alla gestione dei dati, è responsabile della *confidenzialità*, ottenuta ricorrendo ad un chiper simmetrico con una shared key definita dal protocollo handshake, e dell'*integrità* utilizzando un MAC con shared key simile ad HMAC.

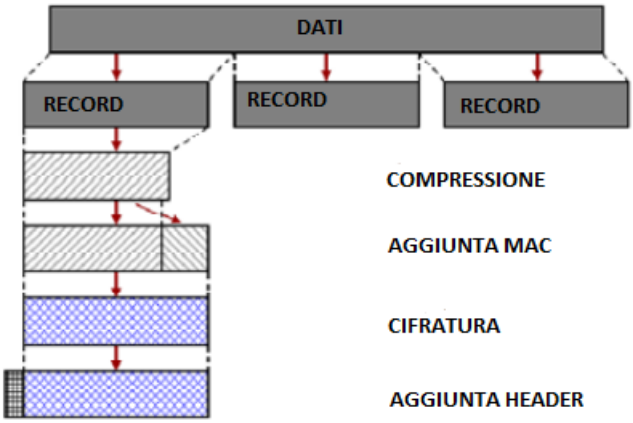


Figura 24: La gestione dei dati da parte del protocollo SSL Record.

Il *Trasport Layer Security* (TLS), definito nella RFC 2246, come anticipato, è il successore del SSL. Seppur molto simile al SSL v3, il TLS offre, tra gli altri, una migliore sicurezza per le caratteristiche di tolleranza alla criptoanalisi, di conformità ai criteri del Federal Information Processing Standards (FIPS) 140-3 (Security Requirements for Cryptographic Modules), di progettazione di MAC e di funzioni hash; inoltre nel TLS, a differenza del SSL, sono stati definiti codici di alert addizionali, introdotti cambiamenti nei cipher supportati, variazioni sui tipi di certificati, negoziazioni e variazioni sul calcolo dei parametri criptografici; *per tali motivi, dunque, l'utilizzo del TLS è decisamente consigliato per la protezione di un sito web indipendentemente dal tipo di informazioni trattate.*

Il TLS, come illustrato, è stato utilizzato per nell'implementazione dell'ultimo scenario esaminato, facendo ricorso al package OpenSSL per la gestione dei certificati digitali; tale package è stato utilizzato nella VM server01 per l'implementazione dell'https mediante Apache e nella VM hack01 per la configurazione del tool SSLSplit, necessitando anch'esso di un certificato per l'erogazione del servizio di proxy TLS/SSL trasparente; come visto il servizio proxy erogato da SSLSplit consente l'attacco di tipo MITM ai danni della VM victim01 che utilizza il protocollo https per connettersi alla VM server01.

Tra i punti di forza del package, va evidenziato che è open source e, continuamente sottoposto a revisioni e manutenzioni; OpenSSL è, inoltre, utilizzato per lo sviluppo di importanti applicazioni quali modSSL e contiene implementazioni di vari algoritmi di crittografia e delle codifiche dei formati dei certificati digitali (CER, CRT, PEM, ecc.). I certificati utilizzati nelle VM server01 e hack01, illustrati nel terzo scenario, sono stati realizzati con i seguenti comandi del package:

openssl genrsa -out ca.key 4096

e

openssl req -new -x509 -days 365 -key ca.key -out ca.crt.

Il primo comando genera una chiave privata RSA a 4096 bit in formato PEM, *ca.key*, mentre il secondo utilizza tale chiave privata per generare un certificato CA radice, *ca.crt*, self-signed con la validità di un anno; al lancio del comando sono richieste le seguenti informazioni che vengono inserite nel certificato: *Country Name*, *State Name*, *Locality Name*, *Organization Name*, *Organizational Unit Name*, *Common Name* ed *Email Address.*

In base a quanto argomentato, per un'efficace remediation, anche nel rispetto delle indicazioni dell'AgID per la modellazione delle minacce ed individuazione delle azioni di mitigazione conformi ai principi del secure/privacy by design, finalizzata a rendere sicuro il protocollo di autenticazione utilizzato da AuthWebApp, le misure da adottare possono essere le seguenti:

- *seguire le "best practice" nella progettazione del protocollo di autenticazione*; l'autenticazione utilizzata da AuthWebApp, realizzata a scopo didattico, prevede un banale confronto tra le credenziali immesse dall'utente e quelle memorizzate in chiaro in un file di una classe java; *in un ambiente di produzione tali credenziali devono essere memorizzate in modo cifrato, secondo quanto previsto dalla normativa vigente, in un database, ospitato in un sistema dedicato, accessibile all'applicazione utilizzata dal web server unicamente mediante un protocollo che utilizza la crittografia*; in tal modo si limita il rischio di accesso alla credenziali in caso di una possibile violazione del front end del sistema; inoltre, a maggiore protezione dell'autenticazione, è opportuno utilizzare una *Multi-Factor Authentication* (MFA) che richieda un secondo canale per la verifica tramite challenge-response con *Time-based One-Time Password* (TOTP), come visto a proposito delle considerazioni sulla debolezza del protocollo S/KEY; tale soluzione, infatti, offre una maggiore protezione anche da tentativi di attacchi di brute force;

- *utilizzare esclusivamente il protocollo HTTPS*; si è più volte evidenziato che il protocollo HTTP, utilizzato nel primo scenario da Apache-Tomacat e nel secondo dall'Apache, configurato come reverse proxy per Tomcat, consente facilmente l'intercettazione delle credenziali di autenticazione tramite attacchi di eavesdropping essendo le stesse trasmesse in chiaro; l'utilizzo del HTTPS, grazie al protocollo SSL/TLS, per le caratteristiche indicate, permette di mitigare i rischi dovuti a tali attacchi;

- *utilizzare una Certification Authority (CA) affidabile per la validazione del dominio e per la firma del certificato associato utilizzato dal Web Server*; l'utilizzo di tale CA risulta, infatti, fondamentale per avere un dominio validato per la pubblicazione di AuthWebApp e per la conseguente firma del certificato utilizzato dal web server da parte della stessa CA; come già sottolineato, la principale criticità del protocollo

sta nel fatto che il certificato utilizzato dall'Apache di server01 è self-signed e, dunque, non affidabile in quanto non firmato da una CA riconosciuta dal browser delle vittima; inoltre nello scenario analizzato, in luogo di un servizio DNS basato su domini validati, è utilizzata una risoluzione dei nomi mediante il file hosts di ciascuna VM; *l'utilizzo di un certificato, firmato da una CA affidabile nota al browser ed associato ad un dominio validato, evita al client la ricezione dell'avviso di sicurezza al primo collegamento in HTTPS al server tramite un url che utilizza tale dominio*; inoltre mitiga i rischi di accettazione di un certificato considerato non affidabile, in quanto non emesso da un'autorità conosciuta, che può essere rilasciato da un servizio "fake" erogato dolosamente allo scopo di carpire le credenziali di autenticazioni; con l'adozione di tali misure, un certificato self-signed, come quello emesso dalla hack01 per l'esecuzione dell'attacco di tipo MITM con SSLsplit, non viene accettato in quanto considerato non attendibile;

- *effettuare una periodica analisi del protocollo di autenticazione e del software utilizzato per implementarlo*; il regolare utilizzo di strumenti diagnostici, quali i tool disponibili nella stessa distribuzione Kali, consente di rilevare eventuali note vulnerabilità relative ai servizi erogati ed al software sviluppato, al fine di una efficace applicazione di patch e/o aggiornamenti di sicurezza che mitigano il rischio di attacchi mirati a sfruttare le vulnerabilità evidenziate;

- *adottare una politica di sicurezza, adeguata agli scopi prefissi, a protezione del protocollo di autenticazione, estesa a tutti i sistemi interessati*; in base agli attacchi illustrati ai danni dei sistemi che partecipano al protocollo di autenticazione utilizzato da AuthWebApp, si evidenzia che una particolare attenzione deve essere dedicata alle configurazioni di sicurezza dei Web Server, Apache-Tomcat ed Apache, installati nella VM server01; oltre alla necessità di proteggere le chiavi private dei certificati utilizzati, una configurazione dei Web Server può prevedere anche eventuali *restrizioni di accesso consentito solamente a determinati IP autorizzati*; tali policy possono essere attuate a livello di configurazione degli stessi servizi web, del firewall in dotazione al sistema operativo e di altri apparati di rete che interagiscono con il web server; *l'utilizzo combinato di un packet filtering con un egress filtering a più livelli sicuramente riduce il rischio di attacchi quali il MITM e lo spoofing*; con riferimento ai casi illustrati, l'attacco dalla VM hack01, che prevede l'ARP poisoning sfruttando la debolezza intrinseca nel protocollo ARP, ovvero la mancanza di un meccanismo di autenticazione, può essere mitigato con l'adozione di appositi software di *network monitoring* che esaminano le attività di rete e ne evidenziano le discordanze; altre contromisure utili possono essere l'attivazione di servizi di *Intrusion Detection System* (IDS) e l'adozione di opportune politiche di sicurezza negli apparati di rete coinvolti quali, ad esempio, l'uso dello standard IEEE 802.1X ed il port security per assicurare un meccanismo di autenticazione ai dispositivi che desiderano collegarsi alla porte di accesso di tali apparati e l'utilizzo di protocolli dello standard IPsec che garantiscono la protezione dell'ARP spoofing.

4 Conclusioni

La realizzazione del progetto, incentrato sulla trattazione dei protocolli di autenticazione, il cui obiettivo è la verifica della rivendicazione di una determinata proprietà, ha permesso di fornire una panoramica di tali protocolli, a partire da un inquadramento più generale relativo alla sicurezza dei sistemi, analizzandone i limiti ed i punti di forza. A riguardo, nelle premesse concettuali introdotte, iniziando dall'illustrazione dei protocolli crittografici, sono stati esposti i principali problemi fondamentali sui quali si basa la crittografia per la sicurezza delle transazioni nei protocolli di autenticazione: il problema della fattorizzazione, del logaritmo discreto, di Diffie-Helmann e della residuosità quadratica; dalla formulazione di tali problemi si è focalizzata l'attenzione sulla sicurezza in termini computazionali, evidenziando, attraverso la trattazione degli algoritmi probabilistici, che la sicurezza dei sistemi è assicurata dalla difficoltà di risoluzione in tempi accettabili dei problemi indicati; ne deriva che il successo di un attacco condotto da un avversario, ovvero un algoritmo che opera in tempo polinomiale, ai danni di un crittosistema è valutabile in termini probabilistici.

Sono state, quindi, esposte le nozioni di base relative ai protocolli di autenticazione, necessarie per la loro caratterizzazione, le proprietà dei protocolli crittografici, le definizioni utili per classificarli in base agli obiettivi di sicurezza che raggiungono e le funzioni hash one-way, fondamentali per l'utilizzo delle applicazioni crittografiche necessarie per garantire la sicurezza nell'esecuzione dei protocolli di autenticazione.

Nella trattazione delle tipologie di autenticazione sono stati definiti i principali scenari la cui classificazione dipende dalle caratteristiche dei partecipanti alla comunicazione; alcuni di tali scenari sono stati approfonditi nel caso di studio oggetto della parte pratica del progetto.

L'esigenza di analizzare la "robustezza" dei protocolli di autenticazione ed i rischi nella sicurezza connessi al loro utilizzo ha motivato la trattazione degli attacchi ai quali possono essere sottoposti; dunque, a partire dal concetto di attacco ad un protocollo di autenticazione, si è passati alla definizione delle principali tipologie che ricorrono di frequente nell'illustrazione delle debolezze di ogni protocollo trattato; infatti, per ciascuno dei protocolli di autenticazione esaminato, sono state indicate le tipologie di attacco a cui è sensibile e le eventuali azioni più efficaci da intraprendere per mitigarne i rischi; dall'esposizione dei protocolli trattati, risulta evidente che ogni contromisura, applicata ad un protocollo per una tecnica di attacco, contribuisce ad incrementarne la sicurezza ma, nello stesso tempo, implica inevitabilmente un aumento della complessità sotto molteplici aspetti.

A conclusione della parte introduttiva, relativa alle premesse concettuali e nozioni di base, sono state definite le convenzioni di comportamento adottate dai partecipanti all'esecuzione dei protocolli di autenticazione; il rispetto di tali convenzioni è previsto anche per un avversario che conduce gli attacchi ai loro danni.

In accordo con l'obiettivo primario del progetto, nella sezione relativa all'esame dei protocolli di autenticazione, sono state analizzate le caratteristiche dei principali protocolli suddivisibili, fondamentalmente, nelle seguenti tre classi: basati su password, su challenge-response ed a conoscenza zero; per ciascuna classe sono state evidenziate le

peculiarità, osservando come la predetta suddivisione non debba essere intesa in senso stretto in quanto i protocolli, in base alla loro progettazione, possono avere caratteristiche riferibili a più di una categoria ed essere classificati secondo molteplici criteri.
Nello specifico, per quanto riguarda quelli basati su password, deboli e dunque insicuri, seppur ancora molto diffusi, sono stati trattati i seguenti: un protocollo "ingenuo" utilizzato nei terminali sin dagli anni settanta, il protocollo di Needham usato nel sistema Unix, il Password Autentication Protocol (PAP), il Challenge Handshake Protocol (CHAP), il One Time Password (OTP), il OTP S/KEY ed il Diffie-Helmann Key Exchange (DH-EXE); per agevolare l'illustrazione di quest'ultimo protocollo, si è ritenuto conveniente introdurre prima l'esposizione dei protocolli Diffie-Helmann (DH) Key Exchange e Encrypted Key Exchange (EKE), versione base del protocollo di Bellovin e Meritt.
Con riferimento ai protocolli challenge-response, che prevedono un'autenticazione forte, in cui una delle due parti sceglie ed invia una quantità tempo-variante, il challenge, e l'altra fornisce come risposta un valore che dipende sia dal challenge ricevuto che dal segreto associato alla propria identità, il response, sono stati illustrati i protocolli basati su algoritmi a chiave simmetrica (crittografia a chiave privata), su Message Authentication Code (MAC), su algoritmi a chiave asimmetrica (crittografia a chiave pubblica) e su un Key Distribution Center (KDC).
L'esame dei protocolli si è concluso con la trattazione di quelli a conoscenza zero (ZKP), la cui la sfida, da parte del soggetto che vuole autenticarsi, consiste nel provare il possesso della conoscenza di determinate informazioni senza rivelare l'informazione stessa o alcuna informazione aggiuntiva alla parte preposta alla verifica.
Considerata la complessità dell'argomento, prima dell'illustrazione dei protocolli di Fiat-Shamir e di Fiege-Fiat-Shamir, si è preferito introdurre gradualmente la tecnica impiegata nella progettazione dei protocolli ZKP, iniziando da un tentativo di costruzione di un banale schema ed evidenziandone le debolezze, per arrivare alla versione migliorata con l'uso di "commitment". Le idee a fondamento dei protocolli ZKP, basate su uno schema di challenge-response, sono state illustrate ricorrendo all'articolo "How to explain Zero-Knowledge Protocollos to Your Children" di Quisquater, Guillou e Berson in cui viene utilizzata la nota storia della caverna di Ali Babà.
Si è, quindi, evidenziato come per le dimostrazioni a conoscenza zero siano utilizzati i protocolli crittografici basati sui problemi NP illustrati nella sezione introduttiva cui non è possibile trovare una soluzione in tempi ragionevoli seppur sia dimostrabile polinomialmente la correttezza di una soluzione proposta; a riguardo sono stati forniti degli esempi di protocolli basati sul logaritmo discreto e sulla residuosità quadratica.
In tutti gli esempi proposti si è sottolineato come le dimostrazioni a conoscenza zero utilizzate permettano di dimostrare di aver trovato una soluzione effettivamente valida ed efficace senza rilevare alcuna informazione che possa permettere ad altri la strategia utilizzata. Infine, per quanto riguarda il protocollo di Fiege-Fiat-Shamir, si è analizzata sia la versione originale che quella revisionata, quest'ultima mirata a ridurre il cosiddetto problema del "ping-pong", fornendo, tra gli altri, un esempio interessante applicabile in uno scenario concreto di un utente si autentica in un ATM o un POS, tramite carta e PIN, per effettuare una delle operazioni consentite.

Lo svolgimento della parte pratica del progetto ha riguardato come caso di studio l'implementazione di una web application che utilizza un protocollo di di autenticazione "ingenuo" basato su password. Si è stabilito che gli scenari in cui avviene l'esecuzione del protocollo siano i seguenti; autenticazione di un utente, che si connette in HTTP tramite browser, ad una web application residente in un web server, realizzato mediante Apache-Tomcat, con il protocollo di autenticazione "ingenuo" basato su password (scenario 1); autenticazione di un utente, che si connette in HTTP tramite browser, ad una web application residente in un web server, realizzato mediante Apache configurato come reverse proxy per Tomcat, con il protocollo di autenticazione "ingenuo" basato su password (scenario 2); autenticazione di un utente, che si connette in HTTPS (HTTP over TLS) tramite browser, ad una web application residente in un web server, realizzato mediante Apache, configurato come reverse proxy per Tomcat, con l'utilizzo del SSL/TLS e di un certificato self-signed, con il protocollo di autenticazione "ingenuo" basato su password (scenario 3).
Nei tre scenari indicati, le cui debolezza erano già state evidenziate nella trattazione relativa ai protocolli basati su password, si è fornita la descrizione del protocollo "ingenuo" utilizzato dalla web application sviluppata, denominata *AuthWebApp*; a riguardo, in un'apposita sezione, sono stati indicati i dettagli dell'implementazione relativi alla tecnologia Java utilizzata, all'IDE Eclipse ed alla sua integrazione con Apache Tomcat, configurati in un ambiente Windows.
L'applicazione è stata progettata prevedendo nella home page una form di login tramite la quale differenti tipologie di utenza, in base ai ruoli assegnati, possono autenticarsi.
A scopo esemplificativo, sono stati configurati due utenti con ruoli distinti; in base a tali ruoli, è stata discriminata la visualizzazione del contenuto delle pagine protette dell'applicazione; l'organizzazione delle pagine pubbliche e private e la loro fruibilità, a seconda della correttezza dell'autenticazione effettuata dall'utente, dei suoi ruoli (autorizzazioni) e della sessione instaurata, è stata illustrata efficacemente tramite la site map di *AuthWebApp*; in tale diagramma sono stati evidenziati i pattern consentiti in base al verificarsi delle condizioni indicate; per ciascun pattern è stata descritta la pagina di riferimento, dettagliando le informazioni presentate all'utente ed evidenziandone la successiva utilità nell'esecuzione dei test di autovalutazione dei sistemi che interagiscono con il protocollo di autenticazione utilizzato dalla web application.

I dettagli dell'implementazione, oltre ad essere documentati nel Javadoc, sono stati illustrati in un'apposita sezione, partendo dalla descrizione dell'organizzazione delle classi ospitate nei packages utilizzati nel progetto; per ciascuna della classe sviluppate, con particolare riferimento alle servlet, sono stati evidenziati gli aspetti peculiari; inoltre sono stati descritti sinteticamente i compiti delle java server pages, già indicati nell'illustrazione delle servlet.

Per la realizzazione degli scenari esemplificativi proposti, è stato utilizzato l'ambiente Oracle Virtual Box nel quale sono state create le tre virtual machine necessarie per l'esecuzione dei test di autovalutazione dei sistemi che partecipano all'esecuzione del protocollo di autenticazione utilizzato da *AuthWebApp*; in ciascuna VM è stata installata la distribuzione Kali GNU/Linux scelta grazie alle sue peculiarità specifiche per la sicurezza informatica; inoltre sono stati installati e configurati i seguenti software nelle

VM di seguito indicate: JDK, Apache Tomcat, OpenSSL e Apache nella VM con il ruolo di server; OpenSSL e SSLsplit in quella con il ruolo di attaccante; nella VM client, dalla quale l'utente (vittima) effettua l'autenticazione, non sono state richieste installazioni aggiuntive in quanto si è utilizzato solamente il browser Firefox in dotazione al sistema. Sia nell'ambiente di virtualizzazione che nei sistemi delle VM sono state effettuate opportune configurazioni per consentirne l'interazione con il sistema host in sicurezza, al fine di evitare possibili disservizi dovuti all'esecuzione dei test di autovalutazione che hanno richiesto anche l'utilizzo di attacchi attivi.

L'autovalutazione dei sistemi, che interagiscono con *AuthWebApp* nell'esecuzione del protocollo di autenticazione basato su password, è stata basata sulla verifica della sicurezza dell'esecuzione del protocollo e, pertanto, oltre al protocollo di autenticazione utilizzato dalla web application, nella valutazione si è reso necessario prendere in considerazione i protocolli di comunicazione (HTTP e HTTP over TLS) utilizzati tra i partecipanti oltreché le loro configurazioni sistemistiche.
Gli scenari di riferimento in cui è avvenuta la valutazione sono stati i tre indicati nella definizione del caso di studio; in ciascun scenario, il metodo seguito è consistito nell'effettuare uno o più attacchi ai danni di sistemi coinvolti nell'esecuzione del protocollo di autenticazione al fine di individuare le credenziali utilizzate da un utente che accedeva alle pagine protette tramite il web form di *AuthWebApp* per poi autenticarsi al posto della vittima inconsapevole. Le tecniche di attacco individuate sono dipese dai protocolli di comunicazione utilizzati e dall'eventuale impiego di algoritmi crittografici. Chiaramente gli scenari proposti sono stati concepiti per evidenziare la debolezza del protocollo di autenticazione utilizzato dalla web application e mostrare che, nonostante le ottimizzazioni nelle configurazioni dei web server e l'utilizzo del protocollo SSL, le criticità permangono; dunque nell'ottica di un'adeguata valutazione della sicurezza dei sistemi che partecipano all'esecuzione del protocollo, si è tenuto conto del fatto che l'applicazione *AuthWebApp* è stata sviluppata a livello didattico, con limitazioni anche della sicurezza della memorizzazione delle credenziali, dell'ingenuità dello stesso protocollo di autenticazione e del tipo di interazione delle VM, realizzate in un ambiente virtuale ben differente da un contesto di produzione; in ogni caso, pur con tutte le limitazioni indicate, gli scenari proposti sono risultati significativi per effettuare una valutazione della sicurezza mirata ad evidenziare le possibili criticità a cui possono essere soggetti i sistemi in caso di configurazioni non adeguate e di utilizzo di applicazioni che richiedono protocolli di autenticazione deboli.

Grazie alle configurazioni realizzate in fase di implementazione della virtualizzazione dei sistemi, i differenti scenari, in cui sono stati effettuati i test, sono stati resi disponibili rapidamente con l'esecuzione di alcune modifiche apportate alla configurazione dei web server utilizzati; inoltre, fatta eccezione per la personalizzazione effettuata per il tool SSLsplit, essendo i software di sniffing e spoofing (Ethereal ed Ettercap) utilizzati per gli attacchi già disponibili nella distribuzione Kali, non sono state necessarie installazioni e configurazioni aggiuntive. Le istruzioni riportate in dettaglio nell'apposita sezione, sono state fornite tenendo conto del fatto che i test fossero eseguiti nell'ordine stabilito per agevolare le modifiche delle configurazione dei web server Apache-Tomcat ed Apache; in ogni caso, gli attacchi illustrati possono essere riprodotti in uno qualunque degli

scenari proposti a prescindere dall'ordine seguito; ovviamente la migliore configurazione dei due web server, che più si adatta ad una situazione reale, è quella indicata nel terzo scenario, seppur si faccia uso di un certificato self-signed decisamente sconsigliato in un ambiente di produzione. Per ciascun scenario indicato, sono stati riportati i test, ovvero gli attacchi effettuati, l'analisi delle origini delle vulnerabilità riscontrate, l'assessment e le possibili attività per la remediation; per quanto riguarda le classificazioni delle vulnerabilità utilizzate nell'assessment si è tenuto conto della conformità dei sistemi alle indicazioni dell'AgID. Considerate le criticità riscontrate si è ritenuto opportuno dedicare alle strategie di remediation una sezione dedicata nella quale sono approfonditi l'implementazione del SSL/TLS e le misure più idonee da adattare in tutti gli scenari indicati; a riguardo, è stato valutato che tali misure possono essere le seguenti: seguire le "best practice" nella progettazione del protocollo di autenticazione, ricorrendo ad una revisione dello stesso che preveda un ulteriore canale per una verifica tramite challenge-respose con Time-based One-Time Password, utilizzare esclusivamente il protocollo HTTPS, utilizzare una CA affidabile per la validazione del dominio e per la firma del certificato associato, effettuare una periodica analisi del protocollo di autenticazione e del software utilizzato per implementarlo ed adottare una politica di sicurezza, adeguata agli scopi prefissi, a protezione del protocollo estesa a tutti i sistemi interessati.

Da quanto realizzato emerge, quindi, l'importanza di un'adeguata conoscenza dei protocolli di autenticazione e del loro scenario di applicazione per effettuare la scelta del protocollo più appropriata per un sistema fin dalla fase di progettazione, al fine del raggiungimento degli obiettivi di sicurezza prefissati.

4.1 Lavori futuri

Pur nella sua lunghezza e complessità, nel progetto sono stati esaminati solamente i protocolli di autenticazione più comuni alla base dei moderni protocolli di sicurezza. Come sottolineato nell'approfondimento della remediation relativa agli scenari illustrati, i molteplici protocolli adottati per l'autenticazione, anche a seguito delle continue scoperte di debolezze e vulnerabilità che ne minacciano la sicurezza, sono in continua evoluzione e risulta davvero un'impresa ardua descriverli in un unico lavoro.

Nel progetto non sono stati trattati i numerosi framework il cui utilizzo è ormai imposto dagli standard, nel rispetto dalla normativa vigente in materia di privacy e sicurezza informatica; si pensi, ad esempio, all'*Extensible Authentication Protocol* (EAP), framework che offre l'estensibilità ai metodi di autenticazione per le tecnologie di accesso alla rete protette di uso comune, quali l'accesso wireless e cablato basato su IEEE 802.1X e le le reti private virtuali (VPN); tra i metodi supportati, il *Tunnelled Transport Layer Security* (TTLS) è il più diffuso in quanto utilizza una chiave dinamica, consente la mutua autenticazione, non richiede user e password, permette la creazione di un tunnel SSL/TLS sicuro e supporta i tradizionali metodi di autenticazione esaminati PAP, CHAP, MS-CHAP, MS-CHAP v2.

Non sono stati trattati, inoltre, i protocolli utilizzati per le autenticazioni nei sistemi federati quali il *Security Assertion Markup Language* (SAML), uno standard aperto che

consente a un Identity Provider (IdP) di autenticare gli utenti e di passare un token di autenticazione ad un'altra applicazione nota come Service Provider (SP), l'*OpenID Connect*, un layer di identità sicuro basato su JSON/REST, che si posiziona sopra al protocollo OAuth 2.0, flessibile e interoperabile in modo che le identità digitali possono essere facilmente utilizzate su servizi desktop e mobile, l'*Active Directory Federation Services* (AD FS), la soluzione di Microsoft basata su sistemi operativi Windows Server che fornisce un meccanismo di Single Sign-On (SSO) tra un ambiente di dominio e le web application, utilizzabile anche tra organizzazioni che hanno domini unici o multipli.

Come anticipato nella sezione relativa all'implementazione della virtualizzazione, applicata ai sistemi che interagiscono con la web application nell'esecuzione del protocollo di autenticazione implementato, la progettazione della transizione dall'architettura monolitica del progetto ad una a microservizi potrebbe rappresentare un'ulteriore attività per approfondire le tecniche di progettazione Domain Driven Design di microservizi magari utilizzando un'architettura "cloud-native".

Ulteriori spunti ad integrazione della trattazione dei protocolli di autenticazione provengono dall'ambito lavorativo; in particolare, si evidenzia l'esigenza di approfondire le problematiche connesse all'aggiornamento dello Shibboleth IdP istituzionale, alla luce della nuova soluzione di Identity and Access Management (IAM) adottata in cloud e dell'integrazione con i servizi federati, e quelle relative agli aggiornamenti dell'IdP e del Resource Provider (RP) che partecipano al servizio federato Eduroam (Education Roaming), grazie al quale gli utenti in mobilità possono autenticarsi in modo semplice e sicuro alla rete wireless di un'organizzazione ospite, utilizzando le stesse credenziali fornite dall'istituzione di appartenenza.

4.2 Cosa è stato appreso

Lo sviluppo del progetto è stato utile per approfondire aspetti delle sicurezza computazionale, già analizzati in ambito matematico, considerato il percorso formativo di provenienza; nello specifico si è avuto modo di evidenziare come la sicurezza delle transazioni nei protocolli di autenticazione sia fortemente dipendente dalla risoluzione di problemi fondamentali usati in crittografia ed il successo di un attacco ai danni di un crittosistema sia valutato in termini probabilistici.

La trattazione dei protocolli esaminati, grazie alle numerose fonti consultate, ha contribuito a migliorare la visione complessiva delle problematiche di sicurezza a cui sono soggetti i sistemi che partecipano all'autenticazione; tale visione, consolidata dall'analisi dei punti di forza e delle criticità dei protocolli, evidenziate anche con alcuni approfondimenti sugli attacchi, ha agevolato l'utilizzo di nuovi approcci metodologici alla cyber security.

L'implementazione pratica del progetto ha permesso l'approfondimento della conoscenza della tecnologia Java grazie allo sviluppo della web application realizzata mediante Eclipse; a riguardo si è avuto modo di analizzare l'integrazione dell'IDE con Apache Tomcat, utilizzato come web container per il deploy delle servlet e delle jsp realizzate.

Per la realizzazione degli scenari esemplificativi proposti, nei quali è stato eseguito il protocollo di autenticazione utilizzato dalla web application sviluppata, si è reso

necessario uno studio dell'ambiente Oracle Virtual Box mirato all'ottenimento della configurazione realizzata, grazie alla quale gli attacchi condotti, anche attivi, con i tool di sniffing e spoofing, ai danni delle virtual machine utilizzate, non arrecano potenziali problemi di sicurezza al sistema host; l'esigenza di utilizzare tali tool ha portato allo scelta della distribuzione Kali GNU/Linux, ritenuta idonea agli obiettivi prefissati, grazie alle sue peculiarità specifiche per l'informatica forense, la sicurezza informatica ed, in particolare, il penetration testing; a tale proposito ne sono state approfondite le principali funzionalità e, nell'ambito dell'implementazione degli scenari utilizzati, i seguenti aspetti tecnici: le modalità di installazione e configurazione della versione Java utilizzata, la stessa dell'host Windows in cui è stato sviluppato il progetto, tenuto conto che la versione in dotazione in Kali non assicurava il corretto deploy in Apache Tomcat del file war del progetto esportato da Eclipse, le configurazioni di avvio automatico dei servizi di Apache-Tomcat, considerata la versione di Java utilizzata e quelle di sicurezza dello stesso web server, e l'utilizzo avanzato dei tool di attacco illustrati.
Nella stessa distribuzione, sono state analizzate anche le configurazioni di sicurezza di Apache-Tomcat e di Apache; per quanto riguarda quest'ultimo web server, si è approfondito l'utilizzo dei moduli per erogare il servizio di reverse proxy ed utilizzare il protocollo https con l'utilizzo di certificati generati tramite il package openSSL.

Infine, come illustrato nell'apposita sezione, a seguito dei test di sicurezza condotti negli scenari indicati, dai risultati dell'analisi e dell'assessment realizzati, si è evidenziata l'esigenza di approfondire l'implementazione del SSL/TLS e di ulteriori misure da adottare per un'efficace strategia di remediation alle debolezze del protocollo di autenticazione, utilizzato dalla web application sviluppata, da applicare in tutti gli scenari proposti nel caso di studio.

Bibliografia

Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Springer Publishing Company, Inc., 1st edition, 2010. ISBN 978-3-642-07716-6.

J. Bullock and Jeft T. Parker. *Wireshark e Metasploit. Dall'analisi di rete alle tecniche di attacco e difesa*. Apogeo, 1st edition, 2019. ISBN 978-88-503-3442-1.

M. Caiazza, G. Laccetti, and G. Schmid. Il metodo induttivo e la verifica di un protocollo crittografico. 2005. URL `https://intranet.icar.cnr.it/wp-content/uploads/2016/11/tr192005.pdf`.

L. Dong and K. Chen. *Cryptographic Protocol: Security Analysis Based on Trusted Freshness*. Springer Berlin Heidelberg, 2012. ISBN 9783642240720. URL `https://books.google.it/books?id=LclKXwAACAAJ`.

Ariel Eizenberg. PPP security features. 2003. URL `http://www.cs.huji.ac.il/~sans/students_lectures/PPP-Security.ppt`.

Simon Garfinkel and Gene Spafford. *Pratical UNIX & Internet Security*. O'Reilly, 1996. ISBN 1-56592-148-8. URL `https://docstore.mik.ua/orelly/networking/puis/`.

Joseph Kizza. Feige-Fiat-Shamir ZKP Scheme Revisited. *International Journal of Computing and ICT Research*, 4, 07 2010.

Neal Koblitz. *A course in Number Theory and Cryptography*. SPringer-Verlag, 2. ed. edition, 1994. ISBN 0-387-94293-9.

Wenbo Mao. *Modern Cryptography: Theory and Practice*. Prentice Hall Professional Technical Reference, 2003. ISBN 978-0-13-066943-8.

Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography*. CRC Press, 2001. ISBN 0-8493-8523-7. URL `https://cacr.uwaterloo.ca/hac/`.

Samuel Michael. S/Key Dungeon Attack. 2011. URL `https://www.miknet.net/security/skey-dungeon-attack/`.

CIS-Sapienza Research Center of Cyber Intelligence and Information Security Sapienza Università di Roma. *Framework Nazionale per la Cybersecurity e la Data Protection*. 2019. URL `https://www.cybersecurityframework.it`.

Raphael Pass and Abhi Shelat. *A course in Cryprography*. 3. ed. edition, 2010. URL `https://www.cs.cornell.edu/courses/cs4830/2010fa/lecnotes.pdf`.

Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, and Soazig Guillou. How to explain zero-knowledge protocols to your children. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*. Springer New York, 1990. ISBN 978-0-387-34805-6.

I. Ristić. *OpenSSL Cookbook. The Definitive Guide to the Most UseFul Command Line Features*. Feisty Duck Ltd, 3ed online edition, 2021. URL `https://www.feistyduck.com/library/openssl-cookbook/online/`.

Berry Schoenmakers. *Lecture Notes Cryptographic Protocols*. 2021. URL `https://www.win.tue.nl/~berry/2DMI00/`.

S. Tanenbaum and Maarten Van Steen. *Sistemi distribuiti*. Pearson, Prentice-Hall, 2. ed. edition, 2007. ISBN 978-88-7192-366-6.

Wade Trappe and Lawrence C. Washington. *Crittografia: con elementi della teoria dei codici*. Pearson, Prentice-Hall, 2. ed. edition, 2009. ISBN 8871925602.

Daniele Venturi. *Crittografia nel Paese delle Meraviglie*. Springer, 2012. ISBN 978-88-470-2480-9.

Elenco delle figure

Elenco dei diagrammi di sequenza dei protocolli

Sommario