Appunti di Internet Security

*Appunti di **Giuseppe Pitruzzella** - Corso di Internet Security @ DMI, UniCT*

Teoria 📚

L'importanza della sicurezza informatica

*Appunti di **Giuseppe Pitruzzella** - Corso di Internet Security @ DMI, UniCt*

La sicurezza informatica è l'informatica. Non esiste un campo dell'informatica dove non è presente la sicurezza.
Per questo motivo, creare un servizio implica garantire la sicurezza per lo stesso. Un esempio è possibile a partire da tutti gli applicativi che installiamo giornalmente, sui cui poniamo notevole fiducia, una fiducia necessaria a credere che nessuno di essi sia un applicativo malevolo, come per esempio, un trojan.

Trojan

Un esempio di trojan (in particolare una certa famiglia di Trojan) è descritto dal codice seguente e nasce a partire dalla ricezione di un file malevolo (*matrice socio-tecnologica*) che ipotizziamo sia uno script chiamato `ls.sh` (che eseguirà le istruzioni al dì sotto e viene eseguito dalla vittima stessa).

Sappiamo che ogni file ha un proprietario ed un gruppo e, normalmente, ogni programma viene eseguito secondo i permessi che possiede l'utente che lo lancia (il comando `whoami` ritorna, come intuibile, chi siamo).

Attivare il permesso `setuid` (il bit `s`) indica al SO di *eseguire il programma con i permessi del proprietario*, in aggiunta ai permessi dell'utente che lancia quest'ultimo. Intuiamo, quindi, che assegnare il bit `s` ad un file che è posseduto da `root` implica il programma sarà aperto con i permessi da `root`.

```
cp /bin/sh /tmp/.xxsh
chmod u+s,o+x /tmp/.xxsh
rm ./ls.sh
ls
```

Del codice soprastante è necessario notare le seguenti nozioni:

- Il primo comando copia la shell `/bin/sh` in una cartella temporanea, ossia `tmp`, notoriamente utilizzata dal sistema per funzioni analoghe (per es. buffer); Il motivo per cui quest'ultima viene copiata all'interno di questa cartella è un euristica *socio-tecnologica*. Inoltre il file copiato è preceduto da un punto, il che indica che sarà visibile solo se il comando ls è accompagnato dal flag `-a` (questa è un euristica socio-tecnologica). Si noti, inoltre, che i permessi del file copiato dipendono dalla distro utilizzata.
- Il secondo comando cambia i permessi in modo conveniente. Infatti, il file, poiché copiato dall'utente, tendenzialmente avrà i suoi stessi permessi. Potrebbe anche accadere che il file copiato abbia gli stessi permessi del proprietario della cartella ricevente, quindi tmp, ossia root. Il tutto dipende dalla policy per il comando cp. Quindi, a partire da una policy, o meglio una scelta progettuale, permetto o meno una famiglia di Trojan.

- Il terzo comando elimina il file ls (eseguibile) all'interno della cartella corrente. Si presuppone, infatti, che vi siano due ls, il vero ls di sistema ed un falso ls, ovvero `ls.sh`.
- Il quarto comando richiama il vero `ls` di sistema, il che potrebbe far intendere alla vittima che ciò che ha eseguito è stato un semplice `ls`.

[*NB*] Ogni comando è eseguito dall'utente vittima, che esegue a sua volta il file `ls.sh` pensando di eseguire il comanda di sistema `ls`.

Il risultato è una shell nel sistema con i permessi di root. In altre parole, l'attaccante è riuscito ad essere root nel sistema della vittima bypassando l'autenticazione.
Si noti esista la possibilità di creare delle varianti a partire da questo attacco (Trojan). Una di queste potrebbe essere il semplice bypass dell'autenticazione, attraverso cui si potrebbe creare un nuovo utente (l'attaccante) root, oppure ancora inviare i dati della vittima ad un suo server a partire da un client ftp.
Il Trojan adesso è concettualmente obsoleto poichè (i) il trojan scaricato da parte della vittima non sarebbe di per sè eseguibile e (ii) nessuno adesso eseguirebbe un file con "`./nome`", ciò nonostante alcuni SO hanno fixato il problema solo pochi anni fa.
Il motivo per cui un tempo il Trojan aveva senso è fortemente legato alla cultura del tempo.

> [$Approfondimento$] La **Open Source INTelligence**, acronimo **OSINT** (in italiano: "Intelligence su fonti aperte"), è quella disciplina dell'intelligence che si occupa della ricerca, raccolta ed analisi di dati e di notizie d'interesse pubblico tratte da fonti aperte. Può essere intesa come la prima fase di un attacco.

> [$Esame$] Cos'è un trojan? Un esempio di trojan?

> [$Esame$] Spiegare l'attacco all'interno di questo primo trojan.

Password

Il concetto di **password** è anch'esso estremamente importante all'interno della sicurezza informatica e con esso anche il concetto di "*scelta della password*", il quale deve essere difficile da indovinare ed allo stesso tempo facile da ricordare.
Sappiamo bene, infatti, che scegliere una parola come "ciao" non è desiderabile per la sicurezza del nostro account poichè potrebbe essere facilmente indovinata da un **attacco dizionario**, un attacco che prevede di provare tutte le password possibili a partire da un certo dizionario di parole.
Oggigiorno sappiamo che ciò non accade spesso, ed il motivo riguarda le regole imposte da un sito web rispetto la scelta della password, la quale *deve* rispettare certi **criteri**.
Purtroppo queste regole non esistono da sempre, infatti sono state introdotte dal **NIST** solo nel 2004, anno in cui vengono pubblicate le regole per la robostezza di una password.

> [*Approfondimento*] Il **NIST**, a partire dal 2004, afferma che una password è robusta non è stata già diffuse a seguito di altri attacchi, non sia una parola del dizionario, non vi siano dei caratteri sequenziali o ripetuti (come "1234" e "aaaa"), non siano delle parole derivate dal nome del servizio (per esempio, Facebook dovrebbe impedire agli utenti di usare password come “Facebook” o “Facebook01” o “Facebook-Giuseppe”) ed, infine, che non siano parole che contengono il nome, il cognome o l’user-id dell’utente.

Le password però possono essere facilmente rotte se il **tempo** lo permette, motivo per cui il *mist* (l'ente che ha disposto i criteri rispetto una password) afferma che ogni password deve essere cambiata ogni 6 mesi. A questo punto l'unico problema potrebbe essere relativo ad cambiamento futile rispetto la nuova password per l'utente.
Un altro problema rispetto le password è il **riutilizzo** di quest'ultima.

> [*Approfondimento*] **Heartbleed** è un bug di sicurezza nella libreria di OpenSSL, un’implementazione open-source ampiamente usata del protocollo TLS. Heartbleed potrebbe essere sfruttato indipendentemente dal fatto che l’istanza OpenSSL stia girando come server o client TLS. È il risultato di una validazione input impropria (data dalla mancanza di controllo dei limiti nell’implementazione dell’estensione *heartbeat* del protocollo TLS.

> Cosa si intende con "*usabilità*"? Il professore spiega questo secondo una vecchia piattaforma che permetteva di invare un messaggio ad un numero telefonico senza alcuna autenticazione.

Attacchi

*Appunti di **Giuseppe Pitruzzella** - Corso di Internet Security @ DMI, UniCt*

Il concetto di sicurezza può essere definito come la prevenzione degli attacchi pur mantenendo usabile il sistema, ovvero la piena funzionalità del sistema.

Attacco reale: furto del dispositivo

Iniziamo a studiare la tassonomia a partire da una prima tipologia di attacco: l'**attacco fisico**. Ogni attacco di questa tipologia prevede un **attacco fondamentale**.
Immaginiamo, quindi, che un attaccante si trovi in mano un dispositivo rubato o penetretion tester che deve verificare la sicurezza di un dispositivo

Lo scenario ipotetico su cui si basano gli attacchi fisici che studiamo si basano su un attaccante
Gli attacchi che studiamo all'interno di questo paragrafo si basano su uno dei seguenti due scenari:
(i) un *penetration-tester* che deve verificare la sicurezza di un dispositivo o (ii) un *attaccante* che vuole *bypassare* la sicurezza di un dispositivo che è stato rubato, due scenari che si basano sulla stessa *cyber-kill-chain*.

> [$Approfondimento$] Una **cyber-kill-chain**, un concetto pubblicato da *Lockheed Martin*, la principale industria americana nel settore della difesa, è una procedura che descrive la struttura dei passi che genera l'intrusione/attacco. L'analisi della *kill chain* permette di capire come un avversario per raggiungere il suo obiettivo debba riuscire a progredire attraverso tutta la catena, mettendo bene in evidenza quali azioni di mitigazione sono efficaci per interrompere la *kill chain* stessa.

Una *cyber-kill-chain* semplificata che studiamo all'interno del corso è la seguente:

1. **Accedere al sistema** (*attacco fondamentale*).
   Per accedere al sistema è necessario violare l'autenticazione, unica **contromisura** rispetto questo attacco fondamentale, la quale può essere formata da autenticazione biometrica e/o password alfanumeriche.
   Purtroppo, sappiamo bene che questa contromisura è facilmente violabile a partire dai metodi studiati nei $mini - challenge$. Abbiamo visto, infatti, che esiste un modo per violare sia sistemi Linux che sistemi Windows (entrambi prevedono di ottenere una shell con i permessi di amministratore).
   1.1 **Sfruttare le funzionalità del sistema**, si pensi per esempio all'utilizzo della potenza computazionale della cpu di una macchina o all'utilizzo dello spazio del suo disco. In questo caso, parliamo di **cyber-security**.
   Una **contromisura** rispetto questa attività è l'autenticazione rispetto la specifica funzionalità (per esempio l'autenticazione al servizio o app). Purtroppo, questa contromisura spesso fallisce in scenari in cui le password vengono salvate dal browser o app.

1.2 **Aquisizione di dati sensibili** all'interno della macchina. In questo caso, parliamo di **privacy** o *data-protection*, ovvero la *cyber-security* istanziata al dato. Una **contromisura** rispetto questa attività è la crittografia dei dati. Purtroppo, spesso vi è molta riluttanza verso la codifica o cifratura di un unità.
[*NB*] In generale la privacy include anche degli aspetti funzionali di cui gode il proprietario del dato.

Attacco reale: pirateria digitale

Distinguiamo diversi tipi di pirateria digitale, ovvero:

- **Furto di proprietà intellettuale**, per es. rispetto un film, il quale viene scaricato illegalmente. Chiaramente questa era una attività popolare durante i primi anni duemila. Con il passare degli anni, infatti, il furto è sempre meno effettuato poichè è cambiato il modello di business legato alle proprietà intellettuali. Quest'ultimo modello è basato sull'acquisto ad prezzo inferiore di servizi che prevedono di poter visualizzare quest'ultime (vedi per es. Netflix).

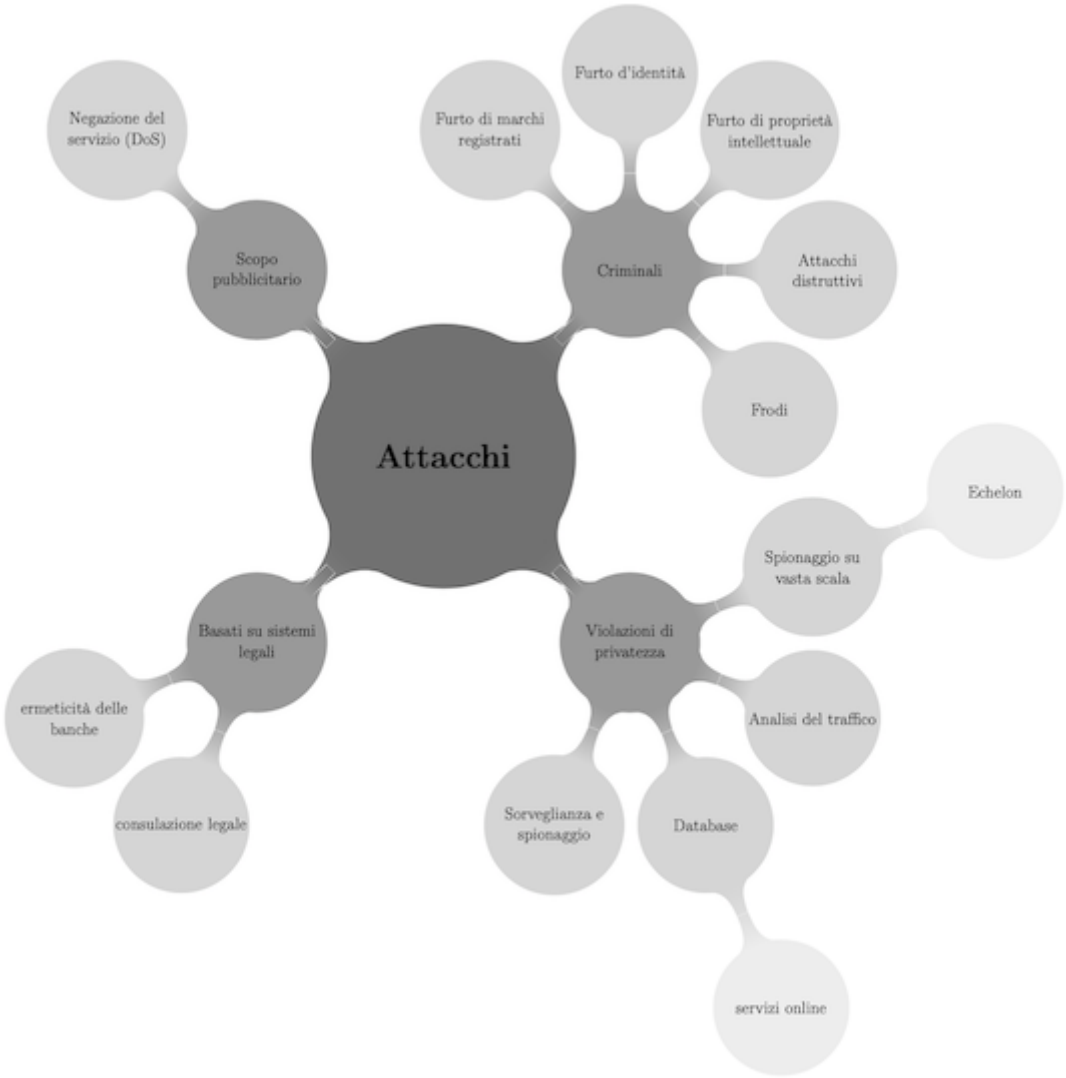
  > [$Approfondimento$] Il termine **watermark**, si riferisce all'inclusione di informazioni all'interno di un file multimediale o di altro genere, che può essere successivamente rilevato o estratto per trarre informazioni sulla sua origine e provenienza. In altre parole, il watermarking dimostra la proprietà di un file multimediale e vuole essere una contromisura verso l'abuso del prodotto. Quest'ultimo potrebbe non essere visibile e non è possibile rimuoverlo. Una tecnica banale di watermarking è l'encoding nei bit meno significativi dei pixel.

- **Furto di identità**, per es. effettuare l'autenticazione come se fossimo il proprietario della macchina. Ancora una volta vale l'autenticazione come unica contromisura.
- **Furto di marchio**.

Tassonomie di attacchi

Notiamo adesso la tassonomia degli attacchi, dagli attacchi di attacchi criminali fino agli attacchi per scopi pubblicitari. Il tutto è rappresentato graficamente attraverso l'immagine

sottostante.



Proprietà di sicurezza

*Appunti di **Giuseppe Pitruzzella** - Corso di Internet Security @ DMI, UniCt*

Proprietà di livello 1

Segretezza o confidenzialità

Prima proprietà di tre che formano la base della piramide rappresentante le proprietà di sicurezza è la confidenzialità. Formalmente la **segretezza** di un oggetto x implica che x non sia noto a chi non è autorizzato a conoscere x.
Parlare di segretezza presuppone che vi sia una **policy**, attraverso cui viene definito chi può essere a conoscienza di un dato segreto. Se chiunque altro (al dì fuori della policy) venisse a conoscenza di quest'ultimo segreto, allora vi sarebbe una violazione della sicurezza.

Autenticazione

Definiamo l'**autenticazione** come la conferma dell'identità di un entità (e.g. un intelocutore come un umano, un processo, un ip), quindi la conferma che egli sia esattamente chi afferma di essere.

Integrità

L'ultimo elemento della tripla che forma la base della piramide è l'**integrità**. Possiamo definire l'integrità (e.g. del dato o del sistema) come la conferma che l'informazione non sia modificata da entità non autorizzate.
Esempi di **misura** per il controllo dell'integrità sono:

- Il **checksum**, una sequenza di bit che, associata al pacchetto trasmesso, viene utilizzata per verificare l'integrità di un dato. L'unico modo per verificare l'integrità di un file scaricato è il confronto con il checksum mostrato dal sito certificato. Tuttavia sarebbe possibile *inviare* (e.g. ad un collega) un file malevolo insieme ad un nuovo checksum opportunamente calcolato relativo a quest'ultimo, il che comunque differirebbe dal checksum mostrato dal sito certificato.
- La **firma elettronica**, un insieme di dati utilizzati come metodo di identificazione informatica. La firma digitale è la tecnologia con cui possono essere effettivamente soddisfatti tutti i requisiti richiesti per dare validità legale ad un documento elettronico firmato digitalmente; garantisce i servizi di integrità, autenticazione e non-ripudio. In particolare le proprietà che deve avere una firma digitale per essere ritenuta valida sono:
    - *Autenticità della firma*: la firma deve assicurare il destinatario che il mittente ha deliberatamente sottoscritto il contenuto del documento.

- *Non falsificabilità*: la firma è la prova che solo il firmatario e nessun altro ha apposto la firma sul documento.
- *Non riusabilità*: la firma fa parte integrante del documento e non deve essere utilizzabile su un altro documento.
- *Non alterabilità*: una volta firmato, il documento non deve poter essere alterato.
- *Non contestabilità*: il firmatario non può rinnegare la paternità dei documenti firmati; la firma attesta la volontà del firmatario di sottoscrivere quanto contenuto nel documento

> E' importante notare che la firma elettronica (o digitale) differisce da un checksum poichè la prima, differentemente dalla seconda, ha un robustezza di natura crittografica. La firma digitale, quindi, conferisce una robustezza crittografica al processo di confronto, il che implica che non è possibile l'esempio relativo al file malevolo con il checksum.

Privatezza (Privacy)

Passiamo adesso alla **privacy**, prima proprietà al dì sopra della base, quindi appartente al livello due. Definiamo la privacy come il *diritto* di un entità di rilasciare o meno i propri dati personali ad altre entità.
Una **misura** per garantire la privacy è dato dalle policy.

> Si noti che vi notevole differenza tra la privacy e la confidenzialità (i.e. segretezza), la quale è una differenza logica ed architetturale. Quest'ultima (i.e. privacy), differentemente dalla confidenzialità, implica un diritto. Potremmo al più definire la privacy come il diritto istanziato alla confidenzialità.

Anonimato

L'**anonimato** è il *diritto* dell'iniziatore di una transazione di rilasciare o meno la propria identità ad alte entità.
Una **misura** per garantire l'anonimato è lo **pseudo-anonimato**, per cui viene effettuata la scissione delle proprie *PII* (acronimo di *personal identifier information*, ossia una stringa (o tupla informativa) che identifica univocamente la persona fisica) effettuando la generalizzazione di questìultime.
Poiché l'anonimato può essere studiato a più livelli architetturali, esistono misure relative a specifici livelli:

- A livello applicativo, attraverso gli *anonymous-web-proxy* che svolgono il ruolo di *man-in-the-middle*;
- A livello di rete, con *TOR*.
  Si noti che la navigazione anonima all'interno di un browser non rende anonimi sulla

rete ma semplicemente rende anonimi nei confronti della macchina su cui si opera poichè non memorizza alcun dato relativo alla sessione.

Proprietà di livello 2

Non-ripudio

La proprietà di non-ripudio indica che l'entità non possa negare la propria partecipazione ad una transazione con uno specifico ruolo. Una **misura** è fornita dalla **PEC**, un sistema che gode della proprietà di non ripudiabilità, il che rende impossibile negare l'invio di una PEC provienente dalla nostra casella.
E' importante non confondere la proprietà di non-ripudio con la proprietà di autenticazione, le quali presentano delle differenze. Per esempio, se Mario Rossi si identifica come Mario Rossi allora esiste un evidenza (i.e. conferma) della mia identità poichè altrimenti l'autenticazione non sarebbe possibile. D'altra parte, quest'ultima autenticazione (differentemente dal non-ripudio) non è *proponibile*, *verificabile* e *dimostrabile* all'esterno, come in egual modo essere il testimone di un crimine non implica incastrare il colpevole; il tutto poichè un avvocato potraebbe smontare la testimonianza di quest'ultimo (a meno della presenza di un elemento di non ripudiabilità).

Relazioni logiche fra autenticazione, anonimato e non-ripudio

1. L'autenticazione è l'opposto dell'anonimato;
2. L'anonimato implica che non vi sia autenticazione;
    1. L'autenticazione coincide con l'opposto dell'anonimato;
3. Autenticazione non implica il non-ripudio;
4. L'esistenza del non-ripudio implica l'esistenza dell'autenticazione;
5. L'anonimato implica l'inesistenza del non-ripudio;
6. Il non-ripudio implica l'inesistenza dell'anonimato.

	Autenticazione	**Anonimato**	**Non ripudio**
Autenticazione	/	Opposto	$\nrightarrow \exists$
Anonimato	$\rightarrow \nexists$	/	$\rightarrow \nexists$
Non ripudio	$\rightarrow \exists$	$\rightarrow \nexists$	/

Il problema dell'esame: il protocollo WATA

Il protocollo WATA si pone l'obiettivo di definire una modalità *sicura* per svolgere un esame. WATA consiste in un **esaminatore**, un **invigilator** ed un **candidato**. L'invigilator e l'esaminatore possono essere la stessa persona; infatti, dal punto di vista della sicurezza

non cambia nulla (d'altra parte, l'esaminatore potrebbe comprarsi l'invigilator).
All'interno di WATA possiamo distinguere 4 diverse **fasi**:

1. **Preparation**.
   Esiste un database di domande dal quale viene estratto randomicamente una tupla di domande; esiste un alfabeto di lunghezza n dal quale viene estratto a random un ID, il quale sarà l'ID del test (trasformato in un bar-code). Esiste, inoltre, un database di *mark* (i.e. voti) inizializzato con gli ID dei test (lo scopo finale è inserire un voto affianco all'ID del test). Stampiamo, quindi, l'ID del test ed il modulo di autenticazione, quindi il test formato dal suo ID, le domande ed il form per le risposte. L'esaminatore firma preventivamente i test nel suo ufficio (firma grafometrica) e da tutto il materiale all' invigilator.

2. **Testing**.
   La fase di testing è la fase in cui lo studente svolge il test.
   Il test passa alle mani del candidato randomicamente. Il candidato fa l'operazione di riempire il modulo di autenticazione col test firmato costruendo un test autenticatore riempito. Per farsi autenticare fornisce all'invigilator il proprio documento di identità e quello che ha appena prodotto, nascondendo gli elementi identificativi. Così l'invigilator controlla la validità del documento, ovvero che l'anagrafica stia nell'elenco dei registrati e che l'anagrafica sul documento corrisponda con quella letta sul test preparato prima. Dopo che il test è stato corretto il candidato riempie il modulo di quel test creando un test riempito. A questo punto taglia il test riempito producendo un token e il a_test (*anonymaze test*). Il candidato da randomicamente il a_test all'invigilator.

3. **Marking**
   L'invigilator da il a_test all'esaminatore e quest'ultimo corregge tutti i test ed inserisce i voti nel database dei voti tramite il bar-code. In questo modo l'esamitore non saprà mai a chi appartiene il test.

4. **Notification**
   Lo studente da il token all'esaminatore per poter avere il proprio voto attraverso la scansione che andava a leggere l'ID del testo nel database delle domande evidenziando il voto. L'esaminatore, dopo aver visto il voto, lo registra nel database dello storico della materia con quel ID.

Disponibilità

La proprietà di **disponibilità** si basa sul concetto che il sistema sia *operante* e *funzionante* in ogni momento. Chiaramente un sistema può essere un sito, un servizio, etc. Qualcosa che può compremettere la disponibilità è il concetto di **DoS** (Denial of Service), un attacco subdolo e malevolo rispetto la negazione del servizio, fondamentale poichè nessun sistema che non funzioni è più interessante. Quest'ultimo si basa sul sovraccarico del load balancer presente all'interno di un server. Un attacco DoS distribuito su più è detto **DDoS**, ovvero

Distributed Denial of Service.
Una misura rispetto un attacco di questo tipo si basa sul filtrare le richieste, chiaramente con un impatto sulle prestazioni.

Proprietà di livello 3

Controllo di accesso

Arriviamo adesso al livello tre della piramide mostrata da Schneier all'interno del suo libro ed iniziamo a trattare il *controllo*, definito più nello specifico dal professore come: **controllo di accesso**. Il controllo di accesso, che è sinonimo di *autorizzazione*, viene definita nel modo seguente: *ciascun utente abbia accesso a tutte e sole le risorse o i servizi per i quali è autorizzato*.
Una buona autorizzazione si basa su dei profili autorizzativi ben definiti delle classi di sistemi e servizi che vengono associate ad un utenza (e.g. loggarsi come `jax` prevede certi *profili autorizzativi*).

Alcune misure per il controllo di accesso sono (i) l'autenticazione dell'utente, attraverso cui ci assicuriamo chi e quali operazioni possa svolgere un generico utente (e.g. il professore quando entra in aula viene autenticato, motivo per cui sappiamo che egli abbia un certo set di autorizzazioni, che includono sedersi alla cattedra e spiegare la lezione), (ii) le politiche di sicurezza (in inglese, "*policy*") e (iii) l'implementazione delle policy stesse, per es. le *ACL* in Linux. Relativamente alle politiche di sicurezza, si noti che un cyber-informatico non può lavorare senza una buona compresione di una determinata policy. Un attacco, infatti, è tale se un attività che sovverte un policy. Si pensi, per es. che fino a poco tempo fa era possibile (all'interno di Ubuntu) per $utente1$ eseguire il comando `ls` su `home/utente2`, il che è permesso dalle policy di Ubuntu stesso, motivo per cui non possiamo definire quest'ultimo un attacco.

Un esempio di politica di sicurezza

Una policy giocattolo è la seguente:

1. Un utente ha il permesso di leggere un qualsiasi file pubblico;
2. Un utente ha il permesso di scrivere solo sui file pubblici di sua proprietà;
3. Un utente ha il divieto di sostituire un file con una sua versione più obsoleta;
4. Un utente ha l'obbligo di cambiare la propria password quando questa scade;
5. Un utente segreto ha il permesso di leggere su un qualunque file non pubblico;
6. Un utente segreto ha il permesso di scrivere su un qualunque file non pubblico;
7. Un amministratore ha il permesso di sostituire un qualunque file con una sua versione più obsoleta;
8. Un utente che non cambia la sua password scaduta (i.e. *negligente*) ha il divieto di compiere qualunque operazione;
9. Un utente che non cambia la sua password scaduta (i.e. *negligente*) non ha discrezione di cambiarla;

Notiamo a partire da questa policy giocattolo i seguenti **elementi** (*fondamentali*):

- **Ruoli**.
  I ruoli presenti all'interno di questa policy giocattolo sono i seguenti: utenti, utenti segreti, amministratori e negligenti.
  E' importante notare che i ruoli, in maniera *connaturata*, presentano delle intersezioni.
  All'interno della policy giocattolo notiamo le seguenti intersezioni:
  $Utenti \subseteq Utenti\ segreti \subseteq Amministratori$. Inoltre, sempre secondo questa policy, esistono degli utenti *negligenti* uguale ad un sottoinsieme dell'insieme degli utenti, quindi vale $Utente \subset Negligenti$.
  [*NB*] Si noti che una policy che non prevede intersezioni è formalmente possibile ma non è implementabile.
- **Utente**.
  Definiamo un utente come una qualunque entità ricopra un certo ruolo;
- **Operazioni**.
  Questa policy regolamenta le seguenti operazioni per i precedenti ruoli: *lettura*, *scrittura*, *downgrade* e *cambio password*;
- **Modalità**, obbligo, permesso, divieto e discrezionalità. In particolare, con disrezionalità intendiamo

A partire dalla definizione delle modalità sopra elencate, potrebbero nascere delle ambiguità sintattiche. Spieghiamo quest'ultime specificando cosa deve essere inteso secondo una determinata modalità svolta su una generica azione a.
Fissata una *modalità base*, fissiamo le **relazioni tra le modalità** *derivate* e la modalità base.
In questo caso, fissiamo come modalità base il concetto di **obbligo** rispetto una generica azione a.

- *Modalità base*: $Obbligatorio(a)$.
- $Vietato(a) = Obbligatorio(\neg a)$.
  Il divieto di svolgere l'azione a è uguale ad affermare che è obbligatorio svolgere l'opposto dell'azione a, ovvero $not(a)$;
- $Permesso(a) = \neg Obbligatorio(\neg a)$.
  Il permesso di svolgere un azione a è uguale alla non esistenza di un obbligo rispetto l'opposto dell'azione a.
- $Discrezionale(a) = \neg Obbligatorio(a)$.
  Affermare che l'azione a è discrezionale indica che non vi è obbligo di svolgere l'azione a

Talvolta, all'interno di una politica, possono esistere delle **inconsistenze**. Distinguiamo due tipi di inconsistenza:

- **Contraddizione**, il che nasce se esiste l'obbligo di svolgere un azione a ed allo stesso tempo non esiste l'obbligo di svolgere quest'ultima attività a. Una contraddizione può essere formalmente definita attraverso la seguente notazione:
  $Obbligatorio(x) \wedge \neg Obbligatorio(x)$. Un esempio di contraddizione all'interno della policy giocattolo esiste secondo le regole 3 e 7.

- **Dilemmi**, se esiste l'obbligo di svolgere un azione a ed allo stesso tempo esiste l'obbligo di svolgere l'opposto di quest'ultima attività a, ossia $\neg\, a$. Una contraddizione può essere formalmente definita attraverso la seguente notazione: $Obbligatorio(x) \wedge Obbligatorio(\neg\, x)$. Un esempio di dilemma all'interno della policy giocattolo esiste secondo le regole 8 e 9.

E' importante notare che le singole regole all'interno della policy sono perfette. Il problema nasce a partire dalla convivenza di quest'ultime, le quali potrebbero generare (come in questo caso) delle inconsistenze. [?]

> [$Esame$] Quali sono gli elementi fondamentali di una policy?

> [$Esame$] Spiegare mediante definizione ed esempi i termini "inconsistenza" e "dilemma".

Un esempio reale di politica sicurezza: MAC

Un esempio reale di politica sicurezza è il modello **MAC** (acronimo di "*Mandatory Access Control*", ovvero "*Controllo di accesso mandatorio*"), un modo per scrivere o implementare politiche mandatorie basate sull'obbligo, le quali vengono utilizzate storicamente in ambito militare (Bell-Lapadula, Biba) o Sistemi Operativi ad alta sicurezza (per es. SELInux, AppArmor e Tomoyo).

> [$Approfondimento$] Mandatory Access Control (MAC) è un meccanismo di sicurezza che limita il livello di controllo che gli utenti (soggetti) hanno sugli oggetti che essi creano. A differenza di un DAC di attuazione, in cui gli utenti hanno il pieno controllo sui propri file, directory, ecc, MAC aggiunge etichette aggiuntive, o categorie, a tutti gli oggetti del file system. Utenti e processi devono avere l'accesso appropriato a queste categorie prima di poter interagire con questi oggetti.

Un esempio reale di politica sicurezza: RBAC

L'opposto rispetto il MAC è il modello **RBAC** (acronimo di *Role-Based Access Control*), un modo per scrivere politiche *non mandatorio* bensì modificabili e basate sui permessi associati a ciascun ruolo. Le policy role-based spesso prevedono una semplificazione delle modalità (per es. le ACL prevedono una semplificazione della modalità basata sul permesso).
Per completezza, definiamo il *permesso* di svolgere un attività a come l'assenza del divieto di svolgere quest'ultima attività, mentre definiamo il *divieto* di svolgere un attività a come il non permesso di svolgere a.
Il modello RBAC è spesso utilizzato all'interno dei comuni SO.

Un esempio di meccanismo implementativo: ACM

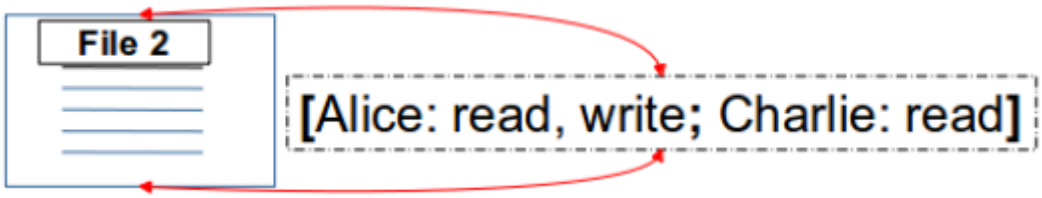
Un secondo esempio reale di politica di sicurezza è **ACM** (acronimo di "*Access Control Matrix*"), il quale si basa su una tabella che specifica quale utenza possa / abbia il *permesso* di fare svolgere una specifica attività.
Un esempio di ACM è rappresentato dalla seguente tabella.

User	File1	File2	Program1
Jax	Read, write	Read	Exec
Bob		Read	
Charlie	Read	Read, write	Read, exec

ACL e CaL

Due meccanismi implementativi per il meccanismo ACM sono:

- **ACL**, (acronimo di *Access Control List*), il quale sceglie di implementare l'ACM attraverso la memorizzazione di ogni colonna della matrice. Un esempio di ACL è la seguente: `Jax: Read, write, Bob: , Charlie: Read`.



- **CaL**, (acronimo di *Capability List*), il quale sceglie di implementare l'ACM attraverso la memorizzazione di ogni riga della matrice. Un esempio di CaL è la seguente: ``.

Modello di attaccante

Attraverso lo Schneier è possibile notare la tassonomia degli attaccanti, i quali possono essere classificati secondo moventi, risorse ed altri criteri ed essere raccontati secondo varie definizioni (e.g. hacker, servizi segreti ed altro ancora). Purtroppo limitarci a studiare quest'ultima sarebbe riduttivo. Per nostra fortuna il professore espleta al meglio l'argomento.

I **modelli di attaccante** ci consentono di astrarre dal contesto e dalle definizione (per es. *spionaggio industrale*, etc...) nella stessa misura in cui la complessità asintotica ci permette di astrarre dalle capacità computazionali della macchina.
Possiamo definire un modello di attaccante come la specifica (i.e *le capacità offensive*) di un preciso attaccante.

Un modello di attaccante è il modello **Dolev-Yao**, nato nel 1983 e così chiamato dai suoi autori "*Danny Dolev*" ed "*Andrew Yao*", il quale definisce l'attaccante come *unico* e *super-potente*, ovvero in grado di controllare l'intera rete, ma non di violare la **crittografia**.

Nel 2003, il prof. Bella definisce un nuovo modello di attaccante, il modello **General Attacker**, più aderente ai tempi recenti ed in cui l'attaccante non è uno solo bensì può essere chiunque. Quest'ultimi non necessariamente hanno interesse nel colludere con altri ma nel caso peggiore hanno totale controllo della rete e non sono in grado di violare la crittografia.

Crittografia

*Appunti di **Giuseppe Pitruzzella** - Corso di Internet Security @ DMI, UniCt*

Introduzione

La crittografia è la scienza di *criptare* (o *codificare*) un messaggio binario m (i.e. un testo in chiaro e comprensibile) in un *critto-testo* o *cypher-text* c (un testo incomprensibile), il quale può essere *decriptato* (o *decodificato*) per ottenere nuovamente m.
In generale, la crittografia è un arte molto antica. Esistono, infatti, tracce risalenti all'antica Grecia.

Crittosistema

Un crittosistema è una coppia di algoritmi, uno per cifrare ed uno per decifrare. Questi due algoritmi hanno una caratteristica in comune: i parametri in input, ovvero il **testo** (i.e. una stringa di bit, cifrata nel caso di decifratura) e la **chiave** (i.e una stringa di bit) e restituiscono un solo valore in *output*. In particolare, notiamo:

- Un algoritmo $\mathcal{E}$ per la **codifica** del messaggio, per cui data una chiave k e un testo m, questo viene codificato attraverso $\mathcal{E}(m, k)$, il quale produce m_k, ossia il *messaggio criptato*.
- Un algoritmo $\mathcal{D}$ per la **decodifica** del messaggio, il quale data una chiave k^1 e un crittotesto m_k, questo viene decodificato attraverso $\mathcal{D}(m_k, k^1)$, che produce m (i.e. il messaggio comprensibile) se **precise condizioni** legano k con k^1.

Queste precise condizioni ci permettono di distinguere i due tipi di crittografia: la **crittografia simmetrica** e la **crittografia asimmetrica**.

> Il termine chiave è associato storicamente ai chiavistelli, quindi l'oggetto che permette di aprire. Noi intenderemo quest'ultima secondo la sua versione digitale, ossia una stringa di bit.

> Possiamo definire la crittoanalisi come la scienza (i.e tecnica) finalizzata a rompere un crittotesto pur senza conoscere la chiave.

Segretezza di una chiave

Abbiamo capito che l'algoritmo (in particolare, crittosistemi) sarà **pubblico** (la specifiche degli algoritmi $\mathcal{E}$ e $\mathcal{D}$) e manterremo segreta la chiave se vogliamo mantenere segreto il contenuto di un crittotesto.
Tuttavia, sappiamo anche che vi è sempre la possibilità (se pur minima) di indovinare la

chiave attraverso un approccio di bruteforce, per cui maggiore è la lunghezza k del segreto, minore è la probabilità di scoprire quest'ultimo. In particolare, data una chiave di lunghezza k, esiste probabilità di indovinare il segreto uguale a:

$$Pr[Guess(k)] = \frac{1}{2^k}$$

Crittografia simmetrica

La prima tipologia di crittografia che studiamo è la storica **crittografia simmetrica**, la quale prevede che la precisa condizione di cui scrivevamo precedentemente sia uguale alla relazione di uguaglianza tra k e k^1.
Formalmente, possiamo definire un crittosistema simmetrico, se l'unico modo per ottenere il testo in chiaro m da un *crittotesto* si basa sull'utilizzo della **stessa chiave** $k^1 = k$ utilizzata per costruire il *crittotesto*. Conseguentemente, per ogni $k \neq k^1$ segue un messaggio diverso da m.
Riassumendo, le **caratteristiche** della crittografia simmetrica sono le seguenti: (i) una chiave $k = k^1$ (tipicamente con una lunghezza uguale a 128 o 256 bit) e (ii) la sua velocità. Alcuni crittosistemi simmetrici sono, per esempio, il *Cifrario di Cesare*, *DES* e *3DES*.

Un **assunzione** fondalmentale si basa sull'assumere che ogni crittosistema sia *pubblico* mentre le chiavi crittografiche sono tipicamente mantenute segrete. Ovviamente mantenere un chiave segreta permette comunque ad un attaccante di provare ad "*indovinare*" la chiave secondo un approccio *bruteforce*. Un approccio di questo tipo ha probabilità di riuscita pari a $\frac{1}{2^n}$, un numero che sicuramente disincentiva rispetto l'attacco ma che ci suggerisce che un modo per irrobustire una chiave sia aumentare la sua lunghezza.

Crittografia simmetrica in rete

Supponiamo che $Alice$ e Bob vogliano comunicare in modo segreto e che chiunque abbia una propria chiave segreta, la quale non deve essere rivelata nello stesso modo in cui non viene rivelata una propria password (dalla quale differisce esclusivamente e banalmente per la lunghezza, la quale è *più grande* per la chiave). Supponiamo che sia $Alice$ che Bob abbiano una chiave, rispettivamente k_a e k_b, utili a cifrare i messaggi da inviare. In questo modo, un attaccante $Charlie$ può al più intercettare il messaggio criptato, garantendo la proprietà di segretezza tra i due intelocutori.
Purtroppo, il protocollo così come indicato *non è funzionante* poiché Bob, per leggere il messaggio criptato da $Alice$, necessita della conoscenza di k_a, chiave non al momento conosciuta da Bob.

Come potrebbe Bob conoscere la chiave di $Alice$, ossia k_a? $Alice$ potrebbe inviare preventivamente la sua chiave a Bob, tuttavia ciò richiede un notevole grado di fiducia; d'altra parte proteggere la chiave k_a con un altra chiave riproporrebbe il problema.
Questo è il vero limite della crittografia simmetrica, il quale si basa sulla chiave stessa, che deve essere conosciuta sia da chi cifra il messaggio che da chi decifra il messaggio stesso.

Limiti della crittografia simmetrica

Come accennato precedentemente, il **limite fondamentale** della crittografia simmetrica si basa sulla condivisione di un segreto iniziale tra $Alice$ e Bob. Tipicamente il segreto condiviso per la comunicazione segreta non è la chiave a *lungo-termine* di uno degli interlocutori, bensì una chiave a *breve-termine* relativa ad entrambi, ossia k_{ab}. In definitiva, Alice utilizza la sua chiave a *lungo-termine* esclusivamente per negoziare un segreto condiviso con l'interlocutore, ovvero una chiave a breve termine per la singola sessione, ossia k_{ab}.

> [$Esame$] Qual è il limite fondamentale della crittografia simmetrica?

> [$Esame$] Si formalizzi una definizione di crittografia simmetrica.

Crittografia asimmetrica

D'altra parte, studiamo adesso la seconda e più recente tipologia di crittografia: la **crittografia asimmetrica**. All'interno di quest'ultima crittografia non esiste una sola chiave bensì una coppia di chiavi in cui l'una è l'inversa dell'altra. Le due chiavi a cui facciamo riferimento sono k e la sua inversa k^{-1} (non riferito all'*operazione matematica* ma così denotata).
Si noti che ciascuna chiave **non si può ricavare dall'altra** (il che è un *problema intrattabile*), motivo per cui la coppia di chiave va generata insieme (i.e. *monoliticamente*).
Attraverso un crittosistema asimmetrico, l'unico modo per ottenere il testo in chiaro da un *crittotesto* si basa sull'utilizzo dell'inversa della chiave utilizzata per costruire il *crittotesto*. Conseguentemente, per ogni $k^1 \neq k^{-1}$ segue un messaggio diverso da m.

Alcuni crittosistemi asimmetrici sono, per **esempio**, *DSA* ed *RSA*.
Caratteristiche della crittografia asimmetrica sono (i) chiavi tipicamente di lunghezza pari a $1024\ bit$ e (ii) una maggior lentezza rispetto la crittografia simmetrica.

Crittografia asimmetrica in rete

Attraverso il paragrafo precedente abbiamo capito che esiste un limite nel momento in cui $Alice$ e Bob vogliano comunicare in segreto secondo la crittografia simmetrica, la quale li costringe a negoziare un segreto a *breve-termine* facendo uso delle loro chiavi a *lungo-termine*.
D'altra parte, secondo la crittografia asimmetrica notiamo un miglioramento della situazione. Attraverso la crittografia asimmetrica, ipotizziamo che $Alice$ sia munita della sua coppia di chiavi (k_a, k_a^{-1}), rispettivamente la coppia *chiave-pubblica* (i.e. conosciuta da *tutti*), *chiave-privata* (i.e. conosciuta dalla sola $Alice$ e, *per assunto*, da Bob) e viceversa per Bob.
Ipotizzando, quindi, la medesima conversazione segreta fra $Alice$ e Bob, quale chiave dovrebbe utilizzare $Alice$ per inviare un messaggio a quest'ultimo? Notiamo, innanzitutto, che se $Alice$ utilizzasse la sua chiave pubblica k_a per criptare il messaggio, allora solo egli stessa potrebbe leggerne il contenuto attraverso la sua chiave privata k_a^{-1}. D'altra parte, se $Alice$ utilizzasse la sua chiave privata k_a^{-1} per criptare il messaggio, allora chiunque potrebbe leggerne il contenuto attraverso sua chiave pubblica k_a.

Soluzione è quindi far sì che $Alice$ utilizzi la chiave pubblica del destinatario stesso (Bob), k_b. In questo, Bob sarà l'unico a poter decriptare il messaggio secondo la sua chiave privata k_b^{-1}, quindi leggere quest'ultimo.

Limiti della crittografia asimmetrica

Il limite della crittografia asimmetrica si basa sulla *verifica* che la chiave pubblica del destinatario appartenga effettivamente ad esso.
Il protocollo, infatti, funziona a partire dall'ottenimento dell'informazione relativa alla chiave pubblica del destinatario Bob, informazione che potrebbe non essere corretta (**limite**).
Immaginiamo, quindi, che un attaccante $Charlie$ con la sua coppia di chiavi (k_c, k_c^{-1}) sia riuscito a fornire ad $Alice$ l'informazione k_c, facendo credere a quest'ultima che sia la chiave pubblica di Bob, k_b. In tal caso, $Alice$ dopo aver criptato il messaggio attraverso k_c invierebbe il tutto; il messaggio criptato $\{m\}_{k_b}$ verrebbe quindi intercettato da $Charlie$, il quale può decriptare il messaggio e leggere il contenuto destinato in origine a Bob.
Supponendo che anche Bob riceva $\{m\}_{k_b}$, allora decriptando il messaggio attraverso la sua chiave privata k_b^{-1}, egli leggerà un contenuto incomprensibile.

Il problema rispetto l'ottenimento della chiave pubblica di Bob per $Alice$ viene risolta attraverso la **certificazione**.

> A partire dalle nostre più rosee ipotesi, possiamo affermare che esiste la possibilità di decriptare e leggere il messaggio esclusivamente se in possesso della chiave privata di Bob. Per noi, infatti, la cifratura è uno strumento che assumiamo funzioni sempre.

Crittosistema sicuro

Possiamo definire un crittosistema sicuro quando, per ogni messaggio m criptato attraverso una qualsiasi chiave k ($\mathcal{E}(m, k)$), decriptando quest'ultimo secondo una qualsiasi chiave k_1 diversa da k (*caso simmetrico*) o k^{-1} (*caso asimmetrico*) ($\mathcal{D}(\mathcal{E}(m, k), k_1)$), allora ottenere n non aumenta le probabilità di ottenere il messaggio m o porzioni di esso. In altre parole, un crittosistema è sicuro quando esiste una sola chiave k (k per il *caso simmetrico* o k^{-1} per il *caso asimmetrico*) in grado di aprire il crittotesto.

Hash crittografico

Teoricamente, possiamo definire un hash crittografico come una funzione in grado di soddisfare i seguenti requisiti:

- Il calcolo dell'hash per un messaggio m è operazione veloce;
- Il calcolo del messaggio m a partire dal suo hash $hash(m)$ è un problema intrattabile;
- Cambiare un bit all'interno di m, ottenendo m_1, fa sì che l'hash calcolato per m_1 sia completamente diverso rispetto l'hash di m;

- Non è possibile trovare lo stesso $hash(m_1) = hash(m_2)$, per due messaggi m_1, m_2 tale che $m_1 \neq m_2$.

Com'è possibile implementare un funzione hash in grado di soddisfare questi requisiti? E' importante non confondere la defizione teorica della funzione hash con una sua ipotetica implementazione, in ogni caso le implementazioni di funzioni hash nel tempo sono state diverse.
Esempi di funzioni hash sono:

- **SHA-1**, la quale ha dimostrato nei limiti nel tempo, ovvero la *non* capacità di soddisfare i requisiti descritti nella teoria;
- **MD5**, che allo stesso modo ha dimostrato nei limiti nel tempo (sono state trovate delle *collisioni*).

Un esempio di utilizzo dell'hash: CTSS + Hashing

Il primo sistema di autenticazione (orientato alla *multi-utenza*) è **CTSS**, sviluppato dall'**MIT** alla fine degli anni $'60$. Si basava su una tabella in cui fossero definiti username e password, attraverso cui l'utente che voleva accedere al sistema avrebbe dovuto inserire la sua password, una stringa che doveva coincidere quella definita all'interno della tabella stessa. Questo è un problema nella misura in cui un attaccante può essere in grado di arrivare alla tabella, motivo per cui, 7 anni dopo (a Cambridge) nasce l'idea di potenziare il sistema secondo una funzione hash.

> E' importante non confondere l'hash con la crittografia, le quali si differenziano l'una dall'altra. Una differenza importante si basa sul fatto che un messaggio criptato può essere decriptato attraverso la corretta chiave, diversamente da un hash crittografico per cui non esiste la possibilità (concreta) di ritornare al messaggio di partenza.

Salting

Come sappiamo, minore è la lunghezza della password e minori saranno i tentativi da svolgere prima di indovinare, anche se alla generazione di una password segue la generazione dell'hash e la successiva comparazione. Dunque, il fatto che sia presente l'hash non rende impensabile un tentativo di *bruteforce*.
Per questo motivo, nasce il concetto di **salting** (*sale*) nasce per incrementare la lunghezza di una password, per *condire*. In questo modo è possibile rendere una password più robusta rispetto *attacchi a dizionario*.

> Storicamente il sale era composto da $12\ bit$, oggi insufficienti.

Aggiunta di una password in Unix

E' interessante notare l'utilizzo del sale all'interno del mondo Unix e, più in generale, come avvenga aggiunta una password all'interno di un sistema come quest'ultimo. Ipotizzando che un *sys admin* generi un nuovo utente con una nuova password, allora sappiamo essere necessario il calcolo della sua versione *hash*, la quale verrà memorizzata nel file *etc/passwd*.
Ci soffermiano, quindi, su come Unix, storicamente, proponeva di svolgere l'hash di quest'ultima password. Sostanzialmente il tutto si basa su una certa tipologia di funzione di encryption, una funzione di *One-Way Encryption*, ovvero `crypt(3)`.
Quest'ultimo è in grado di fornire una versione "*hash*" (formalmente non un *hash*) della password, ovvero una versione di quest'ultima per cui è intrattabile eseguire il *decrypt*. I vantaggi di utilizzare una funzione crittografica e non un hash sono presto detti: attraverso `crypt()`, infatti, possiamo godere della prime e seconda proprietà sopra elencate relative all'hash crittografico e dimenticarci delle altre proprietà di difficile implementazione.
In generale, `crypt(3)` necessita di una stringa da crittografare ed una chiave secondo cui svolgere l'*encrypt*. Questi due parametri sono rispettivamente:

- Una stringa formata da sale ed una stringa costante (solitamente formata soli zero); in particolare il sale era formato da due caratteri scelti randomicamente dal set (a-zA-Z0-9) used to perturb the algorithm in one of 4096 different ways.
- Una chiave di 56 bit ricavata dai 7 bit meno significativi dei primi 8 caratteri della password.

User ID	Salt	Ecrypted Password(Salt + 0s, Key)
jax	7a	r8f94hf7fdvfd8d

Verifica di una password in Unix

La verifica di una password all'interno di un sistema Unix si basa sulla seguente procedura. Dopo che l'utente ha inserito user id e password, il sistema cerca lo user id all'interno della tabella in cui memorizza quest'ultimi dati, trovando conseguentemente la tupla contenente: userid, il sale e la ecrypted password. A questo punto, viene effettuata il confronto tra la live-password, generata a partire dal precedente sale e la password digitata, e la encrypted password.

Freshness

Un ulteriore pezzo di puzzle relativo alla crittografia è dato dalla **freshness** (i.e. *freschezza*, *essere recenti*), non una proprietà ma un **attributo** di un altra proprietà, in particolare dell'autenticazione e la segretezza). Essa, quindi, indica la "freschezza" per la proprietà (e.g. dell'autenticazione).

Il legame tra freshness e **confidenzialità**, ossia il legame tra la freshness ed un segreto è molto importante poiché essendo possibile effettuare il *bruteforce* per ogni segreto, procedura per cui è necessario del tempo, allora maggiore sarà il tempo trascorso dalla generazione del segreto, maggiore il tempo a disposizione per effettuare *bruteforce*. Per questo motivo, un segreto *fresco* è più robusto rispetto ad un attacco bruteforce.
D'altra parte, il legame tra freshness ed **autenticazione** esiste, poichè ciò che autentico ad

un istante t potrebbe non essere autenticato ad un istante $t+1$ (i.e. potrebbe non esser più lui). Si pensi, per esempio, se un bancomat mantenesse l'autenticazione di un utente dopo che quest'ultimo ha svolto la determinata operazione e si è allontanato da esso; in quel caso, qualcun'altro potrebbe svolgere operazioni per conto di quest'ultimo, il che non è desiderabile.

Come è possibile **implementare** la freshness? Sostanzialmente esistono due modi per implementare la freshness, ovvero:

- Attraverso un **timestamp**, o marcatore temporale, scrivendo la data di produzione per una chiave. Quest'ultima viene analizzata e poi accettata solo se "*rientra nella sua finestra temporale*". Si noti che i timestamp hanno senso finchè tutti noi abbiamo un orologio sincronizzato. La sincronizzazione degli orologi su sistemi distribuiti è un problema risolto con un protocollo che è necessario sia sicuro.

- Attraverso **Nonce** (*Number random that is used only Once*), basato sull'idea che si utilizzi per veicolare la freshness sull'autenticazione. Supponiamo che un autenticazione di $Alice$ venga effettuata attraverso l'invio di una chiave segreta; se un attaccante $Charlie$ ha intercettato quest'ultima chiave, allora $Charlie$ potrebbe autenticarsi come Bob. Una **misura** per evitare questo attacco, chiamato **replay attack**, si basa sulla freshness.

- i $Alice$ ->($Nonce_A$)-> Bob *attraverso una condivisione sicura*

- $i+1$ Bob ->(($Nonce_A$ Msg)$_K$)-> $Alice$

- $i+2$ $Alice$ *autentica* Bob.

- $i+3$ Bob *esce. Inizio del tentativo di replay attack da parte di* $Charlie$.

- $i+4$ *Il messaggio di* Bob *ad* $i+1$ *viene intercettato da* $Charlie$.

- $i+5$ $Charlie$ ->(($Nonce_A$ Msg)$_K$)-> $Alice$

- $i+6$ $Alice$ *non accetta più* $Nonce_A$ *poiché non rientra nella sua finestra temporale*

> Una OTP (*One Time Password*) è un codice numerico generato randomicamente che può utilizzato usato solo una volta. Può usare utilizzato come *freshness* poiché cambia continuamente e si basa su una sincronizzazione preconcordata, motivo per cui ciascuno dei due interlocutori conosce questo codice ed a quale intervallo di tempo si riferisce.

Protocolli basilari per la sicurezza

*Appunti di **Giuseppe Pitruzzella** - Corso di Internet Security @ DMI, UniCt*

Introduzione

Iniziamo ad operare il passaggio dalla crittografia alla sicurezza. Come si utilizzano in pratiche le primitive crittografiche studiate nel capitolo precedente? Lo vediamo all'interno di questo capitolo, in cui sposteremo il nostro focus sui protocolli di sicurezza, protocolli cui obiettivo è ottenere proprietà di sicurezza attraverso delle misure crittografiche. In questi protocolli faremo le assunzioni peggiori dal punto di vista offensivo, ovvero le assunzioni Dolev-Yao.

La **sintassi** utilizzata per discutere i protocolli in questo capitolo è la seguente:

- *Messaggi atomici*
  - **Nomi degli agenti**: $A, B, C...$
  - **Chiavi crittografiche**: $k_a, k_b, \dots, k_a^{-1}, k_b^{-1}, \dots, k_{ab}, k_{ac}, \dots$
  - **Nonce**: $N_a, N_b, \dots$
  - **Timestamp**: $T_a, T_b, \dots$
  - **Digest** (i.e. *hash*): $h(m), h(n), \dots$
  - **Etichette**: "Trasferisci \; 100€ \; dal \; conto \; di \; ..."
- *Messaggi composti*
  - **Concatenazioni**, in cui ciascuno può essere un crittotesto: m, m^1;
  - **Crittotesti**, in cui il testo all'interno può essere concatenazione: m_k.

> Si può autenticare un messaggio concatenato del tipo $(m, m^1)_k$? Ricordiamo che autenticazione indica la dimostrazione dell'identità dell'interlocutore, che per estensione della definizione ci riferiamo all'autenticazione del messaggio, ovvero verificare che il messaggio è stato inviato proprio dall'autenticatore ("*Autenticare $Alice$ attraverso il messaggio m*..."). La risposta è no, poichè se così fosse allora chiunque potrebbe svolgere la concatenazione di un messaggio e compromettere l'autenticazione del messaggio.

Il modello di attaccante più comune indica che possiamo immaginare un modello di rete in cui vi è un attaccante che intercetta ogni possibile comunicazione e modifica quest'ultimi a suo piacere. L'unica cosa che l'attaccante non può fare e rompere la cifratura.
Storicamente, i due autori raccontavano il loro modello basandosi su due agenti 007, i quali fidandosi l'uno dell'altro e si trovandosi in due posti lontani, devono comunicare in sicurezza avendo l'intera rete come nemico.

Vent'anni dopo, si osserva che il contesto è cambiato, per cui si ipotizza che un singolo nodo della rete può avere capacità offensive pari al Dolev-Yao.

Come si scrive un protocollo di sicurezza? Fondamentalmente, il tutto si basa sulla seguente notazione: (i) il numero del passo, (ii) il mittente, (iii) una freccia, (iv) il ricevente, (v) due punti, (vi) il messaggio cirttografico; il tutto viene ripetuto per ogni singolo passo del protocollo.

$$NumeroPassoMittente \longrightarrow Ricevente : MessaggioCrittografico$$

> E' importante notare che nel momento in cui basiamo il tutto sull'attaccante DY, allora il messaggio crittografico perde di significato poiché egli stesso potrebbe essere modificarlo.

Protocolli basilari per la segretezza

Un protocollo di sicurezza più *basilare* che studiamo si basa sulla crittografia **simmetrica**, motivo per cui prevede un chiave condivisa tra i soli $Alice$ e Bob (unico prerequisito), attraverso cui $Alice$ si può autenticare con Bob (o viceversa). Questo protocollo è robusto, anche contro DY pur patendo qualche attacco (?).

- $Prerequisito\,1$: Esiste una chiave k_{ab} condivisa tra i soli $Alice$ e Bob;

$$1.\ A \longrightarrow B\ :\ m_{k_{ab}}$$

D'altra parte, esiste anche la variante con la crittografia **asimmetrica**, la quale prevede certi prerequisiti.

- $Prerequisito\,1$: Tutti abbiano una coppia di chiave;
- $Prerequisito\,2$: $Alice$ deve avere la garanzia che la chiave pubblica di Bob sia effettivamente di Bob.

$$1.\ A \longrightarrow B\ :\ m_{k_b}$$

Protocolli basilari per l'autenticazione

Un protocollo basilare per l'autenticazione nel caso **simmetrico** prevede l'introduzione di un assunzione, ovvero che il ricevente dell'autenticazione sappia a chi è associata la chiave di sessione, con chi condivide la chiave di sessione.

- $Prerequisito\,1$: Esiste una chiave k_{ab} condivisa tra i soli $Alice$ e Bob;
- $Prerequisito\,2$: Bob può verificare il $Prerequisito\,1$.
  Relativamente al protocollo stesso, segue quindi un *encrypt* del messaggio per ogni messaggio inviato da $Alice$ a Bob (o viceversa).

$$1.\ A \longrightarrow B\ :\ m_{k_{ab}}$$

D'altra parte, esiste il caso **asimmetrico**, in cui assumono i seguenti prerequisiti:

- $Prerequisito\,1$: $Alice$ possiede una chiave privata $k_{a^{-1}}$ valida;
- $Prerequisito\,2$: Bob può verificare che la chiave pubblica di $Alice$ sia effettivamente di $Alice$ stessa.

Relativamente al protocollo stesso, segue che se $Alice$ vuole autenticarsi con Bob, allora egli esegue l'encrypt del messaggio attraverso la sua chiave privata, il che garantisce autenticazione.

$$1.\ A \longrightarrow B\ :\ m_{k_{a^{-1}}}$$

Combinare segretezza ed autenticazione

Combinare i protocolli basilari per l'autenticazione e la segretezza è possibile; per fare ciò, infatti, sarà sufficiente e necessaria l'unione dei prerequisiti necessari ad ognuna delle due versioni di protocolli.
Nel primo caso, basato su crittografia **simmetrica**, notiamo un protocollo analogo a ciò che abbiamo visto per l'autenticazione:

- $Prerequisito\,1$: Esiste una chiave k_{ab} condivisa tra i soli $Alice$ e Bob;
- $Prerequisito\,2$: Bob può verificare il $Prerequisito\,1$.
  Relativamente al protocollo stesso, segue quindi un *encrypt* del messaggio per ogni messaggio inviato da $Alice$ a Bob (o viceversa).

$$1.\ A \longrightarrow B\ :\ m_{k_{ab}}$$

Nel caso **asimmetrico**, chiaramente, le assunzioni diventano 4. In particolare, notiamo che attraverso il c

- $Prerequisito\,1$: $Alice$ possiede una chiave privata $k_{a^{-1}}$ valida;
- $Prerequisito\,2$: Bob può verificare che la chiave pubblica di $Alice$ sia effettivamente di $Alice$ stessa.
- $Prerequisito\,3$: Bob possiede una chiave privata $k_{b^{-1}}$ valida;
- $Prerequisito\,4$: $Alice$ può verificare che la chiave pubblica di Bob sia effettivamente di Bob stesso.

Secondo quest'ultimo protocollo basato su crittografia **asimmetrica**, possiamo notare due approcci uguali a meno di qualche sfumatura.

- $1.\ A \longrightarrow B\ :\ \{m_{k_a^{-1}}\}_{k_b}$
  Attraverso questo primo approccio, certamente l'unico a poter decriptare il messaggio sarà il destinatario Bob, il quale potrà poi decriptare una seconda volta il messaggio interno secondo la chiave pubblica di $Alice$. Ne segue che questo approccio è migliore se l'obiettivo di $Alice$ è autenticarsi esclusivamente con Bob, quindi garantire la segretezza.

- 1. $A \longrightarrow B \ : \ \{m_{k_b}\}_{k_a^{-1}}$
  Attraverso questo secondo approccio, certamente tutti possono decriptare il messaggio destinato a Bob; tuttavia rimane Bob l'unico che potrà poi decriptare una seconda volta il messaggio interno secondo la sua chiave privata. Ne segue che questo approccio non è il migliore per garantire la segretezza.

Protocolli basilare per l'integrità

Come abbiamo già detto, qualunque messaggio sulla rete potrebbe essere alterato, in particolare anche un messaggio protetto per segretezza ed autenticazione. A questo proposito, potrebbe essere utile l'approccio che prevede di affiancare ad ogni messaggio un checksum, come per esempio, l'hash del messaggio.
Tuttavia, fare ciò non nega all'attaccante che vuole sostituire m con m^1, di sostituire allo stesso $h(m)$ con $h(m^1)$.
E' necessario aggiungere un livello di irrobustezza.

Attraverso la crittografia **asimmetrica**, possiamo potenziare l'approccio precedente facendo uso di una chiave, ovvero la chiave privata del mittente $Alice$. In particolare, criptiamo l'hash del messaggio $h(m)$ con la chiave privata di $Alice$, ottenendo $h(m)_{k_{a^{-1}}}$.
In questo modo, anche se l'attaccante ricalcolasse l'hash egli non potrebbe mai eseguire l'encrypt di quest'ultimo.

$$1. \ A \longrightarrow B \ : \ m, \ h(m)_{k_{a^{-1}}}$$

A questo punto, quando Bob riceve il messaggio di $Alice$, allora egli (i) calcola l'hash del messaggio, (ii) decodifica $h(m)_{k_{a^{-1}}}$ per poi confrontare i due risultati. Se i due digest sono uguali, allora viene garantita l'integrità.
E' importante notare che Bob, con l'operazione (ii) riesce effettivamente ad individuare se il costruttore del digest è anche chi ha inviato il messaggio, ovvero $Alice$. Il tutto, quindi, garantisce autenticazione.

> Non è possibile garantire l'integrità senza garantire l'autenticazione.

La firma digitale

La **firma digitale** ha lo scopo di garantire l'integrità di un documento digitale ed è basata sulla crittografia asimmetrica. I requisiti funzionali di una firma digitale si basa sul fatto che ognuno possa applicare la sua firma digitale per conto di se stesso e mai per conto di altri; inoltre, ognuno deve poter verificare la genuinità di quest'ultima firma (il che non è per una firma cartacea). Una firma digitale si basa su una coppia di **algoritmi**:

- Un algoritmo per la **creazione**;
  Questo primo algoritmo si basa sulle seguenti 3 fasi:
  1. Se devo firmare un documento $Cleartext$, allora inizialmente ne eseguo l'hash, costruendo il **digest**.
  2. In seguito eseguo l'encrypt del digest con la chiave provata del mittente (i.e.

costruttore della firma), ottenendo il **digest crittografato** $Encrypted\,Digest$.
3. A questo punto, la coppia $Cleartext + Encrypted\,Digest$ è uguale alla firma digitale.

> Si noti che relativamente al concetto di firma digitale, non si è mai parlato di segretezza relativa al documento.

- Un algoritmo per la **verifica**.
  Ricevuta un documento firmato, ovvero una coppia $Cleartext + Encrypted\,Digest$, è possibile verificare la genuinità della firma secondo le seguenti 3 fasi:
  1. Eseguo il decrypt di $Encrypted\,Digest$ attraverso la chiave pubblica del mittente (i.e. creatore della firma), ottenendo il digest $Digest^1$.
  2. In seguito, effettuo l'hash del documento $Cleartext$, ottenendo il digest $Digest^2$.
  3. La firma è verificata se $Digest^1 = Digest^2$, il che indica che la coppia ha attraverso la rete senza alterazioni.

  > Qual è la differenza sul piano della sicurezza tra firma grafometrica e digitale? Esistono due grandi differenze rispetto le due: infatti, per la prima diversamente dalla seconda, non esiste (i) autenticazione e (ii) integrità, il che indica che non può garantire che la firma sul foglio sia inalterato.

> Si noti che la chiave privata di cui abbiamo parlato all'interno della firma digitale è fornita da un fornitore di identità digitale.

Il protocollo Diffie-Hellmann

Il protocollo **Diffie-Hellmann**, pubblicato nel 1976, si basa sulla crittografia **simmetrica** (pur somigliando molto alla crittografia asimmetrica data la distinzione delle due chiavi) ed ha l'obiettivo di stabilire un segreto condiviso tra $Alice$ e Bob.
$Alice$ e Bob concordano due parametri **pubblici**, α e β. A questo punto, $Alice$ e Bob generano in modo random rispettivamente X_a ed X_b. A partire da quest'ultimi due, seguono:

$$Y_a = \alpha^{X_a}\ mod\ \beta$$

$$Y_b = \alpha^{X_b}\ mod\ \beta$$

.

> Perchè spedire Y_a sul traffico non espone X_a? Sappiamo che l'inverso dell'esponenziale sia il logaritmo, tuttavia se qualcuno stesse pensando che intercettando una tra le due Y sia possibile arrivare ad α svolgendo l'inverso (i.e. logaritmo), quest'ultimo si sbaglia. Infatti, aggiungere il modulo all'interno del calcolo di Y, rende l'operazione non più un semplice esponenziale, bensì un *logaritmo discreto*, un problema intrattabile.

Nel momento in cui $Alice$ e Bob scambiano le proprie Y, ricevendo rispettivamente Y_b ed Y_a, entrambi *elevano* rispetto il proprio segreto, calcolando: $Y_b^{X_a} \ mod \ \beta$ ed $Y_a^{X_b} \ mod \ \beta$. A questo punto, i due risultati saranno uguali ($Y_b^{X_a} \ mod \ \beta \ = \ Y_a^{X_b} \ mod \ \beta$), motivo per cui possiamo intendere quest'ultimo come il segreto condiviso per i due interlocutori.

$$A \ \ calcola \ \ k_{ab} = Y_b^{X_a} \ mod \ \beta \ = (\alpha^{X_b} \ mod \ \beta)^{X_a} \ mod \ \beta$$

$$B \ \ calcola \ \ k_{ab} = Y_a^{X_b} \ mod \ \beta \ = (\alpha^{X_a} \ mod \ \beta)^{X_b} \ mod \ \beta$$

$$k_{ab} = Y_b^{X_a} \ mod \ \beta \ = \ Y_a^{X_b} \ mod \ \beta$$

> E' importante ricordare che pur definendo le due chiavi X_a ed X_b come chiavi "*private*", queste non sono le chiavi private legate al concetto di crittografia asimmetrica ed il protocollo DH è definito secondo crittografia simmetrica.

> Il protocollo Diffie-Hellmann, purtroppo, non ha previsto l'**autenticazione**, il che indica che $Alice$ e Bob si scambiano rispettivamente Y_a ed Y_b ma entrambi non hanno piena conoscenza di chi sia la persona con chi stanno per concordare una segreto. Ha senso parlare di segretezza di un messaggio a meno dell'autenticazione? No, mai; infatti, supponendo che vi sia un segreto tra due persone, se quest'ultime non si autenticano allora il tutto ha poco senso.

Attacco

Esiste un **attacco** relativo a DH ed all'assenza di autenticazione in quest'ultimo, ovvero:

1. $Alice$ invia a Bob il messaggio contenente Y_a.
2. Il messaggio viene intercettato da $Charlie$, il quale impersonando $Alice$, invia a Bob il messaggio Y_c.
3. Bob, in seguito invia ad $Alice$ il suo Y_b.
4. Il messaggio viene intercettato da $Charlie$, il quale impersonando Bob, invia ad $Alice$ il messaggio Y_c.

> All'interno del punto abbiamo scritto di Charlie che si impersona in Alice, inviando Y_c. Tuttavia, poiché assente l'autenticazione, è formalmente sbagliato scrivere di "*impersonificazione*".

- $1. Alice \longrightarrow Bob : Y_a$
  - $Charlie$ intercetta il messaggio, calcola la sua Y_c attraverso i parametri *pubblici* α e β, ed invia Y_c a Bob.

- $2. Bob \longrightarrow Alice : Y_b$

- $Charlie$ intercetta il messaggio, calcola la sua Y_c attraverso i parametri *pubblici* α e β, ed invia Y_c a Bob.

E' importante notare che dato $k_a = Y_c^{X_a} \ mod \ \beta$ e $k_b = Y_c^{X_b} \ mod \ \beta$, conosciute rispettivamente da $Alice$ e Bob, allora $k_1 \neq k_2$.
Falliscono, quindi, (i) il **requisito funzionale** per cui $Alice$ e Bob potessero avere un segreto condiviso e (ii) la **segretezza** poiché l'attacante, avendo intercettato le due Y, può calcolare sia $k1 = Y_a^{X_c} \ mod \ \beta$ e $k2 = Y_b^{X_c} \ mod \ \beta$, motivo per cui si genera un attacco *man-in-the-middle*.

Conseguenze

L'attacco, quindi, si divide in due fasi:

- Una prima fase in cui per $Charlie$ è possibile intercettare le due Y (le quali non sono segrete) e, data l'assenza di autenticazione, somministrare un nuovo messaggio ad $Alice$ e Bob.
- Una seconda fase in cui $Charlie$ può fare uso delle informazioni Y_a ed Y_b.

Fix

Esistono delle **modifiche** che permettono di irrobustire DH rispetto la prima e/o la seconda fase dell'attacco.
La prima fase (i.e. $1. Alice \longrightarrow Bob : Y_a$) può essere irrobustita facendo uso della chiave privata del mittente; in questo modo è possibile inserire una misura di autenticazione. La seconda fase (i.e. $2. Bob \longrightarrow Alice : Y_b$) può essere irrobustita facendo uso della chiave pubblica del destinatario; in questo modo è possibile inserire una misura di segretezza.
Basta impedire una delle due fasi per fixare l'attacco a DH.
Purtroppo, il fix richiede di ricorrere alla crittografia asimmetrica. Un protocollo per la condivisione di un segreto autentico che non utilizzi primitive asimmetriche è un challenge aperto.

RSA Key Exchange

Un alternativa basata dichiaratamente nel mondo asimmetrico per lo scambio di una chiave esiste ed è **RSA**. RSA si basa sul seguente procedimento; il mittente, $Alice$, *genera randomicamente un segreto* ed esegue l'encrypt di quest'ultimo messsaggio con la chiave pubblica del destinatario. Questo è la scambio di una chiave secondo RSA.
Tuttavia, affinché il protocollo funzioni è necessario che Alice si procuri un **certificato** della chiave pubblica di Bob.

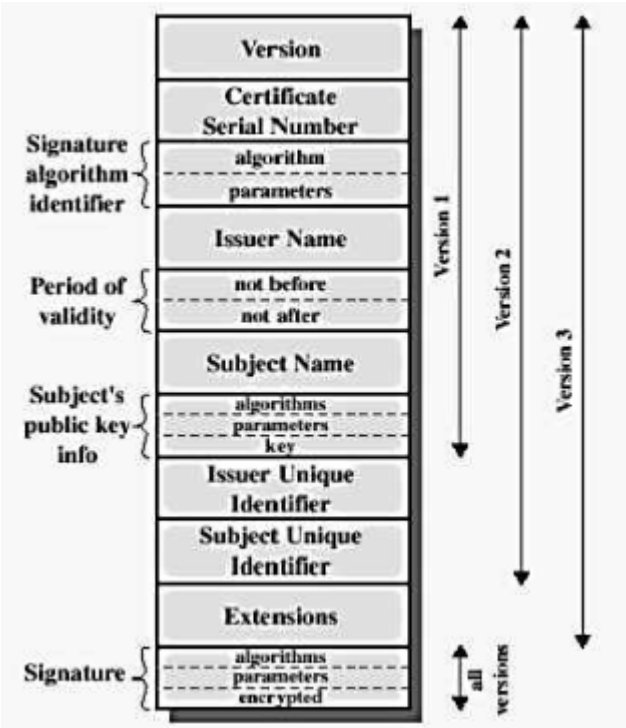
Certificazione

Un primo esempio di certificazione sappiamo essere la carta di identità, un documento che associa un informazione anagrafica ad un informazioni grafica. Tutti i certificati suggellano

associazioni.

Un **certificato digitale** è l'associazione tra l'identità di $Alice$ e una stringa di bit, ovvero la sua *chiave pubblica*. Il certificato garantisce autenticazione ed integrità, tramite la chiave privata k_{ca}^{-1} di una autorità di certificazione;
Una versione semplificata di certificato sarebbe la seguente: $(Alice, k_A)_{k_{ca}^{-1}}$.
Tuttavia, al di là delle semplificazioni, il formato standard del certificato esiste e si chiama **X.509**.

Un certificato è firmato da un'autorità di certificazione (e.g. consorzi di banche), le quali si occupano principalmente di *vendere fiducia*, essere *trustworthiness*.
Si noti che i certificati hanno una validità temporale, ovvero hanno una scadenza.



Public Key Infrastructure (PKI)

Se $Alice$ e Bob vogliono eseguire una sessione *TLS* (basato su cifratura asimmetrica), allora $Alice$ necessita della chiave pubblica di Bob.
Il certificato è l'unico strumento che può dare questa informazione ad $Alice$.
Un certificato della forma *X.509* viene quindi inviato ad $Alice$, che **verifica** quest'ultimo, trovando nome e chiave pubblica di Bob.
Il certificato, per ovvi motivi, può essere alterato ma in quel caso la verifica fallisce.

Cosa vuol dire **verificare un certificato**? Verificare un certificato indica *verificare una firma digitale*, il che indica eseguire la seguente procedura di cui abbiamo già scritto nel capitolo relativo alla firma stessa.

1. Eseguo il decrypt di $Encrypted\,Digest$ attraverso la *chiave pubblica del creatore della firma*, ottenendo il digest $Digest^1$.
2. In seguito, effettuo l'hash del documento $Cleartext$, ottenendo il digest $Digest^2$.
3. La firma è verificata se $Digest^1 = Digest^2$, il che indica che la coppia ha attraverso la rete senza alterazioni.

Per portare a compimento la procedura di verifica è necessaria la conoscenza della chiave pubblica di ca, ovvero k_{ca}, il che ci porta nuovamente al problema iniziale, ovvero: "*chi mi assicura che quella chiave pubblica sia effettivamente dell'ente certificante*?".
E' necessario quindi *re-iterare* il problema (i.e. *verificare una catena di fiducia*) finché non sarà necessario un atto di fede nei confronti dell'**autorità root**.

La catena formata a partire dalla *re-iterazione* del problema non è in realtà una catena, bensì un **albero *n-ario***, formato da una radice che certifica i suoi n figli, i quali a sua volta certificano i propri n figli e così via.
Possiamo definire quest'ultimo albero n-ario come la **Public Key Infrastructure** (PKI).

In definitiva, ripetiamo che il problema della certificazione rispetto la chiave pubblica di Bob prevede una re-iterazione del problema, la quale termina nel momento in cui arriviamo all'autorità root. Si noti che il certificato di root possiede una particolarità rispetto le successive certificazioni date dalle autorità figlie; possiede, infatti, *al di fuori* la chiave privata di root e dentro la sua stessa chiave pubblica.

Nella pratica, è possibile notare tutto ciò di cui abbiamo scritto all'interno del nostro browser, ogni qual volta digitiamo un URL. Modifiche visive all'interno di quest'ultimo (i.e. URL) avvisano l'utente rispetto la genunità del sito Web.
Si noti che è quindi il browser ha decidere di chi fidarsi e di chi non fidarsi (e.g. Google con Semantyc) motivo per cui è necessario tener d'occhio i certificati di root relativi al browser stesso. Ciò è importante poiché pur tenendo conto dei certificati, rimaniamo attaccabili sotto questo punto di vista, per esempio attraverso l'aggiunta di un certificato di root avvenuta a partire da uno script malevolo.

Cosa succede se un nodo è corrotto?

E' interessante chiedersi cosa succederebbe se, per ipotesi, un nodo all'interno dell'albero sia compromesso. In quel caso è importante notare che il danno che quest'ultimo nodo può provocare si muovo da egli stesso verso le foglie (i.e. *verso il basso*), non viceversa. Il tutto *non è retroattivo*, motivo per cui i certificati da attenzionare in tal caso saranno al più successivi all'evento che ha dato inizio alla compromissione del nodo (il che indica l'importanza dell'aspetto temporale).
Tuttavia, se un nodo è compromesso non necessariamente lo sono tutti i nodi al di sotto.
Non confondiamo tutto questo con la perdita di segretezza, la quale non si propaga, ma si riferisce esclusivamente alla perdita di trustworthiness.

Certificati nel browser

Sappiamo che vi è un business legato alle certificazioni, il che indica comprare un certificato per il proprio dominio ed eventualmente per i propri sotto-domini (leggermente diverso per certificati wild-card, quali permettono la certificazione anche per i propri sotto-domini)
Tuttavia, se non volessi comprare un certificato, quindi non volessi acquistare fiducia da qualcuno, allora l'unica opzione è crearsi un certificato da se, un certificato **self-issued**.

Supponendo di trovarsi in una situazione di questo genere, rappresentato dall'immagine sottostante.
Esistono tre opzioni:

- Accettare il certificato in modo permanente;
- Accettare il certificato in modo temporaneo;
- Rifiutare il certificato e non connettersi al sito Web;

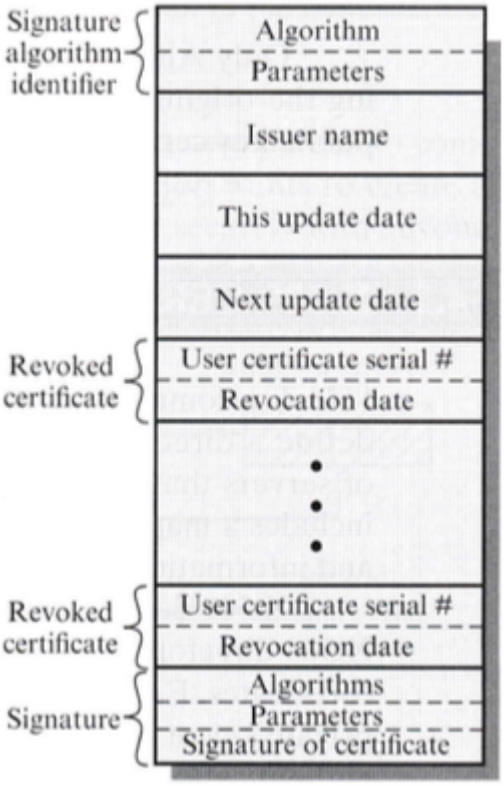
Qual è la scelta più sicura? Certamente se il browser potesse fare la scelta più sicura, allora sarebbe stato il browser stesso ad effettuare la scelta. In questo caso, siamo noi stessi a dover rispondere ad una domanda, una domanda a cui il browser non è in grado di rispondere.
L'utente potrebbe fare un verifica *out-of-band* (i.e. *fuori dal canale*), quindi (i) ispezionare il certificato, (ii) leggere l'hash (i.e. *certificate-finger-print*) e (iii) telefonare al centro di calcolo per verificare quest'ultimo hash.
Il tutto non è scalabile o usabile, motivo per cui potrebbe non accadere.
Rispondiamo, quindi, alla domanda. Accettare il certificato per sempre, implica che potrei potenzialmente essere fregato ogni volta da quel momento in poi; d'altra parte accettare il certificato in modo temporaneo implica accettare dei rischi limitati alla sessione, quindi essere fregati al più una sola volta.

> Negli ultimi tempi, un consorzio si è unito per mettere a disposizione una versione di certificato gratuito, questo è *Let's Encrypt*.

> Abbiamo capito che il senso della certificazione si basa sull'affermare che il contenuto è associato all'URL. Niente ci garantisce che un sito certificato non abbia scopi malevoli.

Revoca di un certificato, CRL

Relativamente al concetto di revoca esiste la **CRL** (acronimo di *Certificate Revocation List*), una lista di certificati revocati con la seguente struttura:



Un certificato revocato indica la revoca della validità del certificato prima della scadenza effettiva di quest'ultimo ed è conseguenza di *data-breach* o richiesta dell'interessato.
Un certificato revocato è firmato dalla medesima autorita di certificazione che ha firmato in principio il certificato stesso.
E' sempre necessario che la revoca di un certificato arrivi ad ogni foglia. In questo modo, evitiamo un **attacco** secondo cui l'attaccante, sfruttando il fatto non sia arrivata quest'ultima revoca, somministra all'utente qualsivoglia contenuto.
Tuttavia, purtroppo, non vi è alcuno standard (i.e. protocollo) rispetto la gestione della revoca. Per questo motivo, originariamente era l'utente stesso ad inserire le CRL all'interno del proprio browser. Ad un certo tempo nasce OCSP, attraverso cui è possibile sincronizzare il proprio browser rispetto una CRL.

> Il più recente strumento per le CRL è Certificate Transparency di Google. "*Per poter criptare il traffico per gli utenti, un sito deve innanzitutto richiedere un certificato a un'autorità di certificazione (CA) attendibile. Il certificato viene poi presentato al browser per l'autenticazione del sito a cui l'utente desidera accedere. Negli ultimi anni, le CA e i certificati emessi si sono rivelati vulnerabili alla compromissione e alla manipolazione, a causa di falle strutturali nel sistema dei certificati HTTPS. Certificate Transparency di Google è stato ideato per proteggere il processo di emissione dei certificati offrendo un framework aperto per il monitoraggio e il controllo dei certificati HTTPS. Grazie al log Certificate Transparency attivi e pubblici, i proprietari dei siti possono cercare su questo sito i nomi di dominio da loro gestiti per verificare che non ci siano errori di emissione dei certificati che fanno riferimento ai loro domini. Google invita tutte le CA a scrivere i certificati emessi in log a prova di manomissione, di tipo append-only e pubblicamente verificabili.*".

Autenticazione

*Appunti di **Giuseppe Pitruzzella** - Corso di Internet Security @ DMI, UniCt*

Esistono diversi tipi di autenticazione come;

- **Persona-persona**, basata sull'incontro tra i due soggetti;
- **Utente-computer**, certamente la più comune, è utilizzata dall'utente per accedere ad un computer;
- **Computer-computer**, utilizzato per accedere ad un computer remoto, per esempio per accedere ad un sito Web;
- **Utente-utente**, tipo di autenticazione rara, ma fattibile.

Autenticazione utente-computer

Relativamente al tipo di autenticazione **utente-computer**, possiamo notare una successiva distinzione:

- Basata su **conoscenza di segreti** prestabiliti e precondivisi, per esempio *password* e *pin*;
- Basata su **possesso di dispositivi magnetici o elettronici**, per esempio *carte magnetiche*, *smart-card* e *smart-token*;
- Basata su **biometria** di caratteristiche fisiche dell'utente, per esempio *impronte* ed *iride*.

Autenticazione basata sulla conoscenza di segreti

Il tipo di autenticazione basato sulla **conoscenza di segreti** si basa sull'implementazione di una password. Esistono 4 tipologie di **rischio** per una password:

- **Guessing**, ovvero indovinare la password stessa. Legate al concetto di *guessing*, esistono i seguenti attacchi:
  - Attacco **standard**, attacchi con una matrice sociale basati sulle parole maggiormente legate all'utente stesso (e.g. *hobby*, *nomi parenti*, *compleanno*, *indirizzi*);
  - Attacco **dizionario**, per cui vengono provate tutte le parole presenti all'interno di un dizionario, talvolta arricchendo il tutto con regole che ricalcano possibili scelte dell'utente (e.g. *parola al contrario, 0 al posto di "o", 1 al posto di "i"*);
  - Attacco **bruteforce**, per cui vengono provate *esaustivamente* tutte le parole costruibili in un dato vocabolario. Abbiamo notato, grazie ai *mini-challenge*, che una password di 8 caratteri è una soglia realistica.

Quali sono le contro misure rispetto quest'ultimi 3 attacchi? Esistono 3 diverse misure per gli attacchi descritti, ovvero:

- **Controllo sulla password**, attraverso cui il sistema controlla che la password non sia banale (e.g. *lunghezza*).
- **Controllo sul numero inserimenti**, secondo cui il sistema limita i tentativi di login per cui pena è il blocco del dispositivo (e.g. *dispositivo mobile*).
- **Uso di CAPTCHA** (acronimo di "*Completely Automated Public Turing Test To Tell Computers and Humans Apart*") è essenzialmente un *turing-test* in grado di riconoscere l'umano dalla macchina ed informare la macchina rispetto questa informazione. Rispetto il captcha, esiste un recente algoritmo basato su Google Street View viola il 99% delle captcha alfanumeriche.

Si noti che spesso, in alcuni dispositivi, vengono adottate una combinazione di quest'ultime misure.

- **Snooping**, quindi indovinare la password *sbirciando*;
- **Spoofing**, ossia scoprire la password tramite un falso (e.g. *fake-login*);
- **Sniffing**, ovvero scoprire la password a partire da un *intercettazione*;

Chiaramente per trovare la giusta password è necessario *bilanciare mnemocità e complessità* di quest'ultima. A partire da questo, esistono le seguenti **implicazioni**:

- Non usare parole del dizionario;
- Usare almeno 8 caratteri;
- Non usare la stessa password per autenticazioni diverse, altrimenti una sola compromissione comprometterebbe altri sistemi.

Tuttavia, potrebbe risultare difficile ricordare tutte le nostre password, motivo per cui nascono dei sistemi cui scopo è conservare al meglio tutte le nostre password (e.g. *all'interno del browser*).

Autenticazione basata sul possesso

Attraverso questo secondo tipo di autenticazione, l'*obiettivo* è quello di convalidare l'identità dell'utente a partire possesso di un determinato oggetto, tipicamente *magnetico* o *elettronico*, che può memorizzare un informazione sensibile;
• Informazione su carta magnetica interamente leggibile;
• Informazione su carta elettronica leggibile coerentemente con interfaccia funzionale;
Oggetti più diffusi per questo scopo sono le *smart-card* e *smart-token*.
Uno *smart-token*, a differenza di una *smart-card*, ha un interfaccia di *I/O utente*, mentre una

smart-card ha *I/O macchina*; inoltre, uno smart token può avere un pulsante o una tastiera per immettere il pin.

Autenticazione basata sulla biometria

Attraverso questo terzo tipo di autenticazione, l'*obiettivo* è quello di convalidare l'identità dell'utente a partire possesso di una determinata **caratteristica personale ed univoca**.
Si basa sull'assunzione della natura che ognuno di noi possegga caratteristiche fisiche univoche, per esempio rispetto le impronte digitali. Si noti che lo stesso non è vero rispetto le caratteristiche comportamentali, le quali possono non essere uniche.

L'autenticazione basata sulla biometria è **meno accurata** rispetto i precedenti due modi, basati sul possesso e conoscenza.

Se una lettera è un concetto e può essere rappresentato facilmente secondo un banale codice ASCII, un impronta digitale non può essere inteso come un concetto bensì come un *oggetto reale*, il quale deve essere dicretizzato secondo determinate scelte.
Un impronta digitale viene rappresentata secondo il campionamento delle minuzie (i.e. i punti di biforcazione) all'interno dell'impronta.
Chiaramente un dito all'interno

Sappiamo che due campioni biometrici non sono mai uguali, motivo per cui alla registrazione dell'impronta dobbiamo registrare più di una volta quest'ultima, creando un template: una "*media*" del campionamento per la nostra impronta.
L'autenticazione avviene a partire dal confronto tra l'impronta "*live*" ed il *template*; tuttavia, se il solo cambio di un bit all'interno dell'hash di una stringa di bit fa sì che il codice risultato sia completamente diverso, il riconoscimento dell'impronta digitale prevede una **tolleranza** rispetto l'impronta live presentata, il che rende evidente la poca fiscalità del riconoscimento.
Il motivo è dato dal fatto che ogni campione biometrico è diverso dall'altro e non esiste un modo per rappresentare un campione biometrico in modo inattaccabile.
Questa fiscalità è presente all'interno dell'autenticazione per conoscenza di una password.

Perchè l'autenticazione basata sulla **biometria è sempre accompagnata da una password**? Essenzialmente per i seguenti tre motivi:

- Rappresentare un campione biometrico può essere imprecisa, altalenante;
- Non è possibile modificare un informazione biometrica, il che rende questa meno robusta;
- Un impronta viene "*lasciata*" dappertutto al nostro tocco di un oggetto.

> Soluzione al terzo problema soprastante potrebbe essere "*Finger Vein*", un sistema sensoristico che riconosce lo schema dei capillari sanguigni all'interno del polpastrello.

> Possiamo classificare le impronte digitali secondo tre tipologie: (i) **loop**, la più diffusa fra le tre, (ii) **arch**, la meno diffusa e (iii) **whorl**, la seconda più diffusa dopo loop.

Protocolli di sicurezza storici

*Appunti di **Giuseppe Pitruzzella** - Corso di Internet Security @ DMI, UniCt*

Il protocollo Needham-Schroder

Il protocollo Needham-Schroder è basato su crittografia **asimmetrica** e su $Alice$ e Bob, i quali desiderano auteticarsi all'interno di una rete ostile Dolev-Yao. In questo scenario, esiste un infrastruttura *PKI* secondo cui tutti gli agenti conoscono le rispettive chiavi pubbliche.
Cosa aggiunge il protocollo Needham-Schroder rispetto un protocollo basilare? Il protocollo Needham-Schroder aggiunge essenzialmente due cose in più rispetto un protocollo basilare: (i) la **mutualità**, per cui $Alice$ può autenticare Bob e viceversa, e (ii) la **freshness**.

> [$Esame$] Com'è possibile aggiungere la freshness ad un protocollo basilare per l'autenticazione?

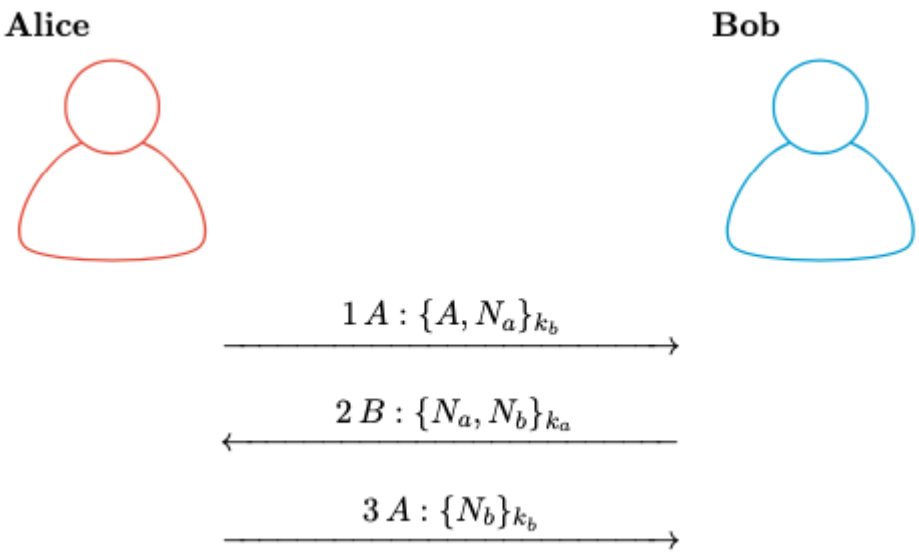
Il protocollo propone una schema *simile* al **protocollo basilare** per l'autenticazione, in cui $Alice$, per autenticare Bob, invia un messaggio a Bob cifrato con la chiave pubblica di quest'ultimo. Poiché l'unico a poter leggere il contenuto del messaggio è solo Bob (attraverso la sua chiave privata), se Bob re-invia il contenuto del messaggio ricevuto ad $Alice$, allora egli dimostra di essere effettivamente Bob.
Il protocollo, in aggiunta all'autenticazione prevede **freshness**, la quale viene implementata secondo la $Nonce$. Per questo motivo, il contenuto del messaggio di cui abbiamo scritto in precedenza diventa effettivamente la $Nonce$ generata da $Alice$.
Inviare la $Nonce$ generata da $Alice$ a Bob indica evitare un possibile *replay-attack*, mentre criptare il messaggio con la chiave pubblica di Bob funge da *challenge* ed è un modo per autenticare Bob.

- $1. Alice \longrightarrow Bob : \{N_A\}_{K_B}$
- $2. Bob \longrightarrow Alice : N_A$

> All'interno di protocollo che aggiunge freshness all'autenticazione, il secondo messaggio inviato da Bob potrebbe essere tranquillamente in chiaro. Questo non deve sorprendere poiché la $Nonce$ una *challenge-response* a cui solo l'effettivo destinatario può rispondere.

Il protocollo basilare che prevede l'aggiunta della freshness all'autenticazione potrebbe concludersi qui, con un secondo ed ultimo messaggio inviato da Bob contenente N_a.
Il protocollo Needham-Schroder però aggiunge altro, ovvero la **mutualità** nell'autenticazione tra $Alice$ e Bob.
Per garantire mutualità, è necessario che non appena Bob riceve il messaggio di $Alice$ (1),

egli non invii esclusivamente N_a, bensì concateni al messaggio anche una sua $Nonce$ (i.e. N_b). A questo punto, Bob esegue l'encrypt del messaggio secondo la chiave pubblica di $Alice$ ed invia il tutto (2).
Il protocollo Needham-Schroder si conclude con un terzo ed ultimo messaggio da parte di $Alice$, in cui dimostra di essere lei inviando N_b a Bob.



Abbiamo trattato la spiegazione del protocollo Needham-Schroder a partire dalla trattazione di un protocollo basilare per l'autenticazione, poi con freshness, poi con mutualità; tuttavia, un passaggio che non abbiamo ancora spiegato al meglio è 1, lo stesso che da inizio al protocollo. All'interno del messaggio al passaggio 1, possiamo notare che oltre ad N_a, vi è anche l'identità di $Alice$ stessa.
Cosa succederebbe se all'interno del messaggio 1, non vi fosse l'identità di $Alice$?
A prima vista, potremmo pensare che l'identità di Alice posta all'interno del messaggio sia *futile* e che Bob possa conoscere il destinatario per il messaggio 2 analizzando l'intestazione del pacchetto ricevuto. Tuttavia, non aggiungere l'identità potrebbe far sì che un attaccante $Charlie$, cambi l'intestazione del pacchetto a livello di trasporto, generando un attacco. In questo modo, infatti, Bob legge l'intestazione precedente ed invia a $Charlie$ il messaggio contenente $\{Nonce_A, Nonce_B\}$.

$$1.\ Alice \longrightarrow Bob\ :\ \{Alice, N_A\}_{k_B}$$

$$Charlie\ intercetta\ il\ messaggio$$

$$1'.\ Charlie \longrightarrow Bob\ :\ \{Alice, N_C\}_{k_B}$$

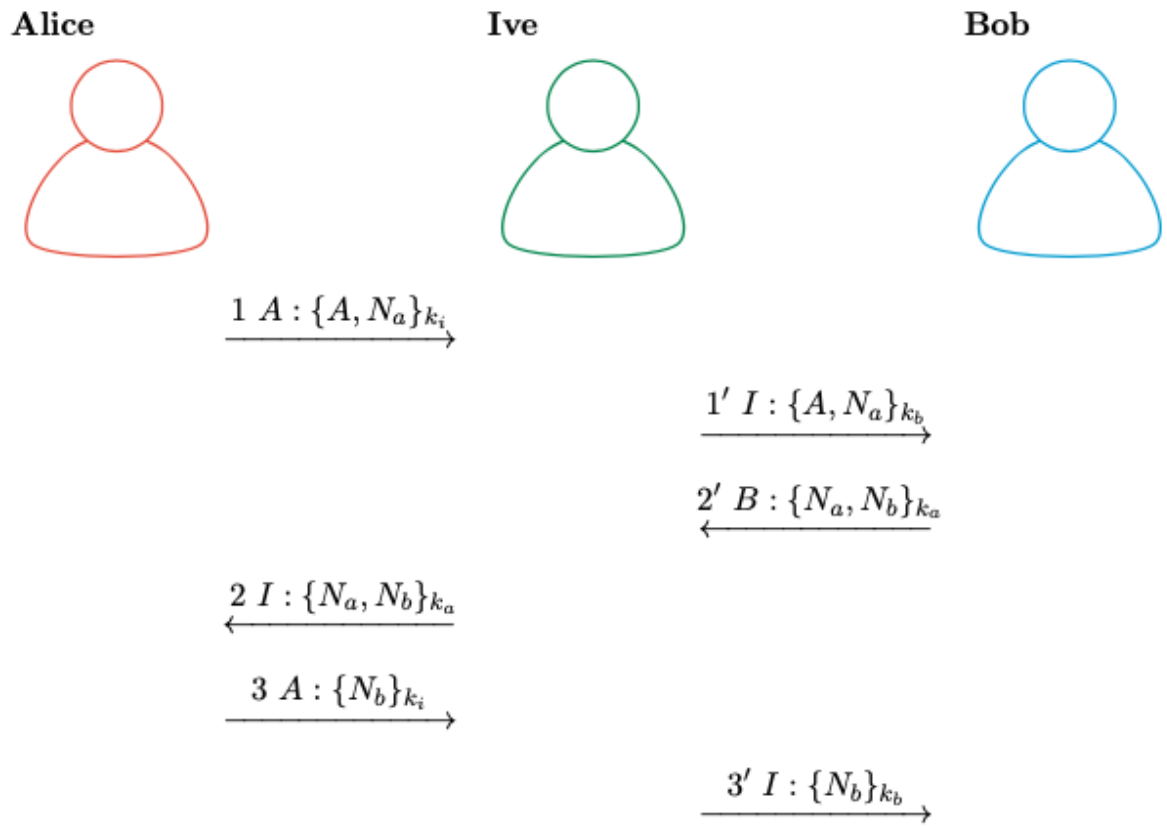
$$2.\ Bob \longrightarrow Alice\ :\ \{N_A, N_B\}_{k_A}$$

Quali sono le conseguenze rispetto l'applicazione del protocollo? All'applicazione del protocollo, $Alice$ è in grado di autenticarsi con Bob e viceversa; inoltre, le $Nonce$ utilizzate e rimaste segrete, possono essere utilizzate per codificare i messaggi successivi tra $Alice$ e Bob.

Attacco

Ad un certo punto, nel 1995, qualcuno osserva l'esistenza di almeno uno scenario di attacco, l'**attacco di Lowe**.
Ipotizziamo che $Alice$ voglia autenticarsi con Ive, motivo per cui da inizio al protocollo un

messaggio contenente la sua identità e $Nonce_{Alice}$ (1).
Ive si comporta da Dolev-Yao, infatti, legge N_a all'interno del messaggio ricevuto da $Alice$, ed invia un messaggio Bob, inviando $\{Alice, N_a\}_{k_b}$ a Bob, applicando il protocollo NS con Bob ma impersonando $Alice$ $(1^{'})$.
A questo punto, Bob replica con un messaggio ad $Alice$ contenente la $Nonce$ di Alice (i.e. N_a) e la sua $Nonce$ (i.e. N_b), codificando la chiave pubblica di $Alice$ (i.e. k_a) $(2^{'})$.
Ive, quindi, interrompe l'invio del messaggio da parte di Bob ed intercetta il messaggio, ma non essendo in grado di rompere la cifratura non può far altro che continuare l'invio del messaggio verso $Alice$ (2).
$Alice$ riceve il messaggio da parte di Ive ed autentica Ive, notando anche la challenge-response generata da Bob, ovvero N_b. Si noti che al dì là della notazione utilizzata, $Alice$, in realtà, pensa che la N_b appartenga ad Ive, e non Bob.
$Alice$ continua ad eseguire il protocollo inviando ad Ive la "sua" $Nonce$, cifrando con la chiave di Ive (3).
A questo punto, Ive replica il messaggio ricevuto da $Alice$ a Bob e segna la fine del protocollo $(3^{'})$.
Ive, in definitiva, ottiene sia $Nonce_{Alice}$ che $Nonce_{Bob}$; in particolare, Ive si autentica con Bob impersonando $Alice$.
Si noti che la conoscenza di $Nonce_{Alice}$ da parte di Ive, non è un attacco alla confidenzialità. L'attacco avviene nel momento in cui Ive scopre $Nonce_{Bob}$, generata da Bob per $Alice$.



> Rispetto l'attacco di Lowe, la vittima diretta che desume qualcosa di erroneo è Bob, la vittima rispetto il furto d'identità è $Alice$.

Fix

Quali sono i possibili **fix** rispetto l'**attacco di Lowe**? Un possibile *fix* riguarda la modifica del protocollo stesso e prevede che all'interno del messaggio 2, oltre ad N_a ed N_b, venga inserito anche l'identità del mittente, ossia Bob. Esistono altri *fix*.

Rispetto l'attacco di Lowe, Needham arrivò ad affermare che l'attacco non avesse senso, affermando che il modello di attaccante su cui si basa l'attacco è diverso rispetto al modello di attaccante che Needham e Schroder avevano *dato per implicito* all'interno della documentazione del protocollo. D'altra parte, la nascita del protocollo è precedente a quella relativa al modello di attaccante Dolev-Yao.
Da questo evento, non esistette più un protocollo che non presentasse una sezione relativa al modello di attaccante all'interno della sua documentazione.

> [$Osservazione$] Needham affermò che il modello di attaccante (dato per implicito) prevedesse che ogni persona si fidasse dell'altra, tuttavia questa affermazione si scontra con l'utilizzo dell'identità di Alice all'interno del punto 1, la quale era necessaria affinché la Nonce di Bob non finisse in mani sbagliate.

> Per questo motivo, possiamo concludere con due diverse affermazioni: (i) la risposta data da Needham rispetto il modello di attaccante utilizzato è vera e quindi l'aggiunta di Alice è stata un no-sense (impossibile, dati gli autori), oppure (ii) il contrario, per cui il modello di attaccante descritto è stato effettivamente utilizzato e Needham e Schroder hanno dimenticato di aggiunere l'identità di Bob all'interno del messaggio al punto 2.

Il protocollo Woo-Lam

Il protocollo Woo-Lam è basato su crittografia **simmetrica** ed ha l'obiettivo di autenticare $Alice$ con Bob e non necessariamente viceversa. Poiché il protocollo è basato su crittografia simmetrica, ciascun utente condivide la propria chiave simmetrica (i.e. a lungo termine), con una forte assunzione circa la protezione offerta da parte del server TTP (acronimo di *Trusted Third Party*).
L'utilizzo del TTP all'interno dei passaggi nel protocollo è del tutto normale, poiché laddove vi sia un crittotesto creato con una chiave a lungo termine del mittente, il ricevente non può che affidarsi al server TTP per il decrypt del messaggio.

$$1.\ A \longrightarrow B\ :\ A$$
$$2.\ B \longrightarrow A\ :\ N_b$$
$$3.\ A \longrightarrow B\ :\ \{N_b\}_{k_a}$$
$$4.\ B \longrightarrow TTP\ :\ \{Alice, \{N_b\}_{k_a}\}_{k_b}$$
$$5.\ TTP \longrightarrow B\ :\ \{N_b\}_{k_b}$$

Il passaggio (1) prevede l'invio di un messaggio contenente l'identità di $Alice$, da parte di $Alice$ verso Bob.
Bob, quindi, suppone che all'altro capo vi possa essere $Alice$, motivo per cui emette un challenge rappresentato da una $Nonce$.
Per accettare la sfida, al passo 3, $Alice$ non può che cifrare il messaggio con la sua chiave a

lungo termine.
Al passo successivo (4), Bob ricorre al TTP, inviando un messaggio contenente l'identità di $Alice$ ed $\{N_b\}_{k_a}$, il tutto cifrato con la chiave a lungo termine di Bob (i.e. $\{Alice, \{N_b\}_{k_a}\}_{k_b}$)
Il TTP è programmato per leggere l'identità della prima parte del messaggio (i.e. $Alice$), cercare la chiave per l'identità ed effettuare il decrypt della seconda parte del messaggio utilizzando la chiave trovata nel database. Il *risultato del decrypt* (i.e. $\{N_b\}$) viene cifrato con la chiave a lungo termine di Bob ed inviato a quest'ultimo all'interno del punto (5).
A questo punto, Bob può effettuare il decrypt del messaggio inviato dal TTP, ed autenticare $Alice$ se ciò che è contenuto nel messaggio è uguale alla $Nonce$ generata da egli al punto 2.

Attacco

Un **attacco** rispetto il protocollo Woo-Lam esiste ed è basato su *parallel-session*, per cui un attaccante $Charlie$ da inizio a due sessioni con Bob, una in cui agisce nel rispetto del protocollo, l'altra in cui impersona $Alice$.
$Charlie$ da inizio al tutto inviando due messaggi a Bob, una in cui afferma di essere $Charlie$, una in cui afferma di essere $Alice$. Per questo motivo, Bob replica con due $Nonce$, N_c ed $N_c{}'$, che invia rispettivamente a $Charlie$ ed $Alice$.
A questo punto, $Charlie$ invia *due volte* lo stesso messaggio a Bob, un messaggio contenente la risposta al challenge, ovvero $N_{b\,k_c}$.
Bob riceve i due messaggi e deduce che i due messaggi siano relativi ad $Alice$ e Bob, motivo per cui crea $\{Alice, \{N_b\}_{k_c}\}_{k_b}$ e $\{Charlie, \{N_b\}_{k_c}\}_{k_b}$ per poi contattare il TTP.
Il TTP cerca la chiave di $Alice$ e $Charlie$ all'interno del database e replica a Bob con il risultato ottenuto, il quale sarà (i) qualcosa di incomprensibile nel caso di decrypt con k_a di $\{N_b\}_{k_c}$ (i.e. $N_b{}''$) ed (ii) N_b nel caso di decrypt con k_a di $\{N_b\}_{k_a}$.
Bob non riconosce $N_b{}''$, motivo per cui fa *abort* per la determinata sessione ma riconosce sulla seconda sessione N_b, il quale era associato ad $Alice$, motivo per cui autentica quest'ultima.
Si noti che (i) l'attacco è possibile poiché Bob non è in grado di distinguere le due sessioni e che (ii) $Charlie$ poteva anche impersonare qualcun'altro anche nel passo $1'$.

1. $C \rightarrow B : A$

    I. $C \rightarrow B : C$

2. $B \rightarrow A : N_b$

    II. $B \rightarrow C : N_b^{'}$

3. $C \rightarrow B : \{N_b\}_{k_c}$

    III. $C \rightarrow B : \{N_b\}_{k_c}$

4. $B \rightarrow TTP : \{A, \{N_b\}_{k_c}\}_{k_b}$

    IV. $B \rightarrow TTP : \{C, \{N_b\}_{k_c}\}_{k_b}$

5. $TTP \rightarrow B : \{N_b^{''}\}_{K_b}$

    V. $TTP \rightarrow B : \{N_b\}_{K_b}$

> Nel messaggio (2), $Charlie$ intercetta N_b; nel messaggio (3), Bob crede che il messaggio ricevuto provenga da $Alice$.

Fix

All'interno dell'attacco, il problema è legato al messaggio contenuto nel punto (5), motivo per cui un **fix** all'attacco è citare l'interlocutore che vogliamo autenticare, quindi $Alice$ e $Charlie$, rispettivamente nel punto (5) e $(5^{'})$.

$$5.\ TTP \longrightarrow B\ :\ \{Alice, N_b^{''}\}_{k_b}$$

$$5^{'}.\ TTP \longrightarrow B\ :\ \{Charlie, N_b\}_{k_b}$$

In questo modo, Bob, non riconoscendo $N_b^{''}$, eseguirebbe un *abort* nel punto (5); tuttavia, Bob, esegue *abort* anche per $(5^{'})$, poiché la $Nonce$ associata a $Charlie$ era $N_b^{'}$, e non N_b.

IPSec

*Appunti di **Giuseppe Pitruzzella** - Corso di Internet Security @ DMI, UniCt*

IPSec può essere inteso come un protocollo **IP sicuro** (i.e. un alternativa ad IP), motivo per cui possiamo aspettarci che utilizzi *tecniche crittografiche*.

> Qual è la differenza tra SSL/TLS e IPSec?
> SSL e TLS si trovano a livello logico superiore, e funzionano anche su una rete insicura mentr IPsec vuole costruire una rete sicure, in altre parole vuole costruire sicurezza.

Anche IPSec, come quasi tutti i protocolli di sicurezza, ha un problema: la distribuzione di una chiave condivisa tra i partecipanti. Oggigiorno, è possibile risolvere questo problema *tipicamente* attraverso l'utilizzo della crittografia asimmetrica, necessaria a mettere in sicurezza una **chiave simmetrica di sessione**, utilizzata per cifrare il traffico in ambedue le direzioni.

Per utilizzare IPSec è sufficiente che almeno gli interlocutori siano compatibili a quest'ultimo. In altre parole, non è necessario che i nodi intermedi tra i due interlocutori siano compatibili ad IPSec. A questa affermazione conseguono le seguenti implicazioni:

- L'utilizzo di IPSec non impone di effettuare cambiamenti sulla rete.
- Un nodo intermedio non deve preoccuparsi del payload all'interno del pacchetto; tuttavia, nel caso in cui il nodo sia malevolo, egli non può leggere il contenuto poiché segreto mediante ESP.

> IPSec è opzionale per IPv4 e mandatorio per IPv6.

IPSec comprende una suite di protocolli per garantire segretezza, autenticazione ed integrità a livello IP, ovvero:

- **IKE**, acronimo di *Internet Key Exchange*, è il protocollo in grado di distribuire una chiave di sessione,
- **AH**, utilizza la chiave di sessione fornita da *IKE* per ottenere autenticazione ed integrità;
- **ESP**, utilizza la chiave di sessione fornita da *IKE* per ottenere segretezza ed *opzionalmente* anche autenticazione.

SA e SAD

IPSec si basa sul concetto di **SA** (acronimo di *Security Association*), una *relazione unidirezionale* fra mittente e destinatario cui relazione sancisce le scelte crittografiche per il

determinato traffico. In una connessione bidirezionale saranno necessarie 2 SA. Una SA può essere intesa come una policy.
Una SA stabilisce come utilizzare AH e/o ESP, il tutto attraverso almeno 3 parametri (tipicamente 6), ovvero:

1. **SPI** (acronimo di *Security Parameter Index*), un indice utile rispetto il concetto di *SAD*.
2. **Destination IP**, solo *unicast*;
3. **ID** del protocollo, il quale può essere uguale ad AH e/o ESP.

I nodi intermedi che supportano IPSec ricevono con il pacchetto anche un valore SPI, ovvero l'identificativo per la determinata SA. Un nodo che supporta IPSec possiede un database (i.e. **SAD**) all'interno del quale sono conservate tutte le SA supportate dal determinato nodo.
Una **SAD** è **formata da**:

- **Info AH**, informazioni sull'algoritmo di autenticazione;
- **Info ESP**, informazioni sull'algoritmo di codifica;
- **Lifetime**, ossia la durata per la determinata SA.

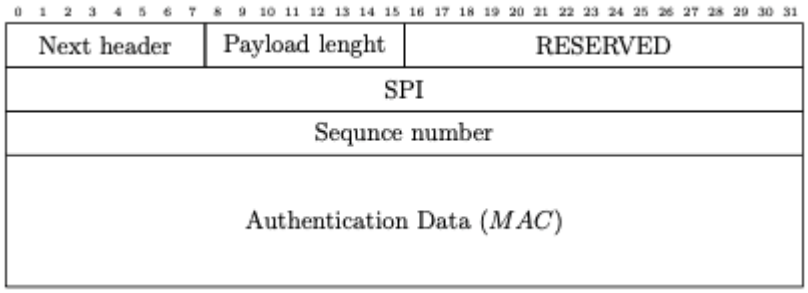
Per **esempio**, supposto l'arrivo di un pacchetto (e.g. AH), il router intermedio che supporta IPSec, nota il valore SPI contenuto all'interno del pacchetto, quindi cerca il valore all'interno del suo database per capire in che modo in cui deve essere trattato il determinato pacchetto. Se il nodo non trova il valore SPI all'interno del suo SAD, allora quest'ultimo inoltra semplicemente il pacchetto.

AH

Cos'è AH? AH è un frammento aggiunto ad ogni pacchetto IP, necessario per ottenere *autenticazione* ed *integrità*. Ricordiamo che non esiste integrità senza autenticazione. Le 2 proprietà sono garantite attraverso l'utilizzo di una *funzione hash* (i.e. `MAC`, *Message Autentication Control*) calcolato secondo una chiave di sessione fornita - precedentemente - secondo IKE, ed il messaggio stesso.
Il frammento AH viene applicato all'intero pacchetto esclusi i campi variabili dell'header IP, modificabili dai nodi intermedi (e.g. ToS, Flags, Fragment Offset, TTL, ...).

Utilizzare IPSec (i.e. AH) previene **attacchi** come:

- *Replay Attack* (i.e. *attacchi di replica*), il quale si lega al concetto di numero di sequenza per un pacchetto; quest'ultima azione, godendo della proprietà di integrità, diventa adesso affidabile. In particolare, l'attacco è impedito secondo l'utilizzo di una finestra di ricezione, da $N - W$ ad N, per cui ogni pacchetto con numero di sequenza minore di $N - W$, non autenticato o giù presente, non viene considerato.
- *IP Spoofing*, per cui il tentativo di alterazione dell'IP viene scoperto dal ricevente nel momento in cui verifica il controllo di integrità per il pacchetto stesso.



AH può essere utilizzato secondo 2 diverse **modalità**:

- **Trasporto**, secondo cui il `MAC` protegge dati e campi non variabili dell'header IP. In questo modo, viene garantita l'autenticazione punto-punto tra i due interlocutori. In altre parole, la modoalità di trasporto, proteggendo il payload di IP ed alcuni campi del suo header, fornisce protezione per i protocolli a livello superiore (e.g. TCP).
- **Tunnel**, secondo cui il `MAC` protegge l'intero pacchetto IP, da considerare come il payload di un nuovo pacchetto IP esterno. In questo modo, viene garantita un autenticazione intermedia legata ai due gateway che collegano $Alice$ e Bob, motivo per cui quest'ultimi due possono anche non utilizzare IPSec. Si noti che, in uno scenario in cui gli interlocutori non utilizzano IPSec, quest'ultimi inviano normali pacchetti IP che vengono successivamente incapsulati in pacchetti IPSec dai gateway.

Figura 7.5: Pacchetto orginario IPV4

Figura 7.6: AH in modalità trasporto IPV4

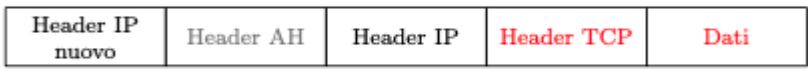
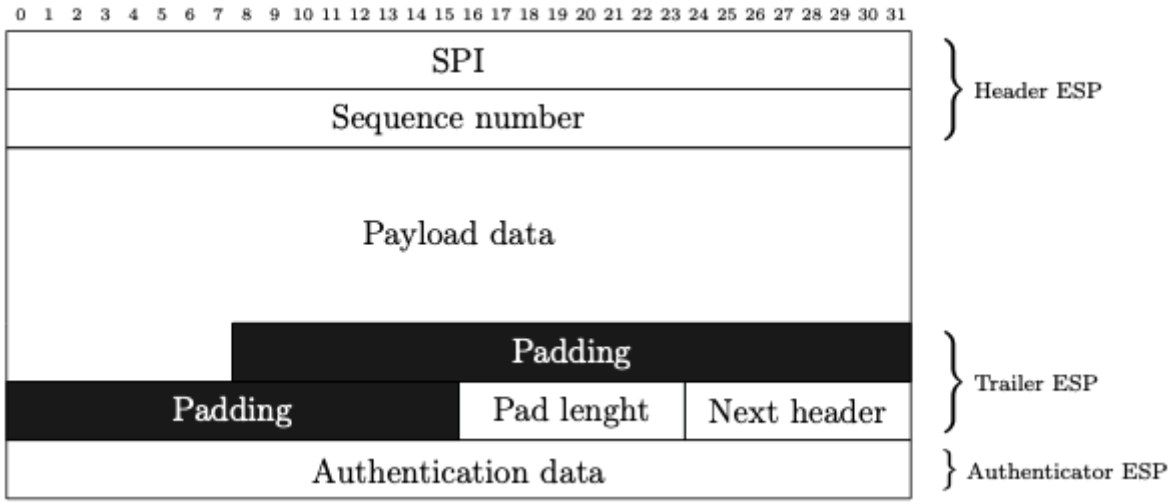


Figura 7.7: AH in modalità tunnel IPV4

> [$Esame$] Qual è un esempio pratico rispetto l'utilizzo della crittografia? Un esempio pratico è IPSec.

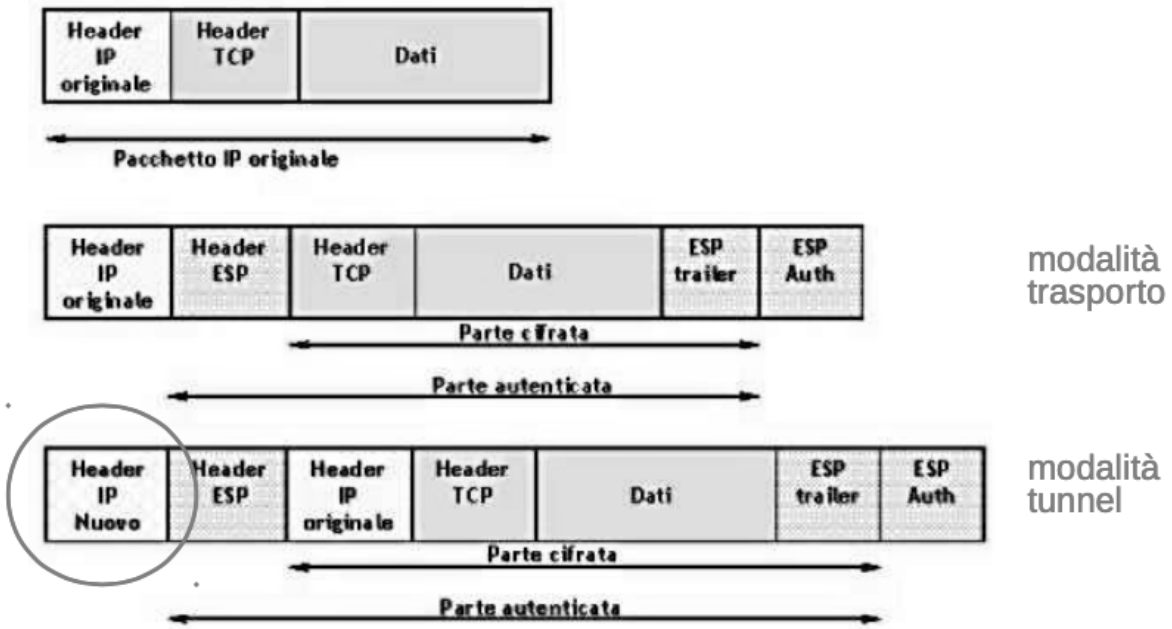
ESP

Cos'è ESP? ESP (acronimo di *Encapsulating Security Payload*) è necessario per fornire segretezza per il contenuto. Il payload per il frammento ESP è un *cypher-text*.



ESP può essere utilizzato secondo 2 diverse **modalità**:

- **Trasporto**, secondo cui la *cifratura* (i.e. ESP) protegge tutto ciò che è al di sopra di egli, ovvero: *TCP*, *Dati* ed *ESP Trailer*. In questo modo, viene garantita segretezza punto-punto tra i due interlocutori. Opzionalmente è in grado di autenticare fino ad ESP stesso. In altre parole, l'utilizzo di ESP in modalità di trasporto permette la creazione di una VPN tra $Alice$ e Bob.
- **Tunnel**, secondo cui la *cifratura* (i.e. ESP) protegge tutto ciò che è al di sopra di egli, ovvero: *IP*, *TCP*, *Dati* ed *ESP Trailer*. Un nuovo header IP incapsula ESP. In questo modo, viene garantita segretezza intermedia legata ai due gateway che collegano $Alice$ e Bob, motivo per cui quest'ultimi due possono anche non utilizzare IPSec. Si noti che, in uno scenario in cui gli interlocutori non utilizzano IPSec, quest'ultimi inviano normali pacchetti IP che vengono successivamente incapsulati in pacchetti IPSec dai gateway. In altre parole, l'utilizzo di ESP in modalità tunnel permette la creazione di una VPN tra vari router di varie reti aziendali.



Combinare AH ed ESP

Talvolta sono possibili **combinazioni di SA** (e.g. combinazione ESP ed AH). Si noti che, come definito all'interno di un RFC, ogni nodo compatibile con IPSec, deve supportare 4 tipi di combinazioni SA. Alcune combinazioni sono:

1. Sicurezza **punto-punto**, ovvero l'utilizzo di AH ed ESP in *trasporto*, quindi tra i 2 interlocutori.
2. Sicurezza **fra intermediari**, ovvero l'utilizzo di AH ed ESP in *tunnel*, quindi l'utilizzo di un tunnel tra i *gateway* che collegano le 2 reti.
3. Sicurezza **punto-punto** e **fra intermediari**, ossia la combinazione delle precedenti, quindi la combinazione di un tunnel tra i due interlocutori (i.e. *vpn punto-punto*), a sua volta protetta da un tunnel tra i gateway che collegano le 2 reti.
4. Sicurezza **punto-punto più tunnel tra host ed intermediario**, ossia la combinazione di un tunnel tra i due interlocutori (i.e. *vpn punto-punto*), a sua volta protetta da un tunnel tra il mittente ed il gateway.

IKE

IKE, il protocollo per la distribuzione di una chiave di una chiave di sessione in IPSec, è la somma di:

- **Oakley**: è una variante di Diffie-Hellmann potenziata (livello applicazione), fornisce chiave (iniziale) di sessione.
- **ISAKMP** (Internet Security Association and Key Management Protocol, livello trasporto) è ciò che espande il segreto inziale in un segreto più lungo, in termini di bit. In altre parole, crea ulteriori chiavi di sessione.

Intrusion Detection

*Appunti di **Giuseppe Pitruzzella** - Corso di Internet Security @ DMI, UniCt*

Introduzione

Iniziamo la trattazione del capitolo relativo all'intrusion detection attraverso la **definizione** stessa di intrusione. Un **intrusione** può essere definito come un **processo** all'interno di una macchina vittima, nata a partire dall'exploit di una vulnerabilità presente all'interno della macchina stessa; il processo potrà svolgere determinate attività malevole nei limiti dei suoi privilegi, i quali possono essere limitati o illimitati.

Conguentemente alla possibile esistenza di un intruso all'interno del nostro sistema, nascono diverse **metodologie** per **nascondere** e - d'altra parte - **rilevare** la loro presenza; in particolare, poiché non possiamo godere di un metodo assoluto e deterministico rispetto il rilevamento di un intruso, gli unici mezzi a nostra disposizione sono basati su euristiche (a cui consegne la possibilità di *falsi-positivi*).

E' importante notare le **differenze** rispetto un **intruso** ed un **virus**: se un virus ha tipicamente scopo distruttivo per il sistema vittima, un intruso nasce esclusivamente come un processo con intenzioni malevole; esiste il caso in cui un intenzione malevola per l'intruso, sia l'effettiva installazione di un virus all'interno del sistema.

> Un intruso non è né un virus né un worm, ma un processo con intenzioni malevoli.

Qual è la **terminologia** legata al concetto di Intrusion Detection?

- **SIEM**, acronimo di *Security information and event managment*, un sistema di monitoraggio il quale permette una visione *compatta* del traffico degli eventi di rete (e.g. monitorare gli accessi logici dei sys admin). Esistono la possibilità di applicare delle logiche ad un SIEM (e.g. molti eventi di un certo tipo possono essere sintomo di un intrusione), motivo per cui esiste la possibilità di utilizzare un SIEM come IDS o IPS. Un SIEM può essere acquistato o affittato come servizio. E' possibile, altrimenti, un servizio di monitoraggio esterno di tipo SIEM, i quali vengono operati in posti hardenizzati all'interno di **SOC** (e.g. Lutech). Inoltre, relativamente ad un SOC, esistono anche servizi di risposta agli incidenti. Un esempio di SIEM è **QRadar** o **Splunk**.
- **FIM**, acronimo *File Integrity Monitoring*, un particolare tipo di prodotto software che permette il controllo di integrità per dei file. Un esempio di FIM è **AIDE**, il quale può essere inteso come un particolare tipo di IDS, cui *logica fondamentale di funzionamento* è l'analisi dell'integrità per dei file. Un analisi dell'integrità può esistere a partire dall'analisi del
- **IDS**, acronimo di Intrusion Detection System, può essere inteso come un tool o un modulo aggiuntivo nel software. Un esempio di IDS è *SNORT* o *Suricata*.
- **IPM**, acronimo di Intrusion Prevention System;

- **AD**, acronimo di Anomaly Detection, un termine più generale e moderno per indicare un IDS.

> Un **intrusione** può essere definita *anche* come l'ottenimento illecito di privilegi superiori a quelli posseduti. In altre parole, è un intrusione se un attaccante guadagna un processo all'interno del mio sistema, ma è *anche* un intrusione se un attaccante - con il suo processo all'interno del mio sistema - riesce in un escalation dei privilegi.

Tecniche di intrusione

Quali sono le **tecniche di intrusione**? Alcune tecniche di intrusione sono: (i) **violazione dell'autenticazione** (e.g. attacchi a dizionario o analisi del contesto, rispettivamente facenti parte di tecniche standard e non), (ii) l'**intercettazione di informazioni sensibile** (e.g. analisi del traffico) e (iii) l'utilizzo di **software nocivi** (e.g. worm *Morris*).

Rilevamento dell'intrusione

Quali sono i modi per **trattare un intrusione**? Il trattamento dell'intrusione si basa su **Intrusion Detection** ed **Intrusion Management**.

Cos'è l'**Intrusion Detection**? L'**Intrusion Detection** può essere intesa come l'operazione per cui è necessario registrare il comportamento del sistema secondo record (i.e. log), ed eseguire l'auditing per quest'ultimi.
Relativamente ai record, esistono: (i) **record nativi** per il deteminato sistema operativo, semplici ed in grado di raccogliere informazioni generali per gli utenti e (ii) **record specifici per il rilevamento**, i quali raccolgono informazioni mirate rispetto il rilevamento.
Il più famoso tipo di *record specifico per il rilevamento* è rappresentato dai **record di Denning**, i quali forniscono informazioni aggiuntive rispetto il *rilevamento* e sono formati da:

- **Soggetto**, ovvero l'utente o il processo che esegue l'azione per mezzo di un comando. Si noti che un soggetto può essere anche un *oggetto*;
- **Azione**, ossia l'operazione svolta dall'utente (e.g.);
- **Oggetto**, ciò su cui viene svolta l'azione;
- **Eccezione**, ovvero se e quale eccezione è stata prodotta in risposta all'azione;
- **Risorse**, ossia le risorse utilizzate (e.g. numero di righe stampate, settori letti, CPU, etc...);
- **Timestamp**, ovvero l'identificativo temporale per l'inizio dell'azione.

Un **esempio** di record di Denning è il seguente:

Soggetto	Azione	Oggetto	Eccezione	Risorse	Timestamp
giamp	esegue	/bin/cp	0	CPU = 00002	11058721678

Soggetto	Azione	Oggetto	Eccezione	Risorse	Timestamp
giamp	scrive	~barba	-1 (i.e. anomalia)	SECTOR=0	11058721680

> I record di Denning, insieme ai record nativi per il SO, forniscono il monitoraggio per il sistema, ovvero i log. I log permettono uno storico, un registro a partire dal quale è possibile l'operazione di auditing, operazione necessaria al rilevamento dell'intrusione.

Quali sono le **tecniche per il rilevamento** dell'intrusione?

- **Tecniche indipendenti dal passato** (i.e. assoluto), su cui si basa - per esempio - il permettere al più 3 inserimenti della password;
- **Tecniche dipendenti dal passato**, a partire dal quale è possibile imparare dal passato ed a partire da questo fare assunzioni sul futuro;

Tecniche di rilevamento, che possono essere indipendenti o dipendenti dal passato, possono essere di tipo:

1. **Statistico** ... . Attraverso il rilevamento statistico Esistono diversi modelli per il rilevamento statistico, tra i quali:
    - **Media e derivazione standard**, calcolati in un arco di tempo, forniscono l'idea del comportamento medio e della variabilità rispetto l'insorgenza di un fenomeno (e.g. con quale frequenza viene effettuato il login nel giorno, nell'ora o nella posizione oppure l'utilizzo delle risorse per un programma).
    - **Operativo**, basato sulla definizione di un limite di accettabilità per un parametro; un *alert* viene ritornato nel momento in cui viene superato il limite (e.g. PIN per il Bancomat).
    - **Multivariazione**;
2. **A regole**, più semplice e rapprentato - per esempio - da *Snort*, il quale si basa un database di regole, fino a 10^4 o 10^6 regole. Un rilevamento a regole è per definizione un rilevamento rigido e fiscale, motivo per cui consegue la nascita di *falsi-negativi*. L'insieme di *falsi-negativi* che possono essere generati è il vero *limite* di *Snort*.

Quali sono le tecniche di protezione contro l'intrusione?

- **Limitare l'automazione**, per cui rilevata l'intrusione, viene . Alcuni esempi sono il blocco della carta o dell'account; tuttavia, un'altra tecnica per limitare l'automazione è talvolta il blocco temporaneo rispetto l'inserimento della password (e.g. pin su smartphone).

- **Honeypot**, risorse fittizie che possano attaccare gli attaccanti. Un esempio di honeypot potrebbe essere una macchina con un nome interessante nel dominio, monitorando quest'ultima con un SIEM ed in particolare con un IDS.

Com'è possibile gestire/prevenire un intrusione? Un intrusione può essere gestita/prevenuta grazie ad un Intrusion Management System, un termine utilizzato dalla direttiva NIST, un regolamento che ci aiuta circa la gestione di un incidente (e.g. intrusione). In linea con la recente direttiva NIST, esistono 4 *fasi* fondamentali per la gestione di un intrusione; un intrusione deve essere: (i) **contenuta**, (ii) **rimossa**, (iii) **riassorbita** e (iv) **punita**.

E' possibile **relazionare il concetto di intrusione all'attacco di negazione di servizio** (i.e. DoS). Un DoS è un attacco che impedisce la normale operatività di un sistema, *degradandola* o *bloccandola* del tutto. Il concetto di *load balancer* fornisce un ottima strategia contro il DoS, il quale prevede l'inoltro della richiesta al server meno "*carico*".
Quali sono le **differenze** tra un intrusione ed una negazione di un servizio?

DoS	Intrusione
Il DoS ha un effetto temporaneo (e.g. max $24h$).	Un processo intruso ha effetti tipicamente permanenti (e.g. una vittima con vulnerabilità non riscontrata potrebbe potenzialmente possedere un intruso per sempre).
Il DoS è immediatamente pubblico, riscontrabile poiché il servizio non è raggiungibile.	Un intrusione spesso non è resa pubblica.
Il DoS ha lo scopo di bloccare il sistema, quindi il blocco delle risorse.	L'intrusione ha lo scopo di sfruttare il sistema attraverso un uso illecito delle risorse.

Esistono 3 **tipologie** di DoS:

- **cDoS**, un DoS di tipo *computazionale*;
- **mDoS**, un DoS relativo alla *memoria*;
- **bDoS**, un DoS legato alla *banda*.

Cos'è un attacco DDoS? Il concetto di DDoS si basa su un insieme di macchine *zombie*, le quali convergono al medesimo *target*.
E' importante notare che non esiste una soluzione control DoS che non vada conformata con una regolazione dell'utilizzo delle risorse da parte del target.
Quali sono le possibile **strategie contro DDoS**?

- **Cookie transformation**, la quale prevede che all'interno di una comunicazione client-server, il server possa guadagnare tempo impegnando computazionalmente il client. L'approccio basato su cookie transformation, si basa sul trasformare una semplice comunicazione client-server, come:

1. $C \longrightarrow S : m_1$
2. $S \longrightarrow C : m_2$
3. $C \longrightarrow S : m_3$

...nella seguente comunicazione:

1. $C \longrightarrow S : m_1$
2. $S \longrightarrow C : m_2\, s$
3. $C \longrightarrow S : m_1,\ m_2\, s,\ H(m_1, m_2\, s)$
4. $C \longrightarrow S : m_2\, c$

In particolare, il server invia un challenge al mittente, un challenge basato su *hash* che il mittente deve soddisfare prima che possa ottenere l'effettiva risposta da parte del server.

- **Captcha** (o *reCaptcha*), basata sul fatto che solo un attività pensate può superare un determinato test.

Firewall

*Appunti di **Giuseppe Pitruzzella** - Corso di Internet Security @ DMI, UniCt*

Introduzione

Cos'è un firewall? Un firewall può essere considerato come un servizio in grado di applicare una policy alle porte, una policy che è desiderabile sia ottimale. Formalmente, un firewall è un dispositivo o un applicativo che controlla il flusso di traffico fra reti con diverse impostazioni di sicurezza.

Qual è il compito di un firewall? Il firewall costituisce una difesa perimetrale a scopo puramente preventivo per l'incidente. Un firewall previene un intruso come un portiere in un condominio previene un intruso attraverso logiche di provenienza e destinazione. Utilizzando un firewall, è desiderabile che l'accesso al sistema sia possibile esclusivamente attraverso le porte gestite dal firewall stesso; tuttavia, ad una cattiva configurazione da parte delle porte per il firewall segue un modo per bypassare il sistema.

Requisiti

Quali sono i **requisiti per un firewall**? E' necessario che ogni firewall soddisfi i seguenti requisiti, ovvero:

1. Tutto il traffico deve passare attraverso il firewall;
2. Deve essere una determinata politica di sicurezza; il corretto funzionamento di un firewall è dipendente dall'applicazione di una corretta policy.
3. Il firewall deve trovarsi in una macchina *hardenizzata*, in letteratura definita come **bastion host**. Un bastion host prevede che all'interno di essa non vi siano servizi aggiuntivi, bensì i soli fondamentali al firewall. In questo modo è possibile maggiore robustezza per quest'ultima.

Policy

Relativamente alle **politiche di sicurezza**, esistono due **tipologie**:

- **Default Deny**, attraverso cui è vietato tutto ciò che non è espressamente permesso (i.e. scrivo cosa puoi fare);
- **Default Permit**, attraverso cui è permesso tutto ciò che non è espressamente vietato (i.e. scrivo cosa non puoi fare);

Limiti

Quali sono i **limiti per un firewall**? Un firewall possiede i seguenti limiti:

1. Non può proteggere da attacchi che hanno il permesso di superare il firewall;
2. Non può proteggere da minacce interne (e.g. all'interno della LAN);
3. Permette una protezione minima rispetto le intrusioni (poiché non un IDS);
4. Può degradare le prestazioni della rete (attraverso filtering e monitoring);
5. Può essere complicata la loro configurazione;
6. Non possono proteggere da attacchi non ancora documentati ai protocolli di sicurezza (e.g. sniffing della transazione).

Tassonomia

Relativamente ai firewall, esiste la seguente **tassonomia**:

Firewall Personale

I **Firewall Personali**, non sono molto diffusi ma consigliabili rispetto dispositivi mobili, i quali possono essere attaccati in una rete interna pubblica. Un firewall personale può essere utilizzato insieme ad un firewall di rete.

Packet-filtering Router

Un **Packet-filtering Router**, prima specializzazione di un firewall personale, è in grado di eseguire il filtraggio dei pacchetti attraverso un serie di regole a livello di rete, quindi relative al semplice IP. Quest'ultima, sostanzialmente, è la differenza rispetto *SNORT*, il quale permette un osservazione più specifica (anche relativa al payload del pacchetto). In particolare, un firewall di questa tipologia analizza i seguenti **campi**:

- **Source IP**;
- **Destination IP**;
- Indirizzi di origine e destinazione rispetto il livello di trasporto, quindi il numero di **porta**;
- Il **protocollo** utilizzato;
- L'**interfaccia** di sorgente e destinazione.

I packet-filtering router hanno il **vantaggio** di essere semplici ma lo **svantaggio** di essere poco espressivi; inoltre, non consentono di gestire informazioni a livello più alto nello stack TCP/IP (non possono essere gestite le funzionalità per l'applicativo, ma esclusivamente l'utilizzo di quest'ultimo)e non consentono autenticazione.

Un **esempio** è possibile:
(i).

Action	Ourhost	Port	Theirhost	Port	Comment
block	*	*	SPIGOT	*	We don't trust these people.

Action	Ourhost	Port	Theirhost	Port	Comment
allow	OUR-GW	25	*	*	Connection to SMTP

In particolare, attraverso il precedente, è possibile notare l'utilizzo di **regole bidirezionali**:

1. Il blocco della comunicazione - attraverso porta ed indirizzo - verso l'host *SPIGOT*;
2. Il permesso al gateway di interagire sulla porta 25, rispetto qualsiasi IP e verso l'esterno (apertura del traffico rispetto il servizio SMTP).

(ii).

Action	Ourhost	Port	Theirhost	Port	Comment
allow	*	*	*	25	Connection to SMTP

In particolare, attraverso il precedente, è possibile notare l'utilizzo di **regole bidirezionale**:

1. Tutte le macchine all'interno della LAN possono collegarsi uscire attraverso la loro porta 25.

(iii).

Action	Ourhost	Port	Theirhost	Port	Flags	Comment
allow	OURHOST	*	*	*		Our outgoing calls
allow	*	*	*	*	ACK	Replies to our call
allow	*	*	*	>1024		Traffic to nonservers

In particolare, attraverso il precedente, è possibile notare l'utilizzo di **regole unidirezionali**:

1. Un insieme di IP può, sostanzialmente, uscire verso l'esterno (qualunque porta);
2. Se il pacchetto possiede flag di ACK, allora può entrare;
3. Possiamo uscire sulle porte maggiori di 1024.

(iv).

Index	Source Address	Source Port	Destination Address	Destination Port	Action	Descrip
1	Any	Any	192.168.1.0	>1023	Allow	...
2	192.168.1.1	Any	Any	Any	Deny	...

Index	Source Address	Source Port	Destination Address	Destination Port	Action	Descrip
3	Any	Any	192.168.1.1	Any	Deny	...
4	192.168.1.0	Any	Any	Any	Allow	...
5	Any	Any	192.168.1.2	SMTP	Allow	...
6	Any	Any	192.168.1.3	HTTP	Allow	...
7	Any	Any	Any	Any	Deny	...

In questo insieme di regole, l'ultima è la **regola di default**. In questo caso il sistema è implementato in modo tale da cercare un matching con la prima regola, altrimenti con la seconda, e così via. In questo modo è implementato l'**utilizzo prioritario delle regole**.

Una specializzazione rispetto un packet-filtering router è rappresentata dai **Stateful-inspection Firewall**, il quale osserva le connessioni. Rappresenta una versione più evoluta ed espressiva rispetto il precedente in quanto permette una visione "*più ad alto livello*" rispetto lo stack TCP/IP. Un firewall di tipo **Stateful-inspection Firewall**, quindi, controlla lo stato della connessione e sfrutta le informazioni provenienti dal livello di trasporto per gestire meglio le applicazioni. Quindi non bisogna semplicemente dire "apertura delle porte maggiori di 1024", ma si può essere più precisi.

Firewall di rete

Application-level Gateway

L'application-level gateway rappresenta un firewall più evoluto, con un software più evoluto che permette essenzialmente l'analisi dell'applicazione stessa; poiché possibile gestire l'applicazione, è possibile anche gestire fonti di autenticazione utente.

Circuit-level Gateway

Il circuit-level gateway si basa su una logica a circuito ed è composto da un insieme di porte relative ad ambedue i lati, aprendo due connessioni vere e proprie.

DMZ

Una DMZ indica una zona demilitarizzata e rappresenta la zona (tipicamente contenente risorse sensibili) circoscritta da un firewall. All'interno della figura sottostante è possibile notare due zone demilitarizzate, una interna ed una esterna; un packet-filtering router viene posto all'esterno rappresentando una difesa primaria.
Quest'ultimo packet-filtering router prevede all'interno una DMZ delimitata attraverso un altro firewall. In questo modo i due firewall possono "*rimpallare*" il traffico.
Seguendo ancora la figura, troviamo un WebServer all'interno della prima DMZ, per cui viene permesso ad esempio che la pagina web del dipartimento sia visibile dall'esterno; il resto del traffico viene inoltrato ad un altro firewall.

Software nocivo

*Appunti di **Giuseppe Pitruzzella** - Corso di Internet Security @ DMI, UniCt*

Introduzione

Quali sono le differenze tra un bug ed un malware?

- Un **bug** può essere inteso come una proprietà inattesa del software;
- Un **malware** è deliberatamente creato per azioni malevoli.

Più in generale, il software nocivo viene creato con l'esplicito scopo di violare la sicurezza di un sistema. Quest'ultimo non necessariamente sfrutta un bug e può giocare su fattori legati all'ingegneria sociale.

Tassonomia

- **Software nocivo Indipendente**
    - **Worm**, il quale può essere definito come "*programma nocivo che infetta altre macchine remote, ciascuna delle quali a loro volta infetta altre macchine remote*". La particolarità rispetto questo primo software nocivo è il suo fattore replicativo. Un worm, d'altra parte, può risiedere all'interno del sistema vittima in modo del tutto indipendente, una caratteristica che ricorda gli intrusi (i.e intrusione). Un worm storico è il **worm di Morris**, il quale riuscì ad infettare 6000 macchine in poche ore. Il worm di Morris era in grado di capire in che direzione andare, riuscire ad andare in quel determinato punto ed autoriprodursi. Il tutto fu possibile attraverso l'utilizzo di tre **bug** noti: (i) l'attacco **known-cyphertext** su file di password, (ii) **BOF** (i.e. buffer-overflow) ed (iii) una **trapdoor** sul programma sendmail. L'attacco di known-cyphertext è un modello di attacco per crittoanalisi cui l'attaccante si assume di avere accesso solo a una serie di testi cifrati, quindi, si tenta di indovinare il *plaintext* (o meglio, la *chiave*) a partire da quest'ultimi crittotesti. Un approccio randomico rispetto la generazione della chiave da parte di un algoritmo di encrypt, previene un attacco known-cyphertext.
        - **Spyware**, una specializzazione dei worm.
    - **Zombie**, ovvero un "*programma nocivo che sfrutta una macchina remota già violata per lanciare nuovi attacchi che difficilmente possono essere ricondotti all'autore dello zombie*". Il frammento di software nocivo è al servizio di un altro malware, il quale rappresenta il suo *controllore centralizzato*. Il concetto di zombie è legato all'attacco *DDoS*, studiato all'interno del precedente capitolo. *Esempi* relativi all'utilizzo di zombie, prevedono la conoscenza di una macchina

violata o violabile, la quale viene utilizzata dall'attaccante per obiettivi che hanno lo scopo di compiere ulteriore danno.

- **Software nocivo per cui è necessario un programma ospitante**
  - **Trapdoor** (o *backdoor*), può essere definita come il "*punto segreto di accesso ad un programma senza le procedure di sicurezza altrimenti previste*". Una backdoor nasce, storicamente, per semplificare il *beta-testing* per un programma che prevede un numero considerevole di condizioni (i.e. if-then-else); grazie ad essa, infatti, erano possibili salti incondizionati verso un blocco estremamente annidato, difficile da raggiungere/simulare altrimenti. Si noti che, formalmente, i salti non sono incondizionati, bensì condizionati da particolari keyword. Le trapdoor diventano una minaccia quando sono utilizzate da programmatori senza scrupoli, al fine di ottenere accessi non autorizzati
  - **Bombalogica**, ovvero un "*frammento di codice di un programma non nocivo pronto ad esplodere quando si verificano certe condizioni*". Una bombalogica è uno specifico malware che esplode nel momento in cui si verificano determinate condizioni, il che indica che il malware è occulto all'interno del sistema, un sistema che funziona correttamente finché non si verificano quest'ultime condizioni. Le determinato condizioni potrebbero essere rappresentate - per esempio - da un certo numero di login effettuati con successo.
  - **Trojan**, ossia un "*un programma utile o apparentemente utile che in fase di esecuzione compie una violazione di sicurezza*". In altre parole, un programma che finge di essere buono nascondendo i suoi interessi malevoli.
  - **Virus**, può essere definito come un "*programma nocivo che viola altri progammi non nocivi, sfruttandoli per propagarsi*". Quest'ultimo programma nocivo sfrutta programmi ospiti e possiede intenzioni malevoli. Un virus potrebbe utilizzare tecniche legate al polimorfismo (e.g. *cambiando firma*) e la criptazione per nascondersi all'interno del sistema vittima. Un **esempio** passato rispetto i virus è legato alle *macro di office*, un modo per automatizzare delle operazioni in *Visual Basic*.

In definitiva, notiamo una **distinzione delicata** rispetto i software nocivi studiati precedentemente:

Tipo	Caratteristiche essenziali	Replicano
Trapdoor	Concede accesso non autorizzato	NO
Bombalogica	Si attiva solo su specifiche condizioni	NO
Cavallo di troia	Ha funzionalità illecite inattese	NO
Virus	Attacca e si propaga tramite qualsiasi mezzo, anche quello fisico	SI
Zombie	Usa una macchina già violata	SI
Worm	Viola macchine ricorsivamente tramite la rete	SI

Struttura tradizionale per un semplice virus

L'immagine sottostante ci propone due tipologie di virus: a sinistra una struttura semplice, a destra una versione compressa.
Entrambi prevedono una **firma**: un marcatore (i.e. codice di riconoscimento) che indica quali file sono stati già infettati dal virus ed identifica il virus stesso.
In particolare, il virus a *sinistra*, prevede 3 *sub-routine* ed un *main*:

- `attach-to-program`, prima routine ad essere azionata dal `main`, appende il virus ad un file random se questo non stato già visionato dal virus stesso.
- `execute-payload`, seconda routine che svolge particolari operazioni sul *payload*;
- `trigger-condition`, terza routine che ritorna true se particolari condizioni sono rispettate;
- `main`, il quale aziona in modo iterativo `attach-to-program` e poi `exeeute-payload` se `trigger-condition` è *true*.

Dove può essere inserito un virus all'interno di un file?

La posizione in cui viene inserito un virus è indipendente dall'effettiva esecuzione del virus stesso sul file. Infatti, conseguentemente all'inserimento di un virus all'interno di un file, le dimensioni per il file vengono modificate, motivo per cui un antivirus può scovare quest'ultimo attenzionando la dimensione per il file stesso. Tuttavia, virus più evoluti possono bypassare il controllo della dimensione da parte di un antivirus, cercando di mantenere inalterata la dimensione dopo l'inserimento del virus (e.g. virus a destra nella figura soprastante).

Brain, un esempio storico di virus

Un virus storico all'interno dei primi sistemi Microsoft è stato Brain, il quale rinominava il nome del disco, era in grado di propagarsi ed opportunamente effetuava una degrado nel disco marcando alcuni settori come danneggiati, rendendo quest'ultimi inutilizzabili.

Rimozione di un software nocivo

Per rimuovere un malware esistono diversi strumenti, tra i quali: *antivirus*, *sistemi immuni* e *software sentinella*.

Antivirus

Gli antivirus rappresentano uno degli strumenti più comuni in grado di rimuovere software nocivo. Un antivirus esegue essenzialmente 3 compiti: l'**individuazione** per un file contenente un virus, **identificazione** rispetto il determinato virus nel file ed **eliminazione** del virus (possibile nei limiti del polimorfismo del virus).
Possiamo distinguere 4 diverse generazioni per gli antivirus:

- **Prima generazione**, basata su un confronto rispetto un database;
    - Scansiona i file alla ricerca di firme contenute nel DB;

    - Controlla le dimensioni per un file rispetto la dimensioni conservate all'interno di un DB.

- **Seconda generazione**, basata sull'utilizzo di euristiche;
    - Ricerca di frammenti di codici associati statisticamente ad un virus;
    - Ricerca di violazione di integrità, associando un checksum (generato attraverso una chiave segreta) ad ogni file.

- **Terza generazione**, basata sulla ricerca di azioni illecite;
- **Quarta generazione**, basata sulle tecniche definite all'interno delle generazioni precedenti. Antivirus utilizzati oggigiorno sono relativi alla quarta generazione. Inoltre, sono programmi a codice chiuso (i.e. commerciali), motivo per cui è difficile stabilire le loro azioni.

> Ad un certo tempo nel passato, un antivirus era percepito come uno strumento in grado di degradare le prestazioni del sistema. L'affermazione è stata vera per un certo tempo, tuttavia non è così già da molto tempo.

> Un ransomware è un software nocivo in grado di cifrare i nostri dati a riposo con una chiave che, se il ransomware è scritto male, può essere ritrovata all'interno del sistema (database), altrimenti non resta che pagare il riscatto.

Sistemi immuni

I sistemi immuni sono un esercizio teorico implementato storicamente attraverso un prototipo degli anni $'90$, un sistema che prevede un antivirus centralizzato ed è per questo in grado di reagire in tempo reale; per questo motivo necessita di un sys admin.

Software sentinella

Un software sentinella è un appendice del sistema operativo in grado di generare warning in caso di accesso a risorse importanti del nostro sistema (e.g. il processo x sta utilizzando più di $\frac{3}{4}$ della tua RAM). Per questo motivo, possiamo considerare il software sentinella come uno strumento in grado di costruire un ulteriore livello di filtraggio rispetto le richieste di un processo. Un software sentinella può decidere autonomamente se concedere determinate risorse al processo, talvolta chiedendo conferma da parte dell'utente.

SSL e TLS

*Appunti di **Giuseppe Pitruzzella** - Corso di Internet Security @ DMI, UniCt*

Introduzione

SSL è un protocollo di sicurezza, o meglio, una suite di protocolli.

TLS è spesso utilizzato insieme ad HTTP, attraverso HTTPS, l'implementazione di HTTP over TLS
HTTPS utilizza la porta 443 piuttosto che la porta 80 ed è in grado di cifrare l'header, contenuti, form e cookie in HTTP.

Il protocollo SSL garantisce segretezza, integrità ed autenticazione. Infatti, alla base di tutto vi è la negoziazione di un chiave simmetrica che possa permettere ai due interlocutori proprietà di segretezza ed autenticazione; una chiave simmetrica consente inoltre la creazione di un MAC, il quale garantisce proprietà di integrità.

TLS è l'implementazione standard rispetto SSL. Può essere considerato come il tentativo di standardizzare SSL da parte di IETF, motivo per cui non viene modificata la struttura del protocollo.

Sessione e connessione

Qual è la differenza tra sessione e connessione in SSL? Una **sessione** può definita in egual modo rispetto la SA in IPSec, quindi una relazione/policy attraverso cui vengono sugellate scelte crittografiche per il determinato traffico tra due interlocutori.
Una sessione si compone di più connessioni; una **connessione** è la forma di trasporto per un determinato tipo di servizio; ogni connessione è legata ad una ed una sola sessione.

Quali sono i campi relativi ad una sessione? All'interno di una sessione, possiamo notare i seguenti campi:

1. **Session Identifier**, una sequenza di byte arbitrariamente scelti dal server.
2. **Peer Certificate**, il certificato $X.509$ per il nodo.
3. **Compression Method**, specifica l'algoritmo di compressione utilizzato prima della codifica.
4. **CipherSuite**, specifica l'algoritmo di crittografia e la funzione di hash.
5. **MasterSecret**, 48 byte condivisi dal client/server.
6. **Is Resumable**, un flag che indica se la sessione può essere ripristinata in nuove connessioni.

Quali sono i campi relativi ad una connessione? All'interno di una connessione, possiamo notare i seguenti campi:

1. **Server Random**, **Client Random**, una sequenza di byte scelti da client e server per ciascuna connessione.
2. **Server MAC Write Secret**, la chiave per MAC utilizzata dal server.
3. **Client MAC Write Secret**, la chiave per MAC utilizzata dal client.
4. **Server Write Key**, la chiave di codifica simmetrica utilizzata dal server
5. **Client Write Key**, la chiave di codifica simmetrica utilizzata dal client
6. **Initialization Vectors**, utilizzato per inizializzare la codifica a blocchi
7. **Sequence Numbers**, numeri sequenziali per i messaggi

> Perché il Master Secret si trova all'interno della sessione? Il MS si trova all'interno della sessione poiché suggella un modo per generare le chiavi per una connessione.

Suite dei protocolli in SSL

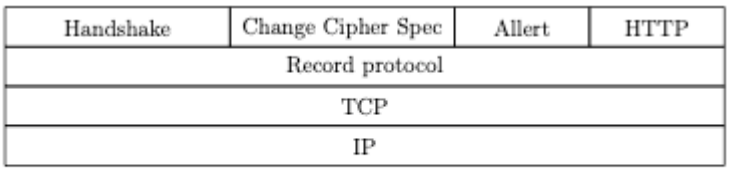


Figura 11.3: Stack protocollare SSL

SSL Handshake Protocol (SSL HP)

SSL Handshake Protocol rappresenta il primo protocollo azionato rispetto la suite di protocolli previsti da SSL. Il protocollo di handshake prevede di stabilire un segreto condiviso tra i due interlocutori, il tutto attraverso messaggi con la seguente formattazione: $| \ Type \ (1 \ byte) \ | \ Length \ (3 \ bytes) \ | \ Content \ (\geq 0 \ bytes) \ |$

Il protocollo di handshake è il protocollo più complesso tra i protocolli di SSL; esso si divide in 4 fasi, necessari in definitiva a creare il **MasterSecret**. Si noti che esiste una *fase* 0 basata sull'invio di un messaggio: **Hello Request**, un messaggio verso il server, a partire dal quale inizia l'handshake.

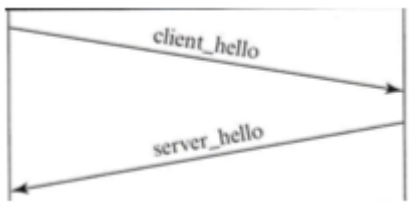
Prima fase

La prima fase si basa su un **agreement a due vie** (i.e. "*botta e risposta*") basato sullo scambio di 2 messaggi tra client e server:

- **Client Hello**, inviato da client, prevede i seguenti parametri:
    1. **Version**, comunica al server la versione più alta supportata dal client.

2. **Random**, formato da un *timestamp* (32 bit) ed una *nonce* (28 bit), è importante in caso di ripristino. Il parametro è anche conosciuto come **Client Random** e può essere inteso come una *nonce* potenziata da un valore temporale (i.e. timestamp). E' utile a garantire freshness, quindi prevenire attacchi di replica.
3. **Session ID**, *uguale* a 0 se il client desidera una nuova connessione su una nuova sessione, mentre *diverso* da 0 se desidera una nuova connessione su una vecchia sessione (non a caso, il valore diverso da 0 sarà uguale al **Session ID** della determinata sessione da ripristinare).
4. **CipherSuite**, rappresenta la lista di preferenze crittografiche. E' utile al client per far sì che egli possa comunicare le sue proposte rispetto le scelte crittografiche (e.g. RSA).
5. `**Compression Method**, una lista di algoritmi di compressione, un modo per sugellare un modo per comprimere il traffico.`

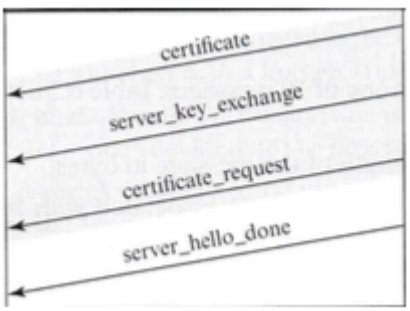
- **Server Hello**, il quale prevede i seguenti parametri:
  1. **Version**, comunica la versione minima tra la più alta supportata dal server e la più alta supportata (comunicata in *Client Hello*) dal client;
  2. **Random**, formato da un *timestamp* (32 bit) ed una *nonce* (28 bit), è importante in caso di ripristino. Il parametro è anche conosciuto come **Server Random** e può essere inteso come una *nonce* potenziata da un valore temporale (i.e. timestamp);
  3. **Session ID**, uguale al valore **Session ID** inviato dal client, se quest'ultimo ha inviato un valore diverso da 0, altrimenti il server genera un nuovo valore per il **Session ID** per la nuova sessione. In particolare, se il valore **Session ID** inviato dal client è *non nullo*, il server cerca il valore nel proprio database. Se il server trova quest'ultimo valore e *decide di accettare l'operazione di ripristino*, allora il protocollo passa direttamente alla fase 4, con il completamento della costruzione dei nuovi **KeyBlock** a partire dal **MasterSecret** per la vecchia sessione;
  4. **CipherSuite**, la singola *cipher suite* scelta del server tra le varie proposte del client;
  5. **Compression Method**, il singolo algoritmo di compressione, scelto del server tra le varie proposte del client.



Seconda fase

La seconda fase si basa sull'invio di 4 messaggi (di cui 3 facoltativi) dal server verso il client:

- **Certificate**, *facoltativo*, attraverso cui il server invia al client una lista di certificati, i quali possono agevolare la catena di fiducia. Il tipo di certificato deve essere appropriato all'algoritmo per lo scambio di chiavi selezionato nella CipherSuite.
- **Server Key Exchange**, *facoltativo*, attraverso cui il server invia una chiave se la sua chiave pubblica (fornita dal certificato) non è sufficiente per permettere lo *scambio di chiavi*, quindi la comunicazione del **PreMasterSecret**. Per esempio, scegliendo DH, **Server Key Exchange** è uguale al parametro pubblico per il server; in particolare, il parametro pubblico del server per DH è uguale ad: $Hash(ClientHello.Random, ServerHello.Random, parsing)$.
- **Certificate Request**, *facoltativo*, è inviato dal server al client per richiederne 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 accettate dal server, ordinati secondo le preferenze del server, ed una lista di nomi di autorità fidate che emettono certificati. Il messaggio **Certificate Request** è facoltativo nella misura in cui all'altro capo vi sia semplicemente il client eseguito nel browser.
- **Server Hello Done**, inviato dal server per indicare la fine del messaggio **Server Hello** e dei messaggi ad esso associati di cui abbiamo scritto sopra. Dopo aver inviato **Server Hello Done**, il server aspetta una risposta da parte del client.

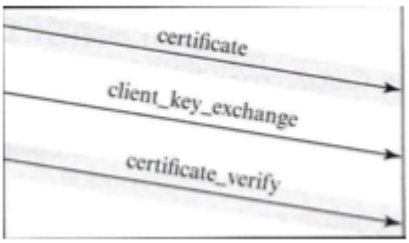


Terza fase

La terza fase si basa sull'invio di 3 messaggi (di cui 2 facoltativi) dal client verso il server:

- **Certificate**, *facoltativo*, secondo cui il client invia il proprio certificato al server per autenticarsi. Il messaggio **Certificate** è facoltativo e nella pratica non viene quasi mai utilizzato poiché quasi mai il client è diverso da un semplice client eseguito nel browser.
- **Client Key Exchange**, rappresenta la controparte di **Server Key Exchange** ed è necessario a generare il parametro **PreMasterSecret**. Il messaggio viene inviato nel caso in cui venga scelta *DH* - terminando la prima parte del protocollo DH - oppure *RSA* (in tal caso non sarà stato inviato **Server Key Exchange**).
- **Certificate Verify**, si basa sull'invio del digest ottenuto a partire dall'hash di tutto il traffico avvenuto fin'ora, da **Client Hello** fino a questo punto. In particolare, ciò che viene inviato è $Hash(MasterSecret, pad2, Hash(HandshakeMessages, MasterSecret, pad1))$. All'interno del messaggio è possibile notare:

- L'applicazione di 2 funzioni hash, tecnica che prende il nome di *hash ricorsivo* ed è utile a potenziare i limiti di invertibilità di una funzione hash;
- La funzione hash è precisamente un hash crittografico, possibile attraverso l'utilizzo di $MasterSecret$;
- Il padding, ovvero $pad1$ e $pad2$, non necessariamente uguali tra loro e necessari a soddisfare i requisiti di lunghezza di input richiesti dalle funzioni hash. Si noti che pad_1 e pad_2 sono rispettivamente rappresentati dal simbolo 6 e $\backslash$, i quali vengono ripetuti rispettivamente 48 e 40 volte per MD5 o SHA.
- Gli $HandshakeMessages$, ossia tutti i messaggi inviati da Client Hello fino ad adesso a meno di Certificate Verify. L'hashing degli $HandshakeMessages$ permette al client il controllo di integrità rispetto tutto ciò che è stato inviato fin'ora.



Quarta fase

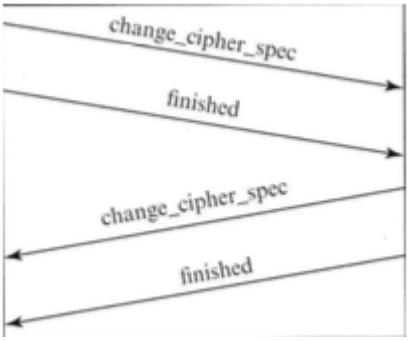
La quarta fase si basa sull'invio di 2 messaggi, 2 dal client verso il server, 2 dal server verso il client:

- **Change Cipher Spec** (i.e. **CCSP**), utilizzato per sugellare le scelte fatte all'interno della prima fase;
- **Finished**, verifica attraverso il seguente calcolo che algoritmi, chiavi e valori segreti siano stati concordati con successo:

  $$MD5(MasterSecret, pad2, MD5(HandshakeMessages, Sender, MasterSecret, pad1))$$
  $$|| \; SHA(MasterSecret, pad2, SHA(HandshakeMessages, Sender, MasterSecret, pad1))$$

  . All'interno del messaggio è possibile notare:

  - L'utilizzo di $MD5$ per poi svolgere SHA, utile ad eseguire un controllo di integrità rispetto tutti i messaggi visti fin'ora in HP. E' importante distinguere tra il controllo di integrità eseguito dal RP ed il controllo di integrità eseguito in HP mediante **finished**.
  - L'utilizzo di 2 funzioni hash diverse (i.e. $MD5$ e SHA), utile a irrobustire rispetto i limiti di invertibilità di ciascuna delle 2 funzioni.
  - Il $MasterSecret$ è necessario a rendere la funzione "*crittografica*".



Il MasterSecret

All'interno di questo paragrafo viene espletato il calcolo del MasterSecret, il quale avviene secondo il seguente calcolo:

$$MasterSecret =$$

$$MD5(PreMasterSecret + SHA('A' + PreMasterSecret + ClientHello.Random + ServerHello.Random))$$

$$MD5(PreMasterSecret + SHA('BB' + PreMasterSecret + ClientHello.Random + ServerHello.Random$$

$$MD5(PreMasterSecret + SHA('CCC' + PreMasterSecret + ClientHello.Random + ServerHello.Rando$$

All'interno del messaggio è possibile notare l'utilizzo di stringhe ogni volta diverse (i.e. "A", "BB", "CCC"), i quali rende il digest MD5 ogni volta diversi;

> Se il **PreMasterSecret** è segreto, anche il **MasterSecret** è segreto? Si. D'altra parte, se il **MasterSecret** è segreto, il **PreMasterSecret** è segreto? In altre parole, si può ricavare il **PreMasterSecret** se il **MasterSecret** è segreto? E' possibile nella misura in cui è possibile invertire l'hash, quindi la risposta è no, nei limiti della robustezza dell'hash

Ripristino di sessione

Una sessione è un associazione logica tra i due interlocutori, all'interno della quale abbiamo visto sia possibile sugellare informazioni come per esempio la **Cipher Suite**. Talvolta, potrebbe essere utile il ripristino di una sessione, la quale permette il vantaggio di godere delle scelte crittografiche già definite in precedenza. Per questo motivo abbiamo anche visto come il server conserva alcune informazioni per permettere il salvataggio dello stato della sessione.
Il tentativo di ripristino di sessione prevede un salto dalla fase 1 alla fase 4; in ogni caso, avviene un ricambio dei parametri random, il che permette una misura contro attacchi di replay.

Conclusioni

SSL può essere riassunto come una suite di protocolli, di cui il protocollo più complesso abbiamo notate essere il protocollo di handshake. Il $MasterSecret$ rimane inalterato per l'intera durata della sessione, ciò che cambia sono i segreti, i quali vengono calcolati -

freschi - per ogni nuova connessione. Dal $MasterSecret$ vengono poi calcolati tutti i segreti necessari:

1. **Server MAC Write Secret**, la chiave per MAC utilizzata dal server.
2. **Client MAC Write Secret**, la chiave per MAC utilizzata dal client.
3. **Server Write Key**, la chiave di codifica simmetrica utilizzata dal server
4. **Client Write Key**, la chiave di codifica simmetrica utilizzata dal client
5. **Initialization Vectors**, utilizzato per inizializzare la codifica a blocchi

In particolare, le precedenti chiavi vengono calcolati a partire da **KeyBlock**. Attraverso $KeyBlock$ è possibile godere di un modo per costruire un "*pezzo di chiave*". Per ottenere una chiave per la connessione sarà necessaria eseguire il calcolo seguente tante volte finché non abbiamo ottenuto un numero di "*pezzi di chiave*" pari a formare la chiave stessa.
$KeyBlock =$
$MD5(MS + SHA('A' + MS + ClientHello.Random + ServerHello.Random)) +$
$MD5(MS + SHA('BB' + MS + ClientHello.Random + ServerHello.Random)) +$
$MD5(MS + SHA('CCC' + MS + ClientHello.Random + ServerHello.Random)) \dots$

SSL Change Cipher Spec Protocol (SSL CCSP)

Il Change Cipher Spec Protocol consiste in 2 messaggi, rispettivamente inviati da client e server; quest'ultimi messaggi sono formati da nient'altro che un byte uguale al valore $'1'$.
L'utilizzo di CCSP rappresenta un "OK" da parte del mittente e sancisce l'inizio dell'utilizzo delle scelte crittografiche negoziate all'interno dell'Handshake Protocol.
In altre parole, conferma lo scambio di messaggi avvenuto in precedenza (con HP) e ad effettuare una sincronizzazine tra client e server.
Si noti esista l'impossibilità di alterare il byte inviato da CCSP grazie alle proprietà garantite dall'RP.

SSL Alert Protocol (SSL AP)

Il protocollo di Alert è utile a generare alert attraverso messaggi composti da 2 soli byte:

- Un byte per il **livello** di allarme, il quale può essere di primo o secondo livello rispettivamente per allarmi *warning*, il quale indica un problema risolvibile (e.g. **certificate_revoked**), oppure *fatal*, a cui segue la terminazione della connessione (e.g. **bad_record_mac**);
- Un byte per un codice utile ad indicare qual è l'**errore**.

SSL Record Protocol (SSL RP)

Supponendo di possedere il segreto condiviso fornito dall'handshake protocol, allora il Record Protocol esegue i seguenti step:

- **Fragment**, attraverso cui viene eseguita la frammentazione dei dati in blocchi da 2^14 byte;
- **Compress**, secondo cui avviene la compressione dei frammenti, opzionale per TLS;
- **Add MAC**, ovvero l'applicazione del MAC al frammento compresso. Questa operazione implica un controllo di integrità eseguito attraverso un chiave simmetrica inserita all'interno dell'hash, un controllo di integrità possibile poichè la chiave è posseduta dai soli interlocutori. Si noti che, posseduta integrità, come sappiamo potremmo godere anche di autenticazione.
- **Encrypt**, all'interno del quale avviene la cifratura dei frammenti, ottenendo segretezza;
- **Add Record**, secondo cui viene aggiunto ad ognuno di essi un header, o meglio, un record SSL. Quest'ultimo individua le caratteristiche per il frammento.

Come viene generato il MAC utilizzato in Add MAC****? Il MAC utilizzato in** Add MAC** viene generato secondo un hash ricorsivo, ovvero:
$hash(MACWriteSecret + pad2 + hash(MACWriteSecret + pad1 + SeqNum + SSLCompressed.Type + S$
Si noti che:

- $MACWriteSecret$ è proprio la chiave simmetrica condivisa tra i due interlocutori;
- Il calcolo del MAC utilizza, nella parte più interna, informazioni relative all'header del frammento (i.e. record SSL).
- Il padding è utile a soddisfare il requisito della due determinate funzione hash;

Com'è formato un record SSL? Un record SSL è formato dai seguenti campi:

- **Content Type**, il quale identifica il tipo di contenuto, o meglio, il protocollo (e.g. HP);
- **Major Version**, versione maggiore;
- **Minor Version**, versione minore;
- **Compressed Length**, il numero di byte del frammento compresso;
- **Data**, contenente dati cifrati.

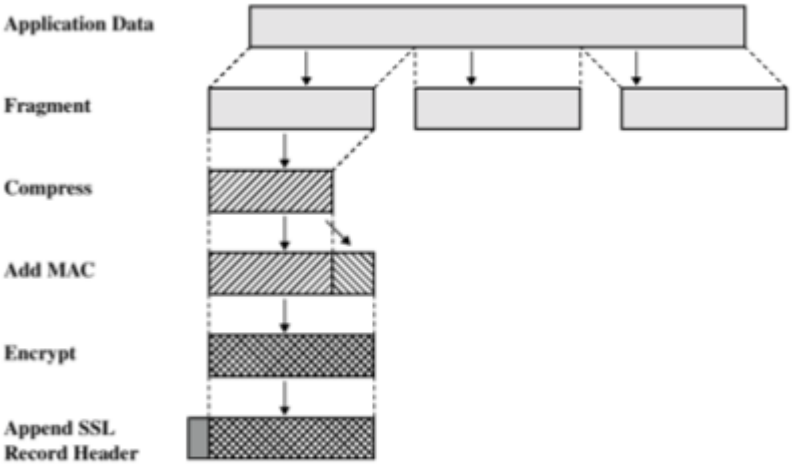


Figura 11.16: Struttura del record protocol

Differenze tra SSL e TLS

TLS può essere considerato come il tentativo di standardizzare SSL, motivo per cui non viene modificata la struttura del protocollo. Quali sono i **cambiamenti** avvenuti con TLS?

- Aggiunta di nuovi **allarmi** come **illegal_paramter** ed **unknown_ca**;
- Standardizzata la funzione di hashing ricorsivo: **HMAC**. HMAC non obbliga all'utilizzo di una determinata funzione di hash e prevede il seguente calcolo:

  $$H(K\ XOR\ opad,\ H(K\ XOR\ ipad,\ message)$$

  All'interno della funzione, leggendo dall'interno verso l'esterno, è possibile notare l'applicazione della funzione di hash H al valore uguale alla concatenazione tra $K\ XOR\ opad$ ed il testo relativo al messaggio M; il risultato di quest'ultima funzione viene a sua volta concatenato con $K\ XOR\ opad$. Si noti che è la chiave K che rende l'hashing un *hashing crittografico*. Si noti che all'interno del calcolo non è stata specificata alcune funzione hash, in questo modo essa può essere decisa attraverso **Cipher Suite** ed è possibile rimediare nel caso in cui una determinata funzione hash venga deprecata.
- Utilizzo di una **PRF** (acronimo di "*Pseudo Random Function*"), una delle migliori sorgenti di randomicità ed è utilizzata da TLS per la generazione di chiavi. La PRF viene implementata in TLS attraverso l'XOR di una duplice applicazione della funzione **P_H**; il fattore "*duplice*", indica che una volta eseguita la **P_H** con $H = MD5$, viene eseguito uno XOR con la funzione **P_H** con $H = SHA1$. Malgrado $SHA1$ ed $MD5$ siano considerate al giorno d'oggi funzioni hash deboli, accade che questo utilizzo nella **P_H** non sia stato ancora violato, motivo per cui può essere definito empiricamente è sicuro.

In definitiva, scriviamo di **funzione di espansione**, *definito* ad un certo tempo in TLS 1.1 ed implementato all'interno di TLS 1.2. Una funzione di espansione permette l'espansione di **Client Hello** attraverso nuovi campi, espansione definita in seguito a **Compression Method**. Per questo motivo, possiamo affermare che vi sia una differenza di lunghezza tra il Client Hello definito in SSL ed il Client Hello definito in TLS 1.2.

Uno dei campi previsti dall'estensione è **Heartbeat**, grazie al quale il client può capire se il server è ancora operativo. Tuttavia, il risultato di una validazione input impropria nell'implementazione dell'estensione **Heartbeat** del protocollo TLS (vulnerabilità classificata come un buffer over-read) ha introdotto il bug di sicurezza di cui abbiamo scritto durante la prima lezione: **Heartbleed**.