INTRODUZIONE ALLA SICUREZZA INFORMATICA

Quando parliamo di sicurezza, ci tocca parlare soprattutto dei possibili attacchi che un sistema può subire. Generalmente abbiamo:

- **attacchi passivi**, che riguardano l'intercettazione e il monitoraggio durante la trasmissione di dati. Questi dati possono essere violati tramite un'intercettazione o analizzando il traffico. Gli attacchi passivi sono difficili da rilevare, ma facili da prevenire.
- **attacchi attivi**, che riguardano la modifica del flusso di dati. Gli attaccanti possono fare questo tipo di violazioni tramite un mascheramento, cioè nascondendosi per non essere riconosciuti dal sistema; oppure potremo trovarci di fronte ad una ripetizione, cioè ad una copia non autorizzata dei dati. I messaggi potrebbero essere modificati e ci potrebbe essere anche un attacco DOS (*Denial of Service*), in cui vengono fatte esaurire determinate risorse di un sistema, debilitandolo. Al contrario, un attacco attivo è semplice da rilevare, ma molto difficile da prevenire.

La sicurezza informatica deve basarsi su alcuni principi, quindi alcune parole chiave:

- **autenticazione**. Bisogna assicurarsi che l'utente sia chi sostiene di essere.
- **controllo degli accessi**. Bisogna impedire l'accesso non autorizzato ad una risorsa.
- **integrità dei dati**. Bisogna impedire la modifica, l'eliminazione o la copia dei dati che vengono trasmessi.
- **non ripudiabilità**. Utente e sistema lavorano insieme ed entrambi non possono negare il proprio coinvolgimento.

Poi, alla base di un sistema di sicurezza, troviamo due componenti fondamentali:

- **trasformazione di sicurezza**. Il messaggio viene cifrato in base ad un accordo tra mittente e destinatario.
- **informazione segreta**. Essa viene associata al messaggio e, di solito, si tratta di una chiave di cifratura, che ci permette di decifrarlo.

CRITTOGRAFIA CLASSICA

Prima di iniziare a parlare di crittografia, occorre dire che, nell'antichità, esisteva anche un altro metodo per la comunicazione o trasmissione dei dati. Si chiamava **steganografia**. La steganografia è l'occultamento del messaggio. Nessun tipo di cifratura, infatti il messaggio veniva semplicemente nascosto, consegnato ad un messaggero e recapitato al destinario. Non sto qui a dire che si tratta di una tecnica molto debole dato che, se il messaggero fosse stato perquisito a dovere, i potenziali attaccanti non avrebbero avuto problemi a intercettare il messaggio. Per questo, si passò alla crittografia. Sulla crittografia si basa molta della sicurezza che noi normalmente studiamo e applichiamo all'informatica. Esistono due tipi fondamentali di crittografia:

- **crittografia simmetrica**, o a chiave unica.
- **crittografia asimmetrica**, o a chiave pubblica.

Un buon modello di crittografia deve poter soddisfare alcuni requisiti importanti. L'algoritmo deve essere pubblico e, quindi, accessibile, mentre la chiave deve essere segreta e, soprattutto, forte. Per ora ci soffermeremo sul modello di crittografia simmetrica. È composta generalmente da cinque elementi:

- **testo in chiaro**, il messaggio originale che deve essere spedito
- **algoritmo di cifratura**, la procedura con cui si trasforma il testo in chiaro in testo cifrato.
- **chiave segreta**, l'informazione segreta che mittente e destinatario si scambiano.
- **testo cifrato**, ciò che restituisce l'algoritmo di cifratura.
- **algoritmo di decifratura**, che prende in input il testo cifrato e la chiave e restituisce il testo in chiaro

Esistono due tecniche principali di crittografia simmetrica. Possiamo trovardi di fronte ad un modello **a sostituzione**, in cui si sostituisce una lettera del testo in chiaro con un'altra lettera di un ipotetico testo cifrato. Oppure un modello **a**

trasposizione, molto simile al precedente, che però effettua una permutazione delle lettere in chiaro.
Andiamo, adesso, ad analizzare alcune tecniche di cifratura a sostituzione che la storia ha reso celebri:

- **cifratura di Cesare**. È una tecnica di cifratura molto basica, che consiste nel sostituire una lettera dell'alfabeto in chiaro con qualche altra lettera messa in una posizione più avanzata. Il cifrario è semplice, ma estremamente debole dato che, anche nel remoto caso non si riuscisse a individuare la chiave, cioè il numero di shift effettuati a destra, ci sarebbero comunque soltanto altre venticinque combinazioni.
- **cifratura monoalfabetica**. È una tecnica che si basa molto su quella di Cesare, utilizzando però due alfabeti, uno in chiaro e l'altro cifrante. Così facendo, si cambia la lettera secondo un ordine casuale, avendo a disposizione 26! combinazione, quindi impossibili da trovare attaccando con forza bruta. Per rendere un cifrario monoalfabetico ancora più complicato, sono state introdotte altre figure cifranti. Le **nulle**, che sono lettere poco frequenti che alterano il significato del testo e son messe lì per aumentare le combinazioni. Gli **omofoni**, in cui una lettera frequente si cifra più volte per evitare che possa essere individuata come un'unità abituale. E poi, possiamo anche aggiungere dei **nomenclatori**. Sono meno consueti, perché sono generalmente dei simboli che sostituiscono parole comuni. Una celebre cifratura monoalfabetica fu quella utilizzata da **Maria Stuarda** e i suoi collaboratori. Istituirono un alfabeto cifrante composto interamente da simboli, nulle, omofoni e nomenclatori. L'allora regina scozzese stava per indire un gigantesco complotto contro sua sorella, la regina d'Inghilterra, Elisabetta I. Questo prima che Sir Francis Walsingham e i suoi crittografi scoprissero l'inganno, decifrassero l'alfabeto e, ovviamente, accusassero Maria di alto tradimento.
- **cifratura di Playfair**. Si tratta di una matrice 5×5, nelle cui prime posizioni va inserita una parola chiave e nelle restanti il resto delle lettere dell'alfabeto. La cifratura avviene per coppie di lettere. Se abbiamo due lettere uguali, si aggiunge un carattere separatore. Ogni lettera della coppia verrà sostituita con la lettera che si trova nella stessa riga e colonna della lettera con cui si trova in coppia.

- **cifratura di Hill**. Si tratta di un modello molto simile a questo precedentemente descritto, dove un totale di m lettere vengono sostituite con m lettere di un testo cifrato. La chiave è una matrice K e due vettori P e C.
- **cifratura di Alberti**. Per secoli la cifratura monoalfabetica era riuscita a mantenere un certo livello di segretezza. Poi, con l'avvento degli arabi, si cominciò ad analizzare le frequenze, abbandonanto le normali tecniche di forza bruta. Così facendo, questo modello crittografico divenne debole e obsoleto. Fino all'arrivo di Leon Battista Alberti, durante il rinascimento. Alberto era un eclettico e, tra le tante discipline in cui si dilettò, c'era anche la crittografia. Alberti aggiunse un doppio alfabeto cifrante a quello in chiaro. Così facendo, si crittava un messaggio passando dall'uno all'altro alfabeto. Così, si introdusse la **cifratura polialfabetica**.
- **cifratura di Vigenère**. Basandosi sulle idee di Alberti, Vigenère trovò forse la più celebre tecnica polialfabetica di tutti i tempi, che lui stesso, al tempo definì *Chiffre Indèchiffrables*. Questo perché, anziche due alfabeti cifranti, ne venivano utilizzati ben 26, il cui ordine shiftava a destra di uno spazio ad ogni cambiamento. Così facendo si otteneva la cosiddetta Tavola di Vigenère. Ogni lettera del testo in chiaro ha più lettere del testo cifrato ad essa associate e, quindi, l'analisi delle frequenza diviene impossibile. Per cifrare si utilizza una chiave e un potenziale attaccante, analizzando il testo cifrato, potrebbe carpirne la lunghezza e, automaticamente, decifrare il testo. Per questo, lo stesso Vigenère, al tempo, propose una soluzione. L'inserimento di un autokey, cioè utilizzare una chiave lunga quanto il testo da cifrare.
- **cifratura One-Time-Pad**. Si utilizza una chiave casuale che è lunga quanto il testo da cifrare, diversa per ogni cifratura o decifratura. Così facendo, si ottiene il cosiddetto **cifrario perfetto**, cioè un cifrario inviolabile, dato che, con la chiave casuale, non si può studiare la frequenza del testo. Con un attacco di forza bruta, poi, si otterrebbero diversi testi di senso compiuto, che non ci faranno capire qual è quello corretto.

Col passare del tempo, la cifratura è diventata molto complessa e richiedeva sempre più tempo trovare algoritmi giusti. Per questo, si passò alla **meccanizzazione**. Cioè alla creazione di macchine cifranti. Quelle più famose sono le macchine a **rotori**, di cui citiamo la celeberrima ***Enigma***, creata da Arthur Scherbius, a base elettromeccanica, che implementava cifrari a sostituzione

polialfabetica, in cui la sostituzione cambiava automaticamente ad ogni lettera cifrata. La decifrazione di Enigma, attuata dalla ***Bomba***, la macchina costruita dal matematico inglese **Alan Turing**, fu fondamentale durante la Seconda Guerra Mondiale, per la vittoria delle potenze alleate.

CRITTOANALISI

Se esiste la crittografia, allora esiste anche la crittoanalisi, cioè lo studio dei metodi per ottenere informazioni cifrate senza avere accesso all'informazione segreta. L'obiettivo principale è quello di trovare la chiave segreta, sfruttando l'algoritmo di cifratura, quindi senza ricorrere ad attacchi di forza bruta che richiederebbero molto tempo. Possiamo definire alcuni modelli principali di crittoanalisi, basati sul tipo di attacco:

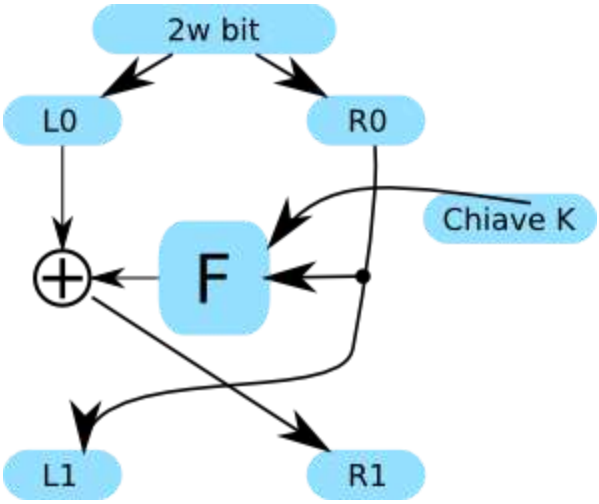
- **Known Ciphertext Attack**. Qui, un ipotetico attaccante, ha accesso solo al messaggio cifrato. L'attacco diventa un successo solo quando è possibile dedurre il testo cifrato o, anche, la chiave di cifratura.
- **Known Plaintext Attack**. L'attaccante è in possesso sia del testo in chiaro e sia di quello cifrato, in modo da poter avere informazioni segrete, ossia la chiave. Un famoso modello di crittoanalisi di questo tipo fu proprio quello attuato dal gruppo di matematici di Bletchley Park, che decifrò Enigma, la macchina cifrante nazista, partendo da testi in chiaro ripetitivi.
- **Chosen Plaintext Attack**. L'attaccante ha la possibilità di scegliere parti del testo in chiaro, in modo da far corrispondere il testo cifrato. Sembra un modello irrealistico ma, dato che la moderna crittografia viene implementata su software o hardware, e non su carta e penna, questo tipo di attacco risulta facilmente conducibile e, soprattutto, diviene fondamentale nel contesto della crittografia simmetrica, dove la chiave è pubblica.
- **Chosen Ciphertext Attack**. L'attaccante sceglie un testo cifrato e ottiene la sua versione in chiaro, usando una chiave non nota. Si tratta di uno dei modelli più potenti ed efficienti, dato che l'attaccante ha generalmente avuto accesso all'algoritmo di decifratura e risale alle informazioni in chiaro.

Su questi modelli si basano molte delle tipologie crittoanalitiche classiche. Analizziamone alcune.

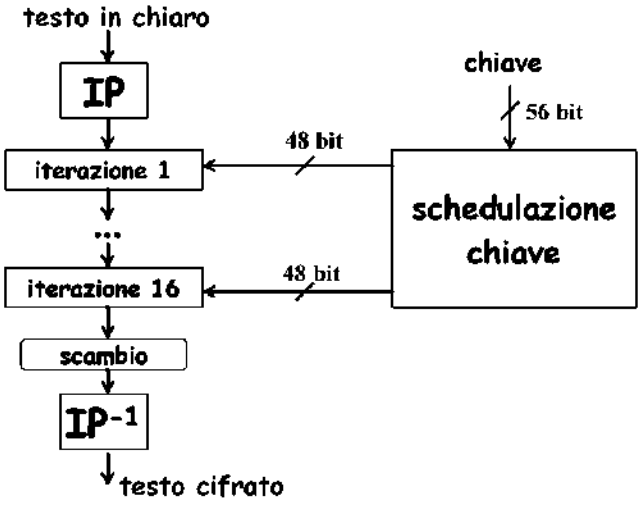
- **Analisi delle frequenze**. Questa metodologia venne introdotta per la prima volta dagli arabi. Si tratta fondamentalmente dello studio della frequenza delle lettere o di gruppi di lettere all'interno di un testo cifrato, basandosi sulle percentuali delle lettere più comuni di un determinato alfabeto all'interno di una determinata lingua (per esempio la lettera *A* ha una frequenza dell'11.74%, mentre la *Z* dello 0.49%).
- **Metodo Kasiski**. Si tratta di un attacco che venne inventato per rompere il cifrario di Vigenère. Questo Kasiski fece notare come all'interno di una tavola di Vigenère, ci sono sequenze di caratteri uguali, poste ad una certa distanza tra loro. Calcolando questa distanza, potremo risalire alla lunghezza della chiave o, quantomeno, ad un suo multiplo. Quindi, la lunghezza della chiave è il MCD (Massimo Comune Divisore) tra le distanze delle sequenze ripetute.
- **Indice di coincidenza**. Si mettono due testi fianco a fianco e si conta il numero di volte che le lettere identiche appaiono nella stessa posizione in entrambi i testi.

CIFRATURA A BLOCCHI

Introduciamo, adesso, un nuovo tipo di cifratura, detta cifratura a blocchi. Fino ad ora, abbiamo analizzato tecniche crittografica a flussi, cioè delle emissioni di dati che vengono cifrati uno per volta. Nella cifratura a blocchi, invece, il testo in chiaro viene scomposto in blocchi ed ognuno viene cifrato come un testo differente. I blocchi hanno una dimensione di 64 o 128 bit. Generalmente, si prende un testo in chiaro e lo si scompone in blocchi di n dimensione, in modo da ottenere testi cifrati di n dimensione. Così facendo, esistono 2^n blocchi di testo in chiaro distinti.
Una prima implementazione di cifratura a blocchi venne inventata da **Horst Feistel** e si basa su una serie di fasi di elaborazioni, in cui su metà dei dati viene effettuata una **sostituzione** e poi una **permutazione**, che scambia le due metà. Ad ogni iterazione si usa una chiave differente. Inoltre, Feistel si basò anche sui **principi di Shannon**: la **diffusione**, ad ogni lettera del testo cifrato si associano più lettere del testo in chiaro; la **confusione**, cioè l'applicazione di un algoritmo di sostituzione complesso. Vediamo adesso, come funzione l'algoritmo di Feistel.



Seguiamo attentamente questo schema. Il blocco è lungo 2w bit, cioè composto da due parti ognuna lunga w bit. Esistono n fasi, quella a sinistra sarà contrassegnata con L e quella a destra con R. F è la nostra funzione crittografica di riferimento. Si dividono i 2w bit in due parti. R0 viene divisa a sua volta in due parti, dove una copia finisce in F e l'altra diventa L1, cioè la L della fase successiva. Con L0 si fa lo ⊕, lo XOR, con l'output di F. Lo XOR è un'operazione che restituisce V (vero) solo se gli ingressi sono diversi tra loro. Il risultato diviene parte di R1, cioè la R successiva. All'interno della F viene inserita anche una chiave K. La buona riuscita di questo algoritmo dipende da diversi fattori, come la dimensione del blocco iniziale, la dimensione della chiave, il numero di fasi. Per decifrare questo tipo di algoritmo vengono invertite le sottochiavi.
Sulla base di questo funzionamento si basò, poi il **DES**, acronimo di *Data Encryption Standard*, cioè l'algoritmo di cifratura simmetrico più utilizzato.



Ci basiamo su questo schema di riferimento. Il DES, in input, prende un testo in chiaro, di 64 bit, e una chiave, di 56 bit. Partiamo dalla parte sinistra del nostro schema, cioè quando il testo in chiaro viene dato in pasto all'algoritmo. Abbiamo innanzitutto una permutazione iniziale. Il blocco viene diviso in due parte, ciascuna da 32 bit. Si ha un'espansione del blocco di destra da 32 a 48 bit. Poi, effettuiamo lo ⊕ tra il blocco di destra e la sottochiave. I 48 bit in output entrano in un modulo di sostituzione, dove passano attraverso otto S-BOX, ossia scatole di sostituzione. Dal modulo di sostituzione escono 32 bit, che entreranno in una permutazione finale inversa da quella iniziale. Le sottochiavi vengono generate perché passano attraverso un processo di schedulazione della chiave, dove i 56

bit iniziale della chiave subiscono una permutazione finale, ottenendo i 48 bit richiesti. Come per Feistel, anche nel DES, si effettua la decifratura invertendo le sottochiavi generate dalla schedulazione. Il DES è un algoritmo molto potente. Il numero totale di chiavi è 2^56, quindi in caso di attacchi di forza bruta, i risultati si avrebbero in millenni. Inoltre, il DES riesce a garantire anche il cosiddetto ***Availance Effect***, cioè quando una piccola variazione nel testo in chiaro produce una grande variazione nel testo cifrato.
Il DES, solitamente, svolge la cifratura su testi da 64 bit ma, nel caso il testo non risultasse all'interno di questa categoria, allora vengono utilizzate alcune modalità operative:

- **ECB** (*Electronic Codebook Chaining*). Si tratta della modalità più semplice, dato che il testo viene scomposto in n blocchi da 64 bit, in modo da effettuare un DES per ognuno di essi. I blocchi sono indipendenti e quindi esulano da errori, ma a causa della cifratura in parallelo potremo trovarci di fronte a blocchi di testi che si ripetono.
- **CBC** (*Cipher Block Chaining*). Anche qui, il testo viene scomposto in 64 bit, anche se, alla prima interazione, il primo blocco entra in uno $\oplus$ con un vettore di inizializzazione IV e poi l'output entra in un DES. Poi, alla seconda interazione, il secondo blocco entra in $\oplus$ con il blocco cifrato e così via. In caso di errori, l'operazione di $\oplus$ limita l'errore alla prima cifratura, ma si tratta di un modello molto lento.
- **CFB** (*Cipher FeedBack*). Il testo è sempre diviso in blocchi da 64 bit e si ha una cifratura sequenziale. L'IV entra in un DES insieme alla chiave e poi viene fatto lo $\oplus$ del primo blocco del testo, che entrerà in un altro DES insieme alla sottochiave e così via. Anche qui, abbiamo una limitazione dell'errore simile al modello precedente. Di questo modello esiste anche un'alternativa, detta **j-bit CFB**, che si propone di superare le limitazioni numeriche dei bit. Infatti viene scelto un valore di j a seconda delle esigenze.
- **OFB** (*Output FeedBack*). Molto simile al precedente, soltanto che, a partire dell'IV viene creata una sequenza indipendente dal messaggio e, per ottenere il testo cifrato, si fa uno $\oplus$ tra il primo blocco e la sequenza precedente. Le operazioni di $\oplus$ sono più rapide, ma non vi è dipendenza tra i blocchi.
- **CTR** (*Counter*). Colui che cifra sceglie un contatore. Il contatore entra nel DES e l'output entra in uno $\oplus$ con il blocco in chiaro, incrementando il contatore

ogni volta che c'è un'iterazione

Col tempo si è anche cercato di migliorare l'algoritmo DES per superare alcune limitazioni che esso imponeva. Per esempio si è pensato all'utilizzo di un **DES doppio**. Il testo è sempre suddiviso in blocchi da 64 bit, però si utilizzano due chiavi. Prima si cifra il primo blocco e l'output rappresenta l'input del secondo DES. Ovviamente, col tempo si è sviluppato anche un **DES triplo**, ossia l'utilizzo di tre chiavi, con lo stesso funzionamento di quello doppio.

AES

Il DES venne proposto dal **NIST**, il *National Institute of Standard Technology* nel 1977. Il DES, però col passare del tempo, ricevette numerose critiche. A parte quelle che riguardavano le dimensioni dei blocchi e delle chiavi, era un software molto lento e con dei criteri costruttivi non chiari. Il NIST, così, si propose di progettare un nuovo algoritmo di cifratura a blocchi per uso commerciale e governativo, che fosse più sicuro ed efficiente. Il concorso pubblico indetto dal NIST venne vinto dal **Progetto Rinjdael**. Il progetto AES-Rinjdael abbandonava definitivamente l'ossatura basata sull'algoritmo di Feistel. Ci teniamo a dire che AES e Rinjdael non sono la stessa cosa, poiché implementano dimensioni e specifiche diverse, ma ci riferiamo ad entrambi perché sono facilmente intercambiabili. Nell'AES la lunghezza del blocco e della chiave sono indipendenti l'una dall'altra. Inoltre, la chiave può avere lunghezza variabile in base al numero di fasi (128, 192, 256 bit), mentre il blocco è solitamente di 128. In genere, anche la chiave è di 128. Ogni operazione fatta su un cifrario di tipo AES è detta ***State***, rappresentabile su una matrice di **bytes**, unità di misura fondamentale dell'algoritmo, che equivale a 8 bit. Per iniziare la codifica, si parte copianto il blocco in State e divendendolo in sedici parti da inserire in una matrice 4×4. La cifratura è costituita da dieci round, fasi. Ogni round è composto da quattro operazioni:

- **SubBytes**. La Substitute Bytes è una permutazione di byte fatta tramite una scatola di sostituzione, una S-Box. Si tratta di una semplice trasformazione, dove i bytes di State vengono sostituiti con i corrispettivi valori all'interno della S-Box.
- **ShiftRows**. Si effettua un shift ciclico a sinistra sulle ultime tre righe di State.
- **MixColumns**. Si effettua la moltiplicazione di bytes colonna per colonna.
- **AddRoundKey**. Si effettua un'operazione di $\oplus$ tra la chiave e lo State, dato che lo State è in byte e la matrice è in word.

All'AES viene data in pasto una chiave a dimensione variabile, che solitamente è di 128 bit, quindi 16 byte e 4 word. Essa entra all'interno di un **modulo di espansione**, che genera sottochiavi da utilizzare ad ogni round, oltre che l'AddRoundKey iniziale.
La particolarità dell'AES è che l'algoritmo di decifratura non è lo stesso della cifratura, dato che usa una sequenza diversa di trasformazioni e, per questo, sarà necessaria una doppia implementazione. Nel caso si volessero gli algoritmi uguali, allora, dovremo cambiare la schedulazione della chiave.

STREAM CIPHER

Quando parliamo di Stream Cipher, parliamo di quella classe di cifrari descritti in precedenza, detti a flusso, in cui il testo viene cifrato un bit per volta. Si tratta di cifrari molto più veloci rispetto a quelli a blocchi. Qui, come chiave, viene utilizzata una **keystream**, una sequenza casuale ottenuta combinando la chiave e il messaggio. Per avviare la cifratura è necessario fare lo $\oplus$ tra il testo in chiaro e la keystream. Citiamo adesso alcuni Stream Ciphers più utilizzati:

- **LFSR** (*Linear Feedback Shift Register*). Si tratta di un tipo di registri a scorrimento, i cui dati in ingresso sono prodotti da una funzione lineare dello stato interno. Sposta i bit di una posizione verso sinistra e sostituisce il posto vuoto facendo lo $\oplus$ esclusivo del bit spostato e del bit precedente. Un LFSR ha diversi parametri che lo contraddistinguono, come il numero n di bit, il bit che inizializza il registro, detto seme.
- **A5/1**. Si tratta di un cifrario a flusso utilizzato per cifrare le comunicazioni effettuate con cellulari **GSM** in Europa e Stati Uniti. Partiamo col dire che una comunicazione GSM è composta da pacchetti di dati, chiamati burst. Un burst viene generato dall'A5/1, così che l'algoritmo sia in grado di produrre 114 bit di dati cifrati tramite uno $\oplus$ tra 114 bit in chiaro e keystream. L'A5/1 utilizza una chiave a 64 bit, anche se sono solo 54 effettivi. L'A5/1 si basa sulla combinazione di tre LFSR, con sincronizzazioni lineari. I registri sono sincronizzati a zero e poi, per 64 fasi, i bit della chiave vengono mescolati con l'i-esimo bit che è aggiunto al bit meno significativo tramite uno $\oplus$. L'algoritmo non è molto potente e ha mostrato diverse vulnerabilità.
- **RC4**. Acronimo di Ron's Code, da Ron Rivest, è l'algoritmo usato generalmente per i protocolli crittografici come SSL e TLS, ma anche in WEP e WPA, ossia la sicurezza delle reti wireless. L'algoritmo genera una keystream, che poi viene combinata con il testo in chiaro tramite uno $\oplus$, ottenendo il testo cifrato. Per generare la keystream, l'algoritmo utilizza una S-Box di 256 byte e due indici di 8 bit, identificati come *i* e *j*. La chiave è di lunghezza variabile, generalmente da 40 a 256 bit.

- **Salsa20**. Questo algoritmo si proponeva di utilizzare operazioni semplici, ottenendo però, una grande velocità di calcolo. Questo si garantì in parte perché la chiave di cifratura viene data in pasto direttamente alla funzione di cifratura, senza dover poi passare per vettori o schedulazioni. La chiave è composta da 256 bit, mentre il vettore di inizializzazione è a 64. L'algoritmo genera un keystream di 2^70 byte e la cifratura avviene in venti operazioni, combinando i byte del messaggio in chiaro con i byte del keystream tramite uno $\oplus$.
- **Chacha20**. Nasce come una variante di Salsa20, ma si propone di superare alcune limitazioni. Per esempio ottiene un keystream di 2^139, ottenendo meno round.

CIFRATURA ASIMMETRICA

Come già anticipato, la crittografia può essere simmetrica, oppure asimmetrica, a chiave pubblica. In realtà si tratta di una definizione non propriamente corretta. In questa cifratura utilizziamo una **chiave privata**, per cifrare il messaggio e una **chiave pubblica**, per decifrarlo. Quest'ultima è resa disponibile all'interno di un **file pubblico**. Per spedire un messaggio, il mittente legge la chiave nel file pubblico e cifra il messaggio utilizzandola. Una volta ricevuto il messaggio, il destinatario utilizza la chiave privata per decifrarlo e leggerlo. Un primo vantaggio di questa crittografia è che non ci sono problemi riguardanti la segretezza della chiave, perché non bisogna tenerla segreta, né tantomeno trovare stratagemmi per scambiarsela in modo sicuro. Il mittente genera un coppia composta da chiave pubblica e privata, inserendo la prima nel file pubblico. Alla base della crittografia asimmetrica vi è la cosiddetta **funzione one-way**. Si tratta di una funzione matematica che gode di diverse proprietà. È ***facile da calcolare***, perché esistono algoritmi in grado di calcolarla in tempo polinomiale, in tempo limitato ad una determinata funzione matematica. Inoltre, ***è difficile da invertire***, cioè è difficile trovare un algoritmo che possa calcolarne la controimmagine, scanso informazioni aggiuntive.
Uno dei più famosi algoritmi di cifratura asimmetrica è detto **RSA**, dal nome dei suoi fondatori Rivest, Shamir e Adleman. Quando si parla di crittografia asimmetrica è opportuno soffermarsi sulla difficoltà nel trovare una coppia di chiavi, pubblica e privata, che siano in relazione tra loro, ma difficili da calcolare se si conosce una delle due. Si parte scegliendo due numeri molto grandi, p e q, primi. Denominiamo $n = p * q$, come il prodotto tra p e q. Definiamo $\Phi(n)$ come $(p-1)(q-1)$, la sua funzione di Eulero. La fattorizzazione di n è segreta. Scelto un arbitrario numero e, coprimo con $\Phi(n)$ diciamo che il MCD (Massimo Comune Divisore) tra e e $\Phi(n)$ è uguale a 1. Calcoliamo d, tale che $ed = 1 mod \Phi(n)$. Il numero e è detto **esponente pubblico**, mentre il numero d è detto **esponente privato**. Entrambi saranno i due secondi valori della coppia di chiavi dopo, ovviamente, il numero n. Una volta generate le chiavi è possibile procedere con il processo di cifratura. Per cifrare un messaggio tramite RSA si fa $M^e mod n$, dove n

ed e vengono letti nel file pubblico. Per decifrare, invece, si fa $C^d mod n$, dove n e d fanno parte della chiave privata.
Per riuscire realmente ad analizzare la sicurezza di RSA, possiamo soffermarci sugli attacchi che questo algoritmo può subire. Il primo, generalmente, è quello che prevede che un ipotetico attaccante conosca la chiave pubblica *PUBK(n, e)* e vuole calcolare *PRIK(n, d)*. Conosce *n* e quindi deve calcolare *d*. La formula è questa $d = e^- 1 mod d \Phi(n)$. Quindi, l'attaccante dovrebbe fattorizzare *n* per ottenere *p* e *q* e poi dovrebbe calcolarsi la funzione di Eulero. E poi può calcolare *d* servendosi di ***Las Vegas***, un algoritmo utilizzato per la fattorizzazione, che da circa il 50% di possibilità di calcolare d e, quindi, rompere il cifrario. A parte questo metodo, tutti gli altri algoritmi hanno una complessità esponenziale e, quindi, non potrebbero optare per una soluzione di questo tipo.

ACCORDO SU CHIAVI

Quando parliamo di un accordo su chiavi, ci riferiamo ad un protocollo crittografico che permette ad un mittente e a un destinatario di stabilire una **chiave condivisa**, segreta, anche utilizzando canali di comunicazione insicuri, quindi pubblici. Una delle tecniche più utilizzate è quella di **Diffie-Hellman**, dal nome dei suoi fondatori. Facciamo un esempio per carpirne il funzionamento. Supponiamo che due entità, chiamate Alice e Bob, vogliano scambiarsi una chiave in modo sicuro. Alice genere pubblicamente un numero P, che è generalmente molto grande, e un **generatore** g. Un generatore, in aritmetica modulare, è una base che, quando viene elevata a potenza, genera tutti i numeri che sono primi con P. Alice, poi, genera un numero casuale, detto x, minore di P e calcola $X = g^x (mod P)$, inviando X a Bob. In questo modo P, X e g sono pubblici, mentre x lo conosce solo Alice. Bob, a sua volta, genera un numero y, minore di P e calcola $Y = g^y (mod P)$, inviando Y ad Alice. Così facendo, P, X, Y e g sono pubblici, ma x lo conosce solo Alice e y solo Bob. Alice calcola $k = Y^x (mod P)$, mentre Bob $k = X^y (mod P)$. Così facendo, si otterrà k, che Alice e Bob utilizzeranno per comunicare con un cifrario, generalmente simmetrico.
Parlando della sicurezza intrinseca dell'accordo su chiavi, supponiamo che un ipotetico attaccante riesca a intercettare le componenti pubbliche, ossia P, g, X e Y. Così facendo, potrebbe facilmente calcolarsi x e y facendo $x = log_g (X)$ e $y = log_g (Y)$. Qui entra in scena il fattore su cui si basa fortemente questo accordo su chiavi, ossia il **problema del logaritmo discreto**. Dato che, in precedenza, abbiamo scelto N come un numero molto grande, quindi 1024 bit o superiore. Calcolarne il logaritmo discreto, quindi, sarebbe computazionalmente impossibile. L'algoritmo, tuttavia, resta esposto all'**attacco *Man in the Middle***. Immaginiamo che un attaccante riesca a intercettare X e, fingendosi Bob, invia un proprio numero Y ad Alice. In questo modo l'attaccante stabilisce una connessione tra lui ed Alice e, quest'ultima, continuerà ad inviare i messaggi, credendo di scambiarseli con Bob. Per questo, serve un meccanismo in grado di identificare Alice e Bob, per questo si ricorre ad un ente certificatore e, quindi, ad

una **firma digitale**. A questo protocollo viene legato anche il concetto di ***forward secrecy***, proposto dallo stesso Diffie nel 1992. In parole povere si tratta di una serie di protocolli di **negazione delle chiavi** dove, se una chiave viene compromessa, le altre sottochiavi generate continuano a restare riservate. I meccanismi di **PFS**, chiamata così perché spesso ci riferiamo alla FS come ***Perfect Forward Secrecy***, sono spesso utilizzati in app di messaggistica istantanea come *Signal*, che utilizza una cifratura end-to-end, o come un browser che utilizza Onion Routing, come *Tor*, che ha bisogno di crittare ogni suo nodo della rete per garantire l'anonimato sul web.

FIRMA DIGITALE

Le tecniche di firma digitale furono introdotte perché era necessario garantire autenticità tra mittente e destinatario, in modo che non si verificassero episodi come quello del *Man in the Middle*. Generalmente, per la firma digitale, si usano algoritmi di cifratura asimmetrica. Prima di poter procedere per parlare dei vari tipi di firma, dobbiamo definire quali sono i tipi di attacco:

- **Key Only Attack**. L'attaccante conosce solo la chiave pubblica del mittente.
- **Known Message Attack**. L'attaccante conosce solo alcuni messaggi con le relative firme.
- **Chosen Message Attack**. L'attaccante sceglie alcuni messaggi e chiede al mittente di firmarli.

Anche gli scopi di un attacco possono essere molteplici:

- **Total Break**. L'attaccante ha intenzione di scoprire la chiave privata del mittente per firmare ogni messaggio.
- **Selective Forgery**. Individuato un messaggio, l'attaccante vuole individuarne la firma.
- **Existential Forgery**. L'attaccante vuole trovare una coppia (M, F), messaggio-firma, che venga riconosciuta correttamente.

La prima firma che andremo ad analizzare sarà quella effettuata tramite l'**algoritmo RSA**, la cui sicurezza si basa sulla difficoltà di fattorizzazione, come abbiamo visto. Anche il processo è pressoché identico, soltanto che bisogna applicare un **algoritmo di firmatura**. Per firmare un messaggio si effettua, quindi, questa operazione: $F = M^d mod n$. Per verificare che la firma sia autentica, il destinatario controlla se è vero $M = F^e mod n$. Nel caso il messaggio da firmare risultasse molto lungo, l'algoritmo è poco efficiente. Così, ricorriamo all'uso delle **funzioni hash**. Il valore hash di un messaggio è una rappresentazione non ambigua e non falsificabile del messaggio. Una funzione hash, generalmente,

comprime, è molto sicuro, perché è impossibile trovare due messaggi con lo stesso valore hash e, inoltre, gode delle proprietà one-way. È facile trovare il valore hash di un messaggio, ma è difficile trovare il messaggio a cui è associato il valore hash. Come già anticipato, la funzione hash comprime e, quindi, questo nostro ipotetico messaggio di grandi dimensioni viene ridotto e si calcola la sua funzione hash. Per effettuare una firma su un messaggio ridotto, calcoliamo il valore hash ed effettuiamo questa operazione: $F = [h(M)]^d mod n$ e, invece se $h(M) = F^e$ allora la firma sarà autentica.
Un'altra tecnica di firma digitale, può essere quella fatta attraverso l'**algoritmo di ElGamal**, dal nome del suo inventore. Anche questa tecnica, come l'accordo Diffie-Hellman, si basa sul problema del logaritmo discreto. Seguendo gli stessi principi dell'accordo su chiavi, viene generata una chiave pubblica e una privata. Poi, tramite alcuni parametri di riferimento si calcola la firma digitale, composta da una doppia (r, s), dove r è il parametri dei numeri casuali e s si riferisce alla funzione hash.
L'ultima tecnica di firma digitale che analizzeremo è il **DSS**, acronimo di *Digital Signature Standard*. Anche questo metodo utilizza le funzioni hash. Il codice hash viene fornito in input ad una funzione di firma, insieme ad un numero casuale k. La firma dipende dalla chiave privata del mittente e da una chiave pubblica globale. Si definisce una **funzione di verifica** che dipende dalla chiave pubblica del mittente e da quella globale e il destinatario genera un codice hash associato al messaggio, inviando alla funzione il codice e la firma. Citiamo anche un'ultima tecnica, molto simile alla precedente, detta **firma digitale remota**, che invia l'hash di un documento ad un modulo, detto **HSM**, acronimo di *Hardware Secure Module*, che sono dispositivi remoti, gestiti da un certificatore accreditato.
Il **Codice dell'Amministrazione Digitale** fornisce anche altri tipi di autenticazione, oltre la firma digitale:

- **firma elettronica**: dei dati in forma elettronica, come password, PIN e tecniche biometriche.
- **firma elettronica avanzata**: sono sempre dati in forma elettronica, però sotto forma di firma. L'esempio più lampante è la firma grafometrica, ottenuta scrivendo con una penna elettronica su uno schermo touch.
- **firma elettronica qualificata**: parliamo, ovviamente, di dati in forma elettronica, che però vengono forniti o creati da mezzi sui quali il firmatario ha

un controllo esclusivo. Per fare un esempio i token sul nostro cellulare oppure le smart-card.

FUNZIONI HASH

Abbiamo parlato precedentemente che le funzioni hash servono, durante la firma digitale, a comprimere messaggi molto lunghi per favorirne la sicurezza. Una funzione hash, però, può essere utilizzata anche per mantenere l'integrità dei dati memorizzati. Una funzione hash ha tre caratteristiche principali:

- **sicurezza debole**: dato un messaggio specifico M è difficile trovare un messaggio specifico M1 che abbia la stessa funzione hash.
- **sicurezza forte**: in generale, è difficile trovare due messaggi con la stessa funzione hash.
- **one-way**: lo stesso principio che abbiamo descritto in precedenza, cioè una funzione facile da calcolare, ma difficile da invertire.

A proposito di due funzioni hash uguali, esiste una formula per calcolare la probabilità che ci siano due messaggi con la stessa funzione hash che è $t = \sqrt{n * 2ln(\frac{1}{1-e})}$, dove n indica il numero di valori hash diversi, t indica il numero di messaggi da scegliere e e è la probabilità di successo.
Tra le funzioni hash crittografiche più famose citiamo **MD5**, acronimo di *Message Digest 5*. Questa funzione si proponeva di avere una sicurezza forte e diretta, oltre che una buona velocità e semplicità. L'unità elementare è un blocco da 512 bit e si opera su word da 16. Ci sono quattro round e, in ognuno di essi, ci sono sedici operazioni. Un algoritmo che si rifornisce del meccanismo di MD è lo **SHA**, acronimo di *Secure Hash Algorithm*. SHA è un algoritmo che utilizza funzioni hash per produrre un MD di lunghezza fissa, partendo da un messaggio di lunghezza variabile. Abbiamo due tipi di implementazioni SHA:

- **SHA-1**, che produce un MD da 160 bit.
- **SHA-2**, diminutivo di SHA-256, che produce MD da 256 bit.

Il primo passo che uno SHA effettua è quello di aggiungere un bit di riempimento al messaggio, fino a far corrispondere la lunghezza al più piccolo multiplo della sua dimensione. Poi, al messaggio precedente vengono aggiunti 64 bit, da cui

otteniamo una lunghezza sempre multiplo della dimensione dell'MD. Viene inizializzato un buffer di 160 bit, suddiviso in cinque registri da 32 bit. Infine, il messaggio viene suddiviso in blocchi. Ogni blocco passa in una funzione di quattro round, con venti operazioni ciascuna. Alla funzione si da in pasto il blocco, una costante k e i valori dei registri. Al termine della computazione, verranno aggiornati i valori successivi.

MAC

Un *Message Authentication Code* è un blocco di dati che serve a garantire l'**autenticazione** e l'**integrità** di un messaggio, generato tramite meccanismi di crittografia simmetrica. Generalmente si invia un messaggio *M* con una chiave *K* ad un modulo MAC, che restituisce un valore *MAC(M, K)*. Nota bene: un modulo MAC fornisce solo autenticazione e integrità, senza soffermarsi sulla riservatezza. Anche qui, per definire la sicurezza di un modulo MAC, bisogna soffermarsi sui tipi di attacco:

- **Known Message Attack**. L'attaccante conosce alcuni messaggi con i loro relativi MAC.
- **Chosen Message Attack**. L'attaccante conosce i messaggi e chiede di calcolarne i MAC.
- **Adaptive Chosen Message Attack**. L'attaccante conosce i messaggi e chiede di calcolarne i MAC, basandosi sulle risposte precedenti.

Gli scopi dell'attacco sono simili a quelli fatti per la firma digitale:

- **Total Break**. L'attaccante vuole determinare la chiave.
- **Selective Forgery**. Dato un messaggio *M*, vuole ricavarsi *y* tale che *y = MAC(M, K)*.
- **Existential Forgery**. L'attaccante vuole determinare una coppia *M, y* tale che *y = MAC(M, K)*.

Il MAC può essere costruito sostanzialmente in due modi:

- **costruzione basata su cifrari a blocchi**. Anche qui, ci sono due tipi di implementazioni. La prima è basata sul meccanismo di cifratura CBC ed è detta **DAA**, acronimo di *Data Authentication Algorithm*. Questo metodo ha alcuni problemi di sicurezza perché, due blocchi dello stesso messaggio in chiaro avrebbero lo stesso MAC. Per questi si è passati all'utilizzo di **CMAC**, acronimo di *Cipher-based MAC*. Si implementa questo algoritmo tramite l'uso

di AES e di un triplo DES. Il testo viene diviso in blocchi della stessa dimensione. La lunghezza del messaggio può essere multiplo della lunghezza dei blocchi, oppure no, e, in quest'ultimo caso, vengono aggiunti dei bit di riempimento. CMAC utilizza tre chiavi, una chiave *K* e due sottochiavi *K1* e *K2*. Per generare le sottochiavi si cifra il blocco costituito interamente da bit nulli, shiftando di un bit a sinistra e facendo lo ⊕ con una costante *k* che dipende dalla dimensione del blocco.

- **costruzione basata su funzioni hash**. Questo tipo è detto **HMAC**, acronimo di *keyed-Hash MAC*. Esso nasce con la necessità di raggiungere alcuni obiettivi, come l'utilizzo delle funzioni hash senza modifiche e la garanzia di avere le stesse caratteristiche di queste ultime. Le componenti di HMAC sono le seguenti. *H*, che è una funzione hash di tipo SHA-1; *n*, che è la lunghezza del valore hash; *M*, un messaggio che viene diviso in blocchi di *b* byte, con *b>n*; *ipad*, byte che viene ripetuto b/8 volte con rappresentazione esadecimale di 36; *opad*, byte che viene ripetuto b/8 volte con rappresentazione esadecimale di 5C; *K*, chiave segreta. La funzione HMAC prende in input la chiave *K* e il messaggio *M* e produce in output il valore MAC di *M* con chiave *K*. Prima calcola lo ⊕ tra *K* e *ipad*, a cui poi vengono accodati *M* e *K*. Poi si fa lo ⊕ tra *K* e *opad*, accodando solo *H*. Alla fine otterremo l'output desiderato. La sicurezza di questo algoritmo dipende dalla funzione hash utilizzata.

CERTIFICATI E PKI

Quando ci siamo soffermati sugli algoritmi di crittografia asimmetrica abbiamo detto che serve generare una chiave pubblica e una privata. La chiave pubblica è inserita in un file pubblico. Per file pubblico abbiamo diverse possibilità:

- **annuncio pubblico**: scambio diretto tra mittente e destinatario, tramite la pubblicazione su un sito web o via e-mail. Questa tecnica, ovviamente, non è molto sicura.
- **directory pubblica**: c'è un'entità privata che gestisce le chiavi pubbliche di tutti e, ogni utente, può gestire la propria chiave. Anche qui ci sono problemi di sicurezza infatti, se venisse violata questa entità, tutte le chiavi andrebbero scoperte.
- **autorità per le chiavi pubbliche**: si tratta di un'autorità che gestisce le directory di chiavi pubbliche. Essa stessa possiede una chiave pubblica nota a tutti gli utenti e, ogni qualvolta l'utente la chiede, la spedisce. Qui abbiamo dei problemi relativi al server che, essendo online ventiquattro ore su ventiquattro, con grande affluenza di utenti, potrebbe generare problemi. Una volta ricevute le rispettive chiavi, mittente e destinatario, che chiameremo ancora Alice e Bob devono essere sicuri che le chiavi siano associate alla persona giusta. Per questo, si ricorre ad un protocollo di **mutua autenticazione**. Cioè, oltre alla chiave, Bob e Alice si scambiano anche un numero casuale. Questo non è sempre possibile, perché l'autorità delle chiavi deve essere sempre attiva.

Per questo motivo, è stato inventato il **certificato digitale**. Esso è composta dalla **chiave pubblica**, associata all'utente e da un certificatore, che è una **stringa binaria**. Inoltre, un certificato può contenere anche un periodo di validità, informazioni sulla chiave e informazioni sull'utente. Lo standard più diffuso per i certificati è detto ***X.509***, che si fa garante dei certificati digitali e delle autorità di certificazione (**CA**). Inoltre definisce l'utilizzo dei certificati all'interno di infrastrutture a chiave pubblica (**PKI**), i processi che consentono alla CA di farsi garante dell'identità di un utente. Ogni volta che una chiave viene compromessa, in generale per violazioni o compromissioni di utente e algoritmi, è possibile

chiedere la **revoca del certificato**. Questo si fa tramite il **CRL**, la *Certificate Revocation List*, cioè una lista di tutti i certificati che sono stati revocati. Generalmente, una PKI è composta da più CA, che sono disposte in maniera gerarchica. Al vertice si trova un **CA-root**, mentre sotto si possono trovare altre **sub-CA**. Da chi si fa firmare la CA-root dato che è la prima CA dell'albero gerarchico? Si firma da sola, basandosi sulla propria *buona reputazione*. Per avere la certezza di una CA valida è necessario a passare a forme di garanzia giuridiche.

PROTOCOLLI SSL E TLS

Transport Layer Security (**TLS**), *Secure Sockets Layer* (**SSL**), *Datagram Transport Layer Security* (**DTLS**). Si tratta di meccanismi crittografici che sono in grado di rendere sicure le comunicazioni dal mittente al destinatario (end-to-end) su reti TCP/IP, fornendo autenticazione, integrità dei dati e confidenzialità. Essi operano a livello di trasporto. Questi protocolli definiscono un canale di **connessione**, a cui è associata una **sessione**. Nella sessione sono presenti i parametri crittografici tra le parti. SSL/TLS è composto da quattro protocolli:

- **Handshake Protocol**. Mette in collegamento mittente e destinatario, negozia le versioni e l'insieme delle ***Ciphersuite***, cioè le varie procedure crittografiche, come autenticazione, accordo su chiavi e algoritmo di cifratura e di MAC.
- **Record Protocol**. Usa i parametri e le specifiche che sono state negoziate all'interno del protocollo precedente. Divide i dati in blocchi, li comprime, applica il MAC, li cifra e poi trasmette il risultato.
- **Change Cipher Spec Protocol**. Viene usato per aggiornare la Ciphersuite, inviando un unico messaggio di update.
- **Alert Protocol**. Trasmette messaggi di alert, generalmente indicando il livello di severità, che sia *warning* o *fatal*, o il tipo di alert, cioè il motivo che ha generato l'alert.

AUTENTICAZIONE UTENTE

Durante una comunicazione, è importante che il sistema metta a disposizione dell'utente alcune procedure per essere identificato in modo univoco all'interno della rete. Deve essere riconosciuto e, poi, deve poter accedere ai suoi servizi. Un primo metodo di autenticazione può essere la conoscenza e il riconoscimento di una **password**. L'utente conosce la password, il sistema la confronta all'interno del database delle password, memorizzando una sua rappresentazione. Per esempio si memorizza la password in chiaro in un file protetto. Questo, però, non garantisce sicurezza per chi riesce a violare il file. Per questo motivo, i grandi sistemi, memorizzano le password **in forma cifrata**. Per esempio, UNIX utilizza una variante del DES in 25 operazioni per cifrare le password, usando la funzione di cifratura `crypt()`. Inoltre, si può memorizzare anche il valore hash della password. Un metodo di difesa contro gli hacker che violano il file delle password è detto *shadow*, che è il nome di un file accessibile solo da root.
Oltre ad attacchi convenzionali, come lo **spionaggio** o l'**estorsione**, una password può essere trovata tramite **ricerca esaustiva**, cioè l'attacco di forza bruta, o tramite **dizionario**, cioè la possibilità che la password sia una parola del dizionario. Per questo, ci sono alcuni criteri di **vulnerabilità per le password**. Per esempio possono essere nomi proprio che indicano il nome di un familiare, il numero dell'appartamento, il nome del proprio animale domestico. Tutti criteri facilmente reperibili. Generalmente, per rendere una password forte, bisogna usare lettere maiuscole e minuscole, numeri, caratteri speciali e devono avere una lunghezza elevata. Per esempio: *DA.nMdCdNV* ci vorrebbero circa ottantamila anni per hackerarla, quindi è una password forte. Inoltre, è sempre consigliato cambiare ciclicamente la propria password, per evitare l'**invecchiamento**.
Oppure, esiste una tecnica chiamata *one-time password* (**OTP**), cioè una password che viene usata soltanto una volta. Ogni OTP viene generata usando funzioni crittografiche e alcuni valori univoci. Per le OTP possiamo avere diversi tipi di approcci:

- **sincronizzazione temporale**. Gli algoritmi si basano sul tempo che intercorre tra l'autenticazione del client da parte del server. In parole povere, le password sono valida per un breve periodo di tempo.
- **in base alla precedente**. La OTP generate sono una coda all'interno di una sequenza predefinita e ognuna viene generata in base al valore della precedente. L'algoritmo matematico più famoso che si basa su questa procedura è detto **algoritmo di Lamport** che utilizza un valore iniziale predefinito e una funzione hash che viene applicata ripetutamente per generare le password.
- **challenge**. La OTP è basata su un numero casuale che viene scelto dal server, oppure da un contatore.

Ovviamente, poi, si può ricorrere alla **Two-Factor Authentication**, cioè si scelgono due fattori per verificare l'identità. In banca si utilizza sia la carta che il PIN. Oppure sia la password che un SMS di conferma, anche se questo non è molto sicuro perché gli SMS possono subire ritardi o finire in spam.

SISTEMI BIOMETRICI

Quando parliamo di sistemio biometrici, facciamo riferimento a dispositivi automatici che sono in grado di identificare e autenticare basandosi su caratteristiche dell'utente. Che siano **fisiche**, come impronta digitale o scansione dell'iride, o **comportamentali**, come firma o voce. I sistemi biometrici hanno delle caratteristiche che sono comuni a tutti:

- **universalità**: ogni individuo deve possedere quella determinata caratteristica biometrica. In natura accade, ma non sempre, infatti c'è un 1% di possibilità che un individuo non abbiamo universalità.
- **unicità**: due persone non possiedono la stessa caratteristica biometrica.
- **permanenza**: le caratteristiche non cambiano nel tempo.
- **catturabilità**: la caratteristica deve poter essere acquisita e memorizzata. Queste procedure sono dette di ***Enrollment***.

Un sistema biometrico deve provvedere:

- **identificazione**. Cerca una corrispondenza all'interno del database.
- **autenticazione**. Confronta un campione con un singolo modello archiviato.

Come già anticipato, ci sono diversi sistemi biometrici. Li analizziamo.

- **impronte digitali**. Basandosi sulla loro anatomia, cioè sulle creste e sulle minuzie, esse vengono acquisite tramite inchiostratura o sensori. Le impronte digitali garantiscono unicità e cambiano solo a causa di cause di forza maggiore, come incidenti o scottature.
- **scansione della retina**. La retina è la membrana più interna del bulbo oculare e si acquisce la sua conformazione proiettando un fascio di luce a bassa intensità nella pupilla. Anche la scansione della retina garantisce unicità e immutabilità, ma è difficile da acquisire e, a causa di malattie, si può compromettere la sua stabilità.

- **geometria della mano**. Si tratta di un sistema abbastanza vecchio, che misura le caratteristiche fisiche della mano. L'acquisizione avviene attraverso apparecchiature radiologiche. Questo sistema garantisce immutabilità e unicità, ma è sensibile a cause di forza maggiore, come incidenti che compromettono la geometria della nostra mano.
- **dinamica della firma**. Si confronta la firma, controllando la pressione e la velocità. Questo sistema è semplice, ma è facilmente influenzabile da alterazioni dovute allo stato d'animo.
- **riconoscimento del volto**. Si basano su immagini che provengono dall'acquisizione del volto, mediante sensori o fotocamere. Sono veloci e semplici, ma vulnerabili, perché visi sintetici sono facilmente replicabili (vedi Tom Cruise in *Mission Impossible*) e poi ci sono problematiche nell'acquisizione, come luminosità o cause di forza maggiore, come incidenti che sfigurano l'utente.
- **riconoscimento della voce**. Si ripete una frase fissa e il sistema analizza tono, dinamica e waveform. È affidabile e flessibile ma anch'esso influenzabile da fattori esterni, come il cambio di voce a seguito di un incidente.

Ovviamente, ci sono anche altri sistemi, come l'analisi del DNA, la forma delle labbra, l'odore, l'andatura ecc. Per definire l'accuratezza biometrica è giusto che non si verificano alcuni eventi:

- **FNMR** (*False Non-Match Rate*). Le stesse caratteristiche sono associate a due utenti diversi.
- **FMR** (*False Match Rate*). Caratteristiche di due utenti risultano associate allo stesso utente.
- **FTC** (*Failure to Capture*). Il dispositivo fallisce l'acquisizione.
- **FTE** (*Failure to Enroll*). Il dispotivo non riconosce gli utenti.

Col tempo, i dispositivi mobili hanno integrato al loro interno sensori per il riconoscimento biometrico, come il ***Touch ID*** di Apple, che riconosce l'impronta digitale. O il ***Face ID*** che riconosce il volto. I sistemi biometrici supportano l'**architettura multimodale**, cioè l'acquisizione e la valutazione di diverse biometrie, svolta in maniera indipendente.

POSTA ELETTRONICA CERTIFICATA (PEC)

La PEC è un tipo di posta elettronica che permette di dare ad un messaggio lo stesso valore legale di un raccomandata con ricevuta di ritorno, garantendo la prova dell'invio e della consegna. Anche questa tecnologia ha degli svantaggio, dato che non è riconosciuta internazionalmente e non è gratuita. Al termine del procedimento PEC vengono rilasciate tre ricevute, quella di accettazione, di presa in carico e di avvenuta consegna.

MALWARE

La parola malware deriva da *malicious software*, ed è fondamentalmente una sequenza di codice nocivo. Un malware è progettato per provare dei danni e alterare il normale comportamento del sistema, agendo in maniera subdola, cioè all'insaputa dell'utente. Spesso, un hacker, utilizza meccanismi di **offuscamento** per occultare l'esecuzione del malware, mentre tramite meccanismi di **packing** lo si comprime per renderlo più rapido. Tutte queste tecniche servono a rendere il malware più difficile da rilevare e analizzare. Un malware prolifera quando ci sono utenti inesperti o misure di sicurezza inadeguate. Un malware, come tutte le componenti software, ha un determinato ciclo di vita:

- **infezione**. Il malware penetra e si installa nel sistema, superando le barriere difensive. L'infezione può avvenire sia tramite web che tramite dispositivi esterni oppure attraverso mail di phishing.
- **quiescienza**. Il malware viene memorizzato nel sistema, ma non è attivo e attende che si verifichi qualcosa. Qui è dove il malware è più vulnerabile ai controlli.
- **replicazione e propagazione**. Una volta verificatisi determinati eventi, il malware si replica e individua dei bersagli.
- **azioni malevoli**. Il malware non si limita ad una specifica operazione, ma combina più problematiche a cascata.

I malware possono essere di diverso tipo:

- **malware a diffusione**. Fanno parte di questa classe i **virus**, i più noti e diffusi malware. Il virus può replicarsi e danneggiare parti del sistema operativo. Il virus necessita di un software ospite in cui inserirsi e iniziare l'infezione. Un tipo particolare di virus è il **worm**, che non necessita di un software ospite e può autopropagarsi, sfruttando le vulnerabilità del sistema operativo. Un altro caso particolare di virus è il **rabbit**, che è in grado di autopropagarsi velocemente, rendendo inutilizzabili le risorse del sistema.

- **malware strumentali**. Di questa categoria fa parte il **trojan**. Il trojan, come dice la parola, è come un cavallo di Troia, che cela la sua vera identità. Appare all'utente come un software utile o, quantomeno, innocuo, infettando il sistema per molteplici scopi. Non è, però, capace di auto-propagarsi. Un altro tipo può essere la **backdoor**. Anch'essa può nascondersi come un trojan e viene usata per fornire accessi privilegiati a determinate parti del software. Poi, citiamo anche lo **spyware** che, come dice la parola, serve a ottenere informazioni sensibili all'insaputa dell'utente.
- **malware per il controllo e l'attacco**. Di questa categoria fa parte il **rootkit**, che permette l'accesso e il controllo al sistema. Il **ransomware**, invece, è in grado di cifrare i dati contenuti nel sistema infettato. Letteralmente vengono sequestrati dei dati e poi si chiede un riscatto. Citiamo anche il **botnet**, che è una serie di host compromessi che si collegano ad un server centrale. Così si creare una rete di componenti infetti che si diffondono molto velocemente.

Tra i malware più famosi citiamo il **Worm di Morris**, un worm non dannoso rilasciato da un computer dell'MIT, che aveva l'obiettivo di individuare la dimensione di internet. È scritto in C ed è composto da nove file. Un altro famoso caso che ha coinvolto un malware, in maniera benefica, riguarda **il caso di San Bernardino**, la sparatoria in un centro sociale per disabili dove hanno perso la vita quattordici persone. L'FBI aveva bisogno di accedere all'IPhone di uno dei due attentatori pakistani ma, a causa dei codici di protezione Apple, non risultò possibile. Nonostante le pressioni iniziali, alla fine Apple fu costretta a rispettare un'ordinanza giuridica e permise all'FBI di creare una backdoor per entrare all'interno del telefono. A differenza di Apple, Android, negli anni, ha sempre dimostrato di essere vulnerabile a numerosi attacchi, tra cui uno in cui bastava un **MMS**, un breve video con un malware, per compromettere la sicurezza del dispositivo.

ANALISI DEI MALWARE

Analizzare un malware significa identificarlo, difendersi ed eliminarlo, sviluppando contromisure adeguate affinché non infetti più. Esistono due tipi di analisi di un malware:

- **analisi statica**. Definisce le metologie per l'analisi del codice e della struttura del malware. Durante questa analisi, esso non viene mai eseguito. Un malware è scritto tramite linguaggi di alto livello, mentre l'analisi viene eseguita tramite assembly. Usando dei disassemblatori è possibile generare del codice assembly per essere analizzato e compreso durante l'analisi.
- **analisi dinamica**. Dopo l'analisi statica, si parte con quella dinamica. Si esamina il malware durante la sua esecuzione, osservandone il comportamento. In questo caso bisogna procedere con attenzione, ricordandoci che stiamo sempre analizzando un software dannoso e, senza le adeguate protezioni, si rischia molto. Per questo, utilizziamo una macchina virtuale, collegata ad una rete air-gap, cioè disconnesso da Internet. Durante l'analisi dinamica si può procedere tramite due approcci. Quando parliamo di **white box**, vengono analizzati tutti quegli aspetti che conducono allo stato del sistema prima dell'infezione, procedendo verso lo stato dopo l'infezione. Di solito si utilizzano debugger ed editor. Invece, con l'approccio **black box** ci focalizziamo soltanto sugli effetti derivanti dall'esecuzione del malware, senza ottenere informazioni dettagliate.

L'analisi di un malware, in ambito mobile, risulta molto più complessa. In generale perché gli smartphone hanno una rete molto più ampia dei desktop e i sistemi operativi sono molto più complessi. Inoltre, gli strumenti mobile per il rilevamento di malware non sono così efficienti.