

APPUNTI SICUREZZA



Andrea Ierardi

SOMMARIO

LEZIONE 1/10/18.....	5
Programma	5
Introduzione	5
LEZIONE 5/10/18.....	9
Classificazione delle vulnerabilità:.....	9
Requisiti di sicurezza:	10
LEZIONE 08/10/18.....	12
Attacchi passivi e attivi.....	12
Sviluppo di un attacco – Cyber Kill Chain.....	14
LEZIONE 12/10/18.....	16
Phishing e social engineering	16
Statistiche.....	16
DDoS	17
Syn Flood (inondazione di SYN)	17
CLOGGING (intasamento).....	17
BANDWIDTH (larghezza di banda).....	18
Protezione	18
LEZIONE 15/10/18.....	19
Gestione degli incidenti:	19
FASI di Reazione ad un attacco	19
Meccanismi di protezione.....	20
Protocollo arbitrato	20
Protocollo giudicato.....	20
Protocollo autogarante	20
Crittografia	21
Crittografia ad uso generale e ristretto	21
Chiavi nei sistemi crittografici	21
SISTEMI A CHIAVE SIMMETRICA	22
One-Time PAD	23
One-Time Pad, svantaggi	23
LEZIONE 19/10/18.....	25
Cifrari a flusso	27
RC4 (Rivest cipher 4)	27
Attacco a Forza Bruta	29
LEZIONE 22/10/18.....	32
Cifrario a flusso e rc4 continua..	32
Cifrari a blocchi (a chiave simmetrica)	36

Principi di SHANNON	37
I cifrari più famosi : AES, DES	38
AES.....	38
DES.....	42
LEZIONE 26/10/18.....	43
Modalità di cifratura dei messaggi lunghi ECB.....	43
Vantaggi di ECB:.....	44
LEZIONE 05/11/18.....	48
CBC (Cipher Block Chaining)	48
CTR counter mode.....	51
OFB (Output Feed Back).....	53
CFB: Cipher Feed Back:.....	55
Uso di CTR in pratica: le modalità GCM e CCM	59
GCM (Galois Counter Mode)	59
CCMP (Counter Mode/CBC-MAC)	60
Cifratura a livelli diversi della pila	61
Cifratura a livello applicazione	61
PGP (Pretty good Privacy)	62
Cifratura a livello di rete	63
LEZIONE 09/11/18.....	64
I PKCS, standard de facto.....	64
Cifratura a livello applicazione: PGP	65
CIfratura livello presentazione: TLS/SSL	66
CIFRATURA A LIVELLO DI RETE: IPSec.....	67
CIFRATURA A LIVELLO DATA-LINK: WPA2	67
Approfondimento di DES	68
Digressione su DES.....	68
3DES (DES triplo).....	69
3-DES standardizzato.....	70
ARinjndael.....	71
LEZIONE 12/11/18.....	74
Riservatezza	75
Cifrario a chiave pubblica per la riservatezza	76
Utilizzo delle chiavi pubbliche	77
Classificazione dei cifrari in base agli attacchi a cui sono vulnerabili	78
Attacco cipher-text only.....	78
Attacco known plaintext.....	78
Attacco chosen plaintext.....	79
Attacco chosen ciphertext.....	79

Problema dello scambio delle chiavi Asimmetriche.....	79
Algoritmo RSA	80
LEZIONE 16/11/18.....	85
RSA continua.....	85
Struttura padding RSA PKCS1 v1.5:.....	86
PGP posta elettronica (schema ibrido).....	87
SSL Handshake protocol	89
LEZIONE continua (part 2)	90
Firma digitale	90
Funzioni hash e collisioni	93
LEZIONE 23/11/18.....	96
Funzioni Hash continua	96
Funzioni hash crittograficamente sicure.....	97
Esempio di uso dello XOR per una funzione hash con struttura di Merkle-Damgård.....	99
SHA-3.....	101
Problema integrità dei dati:.....	103
LEZIONE 26/11/18.....	104
Esistenza di funzioni hash crittograficamente sicure	104
Aspetti di integrità:	106
Integrità a chiave simmetrica.....	106
KEYED HASH o HASH CON CHIAVE:.....	107
HMAC-Hash Message Authentication Code.....	109
FUNZIONAMENTO HMAC:	110
Riservatezza e integrità in contemporanea	111
AEAD-Authentication and Encryption with additional data:	111
LEZIONE 30/11/18.....	113
MAC, HMAC continua	113
SSL/TLS Handshake	114
SSL (Secure socket layer)	115
TLS (Transport Layer Security).....	116
Protocollo di Diffie-Hellman:	119
FUNZIONAMENTO DIFFIE-HELLMAN.....	119
IL PROBLEMA DEI LOGARITMI DISCRETI	120
Scambio di chiavi basato su D- H con TLS.....	122
proteggersi da man in the middle	123
Autenticazione della chiave pubblica.....	124
Certification Authority (CA).....	126
LEZIONE 03/12/18.....	128

Certificati e CA continua.....	128
Certificati di revoca (CRL – Certificate Revocation List)	129
Il formato dei certificati X509	129
LEZIONE 07/12/18.....	131
Discussione sulla differenza tra meccanismi per l'integrità	131
DSA (Digital Signature Algorithm)	133
ECC (elliptic curve cryptography)	136
LEZIONE 10/12/18.....	138
Diffie-Hellman A curve ellittiche	138
DSA a curve ellittiche.....	139
IES (Integrated Encryption standard)	143
Cifratura whatsapp(non la chiede):.....	145
LEZIONE 14/12/18.....	146
Autenticazione di utente o entità: schema AAA.....	146
Come ottenere l'autenticazione.....	146
EVITARE TRANSITO CREDENZIALI IN RETE- SISTEMA CHALLENGE RESPONSE.....	149
MECCANISMO BASATO SUL TIMESTAMP-DATA E ORA	150
Autenticazione TLS con RSA	151
Diffie-hellman con RSA.....	152
LEZIONE 17/12/18.....	153
Dispositivi di sicurezza.....	153
Perimetro di sicurezza.....	153
Vpn.....	154
FIREWALL	155
SCHEMA DI RETE PRIVATA E PUBBLICA CON FIREWALL:.....	156
DMZ (Demilitarized zone).....	156
VANTAGGI DMZ	157
LEZIONE 21/12/18.....	158
Politica di sicurezza (costi).....	158
Proxy FIREWALL.....	159
Il filtro di pacchetti.....	161
Strategia di filtering	162
Ripasso ACK three way Handshake:	163
Filtro di pacchetti dinamico (o statefull).....	164

PROGRAMMA

- Crittografia: Protocolli
- Sicurezza della rete: Dispositivi che servono a garantire la sicurezza della rete
- Sicurezza del web (NON LA METTE IN PROGRAMMA per noi)

L'esame è ORALE: in generale dà la libertà di dare l'esame quando si vuole, magari metterà delle restrizioni su periodi in cui dare l'esame, le date ancora non sono state decise però per quanto riguarda sicurezza sono date di solo verbalizzazione.

Libri: Stallings - Cryptography and network security (c'è una versione in IT ma è vecchia). L'ultima edizione è sconsigliata perché costa un botto → 130€.

Link libro 6 edizione: <https://bucket.daz.cat/9780133354690.pdf>

INTRODUZIONE

Bollettino delle vulnerabilità

CERT (Computer Emergency Response team): Gruppo che dovrebbe essere pronto a rispondere agli attacchi informatici, tutti i paesi hanno questi gruppi.

Gruppo che si occupa della sicurezza informatica, è uno tra quelli più famosi
<https://www.kb.cert.org/vuls>

Contiene le vulnerabilità recentemente trovate recentemente.
<https://www.kb.cert.org/vuls/id/581311>

Vulnerabilità in TP-Link

Siamo di fronte a un Software che ha problemi nell'autenticazione, si possono eseguire applicazioni java senza averne i privilegi inoltre riceve dei dati deserializzati senza fare nessuna verifica e questo comporta che ci sono delle vulnerabilità che l'attaccante può sfruttare.

- 1) Manca l'autenticazione per funzioni critiche, ovvero dati non fidati (input dati all'utente il quale il sistema non si deve fidare, ma in questo caso non è così)

Soluzione:

In questo caso non c'è una soluzione disponibile

<https://www.kb.cert.org/vuls/id/598349>

Permette di far sì che un attaccante faccia finta di essere un proxy così da intercettare tutto il traffico che passa, così da sniffarle ma anche modificarle, è un problema di protocollo.

Soluzione:

Non dovreste farlo, si può fare in un altro modo

<https://www.kb.cert.org/vuls/id/906424>

Il task scheduler di win permette di fare una privilege escalation, ovvero un utente che non ha privilegi da admin può comunque fare una scalata e diventare un amministratore, potendo così sovrascrivere file.

<https://www.kb.cert.org/vuls/id/982149>

I processi

side-channel attack : attacco laterale non diretto al sistema, dipende dal cache L1 e dipende da un cache miss ovvero l'attaccante ottiene info sul sistema tramite il tempo di esecuzione maggiore dovuto alla cache miss. Alcuni sistemi per velocizzare vanno a prendere delle info ancor prima che vengano eseguite, questo permette all'attaccante guardando il tempo di esecuzione di avere parametri importanti di sistema.

soluzione:

Aggiornare il sistema(update del BIOS)

<https://www.kb.cert.org/vuls/id/787952>

Android e Apple app contengono vulnerabilità:

Non verifica correttamente i certificati di entità che si devono autenticare, ovvero accetto che la entità sia quella che mi viene detta senza far verifiche.

uso di hard-coded credential: ovvero info che sono cablate nel Software e quindi recuperabili.

Quindi ci sono delle app preinstallate su alcuni device che sono installate e possono prendere il controllo del sistema.

Soluzione:

non installare app a caso, tipo come Spotify crackato che mi ha fatto scaricare cagge ed è stato bloccato poche settimane fa

<https://www.kb.cert.org/vuls/id/857035>

Internet key Exchange v1:

Se la chiave è debole è facile da ricalcolare tramite attacchi a dizionario/Bruteforce

Soluzione:

usare pass robuste

Attacco al protocollo è diverso da un attacco all'implementazione infatti un attacco al protocollo tocca tutti non il singolo dispositivo!

<https://www.kb.cert.org/vuls/id/641765>

Kernel Linux vulnerabile ad attacchi denial of service

Denial of service (DOS): è un attacco che mira a mettere fuori uso un servizio (DDOS nel caso in cui sia distribuito)

E' un attacco al riassemblamento dei frammenti(del pacchetto)(che doveva essere risolto con ipv6)

Soluzioni: Patch, configurare le impostazioni di default.

Ci sono varie app TCP vulnerabili ad attacco DoS.

Attacco: Resource Exhaustation:

Chiamate molto costose al sistema

Soluzione: Apply patch

Questi attacchi sono importanti in molti sistemi che usano dati importanti.

Per esempio Facebook un anno fa ha inserito 2 software (protezione privacy, e uno per il caricamento di video di compleanno). L'introduzione del secondo ha creato dei varchi nel primo. Queste vulnerabilità sono state utilizzate dagli attaccanti per raccogliere info (personal) importanti .

ICS-CERT:

<https://ics-cert.us-cert.gov/advisories>

Sistemi di controllo industriale sono sistemi che servono per l'automazione industriale (industria 4.0 che è l'evoluzione estrema in cui i vari dispositivi sono collegati tra loro dialogano con applicazione di IA ottimizzano il lavoro e hanno flessibilità di comunicazione con i vari componenti che automatizzano l'industria). Parliamo quindi di sistemi per regolare catene di montaggio, fabbriche di auto, centrali elettriche, distribuzione acqua etc. Un attacco a questi sistemi può comportare perdite ingenti a carico dell'industria attaccata, attacchi a sistemi come acqua elettricità etc. possono avere ripercussioni peggiori.

Sono nati sistemi di automazioni in un mondo tutto loro, aggiornarli è molto difficile (in quanto non si può fermare la produzione con leggerezza per un update) e quindi sono sistemi che hanno una vita molto lunga: qui vediamo le vulnerabilità in questo mondo

Emerson AMS device manager: sfruttabile in remoto, impropria gestione del controllo degli accessi e dei privilegi. E' permessa esecuzione arbitraria di codice la soluzione è sempre di applicare delle Patch.

Fuji Electric Alpha5 Smart Loader:

exploit=è un termine usato per identificare una tipologia di script ,virus ,worm o binario che sfruttando una specifica vulnerabilità presente in un sistema informatico, permette l'esecuzione di codice malevolo su di esso con lo scopo di far ottenere all'attaccante l'acquisizione di privilegi amministrativi.

sono noti malware pubblici che sfruttano questa vulnerabilità: buffer overflow (i dati che vengono allocati in un buffer se vengono scritti dati la cui lunghezza eccede quella del buffer quel che rimane viene scritto in un'altra area di memoria e bisogna vedere come i dati vengono sfruttati in quell'area di memoria potendo potenzialmente causare problemi).

La soluzione sarebbe fare dei controlli sulla lunghezza input banalmente (che vengano rispettati i limiti)

ESERCIZIO LABORATORIO

In questo caso se metto un input troppo grande, vado a sovrascrivere l'indirizzo di ritorno quindi succede che ritorna un valore senza senso(segmentation fault!).

Quindi inserendo dei dati troppo lunghi, essendo vulnerabile a un buffer overflow, abbiamo un DOS quindi il programma non funziona più. Cosa fare? Guardare codice, è intuitivo. Questo è un esempio in cui buffer overflow si può usare per entrare nel sistema. Nell'esempio proposto bastava scrivere abbastanza caratteri da sovrascrivere la "F" con una "T" in modo tale da poter bypassare il controllo.

La scala della gravità dell'attacco arriva fino a 10.

Abbiamo visto vari virus a livello industriale:

Tech data: Mancanza di autenticazione per funzioni critiche(DoS)

Scalances x switches: Validazione dell'input impropria, per esempio arriva l'input e non verifico il formato, si ottiene DoS.

Quindi abbiamo visto 3 vulnerabilità principali:

- Buffer Overflow
- Improper Input Validation
- Scalata di privilegi

Visto che stiamo parlando di sistemi a controllo industriale, parliamo del sito darkreading . Sistemi che hanno password di default che non vengono cambiati. Ci sono dei cataloghi recuperabili in rete. Per esempio Siemens(sistema di controllo industriale)

=====

<https://www.shodan.io/>

Shodan - Search engine

Permette di controllare i dispositivi collegati a Internet (IoT). Dei ricercatori hanno pubblicato un articolo a gennaio che il nostro metodo ha verificato più di 500mila IoT industriali che sono stati trovati a internet vulnerabili, con Software non aggiornati o altro

Abbiamo visto un po' di attacchi la volta scorsa, cominciamo a cercare di classificare alcune cose: gli attacchi che abbiamo visto non sono tutti allo stesso livello, ma sono attacchi a delle vulnerabilità di tipo molto diverso. Alcune a livello di **implementazione**. Altre a livello di **Protocollo** o **Progettazione**. Un altro livello potrebbe essere che il protocollo è corretto, l'implementazione viene fatta bene ma gli algoritmi utilizzati sono **deboli**. Il problema che sia l'**algoritmo** è molto meno frequente rispetto all'implementazione. Sopra all'implementazione anche se uno ha un sistema ben implementato ci possono essere **problemi di procedure**: il sistema è ok, ma la procedura prevede per esempio che la password non sia cambiata con una certa frequenza, oppure riguarda il trasferimento di dati. Sopra la **procedura** rimangono ancora gli **utenti** che sono il problema centrale della sicurezza sia perché possono essere poco formati e sia perché il comportamento dell'utente utilizzatore è imprevedibile.

CLASSIFICAZIONE DELLE VULNERABILITÀ:

- 1) **Utenti**: Se l'utente è pigro e non segue le procedure, se l'utente non è consapevole dei rischi, la sicurezza diventa difficile.
- 2) **Problemi di procedure**: La procedura non prevede che la password sia cambiata con frequenza, Esempio: in un sistema sanitario hanno scritto dati importanti in un CD ed è stato spedito, questo CD è stato intercettato.
- 3) **Implementazione**: Realizzazione che non ha tenuto conto della sicurezza. Si può risolvere con la patch.

4) **Problema di protocollo o/ progettazione**:

Problema di protocollo → è un problema che si estende di più.

Vulnerabilità di progettazione → Per esempio software per un router con password non modificabile.

5) **Algoritmi deboli**.

Più si scende di livello (verso il 5) più è raro avere un problema in quel livello lì ma aumenta anche la difficoltà di risoluzione del problema.

Se guardiamo che tipo di richieste che si fanno alla sicurezza: **Riservatezza dei dati, Identificazione dell'utente che vuole accedere al servizio**

REQUISITI DI SICUREZZA:

- **Riservatezza:** Assicura il controllo individuale o l'influenza delle informazioni solo a chi può accedervi.
- **Integrità dei dati:** I dati o i messaggi non possono essere alterati da una terza parte non autorizzata. L'integrità richiede che ogni alterazione deve essere rilevabile (in quanto è normale che possano esserci delle modifiche - *Integrità del sistema*)
- **Identificazione sicura:** Sapere esattamente qual è l'interlocutore che ho davanti in modo da poter fare il controllo degli accessi
- **Non ripudiabilità (o non rinnegabilità):** Se qualcuno ha prodotto dei dati non può negare di averlo fatto (es: firma dei dati)
- **Disponibilità:** Il servizio deve essere sempre disponibile (es: sistema per le operazioni chirurgiche) e resistente ai DOS/DDOS.

Integrità e identificazione sono meccanismi di **autenticazione** e c'è di mezzo l'**autenticazione di sorgente**: ovvero riuscire a stabilire con certezza da chi è stato scritto un determinato messaggio. Il che dà anche integrità poiché il messaggio non è stato alterato.

Questa classificazione da un'idea delle sfaccettature che vedremo e come giocano l'una con l'altra. La sicurezza non è un concetto relativo: un sistema è sicuro rispetto a certi requisiti. Non è sicuro in senso assoluto perché un sistema proprio per definizione non è mai sicuro al 100%, e non è sicuro in senso assoluto anche per i rischi diversi a seconda di ciò che si deve proteggere.

E' necessaria quindi una valutazione dei rischi, dei costi a cui si vuole incorrere per poter mettere in sicurezza il sistema e tutto questo suggerisce i meccanismi da adottare per la messa in sicurezza. I costi sia economici che in termini dell'usabilità del servizio (se decidiamo di mettere una porta blindata e una con 4-5 serrature deve valerne la pena a livello di servizio e sbattimento).

Un aspetto molto importante legato alla sicurezza è l'**usabilità dei sistemi**: si vorrebbe un sistema facile da usare, però in genere dal punto di vista della sicurezza è un problema: **se un sistema informatico è troppo difficile da usare gli utenti si scocciano e non lo usano**. Se richiedo all'utente di inserire una password complicatissima l'utente la scrive sul pezzo di carta e tanti saluti. Se il sistema non è pensato correttamente per gli utenti, l'utente non coopererà.

Un altro esempio: Navigando sul web si può incorrere in siti con certificati non validi: L'utente medio li skippava (quindi tutti gli avvisi di sicurezza erano completamente ignorati), mentre adesso è più difficile proseguire la navigazione nel caso in cui un certificato non sia valido.

INTRODUZIONE AL PROBLEMA DELLA SICUREZZA(CENNI STORICI)

Internet è nata come un progetto militare americano legato all'utilizzo più efficace delle risorse computazionali nel tempo: qualche grosso computer in grosse università. Il problema era sfruttare tale potenza computazionale da remoto: il progetto quindi nasce in un contesto in cui gli utenti erano un gruppo limitato e gli interessi di ricerca erano limitati, dopo poco esplode l'Internet attuale a livello sociale ed economico. Il problema è che i protocolli di rete sono rimasti invariati: TCP, IP, ARP etc. sono protocolli di cui ci si fida, il dispositivo non fa il minimo controllo sulle risposte dei protocolli. Quello che si vuole fare adesso è cambiare i protocolli ma deve essere una cosa graduale (vedi IPv6) non è semplice. Dato che cambiare i protocolli è difficile si cerca quindi di sovrapporre la sicurezza su protocolli non sicuri con un successo ovviamente limitato.

Questo problema si riporta a un altro livello: data un'azienda che nasce per lavorare in un certo modo quindi senza internet a questo punto si trova di fronte a dei rischi che non erano pensati.

Quello che si sarebbe dovuto fare sarebbe stato partire e ricostruire basandosi su meccanismi che si basano sulla sicurezza.

LABORATORIO: ARPSPOOF CON HUB E SWITCH

Abbiamo già visto attacchi alla **disponibilità** cioè attacchi DoS. Ora proviamo in pratica a fare degli attacchi alla riservatezza, che forse è la più antica.

- **Attacchi alla riservatezza:** Risale al passato, attacchi in campo militare
- Attacco alla riservatezza in un sistema dove si è collegati a un HUB
- Attacco alla riservatezza in un sistema dove si è collegati tramite SWITCH.

Per fare questo si usa il tool (software) net kit → un sistema di emulazione di reti e non di simulazione

Un sistema di simulazione mira a simulare il sistema per poterlo analizzare dal punto di vista delle prestazioni mentre il sistema di emulazione non ha l'ambizione di simulare le prestazioni ma si accontenta di simulare le funzionalità.(Esso è basato su user mode Linux sono dei processi che girano a lv utente e rappresentano altre macchine virtuali che possono essere collegate in rete. Si può anche simulare uno switch)

SNIFFARE IN PRESENZA DI HUB

Con TCPdump sniffiamo le connessioni in presenza di hub il quale manda messaggi broadcast, quindi mettendoci in ascolto (essendo sulla stessa rete) se la connessione non è cifrata tipo SSH riusciamo a sniffare le informazioni e analizzarle con wireshark

SNIFFARE IN PRESENZA DI SWITCH

La richiesta ARP viene inoltrata in broadcast quindi è l'unica cosa che vediamo sniffando sulla rete se siamo in presenza di uno switch a differenza di un hub in quanto lo switch non manda i dati in broadcast. Infatti, se proviamo a fare un TCPdump da pc3 e successivamente se dal pc2 facciamo ping IP PC1 e poi controlliamo su wireshark da pc3 cosa abbiamo sniffato vedremo solo una richiesta di ARP, non vediamo nemmeno la risposta perché già la risposta alla richiesta ARP non è un messaggio broadcast.

L'attaccante su pc3 deve cercare di convincere pc2 di mandargli i pacchetti che dovevano andare alla vittima. (man in the middle)

Il comando ARPSPOOF dice: sull'interfaccia che specifico, manca a pc2 una risposta ARP dicendo "L'IP di pc1 ha questo MAC (quello di pc3)" impersonando pc1. Bombardando di ARP reply pc2, esso lo accetterà e invierà i pacchetti a pc3 invece di pc1.

La risposta di pc1 a pc2 non la vediamo perché essa è diretta e passa dallo switch il quale (a differenza dell'hub) non manda messaggi broadcast a tutti ma inoltra la richiesta al dispositivo corrispondente, e non riusciamo ad intercettarla, riusciamo però a vedere cosa digita pc2, (nel nostro caso la password) se gli facciamo credere che noi siamo pc1.

Per questo esercizio abbiamo dovuto usare ifconfig per vedere su ogni macchina l'indirizzo IP e la interfaccia di rete sulla quale sono situate le macchine.

Poi con il comando : /sbin/sysctl -w net.ipv4.ip_forward=1 inoltriamo i pacchetti a pc1 per fermare questo processo dobbiamo ridare lo stesso comando ma con 0 al posto di 1 /sbin/sysctl -w net.ipv4.ip_forward=0.

Successivamente usiamo il comando arpspoof -i interfaccia -t IP-vittima IP-hostdaimpersonare&

la & serve per mandare il processo in background e successivamente con il comando TCPdump -s0 -w /hosthome/file.pcap sniffiamo il traffico e lo salviamo in un file .pcap che può essere successivamente analizzato.

ATTACCHI PASSIVI E ATTIVI

Ci sono due tipi di attacchi: passivi e attivi.

Facciamo un modello del flusso normale dell'informazione. Il flusso normale dell'informazione si può rappresentare così:

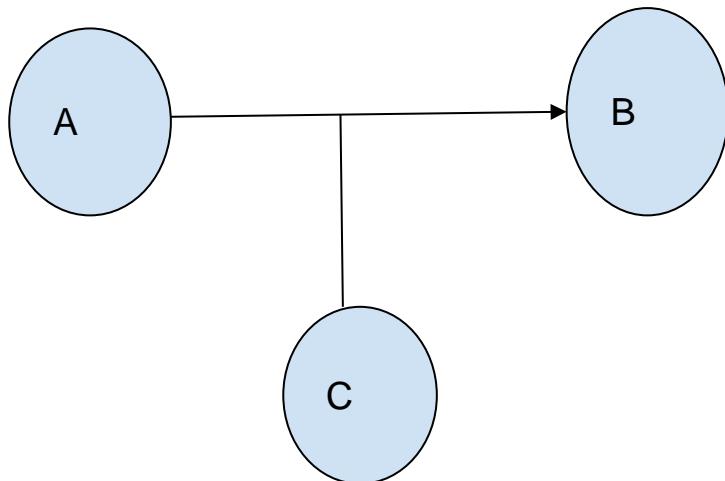


Introduciamo ora l'attaccante in un attacco passivo.

ATTACCO PASSIVO

Un attacco passivo è un attacco in cui l'attaccante riesce a venire a conoscenza di quello che è trasferito da A a B ma non altera il flusso. A e B non se ne accorgono.

L'attaccante si mette in ascolto. Quindi non altera il sistema né il suo funzionamento.

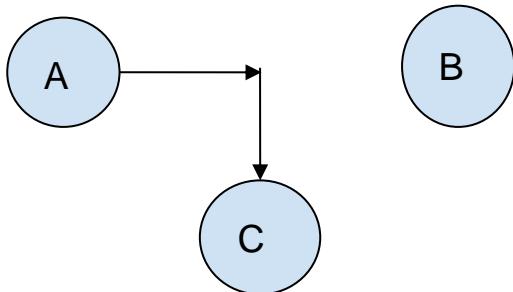


ATTACCO ATTIVO

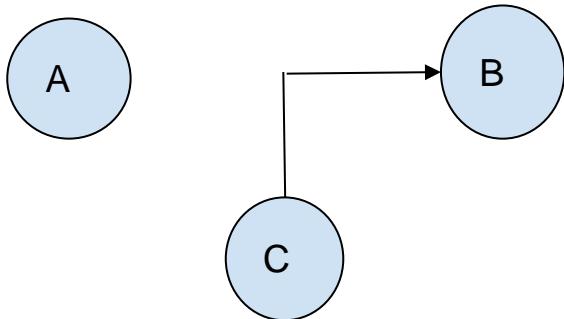
Un attacco attivo è un attacco in cui il flusso viene alterato, per esempio C finge di essere A oppure un'altra situazione è quella in cui i dati vengono alternati da b.

Quindi altera il sistema:

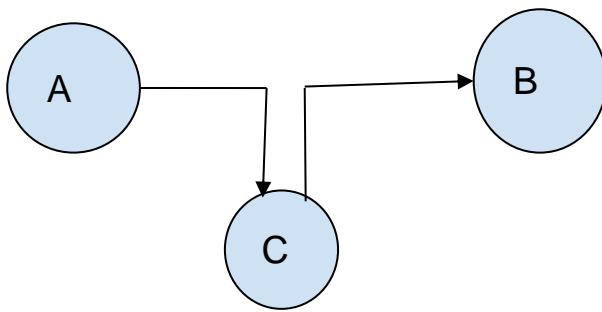
In questo caso B non lo vede neanche viene intercettato da C.



In questo caso C si finge di essere addirittura "A" quindi genera lui il flusso.



Qui i dati vengono modificati e mandati a B.



L'attacco di sniffing in una rete con hub era passivo.

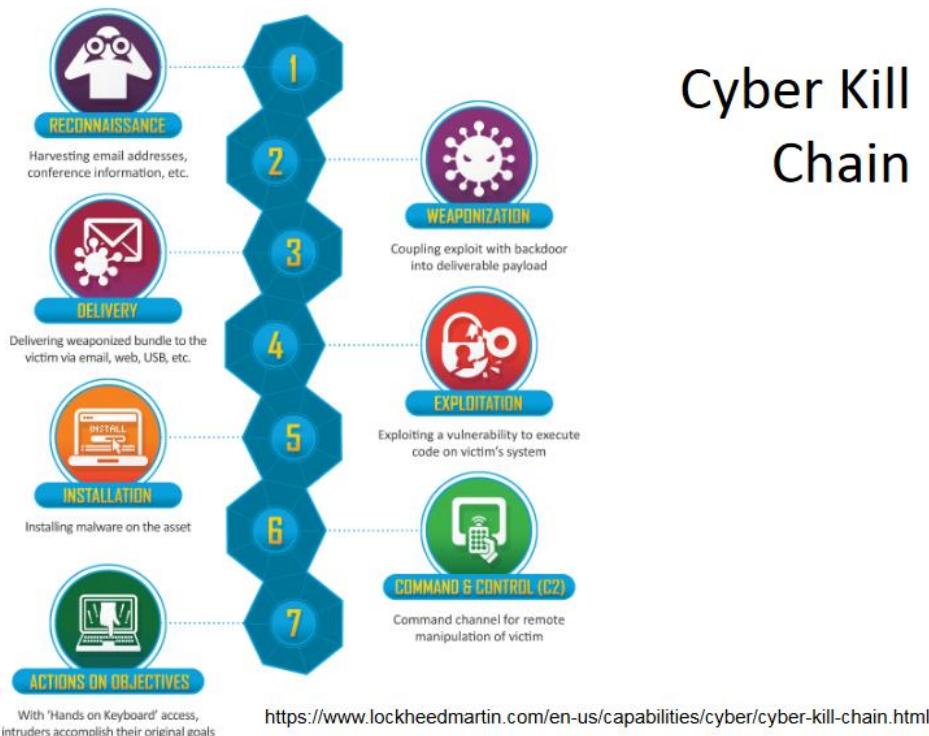
L'attacco di sniffing in una rete con switch era passivo perché all'inizio anche se c'è un denial of service (che è attivo) (ho controllato su sto sito

<https://iacopokahl.com/sviluppo/crittografia/attacchi-all-a-sicurezza/> la prof si deve essere sbagliata) poi inoltra i dati senza modificarli quindi a livello logico è passivo.

Un attacco passivo in un certo senso è un attacco in se meno grave di un attivo perché il sistema continua a funzionare nello stesso modo in cui funzionava prima quindi non ci sono interferenze dirette dell'attaccante però è sbagliato pensare che attacco passivo non sia grave, perché un attacco passivo prelude a un attacco attivo.

SVILUPPO DI UN ATTACCO – CYBER KILL CHAIN

Come si sviluppa un attacco: è un modo per modellare l'attacco.



L'attaccante segue di solito una **Cyber Kill Chain** che deriva dalla Kill Chain che ha un'impronta militare.

- **Reconnaissance** → fase in cui l'attaccante raccoglie informazioni sul sistema da attaccare ed è una parte **passiva** dell'attacco
- **Weaponization** → l'attaccante prepara le armi(riproduzione dell'exploit con backdoor)
- **Delivery** → arma trasferita sul sistema da attaccare(pensiamo ad un malware via e-mail, USB...)
- **Exploitation** → fase in cui malware ha effetto e quindi viene realizzato l'attacco(sfrutta vulnerabilità del sistema per eseguire codice sul sistema della vittima)
- **Installation** → viene installato software per controllare la macchina vittima
- **Command & Control** → Canale di comando remoto per manipolare la vittima. Fase in cui effettivamente controlla la macchina via remoto
- **Actions on Objectives** → Svolge il lavoro previsto.

Quello che c'è sempre in ogni attacco è la Reconnaissance, quindi la fase in cui si prendono info sul sistema.

LABORATORIO: RECONNAISSANCE SU HAL E SIRIO

L'attaccante sa che esistono due reti, una galactica.org e l'altra sirio.net e sa che ci sono due server di posta elettronica. Prima di attaccare ha bisogno degli indirizzi IP di queste reti e capire quali host ci sono in queste reti e quali servizi sono attivi.

Prima cosa scoprire indirizzi IP con dig o nslookup.

In questo sistema ci sono diverse macchine e attaccante lavora sull'host hal.

Dobbiamo raccogliere informazioni sulla rete. §

Prima cosa scoprire indirizzi IP con dig o nslookup

Es

dig www.google.com mi arrivano un bel po' di informazioni. Cosa ci interessa? E' Answer Section abbiamo indirizzo IP del server principale. (dove avremo l'indirizzo IP della macchina (principale) del server di Google.) (è l'indirizzo IP di una macchina non dell'indirizzo di rete di un dominio)

nslookup è molto più compatto, e analogo a dig.

Sono comandi che consultano il DNS per noi quindi riescono a darci l'indirizzo IP di un host di cui dobbiamo conoscere il nome. (nel nostro esempio noi non conosciamo il nome del server di posta elettronica della rete /dominio)

Il primo non ci interessa è l'IP del main server che viene consultato per avere questa risposta, il secondo sì perché è l'indirizzo di destinazione che è uguale a quello di prima.

In questo modo attaccante raccoglie info sul sistema.

Però l'attaccante conosce il nome delle reti non conosce il nome delle macchine sulla rete... però sa che ci sono dei server di posta elettronica e farà delle ipotesi per capire come si chiamano i server di posta elettronica e eseguendo il comando pig su questi nomi (che dovrà indovinare) riuscirà a scoprire l'IP delle macchine della posta elettronica. L'attaccante vuole scoprire tutti gli host e i servizi presenti sulle reti.

Si usa il comando nmap → fa scan della rete

Cosa fa?

Cicla su tutte macchine e manda pacchetti e vede cosa rispondono queste macchine

Manda pacchetti malformati di vario genere per vedere come risponde la macchina (come per esempio il **sistema operativo** della macchina)

Fare SCAN di una rete non è accettabile se non è la propria rete. E' come andare in un parcheggio e cercare di vedere se le portiere delle macchine sono aperte. Non si fa.

Prima bisogna capire indirizzi IP e poi da lì non si sa se sono reti di classe C o altro e dovremmo fare le nostre ipotesi. Con IP possiamo utilizzare nmap e capire i servizi della rete.

Ho il nome del dominio mi serve il nome dei mail server (posta elettronica) dei domini galactica.org e sirio.net. Con un po' di inventiva si può trovare che il nome del server di galactica.org è mail.galactica.org e di Sirio è mail.sirio.net

A questo punto l'attaccante conosce quali sono le varie porte aperte, i servizi attivi, IP e sistema operativo.

A questo punto l'attaccante continuerà dopo aver raccolto informazioni. L'attaccante ha intenzione di mandare posta elettronica a suo piacere come per esempio SPAM con mittente fasullo. Noi sperimenteremo questa possibilità e cercheremo di bloccarla in un semplice modo cioè non permettendo di non fare relay. Si cerca un attacco di tipo Phishing. Cioè si cerca di ottenere informazioni utili (dati da utenti) attraverso uno spam di e-mail con mittente fasullo.

ISTRUZIONI ESERCIZIO SUL DIR

PHISHING E SOCIAL ENGINEERING

Il protocollo SMTP accetta qualsiasi cosa dall'utente senza verificare, quindi una volta che otteniamo il dominio, il server SMTP prende per buono tutto, la stessa cosa succede per il protocollo ARP. Questa fiducia eccessiva dell'ARP si può sfruttare per fare lo sniffing in una rete con SWITCH.

PHISHING

A cosa serve il fingarsi un utente diverso?

Il phishing: l'attaccante convince gli utenti a dargli spontaneamente informazioni di vario genere: Nello stack dei livelli di attacco vi erano in cima gli UTENTI. Il phishing entra sotto la categoria del social engineering che significa che viene fatta leva sulle persone per ottenere informazioni o anche per ottenere che la persona faccia qualcosa che risulti utile all'attaccante.

Esistono dei mercati su cui vengono vendute le informazioni che vengono raccolte: Costi di carte, vendita di credenziali conti in banca, vendita di password per sistemi scada, account Netflix, NowTV etc. Gli acquisti di login di solito servono per ripulirsi, nel senso se l'utente è stato segnalato di solito compra una nuova identità per ripulirsi di ciò che ha fatto in precedenza.

Report McAfee :

Picco di produzione malware 2017 dicembre, i malware per MAC stanno aumentando..

RANSOMWARE

Codici malevoli che se installati su una macchina cifrano tutti i dati e poi chiedono all'utente un riscatto in cambio della password di decodifica, ci sono anche per mobile (Android lockscreen).

La soluzione è farsi un backup per prevenire questi tipi di attacchi. È possibile prendere questi virus installando software a caso.. e bisogna stare attenti anche alle app sul play store perché sebbene dovrebbero essere controllate non sempre viene fatto. In conclusione, installate solo ciò che vi serve.

NEW MALICIOUS URLs

A volte anche solo passando il mouse (senza neanche cliccare) on mouse hover, che ti fanno partire download di virus sul pc.

NEW FACELIKE MALWARE

Serve per indirizzare like su fbFEISBUC.

NEW MALICIOUS SIGNED BINARIES

L'attaccante potrebbe ottenere la firma legittima di un'azienda quindi non ci si può fidare.

STATISTICHE

Parliamo di statistiche:

Statistiche di incidenti, dove sono localizzati, soprattutto sulle Americhe. I settori maggiormente attaccati sono: Assistenza sanitaria ,pubblico, formazione scolastica, ordinanza.

L'assistenza sanitaria è il peggiore, forse perché sono più facili da attaccare. Ci sono malware connessi a server di controllo, ho il malware sulla macchina , è il malware stesso che si collega al server, questo elimina anche il fatto che uno sia una rete privata o protetta perché il malware è all'interno.

SPAM BOTNET

Serie di pc(rete di pc) compromessi sotto il controllo dell'attaccante, per mandare spam di vario tipo.

I server di controllo dei botnet sono soprattutto negli USA. I pc compromessi (le botnet)sono in vendita, l'attaccante compromette o utilizza macchine compromesse.

TROJAN

L'attaccante ricerca le macchine che hanno un certo software installato che può ovviamente sfruttare per poterle controllare e usarle per attacchi. Si chiamano tipicamente **Trojan** (cavallo di troia) i software presenti nella macchina pronti a tradire.

Con una breve ricerca su Google possiamo trovare la lista delle porte utilizzate dai vari trojan, ad esempio porta 21(ftp) , 23(telnet) e tanti anche sulla porta 25.

È facile che su una rete ci sia un server di posta quindi è per questo che ce ne sono vari sulla porta 25, in quanto è probabile che la porta 25 sia aperta.

Con "nmap" l'attaccante può vedere quali sono le porte aperte della vittima e vedere se c'è uno di questi software installati che permettono di collegarsi alla macchina.

Successivamente l'attaccante installa il software di controllo su queste macchine compromesse, poi ci sarà un'altra macchina compromessa(chiamata gestore) sulla quale installerà un software di gestione delle altre macchine.

Si può fare un attacco spam.

Bloccare una di queste macchine non ha effetto, e nemmeno bloccando il gestore perché non è la macchina sulla quale risiede l'attaccante. Un esempio di questi tipi di attacco sono gli attacchi DDoS.

DDOS

DDoS (Distributed Denial of Service) : Quando l'attaccante ha stabilito la sua botnet e installato il gestore, l'attaccante scompare(si eclissa) e da l'ordine al gestore di "inviare l'ordine" alle macchine sotto controllo di inondare la vittima per bloccarne l'accesso.

Quindi la vittima viene congestionata e il servizio che offre è compromesso. Un esempio di attacco DDoS è il **SYN FLOOD** e viene attuato da più computer "complici" (è un attacco distribuito), mentre un DoS è semplicemente un attacco in cui una macchina usa una vulnerabilità del sistema per far crashare un'altra macchina.

SYN FLOOD (INONDAZIONE DI SYN)

SYN FLOOD è un exploit dell'uso normale del TCP, **mira a saturare la memoria dedicata alla connessioni TCP di una macchina**: quando viene inviato un pacchetto SYN al server, quest'ultimo riserva spazio per quella connessione e aspetta una risposta finché non scatta il time-out, se scatta il time-out quello spazio viene liberato. Se le richieste SYN arrivano velocemente e in abbondanza si saturano le connessionimesse prima che il server riesca a liberarle in quanto vengono saturate prima che scatti il time-out, quando lo spazio è pieno non riesce più ad accettare connessioni (per esempio connessioni legittime degli utenti).

CLOGGING (INTASAMENTO)

Il **CLOGGING**(altro tipo di attacco DDoS) **mira invece a saturare le risorse computazionali della macchina attaccata**: ad esempio, in caso di un software mal concepito, su semplice richiesta del client il server esegue computazioni crittografiche pesanti prima di essere certo che la connessione sia stabilita e che non siano richieste a vuoto, con conseguente saturazione di risorse computazionali del server.

Es attacco più vicino al clogging:

700 persone decidono di andare all'ufficio postale a chiedere un francobollo le persone arrivano e riempiono l'ufficio postale e si mettono in coda(fila immensa), come si riconosce chi vuole davvero prendere il francobollo e chi è lì per un attacco?

È molto difficile, eppure l'attacco è easy da fare se si hanno a disposizione complici.

Un attacco di tipo Clogging può essere pensato su Diffie-Hellman, un attaccante finge un indirizzo sorgente di un utente legittimo e invia il suo K_a alla vittima, la vittima farà quindi della computazione per calcolare la chiave segreta. Messaggi ripetuti di questo tipo possono ostruire la potenza computazionale della vittima.

BANDWIDTH (LARGHEZZA DI BANDA)

BANDWIDTH(sistema più semplice di DDoS): **mira a saturare la larghezza di banda :**

La differenza principale tra questo tipo di attacco ed uno di tipo CLOGGING è che **in questo caso i messaggi che vengono inviati non hanno necessariamente un significato** e quindi possono anche essere pacchetti malformati.

Riprendendo l'idea dell'ufficio postale, se convincessimo abbastanza gente a passare tutta insieme a piedi per la strada in cui si trova l'ufficio postale, al punto che non si riesca più a passare, nessuno potrebbe più raggiungere l'ufficio e solo la gente che si trova in quella zona riuscirà a raggiungere l'ufficio perché va a piedi (Bandwidth).

Se invece il gruppo di gente malevola (o comunque sotto il controllo dell'attaccante) entra nell'ufficio postale e si mette in coda e fa richieste complicate allo sportello, sta saturando le risorse dell'ufficio: gli utenti legittimi non riusciranno a raggiungere lo sportello. Qui ci vuole sicuramente meno gente, soprattutto rispetto al caso di un ingorgo pedonale nella strada: più è lunga l'operazione che gli impiegati dell'ufficio postale si apprestano a fare, meno gente sarà necessaria (Clogging).

PROTEZIONE

Per quanto riguarda la protezione:

Se l'attaccante usa il proprio IP (primo caso) oppure se ha il controllo di un router intermedio(secondo caso) (per cui può leggere le risposte del server) **aspettare la risposta non ha senso**, perché la risposta arriverà, il server allocherà le risorse richieste e l'attacco avrà successo. Però il primo caso (l'attaccante usa il proprio IP) implica che un eventuale meccanismo di protezione a monte del server si accorgerà che quel traffico è sospetto (troppe connessioni da uno stesso IP) e le bloccherà prima che arrivino al server. Nel secondo caso, l'attaccante riesce nel suo intento. Però naturalmente è molto difficile essere arrivato al punto di poter controllare tutto il traffico.

Nel caso di un clogging oppure di un syn flood, bastano meno messaggi: ma in alcuni casi c'è la possibilità di bloccare i messaggi a monte, con un dispositivo opportuno che filtra il traffico (Firewall). Siccome i messaggi non sono troppi, questo non intasa il dispositivo (se dimensionato correttamente, naturalmente) e quindi il traffico legittimo passa, e il server non viene intasato perché protetto dal filtro. Questo vale solo se il filtro riesce a distinguere il traffico legittimo da quello illegittimo. Invece se l'attacco mira a saturare la banda, si crea un ingorgo e smette di funzionare tutto (anche il filtro) perché tutto il canale è pieno di spazzatura.

LABORATORIO

Cercheremo di fare un attacco DDoS, e più precisamente il SYNFLOOD.
Cosa proviamo a fare: un attacco DDOS di tipo SYN FLOOD.

In sostanza vengono mandati dei SYN al server, notiamo che si può fare inserendo anche un IP fasullo che non appartiene alla rete. Allora possiamo scrivere uno script per mandare dai vari pc connessi alla rete tanti pacchetti di syn da IP fasulli in modo da bloccare il server web.

La prossima lezione vedremo gli ultimi attacchi più frequenti e inizieremo a parlare della crittografia.

GESTIONE DEGLI INCIDENTI:

Non significa semplicemente cosa faccio quando mi attaccano ma è un ragionamento più complesso perché tutto deve partire sempre dal presupposto che **un sistema verrà sempre attaccato**. Non si può mai presupporre di essere sempre al di sopra di un attacco (non si può mai sottovalutare un attacco), quindi bisogna agire sotto questa ipotesi e bisognerà prendere delle **misure di protezione e monitoraggio del sistema** tramite le quali si può capire se sul sistema c'è qualcosa di anomalo.

Bisogna quindi definire in anticipo le **Procedure** in caso di attacco: per esempio **chi deve essere coinvolto in caso di attacco**. Dopodiché c'è una fase di **rilevamento** dell'attacco, quando si rileva un attacco si mettono in atto le **procedure** e si chiama la persona coinvolta in caso di **attacco**. Chi ha titolo di essere coinvolto? Di solito un gruppo che si chiama CERT, CIRN etc. tipicamente è un gruppo tecnico, però non basta coinvolgere i tecnici che si occupano di sicurezza dell'azienda ma bisognerà coinvolgere qualcuno a livello manageriale perché dovranno essere prese delle decisioni a livello di attacco. Necessariamente bisognerà decidere cosa dire al pubblico riguardo al problema, quindi si dovrà avvertire l'URP che è la parte dell'azienda che si rivolge al pubblico, **potrebbe essere necessario avvertire qualcuno a livello legale (in caso di furto di proprietà intellettuale, dati sensibili, ecc....)**.

FASI DI REAZIONE AD UN ATTACCO

A monte c'è quindi una **reazione ad un attacco**:

1. **MITIGARE I DANNI**
2. **COME PRESENTARE L'INCIDENTE AL PUBBLICO**
3. **ASPETTI LEGALI**
4. **BISOGNA RACCOGLIERE INFORMAZIONI SULL'ATTACCO**
5. **RIPRISTINARE IL SISTEMA.**
6. **POST-MORTEM**

Per fare il (5) può essere necessario utilizzare dei backup e quindi a monte bisogna pensare a protezione e monitoraggio e fare opportuni **backup dei dati**. Inoltre potrebbe essere utile avere dei **backup dei collegamenti in rete verso l'esterno, dispositivi di protezione etc.**

L'ultima fase è la **post-mortem**: dopo che tutte le fasi precedenti sono avvenute, è importante analizzare l'accaduto, quindi è una fase di analisi:

1. **Analisi dell'efficienza del team** che se ne è occupato etc.
2. **Verifica dei problemi di monitoraggio** (è stato rilevato l'attacco in tempo?)
3. **Analisi della reazione** (la reazione è stata utile? Ha funzionato bene?)

Quindi la **fase di post-mortem è una fase di riflessione su quello che è successo** globalmente così il prossimo attacco avrà un impatto inferiore.

Le **procedure sono molto importanti** perché il momento dell'attacco è un momento di panico quindi **conviene avere delle procedure decise a monte in modo tale da non dover fare errori ancora più grossolani**.

MECCANISMI DI PROTEZIONE

Come proteggersi: Ci sono diverse classi di meccanismi di protezione.

La rete è all'interno di un perimetro di sicurezza che è sostanzialmente la "zona franca". La rete deve comunicare con l'esterno quindi questo coinvolge aspetti diversi.

Il perimetro di sicurezza (la "zona sicura") si fa in buona parte con dispositivi fisici (firewall e in parte lo si fa chiudendo dispositivi dietro porte chiuse), nel perimetro di sicurezza abbiamo già il problema dell'utente che abbiamo visto essere un grosso problema.

Tutto quello che non si può proteggere in questo modo è quello che esce verso la rete.

Come ci si protegge? Come garantire che un protocollo di comunicazione soddisfi i requisiti di sicurezza? Ci vogliono dei **protocolli di sicurezza** che soddisfino i vari requisiti come disponibilità, riservatezza, integrità, autenticazione di sorgente, identificazione sicura dell'interlocutore ecc..

PROTOCOLLO ARBITRATO

Il dialogo passa attraverso un arbitro che interviene in tutti gli aspetti del protocollo che garantisce che tutte le parti del protocollo siano eseguite secondo quanto descritto: deve essere sempre disponibile perché se per ogni comunicazione deve esserci un arbitro ci devono essere sempre arbitri disponibili (protocollo arbitrato).

Un grosso problema è che se non c'è l'arbitro o non funziona abbiamo un single point of failure. L'arbitro deve essere sempre disponibile ed entrambi gli interlocutori devono fidarsi di lui.

PROTOCOLLO GIUDICATO

Un altro modo è dire: eseguiamo un protocollo in cui i 2 interlocutori dialogano direttamente fin quando qualcosa va storto, quando succede, i 2 interlocutori fanno riferimento a una terza parte fidata, un giudice che deciderà valutando la situazione (cioè le prove di quel che sta succedendo) chi non ha seguito il protocollo.

È una garanzia a posteriori. I protocolli giudicati esistono e per fare una cosa di questo genere bisognerà raccogliere informazioni con altri protocolli, bisognerà portare qualcosa al giudice che sia un elemento che il giudice può giudicare e quindi introduciamo degli elementi aggiuntivi sul protocollo che si gestiscono con strumenti di crittografia.

PROTOCOLLO AUTO GARANTE

Non serve un giudice. Tramite sistemi crittografici, ognuno verifica con l'altro interlocutore che tutto sia andato bene in ogni parte del protocollo.

CRITTOGRAFIA

Come definizione diciamo che **per crittografia** (scrittura nascosta) **si intendono tutti i meccanismi o procedimenti che tramite delle primitive matematiche garantiscono dei requisiti di riservatezza**. Si parla più tipicamente di cifratura quando si parla di riservatezza mentre il termine crittografico è più generale.

La crittografia si divide in **due grosse famiglie**, una ad **uso ristretto** e l'altra ad **uso generale**.

CRITTOGRAFIA AD USO GENERALE E RISTRETTO

CRITTOGRAFIA AD USO RISTRETTO

Segretezza dell'**algoritmo ad uso ristretto**: sistemi crittografici dove **la sicurezza dipende dalla segretezza dell'algoritmo**.

Due interlocutori usano un protocollo segreto: quando l'attaccante viene a sapere l'algoritmo utilizzato, bisognerà usarne un altro.

Esempio: Viene scelta una fotografia di carta sulla quale uno scrive un messaggio con un pennarello, poi fotografa l'immagine e la mette in formato bitmap e la manda a qualcun altro. L'altro sapendo che è bitmap rimette il formato corretto e riesce a leggerlo. Allora possiamo supporre che questo meccanismo funzioni, ma nel momento in cui l'attaccante capisce quale è l'algoritmo che ci sta dietro bisognerà cambiarlo perché ormai risulterà inefficace e non è semplice trovare un nuovo algoritmo.

CRITTOGRAFIA AD USO GENERALE

Per ovviare a questi problemi si usano **algoritmi ad uso generale**.

L'algoritmo non è più segreto è pubblico quindi può essere integrato nel protocollo, **il meccanismo è l'utilizzo di una o più chiavi** che sono dei pezzetti di informazione **che devono essere mantenuti a loro volta segreti** e che nel contesto di questo algoritmo permettono di ottenere la sicurezza. **La scelta della chiave si ottiene con un meccanismo di generazione della key** quindi **nel momento in cui l'attaccante capisce quale chiave è usata**, l'attaccante **diventa uno degli utenti legittimi** e quindi **gli utenti dovranno cambiare la chiave**.

Le chiavi possono essere una o due tipicamente nei sistemi moderni.

CHIAVI NEI SISTEMI CRITTOGRAFICI

- **A CHIAVE SIMMETRICA (O SEGRETA):**

Una sola chiave che viene condivisa fra tutti i partecipanti al protocollo e questa chiave deve essere mantenuta segreta. I sistemi a chiave simmetrica usano una stessa chiave per cifrare e decifrare.

- **A CHIAVE PUBBLICA (O ASIMMETRICA):**

Si chiamano a chiave pubblica perché viene usata **una coppia di chiavi**: una pubblica e una privata. Questo permette di fare cose che nel contesto precedente non si possono fare. Uno dei partecipanti al protocollo utilizza una delle due chiavi e l'altro usa l'altra. Si dimostra l'integrità confrontando in qualche modo la chiave privata con la chiave pubblica, in ogni caso **le due chiavi sono legate in modo indissolubile** e sono utilizzate nelle due parti del protocollo. È **asimmetrico perché si cifra con la chiave pubblica e si decifra con quella privata**.

SISTEMI A CHIAVE SIMMETRICA

Parliamo dei **cifrari a chiave simmetrica**.

Un **cifrario** consiste in un algoritmo di **cifratura** e un algoritmo di **decifratura**. A priori deve avere anche un generatore di chiavi (un algoritmo) perché abbiamo detto che se la chiave viene compromessa gli utenti legittimi possono generarne una nuova.

I dati e i messaggi in chiaro che devono essere trasferiti (**M**) (In inglese P sta per Plaintext) passano sull'algoritmo insieme alla chiave di cifratura **K** e generano **C** ovvero il testo cifrato (Cyphertext).

L'algoritmo prende in input la stessa chiave, il messaggio C e restituisce il testo in chiaro.

Stiamo quindi parlando dei **cifrari a chiave asimmetrica**:

$$C = m \text{ XOR } k$$

ESEMPIO.

La password del corso dir ? $c = m \text{ XOR } k$

E' un possibile sistema crittografico

m 10110

XOR

k 11010

=====

c 01100

Per decifrare

c 01100

XOR

k 11010

=====

m 10110

L'algoritmo di cifratura è lo XOR, decifratura è ancora lo XOR mentre l'algoritmo di generazione delle chiavi è una generazione pseudo casuale con lunghezza uguale a quella del messaggio in chiaro. L'unico vincolo è la lunghezza della chiave, che deve essere lunga quanto il testo in chiaro.

Questo meccanismo si chiama: One-time PAD.

ONE-TIME PAD

nell'ipotesi che

Deve rispettare queste 3 condizioni:

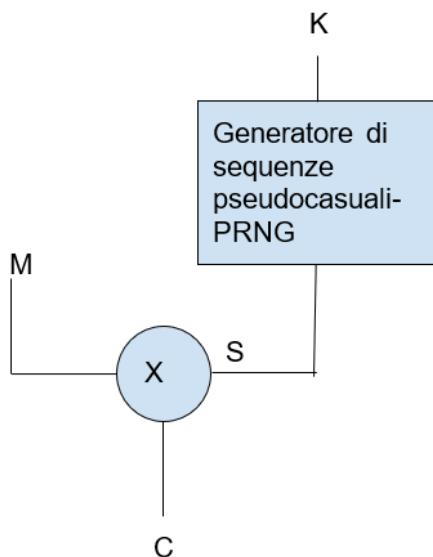
- 1) **k sia lunga quanto il testo in chiaro**
- 2) **k sia scelta in modo casuale**
- 3) **k non sia mai riutilizzata** (per questo si chiama one-time)

Se sono rispettate allora abbiamo **un sistema a segretezza perfetta**: significa che **dal testo cifrato non si può estrarre altra informazione se non la lunghezza del testo in chiaro**. Non si può significare non possibile dal punto di vista di teoria dell'informazione. Effettivamente nonostante la sua semplicità, One-Time Pad è un cifrario a segretezza perfetta. Questo cifrario durante la guerra fredda è stato usato per le comunicazioni tra Casa Bianca e i soldati.

ONE-TIME PAD, SVANTAGGI

Se bisogna trasmettere 1GB di dati, la chiave deve essere lunga come i dati, avremo quindi da trasmettere 2GB complessivamente: 1 di chiave, e 1 di dati. Un altro problema è la trasmissione segreta della chiave: devo avere un canale sicuro e protetto da interferenze esterne per mandare la chiave, per esempio potrò passare tramite un canale differente per poi utilizzarla successivamente. Anche risolvendo il problema della trasmissione della chiave, rimane comunque il problema della sua lunghezza (1gb di dati di chiave)! Inoltre, se per caso il flusso di dati e di chiave vengono desincronizzati, tutto il resto della chiave non è più coordinato con il resto del dato e quindi servono dei meccanismi di sincronizzazione. Questo è il motivo per cui non si può avere la segretezza perfetta facilmente.

Quello che si fa in pratica è sostanzialmente mantenere lo XOR, però prima inseriamo la chiave come input per un generatore di sequenze pseudocasuali (pseudo random number generator PRNG)

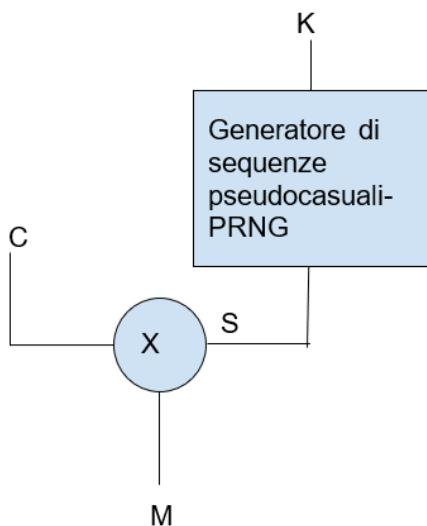


La S sta per sequenza. Con K si intende una chiave, per esempio fino a 256 bit. Quindi non di lunghezza uguale al dato. Chi riceve ha C , come si decifra? Basta mettere in XOR la sequenza pseudocasuale, quindi si mette K al PRNG, da cui si genera S che viene messa in XOR con C e si genera il messaggio M . È sempre un sistema a chiave simmetrica. La differenza è che PRGN è un algoritmo che, preso in input una chiave, genera una sequenza in un certo modo (estende K portandola ad una lunghezza adatta) e questo è fondamentale poiché potrebbe essere un problema decifrare se la sequenza non fosse la stessa.

Naturalmente tutto il peso si sposta sul generatore: prima si metteva una chiave casuale generata, qui abbiamo una sequenza pseudocasuale.

Più questa sequenza assomiglia a una sequenza casuale, migliore è l'algoritmo di cifratura(generatore) e quindi più sarà difficile da scoprire(quindi diciamo più simile a one-time PAD). L'attaccante conoscerà sia il generatore (in quanto è inglobato nel protocollo) sia il fatto che si decifra con lo XOR, l'unica cosa che non conosce è la chiave.

Se il generatore è prevedibile, l'attaccante può fare delle ipotesi sulla sequenza e decifrare naturalmente perché se l'attaccante sa la sequenza potrebbe riuscire a decifrare.



L'algoritmo PRNG espande la mia chiave casuale (che ha una lunghezza finita fino a un massimo di 256 bit) fino ad averla della dimensione giusta(cioè lunga quanto M).

L'algoritmo viene inglobato nel protocollo quindi l'attaccante lo conosce, solo che essendo pseudocasuale è più ostico per l'attaccante scoprire la chiave rispetto a un algoritmo semplice e prevedibile.

Quindi una volta espansa la chiave e generata la mia sequenza pseudocasuale la metto in XOR con C e ottengo il messaggio decriptato M.

Il vantaggio rispetto a One-Time Pad è che non devo per forza mandare una chiave lunga quanto il messaggio, ma basta una chiave più corta, la quale sarà poi estesa dall'algoritmo, con lo svantaggio che più è fatto male l'algoritmo di generazione più sarà facile per l'attaccante scoprire la sequenza e decifrarlo: l'algoritmo, per quanto sia fatto bene, non darà mai una sequenza casuale in output, ma una sequenza che può al massimo sembrare casuale(in realtà non lo è).

Adesso implementiamo il nostro primo algoritmo di cifratura.

Di solito è compito di un crittografo l'implementazione perché si rischia un **porcodio finisci la frase o ti mangio il cranio**.

LEZIONE 19/10/18

La volta scorsa abbiamo parlato di 2 cifrari diversi:

One-time PAD: per cifrare un messaggio m si mette in XOR con k e si ottiene C . Si chiama One-time PAD ed è a cifratura perfetta secondo 3 caratteristiche:

- 1) k sia lunga quanto il testo in chiaro
- 2) k sia scelta in modo casuale
- 3) k non sia mai riutilizzata

Cosa succede se una di queste caratteristiche non viene rispettata?

Es:

Supponiamo di cifrare $m = 101100$ e supponiamo chiave lunga 3 $k=110$, viene ripetuta 110110.

Cosa succede? Quindi con una chiave più corta che deve essere ripetuta.

101100

110110

011010

L'attaccante conosce l'algoritmo utilizzato(XOR), e sa che la chiave è lunga 3 bit (perché per esempio è inglobata dal protocollo). Quindi è più corta di m , e sa che il protocollo dice di riutilizzare la chiave se è troppo piccola.

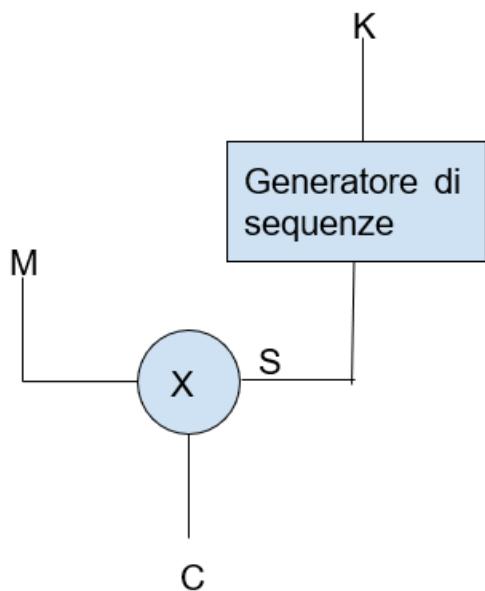
Cosa sa l'attaccante se vede questo? non sa quali sono i bit ma dato che sa la lunghezza della chiave, se vede che il risultato dello XOR si ripete può capire che ci sono delle correlazioni.

Dato che sa che la chiave è lunga 3 bit, e vede che nella prima posizione e nella quarta ho gli stessi bit come risultato vuol dire che in prima e in quarta posizione del messaggio in chiaro c'è lo stesso bit (non so però se è 0 o 1)

Se in aggiunta sapesse che quello che ha cifrato è il flusso (cioè che ogni byte viene cifrato byte a byte) di un certo protocollo sa anche quali sono le intestazioni, il formato e quindi mettendo in correlazione queste informazioni riesce a risalire alla chiave.

Ora questo esempio è importante perché se la chiave è generata a partire da un algoritmo (ad esempio un generatore pseudocasuale), sicuramente c'è una correlazione tra i bit poiché ci sono delle regole specifiche usate dietro alla costruzione di ogni algoritmo. L'attaccante può dunque estrarre delle informazioni importanti da esso. Se comincia a sapere che il primo bit è uguale al 4 oppure delle probabilità che aiutano a produrre una chiave l'attaccante sa più rispetto alla lunghezza della chiave quindi non è più sicuro.

Quando approssimiamo One-time PAD con un generatore di sequenze pseudocasuali, succede che questa sequenza non è scelta in modo totalmente casuale ma ci sono correlazioni fra i bit di questa sequenza che si rispecchiano sulla correlazione fra testo cifrato e quello in chiaro perché c'è di mezzo un algoritmo.



Questo cifrario è sicuro? Offre migliori garanzie quando il generatore PRNG è buono

Si chiama pseudo casuale poiché non è casuale. Deve essere “difficile” in termini computazionali distinguere la stringa prodotta dal generatore rispetto ad una casuale.

Se la stringa è **casuale** e si conoscono i bit della stringa è impossibile sapere i bit che vengono dopo poiché non c'è correlazione.

Se stringa è **pseudo casuale** è possibile scoprire quali sono i bit che seguiranno però più è difficile fare questa previsione quanto più il generatore è buono. Se la stringa è realmente casuale è impossibile mettere in relazione i bit. Essendo pseudo casuale le informazioni ci sono poiché la stringa è generata da un algoritmo. Più è difficile determinare relazioni della pseudo casualità più è difficile determinare relazione tra la chiave e la sequenza. Questi algoritmi si chiamano **cifrari a flusso** in contrapposizione a quelli a blocchi.

CIFRARI A FLUSSO

Mentre arrivano i dati si possono cifrare byte a byte indipendentemente dagli altri. Sono una classe molto importante, un cifrario a flusso importante fu **RC4** (l'abbiamo usato l'altro giorno per aprire la libreria che non funzionava). È semplicissimo ed è uno dei fondamentali aspetti della sicurezza.

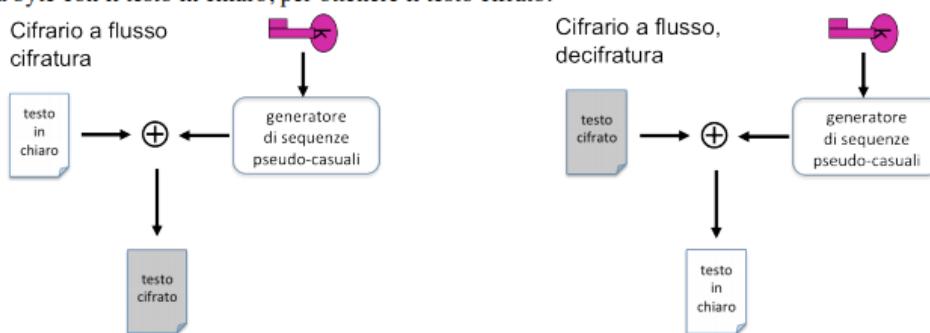
Questi algoritmi si chiamano anche cifrari a flusso per il fatto che **mentre arrivano i dati si possono cifrare bit a bit in teoria, byte a byte in pratica**. I cifrari a flusso sono molto importanti (tipo RC4).

RC4 (RIVEST CIPHER 4)

RC4 è semplicissimo e il suo nome deriva dal creatore Rivest :

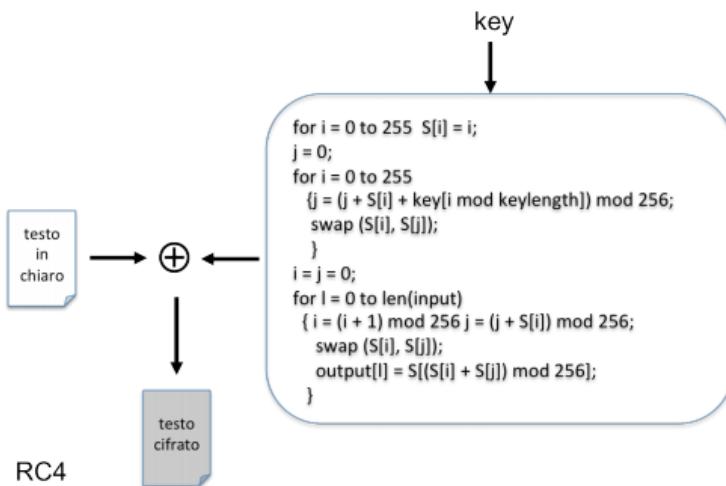
Cifrari a flusso e RC4

I cifrari a flusso hanno tutti la struttura mostrata in figura. Per cifrare, la chiave è data in input ad un generatore di sequenze pseudo-casuali. La sequenza prodotta dal generatore viene messa in XOR byte a byte con il testo in chiaro, per ottenere il testo cifrato.



Per decifrare, deve essere generata la stessa sequenza, dando in input la stessa chiave al generatore. La sequenza viene messa in XOR con il testo cifrato e si ottiene il testo in chiaro.

RC4 si distingue solo per lo specifico generatore che utilizza. La figura seguente mostra la cifratura.



Riempie array di byte tra 0 e 255, rimescola i numeri sulla base della chiave, dopodiché continua a dare in output dei pezzi e mentre lo fa continua a rimescolare.

Rivest Code 4 → inizialmente era stato utilizzato dalla azienda RSA senza diffondere il codice. Questo funzionava ed era apprezzato ma ad una certa qualcuno ha postato il codice su internet. Il fatto di rendere il codice pubblico non ha reso RC4 debole ed è stato ancora utilizzato in particolare per la sua semplicità. In particolare, genera sequenze che si ripetono dopo 10^{100} cifre in output.

Non si deve mai fare ipotesi che il codice rimanga segreto. Nel web nulla è realmente al sicuro: il protocollo WEP del Wi-Fi è stato bucato come un colabrodo: nel giro di 5 minuti collegandosi ad un punto di accesso WEP si riesce a risalire alla chiave. Questo poiché usava RC4 nel modo sbagliato. La chiave (penso intenda sequenza) utilizzata era solo un pezzo di quella generata da RC4 e quindi dopo un po' di tempo veniva riutilizzata, soprattutto con l'aumentare del traffico. Cosa succede ad un cifrario a flusso se viene utilizzata la stessa chiave due volte? Fa la stessa sequenza! Che impatto ha riguardo alla generazione della stessa sequenza?

Se uso la stessa chiave, si utilizza la stessa sequenza per due messaggi diversi. Da qui deriva la vulnerabilità: se ho 2 messaggi M1 e M2 che sono stati messi in XOR con la stessa sequenza, generano due messaggi cifrati C1 e C2 i quali, se messi in XOR, mi generano lo XOR del messaggio in chiaro:

$$C1 = M1 \text{ XOR } S$$

$$C2 = M2 \text{ XOR } S$$

$$C1 \text{ XOR } C2 = M1 \text{ XOR } M2$$

RC4 era un algoritmo di cui ci si fidava ma appunto era utilizzato male.

Dopo che WEP è stato dichiarato debole:

L'anno scorso sono nate le vulnerabilità di WPA/WPA2 poiché nonostante utilizzasse un cifrario a blocchi, questo veniva utilizzato come se fosse a flusso e sono emersi problemi. Quello che è successo è che utilizzando in modo corretto RC4 è troppo complicato: ci sono troppe trappole in cui si rischia di cadere e si rischia di avere un protocollo troppo debole. L'algoritmo è robusto ma il protocollo è debole.

La sicurezza di un sistema è data dal suo anello più debole come in una catena (se si rompe a metà si rompe anche se il resto è robustissimo).

Quindi in un sistema come WEP non importa che il cifrario fosse robustissimo, l'algoritmo era utilizzato male e quindi risultava debole.

ATTACCO A FORZA BRUTA

Un qualunque cifrario (sia a flusso che non) è soggetto SEMPRE con unica singola eccezione soggetto ad attacchi BRUTE FORCE:

- L'attaccante deve conoscere un testo cifrato C e questo si può dare per scontato perché un testo cifrato è proprio quello da non dover proteggere.
- Deve essere noto l'algoritmo, ovvero come viene usato l'algoritmo (protocollo) (qui è molto vaga la definizione, anche protocollo va bene).
- Deve avere un'idea della struttura del testo in chiaro.

Con queste informazioni, l'attaccante prova tutte le possibili chiavi che potrebbero essere state utilizzate, senza un criterio specifico (come quando non so che chiave è quella giusta e allora le provo una per una nella serratura finché non si apre). Succede che per ogni chiave K_i se un testo cifrato C viene decifrato con K_i darà un messaggio M_i .

L'attaccante conoscendo l'algoritmo di cifratura, può creare uno di decifratura , l'unica cosa è che deve trovare la chiave, quindi deve provarle tutte!

Se il testo fosse una sequenza di bit, qualunque chiave andrebbe bene poiché il testo in chiaro verrebbe fuori sempre come una sequenza di bit. Se ad esempio l'attaccante sa che il testo in chiaro è un file in pdf che ha scritto un'azienda per un'altra, allora l'attaccante distinguerà un certo numero di chiavi possibili che daranno in output la struttura di un pdf.

L'algoritmo per decifrare prova tutte le possibili chiavi e per ogni chiave K_i , ci sarà un testo in chiaro M_i e l'attaccante vedendo tutti questi testi in chiaro dovrà essere in grado di capire qual è il messaggio in chiaro giusto.

L'attaccante facendo questo lavoro potrebbe quindi distinguere un certo numero di chiavi che danno in output un testo con quella struttura, questo dipende anche da quanto è grande la struttura e dalla sua fortuna quindi l'attaccante per riconoscere il messaggio testerà tutte le chiavi e a quel punto che l'attaccante abbia ancora delle incertezze è molto bassa.

Si può sempre fare questo genere di attacco tranne su ONE-TIME PAD perché

C 10110 XOR

K 11100

M 01010

Un problema è che la chiave è completamente casuale ed è lunga quanto il testo cifrato.

Ciò significa che qualunque lettera da un'azienda all'altra, il testo in chiaro della lettera sarà di lunghezza corrispondente a quella del testo cifrato. Ruotando su tutte le chiavi binarie possibili su questa lunghezza vengono fuori tutte le stringhe binarie possibili lunghe quanto il testo in chiaro. Questo cosa significa? Lo si può fare ma non dà alcuna informazione: troviamo tutti i pdf possibili con un attacco a forza bruta quindi tutte quelle che hanno quella lunghezza. E' una perdita di tempo, poiché si viene a conoscere le informazioni che si sapevano già prima. Non ha proprio senso poiché non aggiunge alcuna informazione a quelle che l'attaccante già sa, non gli dà alcuna possibilità di capire se l'informazione ottenuta è giusta.

Un attacco forza bruta su ONE-TIME PAD non dà dunque alcuna informazione aggiuntiva.

Esempio:

L'attaccante conosce sempre la struttura del messaggio in chiaro.

Abbiamo un testo cifrato C e per ogni chiave che possiamo provare viene fuori un testo in chiaro diverso. Nel caso di OTP supponiamo che il cifrato sia lungo "l", quindi anche la lunghezza della chiave deve essere lunga "l". Quindi, dobbiamo provare 2^l chiavi, ottenendo quindi 2^l testi in chiaro.

$$|C| = l$$

$$|K| = n = l$$

Messaggi in chiaro possibili = 2^l in quanto possiamo provare 2^l chiavi

NB. in tutti i cfrari l'ipotesi è che si conosca sempre la lunghezza della chiave e anche l'algoritmo di cifratura(in quanto non è possibile tenerlo nascosto, esempio rc4).

Prendiamo ora un cfrario diverso, tipo RC4, con una chiave lunga n quante chiavi dobbiamo trovare per trovarle tutte? 2^n .

quanti messaggi ottengo? 2^n .

quanti sono i testi in chiaro possibili esistenti? 2^L . Dove L è la lunghezza del messaggio.

$$|K| = n \rightarrow 2^n \text{ possibili chiavi} // |K| \text{ lunghezza della chiave}$$

n è la lunghezza della chiave e potrebbe valere 128, 192, 256 bit. (256 al max perché siamo in rc4)

n = 128, 192, 256

Il testo in chiaro sarà lungo un k(1024 bit) come minimo. Da qui possiamo già confrontare con OTP.

Se L = K //lunghezza del messaggio è un certo k cioè 1024 bit

Il numero di chiavi possibili nei 2 casi (cioè rc4 e One-time PAD):

2^{256} nel caso della chiave più robusta con rc4

2^{1024} nel caso si utilizzasse OTP in quanto la lunghezza della chiave deve essere lunga quanto il messaggio

Numero chiavi One-Time Pad



La differenza quindi è molta, con RC4 un attaccante che usa forza bruta riesce a ridurre l'insieme delle chiavi possibili a massimo 2^{256} , mentre con OTP il numero rimane enorme (2^{1024})e quindi inutilizzabile se non spendendo moltissime risorse.

In sostanza OTP NON È ATTACCABILE DA BRUTE FORCE nel senso che non mi da alcuna informazione aggiuntiva, volendo lo posso fare ma non ha tanto senso.

COSTO ATTACCO FORZA BRUTA:

$O(2^n)$ se $|k| = n$

Significa che se usiamo un cifrario che non sia OTP la garanzia di riservatezza che offre è limitata al costo $O(2^n)$ quindi se l'attaccante ha sufficiente denaro e tempo da spendere ci arriverà sicuramente al risultato.

Quindi la sicurezza è RELATIVA perché è basata sulla teoria della complessità (anche la crittografia moderna è basata sulla teoria della complessità) rispetto alla situazione di OTP che è basata sulla teoria dell'informazione.

Infatti, in OTP il testo in chiaro non contiene sufficienti informazioni per risalire ad altro che la lunghezza del testo in chiaro... non c'è informazione!

In questo caso (rc4 e in generale tutti gli altri a parte OTP) l'informazione c'è, solo che ci vuole $O(2^n)$ per estrarla.

Quanto deve essere lunga la chiave di un cifrario normale non OTP? Dipende, deve essere abbastanza lungo affinché $O(2^n)$ sia proibitivo per l'attaccante nel contesto.

I cifrari e in generale la crittografia è basata sulla teoria della complessità ed è relativa.

Cosa significa relativa?

Significa che sostanzialmente con questa crittografia si può simulare la segretezza perfetta.

Esempio.

In guerra il generale deve comunicare "attacchiamo domani all'alba" con un messaggio cifrato. Intelligence sa quali sono le abilità di decifratura del nemico. Se sa che ci metterà una settimana il nemico quel messaggio è protetto come se fosse protetto da OTP (segretezza perfetta) poiché il messaggio ha valore solo fino a domattina. Relativo non solo ai tempi ma anche ai costi.

L'attaccante per fare un brute force dovrà investire del denaro per comprare le macchine con abbastanza forza computazionale.

Esempio 2 (per fare l'esempio sui costi)

Se il segreto vale 1000 e per l'attaccante costa 100k l'attaccante non lo farà poiché non ne vale la pena. Se uno in casa ha al massimo qualcosa che vale 5k non vale la pena mettere una porta blindata da 20k perché se l'attaccante deve spendere anche solo 5k più il rischio di essere beccato non lo farà.

Quindi in generale la sicurezza è relativa e in generale è basato tutto sulla teoria della complessità.

Quindi all'orale non bisogna dire è impossibile, si può dire è computazionalmente impossibile.

Quindi la lunghezza della lunghezza delle chiavi dipende da questo.

Oggi le chiavi vanno generalmente dai 128 ai 256 bit nei cifrari a chiave simmetrica.

CIFRARIO A FLUSSO E RC4 CONTINUA..

Finiamo il discorso su RC4 e i cifrari a flusso.

Un workaround è una soluzione temporanea che aggira il problema.

RC4: abbiamo parlato della sua semplicità realizzativa ma quello che vogliamo fare è lavorare con **OpenSSL** che è un software che useremo da linea di comando anche se in realtà offre una libreria crittografica utilizzata da moltissimi software.

E' la base crittografica di Apache, qualche anno fa è stato fatto un fork con una nuova libreria crittografica che si chiama LibreSSL perché nella precedente libreria si sono trovate molte vulnerabilità.

OpenSSL è una libreria open source, ha senso una libreria crittografica open source?

se la libreria è open source l'attaccante ha buon gioco di andare a guardare il codice e trovare vulnerabilità (questo non succede nei software proprietari).

Bisogna sempre tenere a mente che tenere nascosto qualcosa funzionava solo fino a un certo punto, invece avere una libreria open source significa che sì, l'attaccante può vedere il codice ma anche tutti gli altri e quindi può essere migliorato .

Spesso una libreria proprietaria è anche guardata con un certo sospetto perché non si può capire cosa sta realmente "dietro".

OpenSSL non lo useremo come libreria ma da linea di comando per vedere alcune cose sui sistemi crittografici.

OpenSSL uno lo può aprire da linea di comando e poi inviare i comandi direttamente da "openssl" però è sconsigliato perché il problema è che per modificare il comando bisogna riscriverlo tutto mentre se usiamo la shell basta fare la freccia in su per avere il vecchio comando appena digitato.

`openssl rc4 -p -in file chiaro -K chiave_hex -out file_cifrato`

openssl -> libreria da usare

RC4 -> tipo di cripitura

-p -> fammi vedere i parametri da usare

-in -> file input in chiaro

-K -> chiave in esadecimale

-out -> file di output il file cripato con estensione .bin poiché non è un file di testo leggibile

L'estensione .cif è un'estensione campata lì, serve solo a livello didattico per capire che un file è cripato.

A questo punto qual'è la reazione? Mi dice qual'è il **SALE** ma non lo usa.

Per vedere il file .bin aprire Applications/Accessories/Okteta

Come cripa RC4? Data la chiave K il generatore genera una sequenza pseudocasuale e il risultato è uno XOR byte a byte del testo in chiaro con la sequenza pseudocasuale.

A questo punto rc4 mi chiede la password di cifratura e la inserisco. Me la chiede due volte e cifra con quella chiave (per essere sicuri di cifrare con quella chiave). La key non è la stessa di quella che avete messo, come mai? a partire dalla password che abbiamo inserito openSSL ha calcolato una chiave con un algoritmo.

Come l'ha calcolata? Utilizza delle funzioni hash (di cui parleremo più avanti), è un algoritmo che prende in input la password e la trasforma in una chiave. Se facciamo 2 volte la stessa operazione usando la stessa password verrà fuori la stessa sequenza.

Non è particolarmente utile dato che si tratta di un algoritmo deterministico quindi non rende più forte la chiave in alcun modo. L'attaccante può fare un brute force e generare queste chiavi.

- L'attaccante farà un **attacco di tipo dizionario**:

Il dizionario è un file dove ci sono le password usate più frequentemente. L'attaccante prima di fare attacco su una chiave della lunghezza di KEY con tutti i possibili caratteri esadecimali ne farà uno di tipo dizionario con determinate stringhe più utilizzate.

Anche in questo caso se uno cifra 2 volte mettendo la stessa Password il risultato sarà lo stesso.

- Un altro modo di generare le password in modo sicuro è l'uso del sale (salt).

Un altro modo di cifrare e decifrare le password:

openssl rc4 -p -in prova -out prova.bin

Cosa succede se togliamo -nosalt?

La cosa interessante da fare se lo facciamo una seconda volta, con la stessa password quello che si scopre è che il sale che viene utilizzato è diverso e anche la chiave è diversa di conseguenza. **Il sale è una sequenza pseudocasuale, e serve per generare la chiave di cifratura a partire dalla password.**

Quando uno decifra deve usare la stessa chiave (lo stesso sale per generare la stessa chiave) . Come fa openSSL a sapere quale sale usare?

Non può mantenerlo in memoria ovviamente, ma viene scritto sul file stesso di output. Se apriamo il file lo possiamo vedere.

Quando lungo dovrebbe essere un file cifrato con RC4? Come il file in input. Qui è più lungo per colpa del sale, quindi parte dal sale che trova sul file, usa la passphrase e genera la password (data dalla chiave e dall'aggiunta del salt). L'attaccante può partire dalla chiave e usare lo stesso algoritmo per generare il sale e ritrovare il file.

Cambia qualcosa nella difficoltà dell'attacco per l'attaccante? Ovvero è aumentata la casualità della chiave usata per cifrare?

No perché l'attaccante può tranquillamente partendo dalla chiave usare lo stesso algoritmo che usa openssl per generare la chiave usando il sale che legge dal file quindi il tutto è ancora noto.

Quindi a cosa serve il sale? **Il sale serve per scoraggiare gli attacchi Brute Force su grossi volumi di dati:** se ad esempio l'attaccante viene a conoscenza del file delle password di sistema (quindi ha in mano 3000 password cifrate in qualche modo), se sono cifrate senza sale, l'attaccante prende l'elenco di password più comuni, le cifra 1 a 1 e va a confrontarli con quelli cifrati che ci sono in elenco, se qualcuno di quegli utenti ha usato una password che sta provando l'attaccante, egli lo troverà in questo modo in quanto risulteranno uguali.

ESEMPIO:

Dizionario: P1 P2 P3 Pn - Elenco: C1 C2 C3... Cn

FB: E_k1(p1) = C (prova k1, cifra p1 con k1 e trova un certo C)

E=Encryption

K1=una chiave a caso che serve per criptare la password in chiaro p1 presente nel dizionario

p1=password in chiaro del dizionario

Questo C lo confronto con tutte le C presenti nell'elenco e se l'utente ha usato quella password comune, l'attaccante potrà entrare nel sistema grazie a quella password.

In sostanza una volta che ho trovato la k giusta per criptare la mia password più frequente, il testo cifrato C che genero, lo confronto con tutti i messaggi cifrati presenti nel file, se sono uguali allora ho scoperto la password di un utente e loggarmi sul sistema

Se invece c'è del sale(salt) per ogni password crittografata degli utenti:

Dizionario: p1 p2 p3 ...pn

Elenco: C1 C2 C3.. Cn

FB: Ek1(p1,s1) = C

Come prima ma **con l'aggiunta del sale, l'attaccante è forzato a fare attacchi distinti per ogni password presente nel file dizionario poiché i sali saranno tutti distinti** e non può sfruttare il vantaggio del volume ridotto che avevamo nel caso in cui non ci fosse il salt e il lavoro è più complicato.

Quando un utente invia una password al sistema, questo deve verificare QUELLA password: non è che cifra tutto, non è nemmeno in grado di decifrare la password. Se il sistema è fatto bene, la password viene cifrata e l'operazione di cifratura confronta il risultato con quello che c'è nel file di password.

Qualunque sistema che mi ritorna indietro la password in chiaro è un sistema che non funziona a dovere. Questa è la ragione per cui si raccomanda di non usare sempre la stessa password perché la volta che si incappa in un sistema debole la password è fritta.

Un sistema che gestisce le password in maniera corretta non deve mantenere le password in chiaro, ma le mantiene solo cifrate ed è solo in grado di verificare che la password che sottoponiamo noi è corretta. Questo perché se l'attaccante recupera il file delle password e questo è in chiaro in qualunque momento ha tutte le password. Invece se l'attaccante recupera il file delle password e sono tutte cifrate l'attaccante deve ancora **craccarle**.

La sicurezza di un sistema è la sicurezza del suo componente più debole. Se uno prende in considerazione tutti i servizi a cui accede il sistema la sicurezza è data da quello più debole fra tutti i servizi.

Un'altra ragione per usare il sale è che se anche la password che viene usata per generare la chiave è robusta, l'utente che utilizza più volte la stessa passphrase, se non fosse usato il sale, utilizzerebbe più volte la stessa chiave nel cifrario ,un cifrario a flusso utilizzerebbe quindi più volte la stessa sequenza e questo induce ad attacchi che proveremo a fare adesso.

Ogni volta che l'utente riutilizza la stessa chiave (ricordo che è un cifrario a flusso) se la chiave è la stessa, la sequenza è la stessa, se la sequenza è la stessa, lo XOR dei testi cifrati è uguale allo XOR dei testi in chiaro quindi cosa succede?

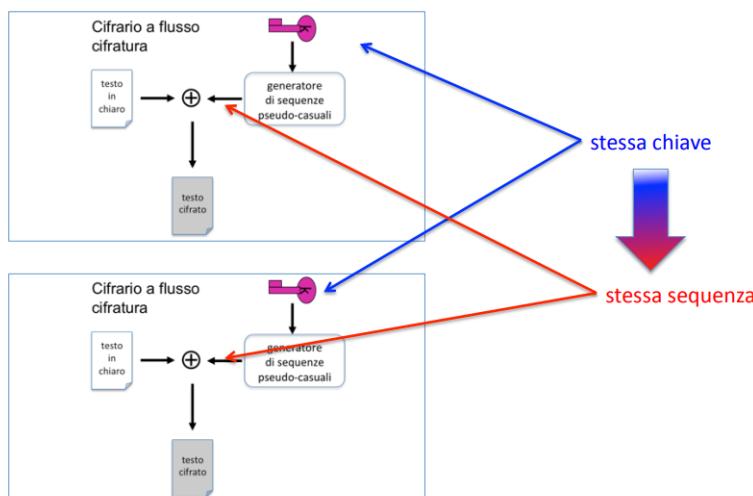
[Descrizione] La chiave genera la stessa sequenza e quindi la cifratura viene fatta con la stessa sequenza: questo significa che se in un caso l'attaccante riesce a conoscere sia testo in chiaro che

testo cifrato, lui ha la sequenza che non è la chiave, però con la stessa sequenza può riuscire a decifrare un testo cifrato arbitrario che sia cifrato con la stessa sequenza.

XOR tra i messaggi cifrati è uguale allo XOR dei testi in chiaro. Stessa chiave genera la stessa sequenza cosa significa? Se per caso attaccante riesce a conoscere sia testo in chiaro e cifrato riesce a risalire alla sequenza (facendo uno XOR) con la quale riesce a capire come decifrare n file cifrati con la stessa chiave (in quanto la sequenza senza salt rimane la stessa).

Se fosse stato usato un salt nel testo cifrato non si può fare questo attacco con lo XOR Perché le sequenze tra i vari testi/file cifrati sono diverse in quanto la seconda volta che viene riutilizzato il cifrario la pass è la stessa ma il sale è diverso. La chiave genera un s' diversa da s e l'attaccante non può fare il giochino che abbiamo fatto.

cifrari a flusso



Utente compila un modulo con dati generali e un secondo modulo con dati privati. Entrambi vengono cifrati con la stessa chiave. Allora l'attaccante ha a disposizione i due testi cifrati e il modulo generale perché il modulo vuoto è noto. Sa inoltre che il modulo è compilato interamente in maiuscolo senza spazi né punti.

Scenario



L'attaccante ha a disposizione il modulo 1 e conosce i dati della vittima:
Mario Rossi, abita in via Garibaldi, 13

ESEMPIO CIFRARIO A FLUSSO.

Facciamo prima uno XOR tra modulo noto e modulo noto cifrato per trovare la sequenza, dato che non è stato utilizzato un salt poi riutilizzo la stessa sequenza mettendola in XOR con il testo cifrato personale per decifrarlo , in quanto dato che non viene utilizzato il salt se do in pasto la stessa chiave all'algoritmo mi genererà la stessa sequenza.

CIFRARI A BLOCCHI (A CHIAVE SIMMETRICA)

I cifrari a blocchi sono algoritmi che cifrano blocchi di testo più grandi di un byte.

Prendono il testo in chiaro e lo dividono in blocchi di lunghezza che dipende dall'algoritmo specifico, noi ne citiamo 2 che sono i più famosi:

DES: Blocchi 64 Bit

AES: Blocchi 128 Bit

Nei cifrari a blocchi, gli algoritmi di cifratura e di decifratura sono in grado di gestire un certo blocco di dati. Vedremo che ci sono diverse scelte per combinare le varie lavorazioni ma **in linea di massima c'è un blocco "b" che viene cifrato con un algoritmo di cifratura tramite una chiave K e viene prodotto un blocco di testo cifrato della stessa lunghezza del blocco di partenza.**

Per decifrare si usa come input il testo cifrato, **si usa l'algoritmo di decifratura (che può essere diverso da quello per cifrare) con chiave K e si ottiene il testo in chiaro.**

La lunghezza del blocco in input 'b' deve essere specificata dall'algoritmo questo porta già ad un problema: **se il testo in chiaro avesse una lunghezza che non è un multiplo della lunghezza dei blocchi, questo dovrà essere completato tramite il Padding** che è **un completamento che si mette in fondo all'ultimo blocco** se questo non ha la lunghezza giusta.

L'algoritmo per il padding che viene usato più comunemente aggiunge comunque un blocco di padding anche se non serve: se il testo è di lunghezza esattamente multipla della lunghezza dei blocchi, allora **viene aggiunto un padding lungo quanto un blocco.**

Per la gestione delle chiavi anche in questo caso **la chiave può essere data da una sequenza di bit con o senza sale.**

Dietro i cifrari a flusso abbiamo visto che il generatore di sequenze pseudocasuali deve generare sequenze quanto più simili a sequenze realmente casuali. Per quanto riguarda **i cifrari a blocchi, essi sono basati sui principi di SHANNON** (1945)(c'è sul madi non obbligatorio)

PRINCIPI DI SHANNON

I principi sono 2 :

1. Principio di confusione
2. Principio di diffusione

Principio di confusione

Il principio di **CONFUSIONE** dice che "La relazione tra il testo cifrato e la chiave deve essere complessa." Ciò significa che **dato un testo cifrato C, l'attaccante non deve poter facilmente estrarre informazioni sulla chiave K.**

Il testo cifrato è generato a partire dal testo in chiaro e dalla chiave quindi l'algoritmo sfrutta la chiave. Questo principio dice: "Attenzione, il cifrario è fatto bene se l'attaccante guardando solo il testo cifrato non riesce facilmente ad avere informazioni sulla chiave".

Noi sappiamo benissimo che tutti i cifrari che si utilizzano normalmente (escluso OTP) tramite l'attacco BF è possibile notare almeno un po' di struttura del testo in chiaro. Quindi in realtà dal testo cifrato si può sempre estrarre tutta la chiave avendo un po' di informazioni su di esso, però questo principio dice che deve essere DIFFICILE riuscirci.

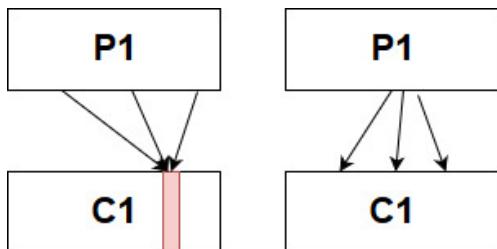
Principio di diffusione

Il principio di **DIFFUSIONE** dice che :

Un bit del testo cifrato C deve dipendere da più bit del testo in chiaro P.

Un bit di P influenza su più di un bit di C.

Dire che un bit di C deve dipendere da uno più bit di P significa che ogni bit presente in P deve avere una relazione con un bit di C.



Se ci sono K buchi nel muro e K+1 piccioni, allora in almeno un buco ci saranno due piccioni..

Sostanzialmente visto che il testo in chiaro è uguale al testo cifrato di lunghezza, se più bit del testo in chiaro influiscono su un unico bit del testo cifrato dovrà essere vero anche il contrario: che un bit del testo in chiaro influisce su più bit del testo cifrato.

Questo significa che se cambia anche solo un bit al testo in chiaro il testo cifrato cambia molto e quindi l'attaccante non ha buon gioco a fare prove di approssimazione del testo in chiaro.

Questo funziona con RC4?

No, non funziona per principio di diffusione in quanto nei cifrari a flusso (non solo RC4) viene cifrato il messaggio facendo uno XOR byte per byte con la chiave. Come si ottiene la diffusione?

I CIFRARI PIÙ FAMOSI : AES, DES

Vediamo com'è realizzato AES. DES e AES sono dei cifrari famosi.

https://www.dir.uniupo.it/pluginfile.php/408119/mod_resource/content/3/13-AES-2013.pdf

AES e DES sono due cifrari famosi, DES è stato per 30 anni lo standard americano, AES è quello che l'ha rimpiazzato.

AES

Come funziona questo cifrario?

Funzionamento di AES:

1. **XOR byte to byte** tra matrice del testo in chiaro con quella della chiave di round 0 (ovvero la chiave inserita dall'utente). Genero chiavi di round successive.
2. **Sostituzione byte** xy modificato con quello della S-Box in posizione riga x, colonna y.
3. **Mescolamento**: a. scorrimento delle righe, b. mescolamento righe con colonne.
4. **XOR con chiave di round**.

Da qui riascolto fino alla fine

L'idea è che si vuole cifrare una stringa di testo in chiaro con una certa chiave a 128 bit: **AES lavora con** uno stato che è **una matrice di byte 4x4** perché AES usa blocchi di 128 bit quindi sono 16 byte e li vede in questa matrice.[Questi non sono i blocchi di testo in chiaro, ma la chiave] **Tutti i cifrari a blocchi**, compreso anche DES **sono organizzati in round**: ogni round **fa delle operazioni sul testo in chiaro modificandolo un po'**, e poi **viene passato in input al round successivo**. Generalmente **a ogni round cambia solo la chiave di round** che è generata a partire dalla chiave di cifratura data in input.

Il primo round fa uno XOR byte a byte con la chiave di round 0 (che coincide con la chiave) e poi vengono generate altre chiavi di round AES (AES ha un totale di 10 round)

primo round (1)

Round 1			
Add Round Key			
7c	3e	95	eb
b3	59	cf	ba
df	0f	a6	f9
61	a0	1b	60
\oplus		a2	4d
0c	2c	27	40
07	a5	a1	94
ab	df	34	16
=			
de	3	a5	3b
bf	75	e8	fa
d8	aa	07	6d
ca	7f	2f	76
Substitute Bytes			
de	73	a5	3b
bf	75	e8	fa
d8	aa	07	6d
ca	7f	2f	76
\rightarrow		1d	8f
08	9d	9b	2d
61	ac	c5	3c
74	d2	15	38
0111 1100 XOR 1010 0010 = ----- 1101 1110			

L. Eaidi - Sicurezza

Il primo round cosa fa? Fa uno XOR byte a byte della chiave di round (che comincia con 7c) **il secondo è il testo in chiaro**. La prima e l'ultima operazione devono essere fatte con la chiave perché altrimenti l'attaccante potrebbe simularle.

La **seconda fase è la sostituzione di byte**: esiste una tabella che si chiama s-box che è predefinita

S-box di AES (da FIPS 197)

S-box di AES (da FIPS 197)

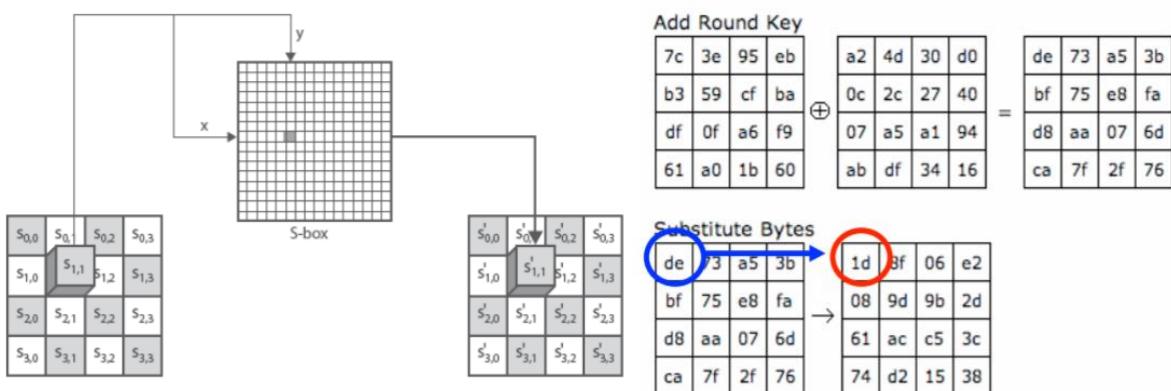
	Y															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

	Y															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

e viene utilizzata così: questo è il testo parzialmente elaborato e viene preso un byte (cioè una casella) da questo testo e questo byte è fatto di 2 porzioni di 4 bit: se x e y sono i caratteri esadecimales che compongono S1 viene pescato nella tabella l'elemento in posizione x,y il quale viene preso e sostituito dentro la nuova matrice. Questa è un'operazione di CONFUSIONE in modo da non dare corrispondenza tra chiave e testo cifrato.

Byte Substitution

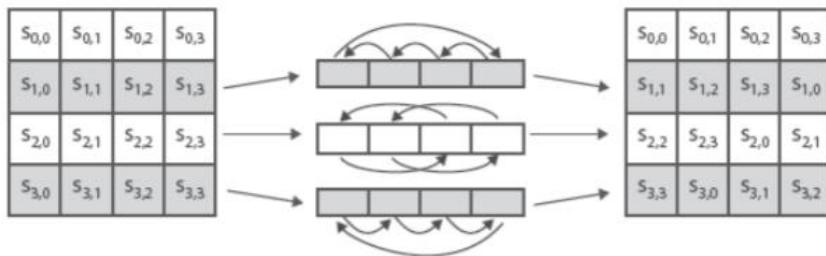
primo round (1)



Fatta la sostituzione sul testo in chiaro cosa succede? il primo byte 'de' viene pescato e sostituito con 1d nella tabella S-box. Questo implica che il testo in chiaro cambia parecchio. Dopodiché ci sono le prime operazioni di **mescolamento** che sono **importanti per la diffusione**

La prima operazione di mescolamento è lo scorrimento di righe

Scorrimento di righe



Lo scorrimento delle righe: la prima riga non viene shiftata, la seconda viene shiftata di 1, la terza di 2 e la quarta di 3. Questo implica che i bit che erano in una posizione vengono spostati a sx e influenzano un'altra parte di testo cifrato in base alle operazioni successive.

Quindi i bit che erano da una parte vengono shiftati (DIFFUSIONE).

Oltre allo scorrimento di righe c'è un'operazione di **mescolamento su colonne** che è un **prodotto righe per colonne di quello che abbiamo elaborato fino ad ora con una matrice prefissata**.

Per generare ogni elemento come si fa? Per generare il prodotto si fa prodotto di tutta la riga con tutta la colonna dell'altra matrice e fa sì che il bit di questo byte dipendono da tutta la riga e tutta la colonna. Queste 3 sono tutte operazioni che non coinvolgono la chiave, almeno finora.

Shift Rows

1d	8f	06	e2
08	9d	9b	2d
61	ac	c5	3c
74	d2	15	38



1d	8f	06	e2
9d	9b	2d	08
c5	3c	61	ac
38	74	d2	15

Mix Columns

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02



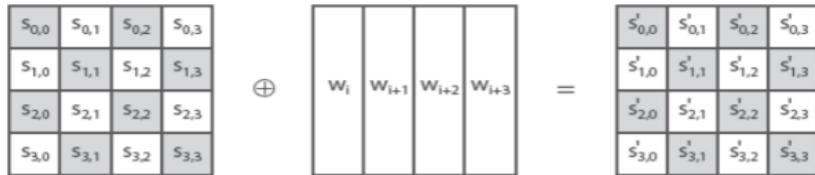
1d	8f	06	e2
9d	9b	2d	08
c5	3c	61	ac
38	74	d2	15



7b	fb	c8	7e
50	92	2d	08
59	f0	84	96
0f	c5	f9	b3

Questa operazione quindi fa sì che i bit di questo byte per esempio 7b dipendono da tutta la riga e da tutta la colonna.

XOR con chiave di round

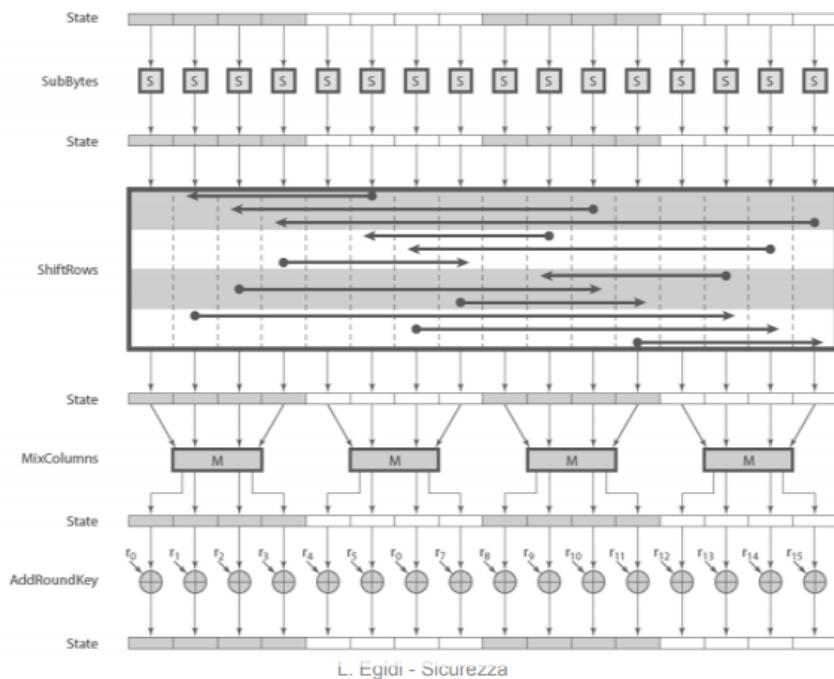


Successivamente viene eseguito uno XOR con la chiave di round per chiudere il tutto.

Per ottenere la diffusione vengono fatte varie permutazioni di bit(spostamenti) e vengono fatte funzioni di XOR con la chiave. La chiave di round viene generata con operazioni varie.... Ruotata S-box + XOR con una costante.

Qui

Round di AES



si vede un riassunto delle operazioni che vengono effettuate.

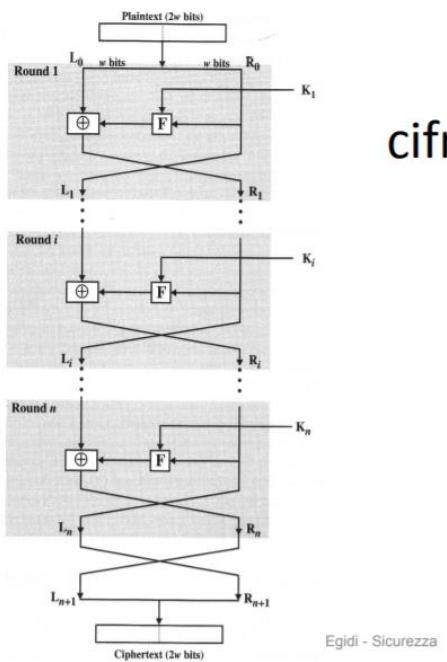
Siccome ogni volta c'è lo shift di righe oltre ad avere il fatto che ogni elemento dipende dai contenuti di una colonna e di una riga, al round successivo, siccome è stato fatto lo shifting di righe si sono anche spostati gli elementi da una colonna all'altra e quindi al prossimo prodotto righe per colonne l'informazione viene ancora più mescolata e vengono aggiunte dipendenze.

Per ottenere **la diffusione** vengono fatte delle permutazioni di bit e poi viene fatta l'operazione di XOR con la chiave(di round).

DES

[NON E' IN PROGRAMMA DICE NELL'AUDIO] DES è un sistema di cifratura di Feistel

sistemi di cifratura di Feistel



L'idea è che il testo in chiaro viene diviso in due parti una parte non viene toccata e viene passata al secondo round, l'altra viene messa in XOR con un qualcosa dove questa "F" prende la metà del testo in chiaro, ne fa una sostituzione, c'è dentro uno XOR con la chiave e poi viene fatto uno XOR con questa metà.

La differenza fondamentale con AES è che non è un sistema di Feistel poiché ad ogni round altera tutto il testo in chiaro o tutto il pre-elaborato mentre quelli di Feistel lavorano sempre e solo su metà. Il vantaggio di questo meccanismo rispetto a quello di AES è che la decifratura è esattamente identica all'inverso ma cambia l'ordine di come vengono usate le chiavi di round.

Mentre con AES è necessario un procedimento di decifratura diverso, con costanti diverse, deve essere implementato anche l'algoritmo di decifratura.

Dover implementare decifratura e cifratura separatamente genera tanti errori quindi è preferibile il sistema DES.

Il passaggio da DES a AES è stato effettuato principalmente per la lunghezza della chiave. DES non solo è stato utilizzato per 30 anni ma era lo standard crittografico degli stati uniti. Non si sono trovati attacchi significativi se non brute force per ottenere qualcosa, perciò essendo la complessità del brute force $O(2^n)$ con n =lunghezza chiave, allora più la chiave è lunga, più difficile è trovare la chiave usata per cifrare.

Se ci fosse un attacco più facile (per esempio un attacco al protocollo) è chiaro che **la lunghezza della chiave non è detto che sia essenziale** per garantire la protezione come per brute force.

Non è dimostrato in teoria, semplicemente si osserva che questa è la situazione quindi si scelgono chiavi lunghe per proteggersi da attacco a forza bruta. **AES è il cifrario che viene utilizzato maggiormente.**

"In AES quando viene utilizzata la chiave data da noi in input? e la chiave di round?"

[premette che non ce lo chiede all'esame]

Tutti i cifrari che funzionano a round (sono cifrari vecchi) generano la chiave di round a partire dalla chiave che l'utente inserisce.

Per quanto riguarda AES, il primo XOR si fa proprio con il testo in chiaro e con la chiave che l'utente mette dentro e le altre sono generate successivamente a partire dalle precedenti.

Vengono generati delle chiavi di round quindi quello che succede è che c'è questo XOR, poi ci sono le altre operazioni del round che sono sostituzione, shift e moltiplicazione e a quel punto viene fatto lo XOR con la seconda chiave di round con la variabile che è stata generata da questo.

Dopo 10 round AES ottiene una diffusione totale, ogni bit dipende da ogni altro bit.

I cifrari a blocchi si contrappongono da quelli a flusso perché sono in grado di cifrare blocchi e quindi i dati finiscono per essere un po' espansi.

La lunghezza dei testi cifrati con **cifrari a flusso** invece è la stessa del testo in chiaro.

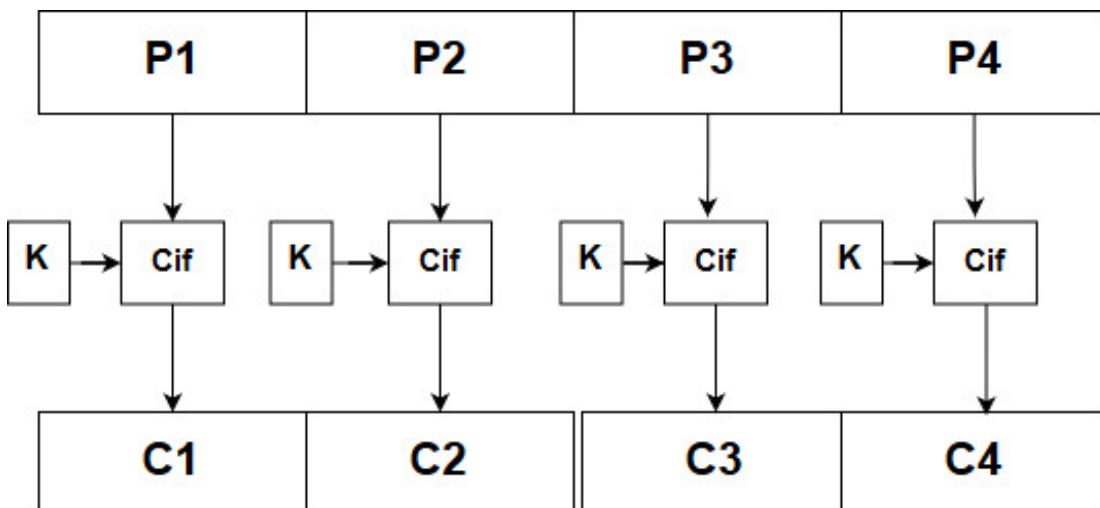
Abbiamo visto che il testo in chiaro viene diviso in blocchi della stessa lunghezza e se manca qualcosa si aggiunge il **padding**.

Il blocco viene cifrato con la chiave di cifratura e si ottiene il testo cifrato. L'idea più semplice di tutte (non è l'unico modo) è cifrare tutti i blocchi nello stesso modo e così otterrò blocchi di testo cifrato, il quale sarà dato dalla concatenazione dei blocchi cifrati.

Il testo cifrato avrà la stessa lunghezza del blocco non cifrato (multiplo della dimensione del blocco) e la decifratura avverrà tramite la chiave K poiché è simmetrica, e si decifrerà blocco per blocco.

Questo è il modo più semplice e intuitivo di utilizzare il cifrario, ovvero una modalità di cifratura che prende il nome di **ECB** (Electronic Code Book mode) **che cifra un blocco per volta**.

MODALITÀ DI CIFRATURA DEI MESSAGGI LUNGHI ECB



Si dice che uno ha cifrato con DES-ECB oppure AES-ECB significa che nel primo caso questo cifrario è DES e la modalità è ECB.

VANTAGGI DI ECB:

1. Semplicità di funzionamento
2. Parallelizzazione della cifratura per singolo blocco
3. In caso di errore su testo cifrato in fase di trasmissione oppure di alterazione del testo cifrato da parte di un attaccante, il testo in chiaro di quel blocco verrà alterato ma gli altri blocchi saranno consistenti
4. La decifratura di ogni blocco è indipendente

Svantaggi: Blocchi uguali vengono cifrati nello stesso modo, con l'analisi di frequenza è facile quindi capirne la struttura dei messaggi. (in seguito spiegato).

ESERCITIAMOCI CON openssl

Con openssl esperimenti di cifratura

```
openssl des-ecb -p -in clear.txt -out cipher.txt -nosalt
```

Se si lascia il sale

```
openssl des-ecb -p -in clear.txt -out cipher.txt
```

Un blocco **DES**(la chiave DES) è grande **64 bit** anche se **in realtà opera su 56 bit in quanto 8 bit sono utilizzati per il checksum. Infatti per ogni byte l'ultimo bit è per il checksum**, quindi il testo cifrato sarà sicuramente un multiplo di 8 byte. Il testo in chiaro sarà sicuramente più corto in quanto viene aggiunto il padding nel testo cifrato, in molti casi potrebbe non essere un multiplo di 8 byte e in questo caso nel testo cifrato si vedrà il completamento.

Deve essere possibile riconoscere i byte aggiunti dal padding rispetto a quelli del testo in chiaro perché un byte potrebbe a priori essere un byte del testo in chiaro.

"In DES si aggiunge un blocco da 8 byte se il testo è un multiplo di 8 byte, se no se non è un multiplo viene completato con l'aggiunta di tot byte per portarlo a un multiplo di 8

Mentre AES essendo a 128 bit poiché la granularità del blocco è di 16 byte, **se il file è un multiplo di 16 byte ne aggiungerà 16 di padding.**

A causa della diffusione, se noi modifichiamo 1 bit del testo cifrato, cambia tutto il testo di quello in chiaro quando andiamo a decifrarlo.

Se un attaccante ha un testo cifrato con RC4 e ha idea del formato, può cambiare un bit che gli interessa (sa per esempio che il 10° bit è una certa cosa) (in qualsiasi cifrario a flusso) lo può cambiare e potrebbe non vedersi niente in quel testo in chiaro viene cambiato solo quel bit e poi viene decifrato nel testo in chiaro(quindi può essere alterato in modo che l'alterazione non sia evidente), mentre in questo modo grazie al principio di diffusione se viene modificato anche un solo bit tutto il blocco viene oscurato in quanto viene fuori un macello in quel blocco e cambia totalmente.

DES cifra con una chiave a 64 bit.

AES invece può utilizzare una chiave da 128, 192 o 256 bit

ES1:

Nel primo esercizio abbiamo provato a cifrare lo stesso file con DES e AES successivamente con Okteta abbiamo aperto il file e abbiamo notato che:

con DES il risultato è questo:

```
cipher.txt ✘  
0000:0000 BC 55 0D 5C 1C B8 B8 2B BC 55 0D 5C 1C B8 B8 2B | ½U. \ . , + ½U. \ . , +  
0000:0010 BC 55 0D 5C 1C B8 B8 2B BC 55 0D 5C 1C B8 B8 2B | ½U. \ . , + ½U. \ . , +  
0000:0020 BC 55 0D 5C 1C B8 B8 2B BC 55 0D 5C 1C B8 B8 2B | ½U. \ . , + ½U. \ . , +  
0000:0030 02 13 DD AD F8 DE C0 6F | ..Y.øpAo
```

Con AES il risultato è questo

```
cipherAES.txt ✘  
0000:0000 45 B4 65 09 FD A8 26 B1 20 9A 8A C2 D7 2A C2 41 | E 'e. ý" &± ..Äx*ÄA  
0000:0010 45 B4 65 09 FD A8 26 B1 20 9A 8A C2 D7 2A C2 41 | E 'e. ý" &± ..Äx*ÄA  
0000:0020 45 B4 65 09 FD A8 26 B1 20 9A 8A C2 D7 2A C2 41 | E 'e. ý" &± ..Äx*ÄA  
0000:0030 61 5A 3D 05 39 E0 06 3F B0 41 5F 77 E3 9D 63 77 | aZ=. 9à. ?°A_wä.cw
```

Quindi con AES il testo cifrato è più lungo in quanto usando una chiave a 128 bit va per multipli di 16 byte e non di 8 byte come DES.

ES2: non mi fa vedere nulla di nuovo sostanzialmente lo Egidi lo balza

Es 3:Modifichiamo un solo bit nel testo cifrato(perché magari l'attaccante vuole manipolare un dato)

Aprire con Okteta

Non modificare il PADDING(ultimo blocco)

Se cambiamo un bit nel testo cifrato per diffusione allora quando decifro avrò il blocco corrispondente al testo cifrato che ho modificato completamente diverso in quanto un bit del testo cifrato dipendono da più bit del testo in chiaro

Se modifico anche solo un bit nel testo in chiaro noteremo che la cifratura è completamente diversa: questo è l'impatto della diffusione, che porta l'attaccante a desistere dall'effettuare un attacco per modificare un messaggio: infatti l'attaccante ha come scopo il modificare il messaggio senza farsi scoprire, e per fare ciò se c'è diffusione la sua modifica verrà propagata sugli altri blocchi, rendendo visibile la modifica.

Con AES usando blocchi di 16 e non di 8 byte, **la parte modificata sarà più grande** in quanto adopera blocchi più grandi.

Es4 (modifica della chiave):

```
openssl des-ecb -nosalt -p -in clear.txt -out cipher.txt -K chiave_in_esadecimale
```

(lunga 64 bit o anche più corta, in quel caso verrà completata con tutti 0) proviamo a cifrare i file, poi cambiare un bit della chiave e vedere i risultati.

1. **CONFUSIONE: Cambiando un bit della chiave, cambia completamente il testo cifrato con essa:** se proviamo a cifrare lo stesso testo in chiaro con una chiave che varia di un solo bit, notiamo che i 2 testi cifrati sono completamente diversi uno dall'altro. Questo effetto si chiama CONFUSIONE(il testo cifrato deve dipendere nel modo più complicato possibile dalla chiave).

- 2. Se modiflico solo il bit finale di un byte della chiave in DES** (deve essere cambiato l'ultimo bit della chiave) **non cambia niente perché l'ultimo bit di ogni byte è utilizzato per una sorta di checksum e quindi non viene utilizzato** (il bit del checksum era una cosa che era stata messa inizialmente e mai usata. Ciò vuol dire che DES utilizza effettivamente 56 bit) **mentre su AES questa cosa non succede.**

Cifrando il file clear.txt di testo: "elefanteelefanteelefanteelefanteelefante" con:

```
openssl des-ecb -nosalt -p -in clear.txt -out cipher.txt
```

con chiave key=6613DDD54D6DB890

cipher.txt	
0000:0000	36 69 3E E6 5F 9B BA 01
0000:0010	36 69 3E E6 5F 9B BA 01
0000:0020	36 69 3E E6 5F 9B BA 01
0000:0030	D7 95 76 C6 E1 F1 C1 1C
	6i>æ_º.6i>æ_º. 6i>æ_º.6i>æ_º. 6i>æ_º.6i>æ_º. x.vÆáñÁ.

```
openssl des-ECB -nosalt -p -in clear.txt -out cipher_m.txt -K 6613DDD64D6DB890
```

cipher_m.txt	
0000:0000	8B A3 A6 06 9C E3 63 06 8B A3 A6 06 9C E3 63 06
0000:0010	8B A3 A6 06 9C E3 63 06 8B A3 A6 06 9C E3 63 06
0000:0020	8B A3 A6 06 9C E3 63 06 8B A3 A6 06 9C E3 63 06
0000:0030	C5 AB 9E 1D F8 BD 8D 66
	.£ ..äc..£ ..äc. .£ ..äc..£ ..äc. .£ ..äc..£ ..äc. Ä«..øl2.f

Qui sono diversi perché non ho modificato l'ultimo bit della chiave ma uno in mezzo, se fosse cambiato l'ultimo bit sarebbero rimasti uguali.

Cosa succederebbe in un cifrario a flusso?

In un qualsiasi cifrario a flusso, se facessimo questi esperimenti vedremmo che se il generatore è abbastanza buono sembreranno sequenze casuali, se cambiamo un bit del testo cifrato, si cambia un bit del testo in chiaro. Viceversa, se cambiamo un bit del testo in chiaro e cifriamo cambia comunque un solo bit del testo cifrato.

Se cambiamo un bit della chiave in un cifrario a flusso?

Cambia tutto se il generatore pseudocasuale è buono.

Se lo facessimo con "One-time PAD" (cioè se cambiamo un solo bit della chiave) cambierebbe solo un bit del testo cifrato. Quindi One-time PAD non ha buona diffusione né buona confusione paradossalmente, pur essendo un cifrario a segretezza perfetta. La caratteristica forte di OTP è che la chiave è **casuale** e lunga quanto il testo in chiaro.

Però effettivamente l'attaccante potrebbe cambiare qualcosa del testo cifrato e se conosce bene la struttura del testo in chiaro questo cambiamento potrebbe non essere rilevabile: ad esempio se sa che in un testo cifrato con OTP o con un qualsiasi cifrario a flusso che ha intercettato il byte 4 corrisponde al numero del saldo di una carta di credito, potrebbe modificarlo per rendere il numero diverso e di conseguenza guadagnarci su senza farsi scoprire.

Il padding c'è sempre e la sua lunghezza varia da 1 byte a tutto il blocco e si trova alla fine del testo cifrato abbiamo visto. Lo standard PKCS#5 specifica un padding deterministico.

una sfida: padding PKCS#5

- il padding c'è sempre e la sua lunghezza varia da 1 byte alla lunghezza del blocco
- lo standard PKCS#5 specifica un padding deterministico
- come si può verificare come è fatto il padding facendo opportune cifrature e decifrature, e manipolando il testo cifrato?

Proviamo a verificare come è fatto il padding facendo opportune cifrature e decifrature e manipolando il testo cifrato.

è sempre riconoscibile ed è deterministico

[ESERCIZIO PER CASA]

Problema:

Supponiamo per esempio di avere un testo in chiaro di 8 byte ripetuti identicamente e di cifrarlo con DES 8 byte. Il problema di questo caso (in un testo come per esempio "elefantelefantelefante") è che se uno vede il testo cifrato ci sono un sacco di cose che capisce, innanzitutto che fa tutti blocchi identici. Naturalmente (non nel nostro caso ma in generale) se il testo è abbastanza lungo, è possibile che i blocchi si ripetono e quindi l'attaccante può notarlo.

Esempio gioco della settimana enigmistica con un cifrario di cesare (antichi cifrari). Questi sono cifrari di sostituzione in cui ogni lettera viene sostituita con un'altra lettera. Uno per esempio è quello di sostituire quella lettera con 3 caratteri più avanti. Questi cifrari sono molto deboli perché è facile fare un'analisi di sequenza. Nella settimana enigmistica (se cifrato) per esempio la prima cosa che faccio è vedere la lettera più frequente. In questo modo analizzando la sequenza e la disposizione delle lettere del testo cifrato si capisce qual è il testo in chiaro. In particolare, più la struttura è nota più è facile per esempio una lettera isolata non può essere una consonante .

Quindi sono molto deboli perché è facile fare un'analisi di frequenza e quindi analizzando anche la disposizione delle lettere nel testo cifrato si capisce qual è il testo in chiaro.

Dato che viene criptato con AES in questo modo:

Dato che è in modalità ECB verranno cifrati a blocchi, quindi per scambiare i dati di 2 persone basterà prendere il messaggio criptato nel punto corrispondente al messaggio in chiaro che ci interessa, e scambiarlo con l'altra parte del messaggio criptato che vogliamo scambiare.

Come utilizzare Okteta e openssl per capire come è fatto il padding:

Non ce lo dice, mannaggia.

Abbiamo parlato di **ECB** che è una modalità per cifrare i singoli blocchi.

Tutti i blocchi vengono cifrati nello stesso modo con la chiave , e alla fine il testo cifrato è una concatenazione dei blocchi cifrati.

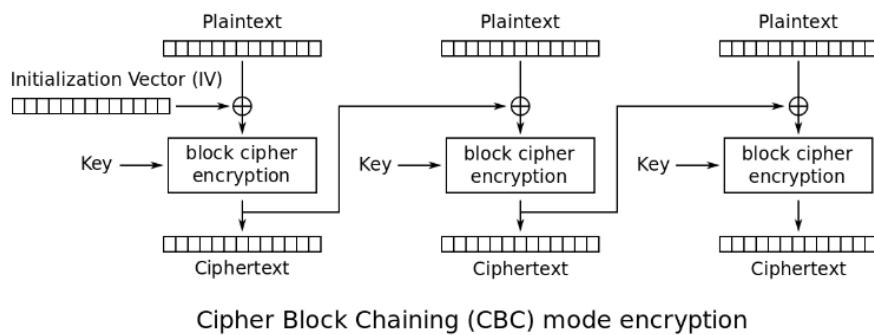
La volta scorsa abbiamo visto che **la debolezza di ECB è che lo stesso blocco viene cifrato sempre nello stesso modo e quindi l'attaccante anche senza conoscere la chiave può farsi un'idea del contenuto del testo** grazie all'**analisi di frequenza**, conoscendo la struttura testo in chiaro (non è una grande informazione per l'attaccante questa) etc.

Si utilizzando quindi altre modalità per cifrare:

CBC (CIPHER BLOCK CHAINING)

CONCATENAMENTO DEI BLOCCHI CIFRATI

cifratura

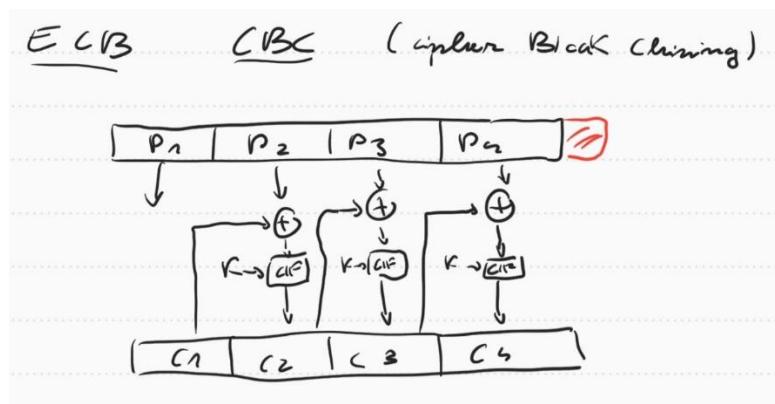


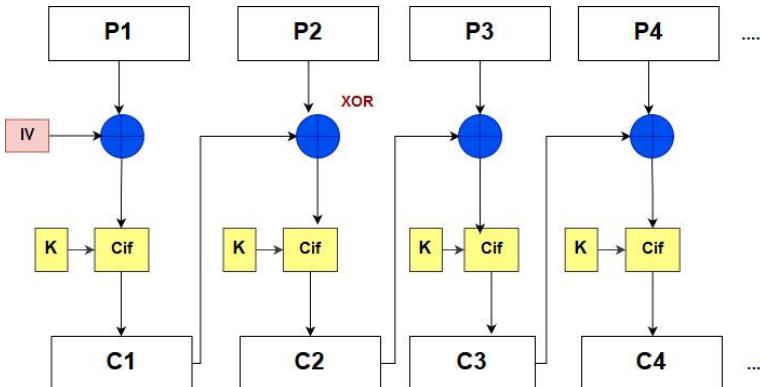
Cipher Block Chaining (CBC) mode encryption

Quando uno scrive un testo scientifico e vuole introdurre un acronimo , nel testo la prima volta scrive Cipher Block Chaining e poi (CBC) tra parentesi (per la relazione tesi!).

CIFRATURA

In CBC abbiamo un testo in chiaro di lunghezza arbitraria, con blocchi della stessa lunghezza ed un eventuale padding (quello in verde).





Supponiamo di aver cifrato già il primo blocco, cosa succede al secondo blocco in chiaro?

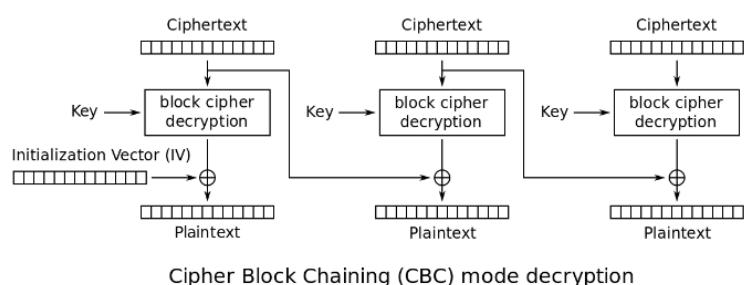
Il primo blocco di testo in chiaro, dato che non possiamo usare il testo cifrato generato dal blocco precedente, viene messo in XOR con un **IV (Initialization Vector)**, il **testo CIFRATO** generato dall'algoritmo di cifratura viene quindi messo in XOR con il **secondo blocco di testo in CHIARO**. Il **risultato viene cifrato con la chiave k** (con un cifrario a blocchi) e si ottiene così il **secondo blocco cifrato**, poi il **secondo** viene usato per metterlo in XOR con il **terzo** e così via.

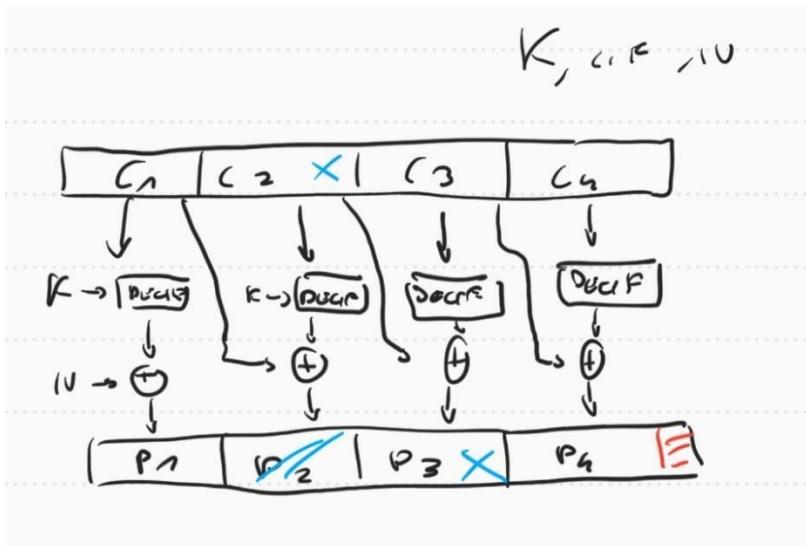
Si chiama così (cipher block chaining) perché c'è **una concatenazione dei testi cifrati con lo XOR del testo in chiaro successivo**. Il **vettore iniziale** non deve necessariamente essere un segreto infatti **è noto mentre k è un segreto**. Spesso l'IV viene scelto in modo pseudo casuale per evitare alcuni attacchi.

DECIFRATURA

Per decifrare si fa l'operazione (quasi) inversa: dal primo testo cifrato C1 si decifra con la chiave e si mette in XOR con l'IV, ottenendo il messaggio in chiaro P1, e C1 viene anche messo in XOR con il risultato della decifratura tramite chiave K di C2, ottenendo P2 e così via.

decifratura





A destinazione bisogna sapere qual è l'IV iniziale. A destinazione arrivano i blocchi cifrati C1,C2,C3,C4 ed è noto la Chiave, il Cifrario e l'IV. Viene decifrato il primo testo cifrato C1 con la chiave K e l'output viene messo in XOR con l'IV e si ottiene il testo in chiaro P1.

C2 viene decifrato con la chiave K ,l'output viene messo in XOR con C1 e si ottiene P2 e così via. Nell'ultimo blocco verrà riconosciuto il padding e verrà cancellato così che il testo in chiaro venga riportato com'era prima della cifratura.

Confronto con ECB:

Esaminiamo cosa succede se cambia un bit del testo in chiaro P2: se il cifrario ha una buona diffusione allora cambia tutto il testo cifrato, in particolare sia C2 che i testi cifrati successivi (C3)... cambieranno completamente perché C2 è cambiato e quindi si propagherà la diffusione sui successivi blocchi cifrati.

Quindi CBC ha la proprietà di diffondere la diffusione al di fuori del singolo blocco. Mentre la diffusione è un principio che in ECB vale sul singolo blocco, qui si propaga su tutti i blocchi successivi.

Se P2 e P3 sono uguali C2 e C3 non saranno uguali perché P2 viene cifrato con C1 XOR P2 mentre P3 viene cifrato con C2 XOR P3 quindi l'effetto per cui si poteva fare un attacco di tipo dizionario non è realizzabile in questo caso.

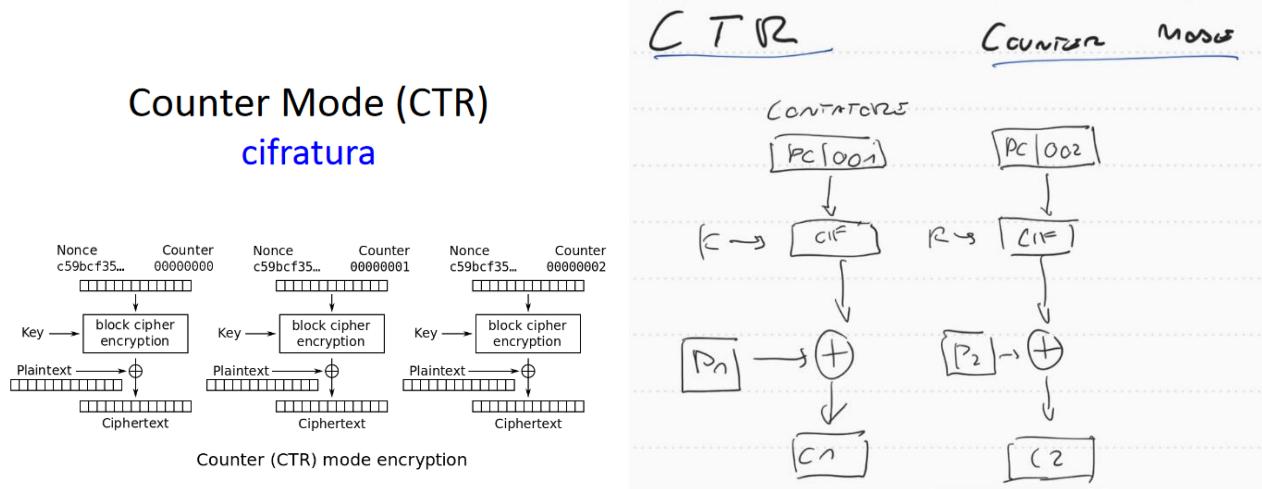
Svantaggi:

1. Non si può parallelizzare le operazioni perché sono da fare sequenzialmente.
2. Se C2 arriva con un bit corrotto, P2 viene corrotto completamente. In P3 sarà sbagliato solo il bit corrispondente nello XOR, mentre P4 non varia perché dipende da C3 e non da C2.

Quindi nella decifratura CBC un bit ha influenza solo nel proprio blocco e in un bit nel blocco successivo, quindi servono 2 testi cifrati consecutivi per decifrare un testo in chiaro: se i 2 testi cifrati che vengono utilizzati per decifrare un blocco sono corretti allora la loro decifratura sarà corretta. Per esempio, nel caso di P2 devono essere corretti C2 e C1 in questo caso non ci sono errori nella decifratura perché P2 non dipende da altri testi cifrati oltre a C1 e C2.

CTR COUNTER MODE

È un'altra modalità di cifratura: Dato un testo in chiaro che è più lungo di un blocco come lo cifro? Possiamo usare CBC ECB ..



CIFRATURA

Parte da un contatore, **in realtà lo standard prevede che sia fatto in parte da una stringa pseudocasuale (nonce) seguita poi da un contatore, lunga quanto un blocco del cifrario che si sta utilizzando**, il contatore viene cifrato con la chiave K e l'output(contatore cifrato) viene messo in XOR con il primo blocco di testo in chiaro P1 e si ottiene il testo il primo testo cifrato C1.

Se è un blocco di AES sarà lungo 128 bit quindi la sequenza pseudocasuale sarà 64 bit e il counter 64 bit

Cosa succede **nel secondo blocco?**

1. **il nonce rimane lo stesso mentre il contatore viene incrementato di 1**
2. **Viene cifrato come prima con lo stesso cifrario con la chiave k**
3. **Viene eseguito lo XOR con il testo in chiaro P2 e si ottiene C2**

Questo quindi viene fatto indipendentemente dal primo, perciò **si può parallelizzare l'operazione di cifratura**.

La vera procedura di cifratura qui è lo XOR (infatti assomiglia a un cifrario a flusso).

Quindi **CTR è un modo per trasformare un cifrario a blocchi in uno a flusso**, in quanto la parte di contatore cifrata genera una sequenza pseudocasuale che messa in XOR con il messaggio in chiaro restituisce un messaggio cifrato.

Sostanzialmente la parte al di sopra della XOR è un generatore di sequenze pseudocasuali con la differenza che le genera blocco per blocco a partire dal contatore cifrato con la chiave. E la cifratura del testo in chiaro non è più fatta con la chiave, ma è invece uno XOR.

IL fatto che somigli a un cifrario a flusso **si porta dietro il problema della NON diffusione con cifratura con lo XOR**: se cambio un bit del testo in chiaro cambio solo un bit del testo cifrato quindi da questo punto di vista è una debolezza: l'attaccante se conosce un minimo di struttura del testo in chiaro (es. dei comandi della telecamera e delle luci) può modificare un bit del testo cifrato che va a modificare un bit del testo in chiaro generando una modifica difficile da individuare mentre lui rimane nell'anonimato, e questa modifica può essere cruciale (spegne la telecamera e la luce per liberare l'ostaggio senza farsi scoprire).

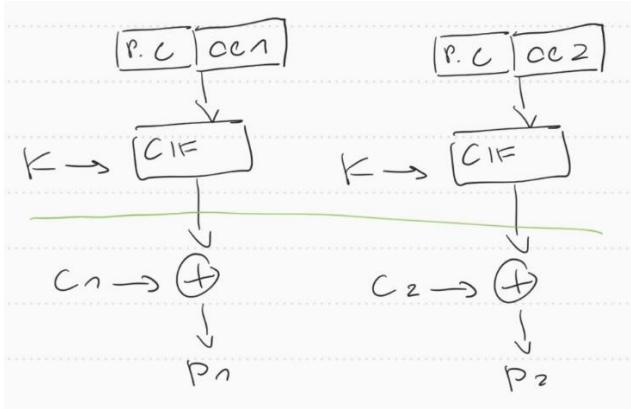
Se i 2 testi in chiaro sono uguali vengono cifrati nello stesso modo come ECB?

No perché questo dipende dalla sequenza pseudocasuale e **dato che il contatore viene incrementato volta per volta** avremo una cifratura diversa in quanto **la sequenza pseudocasuale cambia ad ogni blocco** (il nonce non cambia, però incrementando il contatore è come se cambiasse l'intera sequenza pseudocasuale)

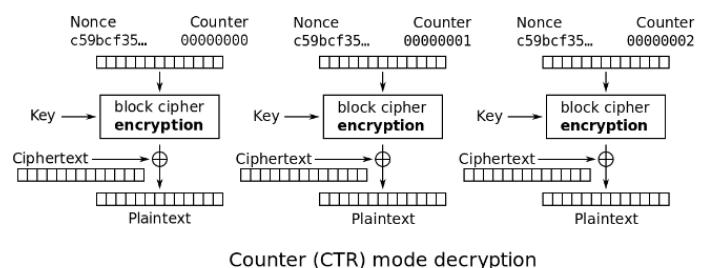
Se il cifrario ha una buona proprietà di diffusione, nonostante tra i diversi contatori venga modificato un solo bit, tra 2 blocchi il risultato cambia completamente.

DECIFRATURA

Come si decifra?



Counter Mode (CTR) decifratura



Dobbiamo conoscere: La chiave e il contatore

Rigenero la sequenza pseudo casuale, quindi la prima parte dell'algoritmo rimane uguale e la cifro con K il risultato di questa operazione lo metto in XOR con il testo cifrato(al posto del testo in chiaro che abbiamo in fase di cifratura) e ottengo il testo in chiaro. In sintesi: **decifro mettendo in XOR il risultato della cifratura con K del nonce + contatore con il blocco cifrato.**

L'attaccante non conosce la **chiave (stessa situazione cifrario a flusso)**.

Notiamo che qui abbiamo un parametro in più.

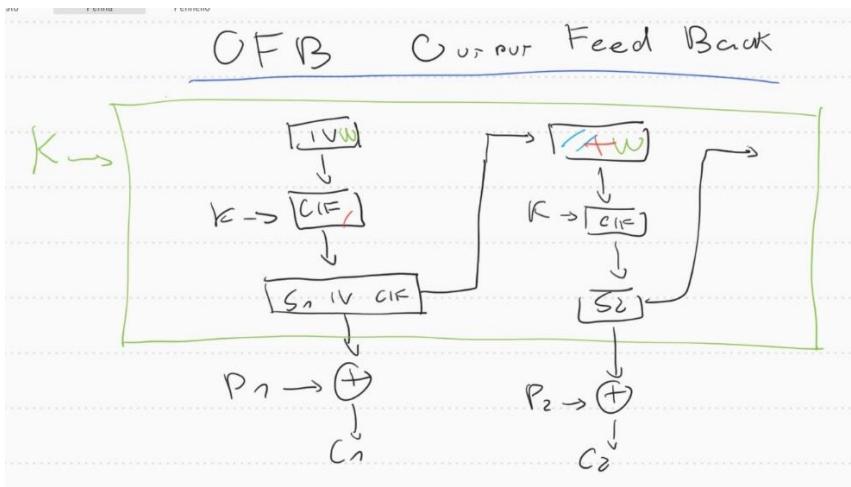
Nei cifrari a flusso abbiamo visto che se si cifravano blocchi differenti con la stessa chiave, dato che il generatore veniva inizializzato solo con la chiave, se noi mettiamo 2 volte la stessa chiave questa ci fornirà la stessa sequenza pseudo casuale in output

Quindi aveva la debolezza che mettendo in XOR i testi cifrato si otteneva lo XOR dei testi in chiaro mentre qui non vale più questo, perché c'è anche il contatore che viene inizializzato in modo diverso, e **anche se il contatore è noto non comporta niente in quanto non essendo nota la chiave c'è un parametro in più che varia.**

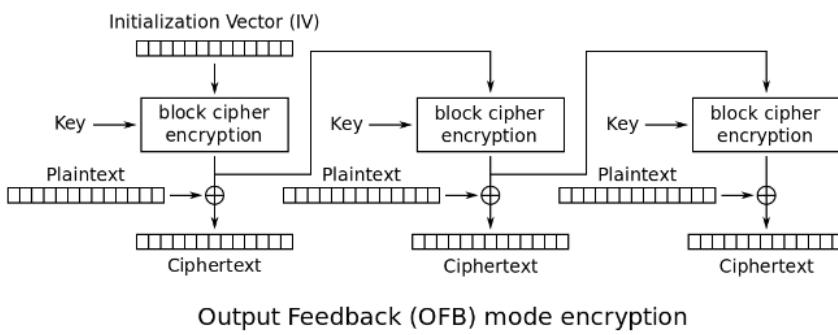
Una cosa fondamentale, abbiamo presentato DES e AES le scorse lezioni e abbiamo visto che **la decifratura di DES era lo stesso algoritmo solo che utilizza le chiavi di round al contrario mentre AES usa un algoritmo di decifratura diverso rispetto a quello di cifratura. CTR ci permette di cifrare e decifrare utilizzando solo un algoritmo di cifratura del cifrario a blocchi che viene utilizzato e quindi è stato usato volentieri con AES** in quanto non c'è più il bisogno di implementare un algoritmo di decifratura apposito.

In realtà ci sono altre modalità di cifratura dei messaggi lunghi, gli altri 2 che vediamo sono altre modalità che tendono a trasformare il cifrario a blocchi in cifrario a flusso: **OFB: Output Feed Back**

OFB (OUTPUT FEED BACK)



OFB (cifratura)



CIFRATURA

OFB prende in input un IV (ma potrebbe essere con lo stesso senso di un contatore), questo viene cifrato con la chiave K ottenendo così IV cifrato, il quale può essere usato in due modi:

1. Nel suo complesso per metterlo in XOR con il testo in chiaro e generare il testo cifrato
2. Si può usare solo un pezzetto dell'IV cifrato generato e quindi in questo modo avremmo generato un byte di sequenza che viene messo in XOR con un byte del testo in chiaro e si ottiene un byte del testo cifrato, dopodiché l'output della cripitura(cioè l'IV cifrato) viene usato sostanzialmente come prossima base(prossimo IV) per generare il blocco successivo di sequenza.

A questo punto(nel caso in cui l'IV cifrato non viene usato nel complesso ma solo in parte) la parte sopra se si vede come una scatola chiusa nel quale si dà in input una chiave e un IV si ha la stessa struttura di un cifrario a flusso

Infatti dato un K e un IV questo genera una sequenza di byte uno dopo l'altro e quindi è un generatore pseudocasuale di un cifrario a flusso, con l'unica differenza che utilizza questo generatore mettendo nel cifrario da una parte AES e dall'altra DES o qualsiasi altro cifrario.

Quindi succede che l'IV cifrato diventa il nuovo IV per il blocco successivo quindi verrà nuovamente cifrato.

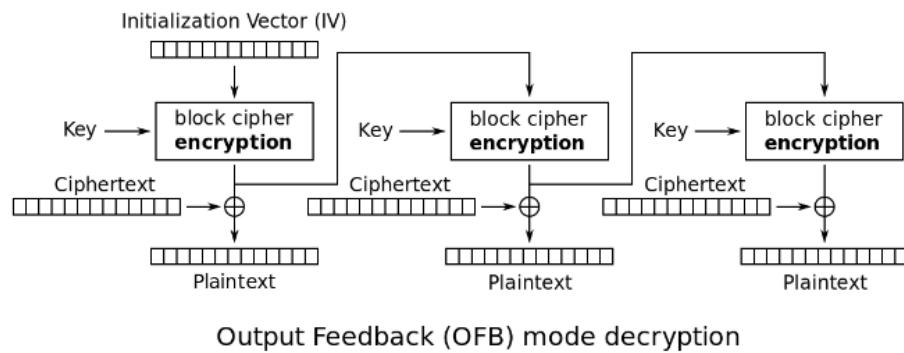
L'aspetto più tecnico e specifico che ci interessa limitatamente è che il generatore pseudocasuale può essere usato totalmente o parzialmente e questo influisce sul modo con cui viene reinizializzato l'IV iniziale successivo.

C'è una dipendenza quindi il vecchio IV cifrato che è stato appena generato e diventa l'input per la cifratura di C2.

DECIFRATURA

Come si decifra?

OFB (decifratura)



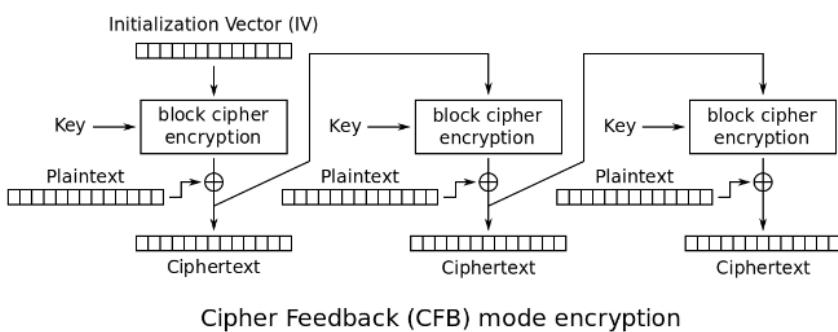
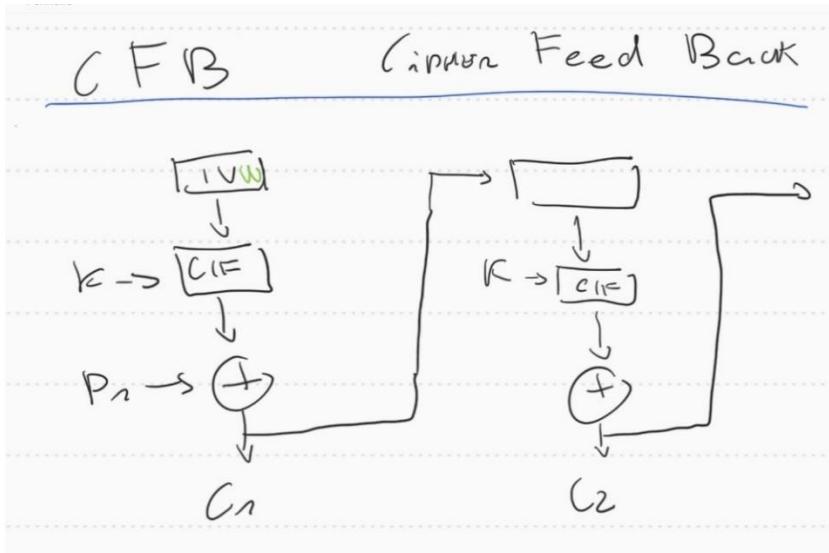
La parte superiore non cambia, e poi si inverte il testo cifrato in input e il testo in chiaro.

Diffusione : No

Parallelizzabile: No però non è necessario avere tutti i testi si potrebbe generare prima la chiave(cioè generare prima i vari IV cifrati) e poi parallelizzare

Errori di trasmissione nel testo cifrato: Determinano dei cambiamenti di bit corrispondenti nel testo in chiaro

CFB: CIPHER FEED BACK:



CIFRATURA

In modo analogo a OFB si parte da un IV che viene cifrato con la chiave K, l'output viene messo in XOR con il testo in chiaro e l'output finale (cioè il testo cifrato) viene riportato come input del successivo blocco e così via. Quindi il risultato del testo cifrato C1 viene cifrato con la chiave K e poi messo in XOR con il testo in chiaro P2 per ottenere C2.

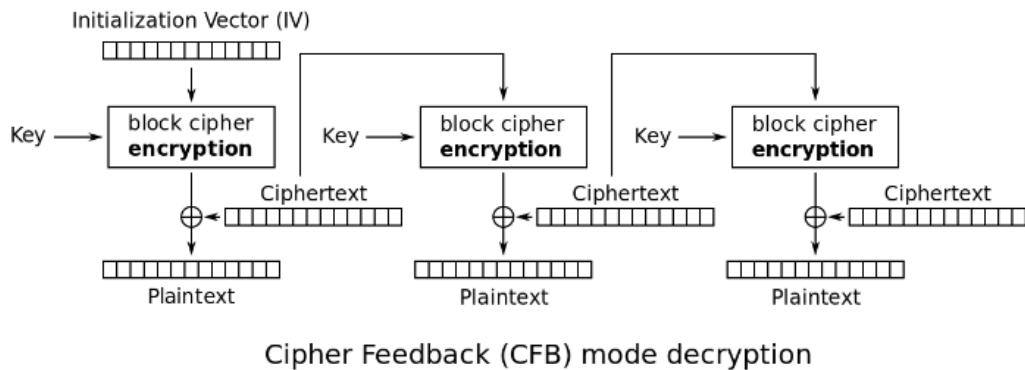
Com'è la diffusione? Se cambiamo un bit del testo in chiaro P1 cambia il testo cifrato C1 in un bit ; però il testo cifrato C1 in base alle condizioni del cifrario(cioè se la diffusione del cifrario sono buone) potrebbe cambiare tutti i testi cifrati dal secondo in avanti.

Quindi la diffusione qui è diversa perché **non c'è diffusione nello stesso blocco(infatti nello stesso blocco cambia solo un bit) ma su blocchi diversi può esserci.**

La cifratura non è parallelizzabile perché per cifrare un blocco successivo ho bisogno del blocco precedente.

Se c'è un errore nella cifratura (un bit del testo cifrato) da lì in poi cambieranno tutti.

DECIFRATURA



Partiamo da C1 e C2, abbiamo la chiave e l'IV.

Prendiamo l'IV lo cifriamo con la chiave K e lo mettiamo in XOR con C1 e otteniamo P1 e poi prendiamo C1 come input del successivo che servirà per generare il prossimo pezzo di sequenza che servirà per metterlo in XOR con C2 e generare P2 e così via.

Se c'è un bit corrotto in C1, l'influenza è che su C1 si corrompe solo un bit, su P2 invece potrebbero potenzialmente cambiare tutti se il cifrario ha sufficiente diffusione, mentre sul blocco successivo ancora (quello corrispondente a c3) essendo in input C2, l'errore non sarà propagato (a meno che C2 Non sia a sua volta corrotto)

La decifratura è parallelizzabile se ad ogni processore do 2 cifrati contigui (es. c1 e c2).

Ricapitolando:

ECB ha diffusione solo all'interno del blocco

CBC ha diffusione all'interno nel blocco e su tutti i successivi

CTR no diffusione

OFB nemmeno perché è in sostanza un cifrario a flusso

CFB ce l'ha all'esterno non nel blocco stesso. Però ha una cifratura non parallelizzabile.

ESERCIZI CBC

Esempio

cifrare un file con DES-CBC (usare l'opzione: nosalt)

Quanto è lungo il testo in chiaro? Deve essere la stessa di ECB (cambia nulla)

Prendiamo il testo da 8 byte uguali (elefanteelefanteelefanteelefante) lo cifriamo e guardiamo l'output

Se guardiamo notiamo che al contrario di quello che succede con ECB non si riconoscono i blocchi uguali. Perché il secondo blocco viene cifrato messo in XOR con il testo cifrato precedente.

0000:0000 53 61 6C 74 65 64 5F 5F 85 D0 B9 C0 AB 1D 2F E7	SalTED_ .D¹À«./ç
0000:0010 FB 70 A4 70 B3 CE BB 6B 8E B3 8E E6 AF 20 9D 3A	Ùp¤p³Ì»k.³.æ- .:
0000:0020 BE 78 FD 47 3B 1F 3D 54 63 EA 9A 55 94 63 F2 BA	¾xýG; .=Tcê.U.còº
0000:0030 99 F5 D2 4E A8 4B CF F1 03 0A 00 E0 4D 9A F8 2E	.öÖN"Kïñ...àM.ø.
0000:0040 CC BF 60 6E 19 C1 E0 FE	Ì`n.Àäþ

Con l'opzione -p mi mostra quale IV(vettore iniziale) viene utilizzato

Ci dice appunto che viene utilizzato un IV iniziale e una certa chiave.

L'IV viene generato con un certo algoritmo deterministico.

Vediamo cosa succede

1) se si cambia un bit del testo in chiaro

Fase di cifratura: Si propaga su tutto il blocco corrispondente se il cifrario è molto diffusivo e inoltre anche tutti i blocchi successivi verranno cambiati perché poi verrà messo in XOR il nuovo testo cifrato

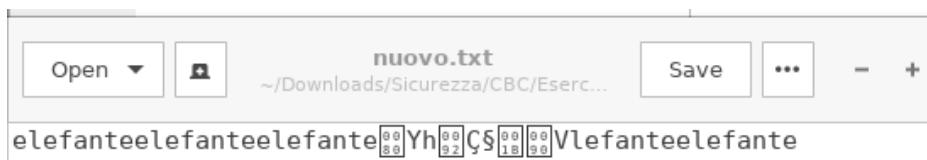
2) se si cambia un bit del testo cifrato in un blocco, non nell'ultimo però (PER IL PADDING)

Nella fase di cifratura: Cambiando un bit del testo cifrato la modifica si propaga su tutti i testi cifrati successivi per via della concatenazione invece nella fase di decifratura la modifica si propaga tutto il blocco corrispondente e un bit di quello successivo

Cambio in C un bit ottengo

elefanteelefanteelefanteYhÇ§Vlefanteelefante

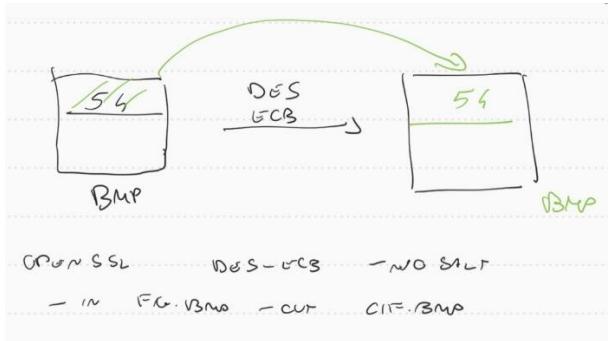
0000:0000 53 61 6C 74 65 64 5F 5F 85 D0 B9 C0 AB 1D 2F E7	Salted_ .D¹À«./ç
0000:0010 FB 70 A4 70 B3 CE BB 6B 8E B3 8E E6 AF 20 9D 3A	Ùp¤p³Ì»k.³.æ- .:
0000:0020 BE 78 FD 47 3B 1F 3D 54 50 EA 9A 55 94 63 F2 BA	¾xýG; .=TPê.U.còº
0000:0030 99 F5 D2 4E A8 4B CF F1 03 0A 00 E0 4D 9A F8 2E	.öÖN"Kïñ...àM.ø.
0000:0040 CC BF 60 6E 19 C1 E0 FE	Ì`n.Àäþ



ESERCIZIO LOGHI:

Piccola introduzione sulla struttura del bitmap(i primi 54 byte sono di intestazione)

Si ha un file BMP quindi un file bitmap. Per ogni pixel usa da 1 a 4 byte



Cifriamo un logo con des-ecb con il comando solito(openssl ecc..)

Una volta cifrato prendo i primi 54 byte(di intestazione) del testo in chiaro e li sostituisco con i primi 54 di quello cifrato.

Questo ci serve per portare in chiaro l'estensione del file bmp in modo da poterla poi aprire come un file bmp. Lo facciamo con Okteta.

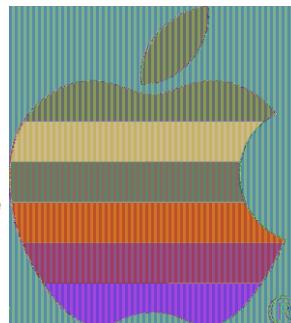
Successivamente cambiamo estensione e la mettiamo .bmp

Facciamo l'esercizio prima cifrandolo con DES-ECB e poi con AES-ECB. Scegliamo il logo che ci piace. Per Info leggere README.txt

Scegliete uno dei loghi in formato BMP che trovate in questa directory.

Cifrare il file con des-ecb. Aprire il file originale e il file cifrato con okteta e copiare i primi 54 byte del file bmp al posto dei primi 54 del file cifrato. Dare l'estensione bmp al file cifrato e aprirlo.

Provare con aes-ecb. Che differenza notate?



Viene fuori un qualcosa di riconoscibile. All'interno del blocco c'è la diffusione. Cifra tutti i blocchi uguali quindi cambiano tutti i blocchi allo stesso modo. Proviamo anche a cifrare con AES 128-ECB 128 o 256 bit della chiave. All'interno del blocco c'è la diffusione

È simile perché i blocchi sono più grandi rispetto a des e quindi mescola i pixel con lo stesso colore.



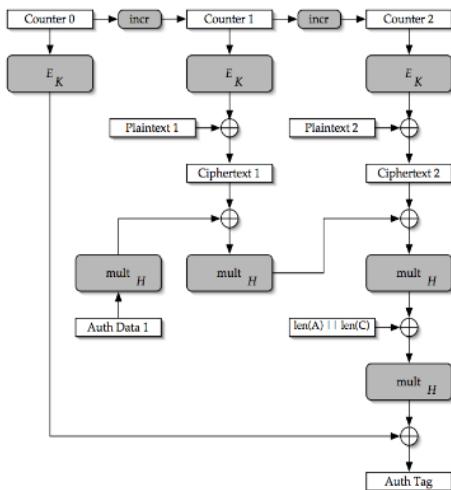
Se lo facciamo con cbc cfb ctr ofb e altri succede che cambia tutto non vengono più elaborati i singoli pixel o blocchi di pixel da soli ma c'è il fenomeno della diffusione o meglio 2 blocchi di pixel uguali vengono cifrati in modo diverso. Questo esercizio è un modo per toccare con mano l'inadeguatezza di ECB. Se si vede così con un immagine su un testo l'attaccante è avvantaggiato

USO DI CTR IN PRATICA: LE MODALITÀ GCM E CCM

Cos'è questo GCM? Scopriamolo

GCM (GALOIS COUNTER MODE)

cifratura



CIFRATURA

LA modalità di cifratura GCM è complessa da attaccare perché oltre a offrire riservatezza offre anche integrità (ha integrato un algoritmo di autenticazione). Il counter viene utilizzato all'inizio per la autenticazione, poi viene incrementato.

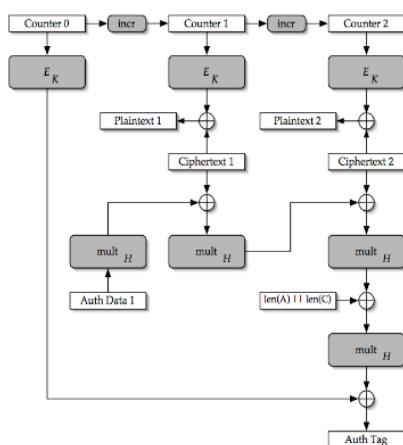
E_K =cifratura con chiave k

Il counter 1 viene cifrato con E_K e poi viene messo un XOR con il testo in chiaro 1 per dare il testo cifrato 1 poi viene incrementato e viene fatta la stessa cosa con il counter 2.

Quindi GCM per la cifratura del testo in chiaro utilizza la modalità CTR (cioè la modalità contatore). Poi c'è tutta un'altra parte che serve per l'autenticazione.

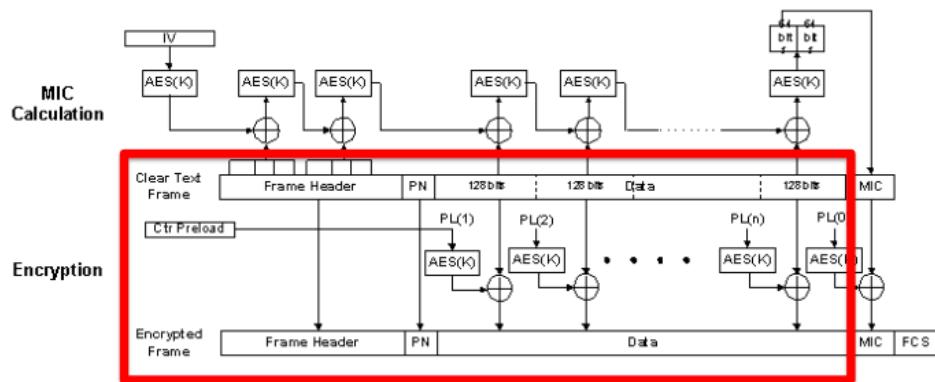
DECIFRATURA

decifratura



CCMP (COUNTER MODE/CBC-MAC)

CCMP è una variante che viene usata da WPA2 ed è un'altra modalità mista: sia per l'integrità che per la riservatezza ed è basata di nuovo sulla modalità contatore.



Il contatore viene cifrato con AES con chiave K e viene messo in XOR con il primo blocco del testo in chiaro (viene cifrata la parte interna del frame non l'header), poi viene incrementato il contatore, viene cifrato con AES e messo in XOR con il secondo blocco e così via. I blocchi vanno a finire nel frame cifrato il quale contiene tutto il payload cifrato (non l'header).

La ragione per cui esistono queste modalità miste la vedremo più avanti.

Sostanzialmente, la modalità contatore ha la debolezza di non avere diffusione quindi è sbagliato usarla da sola: quindi va utilizzata con l'autenticazione per evitare che l'attaccante possa modificare dei bit dal testo cifrato sapendo quali bit del testo in chiaro sta modificando.

Non è detto che 2 meccanismi di sicurezza combinati abbiano come effetto combinare le garanzie dei 2 meccanismi di sicurezza. Dipende da come li combino, potrei introdurre delle debolezze.

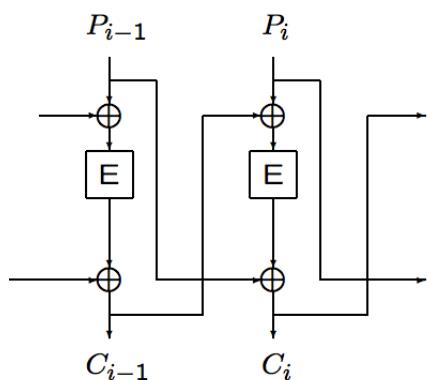
ESEMPIO

Se ho del traffico che passa dal firewall, mettiamo caso che ho del traffico cifrato che sto proteggendo, ottengo riservatezza però se faccio passare del traffico cifrato nel firewall quel traffico non può essere analizzato quindi se lo faccio passare vuol dire che anche l'attaccante può far passare del traffico cifrato che non verrà analizzato.

Quando uno entra in banca e ci sono telecamere. Se io permetto a qualcuno di passare con cappuccio in testa allora attaccante ovviamente andrà col cappuccio in testa. Il problema viene fuori nel cercare di ottenere integrità e riservatezza insieme.

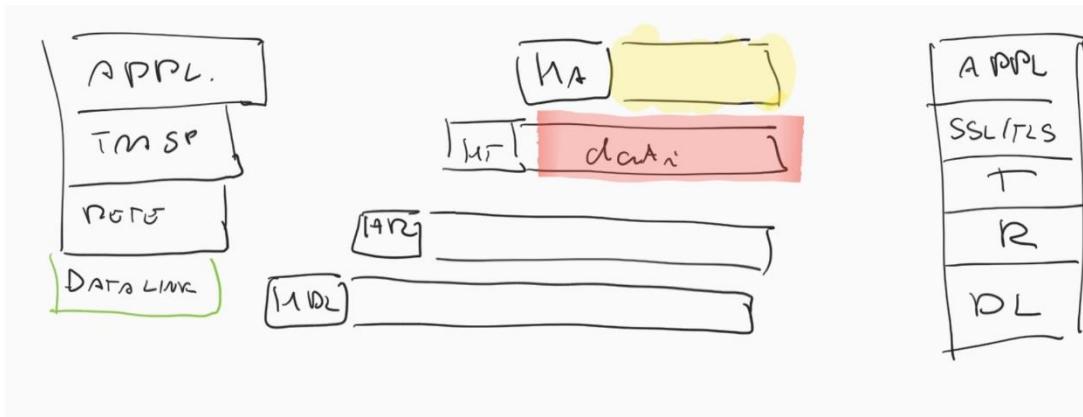
Ci sono altre modalità che non vedremo come CBC-IGE dove c'è uno XOR incrociato tra blocco in chiaro e cifrato successivo con precedente. (tanto non la chiede)

CBC-IGE:



CIFRATURA A LIVELLI DIVERSI DELLA PILA

Abbiamo i livelli di rete: data link, rete, trasporto, applicativo
Pila iso-osi



Se parliamo di traffico in rete sappiamo che l'incapsulamento arriva da livello applicazione a quello di trasporto, rete e infine data link.

applicazione-->header app + payload

trasporto-->header trasporto + payload

rete-->header rete + payload

data link-->header data link+....

Cosa/Quando bisogna cifrare? Si può cifrare i dati dell'applicazione oppure si può cifrare a livello di trasporto, rete etc. **a seconda del livello a cui uno vuole cifrare ottiene vantaggi e svantaggi diversi.**

CIFRATURA A LIVELLO APPLICAZIONE

In posta elettronica uno vuole inviare i messaggi cifrati, quindi i dati che vengono cifrati sono quelli del livello applicazione: i dati sono quindi protetti da quando finisco di scrivere la mail fino all'arrivo.

Quello che passa cifrato sono i dati ma l'intestazione è in chiaro.

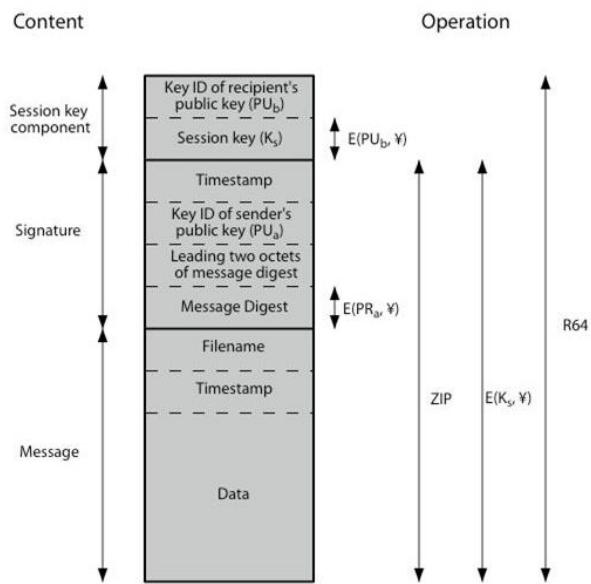
Vantaggio: i dati sono cifrati dall'inizio alla fine (dal momento che ho finito di scrivere l'e-mail all'arrivo) (la macchina cifra i dati e il destinatario decifrerà sulla sua macchina.) .

Svantaggi:

L'attaccante ha un sacco di informazioni accessorie: sa che è un e-mail, sa a chi è destinato.. ecc. Il secondo svantaggio è che è necessario fare questa implementazione nel programma di posta elettronica specifico: questo ha anche vantaggi perché l'algoritmo di decifratura/cifratura può seguire tutte le esigenze di quel dominio applicativo, però se voglio usarlo anche per navigare per qualunque altro programma di chat o altro dovrò implementare di nuovo una decifratura/cifratura dall'altra parte. Ciò nel mondo della sicurezza è un problema poiché ogni volta che implemento un aspetto di sicurezza in un programma rischio di inserire nuovi errori. Un esempio di posta elettronica può essere per esempio PGP.

PGP (PRETTY GOOD PRIVACY)

PGP Message Format



PGP (Pretty good privacy) è un programma in grado di mandare messaggi di posta elettronica cifrati. Il messaggio è composto da dati, nomefile e timestamp (data e ora), quella in signature è l'intestazione che serve per integrità etc. e sopra c'è una parte di scambio delle chiavi.

Il messaggio insieme alla parte della signature viene zippato e cifrato con una chiave simmetrica E_{ks}. Questa cifratura è a livello di applicazione che può essere realizzata con un cifrario a blocchi o a flusso (quindi a chiave simmetrica) e sopra c'è l'autenticazione della posta elettronica che viene fatta con la nozione del contenuto del campo dati a questo livello.

Se uno **cifra a livello trasporto** succede che **l'intestazione dell'applicazione questa volta è nascosta**.

Ci sarà **nell'header di trasporto il numero di porta** anche se non è troppo utile. Non c'è però l'header di livello applicativo ma ci sono sicuramente delle informazioni nascoste. Questo rispetto a quello di prima è un meccanismo che possono utilizzare più processi a livello applicativo poiché è una cifratura a livello più basso. Un modo per fare questa cosa qui è **TLS** poiché si pone appena sotto al livello applicazione nella pila.

Pila ISO/OSI

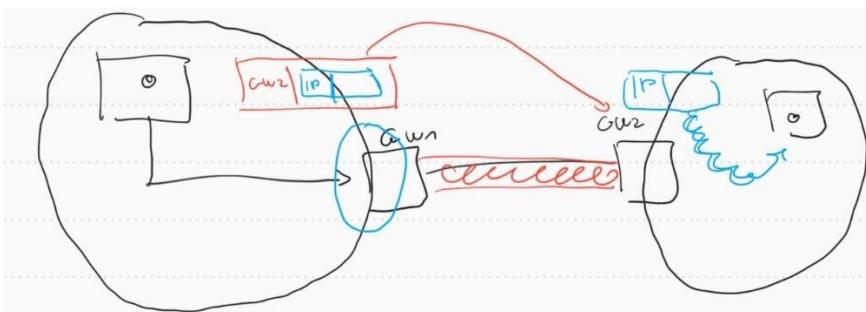


//SSL in realtà non è trasparente all'applicazione (es HTTP deve sapere che usa ssl/TLS)

CIFRATURA A LIVELLO DI RETE

Per **cifrare a livello di rete** significa che quindi **non sappiamo più nemmeno qual è la porta e le informazioni a livello trasporto**, quindi è più in basso (più facile da implementare) ma naturalmente **i dati devono arrivare fino a questo livello dello stack in chiaro**. Chi fa una cosa di questo genere è **IPSEC ESP** è un protocollo che trasforma un protocollo IP in versione sicura che può funzionare in 2 modalità:

- **Modalità trasporto:** È una modalità che **cifra tutti i dati di livello trasporto, si occupa della riservatezza e lascia fuori solo l'header IP e aggiunge un ESPauth**. In sostanza cifra tutti i dati lasciando fuori solo l'header IP come abbiamo visto prima. Questa è una modalità end to end come abbiamo visto prima perché se ciro a Livello di trasporto alla fine decifrerà la macchina a destinazione
- **Modalità tunnel:** È una modalità di cifratura di IPsec, in questo caso stiamo cifrando più in basso, viene **cifrato tutto il datagramma (compreso IP) e poi viene aggiunto un nuovo header IP** (senza l'IP il router avrebbe problemi e il pacchetto non potrebbe circolare per questo si aggiunge un nuovo header IP).



Questa è la configurazione in cui ci possiamo immaginare un'università con 2 sedi distaccate, e utilizza dei canali pubblici per collegarsi però vuole virtualizzare un **canale privato VPN**, quindi **avrà 2 gateway(GW1 e GW2)**.

I dati arrivano in chiaro fino al gateway di uscita della prima sede (GW1), poi se fosse un normale gateway lo instraderebbe al router, ma implementa la modalità tunnel di IPsec: prende il datagramma, lo cifra e lo infila in un altro datagramma nel quale ci mette come indirizzo di destinazione l'indirizzo di gateway2.

Arrivato a GW2, il datagramma che "contiene" quello vero verrà spacciato da GW2, che instraderrà il datagramma in chiaro verso la destinazione. **Quello che vediamo sul canale sarà sempre il traffico tra gateway 1 e gateway 2**: in pratica sappiamo che le 2 aziende stanno comunicando, ma non sappiamo chi (quale macchina) specificamente stia comunicando. Si chiama modalità tunnel perché i dati vengono incapsulati così come arrivano e vengono spediti dall'altra parte e continuano il loro viaggio dall'altra parte.

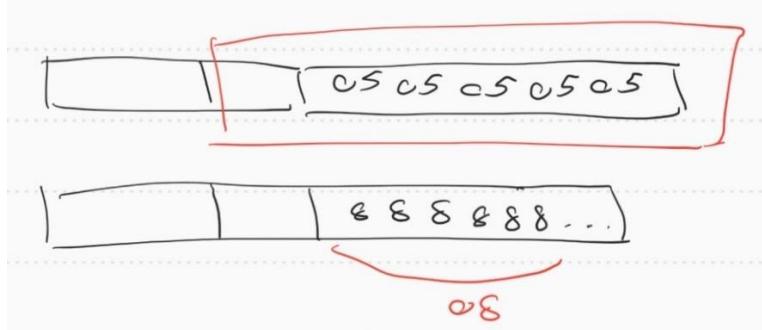
Vantaggi Tunneling:

1. **Senza il tunneling si saprebbe che il messaggio è destinato da macchina 1 a macchina 2, che sarebbe un bel vantaggio per l'attaccante.** (analisi del traffico)
2. **Il tunnel è imposto dall'azienda (tecnicisti ecc.), quindi è garantito che ci sia una forma di privacy verso l'esterno** invece di lasciare che lo faccia con cifratura end to end di utenti singoli, che non è così semplice da fare e non si può controllare così capillarmente, e può quindi portare delle esposizioni di dati che non sono voluti per esempio dall'amministrazione.
3. Prima la cifratura doveva essere fatta sulla singola macchina che è del singolo utente mentre **nel tunnel la cifratura/decifratura viene effettuata a livello di gateway** (fatti apposta per quello e che sono stati dimensionati tipicamente apposta per quello) **che dovrebbero avere performance adeguate per cifrare il traffico dato che non è detto che le singole stazioni (macchine) siano adeguate in questo lavoro**. Ciò non toglie che uno possa cifrare a livello di trasporto dentro un tunnel, per esempio se si decide che quello che le due aziende si stanno dicendo sia da proteggere anche all'interno dell'azienda, possono utilizzare una cifratura a livello di trasporto che poi viene inserita in un tunnel, con lo svantaggio di appesantire l'intero processo di invio dati.

Quindi si può cifrare su più livelli.

I dati vengono sempre cifrati a chiave simmetrica!!

I PKCS, STANDARD DE FACTO



Il padding PKCS#5 cosa fa? (non lo chiede in teoria)

Si chiama PKCS#5 perché è il numero 5 ce ne sono vari forse una decina.

Aggiunge sempre un padding e ha senso quando c'è un blocco che non è completo siccome il cifrario deve lavorare su blocchi completi. Deve quindi essere possibile distinguere il padding dal file: se il file finisse con bit simili a quelli del padding, si potrebbero confondere quindi si aggiunge sempre il padding così siamo sicuri che ci sia.

Sia nel caso in cui il messaggio è un multiplo di 8 byte che nel caso in cui non lo sia, si aggiunge sempre un padding per far sì che diventi un multiplo di 8 (DES).

Ogni byte dice quanti byte ci sono nel padding: se il padding è di 5 byte, i byte del padding diranno tutti 0505050505, se sono 8 diranno tutti 0808080808.... **se il file finisse con tutti 08 il meccanismo trova il padding, elimina il padding lasciando inalterato il msg (quindi era già un multiplo di 8).** quindi non c'è possibilità di eliminare byte del msg o di lasciare byte aggiuntivi. Se decifriamo DES e troviamo che ci sono meno byte di quelli dichiarati ci direbbe BAD magic number.

PKCS significa PUBLIC KEY CRYPTOGRAPHY STANDARD :Questo è uno standard per il cifrario a blocchi.

Questa sigla è sbagliata perché:

PUBLIC KEY: noi stiamo parlando di cifrari a chiave simmetrica (non a chiave pubblica) infatti il padding è definito per cifrari a chiave simmetrica come DES o AES

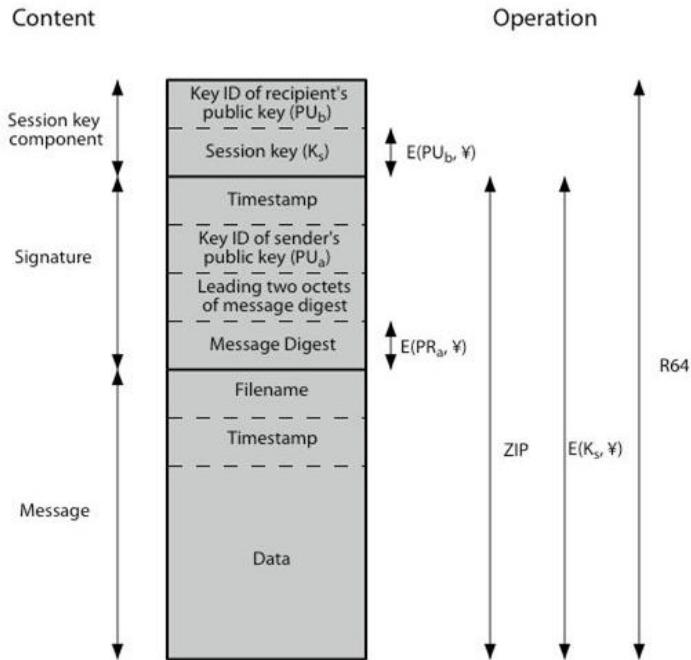
STANDARD: non sono stati standardizzati da un ente. Li hanno proposti così e tutti li usano ma la realtà dei fatti è che se uno implementa qualcosa che non sia nello standard PKCS è difficile che funzioni.

Stiamo parlando del padding che è appropriato e definito per cifrari a blocchi, che sono cifrari a chiave simmetrica. La ragione è che questi standard sono stati introdotti per sistemi a chiave pubblica. Sono stati introdotti dalla RSA (azienda che ha inventato questo cifrario) e sono uno standard di fatto poiché tutti lo utilizzano.

CIFRATURA A LIVELLO APPLICAZIONE: PGP

PGP:(ne abbiamo già parlato la scorsa lezione)

PGP Message Format



Cifratura a livello applicazione. Avviene sui dati dall'applicazione. Può essere molto specifica per quell'applicazione, il male è che ogni APP deve occuparsi di implementare un proprio metodo di cifratura.

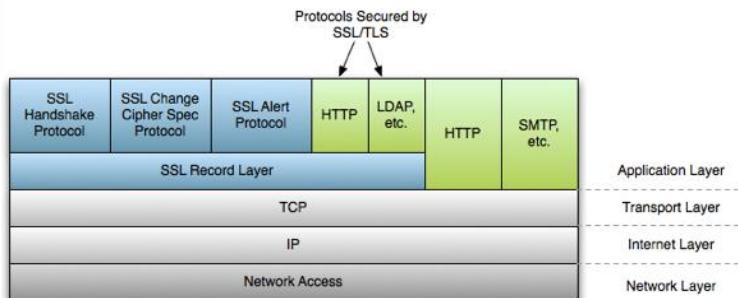
La cifratura usa determinati algoritmi per decifrare. La cifratura avviene a chiave asimmetrica, preferibilmente con AES (anche DES se fosse) o altri cfrari come [Camelia](#) ecc..

CIFRATURA LIVELLO PRESENTAZIONE: TLS/SSL

Cifratura a un livello un po' più basso, precisamente a livello presentazione (a metà tra applicazione e trasporto)

Differenza tra SSL/TLS è minima, SSL È stato introdotto inizialmente per proteggere HTTP poi è stato standardizzato per TLS, la differenza più importante è che a un certo punto SSL è stato abbandonato e non ha più ricevuto attenzione quindi è inopportuno utilizzarlo perché diventa pericoloso in quanto non è stato aggiornato.

Quindi a livello logico è lo stesso mentre a livello pratico no.



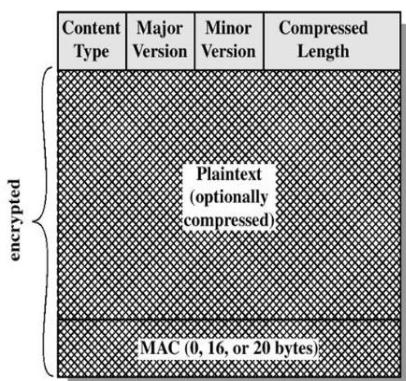
Questo è di SSL ma cambia poco da TLS.

Se HTTP non è protetto, normalmente sta sopra a TCP in quanto viaggia su TCP (HTTP usa TCP). Se invece è protetto, HTTP viaggia sopra il record layer che vuol dire che i dati di HTTP vengono cifrati quindi impacchettati nel modo che vedremo tra poco (figura dopo) e poi viaggia tutto ancora su TCP.

Se io cifro a questo livello, tutta la parte a livello di applicazione viene cifrata.

Il record layer sta sopra TCP e l'HTTP protetto sta sopra al record layer, quei 4 in azzurro sono tutti protocolli in SSL, Il record layer è il protocollo responsabile per la cifratura e l'integrità dei dati.

SSL Record Format



C'è il testo in chiaro che può essere anche compresso, ci sono dati per l'autenticazione di cui parleremo più avanti, e tutto questo è cifrato, e poi c'è un header con informazioni che serve per gestire questi dati.

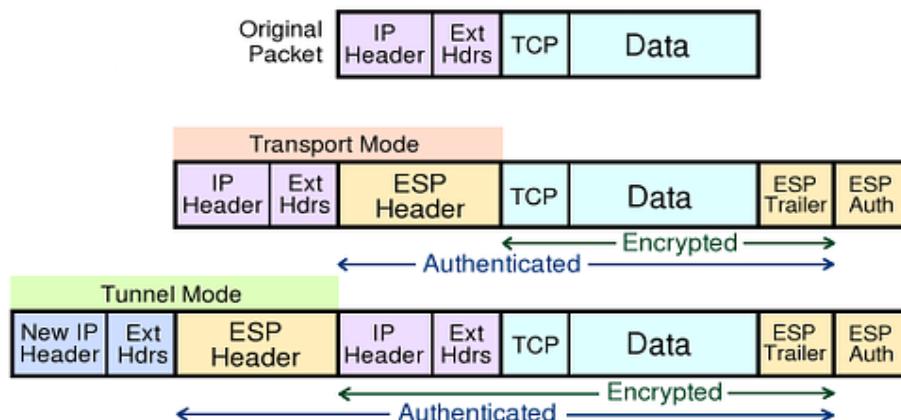
I cifrari che vengono usati sono AES, GCM, CCM, CBC, Camellia etc. questi sono tutti cifrari a chiave simmetrica (fino a un punto sono a blocchi e poi a flusso nella pagina wiki). DES non è più previsto nelle versioni più recenti.

Quindi quello che sta iniziando a emergere (in PGP non è riuscita a farcelo vedere mentre ora in TLS incomincia a farcelo vedere) è che **la mole di dati significativa viene cifrata sempre a chiave simmetrica** salvo situazioni molto particolari. La regola è che i cifrari usati per cifrare i dati che transitano, sono a chiave SIMMETRICA in tutti i protocolli di rete e non solo normalmente è così.

CIFRATURA A LIVELLO DI RETE: IPSEC

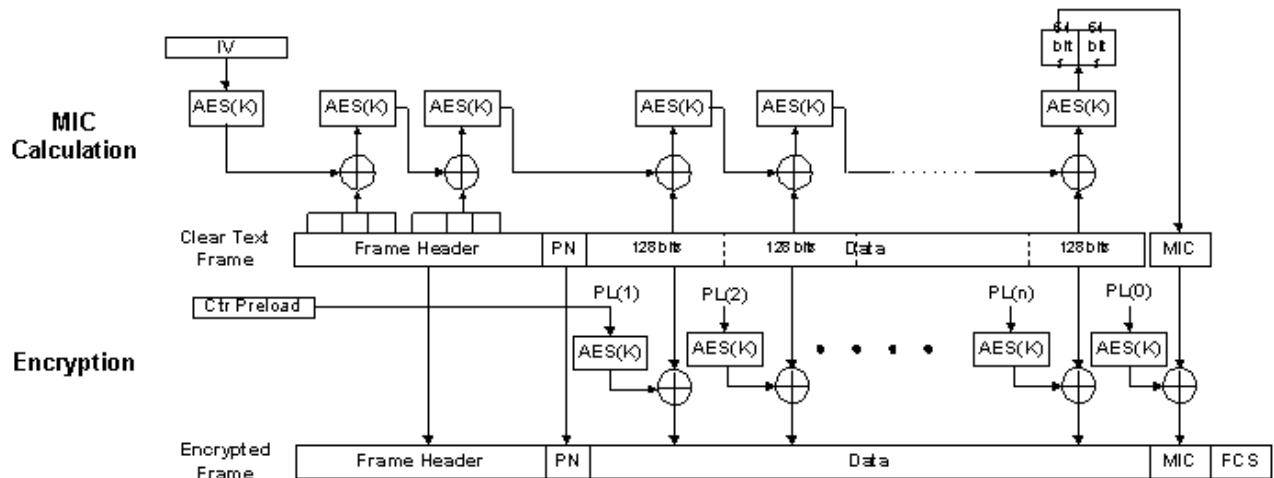
Abbiamo IPsec e algoritmi di cifratura IPsec.

Di IPsec ne abbiamo già parlato e per quanto riguarda gli algoritmi di cifratura è un po' difficile trovarne l'elenco però principalmente abbiamo AES e comunque cifrari a chiave simmetrica.



Ricordiamo che qui siamo a livello trasporto o subito sotto cioè un livello di rete dove viene reinserito l'header

CIFRATURA A LIVELLO DATA-LINK: WPA2



Abbiamo un frame in WPA2, qui il frame header ovviamente rimane in chiaro e i dati interni sono tutti cifrati in AES in modalità contatore (c'è anche la parte sopra per l'autenticazione). **Spera di averci chiarito che i cifrari sono utilizzati per la cifratura normalmente a chiave simmetrica.**

APPROFONDIMENTO DI DES

DIGRESSIONE SU DES

Ogni tanto si parla di DES a 40/56 Bit di chiave e poi a un certo punto DES non viene usato più.

DES abbiamo parlato di quello a 56 bit, è stato lo standard americano per tanti anni e a un certo punto è stato abbandonato.

Non esiste nel mondo della crittografia alcuna dimostrazione della robustezza dei cifrari ovvero tutti i cifrari vanno bene finché qualcuno non trova il modo di bucarli. Si può dimostrare che esiste un cifrario che è sicuro solo se P è diverso da NP.

Ricordiamo che l'attacco forza bruta si può sempre fare.

Quindi dovremmo dimostrare che l'attacco forza bruta è il miglior attacco e per farlo P dovrebbe essere diverso da NP.

Con DES nonostante i 30 anni di utilizzo e le attenzioni che ha avuto non sono state trovate delle vulnerabilità serie, quindi meccanismi per bucare DES che fossero diversi dalla forza bruta. I meccanismi sono generalmente dei grossi sistemi lineari con i dati intermedi dei vari round cercando di trovare correlazioni con testo cifrato e testo in chiaro, testo cifrato e la chiave etc. Con DES nessuno è mai riuscito a trovare risultati significativi quindi è rimasto robusto. A un certo punto è invecchiato a causa della lunghezza della chiave ,è diventata troppo corta.

Come si fa a sapere se la chiave è troppo corta?(Ricordiamo che des usava una chiave a 56 bit e non poteva cambiare la lunghezza)

attacchi (distribuiti) forza bruta a DES

- DES I (1997)
3 mesi e 4 giorni
- DES II (1998)
39 giorni (gennaio), 3 giorni (luglio)
- DES III
22 ore e 25 minuti (gennaio 1999)

Nel 97 la RSA ha lanciato una sfida mettendo a disposizione un testo cifrato con DES promettendo un premio a chi fosse riuscito a produrre per primo il testo in chiaro. Ci hanno messo 3 mesi e 4 giorni.

Nel 98 ,39 giorni a gennaio e 3 giorni a luglio.

Nel gennaio del '99, 22 ore e 25 minuti, quindi si incominciava a preoccuparsi.

Come hanno fatto a fare questi attacchi? Una volta lanciata la sfida, sono stati implementati algoritmi di decifratura per tutte le piattaforme possibili ed è stato distribuito a chi volesse utilizzare l'hardware della propria macchina per cercare di trovare la chiave o almeno per decifrarne qualcuna; è stato quindi un brute force distribuito in modo gigantesco. Dato che il tempo necessario al cracking stava calando in modo preoccupante attraverso un attacco a forza bruta si è pensato che DES non fosse più sicuro.

L'altra cosa successiva interessante sono attacchi a forza bruta con HW specializzato:

attacchi forza bruta (con HW specializzato) a DES

- 1980 (Diffie): \$ 50 milioni, 2 giorni
- 1993 (Wiener): \$1 milione, 3,5 ore
(in ambiente militare: 1988, all'85% del costo)
- 1997 (Diffie-Hellman): \$ 20 milioni, 12 ore
- 1998 (Kocher): \$ 130 000, 112 ore (costruito)

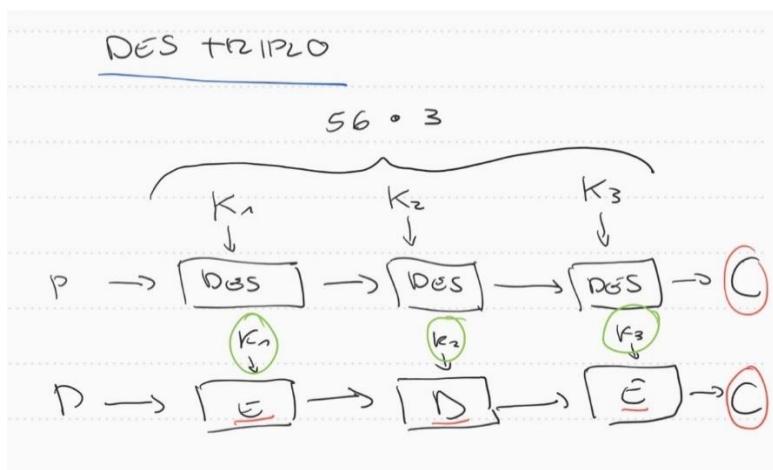
Altra ragione di allarme!!!

Diffie-Hellman ha detto che con 50 Milioni in 2 giorni ecc. tutti gli altri

Notiamo che non tutto è di dominio pubblico: si narra che dopo che Wiener ha pubblicato il suo sistema, egli è stato avvicinato da una persona in ambito militare il quale gli ha confidato che il suo tipo di sistema era già nella mani militari con un costo che andava dal 88 all'85 % di quello che aveva speso lui 5 anni dopo.

3DES (DES TRIPLO)

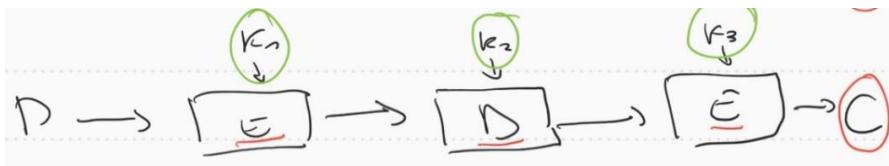
Come si fa a sostituire un cifrario? Viene indetta una gara (in tutto il mondo) per sostituirlo. La prima idea era usare **3DES** ovvero usare DES 3 volte di seguito con 3 chiavi diverse.



il testo in chiaro viene cifrato prima con la chiave 1 poi con la 2 e infine con la 3 e si ottiene infine il testo cifrato. La chiave quindi diventa 56×3 e questo significa che da un attacco a forza bruta ci si protegge molto bene in quanto bisogna trovare 3 chiavi, quindi la lunghezza della chiave diventa di 56×3 bit. Ovviamente questa è stata una soluzione molto veloce per riparare (work-around) però è molto lento come cifrario dato che deve cifrare 3 volte: ci mette 3 volte il tempo di DES.

3-DES STANDARDIZZATO

(che viene usato realmente)



Un altro modo (che è quella realmente standardizzata) è far passare il testo in chiaro prima in una cifratura con DES con chiave K1 poi uso l'algoritmo di decifratura sempre con DES con chiave K2(quindi con una chiave diversa da k1) e infine cifra nuovamente con K1 per ottenere il testo cifrato.

Ricordiamo che la cifratura e la decifratura con DES sono la stessa cosa salvo l'ordine delle chiavi, quindi quando decifra dato che usa una chiave diversa è come se cifrasse nuovamente.

Il vantaggio è che chi usa ancora DES e non 3DES, questo rimane ancora compatibile perché dove c'è K2 inserisce sempre K1 e l'algoritmo diventa analogo ed inoltre la lunghezza della chiave diventa $56*2$ e non $*3$.

È stato standardizzato così per renderlo compatibile per chi usava ancora DES.

Ovviamente è meno sicuro per chi usa ancora DES perché utilizza comunque una chiave da 56 e non da $56*2(k_1,k_2)$.

Sono 3 e non 2 poiché se no si potrebbe fare un attacco in cui si fanno due attacchi a forza bruta in cui si converge nel valore in mezzo e il costo degli attacchi non sarebbe dato da $2^{chiave1+chiave2}$ ma da 2^*2^{chiave} (questo è un commento per chi lo riceve se no lasciamo stare). US NIST che è un istituto americano ha indetto una gara nel 97 per proporre un cifrario a chiave simmetrica che diventerà il nuovo standard e viene scelto Rijndael con AES.

Storia

- necessario sostituire DES
- 3DES lento, blocchi corti
- US NIST ha indetto una gara nel 1997
- 15 candidati accettati nel giugno '98
- 5 selezionati nell'agosto '99
- Rijndael scelto come AES (ottobre 2000)
- standard FIPS PUB 197 nel nov 2001

Criteri di valutazione

➤ iniziali:

- sicurezza – sforzo per crittanalisi pratica
- costo – in termini di efficienza computazionale
- caratteristiche dell'algoritmo e dell'implementazione

➤ finali:

- sicurezza generale
- facilità di implementazione (SW e HW)
- attacchi all'implementazione
- flessibilità (cifratura, decifratura, chiavi,...)

ARINJNDAEL

AES è il nome dello standard (Advanced Encryption Standard) mentre Rijndael il nome del cifrario il quale poi è diventato appunto il nuovo AES.

leggo qui:(storia)

Rijndael

- progettata da Rijmen e Daemen in Belgio
- chiavi da 128/192/256 bit, blocchi da 128 bit
- cifrario **iterativo** (non **feistel**)
 - elabora i dati come blocchi di 4 colonne di 4 byte
 - opera su tutto il blocco di dati ad ogni round
- progettato secondo i criteri:
 - ottenere resistenza agli attacchi noti
 - ottenere codice veloce e compatto su molte CPU
 - semplicità di progettazione

Vita attiva di 20-30 significa molto più lunga per vita non attiva.

Una delle ragioni per cui è stato scelto AES perché ha buona implementazione su architetture molto limitate grazie alla struttura del cifrario.

DES 40 BIT:

Perché da 56 si è passati a 40 bit?

Per molti anni gli USA hanno avuto delle leggi per cui si rifiutavano di esportare cifrari con chiavi troppo lunghe, quindi cifrari (DES) per l'esportazione sicura di dati avevano al massimo 40 bit mentre cifrari per uso interno (DES) a 56 bit.

Quindi per esportare al massimo cifrari a 40 bit si intende quindi vendere la tecnologia al massimo da 40 bit in modo da non dare 'al nemico' la possibilità di conoscere la propria tecnologia.(per esempio permettere di scaricare un browser che poteva cifrare a 56 bit) Questo perché il nemico non doveva cifrare dei segreti con la tecnologia interna. Infatti nelle capacità dei browser vecchi [SCORRE WIKIPEDIA] si vede che c'erano delle versioni a 56 bit e 40 bit ed era concesso per l'esportazione negli stati uniti(quello da 40). E questa idea di militare di limitare l'esportazione non è stata del tutto abbandonata , gli stati uniti hanno smesso di fare questo mentre noi facciamo parte del Wassenaar Arrangement <https://www.wassenaar.org/> che è un accordo nazionale tra vari paesi che prevede di non esportare tecnologia bellica di una certa entità verso paesi etichettati "nemici" e tra le tecnologie belliche ci sono anche i cifrari.

Per esempio java viene offerto in modo standard con utilizzo di chiavi al massimo di 128 bit.

Le ragioni sono di vario genere nel senso che in parte sono protezioni contro il nemico, in parte che se java deve essere esportato dappertutto deve fare i conti con certi paesi che non vogliono che la gente possa cifrare oltre un certo limite di sicurezza, ovvero che non possano comunicare anonimamente tra di loro quindi limitazioni della libertà del cittadino.

[why there are limitations on using encryption with keys beyond certain length]
<https://crypto.stackexchange.com/questions/20524/why-there-are-limitations-on-using-encryption-with-keys-beyond-certain-length>

Why are there limitations on using encryption with keys beyond certain length?

[Ask Question](#)

21 I am writing a java program to encrypt a message using 256-bit AES encryption, but I am getting `illegal key size error`, I have read that I have to use some JCE Unlimited Strength Jurisdiction Policy Files to encrypt/decrypt message with key of 256-bit or longer, but I do not understand why there is a limit on using a key size? Why using 256-bit encryption is disabled in Java by default?

Is it a crime to encrypt/decrypt files using 256-bit keys?

There are many different cryptography laws in different nations. Some countries prohibit export of cryptography software and/or encryption algorithms or cryptoanalysis methods.

In some countries a license is required to use encryption software, and a few countries ban citizens from encrypting their internet communication. Some countries require decryption keys to be recoverable in case of a police investigation.

Issues regarding cryptography law fall into four categories:

- **Export control**, which is the restriction on export of cryptography methods within a country to other countries or commercial entities. There are international export control agreements, the main one being the [Wassenaar Arrangement](#). The Wassenaar Arrangement was created after the dissolution of [COCOM](#) (Coordinating committee for Multilateral Export Controls), which in 1989 "decontrolled password and authentication-only cryptography."
- **Import controls**, which is the restriction on using certain types of cryptography within a country.
- **Patent issues**, which deal with the use of cryptography tools that are patented.
- **Search and seizure issues**, on whether and under what circumstances, a person can be compelled to decrypt data files or reveal an encryption key.

Restrictions on the import of cryptography

Countries may wish to restrict import of cryptography technologies for a number of reasons:

- Cryptography may increase levels of privacy within the country beyond what the government wishes.
- Citizens can anonymously communicate with each other, preventing any external party from monitoring them.
- Encrypted transactions may impede external entities to control the conducting of business.
- Imported cryptography may have backdoors or security holes, intentional or not, which allow foreigners to spy on persons using the imported cryptography; therefore the use of cryptography is restricted to that which the government thinks is safe, or which it develops itself.

The [Electronic Privacy Information Center](#) and Global Internet Liberty Campaign reports use a color code to indicate the level of restriction, with the following meanings:

- **Green**: No restriction
- **Yellow**: License required for importation
- **Red**: Total ban

Here is [List of Countries](#), each with their level of import restrictions.

Country	Status	Updated	Country	Status	Updated	Country	Status	Updated
Angola	Unknown	2000	Latvia	Yellow	2008	South Korea	Yellow	2008
Armenia	Green/Yellow	2000	Lithuania	Yellow	2008	Tatarstan	Unknown	2000
Bahrain	Yellow	2008	Malta	Yellow	2000	Tunisia	Yellow/Red	2008
Belarus	Red	2008	Moldova	Yellow	2008	Turkmenistan	Red	2000
Brunei Darussalam	Yellow/Red	2000	Mongolia	Red	2000	Ukraine	Yellow	2007
Cambodia	Yellow	2008	Morocco	Yellow	2008	Uzbekistan	Red	2000
Czech Republic	Green/Yellow	2008	Myanmar (Burma)	Red	2008	Vietnam	Yellow	2008
China	Yellow	2008	Nepal	Unknown	2000			
Egypt	Yellow	2007	Nicaragua	Unknown	2000			
Ghana	Green	2008	North Korea	Unknown/Red	2008			
Hong Kong	Green/Yellow	2008	Pakistan	Yellow	2008			
Hungary	Green/Yellow	2008	Poland	Green/Yellow	2008			
India	Green/Yellow	2008	Russia	Red	2008			
Iran	Yellow	2008	Rwanda	Unknown	2008			
Iraq	Red	2000	Saudi Arabia	Green	2008			
Israel	Yellow	2008	Singapore	Green	2008			
Khazakstan	Yellow	2008	South Africa	Green/Yellow	2008			

and [List of Countries with their import/export restrictions details and explanation](#)

Import Limits on Cryptographic Algorithms in Java

Due to import regulations in some countries, the Oracle implementation provides a default cryptographic jurisdiction policy file that limits the strength of cryptographic algorithms.

If stronger algorithms are needed (for example, AES with 256-bit keys), the [JCE Unlimited Strength Jurisdiction Policy Files](#) must be obtained and installed in the JDK/JRE.

It is the user's responsibility to verify that this action is permissible under local regulations.

Liste di paesi di cui si diffida. Quindi in verde non ci sono restrizioni sulla crittografia.

giallo licenza richiesta.

rosso non se ne parla proprio.

Anche l'Italia fa parte di questo patto Wassenaar Arrangement e varie regole..

Quindi ci sono anche problemi politici, che poi calpestano i piedi al nostro lavoro da informatici.

La volta scorsa abbiamo implementato AES e abbiamo definito e parlato dei cifrari a chiave simmetrica. In particolare, ci ha fatto vedere che **in tutti i protocolli di rete e in qualunque meccanismo, per cifrare i dati si usano i cifrari a chiave simmetrica.**

La cosa che si può osservare è come sono fatti AES e RC4: tutti e due i meccanismi non sono basati su nozioni matematiche ma sono una manipolazione di bit (shift, XOR etc.) ed effettuano operazioni riassumibili in, appunto, manipolazione di bit.

Le operazioni implementate sono operazioni molto a basso livello, quindi gli algoritmi a chiave simmetrica sono molto veloci da implementare e economici per quanto riguarda l'hardware (infatti sono orientate prima in HW che in SW). In generale, i dati si cifrano a chiave simmetrica in quanto le operazioni di cifratura sono molto semplici e utilizzano algoritmi veloci ed efficienti.

C'è però un problema grosso con questi cifrari e si vede in un protocollo di comunicazione piuttosto che in locale. Dall'altra parte deve essere nota la stessa **chiave simmetrica che deve essere scambiata in modo sicuro** (ovvero come scambio la chiave segreta su un canale non protetto?). Ne aveva accennato anche in ONE-TIME PAD come nell'esempio di comunicazione tra Casa Bianca e Cremlino.

Se si vuole utilizzare la crittografia per proteggere una carta di credito per un acquisto online, è abbastanza evidente che l'utente non può andare lì a vedere la chiave di Amazon. **È importante trovare un modo per scambiare le chiavi segretamente.**

Se si riesce a scambiare con successo una chiave simmetrica, allora vuol dire che è stato creato un canale sicuro attraverso il quale trasferire i dati, **però come scambiamo questa chiave?**

Non se ne esce da questo e **viene in aiuto la crittografia a chiave pubblica**. Il fatto che dica a chiave pubblica non significa che sia tutto pubblico ma **si usa una coppia di chiavi (K_{pub} , K_{priv}) dove una è pubblica (K_{pub}) e una viene mantenuta privata (K_{priv}) legate in modo indissolubile** (si utilizzano sempre in coppia).

La necessità di avere una parte privata è data dal fatto che se si vuole utilizzare un algoritmo che possa essere utilizzato dall'utente legittimo e non dall'attaccante, deve esserci qualche elemento che l'attaccante non ha in mano e che solo gli utenti legittimi conoscono e possono usare: un **segreto condiviso**.

In un sistema crittografico in cui non entrano dati segreti questo non può proteggere niente da nessuno perché se non c'è niente di segreto, quello che possono fare gli utenti legittimi lo possono fare anche gli attaccanti.

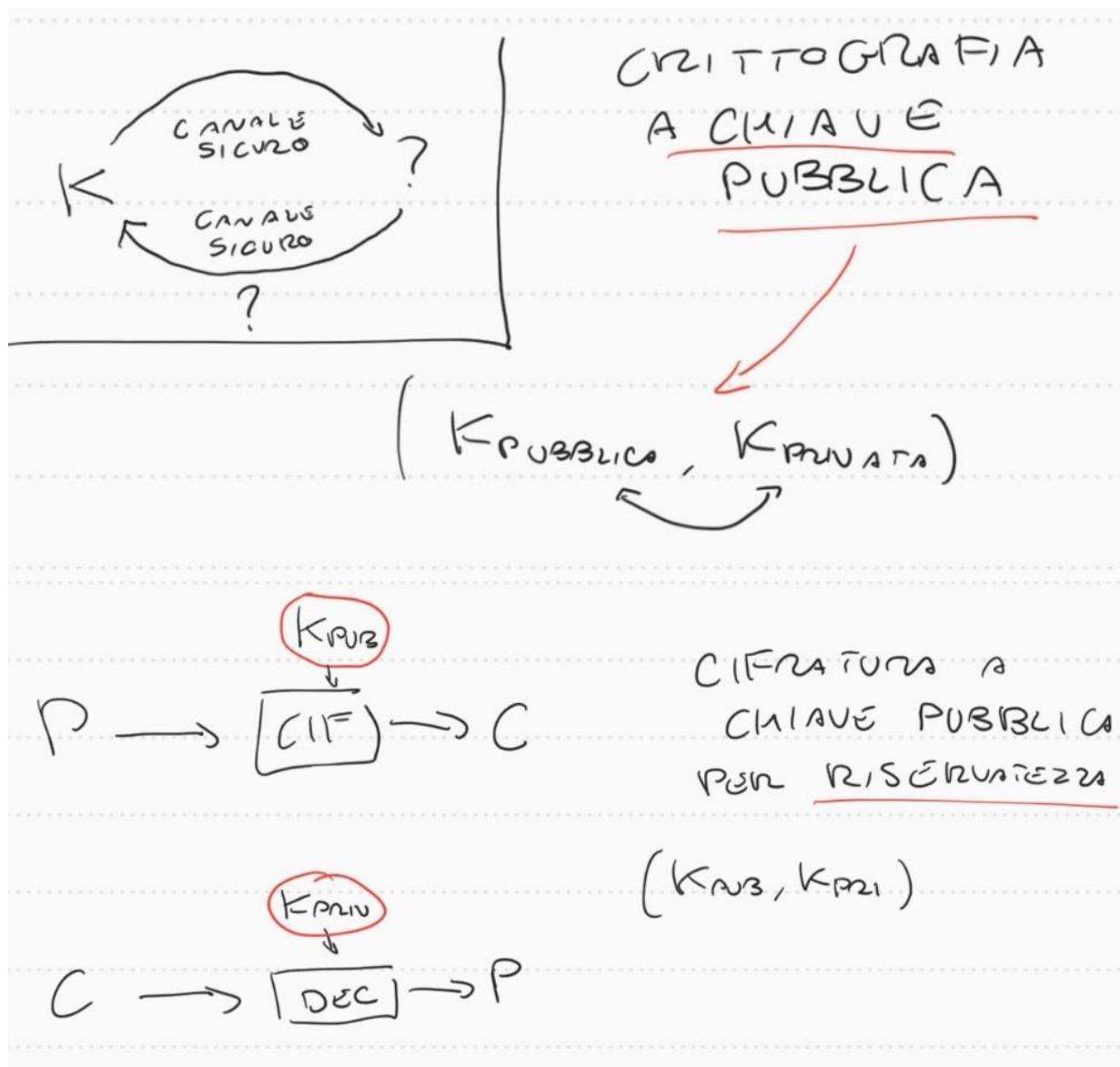
Se nel sistema non mi è chiara la parte segreta c'è qualcosa che non quadra. La chiave pubblica e privata a seconda del sistema varia → Stiamo parlando di cifratura e stiamo parlando di riservatezza.

RISERVATEZZA

Se vogliamo la riservatezza è importante che solo l'utente legittimo ricevendo un messaggio privato sia in grado di decifrarlo. **Il solo soggetto inteso può venire a conoscenza dei dati.** Quindi se C è un testo cifrato a chiave pubblica, quello che vogliamo è che **solo il soggetto interessato sia in grado di decifrarlo**, quindi significa che **la decifratura deve essere legata alla chiave privata!**

In un cfrario a chiave pubblica **per la riservatezza si cifra con la chiave pubblica e si decifra con la chiave privata.**

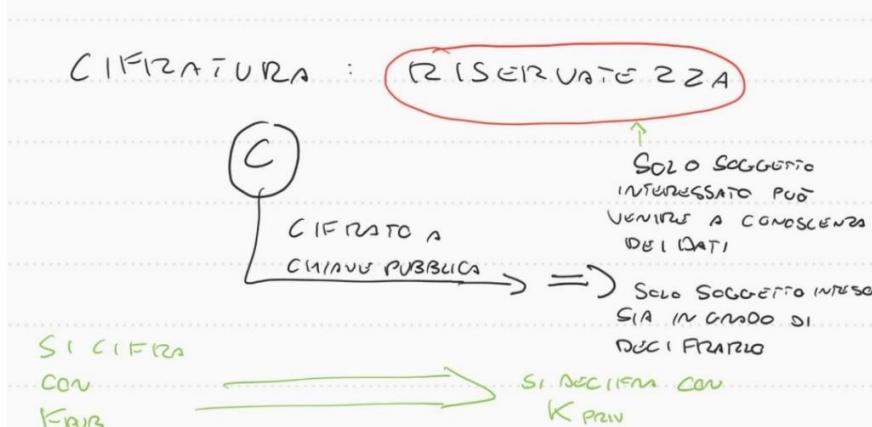
Stiamo ragionando sulla riservatezza. Le chiavi sono legate in questo modo. L'operazione che si fa con una delle due chiavi viene invertita con quella che si fa con l'altra.



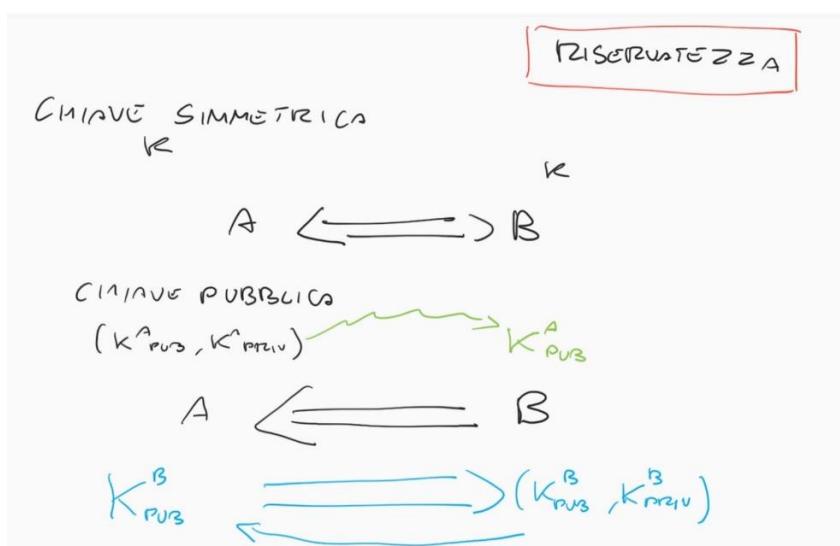
CIFRARIO A CHIAVE PUBBLICA PER LA RISERVATEZZA

Una cifratura con chiave pubblica dà luogo a un testo cifrato.

Un testo cifrato con chiave pubblica dà luogo al testo in chiaro tramite decifratura



Se si decifra il testo cifrato con una certa chiave privata che non è la stessa appartenente alla coppia (K_{pub}, K_{priv}) con la quale si è cifrato, non ritornerà mai il testo in chiaro.



Cosa succede a questo punto?

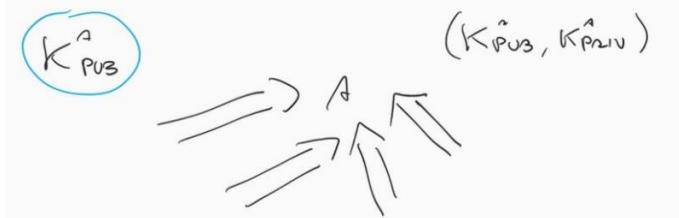
rivediamo la chiave simmetrica da una parte e chiave pubblica dall'altra. (Sempre nel caso della riservatezza)

Se A e B conoscono la stessa chiave simmetrica, allora hanno stabilito un canale bidirezionale privato utilizzando quella chiave simmetrica (A può cifrare e B decifrare o viceversa ma l'importante è che abbiamo la stessa chiave).

Nel caso invece della chiave pubblica, se A possiede una coppia (pubblica, privata), la chiave pubblica (per sua natura di chiave pubblica) si può rendere visibile a tutti e A può mandarla a B senza doverla mantenere segreta, perciò se l'attaccante dovesse conoscere la K_{pub} non comporta problemi di sicurezza per il cifrario. Però il canale di comunicazione va solo da B ad A poiché solo A possiede la propria chiave privata, ma non possiamo mandarla sulla rete a B altrimenti avremmo lo stesso problema che avevamo nello scambiarsi la chiave simmetrica, e inoltre A non può rispondere con un messaggio cifrato a chiave pubblica (se conosce la chiave pubblica l'attaccante lo potrebbe decifrare normalmente). Se anche B si dota di una chiave pubblica, la chiave pubblica di B può essere resa nota e questo definisce un canale bidirezionale tra A e B.

Per avere un canale bidirezionale usando algoritmi a chiave pubblica bisogna fare in modo che A e B si scambino le proprie chiavi pubbliche: in questo modo A può cifrare i messaggi con la chiave pubblica di B e viceversa, ma nessuno può decifrare quei messaggi senza avere la chiave privata di B (o A se da B mandiamo mess. cifrati con chiave pubblica di A).

Se A ha una coppia (K_{pub}^A , K_{priv}^A), non definisce solo un canale da B ad A ma definisce un numero arbitrario di canali da tutto il mondo verso A in modo unidirezionale, proprio perché chiunque può inviare messaggi cifrati ad A se conosce la sua chiave pubblica. In sintesi, **tutti possono cifrare messaggi per A usando la sua chiave pubblica, e solo A potrà decifrare quei messaggi è l'unica macchina a possedere la sua chiave privata K_{priv}^A** .



Posso cifrare per A e mandare messaggi ad A poiché A mantiene la chiave corrispondente. Se io pubblico la mia chiave pubblica sul mio sito chiunque può scrivermi un messaggio cifrato senza che io abbia mai avuto contatti con quello. Mentre a chiave simmetrica è bidirezionale ma prevede uno scambio a priori e solo tra i soggetti interessati della stessa chiave simmetrica. Quindi usare algoritmi a chiave pubblica risolve il problema della circolarità per lo scambio della chiave simmetrica.

Avere una buona chiave pubblica vuol essenzialmente dire che per un attaccante che conosce la chiave pubblica è difficile risalire alla chiave privata corrispondente. Anche se la chiave pubblica è riconosciuta in tutto il mondo è possibile mandare messaggi privati verso A ma nessuno (con risorse accettabili) può risalire alla chiave privata di A e quindi decifrare i messaggi. Notare che per trovare una chiave pubblica si possono provare tutte le stringhe private corrispondenti finché non si trova quella corretta, quindi è necessario che le stringhe siano sufficientemente lunghe affinché l'attacco brute force sia improponibile. Questo discorso non è detto che basti, perché è vero che l'attacco brute force è possibile, ma naturalmente in base al legame tra chiave pubblica e privata è possibile che ci si possa effettuare un attacco più facile della brute force e quindi la lunghezza della chiave deve essere abbastanza ampia per ovviare questa problematica.

DOMANDA Daniele ladro gambera

Se la mia macchina ha distribuito la mia chiave pubblica sulla rete, se una qualsiasi macchina mi manda un messaggio cifrato con la mia chiave pubblica, **solo io posso decifrare quel messaggio in quanto solo io conosco la chiave privata associata a quella pubblica che ho inviato sulla rete**. Quindi, se un attaccante dovesse intercettare il messaggio, questo sarebbe cifrato e non potrebbe leggerlo non avendo a disposizione la chiave privata. **Chiunque voglia decifrare un messaggio con una certa chiave pubblica deve avere la corrispondente chiave privata.**

UTILIZZO DELLE CHIAVI PUBBLICHE

I dati cifrati a chiave simmetrica vengono poi cifrati con la chiave pubblica del destinatario: perciò **le chiavi pubbliche sono utilizzate per scambiare la chiave simmetrica**. Se ho una coppia di chiavi per la cifratura, se uno vuole scrivere e vuole ottenere poi una risposta (quindi creare un canale bidirezionale), cifrerà a chiave pubblica una stringa di dati cifrati con una chiave asimmetrica. L'altra macchina risponderà cifrando il messaggio con DES o AES utilizzando quella chiave simmetrica.

Questo si chiama **utilizzo ibrido della crittografia**: ibrido sta nel fatto che **si usa sia crittografia a chiave pubblica** (scambiarsi la chiave simmetrica in riservatezza) **sia quella simmetrica** (per cifrare i dati in modo sicuro). (In questa parte è stata chiara come il mio pisello nella nebbia vercellese di notte).

CLASSIFICAZIONE DEI CIFRARI IN BASE AGLI ATTACCHI A CUI SONO VULNERABILI

Classificazione vulnerabilità dei cifrari:

un cifrario può essere vulnerabile a diversi attacchi.

VULNERABILITÀ DEI CIFRARI

- ATTACCO CIPHER TEXT ONLY C_i

- ATTACCO KNOWN PLAINTEXT (P_i, C_i)

- ATTACCO CHOSEN PLAINTEXT
SCEGLIE P_i OTTIENE C_i

- ATTACCO CHOSEN CIPHERTEXT
SCEGLIE C_i OTTIENE P_i

ATTACCO CIPHER-TEXT ONLY

Attacco cipher-text only (C_i)

La situazione più comune è che l'attaccante conosca solo il testo cifrato (uno o più testi) C_i . Questa è la situazione più semplice poiché una volta cifrato, il testo è pubblico e facilmente recuperabile: questo tipo di attacco si chiama **attacco cipher-text only**, ed è tipicamente un attacco a forza bruta o un attacco che si basa sulla frequenza dei caratteri.

Dipende fortemente dal cifrario: Un cifrario che ha buone proprietà di diffusione sarà meno soggetto ad attacchi che si basano sulla frequenza dei caratteri del testo cifrato. Un cifrario basato sulla sostituzione alfabetica ad esempio è vulnerabile ad attacco di tipo cipher-text only, mentre per AES l'unico attacco possibile conoscendo il testo cifrato è quello a forza bruta (per quanto ha spiegato fino ad ora).

ATTACCO KNOWN PLAINTEXT

Attacco Known Plaintext: l'attaccante ha una o più coppie (P_i, C_i)

Testo in chiaro, testo cifrato (possono essere più di uno solo, per questo P_i).

L'attaccante ha accesso al testo in chiaro e a quello cifrato, permettendogli di rivelare informazioni segrete come le chiavi. Es. l'Attacco Mario Rossi è un attacco di questo genere: dalla struttura del testo in chiaro comprendo che se ho il template del modulo e so come va compilato, posso sostituirmi a Mario Rossi e fare un casino. Questo grazie al fatto che conosco la coppia testo in chiaro e cifrato. Stiamo facendo ipotesi più forti nel senso che l'attaccante deve possedere questa coppia ed è difficile da ottenere.

ATTACCO CHOSEN PLAINTEXT

Attacco Chosen Plaintext (Sceglie P_i e ottiene C_i)

Attaccante sceglie uno o più testi in chiaro P_i tra i disponibili e ottiene il corrispondente testo cifrato C_i .

ATTACCO CHOSEN CIPHERTEXT

Attacco Chosen Ciphertext (Sceglie C_i e ottiene P_i)

Attaccante può scegliere uno o più testi cifrati C_i e ottenere i corrispondenti testi in chiaro P_i . Da questo può costruire un meccanismo per ottenere altri testi cifrati se riesce a trovare la correlazione tra P_i e C_i .

PROBLEMA DELLO SCAMBIO DELLE CHIAVI ASIMMETRICHE

Non sono le uniche categorie. Questo è un modo per classificare i cifrari in base alle debolezze ma ci sono altre classificazioni ma è tutto un altro mondo e non li tratteremo.

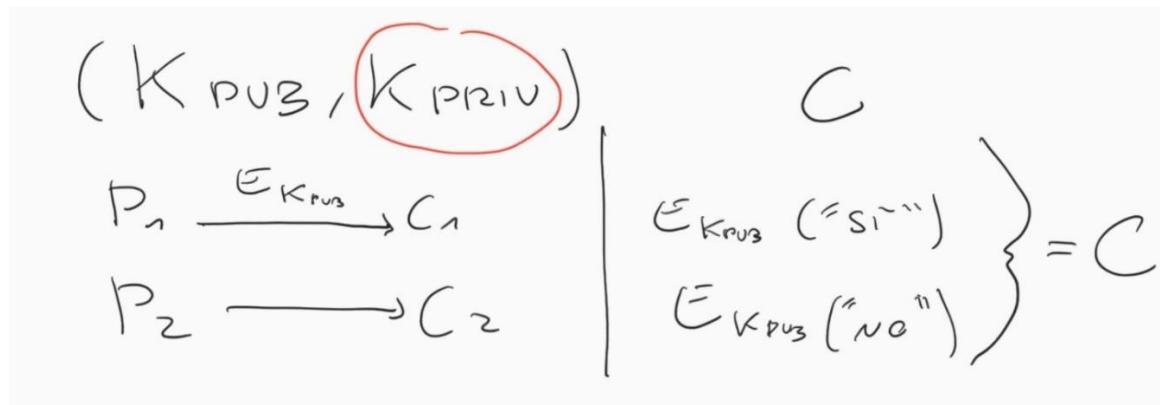
Qualunque cifrario deve essere **sempre** resistente ad un attacco cipher text only. In certi contesti potrebbe essere utilizzabile anche il Plaintext.

Concentriamoci però su attacco Chosen Plaintext: **se pensiamo ad un cifrario a chiave pubblica**, è banale che l'attaccante conosca il testo cifrato, siccome se la chiave pubblica è nota **chiunque può ottenere tutti i testi in chiaro possibili e ottenere il testo cifrato corrispondente** poiché può cifrare quello che vuole. Un attaccante con chiave simmetrica non può fare questo attacco, in quanto non può decifrare i dati cifrati senza avere la chiave.

Questo porta ad una vulnerabilità di cui poi parleremo.

Supponiamo di avere A, B e T con T = attaccante, e che il testo in chiaro sia in un ambito ristretto, ovvero il testo in chiaro è una scelta tra 20 possibili testi in chiaro poiché A sta mandando qualcosa che sia B che T conoscono, per esempio quello che viene spedito è una risposta SI o NO (quindi T sa che il testo in chiaro è SI o NO e conosce quello cifrato).

L'attaccante conoscerà il testo cifrato che contiene SI oppure NO, e da qui cifra SI e NO con la chiave pubblica del ricevente, ottenendo due nuovi testi cifrati nello stesso modo di quelli inviati da A che ha intercettato. Dopo aver confrontato i testi cifrati di A con quelli cifrati da lui, troverà una correlazione in quanto saprà che SI avrà un certo testo cifrato, e NO un altro. Se tutti questi sono i testi in chiaro possibili uno di questi due sarà C. Quindi **un testo cifrato con chiave pubblica è vulnerabile a un attacco di tipo Chosen Plaintext, proprio perché per natura, in un contesto a chiave pubblica, chiunque può cifrare qualunque cosa con quella chiave e quindi conoscere il testo che ha cifrato.**



ALGORITMO RSA

Rivest, Shamir, Adelmann sono gli inventori dell'algoritmo RSA. Rivest lo ricordiamo già da RC4 e hanno inventato questo algoritmo. Qual è l'idea di RSA? È un algoritmo di cifratura a chiave pubblica.

Funzionamento matematico per introdurci l'algoritmo:

ALGORITMI DI CIFRATURA A CHIAVE PUBBLICA

p, q primi (segreti) $A \rightsquigarrow A$
 $n = p \cdot q$ $e : \text{MCD}(e, (p-1)(q-1)) = 1$
 $d : e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$
 $[e \cdot d = q(p-1) \cdot (q-1) + 1]$

$K_{\text{pub}} = (e, n)$ $K_{\text{priv}} = (d, n)$

$m^e \pmod{n} = c$
 $c^d \pmod{n} = m$

1. Vengono introdotti due numeri primi p, q che vengono mantenuti segreti.
2. Li sceglie A che vuole generare la sua coppia di chiavi pubblica e privata.
3. Li moltiplica ottenendo $n = p \cdot q$. Dopodiché A sceglie un 'e' che è co-primo (cioè senza fattori in comune) con $(p-1)*(q-1)$, ovvero $e \rightarrow \text{MCD}(e, (p-1)(q-1))=1$
4. A partire da questo 'e' sceglie una 'd' tale che $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$

Si può anche scrivere che: $e \cdot d = h * (p-1)(q-1) + 1$ (dove h è un certo multiplo)
Oppure si sceglie d in modo che $e \cdot (d-1)$ sia divisibile per $(p-1)(q-1)$

$a \pmod{b} \equiv c \rightarrow a \equiv c \pmod{b}$

$a = q \cdot b + c$ dove q è un certo quoziente

Se dico $37 \pmod{10} = 7$

$37 \pmod{10}$ è congruo a $47 \pmod{10}$. Quindi hanno lo stesso resto.

La chiave pubblica è $K_{\text{pub}} = (e, n)$ e la $K_{\text{priv}} = (d, n)$.

Se qualcuno vuole cifrare un messaggio m , prende m e lo eleva ad 'e', lo calcola in modulo n ottiene il testo cifrato c

$m^e \pmod{n} = c$

$c^d \pmod{n} = m$

Dove m è il messaggio che è un numero.

Quando ha un inverso moltiplicativo? quando è diverso da 0 ??? ho perso ????

Il modo di cifrare e decifrare è questo quindi

$m^e \pmod{n} = c$

$c^d \pmod{n} = m$

$$\begin{aligned}
 m^e \bmod n &= c \\
 c^d \bmod n &= l < n \\
 &= m \bmod n = ?m
 \end{aligned}$$

$$0 \leq m \leq n$$

Cosa succede? cosa vogliono dire tutte queste cose e perché.
Vediamo alcune cose fondamentali

$$m^e \bmod n = c$$

$$c^d \bmod n = l$$

l è il resto della divisione per n quindi è necessariamente più piccolo di n . Allora $l < n$ quindi se vogliamo che questo valore sia **uguale** a m bisogna partire da un valore che è minore di n quindi: $0 \leq m \leq n$.

Questo l è quindi un vincolo sulla lunghezza del messaggio che si può cifrare con RSA.

Se il messaggio cifrato è più grande di n non si troverà m .

Quando $m \bmod n = m$? quando $m < n$

Affinché un cifrario sia utilizzabile, deve essere efficiente **calcolarne i parametri e quindi determinare le chiavi** (non è necessario che sia efficientissimo poiché le chiavi si calcolano una volta ogni tanto). Deve essere efficiente cifrare e decifrare e deve essere difficile per l'attaccante risalire:

1. al testo in chiaro dato il testo cifrato
2. dal testo cifrato alla chiave
3. dalla chiave pubblica a quella privata.

I parametri: **per generare i primi p e q la generazione avviene con una generazione di un numero in pseudo casuale e si verifica con un algoritmo polinomiale deterministico** quelli noti fino ad ora sono troppo pesanti quindi **si fa ricorso ad algoritmi probabilistici** come il test di Miller Rabin (*non si approfondisce questo aspetto*). E' un test che fa varie verifiche finché non arriva a trovare un controesempio. Dopodichè si moltiplicano p e q e si ottiene n .

Si sceglie una '**e**' **casuale e si verifica che questo cosa sia vera e si usa algoritmo di euclide** (che è polinomiale).

Come si fa a determinare d ? E' un'estensione dell'algoritmo di euclide per il MCD e permette anche di trovare ' d ' in questo contesto.

$$\begin{aligned}
 &= 0 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 \\
 &\quad + 1 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 \quad \text{ris=1} \\
 &\quad \overbrace{\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad}^{\substack{a^4 \cdot a^2 \\ (a^2)^2 \cdot (a^{(2)})^2}} \quad \left\{ \begin{array}{l} l = a \\ l = a^2 \\ l = a^4 \\ l = a^8 \\ l = a^{16} \end{array} \right.
 \end{aligned}$$

Elevamento a potenza in modulo: si fa con l'algoritmo dei quadrati ripetuti:

Se uno vuole fare $a^b \bmod n$ scrive **b** in binario quindi $b = 10110$

quindi $a^{10110} \bmod n$

$$a^{10110} \rightarrow 0 * 2^0 + 1 * 2^1 + 1 * 2^2 + 0 * 2^3 + 1 * 2^4$$

$$a^2 * a^4 * a^{16}$$

L'algoritmo determina prima a^2 e guarda l'espansione binaria di b e cosa fa ?

$$a^2 \cdot (a^2)^2 \cdot (a^4)^2$$

$$l = a^2$$

$$l = l^2$$

dopodiché guarda i bit

$$l = a$$

$$\text{poi diventa } l = l^2$$

dove l è una variabile temporanea.

Spiegazione Caggese:

Prende il valore di b in binario, lo scomponete come potenze binarie e poi partendo da ris=1 che è l'elemento neutro della moltiplicazione, ad ogni iterazione di quella somma parziale è 1 (es. $1 * 2^2$), viene moltiplicato il risultato per quel valore fino ad arrivare alla fine della sequenza.

L'algoritmo è guardare tutti i bit e ad ogni iterazione mantenere il valore di l che cambia in base ai bit e moltiplicando il risultato parziale solo quando il bit corrispondente è uno. Poi quello successivo non lo moltiplico

$$\begin{array}{r} \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 \\ \hline 1^3 + 1 \cdot 2^4 \end{array}$$

$$\left\{ \begin{array}{l} l = a \text{ se bit=1} \\ l = a^2 \text{ se bit=1} \\ l = a^4 \\ l = a^8 \\ l = a^{16} \end{array} \right.$$

ris = n.s. l mod n
 ris = ris * l mod n
 |
 |
 |
 |

risultato inizialmente = 1 e se il bit è a 1 si moltiplica ris * l.

NON CE LO CHIEDE ma è per farci vedere che **l'elevamento a potenza in modulo n consente di avere sempre valori contenuti in n**. Si può fare perché le operazioni in modulo dipendono solo dal modulo degli argomenti. Il modulo conviene farlo sempre prima per gestire valori sempre più piccoli che hanno al massimo lunghezza di n.

Il numero di cicli dipende dai bit dell'esponente che non sono mai maggiori di n. Questo si può dedurre dal $\text{MCD}(e, (p-1)*(q-1)) = 1$

Quindi cosa succede? **il numero di iterazioni è al massimo la lunghezza di n e poi c'è anche una moltiplicazione che sono al massimo n in più ci sono n moduli di n quindi il costo è dell'ordine di O(n³)**.

Questo algoritmo è costoso poiché il costo è cubico ma si può rendere più efficienti le operazioni di RSA ma in ogni caso sono costose. È polinomiale e c'è un modo per farlo furbo ma ci dice che è costoso. **Confrontando con i cifrari a chiave simmetrica che fanno manipolazioni di bit, qui stiamo facendo operazioni matematiche complesse più costose dal punto di vista computazionale.**

L'attaccante che ha in mano la chiave pubblica cosa può fare per risalire alla chiave privata?

1. **Può provare tutti i possibili d finché non ottiene m (costo $2^{|n|}$)**
 2. **Può provare a fattorizzare n (costo $e^{\sqrt[3]{|n|}}$)**
1. Sicuramente può fare un attacco forza bruta: vede **c, sa l'algoritmo e conosce m in linea di massima**, allora prova tutti i possibili 'd' finché non ottiene una 'm' che ha senso per lui. Se lo spazio dei messaggi possibili ha ampiezza massima sul messaggio in chiaro, questo è equivalente ad un attacco a forza bruta sulla chiave. Quindi 2^n è il costo di tutti questi attacchi a forza bruta.
 2. Non si limita a questo caso la capacità dell'attaccante, ma **l'attaccante può fare di meglio: attaccante conosce 'n' ed 'e' che sono legati a 'd' da una relazione matematica molto precisa** mentre in un cifrario a chiave simmetrica non c'era nessuna relazione. **Se l'attaccante conoscesse [p-1]*[q-1] potrebbe fare la stessa operazione dell'utente e ottenere d con un costo polinomiale. Non è facile conoscere p-1 e q-1 a partire da n.** Per conoscerli è necessario conoscere p e q. Quindi il punto è che **se l'attaccante riesce a fattorizzare n, l'attaccante riesce a calcolare d in modo efficiente.**

DATO n DETERMINARE p e q
TALE CHE $p \cdot q = n$



Ora il problema è:

Dato n determinare p e q tale per cui $p \cdot q = n$ è come calcolare la **fattorizzazione** di n poiché sono due numeri primi.

Fattorizzazione : determinare numeri primi che lo compongono.

Quando costa fattorizzare n ?

$O(2^{\sqrt{|n|}})$ -Costo sub-esponenziale!

Mentre un attacco a forza bruta $2^{\sqrt{n}}$

La fattorizzazione è quindi **sub esponenziale** (perché l'esponente è una radice di n) mentre Forza bruta è esponenziale. Quindi l'attaccante tenterà sempre di fattorizzare n se possibile e non farà mai un attacco a forza bruta, in quanto costa meno.

La soluzione per difendersi da questo tipo di attacco è utilizzare p e q molto alti.

CHIAVI SIMMETRICHE : 128, 192, 256 bit

CHIAVI RSA : 1024, 2048, 4096 bit

Mentre la lunghezza delle chiavi simmetriche ha lunghezza 128, 192, 256 bit, una chiave RSA vanno da 1024, 2048, 4096 bit ed è colpa della fattorizzazione.

Sono chiavi circa dieci volte più grandi: implica che se **cifratura e decifratura sono pesanti** e se dovessimo pensare di usare delle chiavi più lunghe questo **incide molto di più sulla complessità quindi i cifrari a chiave pubblica sono pesanti**. Da una parte risolvono il problema della chiave simmetrica ma dall'altra sono pesanti.

I **Cifrari a chiave pubblica sono pesanti ed è il motivo per cui si utilizzano cifrari ibridi**.

Prossima volta tratteremo la lunghezza del testo in chiaro per RSA.

RSA CONTINUA

La lunghezza del testo in chiaro in RSA è limitata in modulo. Anche un testo della lunghezza esatta della chiave non viene cifrato con RSA perché dice che è troppo lungo.
Stranamente un'altra cosa di cui abbiamo parlato è il fatto che con un cifrario a chiave pubblica si può sempre scegliere il testo in chiaro e ottenere il testo cifrato corrispondente. Queste due cose sembrano non essere correlate ma in realtà lo sono. Prima di tutto dovremmo riflettere su una cosa: quando si cifra con un cifrario a blocchi il blocco deve essere di quella lunghezza. Però è un problema il fatto che sia una lunghezza del blocco?? NO! c'è il padding(quindi nel caso di des e in generale dei cifrari a blocchi che abbiamo visto fino ad ora, non abbiamo problemi).

Mentre con DES voglio avere un messaggio lungo con RSA ho una lunghezza massima da rispettare altrimenti si otterrà un messaggio di errore. Quando m è più corto di n, RSA non crea un problema e troverà un modo per allungarla mentre se è più lunga non ci sta dentro.

Perché con RSA non è definito un sistema per aumentare il massimo di un blocco?

Perché con RSA non è previsto che vengano cambiati gli standard siccome i **cifrari a chiave pubblica sono troppo pesanti non è previsto** dalle implementazioni che si **decifri un messaggio più lungo** e non è nella testa di nessuno che ci si metta a decifrare messaggi più lunghi proprio per il costo. Se il messaggio è più lungo non si può cifrare con questa modalità e il sistema dà errore.

Se cerco di cifrare con RSA un testo troppo lungo il sistema mi dice "data too large for key size

```
openssl pkeyutl -encrypt -inkey chiave.pem -in prova  
openssl des -in prova -out prova.cif
```

In genere si usa una tecnica ibrida: **si cifra con RSA la chiave asimmetrica, la si manda al destinatario con cui vogliamo trasmettere i dati e non appena questa arriva si inizia a cifrare la comunicazione (il flusso di dati) con la chiave simmetrica.**

Abbiamo visto che la lunghezza del messaggio deve essere < del modulo n perché queste sono trattate come stringhe di bit anche se sono numeri.

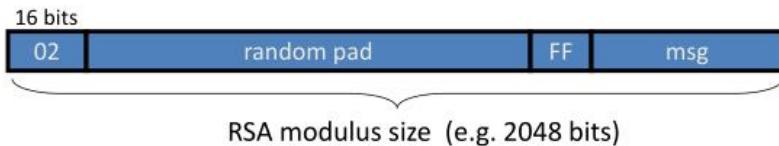
Per evitare il problema dell'attacco a forza bruta del testo in chiaro su messaggi piccoli (es: "si" o "no"), **il testo in chiaro viene sempre incrementato con un padding** che è scelto **in modo pseudo-casuale**, quindi deve esserci e deve avere una certa lunghezza precisa. In questo modo, **avremo diffusione e confusione** così tutto il testo cifrato dipenderà anche dal padding. (L'attaccante non sa più SI o NO, perché a quelli aggiungo anche dei simboli in più, es "SI"+hgdggddgyhyuebvchj).

Quindi chi decifra con la chiave privata non ha problemi, mentre un attaccante che non possiede la chiave non riesce più con un attacco forza bruta a sfruttare la struttura dei messaggi corti (come ad esempio: "si" o "no" che abbiamo visto la lezione scorsa).

STRUTTURA PADDING RSA PKCS1 v1.5:

PKCS1 v1.5

PKCS1 mode 2: (encryption)



- Resulting value is RSA encrypted
- Widely deployed, e.g. in HTTPS

Dan Boneh

Come prima cosa si hanno **16 bit con '02'** in formato esadecimale, seguiti da una stringa detta **random PAD**, poi **il byte FF del messaggio e infine il messaggio stesso**. **Se ho una chiave da 2048 bit non posso cifrare un messaggio da 2048 poiché una parte deve essere riservata al random PAD quindi chi decifra scarta ... e inserisce il modulo**. Viene allungata la porzione di attacco in cui si deve fare attacco forza bruta.

Questo fa sì che anche se il messaggio è in uno spazio limitato di messaggi (come SI o NO) quello che viene cifrato con RSA viene da uno spazio molto più grande, **l'attaccante per fare un attacco di quel tipo lo deve fare su tutti i tipi di messaggi possibili e tutti i random pad possibili**.

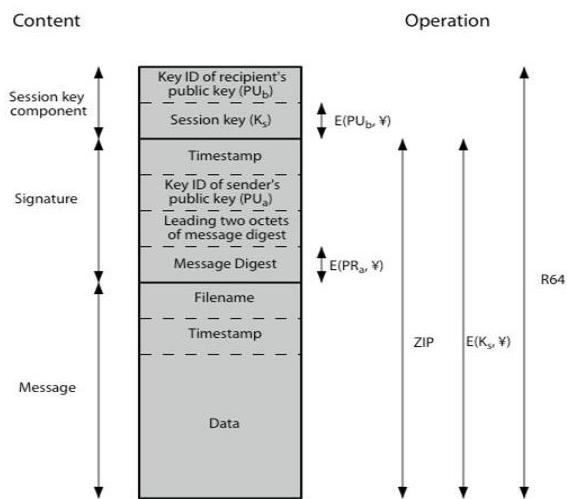
Questo è PKCS1, che specifica come utilizzare RSA per la cifratura e specifica l'utilizzo del random PAD. La versione 2 specifica un'altra modalità che usa funzioni di cui non abbiamo ancora parlato.

Questo che abbiamo appena visto è specificato nella versione 1.5.
I primi 16 bits rappresentano 02 in esadecimale e vengono messi poiché è un formato specifico.

Con questa aggiunta un messaggio in chiaro può essere lungo all'incirca 800 bit.
In generale è abbastanza lungo per trasferire una chiave simmetrica.

PGP POSTA ELETTRONICA (SCHEMA IBRIDO)

PGP Message Format

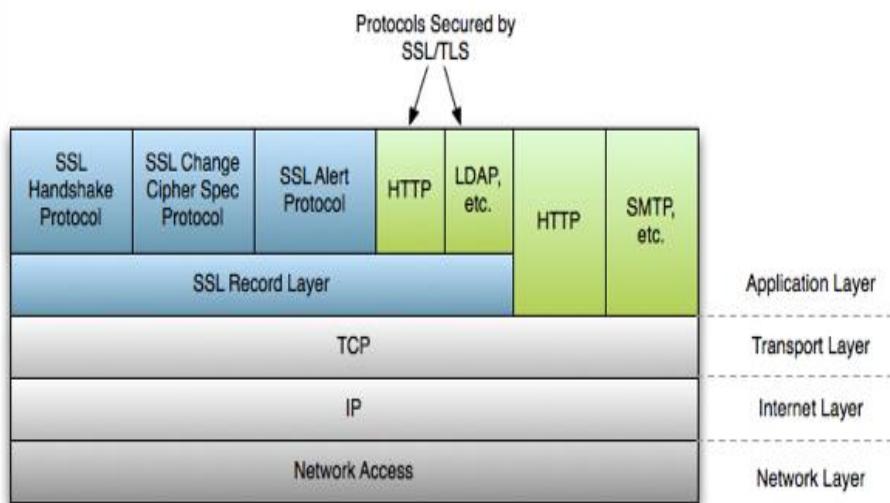


Questo è il corpo della mail in PGP. Tutta questa parte contenuta in ZIP (da Timestamp fino a Data) viene cifrato con una chiave simmetrica, quindi bisogna che il destinatario la conosca. La chiave simmetrica usata è in cima al formato del messaggio (nel campo session key), ed è cifrata con la chiave pubblica del destinatario perché così solo il destinatario può decifrarla con la sua chiave privata, quindi si tratta uno schema ibrido in un algoritmo asincrono. In questa maniera mando tutte le informazioni che servono, perché il grosso del messaggio (i dati principali + le loro informazioni) è cifrato a chiave simmetrica e la chiave simmetrica (da 256 bit) è cifrata con la chiave pubblica del destinatario.

Il destinatario decifra con la propria chiave privata per ottenere la session key e poi decifra tutto il messaggio con la chiave simmetrica ottenuta.

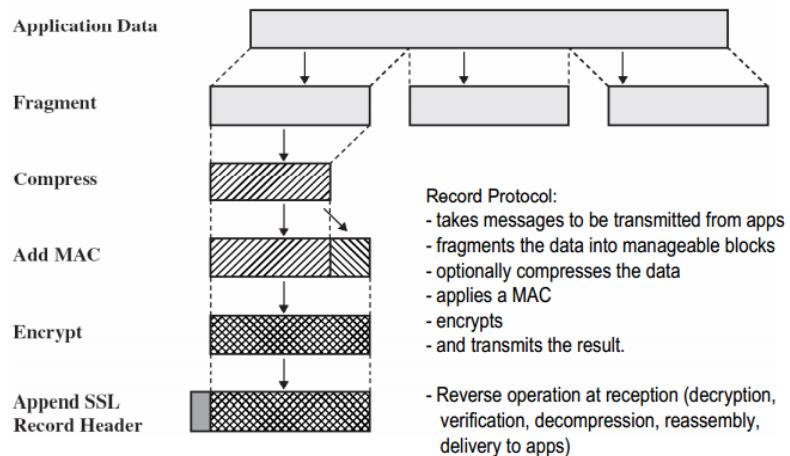
Key ID of recipient's public key è l'ID, serve per identificare quale chiave pubblica è stata usata per cifrare (un destinatario può possedere più di una chiave pubblica). E' un esempio per come si implementa lo schema ibrido ma all'esame non ci chiede a memoria ma lo dobbiamo capire.

Rivediamo la pila con SSL:



Ci ha detto che i segmenti in azzurro sono i protocolli di SSL. Adesso parliamo del Record Layer, il layer più sotto agli altri protocolli di SSL, il quale contiene il Record Protocol Operation:

Record Protocol operation

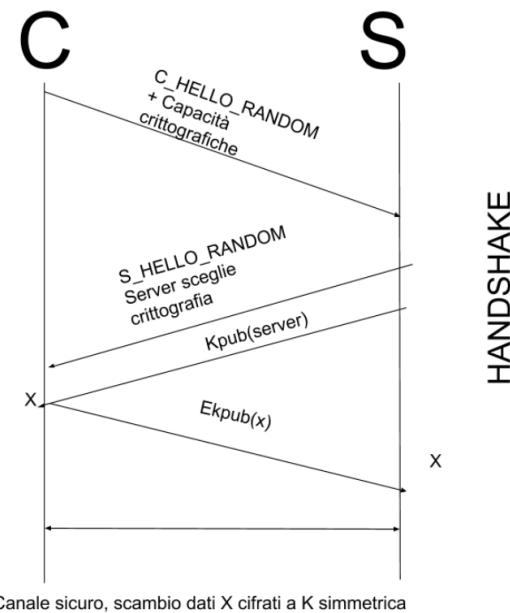


Il record layer sta in HTTP e i dati vengono segmentati a questo livello(fragment), normalmente è TCP che li segmenta ma in questo caso lo fa SSL in quanto il Record Layer si trova in un livello intermedio (tra app e trasporto). Dopo averli frammentati, i dati vengono compressi e il protocollo gli aggiunge un'informazione per l'integrità (per fare in modo che i frammenti arrivino tutti e nell'ordine giusto), li cifra a chiave simmetrica e aggiunge il proprio header al datagram.

Ma dove la trova il record layer la chiave simmetrica da usare?

Si avvale di altri 3 protocolli (quelli azzurri sopra di lui: Handshake, Change Cipher Spec e SSL Alert): quello che ci interessa è il protocollo di Handshake che si attiva quando client e server si mettono d'accordo su quale crittografia usare e sulla chiave pubblica da usare per passarsi la chiave simmetrica. Eventualmente ci potrebbero essere anche dei protocolli di autenticazione.

SSL HANDSHAKE PROTOCOL



Lo scambio di chiave con RSA. Uno dei modi per cifrare la chiave simmetrica è RSA.

- Il client richiede al server una connessione. Nel fare questo, **il client manda al server** due informazioni fondamentali :
 - una stringa pseudocasuale** scelta dal client detta **client_hello_random**, e
 - Informazioni sulle sue capacità crittografiche** (ovvero di quali cifrari dispone) di cui fa un elenco (es. 1 AES-256, 2. DES 40 bit...).
- Il server risponde con** il suo **server_hello_random** (stringa pseudo-casuale generata dal server).
- A questo punto **il server sceglie in base alle capacità del client quale crittografia useranno**, in particolare si sceglie **quella più sicura rispetto allo standard utilizzato da entrambi** (ovvero La più sicura che il client è in grado di gestire ma anche i server se è quest'ultimo ad essere limitato).

A questo punto i due si sono messi d'accordo su alcuni fatti.

Adesso vedremo come avviene lo scambio della chiave con RSA.

- Il server manda al client la propria chiave pubblica.**
- Il client manda al server la stringa cifrata con la chiave pubblica del server** e a questo punto tutti e due hanno in mano **x** (il messaggio)

A questo punto c'è qualcosa di cui ci parlerà più avanti(una piccola fase in mezzo).

Il canale è basato sul segreto x che è simmetrico. **Hanno un segreto condiviso** che potrebbe essere direttamente la chiave simmetrica. **Solo dopo la fine dello scambio parte l'effettivo scambio di dati cifrati a chiave simmetrica come sempre.**

Questo è un **protocollo sincrono** perché è presente il server dall'altra parte.

Questo è il protocollo di Handshake che precede la reale comunicazione. **Le stringhe pseudo-casuali si scelgono per ogni nuova sessione.** Per generare la vera chiave simmetrica **il segreto condiviso viene mescolato con le stringhe pseudo casuali in questo modo la chiave reale non è la stringa pseudo casuale ma è una chiave alla quale la cui creazione hanno partecipato sia client che server con le stringhe pseudo casuali iniziali.** Il che significa che la chiave dipende dal server. Se il client si collega la prima volta, la seconda volta si collega l'attaccante. Se voglio fare un attacco di tipo reply, si potrebbero utilizzare gli stessi messaggi e replicare una sessione identica ma usando la stringa pseudo-casuale del server c'è la garanzia che la chiave di sessione sarà diversa e che i messaggi sono cifrati con una chiave diversa e quindi non avranno più significato per l'attaccante. **L'attaccante può avere la stringa ma non ha il segreto.**

FIRMA DIGITALE

Abbiamo visto l'uso della chiave pubblica per lo scambio di chiavi simmetriche. Naturalmente è comodo, ma con la cifratura a chiave pubblica si possono fare molte cose tra cui fare firme digitali. Cosa si intende per **Firma di un certo documento? Vuol dire assumerne la paternità.** Se uno firma un documento, **eventuali modifiche successive a quel documento saranno ben distinguibili da queste.** Nessuno accetterà mai con una firma appiccicata con un adesivo. **La funzione principale della firma è legare indissolubilmente il documento a chi vi ha messo la firma.**

FIRMA <-> SOGGETTO

FIRMA <-> DOCUMENTO

tutto ciò implica che SOGGETTO<-> DOCUMENTO

Questo genera un legame tra il documento e il soggetto, questo legame ha 3 significati:

1. **Autenticazione:** il soggetto è quello che ha prodotto il documento;
2. **Integrità:** si evita che il documento venga alterato da altri dopo averlo stilato;
3. **Non Ripudiabilità:** La firma è una prova che chi ha firmato ha accettato ciò che c'è scritto sopra e lo ha firmato di sua spontanea volontà.

Queste 3 caratteristiche da cosa derivano?

Le ipotesi dietro a tutto questo sono:

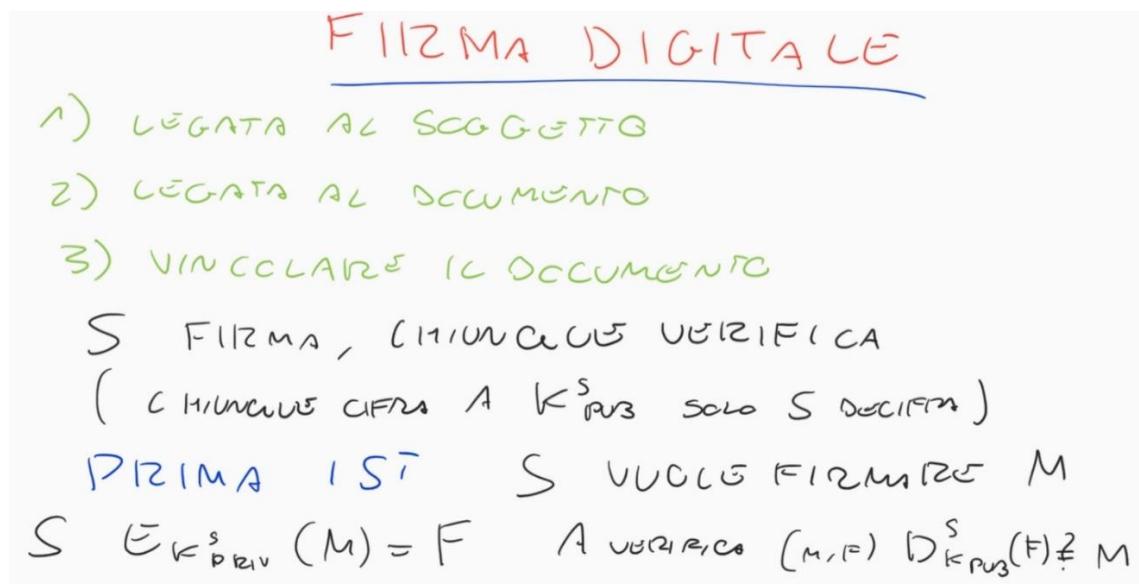
1) il legame della firma con il soggetto questo funziona a meno che qualcuno non falsifichi molto bene la mia firma quindi sembra che sia fatta dal soggetto ma in realtà non è così.
2) Che la firma sia legata indissolubilmente al documento, se uno riesce a copiarla da un'altra parte o riesce a modificare il documento in modo che risulti ancora firmato etc. ovviamente viene a decadere la seconda parte, naturalmente anche su documenti di carta questa cosa non è assolutamente vera perché uno può essere molto bravo e bypassare questi 2 vincoli. (infatti, la firma normalmente, se uno firma un documento senza che ci siano testimoni, se io scrivo su un foglio che darò 1000 euro a Caggesse, lui può andare e cercare di riscuotere questi soldi, ma se io dico che non ho mai firmato niente del genere sta a LUI dimostrare che io ho davvero firmato perché è una firma che può aver falsificato, per questo di solito si fanno ste cose davanti un notaio che garantisce che io ho fatto quella firma).

E' necessario quando si fa una firma digitale riuscire a ottenere queste 2 caratteristiche nel modo migliore possibile in modo tale che per un attaccante sia molto difficile fare un attacco. Il problema è che non si può usare lo stesso modello della firma in un foglio di carta, perché sappiamo benissimo che nel mondo informatico non esiste un notaio che certifica queste cose, dunque **la firma digitale deve essere:**

- 1) Legata al soggetto- autenticazione**
- 2) Legata al documento- non ripudiabilità**
- 3) Deve vincolare il documento perché io non possa modificare quel documento e risultare sempre firmato dalla stessa persona-integrità**

Come otteniamo tutto questo? Tiriamo fuori l'ultima caratteristica della firma: Una firma del soggetto S

- 1) viene scritta da S
- 2) Chiunque può verificare che S ha firmato.



Ricordiamoci una cosa che abbiamo visto nelle crittografia a chiave pubblica. Chiunque può cifrare a chiave pubblica, mentre solo chi ha la chiave privata può decifrare. Allora, qual è l'idea che può venire da questi 2 ultime osservazioni? Usiamo esattamente la stessa tecnica, ma al contrario.

Prima istanza:

Se S vuole firmare M, S cifra con la propria chiave privata il messaggio M, se A vuole verificare, controlla che il documento F cifrato con la chiave privata di S, una volta decifrato con la chiave pubblica di S corrisponda a M. Anche nel mondo digitale abbiamo un documento una firma e chi verifica, verifica che questa F sia effettivamente la cifratura di M.

Quindi se A è l'agente che verifica la firma, A avrà in input sia M che F, a quel punto decifra F con la chiave pubblica (che è legata in modo indissolubile con la chiave privata di chi ha firmato il messaggio) e verifica che decifrando il messaggio cifrato con la chiave privata usando la chiave pubblica corrisponda ad M.

La cifratura è fatta con la chiave privata perché la chiave privata è posseduta normalmente solo da una macchina, mentre invece tutti gli altri hanno la pubblica con la quale verificano.

Un problema aperto è sapere a chi è associata la chiave pubblica.

Utilizziamo il meccanismo sopra citato poiché la chiave pubblica e quella privata sono legate indissolubilmente e sono intercambiabili.

$$e \cdot d = \text{congruo } 1 \text{ mod } (p-1)(q-1)$$

Come vedete questa cosa dice che sono uno l'inverso dell'altro, quindi cifrare con una chiave e decifrare con l'altra è perfettamente simmetrico.

$e \rightarrow$ chiave pubblica

$d \rightarrow$ chiave privata

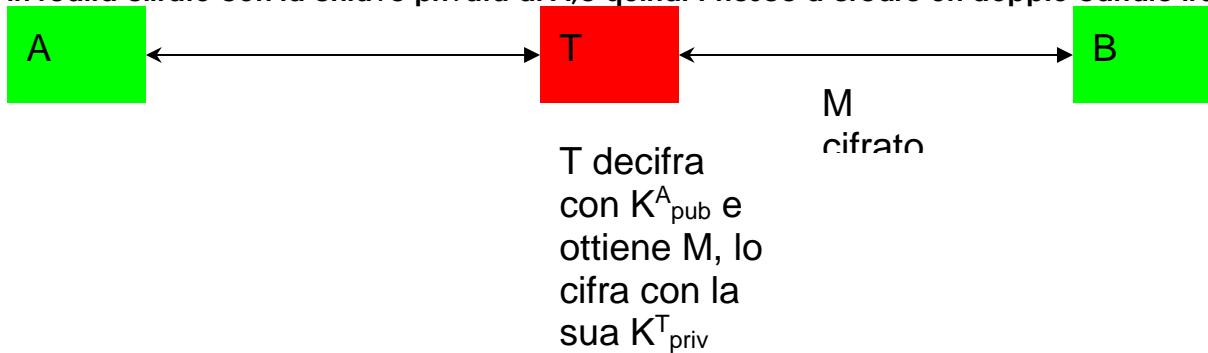
dato che **e, d sono due numeri matematicamente interscambiabili, uno può scegliere uno qualunque dei due come chiave pubblica e l'altro come chiave privata**, uno può sempre cifrare con una chiave e decifrare con l'altra dopo che ha scelto quale è quella pubblica e quale è quella privata.

La ragione per cui ha scritto chiave privata è perché solo S ha la sua chiave privata quindi solo S può firmare e chiunque può decifrarla(verificarla) con la chiave pubblica.

Cifrando con chiave pubblica e privata si ottengono cose diverse.

Chi fa la verifica decifra con la chiave pubblica e potrebbe ottenere una cosa che non è M.

Questa proprietà è valida solo nell'ipotesi che la chiave privata sia in mano ad S e solo S. **Potrebbe succedere che, se si ha un Man in the Middle T tra due soggetti A e B, che T intercetti il messaggio cifrato con la chiave privata di A, lo decifri usando la chiave pubblica di A e quindi crea un canale tra A e T. Poi decifra tutti i messaggi di A e li cifra con la sua chiave privata K_T e li invia a B. B quindi crede di star parlando con A in quanto crede che il messaggio cifrato con la chiave privata di T sia in realtà cifrato con la chiave privata di A, e quindi T riesce a creare un doppio canale tra A e B.**



La chiave privata non la deve vedere nessuno mentre quella pubblica sì.

DOMANDA EDO: quando mi arriva il messaggio cifrato e in chiaro. Non è possibile che attaccante modifichi messaggio in chiaro e cifrato in modo che corrispondano?

Questa è una **proprietà della non malleabilità che significa che è difficile modificare un testo cifrato in modo che si ottenga una modifica minima del testo in chiaro ed è una proprietà del cifrario.**

Abbiamo visto che la firma è legata al soggetto, ora dobbiamo vedere che sia legata al documento, e lo è grazie al principio di non malleabilità che abbiamo esposto prima (proprietà del cifrario)

Ci sono 2 problemi da risolvere ancora:

- La chiave privata e quella pubblica si possono scambiare e dipende solo da quale chiave che io ho deciso di usare come privata, ricordate però che nell'uso pratico non è così. Nel caso pratico openssl usa solo 3 chiavi pubbliche. Se la chiave pubblica è quella cosa succede? per forza quella deve essere la chiave pubblica e non può essere privata poiché altrimenti la saprebbero tutti e sarebbe il segreto di pulcinella XDxD. Non è un errore matematico o implementativo ma di protocollo e di utilizzo perché è stato usato per rendere più efficiente l'operazione di generazione di chiave pubblica.

FUNZIONI HASH E COLLISIONI

1024, 2048, 4096

128 B 256 B 512 B

$$M \rightarrow \boxed{H} \rightarrow H(M)$$

$$H : \{0,1\}^* \longrightarrow \{0,1\}^n \quad n = \begin{cases} 128 \\ 256 \\ 512 \end{cases}$$

Ricordiamo che si cifra con $E_{K_{PRIV}(S)}(M)$; **quanto deve essere lungo al massimo il messaggio che cifra? Deve essere minore della lunghezza della chiave.** Le chiavi hanno lunghezze 1024(128 Byte), 2048(256 Byte), 4096(512 byte).

128 caratteri sono molto pochi per un contratto per esempio.

Quali sono le funzioni per ridurre un input?

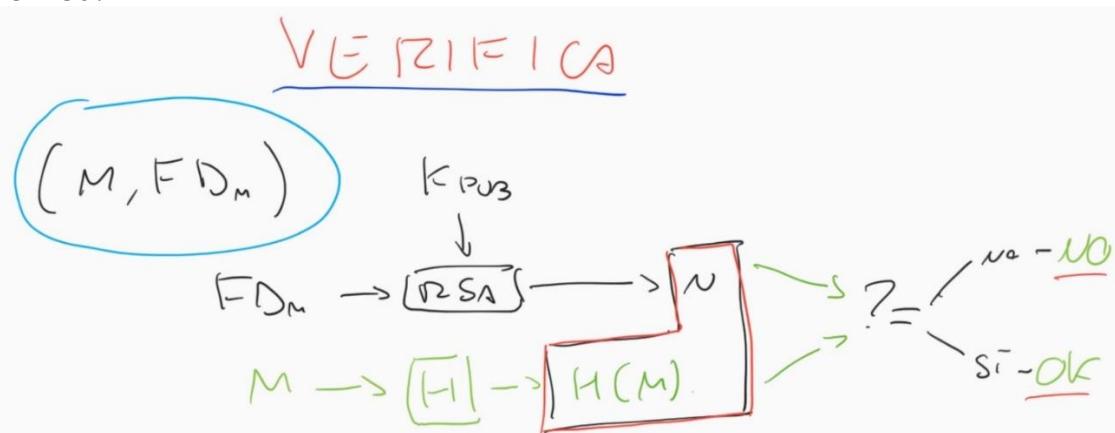
Con una funzione di hash(Hashing).

Una Funzione Hash è una funzione che prende in input un qualunque formato di stringa di bit $\{0,1\}^*$ e restituisce una stringa di lunghezza n che dipende strettamente dalla funzione hash utilizzata (generalmente "n" è abbastanza piccolo). **In crittografia** le funzione hash utilizzate **creano un messaggio lungo n che** nel passato andava da 128 bit mentre **ora parte da 160 fino a 512 bit.**

Il processo di firma digitale funziona nel seguente modo quindi:

$$M \rightarrow \boxed{H} \rightarrow H(M) \xrightarrow{K_{PUB}} \boxed{\text{RSA}} \rightarrow FD$$

Si fa una funzione hash per ridurre il messaggio M e si ottiene $H(M)$. Questo viene cifrato con la chiave privata del soggetto S usando RSA e si ottiene la firma digitale. Poi si passa alla fase di verifica:



Si decifra con RSA usando la chiave pubblica di S sul messaggio firmato (dopo essere stato compresso) e si ottiene un messaggio N . Notare che non si è ottenuto M poiché l'hashing non è invertibile in quanto sono state perse delle informazioni.

Allora semplicemente si riprende M e si calcola l'impronta (cioè si applica la funzione di hash) e si verifica se $H(M)$ appena ottenuto e N ottenuto dalla decifratura RSA del messaggio firmato sono uguali. Se lo sono, la firma è verificata (ritorna OK), altrimenti no (ritorna NO). Utilizzare le funzioni hash non è una cosa leggera e ne parleremo più avanti.

Nel mondo digitale, la firma digitale e quella elettronica sono due cose diverse. In realtà la firma elettronica ha un formato diverso e potrebbe essere anche solo costituita da username e password. Il meccanismo a chiave pubblica invece si chiama firma digitale quando vengono verificati altri requisiti che vedremo più avanti.

Parentesi:

In ambito giuridico le espressioni firma digitale e elettronica sono differenti, elettronica in ambito giuridico significa login, password..

Proviamo a firmare con RSA

Non è buona cosa usare la stessa coppia di chiavi per cifrare e firmare. Diciamo che se un'identità firma documenti che gli vengono sottoposti e poi con la stessa chiave si usa per cifrare e decifrare, qualcuno potrebbe sotoporre un documento che vorrebbe vedere decifrato. Siccome firma e decifra sono fatte con la stessa chiave privata questo potrebbe aprire nuove soluzioni agli attaccanti. **La cosa fondamentale è che la coppia di chiavi sia diversa per la cifratura e per la firma.**

Ogni funzione Hash ha un suo identificativo, perché l'impronta che è stata scritta dice anche con quale hash è stata calcolata.(in questo caso quella selezionata dovrebbe essere l'identificativo di md5)

Infatti su Okteta otterremo quello che ci interessa.

Decifrato e impronta sono diversi:

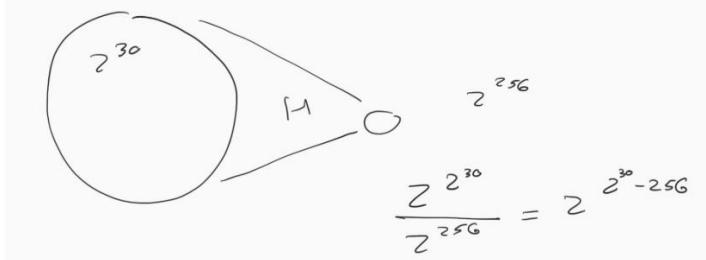
in base al PKCS#1 prima di cifrare con RSA all'impronta deve essere premesso l'AlgorithmIdentifier del digest.

Uno potrebbe sostituire il documento con un'impronta??? (questa parte sarebbe da rivedere ma forse non è importante)

La verifica è come ci ha fatto vedere lei nel disegno (con SI, NO ecc.) ma c'è anche il formato da rispettare che è PKCS. Introduzione della funzione Hash qui in mezzo.

La verifica di una firma digitale viene fatta guardando che la firma decifrata sia uguale all'impronta Hash del documento (quindi non sul documento iniziale). Quindi la verifica è con l'impronta hash.

Abbiamo detto che la funzione hash associa ad una stringa (pseudo)arbitraria una stringa di lunghezza limitata. Prendiamo un mondo di elementi e vogliamo restringerli.



Se prendiamo l'insieme di documenti lunghi un giga, otteniamo $2^{(2^30)}$ documenti possibili. Dopo essere stati compressi formano un insieme di 2^{256} documenti quindi questa non può essere una funzione iniettiva. Non è detto che per due documenti diversi io ottenga documenti compressi diversi. Quanti sono i documenti che in media hanno la stessa impronta?

$$\text{Sono } \rightarrow \frac{2^{2^{30}}}{2^{256}} = \frac{2^{2^{1073741824}}}{2^{256}} = 2^{(2^{30}-256)}$$

Se attaccante usa un file $M' \neq M$ tale che $H(M')=H(M)$, la verifica in questo caso non fallisce. [Quindi chi riceve proverà a decifrare con chiave pubblica il file firmato dal vero proprietario e arriverà ad ottenere un $H(M)$ che è quello reale, ma se l'attaccante sostituisce il file non firmato M (che dovrebbe essere usato da chi riceve per verificare cioè confrontare) con un file M' che ha la stessa impronta $H(M')$ del file M allora in quel caso la verifica non fallirà e l'attaccante sarà riuscito a sostituire il messaggio]

Istante in cui un utente firma un file lungo 1GB sta firmando $2^{(2^{30}-256)}$ file poiché la verifica funziona così.

Naturalmente tra tutti questi file e tutte queste sequenze di bit lunghe un GB non sono tutte scritte in italiano ma sono così tanti che ci saranno in mezzo documenti scritti in italiano e ce ne sono veramente tanti.

Il fatto che ad ogni messaggio corrispondeva in modo univoco una firma ora non è più vero a causa della introduzione della funzione hash. Quindi la funzione hash ci genera un problema. Hash non introduce niente di buono se non la riduzione della lunghezza del messaggio. Dal punto di vista della sicurezza della firma digitale introduce molte vulnerabilità, ciononostante vedremo che è molto più difficile falsificare una firma digitale rispetto a una cartacea. Se portando un documento firmato in digitale a un giudice non è più come nel caso della firma cartacea che chi ha firmato poteva negare di averlo firmato, e chi riceve la firma deve verificare che ha veramente firmato il documento. Con la firma digitale succede esattamente il contrario: è chi ha firmato il documento che deve dimostrare di non averlo fatto.

La situazione che esistano due messaggi con la stessa impronta con Hash si chiama collisione. Una buona funzione di hash deve fare in modo che sia molto difficile avere una collisione (due messaggi diversi m, m' con stesso hash $H(m)=H(m')$).

Per ampliare il raggio di azione dell'attacco, l'attaccante potrebbe utilizzare una funzione hash diversa da quella utilizzata realmente per trovare un messaggio che corrisponda (cioè che trovi una collisione usando una funzione diversa da quella usata su m). Questa è la ragione per cui dentro la firma è scritto con quale funzione hash è stata fatta la firma.

FUNZIONI HASH CONTINUA

Avevamo parlato del fatto che in RSA prima di firmare un documento con RSA, si calcola l'impronta HASH del messaggio per diminuire la lunghezza dei dati. Questo perché la cifratura a chiave pubblica è pesante e inoltre la firma digitale è il risultato dell'operazione quindi si vuole che il risultato (cioè la firma) sia compatto. Questa è la ragione per cui si usa la funzione Hash, però già sappiamo dall'aver usato funzioni hash in altri contesti che hanno il problema delle collisioni.

La funzione Hash può essere buona quanto vogliamo ma le collisioni sono inevitabili! Una funzione hash è definita come funzione che prende in input stringa e restituisce una stringa lunghezza prefissata. Ciò significa che ci sono sempre e solo 2^n possibili impronte di una funzione hash, mentre gli input sono virtualmente illimitati di numero (in realtà un limite c'è ma è molto alto quindi è trascurabile).

Se il testo di cui si vuole calcolare l'hash ha una lunghezza \leq di un certo messaggio n , quindi m molto maggiore di n ($m > > n$), allora $2^m > > > 2^n$ (non è una convenzione standard mettere tanti $>$)

Se $|m| \leq n$ con $m > n$ allora $2^m > 2^n$

quindi significa che le collisioni sono inevitabili.

Se un soggetto firma una certa stringa di bit, sta firmando in realtà tante altre stringhe di bit che hanno la stessa impronta hash ed è un problema grave che va esplorato.

Per risolvere questo problema si usano quindi le funzioni hash crittograficamente sicure, se uno deve fare una hash table non usa questo tipo di hash perché sono più costose, per una hash table i problemi sono diversi: qualunque funzione hash venga scelta, esiste un insieme di dati che si mappa male con quella funzione hash (produce tante collisioni).

In crittografia è una cosa diversa e ne conosciamo già qualcuna:

MD5, SHA1, SHA256, sono funzioni di hash

Che cosa deve essere una funzione hash per essere usata nella sicurezza. Deve essere veloce da calcolare quindi l'efficienza è sempre utile.

FUNZIONI HASH CRITTOGRAFICAMENTE SICURE

FUNZIONI HASH CRITTOGRAFICAMENTE SICURE

$$(n) \quad H : \sum_{\{0,1\}^n} \rightarrow \{0,1\}^m \quad \left| \begin{array}{l} m \leq 5 \\ 5 \leq m \leq 256 \end{array} \right.$$

1) "non invertibile"

MOLTO DIFFICILE dato h a trovare M t.c. $H(M) = h$

$O(2^m)$



Cosa rende una funzione hash crittograficamente sicura:

- La funzione hash deve essere non invertibile (detta anche preimmagine): deve essere molto difficile, data un'impronta "h" generata da $H(M)$, trovare il messaggio M da cui h è stata generata con $H(M) = h$. (cioè deve essere molto difficile risalire a M data la sua impronta h). COSTO: $O(2^n)$, con n lunghezza di h .

E' vero sempre che data un'immagine di una funzione hash non si può sapere qual'è la sua **contro immagine** (ovvero quale fosse l'input dato l'output), e dato che non è iniettiva io non ottengo il messaggio di partenza se faccio l'inverso. Dal punto di vista della crittografia si chiede che sia molto difficile dato un'impronta "h" generata da $H(M')$ trovare un M tale che $H(M) = h$. Molto difficile significa che, dato n lunghezza delle impronte h , l'attaccante può sempre fare un attacco a forza bruta, provando a fare hash di tutti gli M possibili finché non ne trova uno che dà come risultato la stessa impronta "h" (prova tutti i messaggi M possibili finché non trova una collisione). Se i valori della funzione hash non sono uniformemente distribuiti (se uno sceglie a caso degli M trova tutti i valori in output), allora non è una buona funzione hash perché ci sono informazioni su come funziona, e in questo caso gli attaccanti potrebbero sfruttare le informazioni sulla struttura e preferenze della funzione hash per attaccare. Se invece si ha una hashmap che produce in modo uniforme gli output (hash con valori uniformemente distribuiti), succede che solo quando l'attaccante ha provato $O(2^n)$ diversi valori per M troverà un valore per M che come impronta hash dà proprio h . E' quindi sempre possibile con un ordine di $2^n \leftarrow$ COSTO ATTACCO MIGLIORE

Il codominio è $O(2^n)$ quindi se la funzione di hash è ben distribuita, l'attaccante riuscirà a trovare un M tale che $H(M)=h$ nell'ordine di $O(2^n)$ questo è quindi il costo dell'attacco migliore in questa situazione.

In sostanza l'attaccante sceglie a caso in modo distribuito dall'insieme dei messaggi M possibili (quindi sempre in modo diverso ecc.) e in questo caso siamo sicuri che trovi un M tale che $H(M)=h$ nell'ordine di $O(2^n)$ che è la dimensione del codominio.

- **Resistenza alle collisioni in senso debole** (seconda preimmagine): è molto difficile, dato un messaggio M , trovare un messaggio $M' \neq M$ tale che $H(M) = H(M')$. Qui l'attaccante rispetto a prima ha qualcosa di più: un certo M che ha una certa immagine $H(M)$, quindi potrebbe lavorare su questo M per trovare un altro M' con la stessa immagine di M .

Se ciò è molto difficile, allora $O(2^n)$ è l'attacco che si può fare che come quello di prima è a forza bruta. La funzione è resistente alle collisioni in senso debole se $O(2^n)$ è il costo minimo di un attacco a forza bruta che abbia successo.

Sostanzialmente si chiama preimmagine perché un'immagine è già data (M) ed M' è la seconda immagine che viene ricercata. Se esiste un attacco migliore di quello, la funzione hash ha delle regolarità che si possono sfruttare per capire come sono fatte le immagini, preimmagini etc.

- **Resistenza alle collisioni in senso forte** (Resistenza alle collisioni): è molto difficile per l'attaccante determinare due messaggi M, M' ($M \neq M'$) tali che $H(M) = H(M')$.

La differenza tra il caso 2 e il caso 3 è che nel caso 2 M è dato, mentre nel caso 3 l'attaccante li può scegliere entrambi (sia M' che M). Se questa proprietà vale, il costo minimo di un attacco forza bruta è $O(2^n/2)$ che da notare è molto più piccolo di $O(2^n)$ (l'attacco rimane un attacco a forza bruta perché se funzione hash è fatta bene, devo andare a caso).

L'attacco usato è detto di tipo "compleanno" (scegliendo più di un'incognita posso ridurre l'insieme di valori possibili e trovare una collisione in meno tempo): supponiamo che la gente faccia gli anni distribuiti su tutto l'anno, e si fissa un compleanno per ognuna di esse.

Prendendo un'altra persona completamente a caso, la probabilità che sia nata in un certo giorno dell'anno sarà $1/365$. In un gruppo di 23 persone, la probabilità che 2 persone abbiano il compleanno lo stesso giorno sarà più del 50%: ci sono 23^2 coppie di persone circa e quindi la probabilità che 2 tra queste persone compiano gli anni lo stesso giorno è molto più alta. La possibilità di scegliere arbitrariamente sia M che M' riduce la complessità computazionale, poiché fornisce due gradi di libertà e si chiama attacco di tipo compleanno per questo motivo. Per non essere resistente deve esserci un attacco di costo inferiore a questo $2^n/2$.

Perché senso forte? Attaccante in quello debole è più debole. Ci spiega con un esempio legato alla firma digitale

Esempio 1:

Collisione Debole:

Se l'attaccante ha in mano un assegno firmato da un altro fatto così:

M = pagherò 3 euro

$FD = Ek_{privata}(H(M))$

Se vuole farsi pagare di più deve trovare $M' =$ pagherò una qualche cifra.

Non può cambiare M che è prefissato, se trova M' tale che $H(M')=H(M)$ allora la firma di $H(M')$ sarà identica a quella di $H(M)$ quindi questa collisione l'ha dovuta cercare avendo M prefissato. Quindi qui in questo caso l'attaccante è debole siamo nel caso 2.

Esempio 2: Collisione Forte

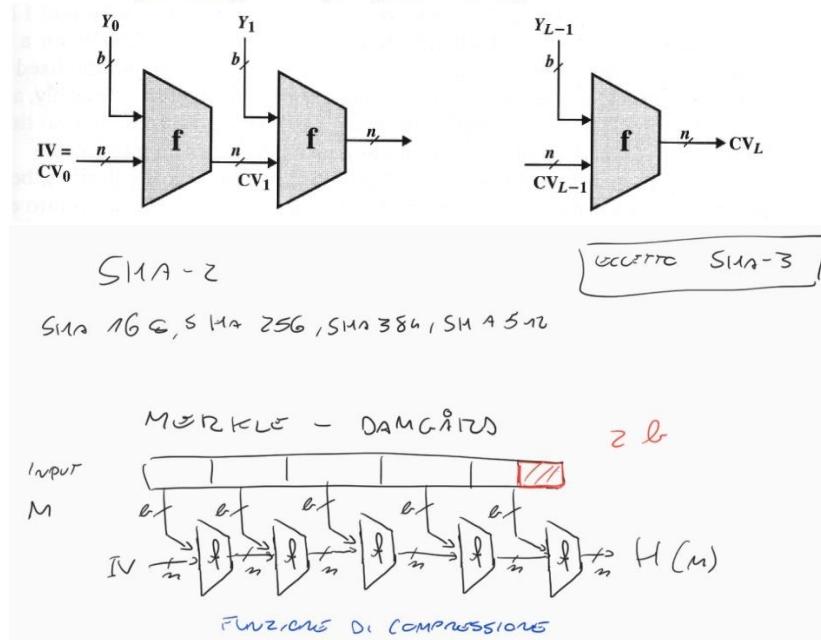
Un ISP ha un servizio di web hosting, ma ha una sua politica selettiva sui contenuti. Naturalmente per evitare che le pagine online rispondano ai requisiti usa un sistema automatico, quando viene proposta una pagina web, questa se va bene viene firmata ($FD(W)$) e questa firma digitale viene salvata, c'è un sistema automatico che fa il giro regolarmente e controlla che tutte le pagine siano correttamente firmate. L'attaccante genera 2 pagine W e W' tali che $H(W)=H(W')$, quindi mostra la pagina W che viene vagliata e accettata e quindi firmata, ma poi l'attaccante mette online la pagina W' con la firma digitale di W . La pagina quindi rimarrà online finché qualcuno non se ne accorgerà. L'attaccante quindi può scegliere sia M che M' , siamo nel caso 3 quindi l'attaccante in questo caso è forte (caso 3).

ESEMPIO DI USO DELLO XOR PER UNA FUNZIONE HASH CON STRUTTURA DI MERKLE-DAMGÅRD

Le funzioni hash eccetto SHA-3, seguono la struttura di MERKLE-DAMGÅRD, questa struttura dice come fare una funzione hash, e vi spiega anche perché l'input può essere di lunghezza arbitraria.

funzione HASH basata su XOR

- `java -jar XORhash.jar <file_di_input>`
- struttura di Merkle-Damgård con:
 - $b=n$: 4 byte; $f=XOR$; IV= 0x00000000
 - padding 1-4 byte (tutti 0x00)



Presi in input b bit per volta che vengono passati ad una funzione di compressione (f) insieme ad un vettore iniziale (noto) di lunghezza N , la funzione di compressione dà in output n bit e viene data in pasto ad un'altra funzione di compressione che prende in input un'altra sequenza di bit e così via. L'unica cosa da dire è che c'è sempre un completamento e parte di questo dice quanto è lungo il messaggio e quindi impone una lunghezza massima ma sicuramente non può superare 2^b . Dopo l'ultimo blocco, l'output dell'ultima funzione di compressione è $H(M)$. Il vettore iniziale IV, fa parte dell'algoritmo non c'è niente di segreto a livello algoritmico. Prima cosa: Tutte le funzioni hash eccetto SHA3 hanno questa struttura e cambia solo la funzione di compressione, ci possono essere attacchi di criptoanalisi ovvero cercare di capire la debolezza delle funzioni oltre all'attacco forza bruta.

sicurezza delle funzioni hash

- MD2: Muller, 2004: pre-image attack in $O(2^{104})$
- RIPEMD-160: relativamente poco usata e poco studiata
- TIGER: Kelsey-Lucks, 2006: trovano collisioni per Tiger limitata a 16 round (su 24) in $O(2^{44})$...

MD2: questi MD nel 2004, Muller ha trovato un attacco pre-immagine in $O(2^{104})$

Il gruppo di Wang ha ottenuto un sacco di risultati disastrosi.

Collisioni per MD5 in poco più di un'ora.

Siccome l'insieme di tutti gli input è molto più grande degli output c'è sempre una collisione ma c'è un problema: non tutti gli input sono significativi, quindi quello che è interessante è avere collisioni che sono legate in modo significativo l'un l'altra. Su MD5 Si riusciva a trovare collisioni tra file di comandi arbitrari e file con con... etc.

attacco di Wang et al.

- 2005: Wang et al. mostrano un attacco per trovare due documenti con la stessa impronta SHA-1 con costo $O(2^{69})$
(contro 2^{80} , se SHA-1 fosse resistente alle collisioni)
- (attacco reso più efficiente in seguito)

Nel 2005 hanno mostrato un attacco per trovare due documenti con la stessa impronta SHA-1 con costo $O(2^{69})$ questo infatti è il terzo tipo di attacco. Ridurre da 2^{80} a 2^{69} si evidenzia debolezza funzione hash. SHA-1 è stato programmato che venga nel 2010 dovuti agli avanzamenti della tecnologia nei quali si può raddoppiare la computazione ogni anno.

Nella firma digitale in Italia si può usare solo SHA-256 e non più SHA1.

ETSI è un'organizzazione europea che si occupa di tecnologia e sicurezza. Questi sono gli algoritmi di firma raccomandati e RSA fa riferimento a RFC. Funzioni hash raccomandate per resistenza in x anni. Le varie SHA 224,256,384,512. A dieci anni SHA 224 non si può usare e le ultime due vanno ancora bene. Poi ci farà vedere come si fanno queste collisioni.

[PAG 25]

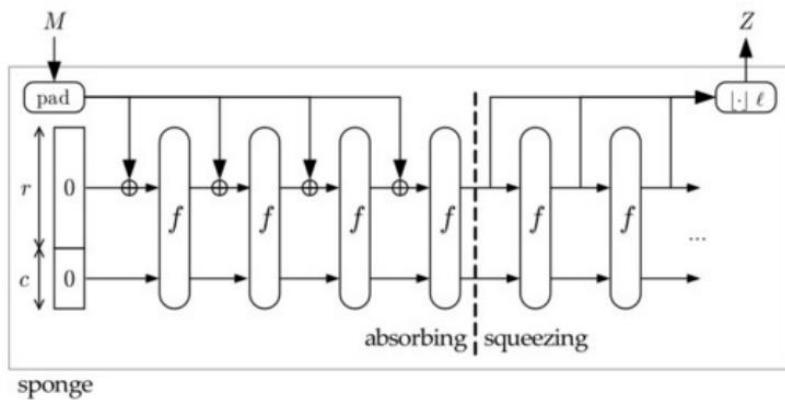
ultima tabella. Non raccomandato di usare RSA con SHA 256 nel 2030.

When Will We See Collisions For SHA-1?

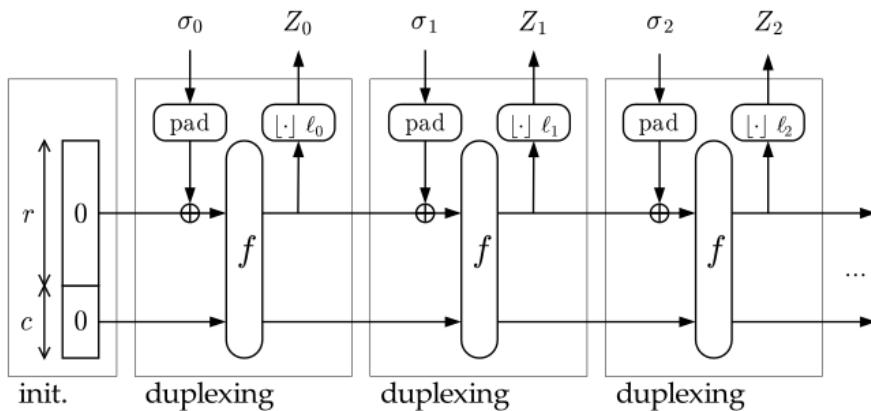
Il procedimento dice: un core oggi offre 2^{31} cicli al secondo, lo stato dell'arte è 8 core in ogni processore per un totale di 2^{34} un server ha circa 4 processori e diventano $2^2 \cdot 2^{34} = 2^{36}$ cicli al secondo e dal momento che ci sono circa 2^{25} sec/anno Conclude che un attacco alle collisioni sarà nel 2018 nel budget in criminalità organizzata e nel 2021 per l'università. Si vede che questo argomento tiene conto di hardware e non di instruction set diversi e miglioramento di altro tipo come uso di GPU. Questo l'ha fatto con riferimento all'attacco di Stevens.

SHA-3

Sponge function (KECCAK)



Avendo fatto un'analisi delle varie debolezze si è deciso di fare una nuova funzione hash, il meccanismo è quello usato per passare da DES ad AES, è stata indetta una gara nel quale sono state selezionate alcune proposte, fatte alcune conferenze e questo SHA-3 è quella che ha vinto.



La funzione Sponge function KECCAK : non ha più la struttura di prima, ma ha una struttura a "spugna" [CHE NON CHIEDE ALL'ESAME] ci sono delle funzioni che sono funzioni di compressione, c'è il messaggio che viene completato e poi diviso in blocchi che viene messo in XOR con le stringhe R e C che sono stringhe di lunghezza configurabile, più è lungo R più è veloce la funzione di hash perché i blocchi sono più lunghi, più è lungo C ci sono più garanzie di sicurezza ci sono. Ogni blocco viene messo in XOR con R e la funzione "F" elabora e si chiama sponge function perché fino a un certo punto assorbe i dati e poi caccia fuori l'output verso la seconda parte dell'implementazione .

Abbiamo visto tutto il quadro, osserviamo ora un po' di cose:

Spiega:

How it works

The above files were generated by exploiting two facts: the block structure of the MD5 function, and the fact that Wang and Yu's technique works for an arbitrary initialization vector. To understand what this means, it is useful to have a general idea of how the MD5 function processes its input. This is done by an iteration method known as the Merkle-Damgard method. A given input file is first padded so that its length will be a multiple of 64 bytes. It is then divided into individual 64-byte blocks M_0, M_1, \dots, M_{n-1} . The MD5 hash is computed by computing a sequence of 16-byte states s_0, \dots, s_n , according to the rule: $s_{i+1} = f(s_i, M_i)$, where f is a certain fixed (and complicated) function. Here, the initial state s_0 is fixed, and is called the **initialization vector**. The final state s_n is the computed MD5 hash.

The method of Wang and Yu makes it possible, for a given initialization vector s , to find two pairs of blocks $\textcolor{red}{M}, \textcolor{red}{M}$ and $\textcolor{red}{N}, \textcolor{red}{N}$, such that $f(f(s, \textcolor{red}{M}), \textcolor{red}{M}) = f(f(s, \textcolor{red}{N}), \textcolor{red}{N})$. It is important that this works for any initialization vector s , and not just for the standard initialization vector s_0 .

Combining these observations, it is possible to find pairs of files of arbitrary length, which are identical except for 128 bytes somewhere in the middle of the file, and which have identical MD5 hash. Indeed, let us write the two files as sequences of 64-byte blocks:

$M_0, M_1, \dots, M_{i-1}, \textcolor{red}{M}_i, \textcolor{red}{M}_{i+1}, M_{i+2}, \dots, M_n$,

$M_0, M_1, \dots, M_{i-1}, \textcolor{red}{N}_i, \textcolor{red}{N}_{i+1}, M_{i+2}, \dots, M_n$.

The blocks at the beginning of the files, M_0, \dots, M_{i-1} , can be chosen arbitrarily. Suppose that the internal state of the MD5 hash function after processing these blocks is s_i . Now we can apply Wang and Yu's method to the initialization vector s_i , to find two pairs of blocks $\textcolor{red}{M}_i, \textcolor{red}{M}_{i+1}$ and $\textcolor{red}{N}_i, \textcolor{red}{N}_{i+1}$, such that

$$s_{i+2} = f(f(s_i, \textcolor{red}{M}_i), \textcolor{red}{M}_{i+1}) = f(f(s_i, \textcolor{red}{N}_i), \textcolor{red}{N}_{i+1}).$$

This guarantees that the internal state s_{i+2} after the $i+2$ st block will be the same for the two files. Finally, the remaining blocks M_{i+2}, \dots, M_n can again be chosen arbitrarily.

So how can we use this technique to produce a pair of programs (or postscript files) that have identical MD5 hash, yet behave in arbitrary different ways? This is simple. All we have to do is write the two programs like this:

```
Program 1: if (data1 == data1) then { good_program } else { evil_program }
Program 2: if (data2 == data1) then { good_program } else { evil_program }
```

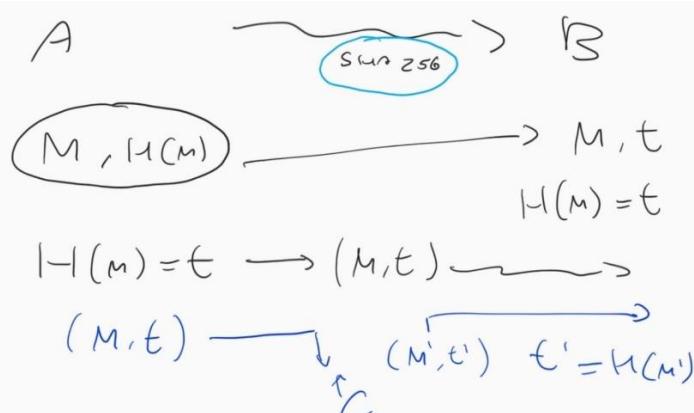
and arrange things so that "data1" = $\textcolor{red}{M}_i, \textcolor{red}{M}_{i+1}$ and "data2" = $\textcolor{red}{N}_i, \textcolor{red}{N}_{i+1}$ in the above scheme. This can even be done in a compiled program, by first compiling it with dummy values for data1 and data2, and later replacing them with the properly computed values.

Hanno la stessa impronta e ci troviamo di fronte ad uno scherzo che sono due eseguibili che hanno la stessa impronta tramite md5. Esiste il modo di costruire questi programmi e ci ha messo a disposizione tutto il programma che con istruzioni ci permette di creare sfruttando struttura di M-Damgard per cui localizza la ricerca della collisione. Succede che se da m_{i+1} in avanti, gli output di questi due sono uguali allora quelli dopo saranno uguali. Se struttura è uguale quello che arriva dalla chain variable l'idea è quella di lavorare solo su due blocchi e trovane altri due in modo che dato un IV uguale questi due blocchi hanno la stessa impronta hash e giocando su questa cosa la ricerca è molto localizzata e permette di costruire documenti uguali ma diversi solo nella parte centrale e costruire documenti significativi come per esempio dei programmi che sono sostanzialmente due programmi che contengono lo stesso codice ma che hanno un if davanti diverso.

PROBLEMA INTEGRITÀ DEI DATI:

Supponiamo che A voglia garantire l'integrità dei dati che invia a B

A → B



si decide di usare SHA256 e si calcola l'impronta di M ($H(M)=t$)

A manda la coppia $(M, H(M))$. B riceve questo, prende (M, t) , ricalcola $H(M)=t$ e li confronta: se sono uguali allora nessuno ha alterato i dati durante il tragitto. Quindi B è contento perché sa che l'attaccante non ha alterato i dati. Tutto questo garantisce integrità dei dati, ma B non sa chi gli ha spedito quei dati. **Quindi non ho garanzie sulla sicurezza del dato che arriva a B** e quello che è disegnato non è quindi uno schema per l'integrità. Se introduciamo un Man in the middle C che riceve i dati, questo li decifra e li tiene per sé, successivamente crea un suo messaggio M' e la sua impronta t' e li spedisce a B, mandando dati fasulli a B. C quindi ora possiede i dati di A e ha mandato dei dati fasulli a B. Quindi, **B non è sicuro della provenienza di quei dati**. Questo avviene perché **le funzioni hash sono pubbliche e quindi non c'è nessun segreto: quello che può fare A, lo può fare B e lo può fare anche C**.

Questo infatti protegge da errori di trasmissione ma non può proteggere in alcun modo chi riceve poiché non c'è un segreto.

In questo contesto l'attaccante potrebbe entrare su un sito e modificare il codice, però a quel punto modificherebbe anche l'impronta.

Come si risolve questa situazione? Con la firma!

Mettiamo casi di voler firmare un software per evitare la situazione descritta prima.

Se il software è firmato, a questo punto una firma è stata associata al software quindi l'attaccante non può più modificare il software con leggerezza perché non riesce a modificare la firma (perché non conosce la chiave privata usata). Quindi l'unica cosa che può fare l'attaccante è trovare una collisione, per esempio "evilize" che abbiamo visto prima se la firma di un SOFTWARE è basata su MD5, può scambiare i 2 software ad esempio da un playstore. (in realtà il discorso è un po' più complesso manco dall'audio si capisce)

Ci vogliono un paio d'ore per fare l'attacco forza bruta di questo genere.

ESEMPIO

PAPERONE = MARCO RIZZI (talmente ricco che non si alza per venire a lezione)

prendiamo file e firmiamo e succede proprio questo. La mia firma vale anche per un altro documento.

Collide tra tutte le funzioni hash questo perché **è lo stesso documento** e quindi collide con tutte le funzioni hash (in classe altri hanno usato SHA 256 ecc.. noi invece md5)

In realtà però dentro ci sono scritte cose differenti.

Why funziona sta cosa??

Perché abbiamo fatto un file con la stessa struttura pdf di quello firmato da paperone, però ci abbiamo legato un file html con scritto che mi deve dei soldi.

La verifica viene fatta sul pdf quindi risulterà firmato in quanto il controllo viene fatto sulla parte pdf, però al suo interno vi è un file salvato in html, quindi quando lo aprirò sul desktop mi appare la parte che mi dice che gli devo dei soldi. Questo è un attacco al protocollo non alle funzioni, per questo specificare il formato del file utilizzato è importante per evitare attacchi di questo tipo.

ESISTENZA DI FUNZIONI HASH CRITTOGRAFICAMENTE SICURE

La funzione hash è una funzione crittografica quindi bisogna partire dal presupposto che sia pubblica, **non è in sé uno strumento sufficiente per ottenere l'integrità o alcuna proprietà di sicurezza perché non essendoci alcuna cosa segreta nella funzione hash, tutto quello che fa un utente legittimo lo può fare anche un attaccante.** Anche per le funzioni hash come tutte le funzioni, **non si può dimostrare che sia crittograficamente sicura in modo assoluto perché è equivalente a dimostrare che P diverso da NP.** Quelle che si utilizzano finora di funzioni hash si usano perché non esistono attacchi al momento fattibili in breve tempo.

Per ottenere l'integrità si può usare la firma digitale, anche qui con il beneficio del dubbio perché se un giorno uno pubblicasse l'articolo che dimostra che con sha-256 si possono creare collisioni velocemente potrebbe diventare un disastro.

Non succedono così da un momento all'altro di solito, ma ci sono funzioni come abbiamo visto con SHA-1 che si avvicinano l'un l'altra e si comincia a perdere fiducia nelle funzioni prima ancora di essere realmente bucate e si passa ad altro.

C'è il problema che cose importanti potrebbero già essere vulnerabili e fare un upgrade in generale non è così banale sia per i costi sia per il tempo. Ci sono anche sistemi in cui fare aggiornamenti è un problema o non è sempre possibile (esempio centrale nucleare non si può fermare per fare un reboot).

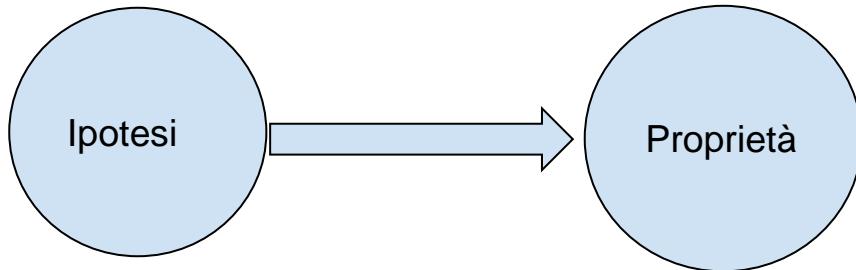
La sicurezza può essere così detta:

Certificational: La certificazione viene dall'uso, viene usato, testato, si provano degli attacchi ma il sistema resiste. Naturalmente questo tipo di sicurezza ha dei limiti molto evidenti. Quello che si vorrebbe è riuscire a dimostrare qualcosa e qui entra il gioco la parte provable e da quello che ci ha detto finora non è che si possono dimostrare le cose in assoluto. (la certificational è quella di cui abbiamo parlato fino ad ora)

Provable: Sulla base di una certa ipotesi, si dimostrano le proprietà di sicurezza del sistema.

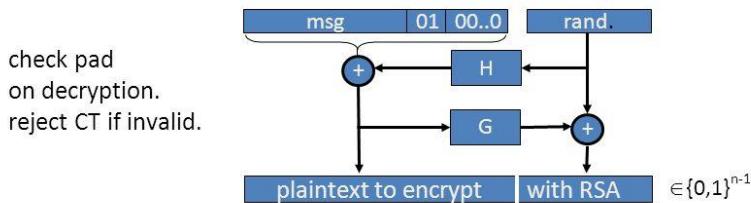
Facendo una certa ipotesi sulle caratteristiche di qualche sistema per qualche funzione matematica che viene utilizzata, si riesce a dimostrare questa proprietà ma non è dimostrata in assoluto poiché l'ipotesi rimane un'ipotesi.

ES:



PKCS1 v2.0: OAEP

New preprocessing function: OAEP [BR94]



Thm [FOPS'01]: RSA is a trap-door permutation \Rightarrow
RSA-OAEP is CCA secure when H, G are *random oracles*

in practice: use SHA-256 for H and G

Dan Boneh

PKCS1 v2.0: OAEP E' un modo di utilizzare RSA per cifrare. Cioè in sostanza il msg è già l'impronta del msg e successivamente usiamo questo pkcs1 v2.0 solo per cifrare con RSA quindi per fare la firma digitale.

OAEP è un altro meccanismo che prevede di concatenare il messaggio con 01 e un PAD di tutti zeri. Viene messo in XOR con una stringa pseudocasuale di cui viene prima calcolato l'hash e si ottiene il testo in chiaro da cifrare (successivamente con RSA), a sto punto viene calcolato l'hash di questo XOR appena fatto e viene messo in XOR con la stringa pseudo. A questo punto tutto viene cifrato con RSA.

In questo modo è reversibile.

Perché utilizzano questa cosa qua? mentre l'altro meccanismo ha una sicurezza Certificational, qui c'è una dimostrazione possibile (è quindi Provable). Se RSA è una funzione che si calcola facilmente ma è difficilmente invertibile **a meno che non si abbia in mano la chiave**, nell'ipotesi in cui H e G siano funzioni hash casuali, allora si dimostra che RSA OAEP è immune a attacchi di tipo **Chosen Ciphertext (CCA)** cioè l'attaccante sceglie il testo cifrato e ottiene il chiaro corrispondente.

Nell'ipotesi che RSA e Hash abbiano queste proprietà si riesce a dimostrare che questo schema è sicuro contro questi attacchi.

Nel '98 è stato proposto un attacco che aveva successo su RSA con PKCS1 v1.5 ma su PKCS1 v2.0 non aveva successo. questo per dire il valore di una dimostrazione di questo genere.

Che RSA abbia proprietà di non invertibilità e che H e G siano funzioni che generano stringhe casuali, sono proprietà che non sono dimostrate, in pratica è usato con SHA-256. (Non lo chiede all'esame, ma è buono da sapere)

Ci teneva a precisare che basandosi su un'ipotesi che risulta valida si riesce a dimostrare un sistema di sicurezza nuovo. Nel giorno in cui si dimostra che RSA non è Certificational allora RSA OAEP sarebbe di conseguenza vulnerabile ad attacchi Chosen Ciphertext. C'è il trade-off: l'efficienza del sistema rispetto alle proprietà di sicurezza. C'è un altro modo che i ricercatori hanno proposto al posto di OAEP che si basa sulle proprietà con un'ipotesi più debole quindi preferibile ma il sistema è complesso e pesante quindi diventa meno utilizzabile rispetto a questo.

Non ci chiederà questo schema ma pretende l'idea di quello che si può dimostrare e non nella sicurezza informatica.

ASPETTI DI INTEGRITÀ:

Integrità H+RSA → Firma Digitale

Siccome la firma digitale ottenuta da RSA è legata al documento, questo offre integrità.

L'integrità entro alcuni limiti si può anche ottenere con dei cifrari a chiave simmetrica: usando un cifrario che rispetti il principio di diffusione, l'attaccante non riesce facilmente a modificare qualche bit del testo cifrato e in chiaro in modo che corrispondano, ma abbiamo visto che dipende molto dalla cifratura che si utilizza e tutte le volte che la cifratura diventa a flusso, l'integrità scompare perché scompare la diffusione, anche con CBC c'è un certo margine perché in certi casi si riesce a modificare qualcosa, altro diventerà illeggibile etc. Nessuno si accorge in realtà che sono alterati e potrebbe essere un problema.

Vogliamo quindi ottenere meccanismi di integrità che sono validi.

Per ottenere l'integrità bisogna quindi pensare a meccanismi validi in generale:

L'idea di utilizzare un cifrario presenta un altro problema: non sempre quando si vuole integrità si vuole anche riservatezza, questo perché avere anche la riservatezza significa che il messaggio arriva cifrato e devo decifrarlo per utilizzarlo. Molto spesso le reazioni degli attuatori devono essere molto veloci, ancora di più con l'IOT i quali spesso sono dispositivi limitati come prestazioni e che devono reagire molto velocemente, il messaggio cifrato introduce un rallentamento non desiderabile, non ha molta importanza nascondere all'attaccante il comando inviato a un dispositivo terminale, ma è molto più importante evitare che l'attaccante modifichi il comando da inviare al dispositivo finale per evitare problematiche peggiori.

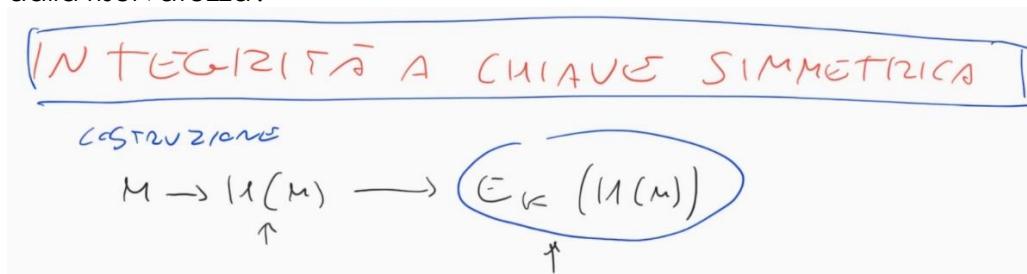
Si deve quindi avere la possibilità di gestirle separatamente.

INTEGRITÀ A CHIAVE SIMMETRICA

RSA È un cifrario a chiave pubblica, ma le operazioni sono lente e i dati che servono per verificare la firma digitale hanno una lunghezza significativa che può essere proibitiva in protocolli di rete: in una rete in cui non c'è una grande ampiezza di banda **può causare appesantimento del traffico quindi anche per quello non è particolarmente desiderabile**.
Quindi si cerca di ottenere l'integrità a chiave simmetrica in modo che sia :

1. Efficiente
2. Separabile dalla riservatezza.

Come si fa a ottenere integrità a chiave simmetrica e in modo che sia efficiente e separabile dalla riservatezza?

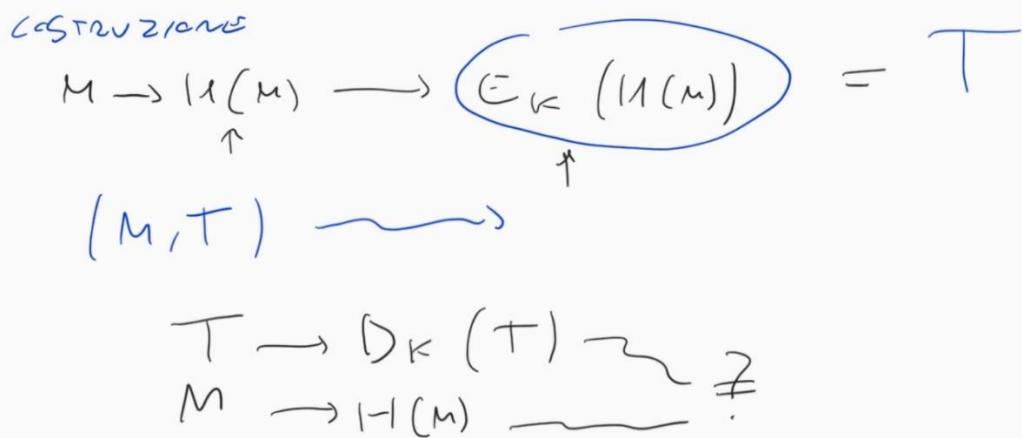


Decidiamo di utilizzare un cifrario a chiave simmetrica (perché a chiave pubblica sarebbe troppo pesante per questo lavoro) per cifrare l'impronta ottenuta usando la funzione hash sul messaggio M:

$$H \rightarrow H(M) \rightarrow E_k(H(M))$$

$E_k(H(M))$ viene chiamato **sigillo di integrità**.

Ci abbiamo messo un hash di mezzo così è possibile cifrare un messaggio lungo più di 256 bit con cifrari a chiave simmetrica che sono molto efficienti. A destinazione come verifico?



Adesso stiamo parlando di integrità senza riservatezza, dall'altra parte arriverà il messaggio M con questo tag T di integrità: (M, T), come faccio a dire se corrisponde?

$$T \rightarrow D_K(T) == M \rightarrow H(M)$$

Ovvero si verifica che, prendendo il tag di integrità ricevuto T e decifrandolo con la chiave K, questo sia uguale all'impronta di M calcolata con la stessa funzione hash.

Rimane comunque il problema della chiave simmetrica da scambiare però.

KEYED HASH O HASH CON CHIAVE:

$$M \xrightarrow{} H(M||S) \rightarrow T$$

$$A^S \qquad \qquad \qquad B^S$$

$$(M, T) \longrightarrow$$

Supponiamo di non voler usare un cifrario, ma di usare solo la funzione hash:

Abbiamo **due interlocutori A e B che vogliono scambiarsi**, oltre ai messaggi, **un segreto S. A invia l'impronta del messaggio M concatenato al segreto S, ovvero il tag di integrità T=M→H(M||S).** T è l'hash del messaggio concatenato (||) con il segreto. A questo punto **sia A che B condividono il segreto, perciò diventano A^S e B^S.**

Ora, se A^S invia (M, T) a B^S, B^S riceve M, concatena con S (che già conosce), lo dà in pasto alla funzione hash e verifica che l'output corrisponda con T.

L'attaccante che vede passare M e T **non può fare niente se non conosce il segreto**, a meno che non effettui un attacco a forza bruta.

Questo meccanismo si chiama proprio **hash con chiave** (Keyed Hash) ed è un meccanismo molto utilizzato.

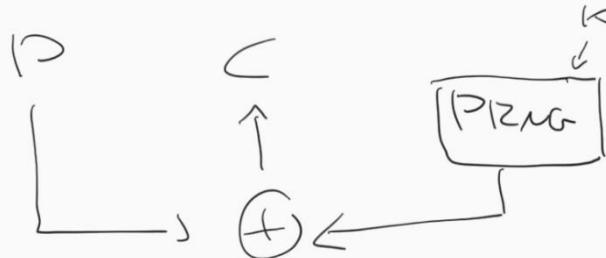
Torniamo in questo esempio.

$$M \rightarrow H(M) \rightarrow E_K(H(M))$$

$$(M, E_K(H(M)))$$

$E \rightarrow$ cifrario a flusso

\rightarrow blocchi cifrati



$$M \rightarrow H(M) \rightarrow E_K(H(M))$$

Vediamo circolare questo: $(M, E_K(H(M)))$

$E \rightarrow$ Se usassimo un cifrario a flusso oppure uno a blocchi con modalità CTR oppure con modalità OFB cos'hanno in comune

OFB e CTR trasformano un cifrario a blocchi in uno a flusso. CTR, OFB e cifrari a flusso si possono rappresentare come generatori di sequenze pseudocasuali.
P corrisponde a $H(M)$, C corrisponde a T.

Questo meccanismo garantisce l'integrità? Cosa può fare l'attaccante?

$$T \oplus H(M) = S$$

Fa

$$T \oplus H(M) = S$$

$$S \oplus H(M') = T'$$

$$S \oplus H(M') = T'$$

$T \oplus H(M)$ cosa ottiene? ottengo il numero casuale generato S. A questo punto crea un suo testo M'. Quindi fa $S \oplus H(M') = T'$ a questo punto attaccante non conosceva la chiave e non la conoscerà mai però siccome è un cifrario a flusso inverte parzialmente facendo $T \oplus H(M)$ e ottiene la sequenza e da lì della chiave non ce ne facciamo nulla. Manda T' , M' al destinatario il quale fa la sua verifica che avrà successo in quanto è stato calcolato dalla stessa sequenza che era stata calcolata con la chiave K.

Quindi se il cifrario è a flusso, il meccanismo non funziona.

Non c'è integrità perché con quel meccanismo può ricifrare una cosa diversa da spedire al destinatario.

Bisogna stare attenti a come si combinano i vari ingredienti.

Questi cifrari sono cifrari in cui avere la coppia testo cifrato e testo in chiaro non va bene. Infatti in questi cifrari salta l'integrità e quindi il meccanismo non funziona con i cifrari a flusso.

Prima avevamo hash e cifratura, qua in un passo solo con l'hash otteniamo integrità

$$M \rightarrow H(M || S) \rightarrow T$$

A^S

B^S

$$(M, T) \rightarrow$$

In questo esempio si mantiene l'integrità **se l'hash è crittograficamente sicura**: l'attaccante dovrebbe fare un doppio attacco brute-force, prima per trovare S e poi su M in modo da poter mandare un altro M'.

Il modo in cui si usa questo schema per ottenere l'integrità è detto HMAC.

HMAC-HASH MESSAGE AUTHENTICATION CODE

MAC è il nome di tutti i sistemi per l'integrità a chiave simmetrica -> Message authentication code /In realtà in alcuni libri si trova MIC dove "I" sta per integrity.

In alcuni libri usano mic in modo più generico.

MAC In chiave simmetrica mentre MIC per tutto anche per la firma digitale, simmetrica e asimmetrica.

H sta per hash perché MAC è basato sull'hash.

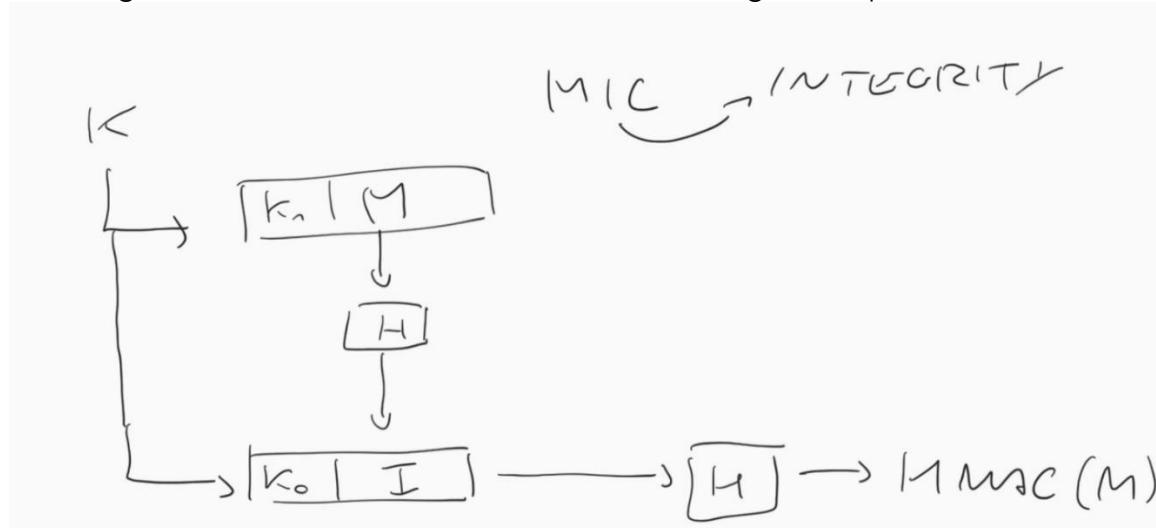
La logica è sempre quella di fare un hash con un segreto.

Utilizza una chiave k_i e una chiave k_o , entrambe queste chiavi derivano da un'unica chiave K.

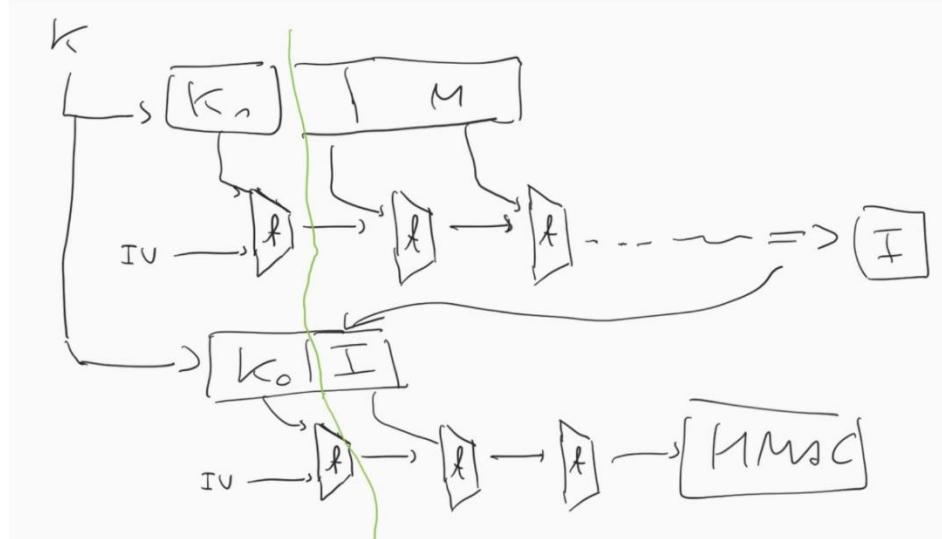
FUNZIONAMENTO HMAC:

HMAC è fatto utilizzando un **doppio hash**:

Date 2 chiavi K_i (input) e K_o (output), per prima cosa si cifra con K_i il messaggio M, poi si calcola l'hashing sul risultato della cifratura ottenendo un tag I , che poi si ricifra con K_o .



Qui sotto possiamo notare lo stesso disegno di sopra, ma più nel dettaglio.



FUNZIONAMENTO HMAC:

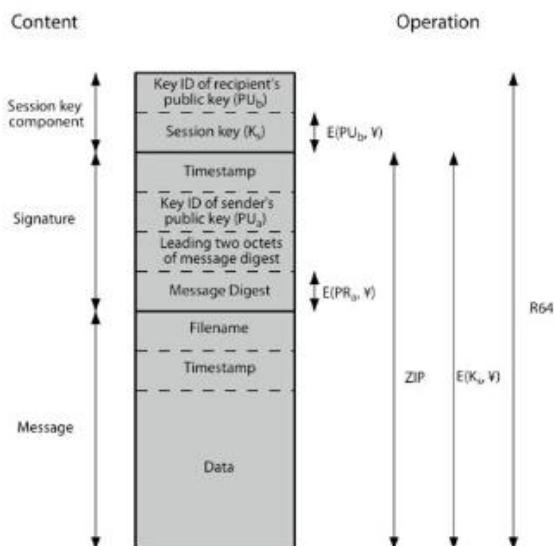
Si utilizza un doppio hash.

1. si concatena il messaggio M alla prima chiave K_i .
2. M e K_i vengono date alla funzione di hashing tante volte quanti sono i blocchi di cui M è composto, e restituisce come output un singolo blocco (il perché è spiegato nel punto 3) "I", ovvero il codice di integrità.
3. Il codice di integrità "I" viene concatenato con la chiave K_o e ne viene nuovamente calcolato l'hash, stavolta con solo 2 passi in quanto "I" equivale a un singolo blocco. Si ottiene così $\text{HMAC}(M)$.

La logica è quella del sistema che abbiamo visto prima, solo che qui viene ripetuta due volte (con K_i e K_o) per proteggere da certi attacchi.

Finché non cambia la chiave la parte a sinistra può essere pre-computata. Con SHA3, l'input della seconda funzione di compressione è l'output della prima funzione di compressione: quindi sostanzialmente si può fare prima l'hash di K_i e poi trattarlo come un vettore iniziale IV diverso per fare l'hash $H(M)$, e lo stesso K_o dopo averlo concatenato con I diventa IV per l'hashing della concatenazione e così via.

Di questo HMAC ne parleremo ancora. Ora vediamo com'è utilizzato in pratica il MAC.



Cosa fa PGP? utilizza la firma digitale e questo non è un problema poiché è un protocollo asincrono e non si accorge neanche che si sta calcolando la firma digitale e dall'altra parte è veloce come controllo. Quindi abbiamo time stamp con data e ora. ID chiave pubblica di chi ha inviato il messaggio per identificare quali delle chiavi del mittente ha usato ,per semplificare la correttezza della chiave. Poi ci sono i primi due byte del digest per vedere che non ci siano errori di verifica. I dati sono formati dai: dati veri e nomefile , timestamp e questi dati del MESSAGE più i dati DELLA SIGNATURE tutto viene zippato e cifrato con chiave simmetrica la quale è cifrata con chiave pubblica del destinatario. ID delle chiavi sono le impronte delle chiavi. E' molto più efficiente verificare l'impronta rispetto a tutta la chiave. La chiave pubblica non è un problema mandarla ma è solo una questione di spazio.

Qui abbiamo visto che su SSL il formato prevede che venga inserito un MAC in fondo e di nuovo il MAC.

RISERVATEZZA E INTEGRITÀ IN CONTEMPORANEA

AEAD-AUTHENTICATION AND ENCRYPTION WITH ADDITIONAL DATA:

https://www.dir.uniupo.it/pluginfile.php/407948/mod_resource/content/0/CTR-GCM-CCMP.pdf

Gli AEAD sono schemi come GCM e CCM, in cui la cifratura avviene con CTR con in aggiunta anche la parte di autenticazione: il contatore 0 viene cifrato e messo in XOR con i dati di autenticazione (additional data), che vengono elaborati con una moltiplicazione in campo finito (GCM) che è un'aritmetica in modulo, e c'è lo XOR del testo cifrato, questo produce un oggetto che è il tag dell'autenticazione. In realtà l'autenticazione viene calcolata sul testo cifrato, mentre in PGP si calcolava sul testo in chiaro.

L'altro modo è CCMP dove si avevano contatori calcolati con AES e messi in XOR con il testo in chiaro e questa è la cifratura con AES-CTR. Il tag è calcolato sul testo in chiaro e cifrato dopo è come PGP. La fase di autenticazione è indipendente dalla cifratura.

TLS utilizza questi schemi e questo si trova se andiamo a vedere security nel sito (es sito qualunque). AES 128 GCM che è il primo che ci ha mostrato. AES- GCM è una modalità di questo tipo dove la parte sotto è GCM e sopra è AES. In realtà questo è quello utilizzato da TLS versione 1.2 o 1.1.

Se vediamo protocol operation di SSL e TLS viene calcolato MAC e poi calcolato tutto quanto. E' un altro schema AEAD.

L'ultimo esempio potrebbe essere SSH che ci dice che utilizza per l'integrità HMAC con SHA2-256. Quindi non usa uno schema AEAD ma il meccanismo di integrità HMAC.

C viene compresso e cifrato e viene calcolato il MAC e giustapposto. al contrario degli altri il MAC non viene mai cifrato ma viene calcolato non sul testo in chiaro ma sul testo di cifratura. Quindi vengono affiancati cifratura e dati per autenticazione ma non vengono mai uniti.

Ultimo esempio potrebbe essere WEP (protocollo Wi-Fi prima di WPA). In questo modo viene cifrato hash oltre che il messaggio e si ottiene anche integrità solo che è tutto cifrato poiché la riservatezza si è pensata necessaria.

Quale vale la pena utilizzare? E' molto difficile fare queste scelte. Uso A and E

Un meccanismo del tipo E e A dice che il destinatario non deve neanche autenticare se sono arrivati corretti ma deve prima verificare l'integrità. Questo A and E separati del WEP permette di separare integrità e autenticazione e permette di [dio cane]

A parte queste applicazioni ci sono dei problemi più profondi per cui alcune di queste modalità sono sicure con certi cifrari perché altrimenti l'uso del MAC sul testo in chiaro può rivelare informazioni sul testo in chiaro. Come al solito in crittografia possono indebolirsi l'una con l'altra. Il punto è che non tutti sono crittografi, quindi hanno pensato ai meccanismi AEAD che offrono riservatezza e autenticazione in un pacchetto unico e pensati come soluzione in tutte queste trappole di cifratura autenticazione che sono un reale problema. Offrono un meccanismo preconfezionato che evita problemi di questo genere e utilizzabili con queste primitive. Mai cercare di fare da sé.

Manca IPSec che ci dice che gli algoritmi di cifratura con algoritmi di autenticazione sono HMAC e MD5 HMAC SHA1 E HMAC SHA2. Però il meccanismo in questo caso è E & A. Non dobbiamo sapere queste cose ma ci fa vedere esempi sull'uso di questi meccanismi e si discutono i rischi e introduzione a AEAD.

Conviene utilizzare i meccanismi AEAD? Si offrono riservatezza e autenticazione in un colpo solo.

LABORATORIO → IMPLEMENTARE HMAP

https://www.dir.uniupo.it/pluginfile.php/407938/mod_resource/content/7/hmac.pdf

Ci fa fare una cosa che non dovremmo mai fare poiché non si implementano sistemi crittografici.

HMAC prende una chiave *in input* ma usa due chiavi: k_i e K_o . Le chiavi devono essere di una lunghezza prefissata che è b bit e per questo si può fare preelaborazione della chiave. Se la chiave è più lunga se ne calcola l'impronta hash e si ottiene una lunga n ed è molto facile che sia più corta di b e vengono aggiunti 0000 in fondo. Qualunque sia la chiave *in input* consente di partire da una chiave lunga b .

Da questa chiave si ottengono chiavi K_i e K_o che si chiamano i_key_pad e o_key_pad . K_i viene messo in XOR con una stringa prefissata dall'algoritmo che sono tu 36 esadecimale e a destra si mette in XOR con 5c in esadecimale e si ottengono le due chiavi.

Classe java Message Digest serve per implementare la funzione hash. Le lunghezze dei blocchi le otteniamo dalla slide.

Cosa succede della chiave? la leggiamo come byte array da file e se troppo lunga ne calcoliamo il digest e facciamo hash digest e restituisce impronta con chiave k.

Basta fare un ciclo e non serve avere un array lungo b che contiene tutti i bit che contiene 0x36 ma basta ripetere n volte.

Calcoliamo hash di i_key_pad concatenata al messaggio e poi calcolare hash ma è uno sforzo inutile siccome i_key_pad è lunga un blocco basta fare impronta e poi concatenare con tutti gli altri.

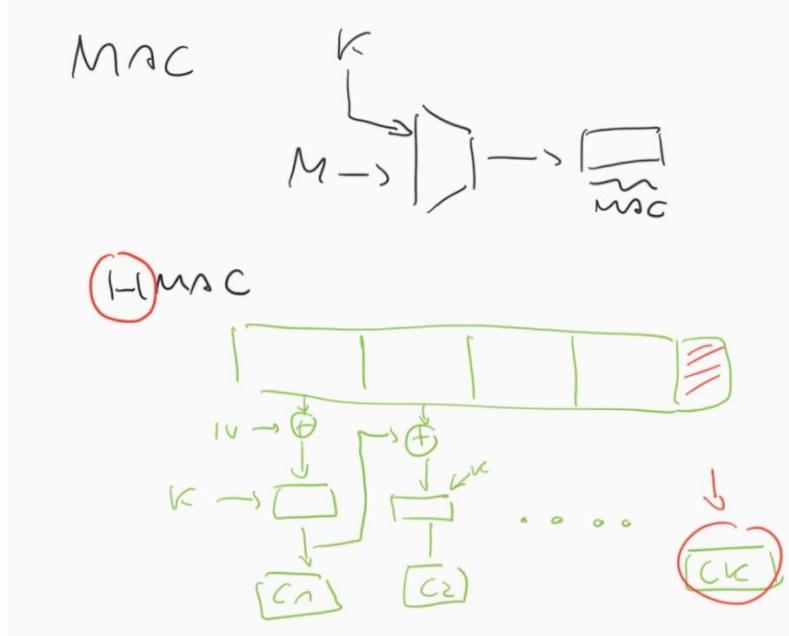
Facciamo void update della chiave e poi void digest di tutto il resto.

La stessa cosa col secondo MAC. Basta fare hash di update e hash digest.

Calcoliamo HMAC in questo modo e usiamo HMAC di java e costruiamo chiave con SecretKeySpec e specifichiamo l'algoritmo e ne scegliamo uno di quelli e la classe Mac la otteniamo con getInstance dello specifico algoritmo e otteniamo HMAC. Inizializziamo HMAC dando la chiave e con do finale facciamo questa operazione lo confrontiamo con quello che abbiamo fatto.

MAC, HMAC CONTINUA

I MAC sono algoritmi a chiave simmetrica che usano certi schemi per mantenere l'integrità, in particolare abbiamo visto l'hash con chiave e l'HMAC. Più in generale un MAC è strutturato così:



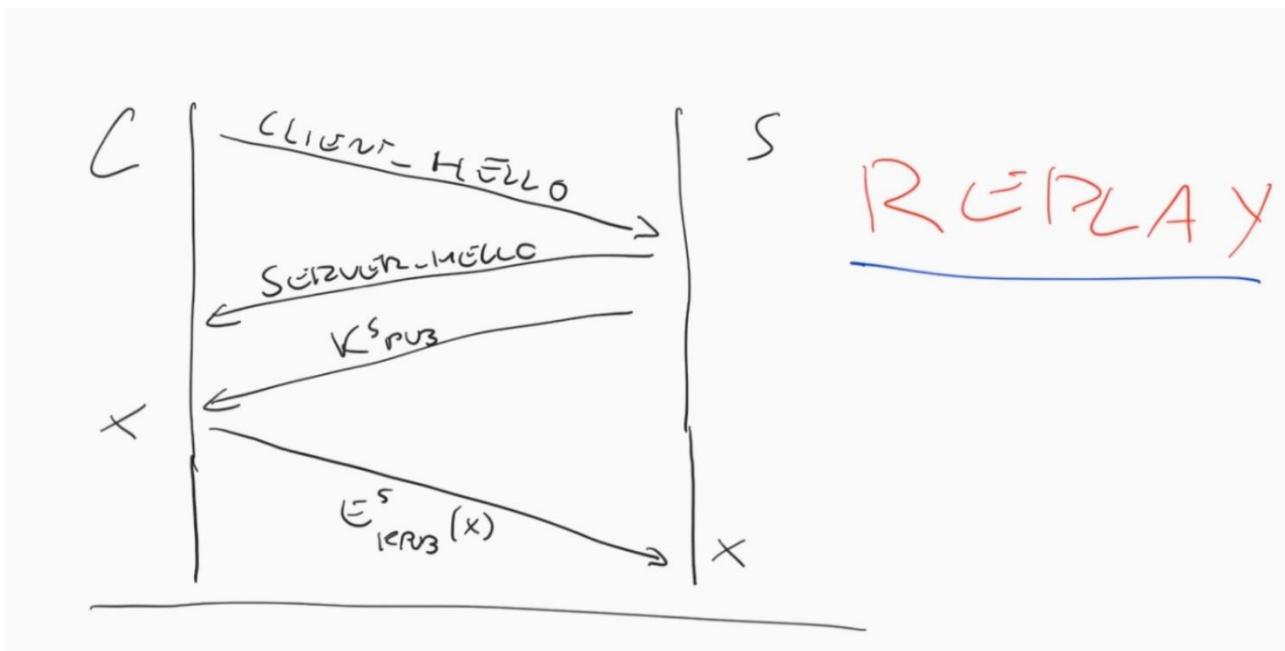
Prende in input una qualunque quantità di dati e una chiave, e restituisce un impronta di lunghezza prefissata. L'abbiamo visto su HMAC perché la H sta per funzione di hash e proprio utilizzando la funzione di hash, la lunghezza di output è prefissata. Un MAC può essere basato su CBC: ad esempio se proviamo a determinare il MAC di una stringa si prendono i blocchi, eventualmente si completa il blocco finale. Ciascun blocco viene cifrato con un cifrario K a chiave simmetrica e si ottiene il testo cifrato che viene rimesso in XOR con il secondo blocco del testo in chiaro e così via.

In partenza c'è IV iniziale. Alla fine l'ultimo blocco del testo cifrato è un testo cifrato C_k il quale contiene una proprietà: se anche uno cambia un bit nel primo blocco del testo in chiaro, anche C_k ne risente. Questo perché **C_k è legato a tutto il testo in chiaro a causa del meccanismo di concatenazione.**

È adatto ad essere utilizzato per **l'integrità: qualunque modifica su qualunque punto del testo in chiaro si riflette anche nell'ultimo blocco del testo cifrato**. Questo è anche grazie al principio di diffusione di cui ce ne sono 2 tipi: se *il cifrario è buono, la diffusione si ha a livello di blocchi*, poi c'è la propagazione dovuta a CBC. Utilizziamo C_k , perché sull'ultimo blocco sappiamo che qualsiasi modifica ad un bit viene riflessa sull'ultimo blocco di testo cifrato, ha la caratteristica di poter essere un MAC perché ha lunghezza prefissata che dipende dal cifrario. La chiave viene ripetuta ad ogni passo.

Nella pratica, è utilizzato di più l'HMAC perché questo che abbiamo visto ora essendo un cifrario vuol dire che è invertibile. Anche se la chiave viene mantenuta segreta, rispetto al meccanismo basato sull'hash, questo ha il problema che il fatto di essere invertibile lo rende più manipolabile di una funzione hash, inoltre è più pesante dell'hash perché un procedimento di cifratura è più pesante del calcolo dell'hash.

SSL/TLS HANDSHAKE



TLS prevede una fase di handshake durante la quale i due sistemi negoziano la crittografia da usare e si scambiano una chiave di sessione che verrà poi usata per cifrare i dati a chiave simmetrica. In questa fase si fa anche l'autenticazione di client e server.

Nella fase di handshake, il client si rivolge al server in una prima richiesta di connessione mandandogli una stringa CLIENT_HELLO random (e le capacità crittografiche) generata dal client e inviata in chiaro.

Il server risponde con le proprie capacità crittografiche e manda la propria stringa SERVER_HELLO random. A questo punto il client genera una stringa pseudocasuale (che sarà la chiave simmetrica), il server gli invia la propria chiave pubblica RSA e il client cifra la stringa pseudocasuale appena generata usando la chiave pubblica che il server gli ha mandato e rimanda il tutto al server. A questo punto entrambi condividono un segreto, e ciò permette loro di usarlo come chiave di sessione.

E' pericoloso usarla direttamente come chiave di sessione perché introduce nuovi attacchi, ad esempio quello REPLAY che funziona in questo modo: se un man in the middle riesce a sniffare il pacchetto con la chiave pubblica del server, può inviare lui il messaggio contenente la chiave pubblica del server e ripetere i messaggi che ha visto passare dal server al client (per questo REPLAY).

Attaccante quindi replica i messaggi che ha visto passare senza alterarli, ma sa l'effetto che avranno questi messaggi e li replica.

Quindi avviene tutto senza alterare i messaggi, però magari l'attaccante sa che effetto hanno questi messaggi e per questo li ripete a suo vantaggio.

Per evitare questo tipo di attacco è importante che la chiave di sessione (di cifratura) sia legata a una sola sessione e non ripetibile da una sessione ad un'altra.

SSL (SECURE SOCKET LAYER)

SSL: chiave di sessione

```
master_secret = MD5(pre_master_secret || SHA('A' || pre_master_secret ||
                                              ClientHello.random || ServerHello.random)) ||
MD5(pre_master_secret || SHA('BB' || pre_master_secret ||
                                              ClientHello.random || ServerHello.random)) ||
MD5(pre_master_secret || SHA('CCC' || pre_master_secret ||
                                              ClientHello.random || ServerHello.random))

key_block = MD5(master_secret || SHA('A' || master_secret ||
                                              ServerHello.random || ClientHello.random)) ||
MD5(master_secret || SHA('BB' || master_secret ||
                                              ServerHello.random || ClientHello.random)) ||
MD5(master_secret || SHA('CCC' || master_secret ||
                                              ServerHello.random || ClientHello.random)) || ...
```

Parliamo prima di SSL:

SSL riceve la "x" ovvero la premaster_secret, che viene concatenata con varie elaborazioni di funzioni HASH (SHA, MD5) della stringa x, della stringa random mandata dal client e della stringa random mandata dal server. In questo modo con una serie di iterazioni, viene costruito il mastersecret e con un'operazione analoga anche la key_block.

Cosa si ottiene in questo modo?

In questo modo il blocco di chiave è costruito sia dal segreto condiviso (che attaccante non conosce) che le due stringhe che l'attaccante può conoscere che dipendono dalla sessione. Quindi la chiave di sessione prodotta anche se il client fosse malevole dipende anche da dati che ci ha messo il server e quindi la chiave di sessione sarà diversa da quella di un'altra sessione.

Parliamo ora di TLS:

TLS (TRANSPORT LAYER SECURITY)

TLSv1.1, 1.2 e 1.3: chiave di sessione

(RFC 4346 - apr 2006; RFC 5246 - ago 2008; IETF Internet Draft apr 2014)

- key-block = PRF (MasterSecret, "key expansion",
ServerRandom || ClientRandom)



(lunghezza arbitraria)

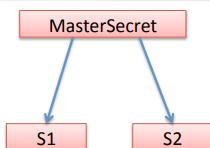
Il blocco di chiave viene generato da una PRF (funzione pseudo casuale) che dipende dal mastersecret e le stringhe Server_random e Client_random, la key_expansion server per dare struttura.

PRF

- PRF (secret, label, seed) =
P_SHA256(MS, label || seed)

```
secret = MasterSecret  
label = "key expansion"  
seed = ServerRandom || ClientRandom
```

- In TLS v1.1 era:
PRF (secret, label, seed) =
P_MD5(S1, label || seed) ⊕
P_SHA-1(S2, label || seed)



Questa PRF è basata sostanzialmente su HMAC che è basato su una funzione hash (SHA256 prima era MD5). La funzione P_hash parte da un segreto e dal seme e calcola una sequenza di HMAC dal segreto di A(i) concatenato col seme.

Il seed è dato da key expansion, server_random e client_random.

- P_hash(secret, seed) = $\xrightarrow{\text{(lunghezza arbitraria)}}$
 $\text{HMAC_hash}(\text{secret}, \text{A}(1) \parallel \text{seed}) \parallel$
 $\text{HMAC_hash}(\text{secret}, \text{A}(2) \parallel \text{seed}) \parallel$
 $\text{HMAC_hash}(\text{secret}, \text{A}(3) \parallel \text{seed}) \parallel \dots$

secret è S1 oppure S2
 seed="key expansion" || ServerRandom || ClientRandom

$$A(0) = \text{seed} \quad A(i) = \text{HMAC_hash}(\text{secret}, A(i-1))$$

hash = SHA256 (era MD5 oppure SHA-1 per TLSv1.1)

Il segreto viene fuori dal master-secret. Pre_mastersecret è quello scambiato mentre master secret è un segreto già elaborato che contiene server e client random. L'idea è che può essere usato in sessioni diverse per riavviare una sessione diversa ricostruendo un blocco di chiavi diverso da quello precedente senza tornare indietro a ricondividere il segreto e rendendo efficiente la fase di handshaking. SSL usa funzioni hash poiché sappiamo che HMAC è fatto con funzioni Hash.

Il keyblock è una stringa della lunghezza giusta che è più o meno lunga e le varie applicazioni di HMAC vengono poi concatenate con HMAC successivo e questa stringa può essere di una lunghezza arbitraria.

Perché lunga e quanto deve essere lunga questa stringa?

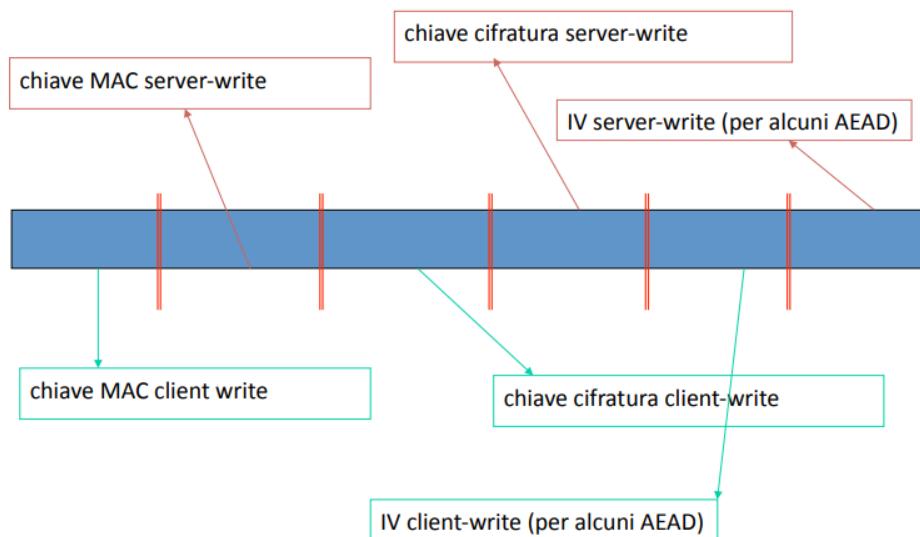
Una volta prodotto il materiale per la chiave viene spezzettato e il primo segmento viene usato come chiave MAC client write. Quando un Client scrive e manda un messaggio ad un server calcola il proprio MAC usando questo come chiave. Quando il server risponde MAC server-write usa questa chiave.

La chiave cifratura: quando client scrive, cifra con questa chiave e lo stesso il server cifra con questa chiave e poi se serve vengono usati i segmenti successivi iniziali (IV) che prevedono uso di un vettore iniziale.

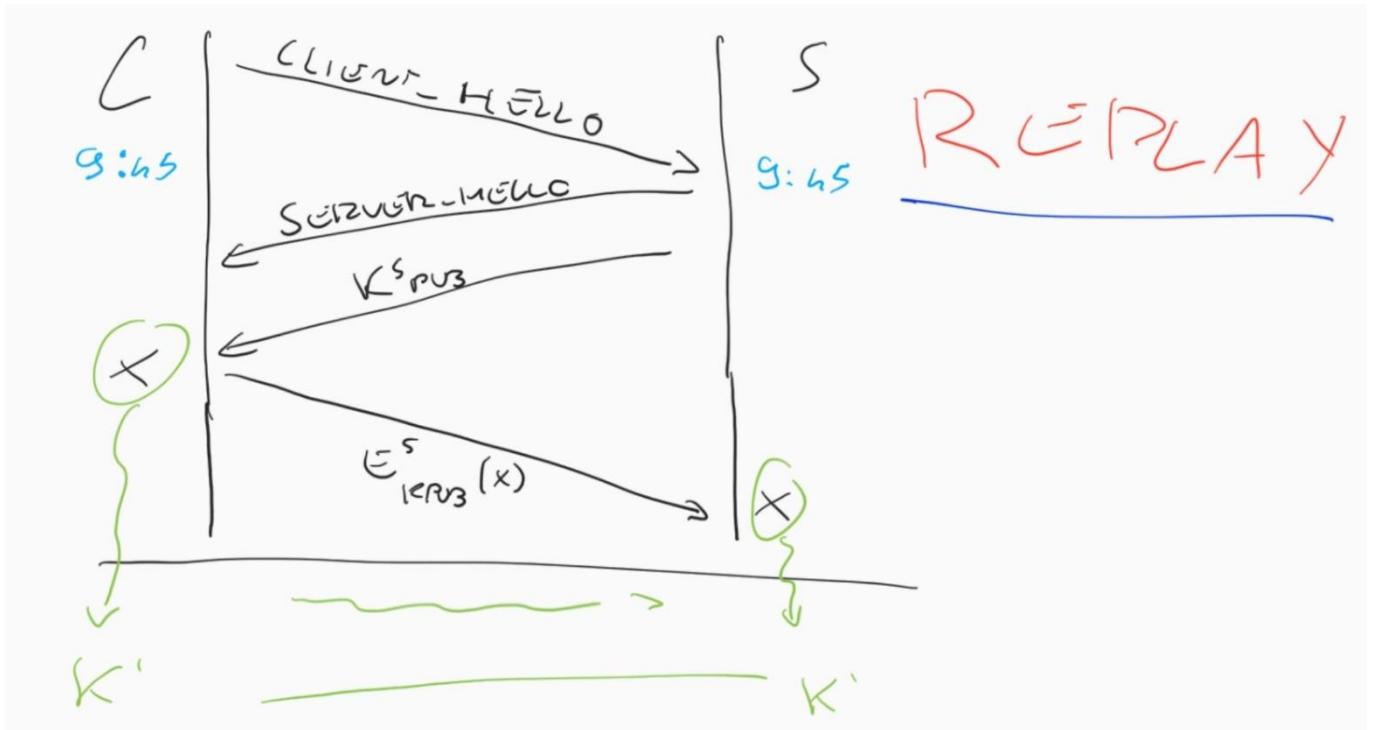
Vede tutto il materiale di chiave prodotto per la sessione TLS.

E' da notare l'uso delle stringhe pseudo casuali per legare la chiave alla sessione.

chiave di sessione



Legare le chiavi o il messaggio alla sessione è importante per contrastare attacchi replay, si può fare sia con stringhe generate una volta per sessione e poi buttate via (NONCE), oppure con la data e l'ora direttamente. Farlo con data e ora ha il vantaggio che non è necessario scambiarsi niente, ma in questo modo client e server devono essere sincronizzati. Si avrebbe garanzia che master secret e blocco di chiavi dipendono dal quella sessione perché semplicemente sono fatte generate in quell'ora e momento preciso. Il problema è che un'operazione di questo genere prevede che client e server siano sincronizzati.



TLS non usa questo meccanismo perché ci può essere latenza (da una parte all'altra del mondo), fusi orari diversi e si vuole per interoperabilità se uno ha una macchina dove data e ora siano sbagliati riesce comunque a legare la chiave e il materiale crittografico a quella sessione specifica. Usare data e ora ha il problema in cui c'è una latenza della rete e si presuppone un certo intervallo in cui data e ora possono variare, c'è quindi un trade-off poiché se uno usa un intervallo ampio permette attacchi replay mentre se non è ampio taglia fuori dei client di latenza più alta ed è un problema.

Per questo TLS usa le stringhe pseudocasuali e risolve il problema.

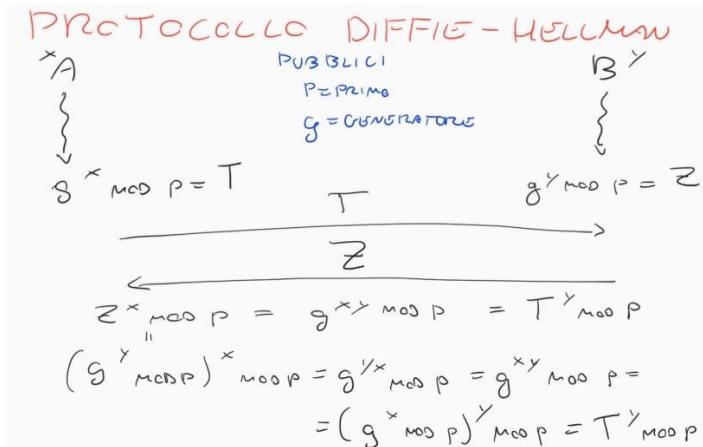
Torniamo indietro nel disegno. Il segreto x è stato scambiato ed entrambi lo condividono. Il pre-mastersecret e il master secret non viaggeranno più in rete e ciascuno farà le proprie operazioni e costruiranno entrambi una chiave K sia il client che il server e poi la condivideranno.

Il protocollo SSL è un protocollo di comunicazione di errore.

Se per C è 7.30 mentre per S è 9.45 le chiavi saranno diverse. Se fossero sincronizzati entrambi con 9.45 concorderebbero sulla chiave giusta e l'attaccante che prova a fare attacco replay, se usa chiave K' , il server non l'accetterà e potrebbe esserci un momento in cui la chiave usa stessa data e ora e il server potrebbe adattarsi.

Il vantaggio di usare data e ora è che non va scambiata: tutti sanno che ore sono, mentre la stringa pseudocasuale va per forza scambiata. Se si decide che siano sincronizzati al minuto allora si prende ora nei dieci minuti in modo che tutti useranno la stessa data e ora. Più si accetta arrotondamento più si lascia all'attaccante la possibilità di un attacco di tipo replay. Supponiamo che X venga usata come chiave. Attaccante manda handshake al server e comincia a replicare esattamente quello che ha visto e replay gli scambi che ha visto passare tra C e S. Attaccante pensa di fare danni a caso oppure sa che funzione ha quell'operazione e sfrutta quella cosa lì. Per esempio compravendita di titoli sta replicando la richiesta di questi titoli. Lo vedremo ancora di più col prossimo sistema crittografico che vedremo ovvero il protocollo di Diffie-Hellman.

PROTOCOLLO DI DIFFIE-HELLMAN:



Supponiamo di avere A e B che si mettono d'accordo su dei parametri pubblici P,G dove P è primo e G un generatore.

FUNZIONAMENTO DIFFIE-HELLMAN

1. A sceglie un segreto X;
2. B sceglie un segreto Y;
3. X e Y devono essere $< p-1$;
4. A calcola $T = g^x \text{ mod } p$;
5. B calcola $Z = g^y \text{ mod } p$;
6. A manda a B la sua T e B risponde con la sua Z ;
7. A calcola $Z^x \text{ mod } p$ e B calcola $T^y \text{ mod } p$ i quali sono entrambi uguali a $g^{xy} \text{ mod } p$, questi non transitano sulla rete ma ognuno se li calcola per i fatti suoi.

Il fatto che sia A che B possono calcolarsi $Z^x \text{ mod } p$ e $T^y \text{ mod } p$ rispettivamente è perché $Z^x \text{ mod } p = (g^y \text{ mod } p)^x \text{ mod } p = g^{xy} \text{ mod } p$. Anche gli elevamenti a potenza in modulo valgono nella proprietà commutativa quindi $g^{yx} \text{ mod } p = g^{xy} \text{ mod } p = (g^x \text{ mod } p)^y \text{ mod } p$ che non è altro che T.

Nessuno ha rivelato il proprio segreto passandolo sulla rete, ma hanno calcolato entrambi la stessa cosa.

L'unica cosa che l'attaccante può vedere sono T e Z. Può ricavarne X e Y rispettivamente? Potrebbe fare un attacco a forza bruta dove prova tutte le x possibili finché non trova $g^x \text{ mod } p = T$. Se rendo T abbastanza lungo, diventa più facile evitare il problema di un attacco forza bruta (ma di conseguenza anche p deve essere molto lungo). Però come per il caso di RSA **l'attacco forza bruta non è l'attacco più efficiente** che può fare l'attaccante: questo perché (come per RSA) DH si basa su un'operazione matematica che quindi avrà una funzione inversa che l'attaccante cercherà di sfruttare, e l'inversa in questo caso è un LOGARITMO DISCRETO:

$T = g^x \text{ mod } p$, dato T e noti g e p determinare x

IL PROBLEMA DEI LOGARITMI DISCRETI

$$T = \underbrace{g^x}_{\text{mod } p}$$

$|p| = 1024$
 2048
4096 bit

Dato T e noti g e p

Determinare x

PROBLEMA DEI LOGARITMI DISCRETI

$$b = a^x \quad x = \log_a b$$

$$b = a^x \quad x = \log_a b$$

Il problema dei log discreti risulta al momento equivalente al problema della fattorizzazione. E' un problema sub esponenziale (quindi risolvibile con un algoritmo sub-esponenziale!) e ciò implica che l'attaccante non farà mai attacco a forza bruta e andrà a provare a risolvere il problema dei logaritmi discreti (poiché costa meno).

Questo vuol dire che i parametri devono essere molto lunghi per rendere l'attacco più difficile.

Quindi $|p| = 1024, 2048$ e **4096 bit** così come per RSA, ed è sub esponenziale:

$$\mathcal{O}(2^{\sqrt{n}})!!!$$

Nonostante calcolare un normale logaritmo sia facile, **al momento non è noto un algoritmo efficiente (quindi polinomiale) per calcolare il logaritmo discreto. L'attaccante non deve necessariamente ricavare X da T : se riesce a ricavare X e Y riesce a calcolare il segreto come fosse A.** Potrebbe in linea di principio trovare un modo di combinare T e Z senza risalire a X e Y ma nessuno ha mai trovato un metodo che non passi per i log discreti.
E' efficiente ma siamo sempre nel mondo della crittografia a chiave pubblica e non sarà mai efficiente come una a chiave simmetrica.

=====!!!!!!IMPORTANTE!!!!!!=====

DIFFIE-HELLMAN NON E' UN CIFRARIO! E' un protocollo per lo scambio di un segreto su canale pubblico.

Noi ci fidiamo del canale, non c'è alcuna ipotesi che A e B condividano un segreto a priori, ma sicuramente alla fine del protocollo ne condivideranno uno.

Ora passiamo a definire cos'è questo generatore g:

$$\begin{aligned}g \text{ genera } & \{1, 2, \dots, p-1\} \\& \equiv \\& \left\{ g, g^2 \bmod p, g^3 \bmod p, \dots, g^{p-1} \bmod p \right\} \\p = ? & \quad g = 3 \\& \left\{ \begin{array}{l} 3, 2, 3^2 \bmod 7 = 6, 3^4 \bmod 7 = 4, 3^5 \bmod 7 = 5, \\ 3^6 \bmod 7 = 1 \end{array} \right\}\end{aligned}$$

Il generatore G è un **generatore di un insieme di numeri che sono compresi tra 1 e p-1**, cioè un generatore che genera $\{1, 2, \dots, p-1\}$.

Questo vuol dire che l'insieme $\{g, g^2 \bmod p, g^3 \bmod p, \dots, g^{p-1} \bmod p\}$ è uguale all'insieme $\{1, 2, \dots, p-1\}$ questo però in ordine sparso.

ESEMPIO:

Prendiamo $p=7$ e $g=3$

Abbiamo l'insieme:

$$\{3, 2, 3^3 \bmod 7 = 6, 3^4 \bmod 7 = 4, 3^5 \bmod 7 = 5, 3^6 \bmod 7 = 1\}$$

sono generati tutti i valori tra 1 e 6 (cioè fino a $p-1$).

Se riusciamo ad avere un generatore di questo tipo lo spazio dei valori è il massimo possibile poiché contiene tutti i valori tra 1 e $p-1$ però potrebbe non generarli tutti.

Per esempio:

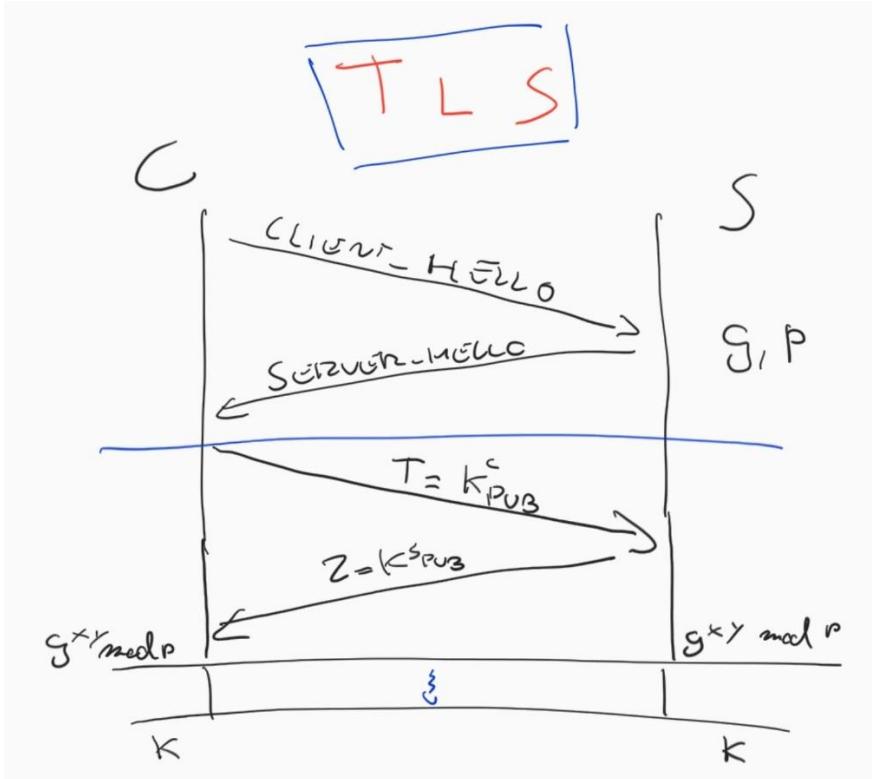
$$p=7 \quad g=2$$

$$\{2, 4, 1, 2, 4, \dots\} = \{2, 4, 1\}$$

Quindi a volte un generatore non genera tutti i valori. Questa scelta potrebbe creare il problema che l'attaccante non debba cercare tra 7 valori ma sono la metà. Nello scegliere il generatore quindi conviene prima verificare che generi tutti i valori possibili. Per DH esistono una scelta di p e 5 generatori prefissati e si usano quelli come standard.

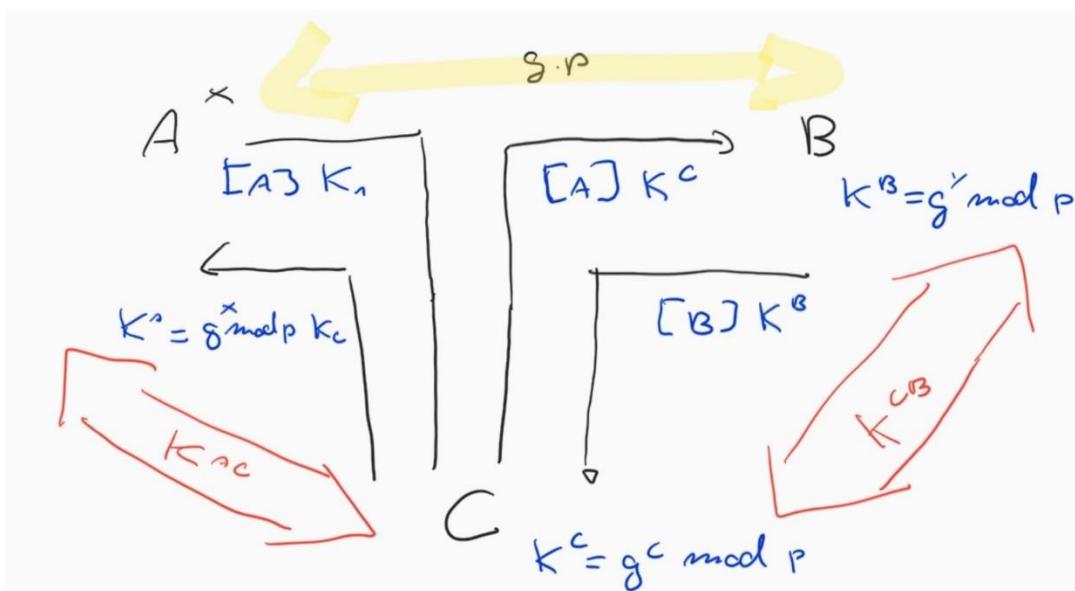
Finché A e B non modificano il loro segreto, quello condiviso rimane sempre lo stesso. Quindi non si usa mai un segreto condiviso come chiave.

SCAMBIO DI CHIAVI BASATO SU D- H CON TLS



1. Client e server si scambiano **C_hello** e **S_hello** e le capacità crittografiche e decidono di usare D-H per scambiare le chiavi, e in questa fase si scambiano anche **p** e **g**.
2. Si inviano **T** e **Z** e sia **A** che **B** si calcolano $g^{xy} \text{ mod } p$. **T** e **Z** sono le chiavi pubbliche dei 2 sistemi e in base a d esse si costruiscono il segreto condiviso che però è scelto sia dal client che dal server quindi in questo modello (rispetto a RSA che sceglieva solo il client) il segreto è più forte.
C'è un problema: finché non cambiamo le chiavi DH il segreto condiviso è sempre lo stesso; però dato che la generazione della chiave viene fatta grazie alle stringhe random (client_hello e server_hello) **il sistema genera una chiave diversa anche se il segreto condiviso è lo stesso, perché usa stringhe random diverse per ogni sessione.**
3. Ad ogni diversa sessione verrà generata una chiave diversa anche se il segreto è lo stesso. Qui di nuovo attaccante potrebbe replicare la chiave pubblica e poi replicare i messaggi che vengono dopo cifrati con $g^{xy} \text{ mod } p$.
Qui c'è la fase di collisione della chiave che coinvolge le stringhe pseudo casuali quindi la chiave dipende in modo specifico dalla sessione.

PROTEGGERSI DA MAN IN THE MIDDLE

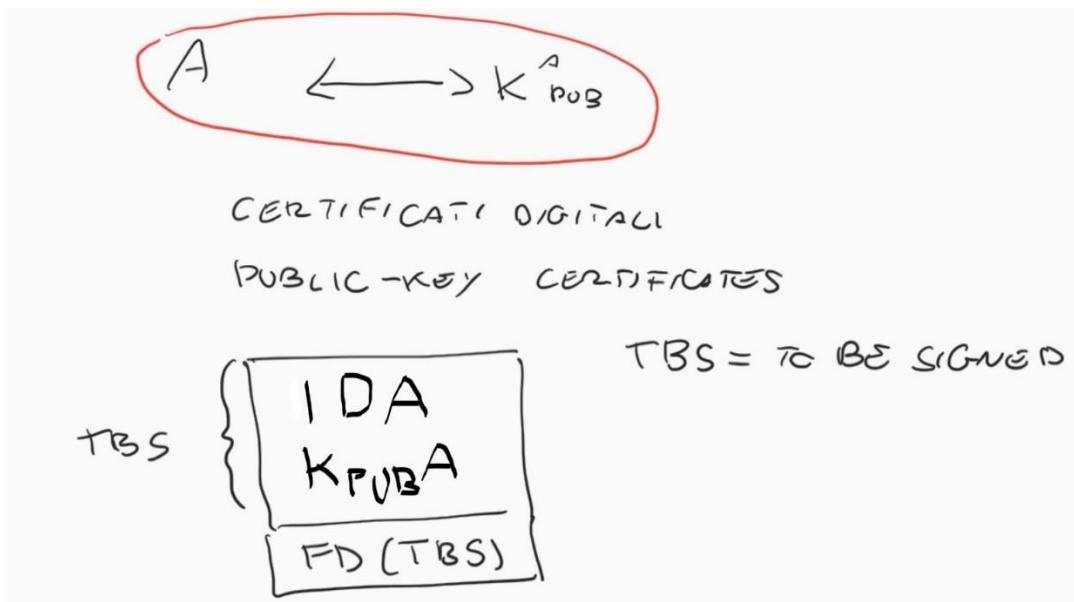


Nulla impedisce ad un C di intercettare il traffico che manda A quindi un certo K^a . C ha la sua Z che genera la sua K^c e la manda a B dicendo di essere A. Quindi B risponde a C(pensando che fosse A) con i suoi dati. C intercetta questo messaggio e costruisce un segreto condiviso con B quindi B e C hanno chiave condivisa K^{cb} con cui comunicare e lo stesso avrà con A quindi un canale AC condiviso con A K^{ac} . A e B non lo sanno ma A e B hanno l'impressione di aver costruito un proprio canale privato tra di loro.

Non c'è alcun attacco alla crittografia se non il fatto che nella fase di scambio di chiavi C si è intromesso ed è riuscito ad impersonare B per A e A per B, stabilendo quindi due canali (uno tra A e C, l'altro tra C e B). Questo è un attacco Man in the middle ed è possibile poiché non c'è nessun tentativo di autenticare la sorgente o dimostrare integrità delle chiavi che passano le quali essendo pubbliche possono essere rese pubbliche quindi è lo stesso problema che abbiamo incontrato con RSA.

Come ci si protegge da questi attacchi Man in the middle?
Il modo è l'autenticazione della chiave pubblica

AUTENTICAZIONE DELLA CHIAVE PUBBLICA



Vuol dire trovare un modo per **legare la chiave pubblica di A all'utente A in modo che sia difficile rompere il legame oppure forgiare un legame con un'altra entità**. Come si fa a fare questo? **si usano i certificati digitali o di chiave pubblica cioè i Public-Key Certificates.**

Qual è l'idea di un certificato?

Pensiamo all'idea di carta d'identità. Quello che succede è che viene controllata la foto della mia carta d'identità con il mio viso allora viene accettato.

Uno potrebbe anche rivolgersi sempre ad un'autorità (come l'anagrafe) chiedendo se la chiave pubblica che riceve è quella veramente di "A" e questo è la stessa cosa di essere portati in anagrafe quindi è un collo di bottiglia abbastanza grave.

Quello che si fa è chiedere la carta d'identità quindi poiché è una cosa che si può controllare (viso) e si possono controllare i dati (i dati legati dal viso). Questo è anche perché la carta d'identità è difficile da falsificare.

Nel mondo digitale come si costruisce questo certificato?

Non sono una cosa spaziale ma banale e per capire la logica bisogna capire bene la funzione dei certificati che è quella di legare un soggetto A alla chiave pubblica K^A_{pub} .

Un certificato quindi non fa nient'altro che generare un'associazione tra un soggetto A e la sua chiave pubblica K^A_{pub} . Nel mondo digitale il possesso del certificato è secondario, poiché uno può fare una copia esatta del documento. Come nel mondo reale chi ha una fotocopia non ruba l'identità.

Quindi deve essere un documento pubblico e deve essere poter essere in mano a chiunque perché tutti devono poter verificare.

Quindi perché queste due informazioni siano legate devono essere presenti tutte e due sul certificato quindi **il certificato di una chiave pubblica di A deve contenere assolutamente l'ID di A e la chiave pubblica di A** (senza queste due non serve assolutamente a niente).

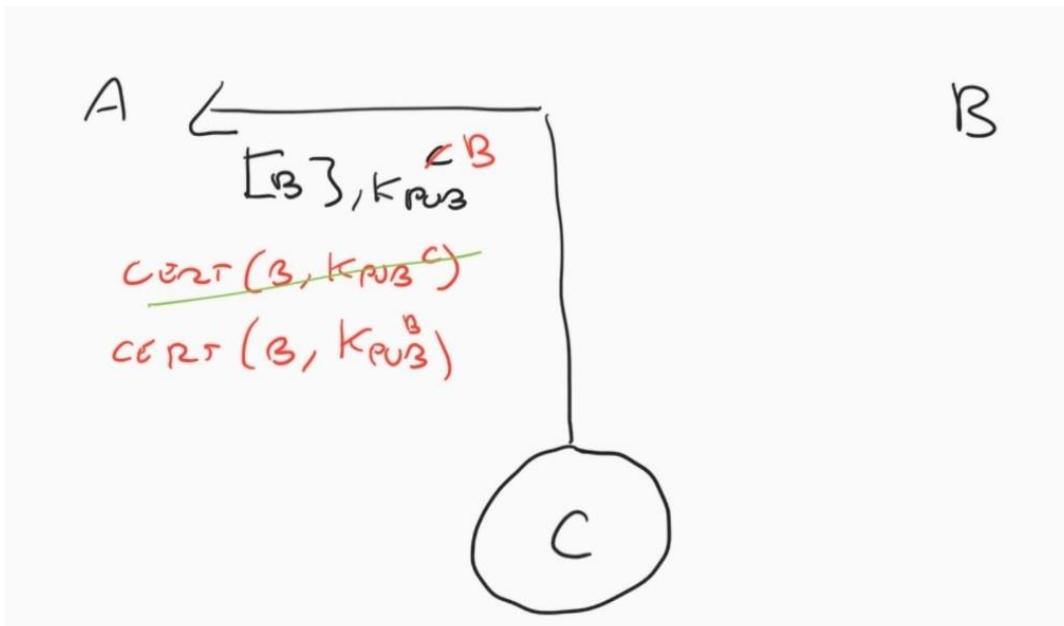
Queste informazioni devono poi anche essere legate.

Come in carta d'identità è tutto su carta e non si può sostituire la foto altrimenti si strappa tutto ed è difficile. E' possibile attaccare l'integrità del certificato, perciò deve essere molto difficile prendere il certificato di A e alterarne il contenuto, ma allo stesso tempo vogliamo

Che chiunque possa verificare che il contenuto del certificato non sia stato alterato: **deve essere facile la verifica e difficile l'alterazione** (nessuno deve essere in grado di modificare la chiave pubblica o il ID).

Per fare ciò si potrebbe usare il meccanismo della firma digitale. C'è la firma digitale fatta con una chiave privata, ciò va bene perché prende la firma digitale di TBS (To be signed) che è la parte che deve essere firmata (guarda disegno). Usando RSA abbiamo visto come firmare, cioè facendo l'impronta hash del messaggio e poi cifrarlo con chiave privata.

Ciò risolve il fatto che chiunque può verificare una chiave privata avendo in mano la chiave pubblica associata (se la privata non è stata alterata). Se la funzione di hash è robusta e l'algoritmo di firma ha tutte le proprietà che vogliamo, la firma funziona correttamente e si ha la certezza che la chiave pubblica di A sia associata a quella privata.



Nel contesto di questo attacco (il Man in the middle visto prima), se a C si chiede il certificato della chiave pubblica che ha mandato ad A, allora C non può più sostituire la K_{pub} di B con la sua:

A vuole vedere un certificato che attesti che la chiave pubblica ricevuta è quella di B, quindi è difficile falsificare il certificato, perché C può sì mandare il certificato di B con dentro la chiave pubblica di C, ma A verifica che la chiave specificata nel certificato è diversa da quella che gli è arrivata.

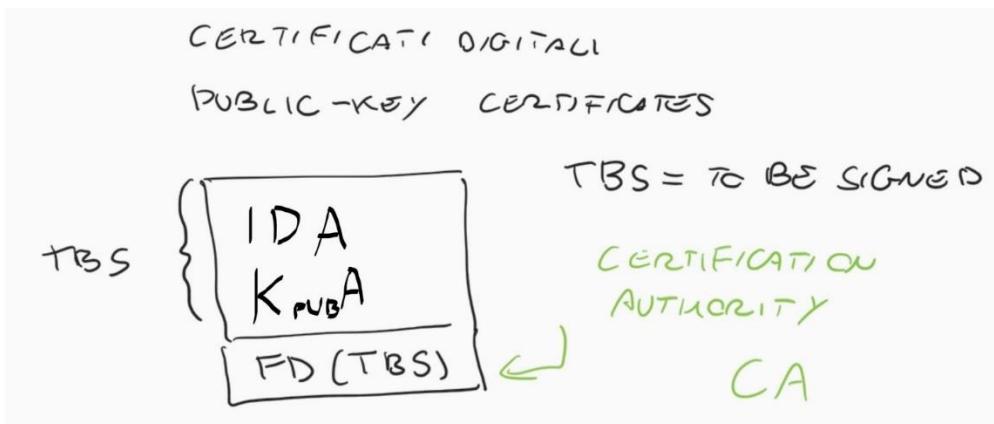
C potrebbe mandare direttamente la chiave pubblica di B ad A, ma così non riesce a creare un canale condiviso con A, in quanto non possiede la chiave privata di B e quindi non può decifrare.

L'attaccante può solamente mandare il certificato di B, ma non gli serve a niente poiché A accetterà solo se la chiave corrisponde a quella nel certificato, ma se è quella di B e C non conosce la chiave privata di B allora non può procedere nell'attacco.

C potrebbe conoscere la chiave privata di B? Non può averla per definizione: Se C possedesse la chiave privata di B allora in questo contesto C sarebbe B. Non può ricavarla nemmeno dalle chiavi pubbliche poiché è molto difficile e ci metterebbe tanto tempo.

Chi rilascia i certificati e ne verifica la validità? le CERTIFICATION AUTHORITY (CA).

CERTIFICATION AUTHORITY (CA)



Siamo soddisfatti di questo meccanismo a firma digitale? Il problema è chi firma. Chi certifica il certificato? Quindi ci vuole un'autorità di certificazione.

Certification Authority (CA). La CA è un ente che firma e rilascia il certificato, e che quindi deve avere una coppia di chiavi privata e pubblica con cui firma i certificati.

CA $(K_{\text{PRIV}}^{\text{CA}}, K_{\text{PUB}}^{\text{CA}})$

Si può vedere anche su browser. Molto spesso le CA sono private o statali.

Cos'è il certificato dell'autorità?

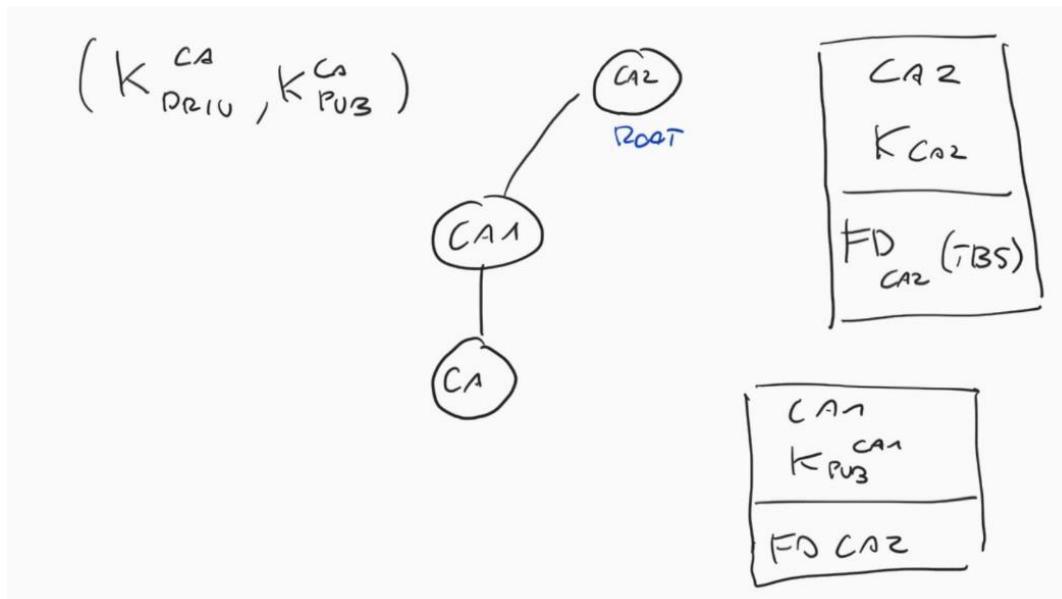


Si verifica firma con autorità di certificazione. Come si fa ad essere certi?

Il problema si ripropone, il problema che avevamo prima era che la chiave pubblica fosse effettivamente di A.

Adesso c'è lo stesso problema per CA.

Chi ci garantisce che questa pubblica sia dell'autorità di certificazione e non sia dell'attaccante che ci ha rifilato questa chiave pubblica dicendo che è lui la CA?
 In generale una CA ha un suo certificato che dice che una è un CA e l'altra è una chiave pubblica della CA, ed è firmato a sua volta da una CA che è gerarchicamente più in alto di essa.
 QUINDI ESISTE UNA GERARCHIA AD ALBERO DI CA che ha come radice una CA-ROOT che certifica la validità di tutte le CA al di sotto nella gerarchia.



è un ciclo infinito.

Un'autorità di certificazione prende il nome di CA root quando rilascia il certificato a se stessa e non è più un problema informatico.

La verifica se CA2 è Root è fatta da CA2 (in quanto è una root) e non è più un problema informatico **ma è dal punto di vista dei contenuti se vogliamo, come posso fidarmi di una CA che rilascia il proprio certificato?** Io posso diventare autorità di certificazione auto-firmarmi il certificato. Il problema è chi riconosce la validità del certificato e non è più una questione informatica .

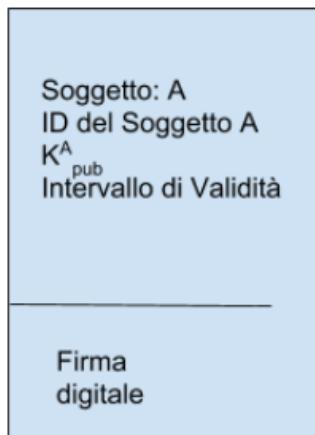
Le CA sono aziende note e conosciute per la loro affidabilità: per esempio in Italia esiste un accreditamento delle CA che devono soddisfare una serie di requisiti. Per esempio, se io usassi CA per il corso saremmo costretti a fidarci che la prof è il capo e la fiducia discende da quello. **Nel mondo vero è tutto a fini commerciali basati sul valore dell'azienda e su quello che è successo.**

CERTIFICATI E CA CONTINUA..

Riprendiamo il discorso sui certificati delle chiavi pubbliche: abbiamo stabilito che siccome lo scopo del certificato digitale è legare una chiave pubblica a un soggetto, due elementi del certificato sono la chiave pubblica e il soggetto, il legame è reso difficilmente dissolubile dalla firma digitale di un'autorità di certificazione(CA). Un certificato ha senso se è un oggetto pubblico e si deve poter mostrare al pubblico: questo implica che nel certificato c'è la chiave pubblica e non può esserci un dato segreto.

Naturalmente però diciamo che se K^A_{pub} è chiave pubblica di A, ciò implica che A è il soggetto. Chi ha la chiave privata di A può spacciarsi per A, ma non c'è modo per altri a parte A di averla quindi chi ha la chiave privata di A deve essere A. L'identificativo di A potrebbe essere ad esempio nome cognome, mail o qualsiasi cosa che lo certifichi. Dietro a questo naturalmente c'è una parte operativa dietro che è di una CA. Questa CA è una coppia di chiavi di firma. Dal punto di vista operativo c'è un'autorità di registrazione che è il front-end, quello che parla col pubblico, **uno che voglia certificare la propria chiave pubblica, regista la propria certificazione alla CA** (Certification Authority) **che firmerà il certificato, di mezzo c'è un'identificazione del soggetto, che a un certo punto deve essere identificato in qualche modo e poi verrà certificata nel certificato.**

Struttura Certificato di A:



L'identificazione dipende anche dal livello di certificato che si vuole. All'inizio quando si iniziavano a diffondere i certificati "verisign" come forma di pubblicità permetteva a chiunque di rilasciare un certificato della propria chiave pubblica con l'unica richiesta che la persona avesse accesso alla posta elettronica che si faceva certificare. E 'un livello di certificazione molto basso.

Il livello di identificazione cambia a seconda dell'uso del certificato e dalla garanzia che è disposta a offrire l'autorità di certificazione.

Un'altra cosa importante in un certificato è l'intervallo di validità: qualunque segreto deve avere una vita limitata perché più cose si firmano più si dà in mano all'attaccante del materiale per attaccare e compromettere la chiave. Ci sono anche ragioni di tipo pratico come meccanismi per verificare l'identità o persone che hanno un ruolo per un certo tempo. Ha senso che il certificato abbia una sua data di scadenza.

Quanto è ampio questo intervallo di validità? E' un parametro che **deve essere regolato in base alla necessità** (troppo lungo apre ad attacchi, troppo corto si perde tempo) **quindi bisogna trovare il giusto trade off.** Un certificato ha quindi una data scadenza, ma non basta perché se la chiave privata viene compromessa si deve revocare il certificato. **Quindi deve essere possibile revocare i certificati (ad esempio in caso di compromissione della chiave privata), più è lungo l'intervallo di validità, più ci saranno in giro certificati non validi.**

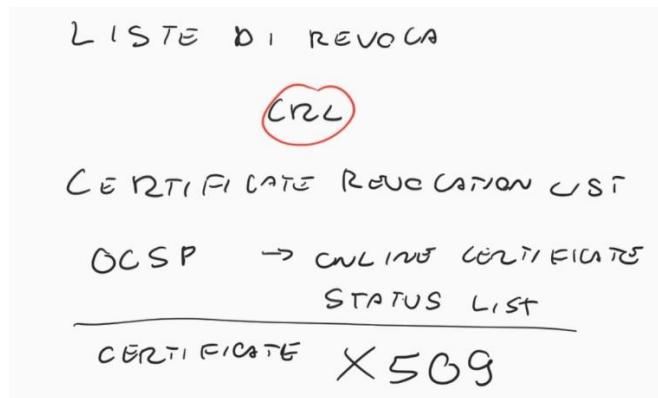
CERTIFICATI DI REVOCA (CRL – CERTIFICATE REVOCATION LIST)

Si risolve mettendo delle LISTE DI REVOCA DEI CERTIFICATI CRL: Certificate Revocation List, le CRL sono elenchi di certificati che non sono più validi anche se sono ancora all'interno del loro intervallo di validità quindi una CA deve mantenere liste di revoca che contengono dei certificati che sono ancora validi, arrivata alla data di scadenza non è più importante mantenerle. Se l'intervallo è molto ampio, sono più corpose ovviamente.

L'utilità del certificato è quello di verificare l'associazione tra utente soggetto e chiave pubblica, questa verifica da sola non può funzionare perché ogni volta che uno usa un certificato bisognerebbe andare a controllare che non sia stato revocato. **Se l'autorità di certificazione è grossa i documenti possono essere molto grandi (quindi le liste di revoca possono essere molto grandi) e quindi creano un ritardo per esaminare la validità.** Creano un certo peso al procedimento poiché bisogna scaricare file grossi. **In realtà esiste un protocollo che si chiama OCSP -> Online Certificate Status Protocol che è un protocollo per la verifica della validità di un certificato** (risponde sì o no se un certificato è valido).

Client scrive a server OCSP e chiede se certificato è valido e server risponde sì o no. Questo sposta le liste di revoca in un server senza che debbano essere scaricate da un client. le risposte OCSP devono essere a loro volta autenticate e firmate dal server OCSP però le risposte di errore non sono firmate ed è una debolezza del protocollo poiché un attaccante potrebbe inviare errori a caso.

IL FORMATO DEI CERTIFICATI X509



Formato dei certificati

Preferences → certificati

Il formato dei certificati che si usa è x509 solitamente. Ci fa vedere Verisign.

In generale c'è il nome del soggetto , organizzazione, numero di serie, nome di chi lo rilascia organizzazione e dipartimento, periodo di validità. Fingerprint sono degli hash.

I dettagli del certificato:

Versione :di solito la x509 versione 3

Numero di serie

Firma: questo campo dice quale firma viene usata, SHA-256 con firma RSA rispettando lo standard PKCS#1

Campi con vari dati vari sull'organizzazione Es: Issues= Chi lo rilascia.

Validità: si distingue in Non prima e Non dopo(di una certa data)

Informazioni sulla chiave pubblica: la chiave pubblica è una chiave per cifrature RSA con PKCS1 con 2048 bit con il solito esponente che non è 3 a due bit (65537)

Estensioni: critico perché specificano se è un'autorità di certificazione. In questo esempio non è critico e c'è l'url del server OCSP

Certificate basic constraints: specifica se è un'autorità di certificazione oppure no. Maximum number of intermediate Cas=0 significa che deve essere rilasciata dalla root ,quindi non può esserci una catena di autorità intermedi.

CRL Distribution Points: punto di distribuzione delle CRL con URI

Uso della chiave certificato è una firma certificata quindi una firma CRL.

Nome alternativo

subject key ID è un hash della chiave pubblica e quello dopo è un hash della chiave di certificazione che ha firmato questo certificato.

La firma del certificato che ha ancora SHA-256 con RSA secondo standard PKCS#1

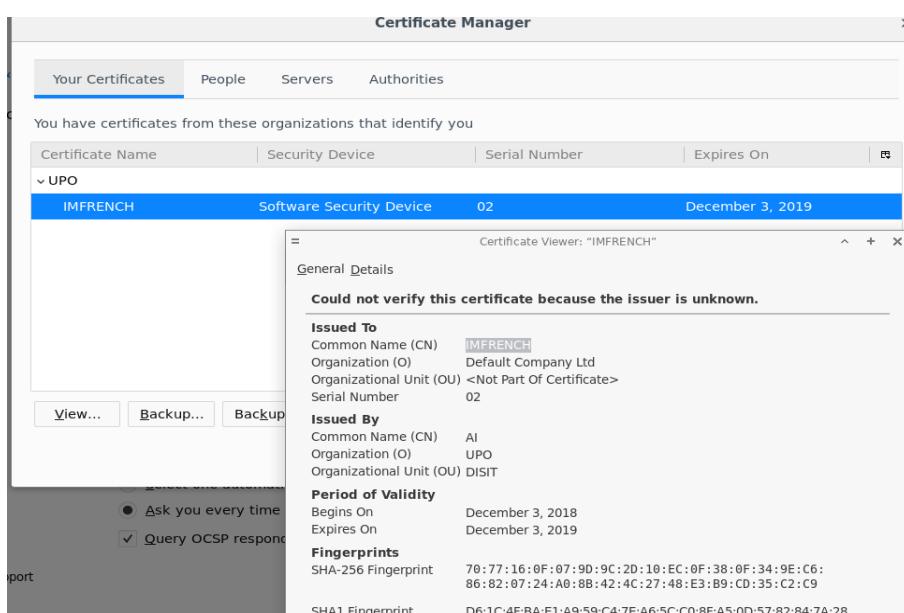
Chi l'ha rilasciato ha gerarchia rilasciato dalla public primary certification authority. La gerarchia è specificata nei basic constraint.

Non vuole che noi conosciamo tutti questi campi. Le cose importanti dentro a un certificato sono la logica **quindi dobbiamo sapere che dentro un certificato digitale ci sono soggetto, ID del soggetto, chiave pubblica (mai la privata!!!) e una firma digitale**. Si potrebbe fare a meno dell'intervallo di validità ma non si può fare a meno di questi tre elementi citati prima.

Quando scarichiamo un browser c'è un elenco predefinito di autorità di certificazione che sono quelle accettate e fidate.

La CA root DigiNotar è stata vittima di un attacco informatico nel 2011 che è stato eclatante, era l'autorità di certificazione del governo olandese e aveva un livello di sicurezza abbastanza infimo (c'è nel madi qualche approfondimento). Quando è stato compromesso sono stati zitti, in realtà è successo qualcosa di molto grave perché sono stati rilasciati dei certificati per Google al governo iraniano e gli servivano questi certificati per fare man in the middle per autenticare dei finiti server Google per beccare i dissidenti.

Non è affidabile quindi e non si può certificare.



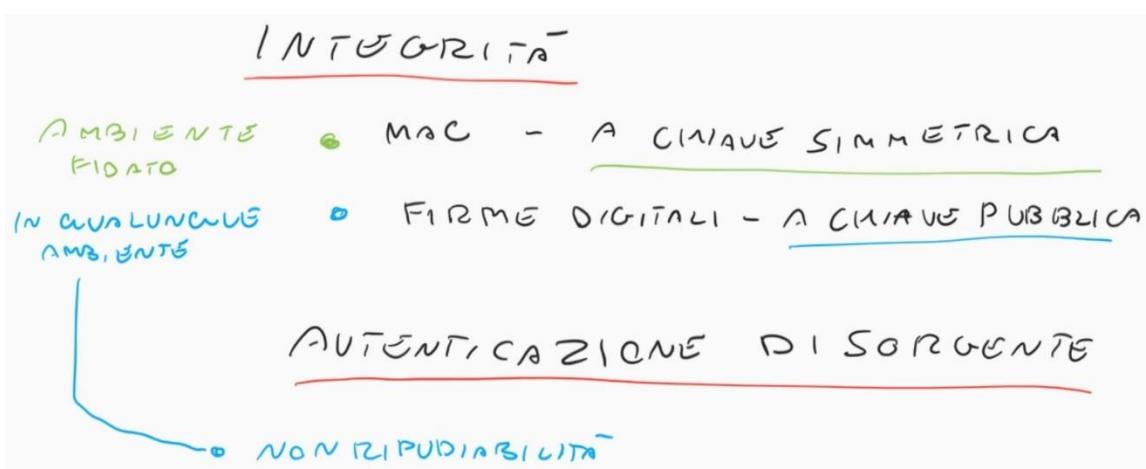
DISCUSSIONE SULLA DIFFERENZA TRA MECCANISMI PER L'INTEGRITÀ

Precisazione:

integrità significa non tanto impedire che un attaccante modifichi un documento, quanto rendere evidenti le eventuali modifiche: anche se l'attaccante potrà modificare dati e messaggi, se sono protetti bene gli utenti legittimi potranno accorgersi se ci sono state manomissioni. Quindi un attaccante può comunque modificarli per cui diciamo che **un meccanismo per l'integrità deve essere affiancato a un meccanismo come quello di replica dei dati inviati** quindi una forma di ridondanza(es. backup).

Già è molto importante accorgersi che i dati sono stati alterati.

Abbiamo 2 strumenti per verificare l'integrità:



- In un ambiente fidato: MAC - A chiave simmetrica
- In qualunque ambiente: Firme digitali - A Chiave pubblica

In tutti e 2 i casi viene esposta un'alterazione eventuale del documento, sia il MAC che le firme digitali sono legate ai documenti: in tutti e 2 i casi perché si usa una chiave sostanzialmente e perché l'oggetto (ovvero uno degli input dell'algoritmo) è sempre il documento, in tutti e 2 i casi c'è sempre di mezzo una funzione hash(in realtà non c'è una funzione hash nel caso di un MAC basato sul CBC ma anche in quel caso se prendete un documento di lunghezza qualunque è un'impronta prefissata che è nell'ultimo blocco è chiaro che non c'è una forma di hash all'interno ma è una forma di hashing anche quella).

C'è sempre quindi questa contraddizione nel meccanismo di integrità: **siccome si vuole una porzione di dati compatta per garantire l'integrità, sono inevitabili le collisioni**. Si deve rendere difficile trovare all'attaccante una collisione maliziosamente usando le funzioni hash, non si vuole nemmeno che la collisione avvenga per caso: **se la funzione hash utilizzata ha una distribuzione abbastanza uniforme che è comunque è fondamentale per renderle crittograficamente sicure, la probabilità che casualmente avvenga la collisione è molto bassa**. La probabilità che una collisione avvenga a caso e la capacità dell'attaccante di trovare una collisione sono due cose diverse, perché la probabilità che una collisione avvenga a caso può essere sfruttata anche dall'attaccante tramite un attacco a forza bruta, ma la possibilità dell'attaccante di fare di meglio è basata su debolezze della funzione Hash utilizzata o nel caso del MAC CBC la debolezza potrebbe essere nel meccanismo.

Questi meccanismi non servono solo per l'integrità ma anche per l'autenticazione di sorgente che è legato all'integrità perché se un documento è alterato si potrebbe dire che la sorgente non è quella originale ma un'altra.

L'Autenticazione di sorgente si ottiene in tutti e 2 i casi, ma in modo diverso: **quando si parla di un MAC, l'aspetto fondamentale è la simmetria della chiave. Per generare il MAC e verificarlo si usa la stessa chiave, la procedura di verifica è una rigenerazione del MAC.**

Chi verifica è anche in grado di produrre. **Nel caso delle firme digitali, la chiave è pubblica e la verifica si fa con una chiave diversa con quella con cui si genera l'oggetto quindi chi verifica non deve necessariamente essere in grado di generare lui stesso una firma.**

Questa è la differenza fondamentale perché nella pratica il MAC ha senso in un ambiente di cui ci si fida dove c'è un interlocutore fidato, la firma digitale si può utilizzare in qualunque ambiente.

Nel caso del MAC, è utilizzato per protocolli di rete. **Cosa significa ambiente fidato?**

Se A e B parlano fra loro e vogliono evitare che una terza parte modifichi, intervenga, legga i dati, si può usare un MAC. Se A vede arrivare un messaggio in cui il MAC È calcolato con la chiave simmetrica che ha condiviso solo con B, allora lui è B perché la chiave simmetrica è condivisa.

Né A né B, se B ha fatto una promessa (ti do 100k), possono dire al giudice che B ha scritto questo perché ha questo MAC e quindi è stato B, perché il giudice nel momento in cui è in grado di verificare il MAC che corrisponde al testo già il giudice stesso è in grado di generare un documento diverso, ma A potrebbe averlo generato, quindi **non è possibile dire se il documento sia creato da A o da B (entrambi hanno lo stesso MAC), ma sicuramente non è stato creato da una terza parte (solo loro possiedono quel MAC)**, quindi in questo senso si è in un ambiente fidato.

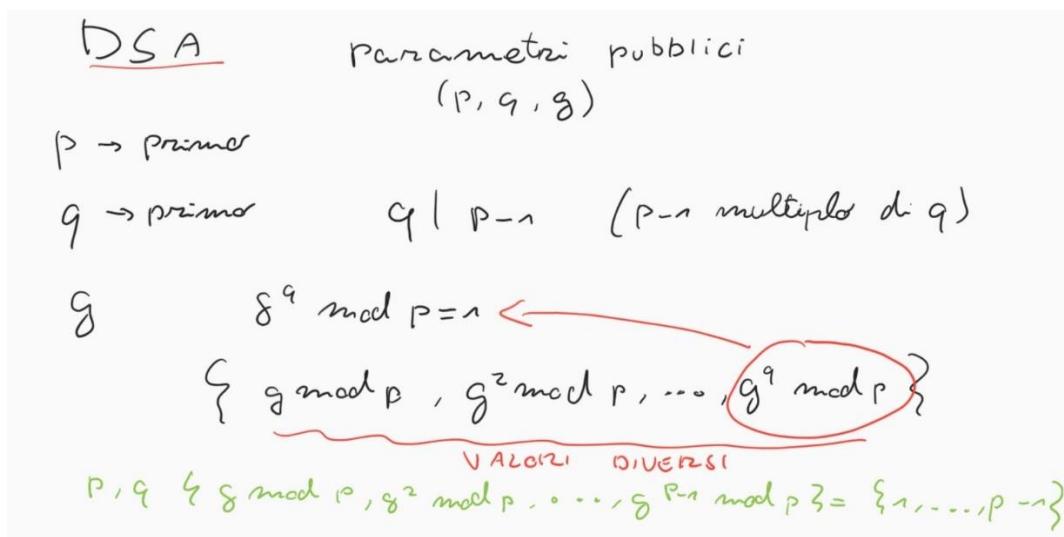
La situazione con la firma digitale è molto diversa perché siccome chi verifica non può necessariamente generare se la chiave privata è conservata correttamente da B. A che ha in mano qualcosa firmato da B può andare da una terza parte e dire che B mi ha firmato questo. A meno di sfruttare una quantità ingente di risorse, non è in grado di generare quel tipo di oggetto. Questa è la differenza sostanziale, in un ambiente in cui vuole tutelarsi da un interlocutore di usa firma digitale. Se il problema è il mondo esterno va bene un meccanismo a chiave simmetrica. L'autenticazione di sorgente quindi Si ottiene in tutti e due i casi e in un caso è una cosa interna (ambiente fidato) mentre nell'altro caso diventa una cosa opponibile a terzi.

Questo implica che **la firma digitale ha la proprietà della non ripudiabilità perché se la chiave è conservata correttamente, B non può negare di aver prodotto quella firma perché solo B può produrlo**, mentre gli altri non possono. Si inseriscono 2 altri aspetti, uno è il certificato **perché tutto questo discorso di opponibilità (andare dal giudice etc.) funziona solo se c'è dietro un certificato che dimostra che la chiave pubblica è di B e quindi B non può negare**, d'altra parte la chiave privata potrebbe essere stata compromessa, **sta a B conservare la propria chiave privata correttamente e segnalare se la propria chiave privata è stata compromessa.** Perché non ha senso, dato che siamo in un ambiente fidato.

DSA (DIGITAL SIGNATURE ALGORITHM)

La firma digitale finora l'abbiamo fatta con RSA+HASH, ma non è l'unico modo di ottenere una firma digitale: l'altro modo è **DSA** (Digital Signature Algorithm) ed è lo standard americano. **DSA al contrario di RSA è basato sul problema dei logaritmi discreti come Diffie-Hellman, siccome è basato su Diffie-Hellman tutto sommato quello che si può immaginare è che non c'è dietro un cifrario, non è come RSA dove si cifra l'impronta per renderla inalterabile. In questo caso il meccanismo è tutto diverso.** Come funziona?

Non ci chiederà il funzionamento completo di DSA ma è importante capire alcuni aspetti



Giallo è esempio che aveva fatto su Diffie Hellman

Per DSA servono dei parametri pubblici come DH, in questo caso sono p, q e g :

p numero primo qualsiasi

q numero primo che divide $p-1$, ovvero $p-1$ è multiplo di q .

g è il generatore $g^q \text{ mod } p = 1$ { $g \text{ mod } p, g^2 \text{ mod } p, \dots, g^q \text{ mod } p$ }, otteniamo tutti valori diversi che sono tutti q e siccome $g^q \text{ mod } p = 1$ tutti i valori diversi sono esattamente q e poi si ricomincia.

In DH il requisito era scegliere p, g tali che $\{g \text{ mod } p, g^2 \text{ mod } p, \dots, g^{p-1} \text{ mod } p\} = \{1, \dots, p-1\}$ questo è più un caso come il controesempio visto per DH.

Con DH avevamo $p = 7$ e $g = 3$ ottenevamo 6 valori mentre con $g = 2$ ottenevamo solo 3 valori. In DSA immaginiamo la situazione $p = 7$ $q = 3$ e $g = 2$, si può notare che q (3) divide $p-1$ (6).

Se G non genera tutto questo ma una parte allora è sicuro che generare un numero che divide $p-1$ (questo è obbligatorio). Per DSA si richiede che q sia primo e il quadro è esattamente quello di DH. Per DSA si vogliono p primo, q primo che divida $p-1$ (3 è primo e divide $p-1$ che è 6 → 3 divide 6), non può essere un multiplo e avere un g che genera un numero di valori che è un multiplo di $p-1$ perché in ogni caso $g^{p-1} \text{ mod } p$ fa sempre 1.

Quindi adesso $g = 2$ cosa genera? genera solo insieme {2, 4, 1} che poi si ripetono quindi sono 3.

Per DSA avremo bisogno di una situazione di questo genere che abbiamo scartato per DH, mentre per DSA ci va bene. Quindi Adesso noi vogliamo generatori di questo genere che producano quindi insiemi di risultati più piccoli rispetto a quelli che volevamo con DH. Per questo spesso si usa come generatore 2 in modo che DSA e Diffie-Hellman siano interscambiabili.

FUNZIONE HASH H

A : X PRIVATA

$$Y = g^x \bmod p$$

SELEZIONARE $0 < k < q$ PSEUDO-CASUALE

$$(r, s) \quad r = (g^k \bmod p) \bmod q$$

$$s = k^{-1} (H(m) + xr) \bmod q$$

Poi serve una funzione hash; un utente che fa la firma A ha bisogno di una chiave privata e una pubblica che si ottiene come per DH con $Y = g^x \bmod p$

Per fare una firma è necessario selezionare un numero k pseudo casuale che sia compreso tra 0 e q.

La firma è una coppia r,s ottenuta con $r = ((g^k \bmod p) \bmod q)$ ed $s = k^{-1} (H(m) + xr) \bmod q$.

La chiave privata è X e quella pubblica è Y e ci servono i parametri p,q e g. C'è l'hash di M, quindi è indifferente firmare un messaggio M con qualunque messaggio che ha la stessa impronta.

Se usa un altro M con la stessa impronta, r,s vengono identici e quindi c'è il solito problema delle collisioni ed è meglio utilizzare funzioni crittograficamente sicure come in questo schema.

Computazionalmente ci sono elevamenti a potenza in modulo, somme, prodotti in modulo e diciamo che dal punto di vista computazionale la cosa più pesante è l'elevamento a potenza.

La verifica della firma ce la fa vedere su questa slide:

verifica della firma

$$(g^{H(m)s^{-1}} \bmod q \cdot y^{rs^{-1}} \bmod p) \bmod q = ? r$$

perché

$$(g^{H(m)s^{-1}} \bmod q \cdot g^{xrs^{-1}} \bmod p) \bmod q = \dots \text{poiché } y = g^x \bmod p$$

$$(g^{H(m)s^{-1}} \bmod q + xrs^{-1} \bmod p) \bmod q =$$

$$(g^{(H(m)s^{-1} + xrs^{-1})} \bmod p) \bmod q =$$

$$(g^{s^{-1}(H(m) + xr)} \bmod p) \bmod q =$$

$$(g^k \bmod p) \bmod q = \dots \text{poiché } s^{-1} = k^{-1} (H(m) + xr) \dots$$

$$(g^k \bmod p) \bmod q = \dots \text{poiché } 0 < k < q \dots$$

r

ricordiamo che s e r sono gli elementi che compongono la firma. La verifica delle firma come viene fatta? non è necessario guardare tutti i dettagli. chi deve verificare la firma ha in mano M , ha in mano la firma (r,s), conosce la chiave pubblica e i parametri pubblici e sa qual è la funzione hash utilizzata. quindi da.. può calcolare H(M) g , p e q sono pubblici. Fa operazione sulla slide e l'idea di questa operazione è che dentro y c'è g^x modulo p. Quindi questo y è g^x c'è quindi c'è g elevato alle cose nella slide ed è tutto un esponente e tutta la prima parte dell'esponente assomiglia moltissimo al modo in cui è stato calcolato S, che è invece calcolato con k^{-1} . Quindi questa parte s^{-1} ($H(m) + xr$) è uguale a k. La verifica parte da tutto quello che l'utente sa. Quindi parametri pubblici, chiave pubblica e messaggio ed (r,s). Sono tutti conti algebrici abbastanza semplici. Con RSA la verifica si decifrava e si confrontavano poi le impronte. Qui usando (r,s) si deve riottenere r.

E' un procedimento in avanti poiché non si torna indietro. Questo meccanismo è stato scelto come standard americano e non RSA perché avevamo parlato dei problemi legati all'esportazione di cifrari per non mettere mano a paesi non amici dei sistemi crittografici forti che potevano servire per inviare messaggi segreti contro gli USA protetti con una crittografia offerta dagli USA stessi a loro su un piatto d'argento.

Quindi hanno disaccoppiato la cifratura con la firma (mentre DSA non li disaccoppia) e scelto RSA come algoritmo di firma e in questo modo l'algoritmo di firma l'hanno potuto diffondere in tutto il mondo per firmare. Il problema non è se il nemico firma ma che il nemico può usarlo per cifrare per scambiarsi messaggi protetti molto bene. La scelta di RSA come schema di firma standard è stata fatta sotto la base di questa osservazione. In realtà poi si pensa che sia fatta sulla base di questa osservazione poiché ci sono osservazioni di carattere militare e ci sono state molte polemiche per introdurre DSA.

Il numero K che viene scelto, per cui sembra una cosa secondaria ma è molto importante non solo che sia pseudocasuale, ma anche diverso per ogni messaggio. perché ?

★ se si riutilizza k

- se si utilizza lo stesso valore k per firmare M_1 ed M_2 con la stessa coppia di chiavi (x,Y) –notate che p,q,g sono gli stessi:
- firme: (r_1,s_1) e (r_2,s_2)
- $r_1=r_2=(g^k \bmod p) \bmod q$ (chiamiamolo r)
- $s_1=k^{-1}(H(M_1)+xr) \bmod q$
- $s_2=k^{-1}(H(M_2)+xr) \bmod q$
- poiché $H(M_1)$, $H(M_2)$, s_1 , s_2 ed $r=r_1=r_2$ sono noti pubblicamente, dalle due equazioni qui sopra l'attaccante può ricavare k e x

L'attacco che si può fare se k si riutilizza: supponiamo che un utente utilizzi lo stesso k per firmare M_1 e M_2 , ci sono 2 firme diverse ma le 2 r sono uguali, a questo punto s contiene R,S è basato su r ed i k son gli stessi e gli r son gli stessi, $H(M_1)$ è noto perché i messaggi sono noti, r è noto, x no, k non è noto e q è noto perché è un parametro pubblico, noi abbiamo due equazioni e 2 incognite (k e x), su questo nel 2011 c'è stato un attacco alla Sony ps3 in cui attaccanti sono stati in grado di bucare ed estrarre la chiave perché il generatore di k non era robusto e quindi era possibile prevedere quando la stessa chiave k ritornava e in questo modo si riusciva a vedere la chiave x di firma.

Il problema era quindi il modo in cui è stato utilizzato, non DSA in se.

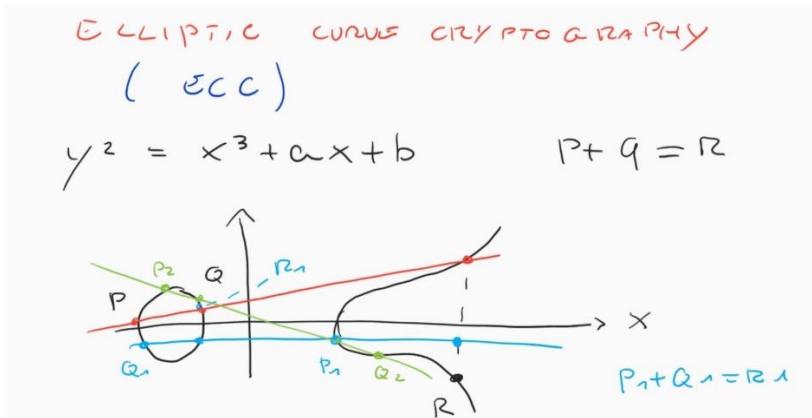
Qual è la lunghezza delle chiavi RSA? sarà come DH perché è basato sullo stesso problema dei logaritmi discreti, che è esponenziale anche se non pienamente e quindi richiede chiavi lunghe almeno 1024 bit.

Avere chiavi di 1,2,4kB è molto pesante, le chiavi dovrebbero essere anche più lunghe e questo è un problema su dispositivi con capacità computazionale limitata, sensori dove bisogna verificare integrità a chiave pubblica, un contesto in cui ci sono tanti sensori con dati rilevanti, se l'attaccante riesce a modificare i dati che vengono dai sensori delle varie apparecchiature il sistema centrale se riceve dei dati alterati magari da comandi sbagliati e può fare gravi danni.

L'ambiente è fidato quindi uno potrebbe usare chiave simmetrica ma non è detto perché la struttura potrebbe essere molto grande, con sensori in zone protette, l'attaccante così dovrebbe mettere le mani sul sensore e se ce le mette sopra potrebbe risalire alla chiave simmetrica e quindi la chiave dovrebbe essere differente per ogni sensore e quindi a un certo punto lo si vuole fare a chiave pubblica ma la potenza computazionale limitata dei dispositivi crea un problema.

Se uno sul cellulare vuole andare su un sito HTTPS macina chiavi di lunghezze importanti, più è grossa al chiave più lo spazio è occupato, più energia ci vuole per l'elaborazione, se uno ha un dispositivo mobile con potenza ridotta, questa crittografia a chiave pubblica con chiavi così grosse è un problema, anche il surriscaldamento del dispositivo è un problema rilevante. La crittografia a chiave pubblica non è l'idea vincente e si può migliorare, si aspira quindi a crittografia a chiave pubblica con chiavi più compatte.

ECC (ELLIPTIC CURVE CRYPTOGRAPHY)

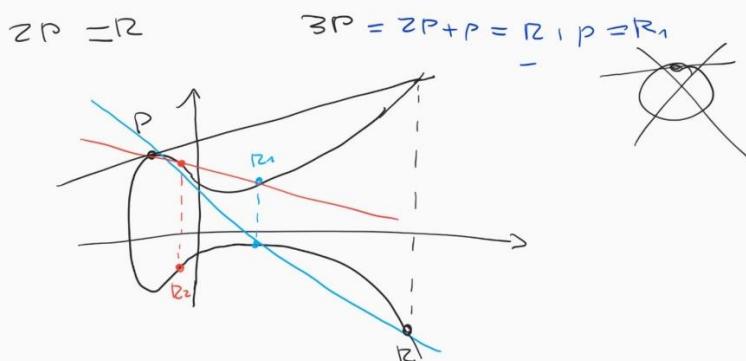


Famiglia di primitive crittografiche che raggiungono lo stesso livello di protezione da attacchi rispetto a sistemi a chiave pubblica con chiavi decisamente più corte grazie alle proprietà matematiche. Usa infatti le equazioni delle curve ellittiche come per esempio $y^2 = x^3 + ax + b$ e potrebbe avere una forma del genere come in figura e potrebbe anche essere sconnessa a seconda del valore di a e b .

C'è una proprietà interessante che useremo nelle curve ellittiche: **Una qualunque retta del piano interseca la curva ellittica in 3 punti e sulla base di questa proprietà si possono definire una somma di punti in questo modo. Se abbiamo p e q , si ottiene $p+q$ andando a prendere il terzo punto di intersezione e si prende il suo simmetrico dall'altra parte della curva e questa è la costruzione.**

$$P+Q=R.$$

Se io prendessi un'altra retta si interseca in 3 punti e prendessi p_1 e q_1 , $p_1+q_1 = r_1$. I punti devono essere sulle curve e se dovessi fare la somma tra P_2 e Q_2 traccio la retta tra i punti che interseca la curva in un terzo punto e la somma sta lì nel punto in basso. **La somma è ben definita comunque si prendano i punti sulla curva.** Si prende terzo punto di intersezione che è unico e si prende intersezione delle X. r è il simmetrico del terzo punto di intersezione. **Non dobbiamo impararla.**



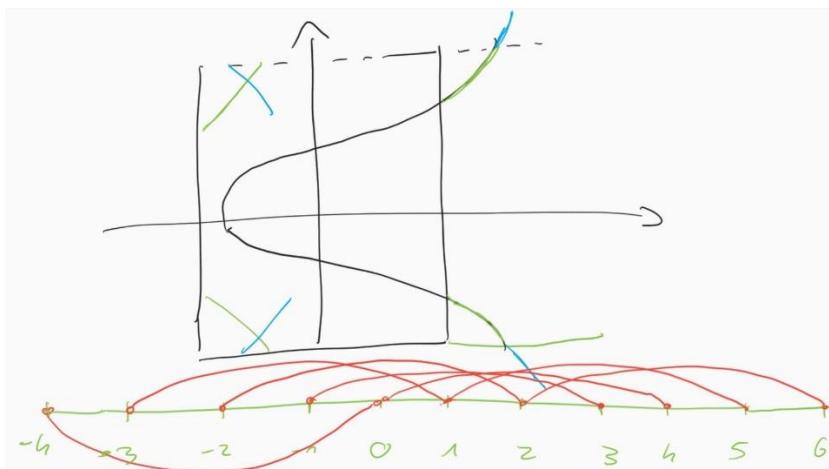
Se io volessi fare 2 volte P quindi $2P$. Abbiamo Detto che la retta interseca curva in tre punti. Una tangente ad una curva è tangente ad una doppia intersezione di P . Quindi la tangente di P ha due intersezioni avrà un'altra intersezione che sarà il punto R . Qualunque retta taglia circonferenza in due punti tranne quelle tangenti che intersecano in un solo punto (anche se doppia con i calcoli matematici). Tangente quindi interseca due volte sullo stesso punto. $2P = R$ quindi interseca in 3 punti perché P viene considerato due volte.

Quindi $3P = 2P + P$ quindi è uguale a $R + P$.

Supponiamo retta [colore azzurro] succede che in mezzo c'è terza intersezione R_1 e il risultato è $R + P = R_1$. Potrò andare avanti con $4P$ ecc.

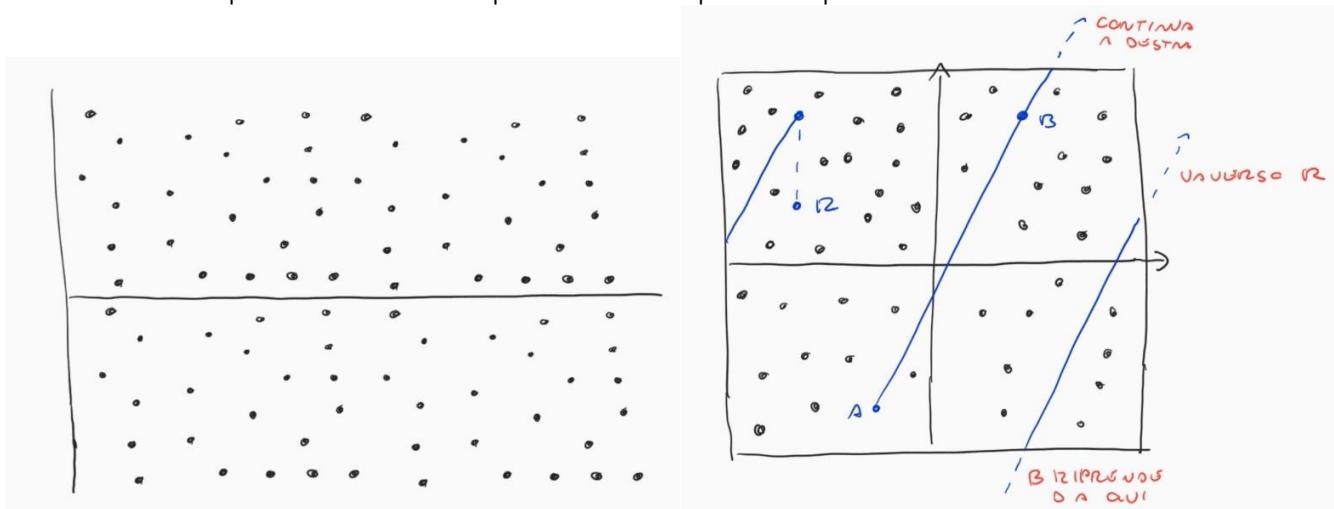
C'è un problema supponendo di voler fare somma di questo punto con questo punto [linea verde] potrebbe essere molto lontano e intersezione potrebbe essere molto lontana e somma potrebbe assumere valori molto grandi.

Usavamo modulo per mantenere valori in un ambito limitato.



Sostanzialmente quello che si fa con le curve ellittiche è considerare una curva ellittica in modulo. Questo significa che se considero dei numeri interi e li voglio prendere in modulo 4 per esempio, i valori saranno da 0 a 3, dove 4 sarà mappato in 0, 5 in 1 e così via. Tutti i valori vengono rimappati in modulo su 0 - 3. Quindi tutti i numeri vengono rimappati. Nel caso delle curve ellittiche si fa qualcosa del genere questi valori dentro vengono rimappati dentro il dominio. Quindi pezzi ricomparirebbero su e giù e la curva viene rimappata tutta dentro un quadrato.

Quando si parla di DH e RSA non solo lavoriamo in modulo, lavoriamo sul mondo discreto (non lavoriamo su numeri con la virgola) e rimappando il tutto e si otterrà un quadrato pieno di curve. Si usano solo i punti interi quindi solo le intersezioni su quadretti. La curva diventerà quindi un ammasso di punti. Diventa un quadrato tutto pieno di punti discreti



Le somme adesso sono la stessa cosa. Ci fornisce un'operazione che è quella della normale somma con elementi neutri, somme di numeri con se stessi. Tutte le proprietà della somma che utilizzeremo e non abbiamo ancora un sistema crittografico basato su questo e vedremo le prossime lezioni

La matematica delle curve ellittiche è difficile ed in realtà questo è il suo vantaggio. Se ho la stessa sicurezza dei modelli precedenti.

DIFFIE-HELLMAN A CURVE ELLITTICHE

Ha cominciato a parlare di crittografia a curve ellittiche e per il momento. Quello che ci ha fatto vedere (la somma dei punti) è semplicemente perché non vuole mai che quello che si fa sembri magia ma per darci un'idea di cosa si tratta. NON è importante la somma dei punti, tangente ecc. **Ma è importante sapere che sulle curve ellittiche si ottiene una somma di punti che ha tutte le proprietà della somma e del prodotto nell'aritmetica modulare.** L'elemento neutro è lo zero per la somma, per il prodotto l'1, ma la proprietà associativa vale per tutte e 2 nello stesso modo. Abbiamo parlato di somme dello stesso punto più volte e a cosa corrisponde nella moltiplicazione? elevamento a potenza e questo è quello che ci serve sapere.

Avendo chiarito il rapporto tra somme e prodotti, ci introduce la crittografia a curve ellittiche che sono sistemi crittografici basati su questa matematica. **Diffie-Hellman si può rendere a curve ellittiche sostituendo i prodotti con le somme.**

DH

p, g
chiavi private: X_A, X_B
chiavi pubbliche:
 $Y_A = g^{X_A} \text{ mod } p$
 $Y_B = g^{X_B} \text{ mod } p$
calcolo del segreto:
 $Y_A^{X_B} \text{ mod } p = Y_B^{X_A} \text{ mod } p$
 $= g^{X_A X_B} \text{ mod } p = g^{X_B X_A} \text{ mod } p$
 $= S$

ECDH

curva, G
chiavi private: d_A, d_B
chiavi pubbliche:
 $Q_A = d_A G$
 $Q_B = d_B G$
calcolo del segreto:
 $d_B Q_A = d_A Q_B$
 $= d_A d_B G = d_B d_A G = (x_1, x_2)$
 $S = x_1$

Abbiamo una curva e un generatore G che corrisponde a un punto sulla curva. chiavi private: X_A e X_B sono due numeri su ECDH: d_A e d_B

$g^x \text{ mod } p$ in D-H corrisponde alla somma di G con se stesso B volte (moltiplicazione) $\rightarrow Q_B = d_B G$
Un numero per un punto fa un punto. Se faccio $G+G$ viene un altro punto, se lo faccio d_A volte viene il punto Q_A .

Le chiavi pubbliche sono dei punti mentre quelle private sono dei numeri.

Il calcolo del segreto è dato dalla somma del punto chiave pubblica dell'altro moltiplicato tante volte quanto il numero di volte che è il segreto. Nel caso dalla parte di A moltiplicherebbe per d_A .

Il segreto corrisponde a $g^{x_A x_B} \text{ mod } p$ di D-H dove qua corrisponde alla somma di G per se stesso $d_A * d_B$ volte.

Generiamo un punto S che è una coppia (x_1, x_2) .

ECDH da un segreto S che è un punto, ovvero una coppia di coordinate quindi ne basta sceglierne una. Il segreto condiviso diventa per esempio X1.

In D-H normale prima era un numero ora invece si deve scegliere.

DH a curve ellittiche si può pensare come DH con una matematica diversa, la ragione per cui si usa la notazione additiva è perché è più appropriata dal punto di vista matematico. Questo ci dice che si usa D-H per lo scambio di chiavi nello stesso modo.

Si usa D-H a curve ellittiche nello stesso modo di D-H nell'aritmetica modulare, la matematica che ci sta dietro è più complicata però alla fine uno che ha quella somma fa le stesse cose.
Oltre a D-H a curve ellittiche vediamo la firma digitale DSA a curva ellittica.

DSA A CURVE ELLITTICHE

DSA	$z = H(M)$ eventualm. troncato	ECDSA
<p>p, g, q</p> <p>chiavi: privata x, pubblica $y = g^x \pmod{p}$</p> <p>$0 < k < q$ diverso per ogni M</p> <p>firma:</p> $r = (g^k \pmod{p}) \pmod{q}$ $s = k^{-1} (z + x r) \pmod{q}$ <p>verifica:</p> $g^{zs^{-1} \pmod{q}} y^{rs^{-1} \pmod{q}} \pmod{p} = ? r$		<p>curva, G, n</p> <p>chiavi: privata d, pubblica $Q = dG$</p> <p>$0 < k < n$ diverso per ogni M</p> <p>firma: $(x_1, x_2) = kG$</p> $r = x_1 \pmod{n}$ $s = k^{-1} (z + d r) \pmod{n}$ <p>verifica:</p> $(w_1, w_2) = z s^{-1} G + r s^{-1} Q$ $w_1 \pmod{n} = ? r$

La firma digitale a curve ellittiche si può vedere come curva, G ed n . In DSA normale con G genero q valori differenti, con ECDSA si generano n valori differenti, elevando a potenza con se stesso.

La chiave privata è un numero "d", chiave pubblica come con DH-EC è un d^*G e quindi un punto.

scegliamo un k compreso tra 0 e n diverso per ogni messaggio M

per calcolare r ed s , ci ricordiamo che k moltiplicato per G è un punto di coordinate (x_1, x_2) .

r è dato da $x_1 \pmod{n}$ (così è un numero) e poi si calcola s

Invece prima avevamo $(g^k \pmod{p}) \pmod{q}$, perciò in ECDSA $x_1 = g^k \pmod{p}$, mentre \pmod{q} è equivalente a \pmod{n} . La verifica prima era un elevamento a potenza mentre qua più o meno è la stessa cosa e infine si confronta che la prima coordinata sia uguale a r .

NON CI CHIEDERÀ I DETTAGLI, però ci fa capire che sono la stessa cosa.

z è l'hash di M, la firma è sempre fatta sulla impronta di M quindi come per RSA anche per ECDSA c'è il problema delle collisioni, le firme sono identiche se hanno impronte uguali. **LA funzione di hash deve essere crittograficamente sicura.**

Quello che dobbiamo sapere dell'uno e dell'altra è che è un meccanismo a chiave pubblica c'è una chiave pubblica e una privata che la firma è fatta da due parti, r e s . Per verificare la firma serve sempre il messaggio

Ci sono parametri pubblici che sono p e q .

Dati questi parametri il messaggio e la firma si può verificare.

Da capire per l'utilizzo è che k (parametro che serve per la firma) deve essere sempre diverso per ogni messaggio quindi generato pseudo-casualmente. Il generatore pseudocasuale deve essere buono perché altrimenti si può prevedere il valore di k e quindi rendere il sistema vulnerabile ad attacchi.

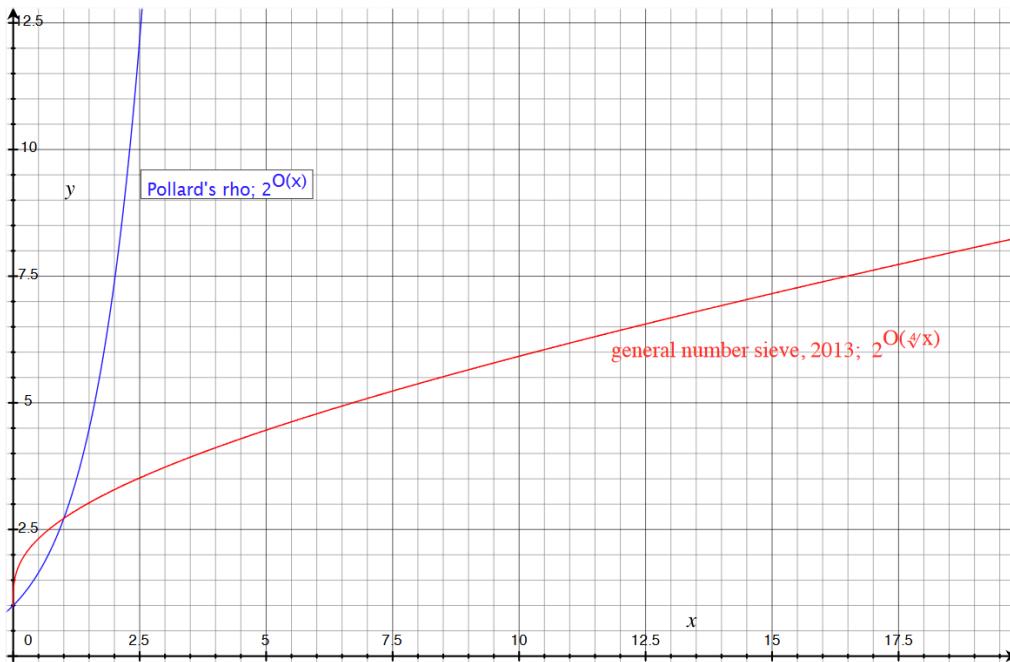
A questo punto abbiamo D-H e DSA basati sulle curve ellittiche.

Qual è il vantaggio di questo lavoro?

Essendo l'analogico nel mondo delle curve ellittiche la crittografia è protetta dal problema dei logaritmi discreti. tra parentesi, non esiste nelle curve ellittiche un analogo dell'RSA(che operava con la fattorizzazione).

Quindi siamo nel mondo di D-H e delle curve ellittiche.

Perché usare questa matematica più complessa (ECC) invece di quella che avevamo visto (Log discreti)? La ragione è data da questo:



Il modo in cui cresce la complessità dell'algoritmo per determinare i logaritmi discreti (e tra parentesi anche la fattorizzazione di un numero) nel mondo dell'aritmetica lineare come vediamo non è esponenziale ma è sub-esponenziale.

La rho di Pollard è il modo più efficiente che si conosce per calcolare i logaritmi discreti nel mondo delle curve ellittiche e come vediamo è esponenziale. Sull'asse delle x abbiamo la lunghezza della chiave mentre sull'asse delle y abbiamo il tempo che deve impiegare l'attaccante per bucare la chiave.

Quindi nel mondo delle curve ellittiche si ottiene una difficoltà per l'attaccante molto prima.

Queste sono le linee guida del NIST (istituto nazionale degli standard tecnologici americano) di qualche anno fa.

Raccomandazioni:

Ricordiamo che **gli schemi crittografici usati sono ibridi**. Crittografia a chiave pubblica per scambiare chiavi e simmetrica per scambiare dati.

Quindi la crittografia a chiave pubblica e simmetrica devono essere ben bilanciate perché se uno usa un sistema crittografico a chiave simmetrica forte e un sistema a chiave pubblica debole l'attaccante attaccherà quello o k pubblica per ottenere la k simmetrica, altrimenti attaccherebbe direttamente quello a k simmetrica.

Il punto è che **l'attaccante attaccherà sempre il più debole tra i due schemi** e usarlo uno più debole rispetto all'altro è uno spreco di risorse per gli utenti legittimi perché non ci fanno niente con un eccesso di sicurezza da una parte perché tanto l'attaccante farà l'attacco all'altra parte. **Quindi ha senso tarare le chiavi in coppia.**

Questa tabella ci dice che se ho una chiave di 128 bit ci dovrà essere una chiave da 3072 di RSA per avere lo stesso livello di sicurezza.

Con 192 bit di AES corrispondono 7680 RSA, mentre per 256 AES avrò 15360 RSA.

DH è analogo a RSA come ragionamento di aritmetica modulare.

Con le curve ellittiche per la difficoltà dell'attacco l'ordine di grandezza è circa 10 volte più piccolo e cresce più lentamente come si vede sulla curva.

Nella crittografia a curve ellittiche, l'effetto è che si possono usare chiavi pubbliche molto più corte nell'ordine di grandezza paragonabili a quelli della crittografia a chiavi simmetriche.

Quindi tutti i problemi di memoria ridotta, batterie limitate con problemi di surriscaldamento, computazione ecc. si risolvono con l'utilizzo di una crittografia a curve ellittiche evitando di avere così chiavi pubbliche troppo lunghe. Le curve ellittiche vanno bene ma ci sono sempre dei lati negativi. Come sempre nella crittografia non è dimostrato niente. Quello di cui stiamo parlando sono algoritmi noti, c'è una corrente di pensiero che dice che il punto è che la crittografia a curve ellittiche è nuova rispetto a quella basata sull'aritmetica modulare e sono problemi studiati meno quindi non è detto che siano più difficili in realtà è solo che sono stati studiati di meno.

Per il momento come ha fatto l'RSA su DES che ha proposto challenge la stessa cosa è stata fatta dalla Certicom per la crittografia a curve ellittiche e al momento le chiavi a 131 sono oltre il limite. (The 109-bit challenges have been solved, while the 131-bit challenges will require significantly more resources to solve.)

<https://www.certicom.com/content/certicom/en/the-certicom-ecc-challenge.html>

Quindi la challenge su 131 bit è ancora aperta e quindi effettivamente siamo ancora in sfida.

Il fatto è che la Certicom ha dei brevetti su molte curve ellittiche, in realtà la Certicom è stata acquisita dalla Blackberry ed è proprietaria di 150 brevetti sulle curve ellittiche, non è chiaro se hanno brevetti sulle curve o sulle implementazioni però il problema è che questi brevetti creano delle difficoltà perché ad esempio hanno citato la Sony perché utilizzava le curve ellittiche in modo non del tutto legale in quanto senza il brevetto della Certicom.

Questo scoraggia un po' l'utilizzo e la diffusione delle curve ellittiche.

L'NSA (national security agency) degli stati uniti si è rassicurata per l'utilizzo di alcuni di questi brevetti.

Quindi c'è questo problema dei brevetti, ma ce n'è anche un altro perché a causa della complessità del contesto c'è un altro problema legato all'implementazione.

Storia delle safe curves:

anche se le curve sono buone (matematicamente è tutto a posto, in teoria tutto funziona), ci possono essere alcuni problemi ad utilizzare alcune curve legati all'implementazione.

- per alcuni punti rari delle curve può produrre risultati sbagliati.
- Your implementation leaks secret data when the input isn't a curve point. (quando l'input non è un punto della curva)
- Your implementation leaks secret data through branch timing. (se uno verifica i timing dei branch, in base a tempi di esecuzione dell'algoritmo)
- Your implementation leaks secret data through cache timing. (Attraverso l'analisi del tempo di cache può verificare alcune cose)

Questo perché le implementazioni non sono adeguate.

Quello che dice l'articolo è che il sistema ideale non è interattivo, invece nel caso del mondo reale quello che si utilizza può accettare input dati dall'attaccante e quindi l'attaccante ci può giocare. Quello che rivela il sistema teorico rivela solo aspetti non critici quindi non dovrebbe rivelare tempi di algoritmi che invece nell'implementazione si possono vedere.

Le implementazioni reali possono commettere errori su alcuni punti, anche se la teoria è perfetta le implementazioni introducono problemi di sicurezza più o meno inevitabili, **anche se la curva è buona, bisogna confrontarsi con il mondo reale e stare attenti alle implementazioni. Una classe di curve che si chiamano safe curves, hanno il vantaggio che qualunque buona implementazione di queste curve non rilasciano le debolezze elencate**, allora questo movimento delle safe curves raccomanda l'utilizzo di safe curves.

Qui c'è un elenco di varie curve e dice se sono safe:

La curva 255519 in particolare è una curva considerata safe ed è molto importante e ha questo numero perché il modulo è = $2^{255} - 19$ ed è questa la ragione XD

Non è brevettata ma libera ed è anche safe quindi IT'S FREE :)

Quindi questo è perché può fare bene saperne e sottolinea un aspetto che le implementazioni dei sistemi crittografici sono una dannazione e non sono una cosa banale da fare.

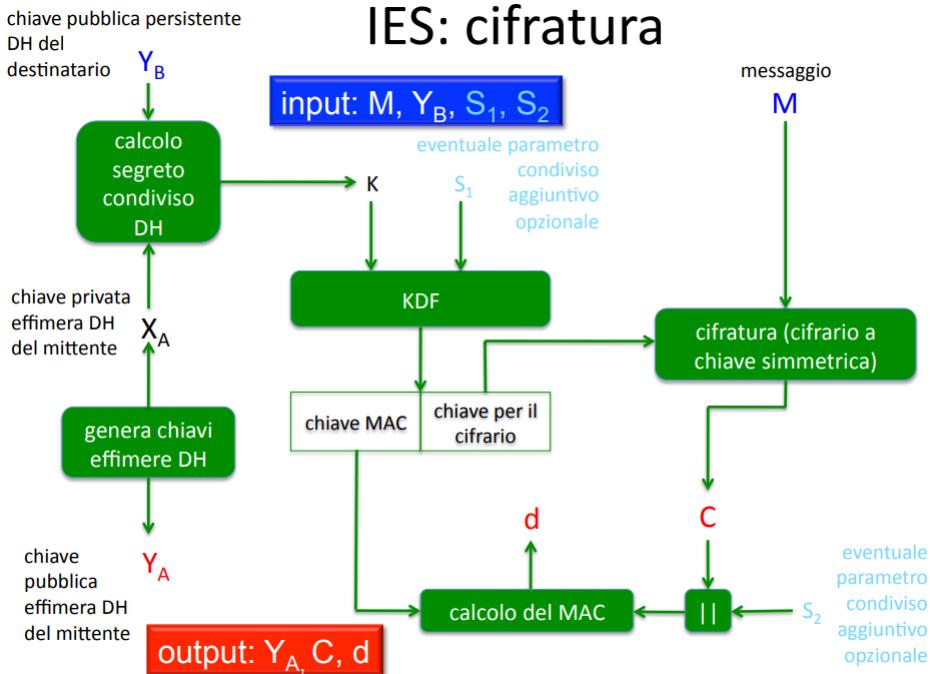
Nascondono un sacco di trappole e difficoltà nascoste che un crittografo non è in grado di immaginare. In generale la raccomandazione è quella di evitare di implementare algoritmi crittografici. Protocolli sì utilizzando le librerie (sempre restando molto attenti) mentre per gli algoritmi no.

Non possiamo usare RSA, ma il fatto di non usare RSA non ci crea problemi perché possiamo usare D-H per scambiare le chiavi, non abbiamo le chiavi simmetriche e non abbiamo un modo per cifrare a chiave pubblica che avevamo tutto sommato con RSA.

È possibile che leggiamo in giro che esiste un cifrario basato su D-H, questa cosa è impossibile.

Quello che viene spesso chiamato cifrario basato su DH È IES che è un protocollo e significa Integrated Encryption standard.

IES (INTEGRATED ENCRYPTION STANDARD)



CIFRATURA

E' uno schema ibrido di cifratura infatti succede che c'è uno scambio di chiavi, chi vuole cifrare qualcosa ha la chiave pubblica del destinatario, quindi sono DH normale ma potrei anche essere in DH a curve ellittiche e si chiamerebbe ECIES.

Il mittente genera chiavi effimere di DH (cioè li genera per usarle solo 1 volta e poi le butta) con la k privata effimera D-H e la k pubblica dell'altra parte (y_b) genera un segreto condiviso, dall'altra parte invece la k pubblica effimera Y_a la deve dare in output dall'altra parte in quanto servirà per rigenerare il segreto condiviso.

Il segreto condiviso viene dato in input a un generatore di sequenze pseudocasuali KDF insieme a un altro parametro condiviso opzionale (ci può essere oppure no) e genera dei dati che vengono spezzati in chiave per l'integrità e chiave per la cifratura.

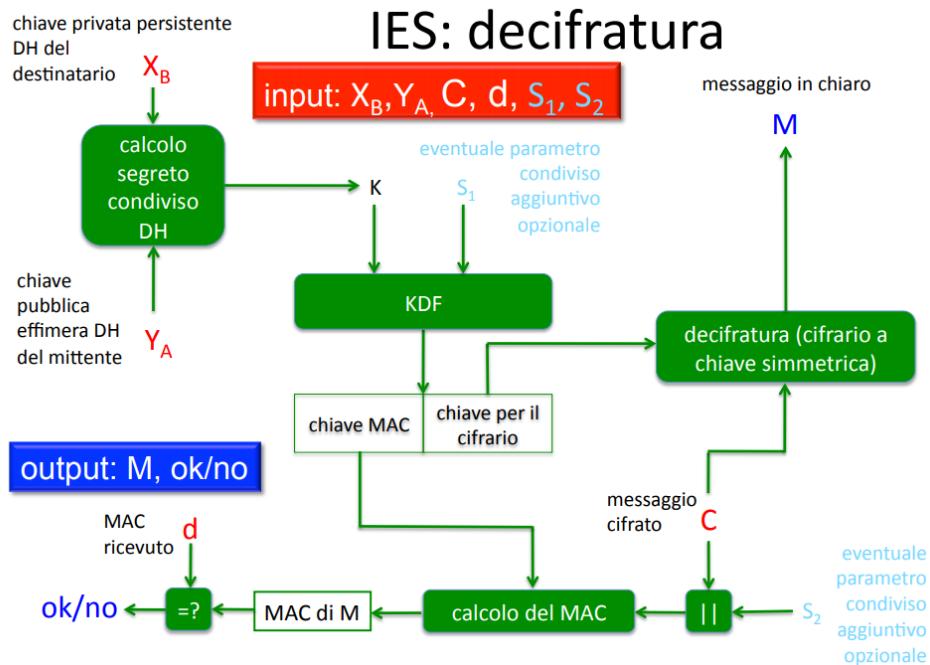
La chiave di integrità viene usata per calcolare il MAC sul testo cifrato.

La chiave per il cifrario prende il messaggio e lo cifra a chiave simmetrica, quindi questo sistema è parametrico (può essere usato qualunque cfrario a k simmetrica qui) si ottiene il testo cifrato che viene eventualmente concatenato con un altro parametro condiviso opzionale e viene calcolato il MAC utilizzando la chiave di integrità e si ottiene il valore d.

L'output di tutto questo meccanismo è: chiave pubblica(Y_a), testo cifrato(C), e i dati per integrità (d)

Questo è semplicemente uno schema ibrido, si usano le chiavi pubbliche per arrivare a un segreto condiviso, e da esso come al solito si generano le chiavi di cifratura e poi si cifra e si calcola l'output.

DECIFRATURA:



Dall'altra parte qual è l'input di chi deve decifrare?

La k privata D-H persistente del destinatario la kPub effimera dell'altro che è arrivata con il messaggio, il test, messaggio, cfrato, d(MAC) ed eventualmente S1 e S2
Usando la k privata e la k pub effimera del destinatario calcola di nuovo il segreto condiviso K.
Riottiene la chiave Mac e la chiave per il cfrario, decifra C da una parte e ottiene il msg in chiaro e dall'altra riconcatena C a s2 se c'era e rigenera il Mac con la chiave Mac e verifica se il Mac corrisponde al d che ha ricevuto.

L'output di questo meccanismo è il messaggio in chiaro e la verifica per l'integrità.
Semplicemente uno schema crittografico ibrido, in realtà esistono dei cfrari basati su D-H di cui non parleremo. Basati più sui log discreti che su DH. Non sono DH naturalmente quindi attenzione, **ricordiamo che DH non è un cfrario quindi qualunque tipo di cfratura basata su DH significa qualcosa di diverso visto che si ispira a DH ma non è la stessa cosa.**

DAL PUNTO DI VISTA PRATICO

Con openssl si possono fare tutte queste operazioni che si fanno con tutti gli altri cfrari a kpub, si generano i parametri e si può notare che non specifica se viene usato DH o DSA perché con DH bisognerebbe trovare un generatore in modo che vengano generati tutti i possibili valori in modulo p però spesso si usa una G tale che in modo che le potenze assumano un numero di valori pari a q e non a p è semplicemente un modo per avere gli stessi parametri e usare DSA oppure D-H.

Non ci fa vedere esercitazione, i comandi sono gli stessi quindi non aggiungerà molto al meccanismo in più possiamo vedere solo l'elenco delle curve. Si generano chiavi pubbliche, si estraggono chiavi pubbliche ecc. Si firma allo stesso modo e si può generare il segreto nello stesso modo e non si vede nell'utilizzo pratico se uno sta usando quello a curve ellittiche oppure no, lo vede solo dalla lunghezza delle chiavi e non è che sia molto utile un esercizio di questo genere.

CIFRATURA WHATSAPP(NON LA CHIEDE):

Tutta la cifratura di WhatsApp è basata sulle curve ellittiche e può essere interessante vedere come funziona il protocollo. Utilizza come crittografia a chiave pubblica quella a curve ellittiche. La curva che usa è quella 25519 di cui ci parlava che è valutata safe e non è protetta da copyright e inoltre è una delle curve più veloci. Il client la prima che si installa WhatsApp genera un identity key pair a lungo termine (coppia chiave privata e pubblica. Poi genera delle chiavi firmate dalla chiave e poi delle chiavi ONE time (usate una volta sola) . Queste chiavi vengono generate all'installazione e vengono mandate tutte le parti pubbliche al server e mai le chiavi private. Il server avrà una cosa di chiavi pubbliche e quando finiscono vengono riempite dal client.

Le sessioni sono una per chat e viene inizializzata a meno che si perdano dei dati. Initiator chiede al server ID pub, Sig Pub e una OT Pub. Se per caso avesse finito le chiavi pubbliche di quell'utente non le manda. Chi comincia conversazione calcola un master secret e si hanno recipient è quello che riceve. Quello che comincia (Initiator) crea una coppia di chiavi effimere pubblica e privata e mettere come iniziatore la propria chiavi privata. Si calcola master_secret con ECDH (DH a chiavi ellittiche) con vari concatenamenti.

Adesso ha questo segreto aster legato a parametri a cui hanno contribuito tutti. quindi attaccante che non conosce chiavi private fa molta fatica calcolare master secret che contiene cose private. A partire dal master secret un generatore di sequenze pseudo casuali HKDF(key derivation function) basata su HMAC a partire dal master secret 32 byte per una chiave root e 32 byte per una chiave chain. Questa è quella usata per TLS. A questo punto abbiamo queste due chiavi. Finché il destinatario non risponde e quindi se uno manda messaggi chi invia messaggi chi comincia nell'intestazione del messaggio mette tutte le tavole di dati per le chiavi → E Initiator e l Initiator che consentiranno al destinatario di costruire le chiavi. Finché uno non risponde potrebbe andare persi i messaggi e chi inizia chat mette proprie chiavi pubbliche per mettersi 'd'accordo sul master secret. Destinatario calcola master secret, root e chain allo stesso modo e cancella chiavi pubblica e verrà usata una sola volta.

Come funziona cifratura dei messaggi? Ha bisogno di una chiave di messaggi da 80 byte e ha 32 byte come chiave AES-256, 32 byte chiave HMAC SHA256 e 16 byte come IV per CBC poiché cifrato con AES 256 CBC. Si fa HMAC-SHA 256 della chain key in esadecimale.

Chain KEY cambia ad ogni messaggi.

Questo meccanismo delle chiavi si chiama double ratchet algorithm. La chain key viene aggiornata e anche la message key. Questo succede anche se interlocutore non risponde. In aggiunta con ogni messaggio viene inviata anche una nuova chiave pubblica effimera quindi quando destinatari risponde include la propria chiave effimera e calcolano un segreto effimero . La chain e la root vengono rigenerate a partire da questo segreto effimero.

Succede che se anche una chiave venisse compromessa (quindi chiave che cifra un certo messaggio) da quella message key è difficile ricavare HMAC chain key e a quel punto la chain key è molto diversa e quindi se l'attaccante riesce a sapere la chain key riesce a sapere quelle successive ma non quelle precedenti poiché 'è hash in mezzo. Permette di tutelare i messaggi precedenti. I messaggi successivi non riuscirà a decifrare poiché c'è la cosa che dà message key è difficile trovare la chain key. Inoltre siccome c'è questo scambio di chiavi effimere continuo che aggiorna la root key ogni volta che i due hanno avuto successo con lo scambio dei messaggi, la root key cambia spesso Anche se riesce a trovare message key non gli serve a niente perché dalla root key si genera tutto. Lo scopo è la forward secrecy, quindi non posso trovare i messaggi successivi.

plausible deniability, se qualcuno decifra un messaggio non può dire di essere stato lui. Quindi c'è questo dinamismo. In tutto questo i messaggi possono essere persi o non arrivare in ordine avendo un meccanismo robusto. Questa è la security end-to-end. I server in teoria il messaggio una volta che lascia il vostro cellulare viaggia cifrato e ha tutta questa protezione. Cosa succede con WhatsApp web? Il cellulare scambia delle chiavi di sessione con il browser e i messaggi arrivano al cellulare che li cifra e li decifra e in quel modo c'è di nuovo end to end secrecy dove i dati vengono decifrati dal browser in locale. Per trasmissione di video e musica cosa succede? il mittente genera una chiave di sessione AES 256, chiave 256 bit per HMAC e un IV pseudo casuali, cifra allegato e carica il file cifrato online e invia chiavi e iv cifrati con un normale messaggio WhatsApp. Quindi destinatario carica file e lo decifra a chiave simmetrica. Da notare schema ibrido in ogni caso. Nel gruppo non tutti hanno la stessa chiave, quindi il mittente calcola una chiave chain e una coppia di chiavi per la firma, compila un messaggio con chain key e signature pubblica e li usa in modo separato ad ogni utente cifrato nel solito modo. Per i messaggi message key e chain key vengono aggiornate e generate allo stesso modo. Integrità del messaggio è firmata con la chiave di firma e in questo modo in tutto il gruppo una coach ha scritto l'ha scritta lui. In HMAC si dava fiducia al gruppo. Chi lascia gruppo viene escluso crittograficamente.

Per le chiamate protocol SRTP e invia un messaggio insieme alla richiesta di chiamata. Tutto questo lascia aperto attacchi man in the middle dato che non rimangono autenticato. Il meccanismo di verifica è con un out-of -band verification che ci fa vedere chiave pubblica su un settaggio diverso e si verifica che uno usi have i quella persona. Sotto a tutto questo c'è una sicurezza a livello trasporto che è fatta su noise pipe(alternativa a TLS) e usa scambio di chiavi con la stessa curva con AES_GCM e SHA256. Con chiavi a 2048 bit il telefono non ci riuscirebbe.

AUTENTICAZIONE DI UTENTE O ENTITÀ: SCHEMA AAA

Oggi affrontiamo il problema dell'autenticazione. In questo caso parleremo di **autenticazione di utente o identità, quindi autenticazione sicura**. Nel caso di scambio della chiave pubblica abbiamo visto che si certifica la chiave pubblica per dimostrarne l'appartenenza. Questo però è un aspetto di un problema più ampio dell'autenticazione di un utente. **E' il paradigma AAA (Authentication, Authorization, Accounting): quando si ha un sistema si deve essere in grado di autenticare chi richiede il servizio e identificarlo in modo sicuro. Avendo identificato il soggetto dare al soggetto i permessi corretti relativi a quel servizio.**

Accounting → registrare quello che fa utente per varie ragioni (economiche, di sicurezza e altro).

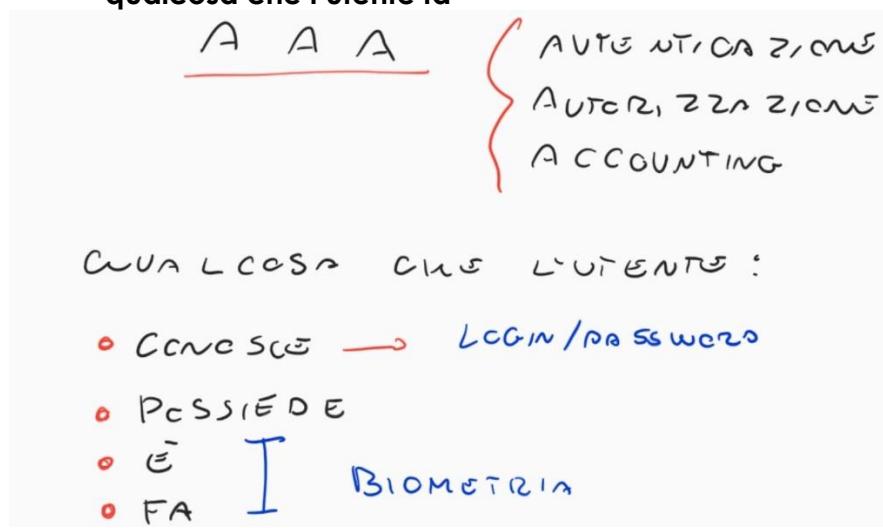
Una volta che ho la certezza del soggetto che utilizza un servizio devo essere in grado di autorizzarlo, che deve essere fatto a priori rispetto a decisioni che devono essere prese in precedenza. C'è dietro tutta la parte di decidere chi deve fare cosa. Ci sono vari modelli che si possono sfruttare per scegliere cosa fare. Non parleremo di autorizzazione nel dettaglio ma questo è il contesto. Non parliamo neanche di accounting in particolare ma era necessario specificare il contesto. **Quello che è importante da capire è che autenticazione e autorizzazione sono due aspetti differenti.**

COME OTTENERE L'AUTENTICAZIONE

Autenticazione come si ottiene?

Nel mondo reale per identificare una persona la si guarda in faccia e quello che uno ha è il viso infatti i documenti di riconoscimento hanno una fotografia. Ci sono vari modi per identificare una persona, **è necessario che l'entità debba presentare in qualche modo un'informazione che identifica in modo univoco il soggetto, questa informazione può essere di diversa natura ovvero:**

- qualcosa che l'utente conosce → segreto che solo utente è a conoscenza (login e password)
- qualcosa che l'utente possiede
- qualcosa che l'utente è
- qualcosa che l'utente fa



- **Qualcosa che l'utente conosce: Un segreto di cui solo l'utente è a conoscenza.**
Per esempio, login e password.

Problemi di login e password: la password viaggia in rete quindi se non è protetta opportunamente a livello di crittografia ci possono essere problemi. Deve essere riconosciuta a destinazione e quindi deve essere riconosciuta in qualche modo da qualche parte. **Il server deve mantenere la password per confrontarla. Il modo corretto di mantenere una password in modo che sia cifrato così che se l'attaccante trova il file delle password non riesce a trovare le password e inoltre deve essere cifrato con una cifratura non reversibile neanche dal server**(in quanto il server non deve essere a conoscenza delle password in chiaro). **Normalmente le password sono memorizzate con un hash o come un HMAC che usa molto spesso una parte della password.** Avevamo fatto il discorso del sale (salt), se ogni password è cifrata con sale diverso anche se sale conservato in chiaro insieme alla password, l'attaccante può fare un attacco forza bruta su una singola password come prima però non riesce a fare un attacco a forza bruta su tutte le password insieme. Su 1000 password una debole ci sarà sicuro e quindi su mille password si potrebbe fare un'unica cifratura per capire di chi è la password. Se c'è il sale non può fare questo lavoro "all'ingrosso" e quindi deve fare per forza password per password e diventa più difficile. Però qui vengono fuori tutti i problemi tipici: password comuni, ovvie e semplici. **Se uno usa una password banale l'attaccante fa attacco forza bruta su dizionario ristretto ed è facile risalire alla password.** Se password è corta con attacco forza bruta è facile. **Per questo sistemi di password suggeriscono almeno 8 caratteri e caratteri speciali. Meglio avere password che sono lunghe e sequenze di parole che uno ricorda ma molto lunghe.** Se il sistema non è facile da usare per l'utente allora utente cercherà di raggiungere il sistema. Se chiedo 16 caratteri utente la scriverà su un foglio e quindi sarà attaccabile con social engineering. Se si chiedono 12 password del sistema, utente le metterà tutte uguali. Quindi bisogna pensare sistemi di sicurezza che vadano nei confronti dell'utente. Se la password è lunga e complicata utente non farà logout perché è una scocciatura inserire la password.

Se invece poi la password viene buttata via non c'è il problema che attaccante riesca a risalire alla password con un attacco a forza bruta.

Qual è il problema di password uguali? se uno ha tante password uguali su tanti server ce ne saranno alcuni sicuri e altri che la terranno in chiaro oppure la manterranno con una protezione debole(quindi sono server che mantengono le password in modo non sicuro). Quindi chi attacca un utente può provare in tutti i siti che gli interessano per vedere se è stata usata la stessa password altrove e la scopre. Il sistema debole mette la password a disposizione dell'attaccante e quello forte non sarà più forte ma diventa debole poiché c'è un sistema più debole di lui. Questo è basato tutto sull'attività della conoscenza e non è sotto il mio controllo se ho qualcosa salvato da qualche parte.

In sostanza il problema principale come avevamo già detto è l'utente.

- **Qualcosa che l'utente possiede**

La chiave di casa è una forma di autenticazione per l'utente legittimo è quello che ha in mano alla chiave. Però se uno le perde chi la trova diventa utente legittimo e mi può entrare in casa. E' un oggetto che uno dovrebbe portarsi dietro, può andare perso e c'è un costo per mantenerlo e di solito c'è hw per gestire quell'oggetto, mentre nel caso della password a volte è un Software che gestisce il sistema.

Qualcosa che l'utente È e FA fanno parte dei sistemi biometrici!

- **Ciò che l'utente E'**

PROBLEMA DI E' Significa **impronte digitali, riconoscimento facciale, scan della retina ecc.. Il sistema usa caratteristiche statiche dell'utente**. Ha praticità che non segreto **non si può trasferire e uno non dimentica perché uno non perde la faccia, da questo punto di vista è vantaggioso ma c'è il problema dei falsi negativi**: A volte il sistema del riconoscimento delle impronte non le riconosce perché il dito è sudato, secco, sporco o altro. **Un sistema di autenticazione non deve avere assolutamente falsi positivi**: Non riconosce un soggetto A come se fosse C. C sostiene di essere A e il dispositivo lo si riconosce come A.

Il Falso negativo: A cerca di entrare nel sistema ma il sistema non lo riconosce. Un sistema di autenticazione non deve avere falsi positivi ma per essere efficace deve avere meno possibili falsi negativi. **Deve essere raro e se non lo è influisce sull'usabilità di un sistema**. I sistemi biometrici hanno una sorta di praticità. . Ci può essere un sistema di mezzo che riconosce l'impronta e una volta riconosciuta viene trasformata in informazione e una sorta di bit. Chi fa man in the middle tra impronta e sistema dietro potrebbe intercettare un impronta legittima e riutilizzarla.

- **Qualcosa che l'utente FA**

E' un aspetto dinamico, ad esempio riconosce quello che utente scrive sulla tastiera. Con IA si può riconoscere una persona nel modo in cui digita ma spesso sono soggetti a falsi negativi.

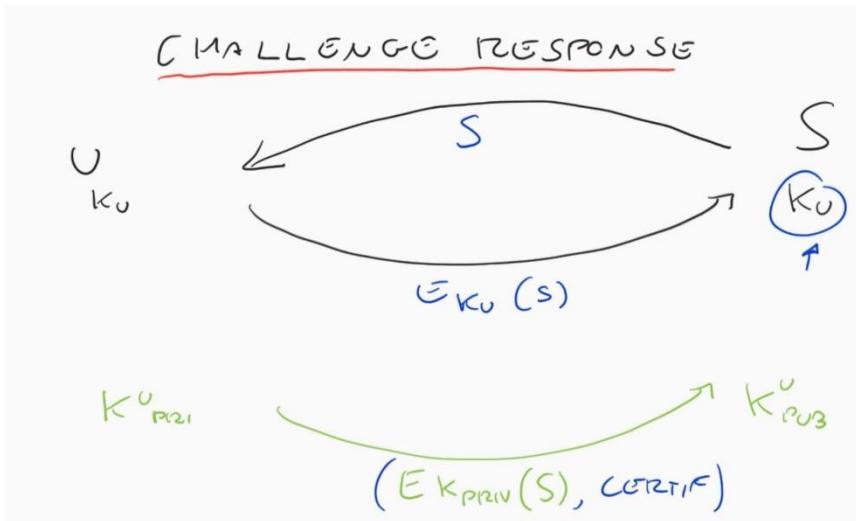
Per il sistema al momento in cui uno sceglie il sistema di autenticazione, l'utente legittimo diventa quella password, quella chiave, quel dito o quel modo di digitare e non c'è altro modo per identificare l'utente se non il segreto che si è detto associare all'utente. Ognuno di questi sistemi ha i suoi lati positivi e negativi e si usano meccanismi cosiddetti di autenticazione forte(two-factor) Sono meccanismi che usano contemporaneamente due di questi fattori qui. Di solito sono conosce e possiede. Esempio sono la banca oppure sul cellulare che genera una chiave onetime che identifica utente che mette una password e poi viene identificata da un altro numero che genera da un dispositivo.

Sono dispositivi tump resistant quindi resistenti a manomissioni come per esempio chiavette che producono chiavi OTP che avranno un seed che dipende dall'utente e sono basati o su funzioni hash quindi sul numero per cui a partire dal seed vengono generate sequenze di chiavi e questo significa che il server dall'altra parte ha lo stesso seed e la stessa funzione hash per verificare utente. Se l'utente e il server vanno out of sync bisogna risincronizzare i sistemi, generalmente i server hanno meccanismi per cui valutano una finestra temporale e se l'utente si sposta all'interno di quella finestra se ne accorgono oppure sono basati su data e ora le password generate e quindi il numero di password generate dipende da data e ora e quindi non c'è il problema di spreco delle password se l'utente sbaglia la password ma gli orologi devono essere sincronizzati. Sono basati su funzioni hash.

Non importa come uno pensa un sistema di sicurezza esisterà sicuramente qualcuno che cercherà di raggiarlo. Nell'autenticazione a due fattori, per essere vulnerabile un utente deve perdere token e anche password nello stesso tempo. Infatti i bancomat alla gente viene raccomandato di non tenere pin e bancomat assieme altrimenti si vanifica il doppio sistema. Sono molto comuni sistemi in cui si basano sul possesso di cellulare e in questo modo si aggiunge ai due fattori il doppio canale così che se uno lavora su un computer compromesso, mi arriva sul cellulare la richiesta di approvare il bonifico per quella cifra a quel destinatario. Se il browser fosse compromesso con un man in the middle che intercetta dati della banca e me ne manda altri e chiede bonifico da 100k euro all'attaccante mentre io ne vedo 100€ al padrone di casa allora qualunque meccanismo di sicurezza se attaccante è un mezzo può essere vanificato dall'attaccante. Però il fatto che mi arrivi codice sul cellulare è difficile che attaccante riesca a compromettere sia cellulare che browser è difficile. E' una successivo rafforzamento del sistema. E' un'autenticazione a due fattori perché il messaggio mi vien mandato su quella sim.

Però tutti questi sistemi in sé finora per come ne abbiamo parlato rimane il problema di dati che passano in rete e dati che vengono registrati dall'altra parte. Si possono eliminare queste due cose e diventano interessanti i sistemi di autenticazione.

EVITARE TRANSITO CREDENZIALI IN RETE- SISTEMA CHALLENGE RESPONSE



Si usa un meccanismo che chiede all'utente di dimostrare di possedere quelle credenziali (challenge) ma non richiede il trasferimento di quelle credenziali. Scenario: l'utente e il server condividono una chiave simmetrica, il server chiede all'utente di cifrare qualcosa. Se nell'ipotesi che utente e server siano unici a condividere questo segreto, questa è la dimostrazione che il server sta parlando con l'utente ed il segreto non transita in rete. Il meccanismo descritto va nella categoria della challenge response.

Per ogni utente il server mantiene una chiave dell'utente, e questa chiave deve essere trasmessa a priori (fase di setup), si può pensare ad una password e i due non condividono una password ma una chiave che deve essere in chiaro presso il server perché il server la deve poter utilizzare).

Il server chiede all'utente di cifrare una stringa S pseudocasuale che cambia di volta in volta. Successivamente l'utente invia S cifrata con la sua chiave K_U che condivide col server. Quindi è una password one time, e quindi usata una volta sola, le credenziali non viaggiano in rete, ma viaggia un prodotto delle credenziali. Se il sistema crittografico è buono, è difficile da testo cifrato risalire alla chiave di cifratura e pertanto le credenziali (ovvero la chiave K_U) rimangono al sicuro. Il server decifra e verifica che è S sia uguale a quello che è stato decifrato e in caso positivo sa che dall'altra parte c'è qualcuno che conosce la chiave, ma non sa chi è; nel corretto funzionamento di sistema si suppone che sia K_U . Il problema di K_U è che è salvata nel server per essere utilizzata.

Sistemo le password a priori e dopodiché la chiave non ci passa più in rete e naturalmente il canale deve essere affidabile.

Il problema che rimane non è che la chiave passa in rete ma il problema è che il server ha bisogno di ottenere le credenziali dell'utente (K_U). Lo si può evitare usando un sistema challenge response di scambio di K_U a Chiave pubblica: in questo caso il server ha $K_{U_{PUB}}$ e utente ha la propria chiave privata $K_{U_{PRIV}}$. Dopo la challenge (server manda S), la response dell'utente sarà la cifratura di S usando $K_{U_{PRIV}}$, quindi una sorta di firma digitale (potrebbe esserci impronte in mezzo). A questo punto il server ha la chiave pubblica e quindi non passerà mai la chiave privata. Per verificare, passa in rete un testo cifrato e a destinazione viene verificato con la chiave pubblica.

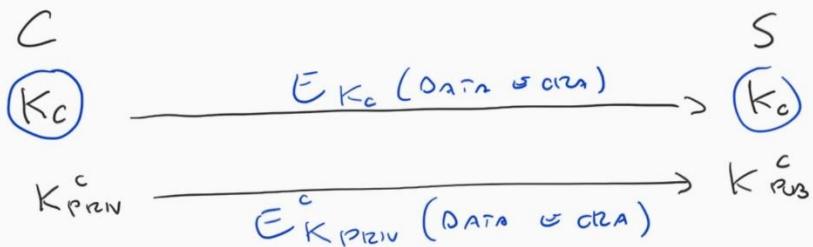
Per verificare che ciò che arriva al server sia effettivamente dell'utente U si manda il certificato che viene estratto oltre a mandare al server la S cifrata con $K_{U_{PRIV}}$ e il server verifica veridicità del certificato.

C'è prima la stringa pseudo casuale e poi cifratura e questa cosa può essere ritenuta troppo pesante. Si può fare la stessa cosa basata sul time-stamp per dire data e ora.

MECCANISMO BASATO SUL TIMESTAMP-DATA E ORA

TIMESTAMP

DATA E ORA



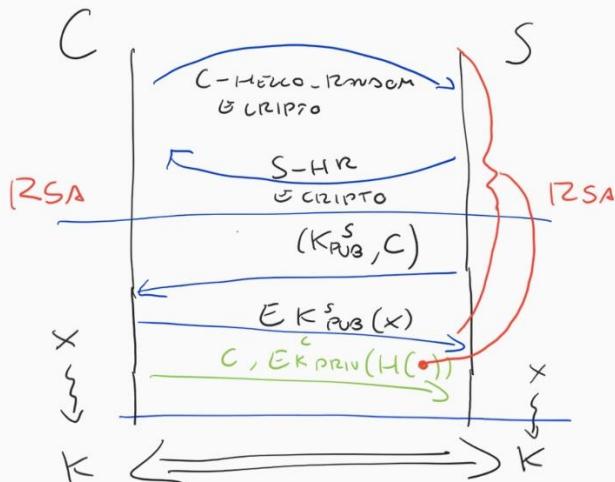
Il server va prima a vedere a che data e ora decifrare. A questo punto l'utente (Client) vuole autenticarsi presso il server in un certo modo (sia a chiave simmetrica che a pubblica). A questo punto allora l'utente C manda cifrando con la propria chiave simmetrica data e ora oppure cifra data e ora con la propria chiave privata in caso di sistema a chiave pubblica. **Rispetto al challenge response dove si cifra la Stringa pseudo casuale S, si cifra data e ora.** Sulla rete non si vedono passare credenziali, ma una sorta di password che non è onetime ma in realtà è a breve durata. Questo meccanismo funziona se il server decifra e verifica che data e ora siano corrette. Quindi prima di tutto Client e server devono essere sincronizzati per fare in modo che funzioni tutto. Anche se sono sincronizzati può esserci latenza e per renderlo utilizzabile bisogna che il server ammetta una certa finestra temporale entro la quale accetta i messaggi con una certa latenza e desincronizzazione degli orologi. In una certa finestra un attaccante può riutilizzare quello che ha visto passare, **Se la finestra dura 2 Min, l'attaccante ha 2 minuti per utilizzare i dati visti.** Più la finestra è grande, più tempo si dà all'attaccante, altrimenti se troppo corta non si dà il tempo di accettare connessione tra C e S. In questo caso quindi è preferibile il sistema basato su challenge response.

Sono particolarmente importanti questi 2 meccanismi e cerchiamo di capire Challenge response soprattutto quello base pubblica!!!! TLS SI BASA SU QUESTO

TLS vediamo che c'è un cambiamento tra TLS 1.2 e quello 1.3. Vedremo TLS 1.2 che è più istruttivo.

AUTENTICAZIONE TLS CON RSA

Nella fase di scambio di chiavi con TLS può avvenire l'autenticazione se richiesta e tipicamente avviene ormai nel server, allo stesso modo il server potrebbe chiedere l'autenticazione dell'utente. Con RSA abbiamo detto che nell'autenticazione il server manda al client la chiave RSA per lo scambio di segreto.



Client-To-Server: C_hello_random e capacità crittografiche

Server-To-Client: S-HR e capacità crittografiche

Supponiamo di usare RSA per lo scambio del segreto (che non è più ammesso in TLS1.3)

Il server manda la sua chiave pubblica RSA al client, che a sua volta genera una stringa pseudo casuale e la manda al server cifrando con la chiave pubblica del server. Il server decifra e viene generata una chiave di sessione con la quale i 2 hanno il loro canale privato. Se il server oltre alla chiave pubblica manda anche il certificato associato a quella chiave, il Client verifica la validità del certificato e vede che effettivamente è quella la chiave pubblica del server. Il client deve fidarsi della CA e spesso accade che il browser del client non si fidi della CA e viene sollevata un'eccezione. Un altro fatto è che il Client (browser) deve verificare che il soggetto con cui sta parlando sia proprio il server. Il certificato non è rilasciato alla stessa URL che compare e quindi il browser può dire: no questo certificato non va bene.

Siamo tranquilli che il server è chi gli dice di essere al punto E_{pub} (X)? Se al posto del server ci fosse stato un attaccante sarebbe cambiato qualcosa dal punto di vista del client? **La vera autenticazione avviene dopo quando il server deve utilizzare X, ovvero la chiave K.** Nel protocollo ciphersuite tutti si scambiano i messaggi precedenti cifrati con la chiave K. Tutti i messaggi precedenti dipendono da questa sessione soprattutto perché ci sono dentro 2 hello random e sono una sorta di challenge. I due la cifrano, però scambiandosi un messaggio cifrato con la chiave K il client sa che il server è riuscito a recuperare X e che è in possesso della chiave privata corrispondente alla chiave pubblica e avendo la chiave privata è a tutti gli effetti il server.

Quello che manca fin qui è che un certificato è un oggetto pubblico e questo potrebbe averlo inviata chiunque. La vera autenticazione è quando il server dimostra di avere la chiave privata e lo implicitamente dimostra quando riesce a ottenere la chiave k giusta perché ha la chiave privata e ed è a tutti gli effetti lui. Il client non è autenticato in alcun modo. **Quando lo scambio è basato su RSA, il server può chiedere al client di autenticarsi. Il client richiede una chiave di firma RSA e perché in questo caso hanno scelto RSA e a questo punto anche lui deve dimostrare di avere la chiave privata precedente e per farlo dovrà firmare qualcosa,** in generale **firma tutti i messaggi fino a $E_{pub}(x)$ che contengono una stringa hello_random.**

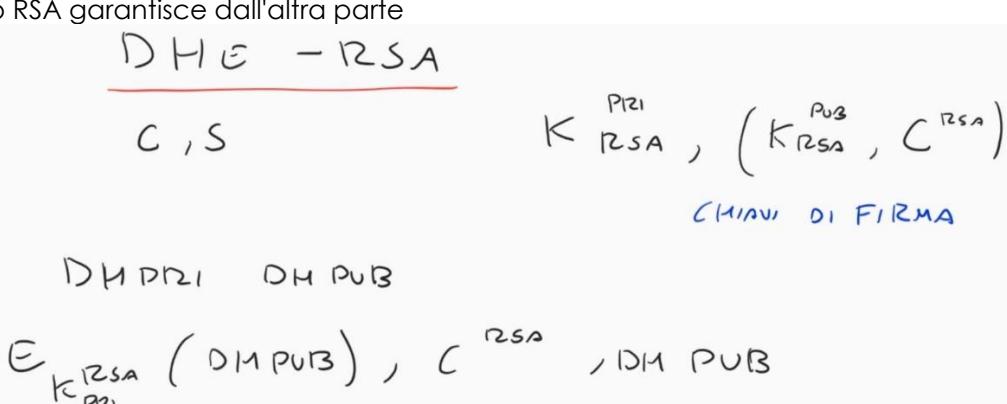
Nell'ultimo anno si è cercato di eliminare certificati con firme rilasciati su MD5. Adesso i browser non accettano più certificati di Server di MD5. E' stato fatto un attacco nel quale degli hackers hanno usato una CA e hanno usato un meccanismo per trovare collisioni in MD5 si son fatti rilasciare un certificato dal server avendo pronto il secondo certificato che aveva la stessa impronta. A questo punto han potuto mettere su un sito certificato correttamente, di cui si ha chiave pubblica e privata che SEMBRA quello originale, in realtà ha solo sfruttato una collisione.

A parte i possibili attacchi sulle collisioni potrebbe essere più facile attaccare una CA e farsi rilasciare certificati.

DIFFIE-HELLMAN CON RSA

DH con RSA è un metodo di scambio di chiavi, il metodo DH-RSA TLS significa che i 2 si scambiano le chiavi utilizzando RSA, ma ciascuno dei 2 ha chiave DH, mentre

RSA è l'algoritmo di firma usato per certificare la chiave DH. Basano lo scambio su DH ma autenticazione avviene con RSA. Analogamente a questo c'è DH-DSS (che vuol dire DSA data signature standard/algorithm), ECDH-ECDSA entrambi a curve ellittiche. Poi abbiamo tutti questi meccanismi non ammessi con "no" a fianco nel nuovo TLS, quelli ammessi sono quelli azzurri tranne DHE-DSS. Perché DHE? DH effimero quindi significa che le chiavi sono generate lì per lì. Prima erano permanenti e certificate, quelle effimere non possono essere certificate perché create in quel momento. Qual è il vantaggio? Le chiavi usate in una sessione vengono poi gettate via, perciò il segreto generato vale solo per quella sessione. Per questo si parla di **forward secrecy** → il segreto che c'è adesso è protetto nel futuro quindi l'attaccante in un futuro non può decifrare messaggio venendo a conoscenza della chiave. Dato che non vengono più riutilizzate le chiavi, quella chiave lì non esiste più. Vengono firmate con una chiave RSA che a sua volta è certificata perché appunto le chiavi effimere non possono essere certificate. Quindi l'utente genera la chiave pubblica e privata con DHE, firma la propria chiave pubblica con la propria chiave RSA, manda chiave DH pubblica e firmata insieme al certificato della chiave pubblica RSA. così il certificato chiave pub RSA garantisce dall'altra parte



C, S ciascuno genera le proprie chiavi DH_{priv} e DH_{pub} e ha una chiave RSA K_{RSA}^{priv} privata e K_{RSA}^{pub} pubblica, rilasciata per la firma con tanto di certificato C^{RSA} . Queste ultime due sono dette chiavi di firma. Il client firma (cifra) DH_{pub} con K_{RSA}^{priv} , e manda questo assieme al certificato RSA e a DH_{pub} in chiaro. A destinazione si verifica dal certificato che la K_{RSA}^{pub} sia effettivamente legata a quel certificato C^{RSA} . Manca ancora un pezzettino, perché tutto questo funziona se e solo se il client è in possesso di DH_{priv} , altrimenti potrebbe aver intercettato questi dati e averli replicati (attacco replay) e fingere di essere A quando non lo è. Si sa che il client possiede DH_{priv} quando genera la chiave di sessione: se è quella giusta avrà sicuramente usato DH_{priv} per generarla. Il protocollo prevede che sia effimera, ma poi uno la genera come vuole. Nella firma ci sarà anche un timestamp per evitare qualunque problema e tutto questo funziona. Il fatto che il mittente abbia firmato questo DH_{pub} dimostra che ha la K_{RSA}^{priv} e dimostra di essere lui. Il fatto che questo oggetto sia trasferito non dimostra che effettivamente è lui, perché la chiave pubblica l'ha inventata lui, ma se c'è un timestamp dentro è più improbabile ma non impossibile, quindi l'unico momento in cui tutta la catena è corretta è quando DH_{priv} viene usata e quando il soggetto ricava il corretto segreto DH con la corretta chiave di sessione. Se l'attaccante non ha DH_{priv} non riesce a fare nulla. Attaccante potrebbe avere un'altra chiave privata e pubblica, ma non può firmare. L'altra parte non vede azione di firma e potrebbe essere che questi sono stati riutilizzati. Dato che tutto quello che viaggia in rete può essere riutilizzato, il meccanismo deve avere un punto in cui meccanismi di chiave privata non viaggiano in rete e infatti si usa chiave di sessione.

Non ha nessuna importanza conoscere questi protocolli ma è un esercizio di ragionamento sull'utilizzo corretto di questa crittografica e lo si usano per esercizi.

PSK → Pre-Shared-Key c'è uno scambio a priori e serve per l'autenticazione

Kerberos → Vengono resi i certificati a chiave simmetrica, l'utente si autentica presso il server di autenticazione e il server di autenticazione rilascia un ticket che sono dei dati rilasciati alla fine da usare su un server. L'authentication server lascia un ticket "questo client lo puoi fare girare fino alle 16" questo messaggio viene cifrato con la chiave simmetrica che condivide con il server e quindi quando l'utente presenta questo messaggio al server questo lo riconosce, lo valida e offre il servizio.

=====>AUTENTICAZIONE E' IMPORTANTE E CI TIENE (ORALE)<=====

Configurazione server apache locale e non funzionava per un dettaglio. Quindi ora rifacciamo esercizio che prof si è dimenticata un comando

A questo punto la nostra CA è verificata e il server si fida.

C'è il modo di avere il file della chiave server con la password e mettere la password lì.

A questo punto, ci sono tanti tentativi di rendere meno vulnerabile il meccanismo del rilascio di certificati. Se uno riesce a compromettere una qualunque delle CA di cui si fida il browser, il problema è che può rilasciare un certificato per qualunque sito e c'è un problema abbastanza pesante e l'idea era legare verifica di certificazione al DNS. Esiste tutto un meccanismo per fare in modo che sia lo stesso DNS oltre a dare info solite dia informazioni su come è certificato un certo sito. Uno può sempre compromettere DNS e scriverci. Una debolezza è che questo permette di definire in varie modalità di autenticazione del sito tra cui un certificato auto rilasciato e se uno entra nel server DNS può modificare queste informazioni. Se uno deve violare un server specifico deve andare direttamente al DNS di zona e deve essere nelle vicinanze fisicamente ed è più difficile per l'attaccante.

Siamo arrivati all'ultimo argomento fondamentale del corso che sono i dispositivi di sicurezza.

Se avremo tempo faremo esercitazione su browser che però non sarà in programma.

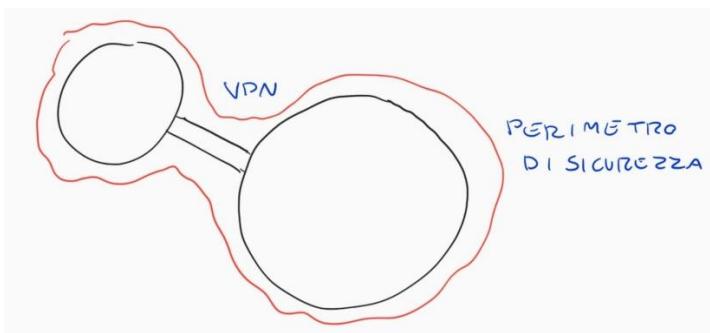
DISPOSITIVI DI SICUREZZA

PERIMETRO DI SICUREZZA



Possiamo distinguere per ogni azienda un **perimetro di sicurezza** che si può immaginare come un'area controllata anche fisicamente dall'azienda, viene tipicamente assimilato a un castello medievale. Ha la caratteristica di essere protetto anche fisicamente. **Avere una rete isolata è raro e il collegamento della rete con l'esterno è molto importante ma qualunque collegamento con l'esterno porta dei problemi.** Noi Abbiamo lo studio della crittografia che ci ha permesso di pensare a dei protocolli come TLS che permettono di avere delle connessioni virtualizzate per quanto riguarda la riservatezza e integrità inoltre abbiamo visto strumenti come la firma digitale i quali sono tutti strumenti che permettono di uscire dal perimetro di sicurezza PROTETTI. Quindi per analogia al castello medievale corrisponde al soldato che esce con l'armatura. **Se ho 2 perimetri di sicurezza disgiunti, una connessione sicura tra i 2 non esiste** (a meno di un cavo dedicato) **però se uno volesse farlo potrebbe usare una VPN anche su grossa area geografica.**

VPN



Cosa succede?

La crittografia permette di estenderlo ma non è solo questo tipo di rapporto con l'esterno, si vuole anche un rapporto reale con l'esterno, per esempio il server web per il quale si vuole che si possa accedere anche dall'esterno. Quindi sono indispensabili contatti verso l'esterno di cui non ci si può fidare.

Se un'azienda si connette con una banca, la banca non deve sottostare alla sicurezza dell'azienda.

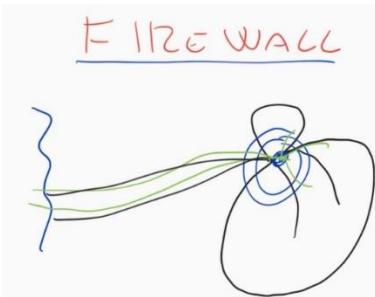
All'interno di un'azienda è possibile avere livelli diversi di requisiti. Questo è vero anche all'interno di una stessa organizzazione come l'università dove abbiamo livelli diversi di requisiti. Abbiamo amministrazione che deve essere gestita ad un livello di sicurezza più alto. C'è la rete dei docenti che hanno bisogno per lavorare con una certa libertà diversa da quella dell'amministrazione e ci sono laboratori di studenti che sono iscritti all'università però non sono dipendenti e il rapporto non è lungo come per i docenti. (SI SPERA CHE GLI STUDENTI ESCANO PRIMA).

Il modo di trattare un laboratorio studenti non può essere trattato allo stesso modo della rete dei docenti, magari diversi laboratori hanno bisogno diversi tipi di livelli di sicurezza per esempio ci può essere il laboratorio di ricerca con una diversa gestione. Avevamo un laboratorio di sicurezza dove tutti potevano fare quello che volevano ma si può anche fare con net kit.

Uno deve configurare la propria rete impostando e unendo delle sottoreti che hanno requisiti di funzionalità differenti, poiché altrimenti non si potrebbe lavorare se si dovessero rispettare tutti i requisiti di sicurezza di tutti, ci sarebbero troppe regole da rispettare. Ci può essere il flusso di una rete collaboratrice e il flusso verso l'esterno. Come si gestiscono questi scambi di dati in modo da minimizzare i rischi di sicurezza?

Quello che si fa è **organizzare nel punto di uscita che dovrebbe essere uno solo** (ma potrebbero essercene più di uno) ,si organizza **quello che si chiama FIREWALL**.

FIREWALL



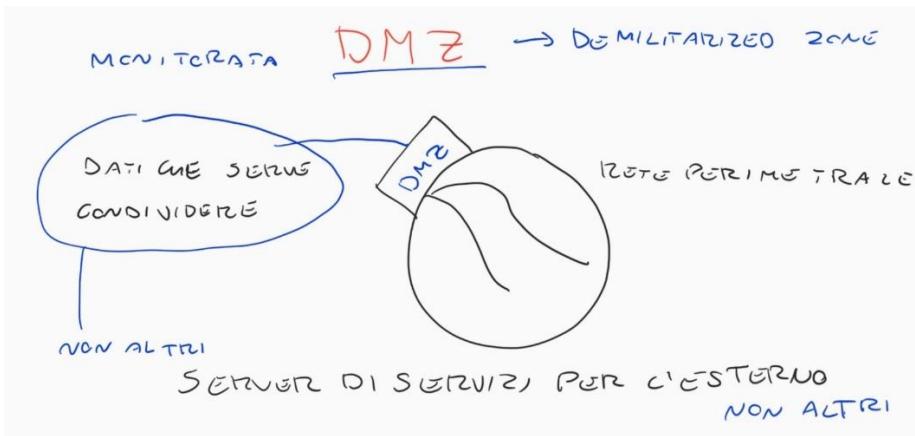
In generale, il termine **FIREWALL** indica un insieme di tecnologie Hardware e Software che servono per controllare il traffico. Notiamo che ha detto su un punto quindi nel momento in cui si configura un firewall ci si aspetta che tutto il traffico passi da questo firewall. Queste però non sono ipotesi ovvie e scontate come il perimetro di sicurezza. Tutti i cavi sono tirati e arrivano alle unità di controllo con switch che sono in stanze inaccessibili perché la sicurezza fisica è la prima base. Se uno riuscisse a staccare cavi e scambiarli fa un macello anche senza vere competenza. Così come le sale server sono chiuse dove uno potrebbe far danno senza avere alcuna competenza. Però questo non è sufficiente per dire che il perimetro di sicurezza finisce qui, abbiamo anche dei collegamenti wireless. Uno può parcheggiare la macchina fuori e molto probabilmente il segnale lo riceve anche da fuori quindi non è così ovvio che il perimetro finisce dove finiscono i muri della struttura. **La rete wireless deve essere trattata come una rete esterna.**

Poi non ci sono solo questi accessi dalla rete interna (il cerchio blu): potrebbe essere che l'utente abbia una chiavetta usb (quelle per collegarsi ad Internet come quella di Edoardo marcia) e possa collegarsi a Internet tramite questa oppure portare dei dati dall'esterno verso l'interno senza passare per il firewall. Questa ipotesi di lavoro dell'avere un perimetro di sicurezza con accesso localizzato in uno o più punti del firewall è un'ipotesi un pochino ottimista e non è così ovvio organizzarla. Uno potrebbe pensare a qualcosa di un po' diverso: **invece di blindare l'uscita, cosa succederebbe se provassimo a blindare le singole macchine?** In questo modo ciascuna macchina è sicura teoricamente, **un attaccante** cosa potrebbe fare?

1. **Potrebbe sempre fare DDOS alla rete interna perché non c'è una protezione di accesso.** Se uno avesse postazioni sicure ma la rete non lo è, se non è possibile accedere al database, o al server o mails allora non si può lavorare lo stesso.
2. **Rendere del tutto sicure delle macchine non è banale soprattutto in un ambiente come un'azienda** che comporta esigenze differenti. **In un ambiente con un controllo di sicurezza molto forte come una banca è fattibile dato che si hanno gli stessi Software e Sistemi operativi ovunque.** **In un ambiente come l'università ci sarebbe da spararsi per esigenze di studio e di ricerca poiché ci sono una varietà di SO di diverse versioni e Software installati** che riuscire a stare dietro a tutto sarebbe una cosa complicata. E' già difficile in un ambiente omogeneo figurarsi in uno non omogeneo.

Un limite del firewall è che controlla il traffico che lo attraversa, tutto il resto no. Un firewall incrementa il livello di sicurezza e rende più semplice la gestione. Un esempio è il box in cui metto a giocare il bambino. Dentro il box si potrebbe fare male però finché rimane dentro con i suoi giochi più di tanto non riesce a fare. I due aspetti, **firewall e sicurezza sull'host non sono contraddittori ma è buona cosa adottarli entrambi:** se l'attaccante dovesse violare difesa del firewall ci sarebbero da attaccare ancora macchine con un certo livello di sicurezza. Se l'attaccante dovesse violare la macchina perché l'utente ha aggiunto un altro canale di ingresso le altre sono comunque protette dal firewall è questa l'idea.(In sostanza la doppia protezione ci piace , un po' come al manzo)
Il firewall quali sono gli aspetti fondamentali?

SCHEMA DI RETE PRIVATA E PUBBLICA CON FIREWALL:



- **Reti interne o intranet:** Sono delle reti private, che hanno la necessità di contattare ed essere contattati dall'esterno. Se noi andiamo in un ufficio postale abbiamo accesso all'area del pubblico e quella che c'è dietro è l'area protetta dal pubblico.
- **DMZ (demilitarized zone o rete perimetrale):** La DMZ è una rete che fa da cuscinetto tra la rete interna e l'esterno, è molto importante perché crea una sorta di isolamento tra esterno e interno, è **pensata per l'accesso di utenza esterna non verificata** e quindi da una parte è una porzione della propria rete, dall'altra è pensata per l'accesso del pubblico. È una rete sulla quale ci saranno dati che sono da condividere con l'esterno, questi dati sono su dei server che devono essere lì per l'accesso con l'esterno. Deve essere ben monitorata perché non ci si fida di chi accede a quel server. Non ci devono essere dati che non si vogliono condividere con l'esterno (in quanto è molto soggetta ad attacchi) poi ci devono essere i server per i servizi con l'esterno e anche qui non altri. Se un servizio server lavora solo internamente non lo vado a mettere nella DMZ.

DMZ (DEMILITARIZED ZONE)

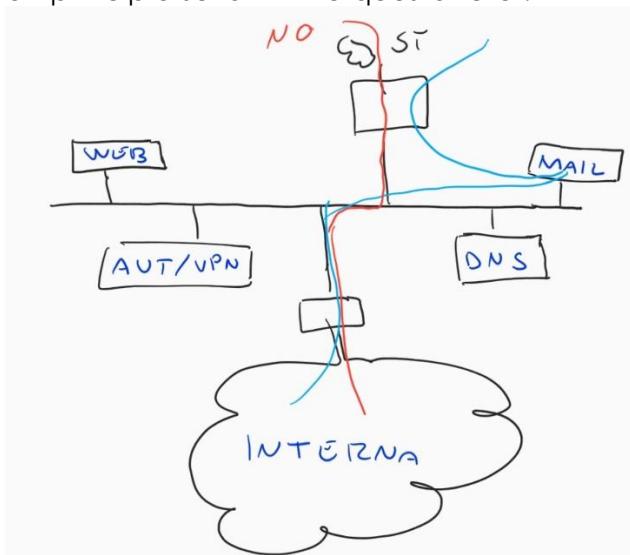
Che tipo di dati si possono mettere sulla DMZ?

SULLA DMZ

- WEB SERVER
 - MAIL
 - VPN/ ACCESSO IN REMOTO
 - DNS
-
- Web server
 - Mail server
 - VPN/Accesso in remoto
 - DNS server (se no non funzionerebbe la rete)

Ora questi servizi possono essere anche duplicati e replicati all'interno per esempio all'interno ci possono essere dei DNS che non si vuole pubblicizzare all'esterno e quindi per esempio possono esserci altri server diversi, un altro esempio potrebbe essere un mail server interno se c'è una preoccupazione di un attacco DoS. Soprattutto in caso di attacco che rende le connessioni inservibili verso l'esterno c'è comunque un mail server interno che è quello che continua a funzionare.

Un principio se la DMZ è questa rete :



che ha un accesso verso l'interno e un accesso verso l'esterno e qui ci sono web server, mail, DNS, autenticazione/VPN, **un principio fondamentale che deve rispettare la DMZ è di vietare connessioni dirette tra interno ed esterno**(**linea rossa**).

Ogni connessione si deve spezzare sulla DMZ. Per esempio con la mail funziona perfettamente, l'utente interno contatta il server mail per ricevere posta e sarà lui ad iniziare una connessione verso esterno(**linea blu**) quindi la connessione è spezzata.

VANTAGGI DMZ

1. **Un vantaggio immediato è che un attacco di saturazione di banda(DOS) non riesce ad arrivare all'interno poiché dovrebbe essere replicato dalle macchine sulla DMZ** (cosa che non avviene perché un attacco in DMZ a una macchina si ripercuote solo su quella macchina).
2. **l'attaccante non ha possibilità di ottenere un accesso diretto alla rete interna** (percorso rosso) **perché deve prima attaccare e compromettere uno dei servizi sulla DMZ** e quindi passare da uno degli host della DMZ, di conseguenza fermandosi su quella macchina in quanto non si propaga l'attacco.

Torniamo un attimo a quello che c'è sulla DMZ: ci sono i servizi che servono verso l'esterno e non altri, quindi sappiamo perfettamente i servizi che sono presenti di conseguenza quello che c'è sopra è monitorato.

Una macchina sulla DMZ prende il nome di Bastion Host. Nell'immagine medioevale si pensa a torri essenziali le quali non avevano scale ma erano oggetti monolitici senza appigli in modo che l'attaccante abbia pochi appigli per attaccare. Un Host sulla DMZ deve essere pensato in modo simile, ovvero come una macchina essenziale, che offre il servizio che deve offrire, dimensionata abbastanza bene (cioè deve essere in grado di gestire un certo numero di servizi/utenti), deve avere dei backup rimovibili perché se succede qualcosa al bastion host deve essere possibili ripristinare la situazione iniziale utilizzando un backup.

NON ci devono essere account utente e dati personali che viaggiano se non è strettamente necessario poiché introducono una vulnerabilità, e questi hanno bisogno di forte protezione. Le macchine sulla DMZ devono essere configurate sull'idea che verranno attaccate prima o poi.

POLITICA DI SICUREZZA (COSTI)

La volta scorsa abbiamo parlato di organizzazione della rete, sicurezza e firewall. A monte del lavoro tecnico di cui stiamo parlando ci deve essere una parte gestionale, non è solo un tecnico che si occupa della sicurezza della rete, perché ha un margine di scelta limitato. Ci sono sempre dei rischi perché i servizi portano dei rischi intrinseci e **qualcuno deve decidere quali sono i servizi che si vogliono offrire e quali sono i rischi che si accetta di correre**, quali sono i costi che si accetta di pagare per la protezione della rete.

Per costi si intendono:

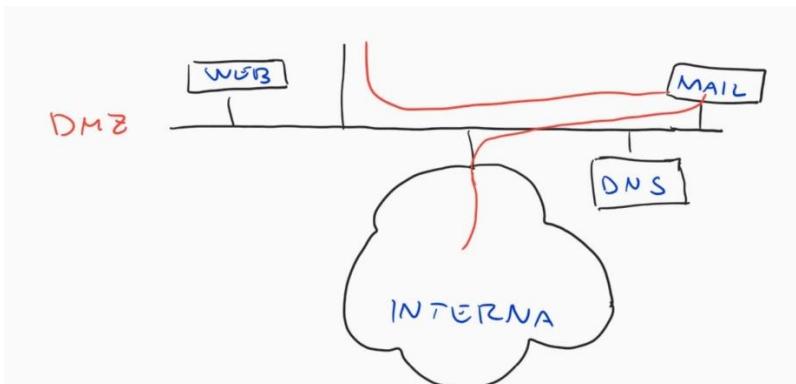
1. **i costi monetari** per comprare licenze software etc..
2. **costi di personale**, ovvero pagare gente che deve fare quello che è richiesto, decidere tra l'altro se farlo fare internamente o esternamente.
3. **costo in termini di peggioramento del servizio** perché le funzionalità di protezione vanno a incidere sul funzionamento del servizio.

Se do all'utente username e password per accedere , l'utente non può accedere e basta deve prima digitare username e password ; se io metto in mano un token a un utente lo costringo a portarselo dietro, più complicato è il sistema più l'utente può essere seccato e preferire un'azienda piuttosto che un'altra, può fare fatica a entrare nel servizio se il sistema di autenticazione ha tanti falsi negativi e questo magari incide sui costi del call center, che risolve il problema etc... **Ci deve essere qualcuno che decide che cosa si vuole fare rispetto a questi parametri** che abbiamo appena elencato, **successivamente si può cominciare a fare una valutazione tecnica però a priori c'è una scelta proprio gestionale** dopo interviene una fase di dettaglio della situazione in cui la scelta non è più né puramente gestionale né puramente tecnica perché **il tecnico ci mette le sue competenze sulle soluzioni possibili ma il manager ci mette le sue scelte in termini di costi!!** Quindi deve essere un lavoro coordinato. Poi a livello più basso sarà il tecnico che implementerà le scelte fatte in base alle politiche stabilite e alle procedure definite. **Quando abbiamo parlato di gestione di incidenti abbiamo detto che devono essere definite delle procedure per la gestione della sicurezza**, per esempio se uno vuole dare un account a un collaboratore in università, va dal tecnico e glie lo chiede , il **tecnico non ha diritto di decidere se può avere un account oppure no, ci deve essere una procedura che definisce chi può averlo**, quindi non è una scelta del tecnico ma deve esserci una procedura definita a priori. **Questo implica che devono essere definite a priori le procedure e si devono monitorare nel tempo e modificare in base a esigenze e tecnologie nuove.**

Nel definire questi livelli a partire dal livello più astratto di obiettivi e missioni dell'azienda fino al livello più basso di procedure specifiche di gestione **tutto questo deve essere fatto tenendo sempre conto dell'usabilità dei sistemi** (degli utenti e del personale): non si può imporre alla gente qualcosa di assurdamente restrittivo o complicato perché altrimenti la gente cercherà di aggirarlo. Se si vuole proporre qualcosa di restrittivo bisogna dire perché in modo chiaro e preciso, quindi entra in gioco la formazione, la gente deve sapere perché si deve fare in un certo modo in modo che sappiano i rischi e non cerchino di bypassare le regole/procedure ecc.. In tutto questo quadro vanno definiti anche i compiti di tutte le persone/personale coinvolte/o. Per esempio : se gli aggiornamenti software continuano a uscire vuol dire che ci saranno sempre nuove vulnerabilità, chi li deve fare questi aggiornamenti? Deve esserci un responsabile che li deve fare, se succede qualcosa perché non sono stati fatti gli update è colpa di quel responsabile quindi deve avere l'autorità di fare quello che deve fare, non deve sottostare al tecnico che gli dice che questa settimana non lo può fare o cose del genere. Questa parte di politica di sicurezza la trovate sul libro di Cinotti in biblioteca.

PROXY FIREWALL

Eraamo rimasti a questa situazione.

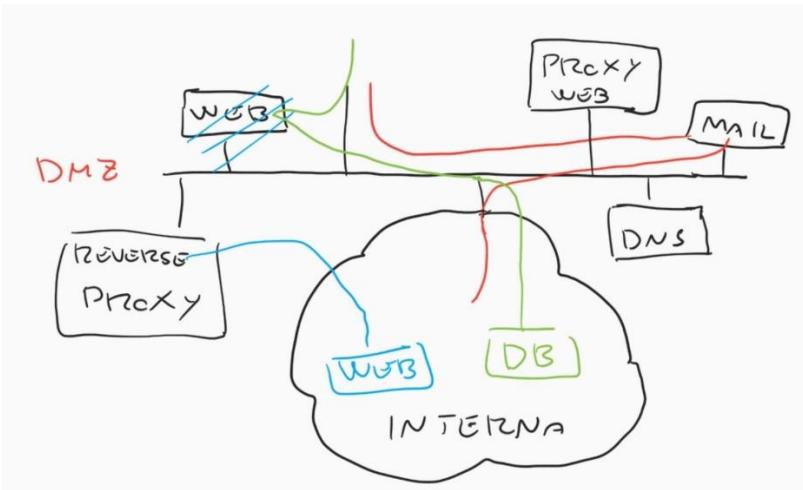


Abbiamo una rete interna, una rete perimetrale interna dove c'è DMZ, un DNS un server WEB una Mail etc... ogni connessione deve passare dalla DMZ. I dati importanti non devono stare nella DMZ: se il server web deve accedere a un DB, tipicamente quest'ultimo è nella rete interna e quando arriva una richiesta al server web, sarà lui che la inoltra al DB e poi quando riceve la risposta la propone all'esterno (in questo modo proteggiamo il DB a un livello maggiore), **l'importante è che il traffico non sia mai diretto dall'esterno verso l'interno. Per mantenere questa regola** che per esempio **per gli utenti interni che devono andare su internet tipicamente si mette un proxy** (generalmente un proxy web) **sulla DMZ in modo che l'utente che vuole navigare su internet si rivolge al proxy e il proxy fa la richiesta verso l'esterno**, inizialmente sono stati introdotti per gestione più efficace della banda, se io carico in cache del proxy certi siti più visitati sarà più efficiente il nostro sistema e ottimizzo lo spreco di banda. **Il proxy sostanzialmente è un dispositivo che fa da client e da server, quindi conosce il protocollo e questo è interessante dal punto di vista della sicurezza perché innanzitutto spezza la connessione**, ma questo lo può fare anche un dispositivo che non conosce il protocollo ad esempio un proxy generico che spezza la connessione TCP. **Ma se conosce il protocollo può controllare che effettivamente il protocollo sia quello**, per esempio che uno non abbia fatto un tunnel e che dentro l'HTTP non stia facendo passare un protocollo diverso, **può richiedere autenticazione, fare controlli sui siti web che vengono visitati ecc.. quindi può diversificare sul tipo di servizio che offre**.

Il più comune è il proxy web server ma si può fare per qualsiasi tipo. **Uno potrebbe pensare di impostare la sicurezza direttamente sui proxy ma il problema è che ci devono essere dei proxy applicativi per ogni applicazione e ci sono applicazioni per cui costruire un proxy ad hoc è difficile.**

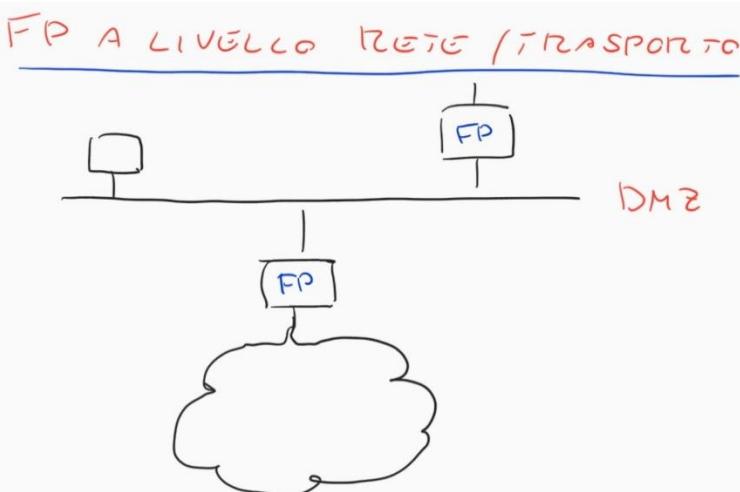
Il proxy è comunque una soluzione quindi.

Inoltre si può utilizzare anche al contrario, **il REVERSE PROXY accetta connessioni dall'esterno e le inoltra verso l'interno, così uno può mettere il server web nella rete interna** (quindi mettendolo più al sicuro) e avere un reverse proxy che fa il lavoro del proxy ma è più esposto perché è sulla DMZ (in realtà anche prima lo era, cambia solo la direzione). Questa rete però l'ha disegnata male perché nella rete interna a sto punto uno dovrebbe avere almeno 2 sottoreti diverse, una per gli utenti e una per il server perché avere l'utente sulla stessa rete dei server è un rischio dato che l'utente è un babbo di minchia.



In tutto questo come si fa a ottenere che il traffico si comporti in questo modo e che viet all'attaccante dall'esterno di contattare direttamente il database, per vietare questo si usano i filtri di pacchetti che molto spesso vengono chiamati firewall per l'importanza che veste e vengono inseriti generalmente uno sulla DMZ e uno dopo la DMZ.

IL FILTRO DI PACCHETTI



Il filtro di pacchetti analizza il flusso di pacchetti e decide quali pacchetti possono passare o meno. I filtri possono lavorare a **vari livelli**, il più comune è a livello **rete/trasporto**, cioè vuol dire che **guarda dentro l'intestazione dei pacchetti a livello rete e a livello trasporto**, raccoglie informazioni su quelle intestazioni e poi decide cosa fare sulla base di quelle informazioni. Si può fare anche un filtro di pacchetti **a livello applicativo** e in questo caso a differenza del **filtro di trasporto riesce a entrare nel merito dell'applicazione ed è quindi più specifico**, ma comporta **2 problemi: deve conoscere le applicazioni ed è più lento quindi ci vuole una macchina più potente** per fare un filtro a livello applicativo.

Mentre un filtro a livello rete/trasporto che è il più classico, è un dispositivo molto efficiente e c'è dappertutto.

Un filtro di pacchetti a livello di trasporto controlla un pacchetto tramite una tabella contenente i seguenti campi:

- la direzione del traffico (inbound = verso l'interno, outbound = verso l'esterno) oppure l'interfaccia di ingresso/uscita.
- l'IP Sorgente da cui provengono i pacchetti.(es.200.123.132.22, any)
- IP Destinazione a cui sono diretti i pacchetti. (come sopra)
- protocollo di trasporto usato (TCP,UDP).
- porte sorgente da cui accettare/rifiutare i pacchetti (es.1024, any).
- porta destinazione a cui sono destinati i pacchetti.
- Che tipo di ACK accettare/rifiutare (0= solo pacchetti che aprono una connessione, 1 = solo pacchetti con connessione già aperta, any=tutti i pacchetti).
- l'azione che deve intraprendere (ACCEPT, FORWARD O DROP).

Come funziona:

(INTERNO) DIREZ	IP SCR	IP DEST	PROT TRASP	P. SCR	P. DEST	ACK	?
INBOUND	ANY	IP DB INTERNO	TCP	ANY	ANY	ANY	DROP

ES: Arriva un pacchetto dall'esterno all'interno INBOUND, supponiamo che l'IP sorgente sia qualsiasi cosa, supponiamo che IP destinazione sia quello di un database interno, protocollo di trasporto TCP, porta sorgente qualunque, porta destinazione qualunque, ACK any, azione: DROP quindi tutti i pacchetti che stanno entrando in quella rete diretti al database vengono buttati via.

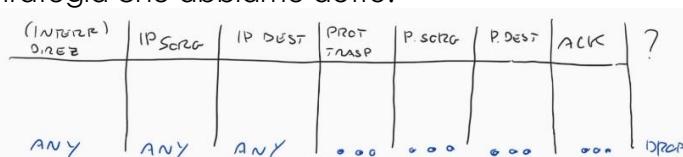
STRATEGIA DI FILTERING

A livello manageriale un filtro di pacchetti lavora tramite una politica che può avere 2 strategie molto diverse:

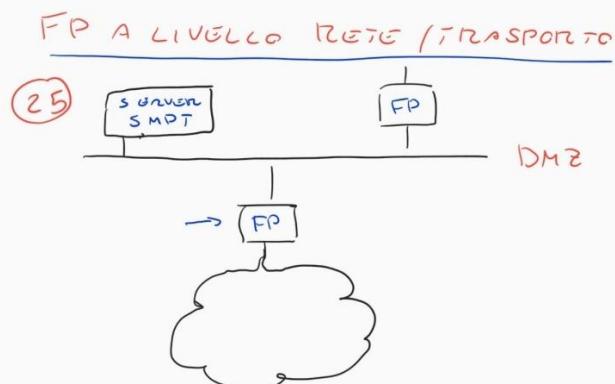
1. **una strategia più restrittiva** che dice che “tutto quello che non è esplicitamente permesso è vietato”. Questo significa che io chiudo la rete e **decido io quali servizi possono entrare e uscire**, è **molto rigida e difficile da far accettare agli utenti**, ma è la strategia che offre più vantaggi per la sicurezza della rete. In questo modo è esplicitamente detto quali sono i servizi che passano.
2. **una strategia permissiva** che dice che “tutto quello che non è esplicitamente vietato è permesso”. Mi segno solo i pacchetti che sono vietati, tutto il resto passa. Questo significa che **un servizio che per esempio non mi serve ma non è particolarmente rischioso lo lascio funzionare**, dal punto di vista degli utenti è meglio ma dal punto di vista della sicurezza è un problema perché non ho la percezione di tutti i servizi che passano ma solo quelli dei servizi che io introduco.

Una banca adotterà una strategia del primo tipo, un'università quella del secondo tipo.

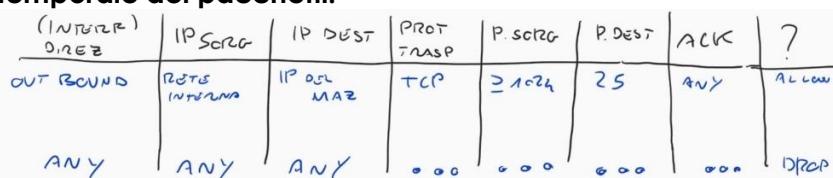
Supponiamo di avere una strategia restrittiva che mi dice, “qualsiasi cosa” → DROP ovvero la prima strategia che abbiamo detto.



Per esempio, se vogliamo permettere agli utenti interni alla rete di mandare posta elettronica, dobbiamo permettere ai pacchetti che vanno a un server SMTP di passare, dove stiamo lavorando? supponiamo di star lavorando sul Filtro di Pacchetti appena sopra la nuvola, e di avere l'SMTP sulla DMZ (porta 25).



Supponiamo di avere sul filtro un pacchetto in uscita che viene da un IP sorgente della rete interna e va verso l'IP del server SMTP(come destinazione) usando TCP, con porta sorgente porta client quindi ≥ 1024 , destinazione porta 25 e l'ACK significa che vogliamo che i pacchetti vengano trattati ognuno per se, vedo passare un pacchetto e decido cosa farne, **Il FP non ha la cognizione della connessione che c'è dietro, Il FP che fa così viene detto statico perché non sa qual è il flusso temporale dei pacchetti.**



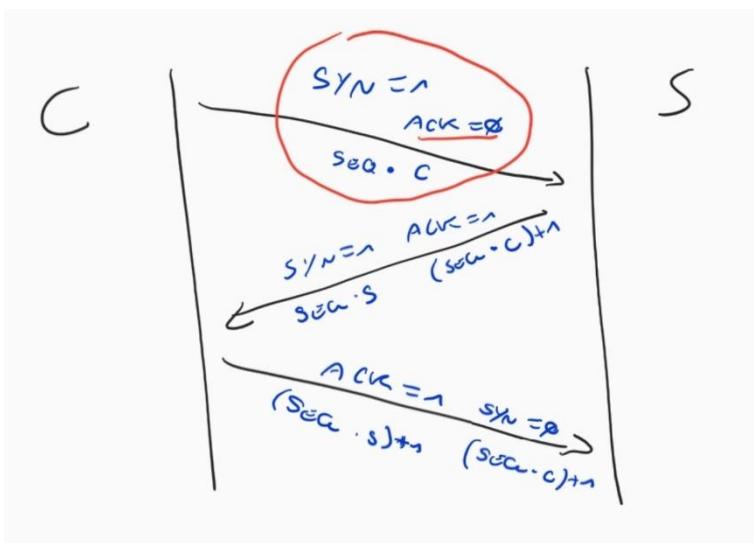
Il problema di questo filtro è che non riconosce una connessione, non possiamo fare qualcosa per impedire all'attaccante di aprire connessioni nel verso sbagliato per esempio.

In questo caso noi vogliamo che l'utente possa aprire una connessione verso il server SMTP non vogliamo per esempio che dal server SMTP compromesso l'attaccante possa aprire una connessione sulla macchina dell'utente sulla porta client.

RIPASSO ACK THREE WAY HANDSHAKE:

ACK=1 in tutti i pacchetti tranne il primo, perché nel primo pacchetto c'è il SYN=1 che dice: "sincronizzati con il numero di sequenza nuovo", nel secondo pacchetto il server dice "ok ora sincronizzati tu con il mio numero di sequenza e mette l'ACK al numero di sequenza ricevuto". terzo messaggio non c'è più il SYN=1 ma c'è solo l'ACK del client da lì in avanti l'ack=1 e SYN=0.

Quello che distingue il pacchetto che apre la connessione dagli altri è che l'ack è a 0 su quel pacchetto, **guardare l'ack significa cercare di capire se quel pacchetto serve per aprire una nuova connessione, l'ack può essere sia a 0 che a 1** in questo caso di esempio perché vogliamo permettere agli utenti di aprire una connessione sul server SMTP. Però non abbiamo permesso all'utente di collegarsi al server perché se non riceve la risposta e in questo caso non la riceve(che è un pacchetto che deve passare dal FP) non verrà fatto passare dalla seconda regola(quella any, any .. drop) quindi **scriveremo un'altra regola e imponiamo che i pacchetti entranti che vengono dal server mail diretti alla rete interna, con TCP, con porta sorgente 25 e porta destinazione >= 1024 e ACK=1, così permettiamo solo pacchetti dove la connessione è già stata stabilita** per l'ack ora torna il discorso fatto prima noi vogliamo che il server mail sulla DMZ compromesso non vogliamo che l'attaccante lì sopra apra una connessione verso la porta client di un utente dove ci può essere installato un cavallo di troia, quindi chiediamo che l'ACK sia sempre a 1 perciò ammettiamo solo pacchetti che fanno parte di una connessione già iniziata.



Nel primo pacchetto c'è syn=1 e ack=0 e vuol dire sincronizzati con il numero di sequenza client.

Il server risponde con syn= 1 perché sta mandando un numero di sequenza server, e ack =1 perché sta dicendo che sta aspettando su sequenza client 1, cioè ho ricevuto il tuo numero di sequenza e il prossimo pacchetto che aspetto è quello con numero di sequenza successivo. Il client risponde con pacchetto con ack a 1, syn a 0 ,numero di sequenza server+1, e anche sequenza client.

Quindi il pacchetto che inizia la connessione è l'unico che ha ACK=0 quindi per evitare che vengano aperte connessioni nella direzione sbagliata viene richiesto che i pacchetti in arrivo dal server abbiano ack=1.

Questo tipo di filtro di pacchetti che non sa nulla sulle connessioni può naturalmente essere preso di mira da attacchi che saturano la banda che manda pacchetti a caso interni.

Se vengono mandati a caso pacchetti che rispettano quelle caratteristiche quelli passano perché non si fanno altri tipi di controllo. Questo è un problema.

Un altro problema è che il filtro lo fa sulla base delle porte, uno intende che sulla porta 25 normalmente c'è il server SMTP e quelle ≥ 1024 sono quelle dei client, però **sulla porta ≥ 1024 potrebbe esserci un server malevolo**, se ho questa possibilità **supponiamo che sia aperto il traffico sulla porta 80 verso l'esterno** e un utente vuole collegarsi con SSH verso la macchina a casa, ma questo è vietato dalla politica dell'azienda è aperta solo la porta 80 in uscita, aprire la porta (cosa vuol dire che è aperta la porta 80 sul firewall? non vuol dire che è aperta la porta 80 sul firewall) **vuol dire che il firewall lascia passare i pacchetti destinati a quella porta, se sul firewall la porta 80 è aperta verso l'esterno** e ammetto che si possano contattare tutti i server web.

Se decido invece di mettere un SSH sulla mia macchina a casa per collegarmi, metto un SSH che ascolta sulla porta 80!! Questo filtro non è in grado di distinguere ciò perché non entra nel merito applicativo guarda solo la porta e se le porte corrispondono il traffico passa.

Quindi è un sistema molto semplice, molto flessibile che non deve conoscere protocolli applicativi ma naturalmente il lato negativo è che non è abbastanza colto. Questo 2° problema **si risolve solo con un protocollo a livello applicativo**, invece l'altro problema cioè **avere pacchetti spuri che possono passare rispettando le regole** (penso che parli dell'attacco a saturazione di banda qui) **si risolve con un filtro di pacchetti dinamico**.

FILTRO DI PACCHETTI DINAMICO (O STATEFULL)



Un Filtro di pacchetti DINAMICO o STATEFUL (con memoria) statefull vuol dire che ha uno stato cioè ha memoria di ciò che è successo: infatti un **FP Statefull è in grado di ricordare che è stata aperta una connessione quindi è in grado di riconoscere i pacchetti che appartengono a una certa connessione**, come fa? Deve mantenere le coppie IP e porte.

Il filtro dinamico è il più semplice da programmare di quello statico (o stateless) perché non deve stare a ricordarsi i pacchetti che entrano o escono dice solo che accetta le connessioni al SMTP server in uscita e lui fa il resto. **Ma un filtro dinamico è meno controllabile e si capisce meno quello che sta facendo** quindi non è sempre una buona idea.

L'altro tipo di filtri che abbiamo visto prima si chiamano statico o stateless, mentre il **filtro dinamico si chiama così perché osserva il traffico e cambia qualcosa nel suo comportamento** se vede aprire una connessione accetta anche i pacchetti di ritorno.

Dato che ha questa memoria **non è più possibile mandare i pacchetti spuri che non appartengono a nessuna connessione perché lui è in grado di distinguerli e bloccarli** (il problema delle porte rimane, per risolverlo serve un FP a lv applicativo).

Programmare un filtro di pacchetti non è semplice: bisogna correttamente **correlare le regole perché vengono controllate in ordine** e quindi **se uno mette prima una regola più comprensiva rispetto a una restrittiva si ferma alla prima e la seconda non verrà mai considerata**, quindi bisogna capire che cosa si fa.

Ci sono 2 regole molto importanti generali da mettere a livello di filtro di pacchetti sono regole che limitano l'IP spoofing (attaccante finge un IP sorgente che non è il suo), due regole semplicissime andrebbero sempre messe:

- **regola di protezione da IPSpoofing esterno**

PROTEZIONE DA IP SPOOFING DALL'ESTERNO

IN BOUND | IP INT. | ANY | ANY | ANY | ANY | DROP

se il pacchetto sta entrando e viene da un IP interno, ovunque sia diretto con qualsiasi protocollo, porta, porta-destinazione, e ACK, non accettarlo. Perché se viene dall'esterno e ha un IP interno come sorgente non è un pacchetto vero, ma uno creato per fare spoofing.

A cosa potrebbe servire? Potrebbe servire per una echo request a tutte le macchine della rete (una rete grossa) con IP sorgente una macchina interna e tutte queste macchine rispondono.

In generale un messaggio broadcast non dovrebbe essere ammesso (fatto entrare) nella rete, perché un attaccante può provocare tante risposte con 1 solo messaggio e quindi accettando una richiesta in broadcast dall'esterno nella mia rete faccio il gioco dell'attaccante quindi blocco anche questo.

Questa è una regola di protezione della rete propria.

PROTEZIONE DA IP SPOOFING DALL'ESTERNO

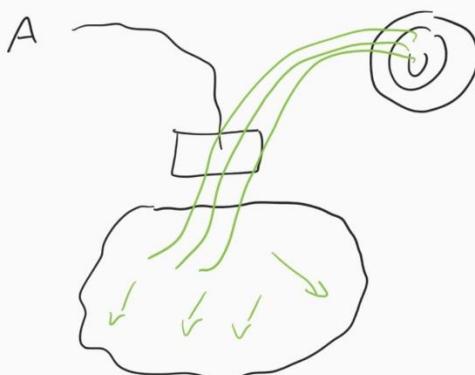
OUT BOUND | IP. EST | —————— DROP

- **regola di protezione da IPSpoofing dall'interno:**

è una regola etica di convivenza civile che dice di **limitare che i miei utenti interni facciano spoofing** (verso l'esterno non all'interno della rete, non limita il fatto che io possa fare spoofing fingendosi di essere un'altra macchina all'interno della mia stessa rete locale), quindi **tutti quelli che escono dalla mia rete ma hanno un IP esterno come IP sorgente, qualsiasi cosa sia il resto, non lo lascio passare**. Se tutte le reti lo facessero l'IPspoofing sarebbe difficile da fare.

Per mettere insieme le due cose dette prima:

Supponiamo che possano entrare messaggi broadcast e si possa fare spoofing dall'esterno.



l'attaccante invia una echo request in broadcast con IP sorgente quello della vittima, tutte la rete risponde alla vittima se lo fa con una certa frequenza la fa collassare.

Per spoofing interno servono dei sistemi di monitoraggio interni.