Crittografia 1
Introduzione alla crittografia

Riccardo Zanotto
CISPA Helmholtz Center
for Information Security

https://cybersecnatlab.it

License & Disclaimer

License Information

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

Argomenti

- Che cos'è e a cosa serve la crittografia
- Cifratura a chiave simmetrica
- Generatori di numeri pseudo-casuali
- Proprietà e sicurezza dei cifrari
- Diffie-Hellman: come scambiare chiavi
- Crittografia a chiave pubblica
- Funzioni di hash

Che cos'è la crittografia

CRITTO – GRAFIA

nascosta scrittura

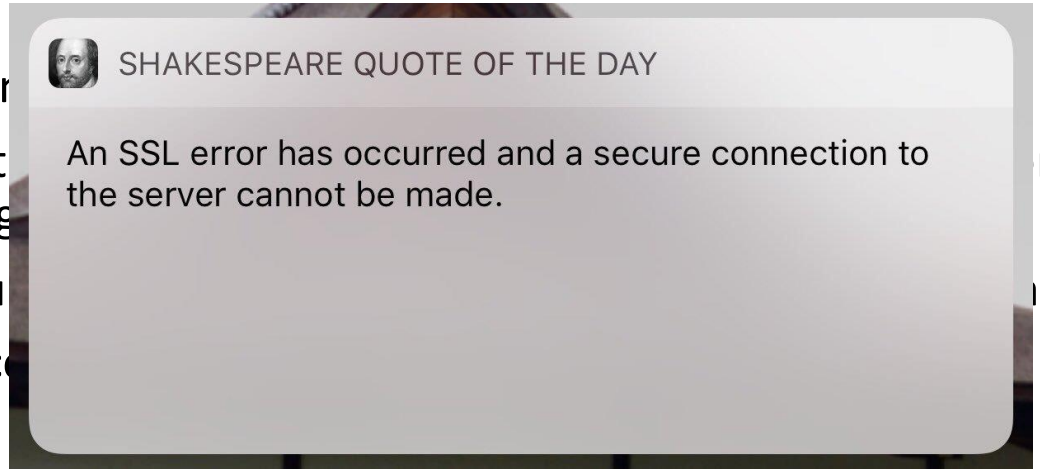
- Nata nell'antichità per spostare comunicazioni strategiche tra generali
- Per molto tempo è stata sinonimo di "cifratura": rendere incomprensibile un messaggio che deve restare segreto
- Ad oggi è una scienza precisa, all'intersezione tra matematica e informatica
- Fino allo scorso secolo era più un'arte, affidata a linguisti.

Che cos'è la crittografia

CRITTO – GRAFIA

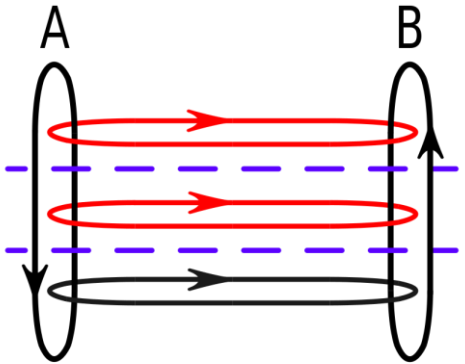
nascosta scrittura

- Nata nell'ar generali
- Per molto t mprensibile un messagg
- Ad oggi è u e informatica
- Fino allo sc



Protocolli sicuri

- Un **protocollo** è un insieme di regole condivise tra più entità per raggiungere uno scopo comune (es. comunicazione via TCP, o via USB)
- "Protocollo per mangiare in mensa" (giorno 1): prendere da mangiare, passare alla cassa, dire di essere di OliCyber, mangiare.
- Il protocollo deve avere uno **scopo** e delle proprietà di **sicurezza**: quali danni può causare qualcuno che interagisce col protocollo senza seguirne le direttive?

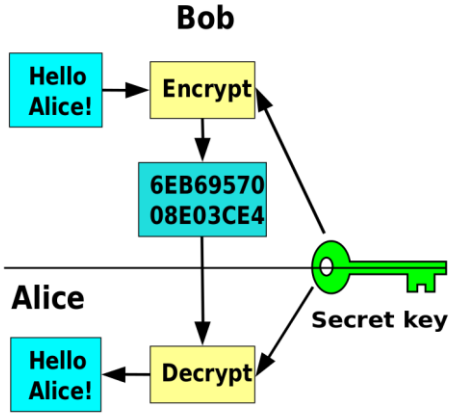


Usi della crittografia

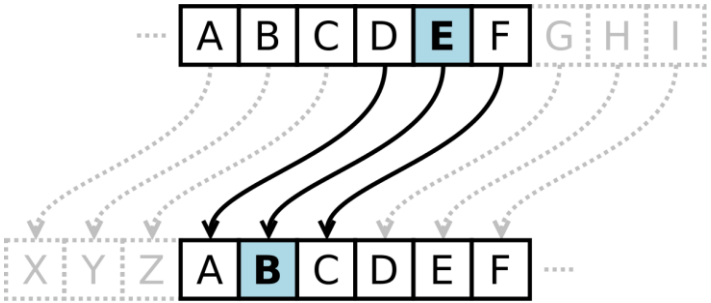
- Oggi la crittografia è diffusa ovunque, principalmente grazie ad Internet; spesso è inclusa nei protocolli stessi di comunicazione
- Telefonia mobile: GSM, GPRS, 4G (SNOW) cifrano i dati tra il dispositivo e la cella telefonica
- HTTPS: rende sicure le comunicazioni con i siti web
- Disk encryption: protegge i dati personali dal furto
- Signal: protegge l'instant messaging
- ~~Blockchain: permette di raggiungere un consenso distribuito~~

Cifratura simmetrica

- La più antica primitiva crittografica: permette di nascondere un messaggio in modo che solo il destinatario voluto possa leggerlo
- Dipende da una **chiave** condivisa k
- Funzione **Encrypt(k,m) = c** prende in input un messaggio m e la chiave k, restituisce un testo cifrato c
- Funzione **Decrypt(k,c) = m'** ha in input un testo cifrato c e la chiave k, produce un messaggio in chiaro m'
- Correttezza: vogliamo che **Decrypt(k, Encrypt(k,m)) = m**



Cifrario di Cesare / ROT-x



- Il più famoso cifrario dell'antichità; fa parte della famiglia "a sostituzione monoalfabetica"
- Cifra lettera per lettera, sostituendo ognuna con quella tre posti più avanti nell'alfabeto; la decifratura torna indietro di tre posti

Principio di Kerckhoffs

- Il cifrario di Cesare è debole: una volta che sai come funziona, puoi leggere tutti i messaggi
- Anche sostituendo lo shift di 3 con uno shift arbitrario tra 1 e 26 è rotto
- Bisogna assumere che il "nemico" riesca a recuperare il macchinario/algoritmo cifrante (es. Enigma): la sicurezza deve stare nella segretezza della **chiave**, non nella segretezza dell'algoritmo
- Ancora più rilevante oggi, con sistemi distribuiti: molto meglio affidarsi a un cifrario ben studiato invece che alla *security through obscurity*, perché l'algoritmo verrà prima o poi reversato

Nascondere la chiave?

- Come definire la sicurezza di un cifrario?
- *Un cifrario è sicuro se è impossibile ricostruire la chiave*
- Encrypt(k, m) = m. È "sicuro"?

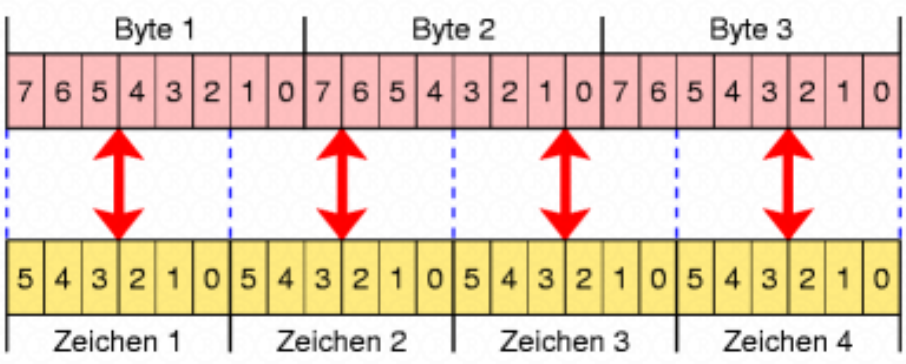
Nascondere la chiave?

- Come definire la sicurezza di un cifrario?
- *Un cifrario è sicuro se è impossibile ricostruire la chiave*
- Encrypt(k, m) = m. È "sicuro"?
- La definizione "vera" è IND-CPA, o sicurezza semantica:
  - "È impossibile distinguere due testi cifrati"
  - "Un testo cifrato non rivela nessuna informazione sul messaggio in chiaro"
  - "I testi cifrati sembrano tutti random"

Cifratura e codifica

- La **codifica** è un modo per rappresentare l'informazione in un modo diverso
- Assomiglia alla cifratura, ma è **deterministica** rispetto al messaggio: Encode(m) = c e Decode(c) = m, senza nessuna chiave
- Esempi: rappresentazione hex dei bytes, base64, rot13, Cesare con shift di 3
- In particolare, la codifica **NON** nasconde il messaggio; la cifratura (se fatta bene) sì

Cifratura e codifica

- La **codifica** è un modo per rappresentare l'informazione in un modo diverso
- Assomiglia al


l messaggio: Encode(m)=c
- Esempi: rapp Cesare con shift di 3
- In particolare a cifratura (se fatta bene) sì

Proprietà di sicurezza

- Alcune proprietà che i protocolli crittografici permettono di ottenere
- Confidenzialità: il dato è visibile solo a chi ne ha il permesso
- Integrità: il dato non può essere modificato
- Autenticazione: verificare l'identità di un utente
- Non-ripudio: ???
- Alcuni schemi di cifratura garantiscono solo confidenzialità, altri confidenzialità, integrità ed eventualmente autenticazione

Proprietà di sicurezza

- Alcune p… ottenere
- Confider…
- Integrità…
- Autentic…
- Non-ripu…
- Alcuni s… altri confider…

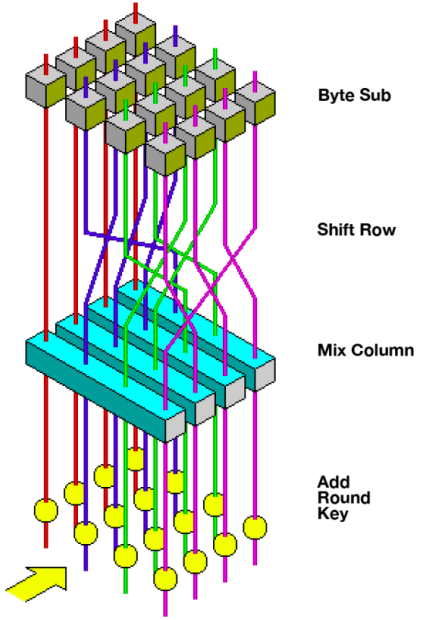
Nonrepudiation

A legal-sounding term invented by standards group members without consultation with lawyers to describe a service that crypto can't provide. X.509 eventually replaced it with "content commitment", but by then the damage had been done.

Cifrario AES

- Advanced Encryption Standard (2001)
- Cifrario "a **blocchi**": m, k, c hanno tutti 16 bytes
- Internamente ha più **round**
- Fissata una chiave, AES_k **permuta** in modo random tutte le stringhe lunghe 16 byte

00000000000000000000000000000000	acbcb0f30020f68a063a53413fd5cfc2
00000000000000000000000000000001	7d7cfd397f3f02a18212fe61fddc0920
….	….
a72b7f9ed71c5b07f7ac313e6fb019dd	fdb97b28be674993176a713242c48d2a
ffffffffffffffffffffffffffffffff	b76716fded8aa910750f5ebd8e021169

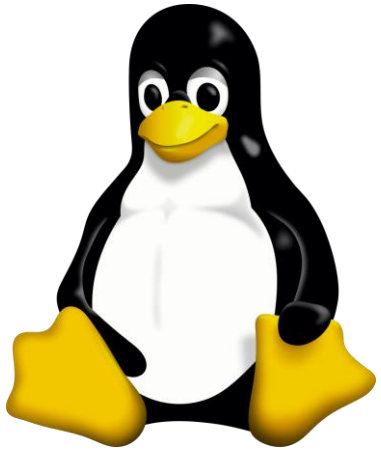


Modes of operation

- AES è un ottimo cifrario per lunghezze fissate (16 byte)
- Per cifrare messaggi arbitrari, bisogna utilizzare più volte la primitiva di cifratura; ci sono diverse **modalità di funzionamento** con cui farlo

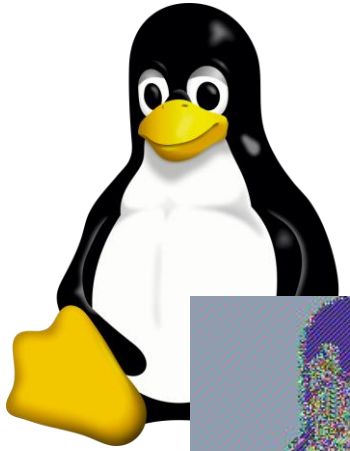
Modes of operation

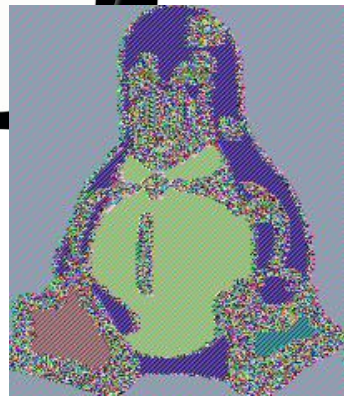
- AES è un ottimo cifrario per lunghezze fissate (16 byte)
- Per cifrare messaggi arbitrari, bisogna utilizzare più volte la primitiva di cifratura; ci sono diverse **modalità di funzionamento** con cui farlo
- **ECB**:
  - Si spezza il messaggio in blocchi $m = m_1 \mid m_2 \mid \dots \mid m_n$
  - Si cifra blocco per blocco $c = AES_k(m_1) \mid AES_k(m_2) \mid \dots \mid AES_k(m_n)$



Modes of operation

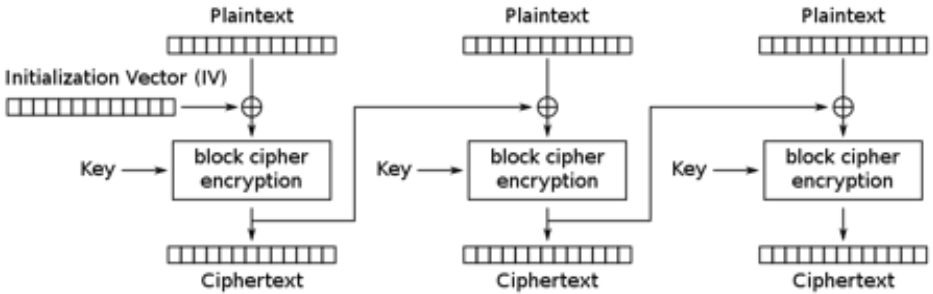
- AES è un ottimo cifrario per lunghezze fissate (16 byte)
- Per cifrare messaggi arbitrari, bisogna utilizzare più volte la primitiva di cifratura; ci sono diverse **modalità di funzionamento** con cui farlo
- **ECB**:
  - Si spezza il messaggio in blocchi $m = m_1 \mid m_2 \mid \dots \mid m_n$
  - Si cifra blocco per blocco $c = AES_k(m_1) \mid AES_k(m_2) \mid \dots \mid AES_k(m_n)$
- Se $m_1 = m_3$, il testo cifrato ha una ripetizione; non sembra più random, cioè è insicuro





Modes of operation

- Si preferisce usare altre modalità, come **CBC**, ma serve un **nonce**, o **IV random**
- Nota: CBC non dà integrità/autenticazione



Cipher Block Chaining (CBC) mode encryption

Bit di sicurezza

- AES è ~~"military-grade encryption"~~
- Dà circa *128* bit di sicurezza*
- Strategia "ottimale" contro un blocco cifrato con AES: occorrono 2^{128*} operazioni (equivalente a bruteforce di tutte le chiavi)
- Esiste una versione di AES con chiavi da 32 bytes, ovvero 256 bit di sicurezza
- Un cifrario è "rotto" quando c'è un attacco *significativamente* più efficiente del bruteforce

Military grade

Cryptography marketed as military grade is often to crypto what military music is to music.

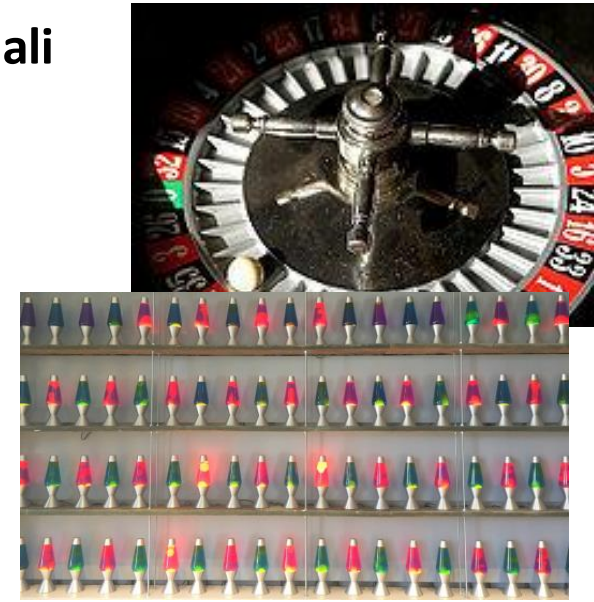
*Attualmente la sicurezza di AES è di circa 126 bit, cioè servono 2^{126} operazioni

Numeri casuali

- Spesso in crittografia occorre generare numeri **casuali**
- È più casuale 00000000 o 01100010?
- Cosa vuol dire random?

Numeri casuali

- Spesso in crittografia occorre generare numeri **casuali**
- È più casuale 00000000 o 01100010?
- Cosa vuol dire random? **Impossibile da predire**
- Come si generano se i computer sono deterministici? Si possono usare misurazioni hardware, come la temperatura o altri processi stocastici microscopici
- Questi metodi sono **lenti** e non sempre disponibili



Cloudflare LavaRand © @mahtin

PRNG

- Uno Pseudo-Random Number Generator è un algoritmo **deterministico** che espande un **seed** "corto" in un lungo **flusso** di bit che **sembrano** casuali
- PRNG(0110) = 00110101010100110010010101110110…
- Un PRNG è sicuro se è **impredicibile**: data la sequenza di bit prodotti finora, deve essere molto difficile calcolare il bit successivo
- Esempi non sicuri: LCG, LFSR, MT19937
- Esempi sicuri: ChaCha20, LegendrePRF

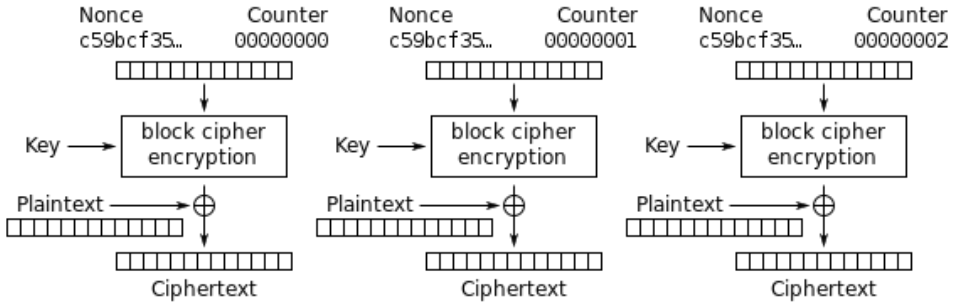
Netscape fail

- Il PRNG implementato da Netscape per SSL usava come seed il timestamp e l'id del processo
- Un PRNG va seedato con alta **entropia**

```
global variable seed;

RNG_CreateContext()
    (seconds, microseconds) = time of day; /* Time elapsed since 1970 */
    pid = process ID;  ppid = parent process ID;
    a = mklcpr(microseconds);
    b = mklcpr(pid + seconds + (ppid << 12));
     seed = MD5(a, b);

mklcpr(x) /* not cryptographically significant; shown for completeness */
    return ((0xDEECE66D * x + 0x2BBB62DC) >> 1);

MD5() /* a very good standard mixing function, source omitted */

RNG_GenerateRandomBytes()
   x = MD5(seed);
   seed = seed + 1;
   return x;
```

AES-CTR

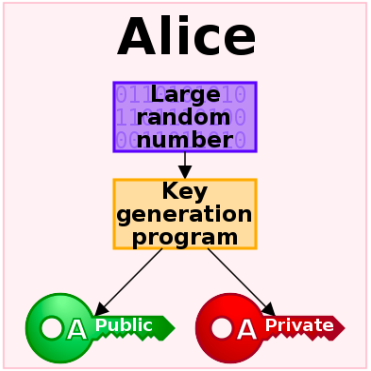
- Possiamo generare numeri pseudo-random usando AES
- Con il "**counter mode**" si fissano una chiave k e un nonce n
- PRNG(k, n) = $AES_k(n)$ | $AES_k(n+1)$ | $AES_k(n+2)$
- AES è sicuro, quindi tutte le cifrature "**sembrano random**"



Counter (CTR) mode encryption

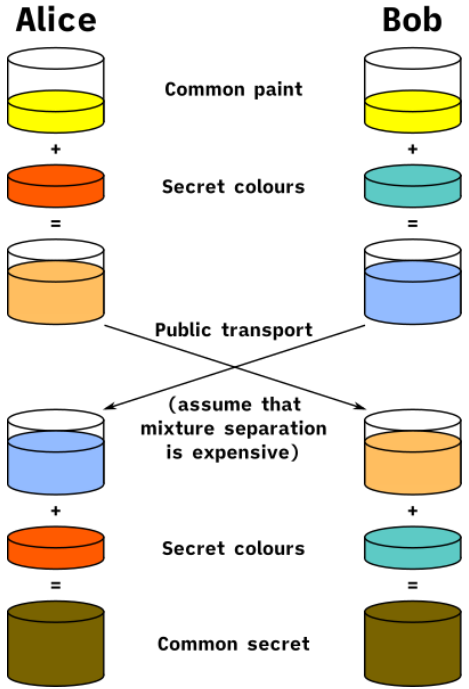
Crittografia a chiave pubblica

- Con una chiave condivisa di 16 byte, AES con la giusta modalità permette di cifrare in modo sicuro TB di dati
- Come si ottiene una chiave **segreta** condivisa se si può parlare solo su canale **pubblico**? (es. Web)
- Le costruzioni "a chiave pubblica" risolvono questo problema
- Ogni utente genera una **coppia** di chiavi
- Usando la chiave **pubblica** di Alice, ci sono operazioni che solo Alice potrà invertire (con la sua chiave **privata**)

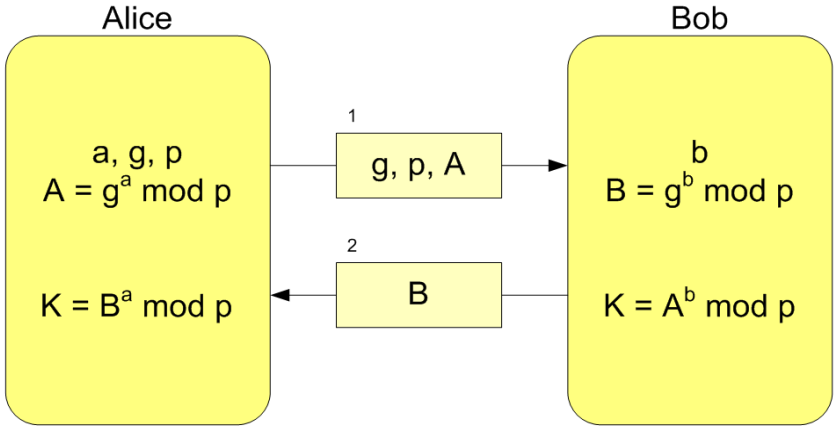


Scambio di chiavi: Diffie-Hellman

- La crittografia a chiave pubblica è nata con la soluzione per lo scambio di chiavi proposto da Diffie e Hellman
- Parametri comuni
- Entrambi i partecipanti generano un segreto casuale
- Ottengono un valore pubblico dal proprio segreto
- Usando il dato pubblico dell'altro e il proprio segreto, generano un valore condiviso



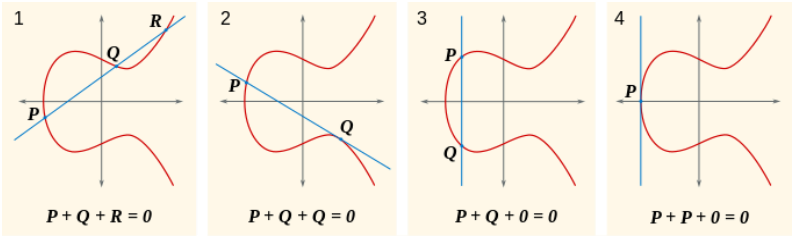
Esempi di DH



$K = A^b \bmod p = (g^a \bmod p)^b \bmod p = g^{ab} \bmod p = (g^b \bmod p)^a \bmod p = B^a \bmod p$

DH classico

ECDH



Perché DH è sicuro

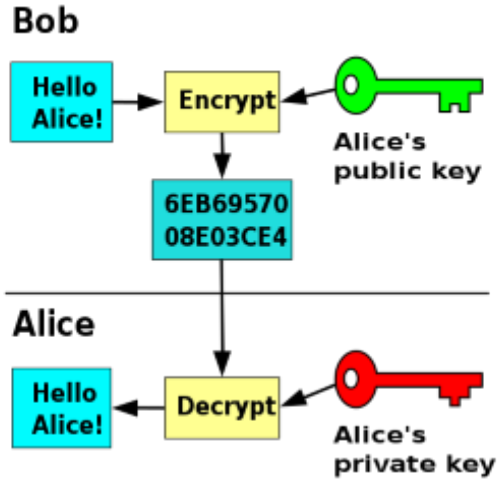
- La sicurezza del protocollo Diffie-Hellman deriva dal non saper ricavare il valore segreto da quello pubblico
- Secret -> public key è **facile**
- Public key -> secret è **difficile**
- Nel caso standard g^x, il calcolo di x è detto *problema del logaritmo discreto*, che in generale è difficile
- Es: se p ha 3072 bit, calcolare il dlog costa circa 2^{128} operazioni

Attacco man-in-the-middle

- Lo scambio di chiavi Diffie-Hellman può essere facilmente attaccato da un avversario con accesso alla rete (man-in-the-middle)

Cifratura a chiave pubblica

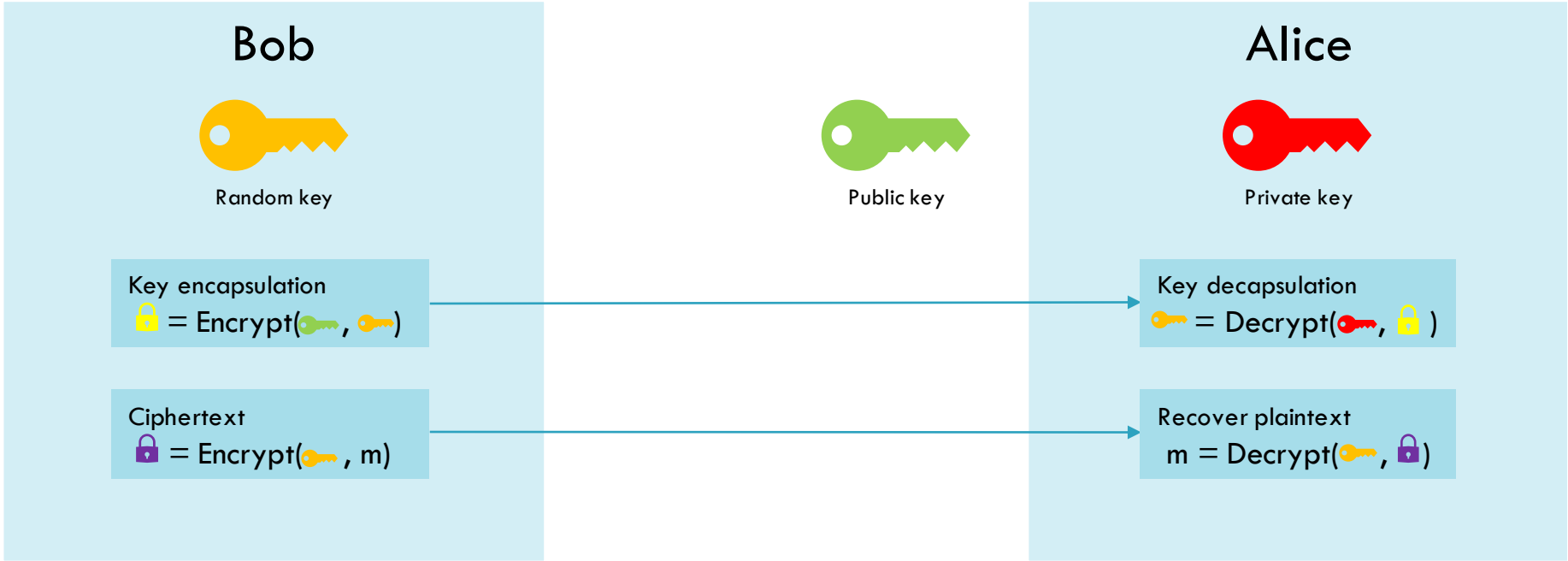
- Permette di cifrare messaggi usando chiavi pubbliche
- Funzione **KeyGen() = (sk, pk)** genera la coppia di chiavi privata/pubblica
- Funzione **Encrypt(pk, m) = c** usa la chiave **pubblica** per cifrare un messaggio verso quel destinatario
- Funzione **Decrypt(sk, c) = m** usa la chiave **privata** di chi riceve il testo cifrato per recuperare il messaggio



RSA

- Il primo cifrario a chiave pubblica: Rivest, Shamir, Adleman 1977
- La sua sicurezza si basa sul problema della **fattorizzazione** di numeri interi, per cui non si hanno algoritmi efficienti
- Chiavi da 2048 o 4096 bit
- Chiave pubblica: pk = (e, N); chiave privata: sk = d
- Encrypt(pk, m) = m^e % N
- Decrypt(sk, c) = c^d % N
- Nota: per far funzionare il tutto, sk e pk devono essere correlate *in un qualche modo*

Cifratura ibrida

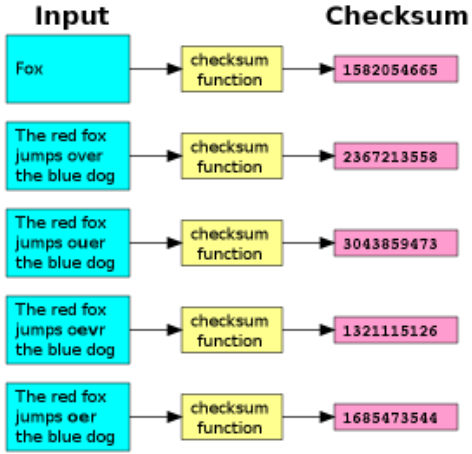


Quindi abbiamo risolto tutto?

- Possiamo "buttare" la crittografia simmetrica? **No**
  - La crittografia asimmetrica è in generale molto **lenta**
  - Si basa su problemi matematici che ci "obbligano" a cifrare **pochi dati** per volta
  - Non risolve (ancora) i problemi di integrità e autenticazione

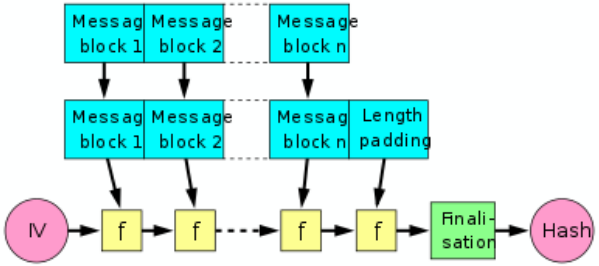
Funzioni di hash

- Una funzione di hash trasforma un input **arbitrario** in una stringa di lunghezza **fissata**
- Può essere utilizzata come "**fingerprint**" di un file
- Esempio: quando si scarica un file grosso, c'è spesso anche md5sum o sha256sum. Serve per controllare l'**integrità** del file (contro errori accidentali)
- Può essere usata per "nascondere" un dato (es. password hashing), ma è un utilizzo improprio

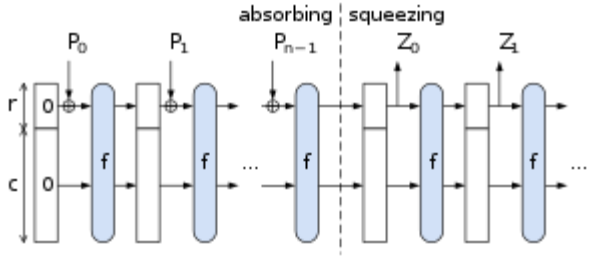


Esempi di hash

- L'hash più diffuso al momento è SHA256, che ha una costruzione di tipo Merkle-Damgard
- L'hash più moderno è SHA-3, o Keccak, e ha una struttura "a spugna"



Merkle-Damgard



Sponge

Proprietà di sicurezza

- L'output della funzione di hash H è detto **digest**
- **Pre-image resistance**: dato un digest H(m), è impossibile ricostruire m
- **Second pre-image resistance**: dati un messaggio m e il suo digest H(m), è impossibile costruire un altro messaggio m' con H(m')=H(m)
- **Collision resistance**: è impossibile costruire due messaggi diversi m_1, m_2 che abbiano lo stesso digest $H(m_1)=H(m_2)$

Osservazioni sulla sicurezza

- Ogni funzione di hash ha delle collisioni. Quello che si richiede è che sia difficile **calcolarle esplicitamente**
- Se sappiamo a priori che un digest è della forma H(1), H(2), …, H(1000000) possiamo trovare una pre-immagine semplicemente calcolando tutti i digest
- Per avere 128 bit di sicurezza il digest deve essere lungo almeno 256 bit (birthday bound)

Password e hash

- Si può usare la pre-image resistance per salvare su un server una password come `H(psw)`. È **vulnerabile** ad attacchi bruteforce/dizionario
- Il problema qui è **concettuale**: una funzione di hash è pensata per essere calcolata velocemente, quindi gli attacchi sono **efficienti**
- Esistono le **PBKDF** a questo scopo: servono più risorse per calcolarle
- Es: bcrypt, scrypt, Argon2
- Paradigma alternativo per autenticarsi: non inviare la password al server, ma eseguire un protocollo interattivo per convincere il server di conoscere la password (**PAKE**)

Autenticare con gli hash?

- Se vogliamo "certificare" un messaggio m, per esempio un cookie, potremmo concatenare m al digest H(m)
- Esempio: per autenticare Alice, salviamo il cookie `Alice|H("Alice")`
- Problema: Alice può impersonare Bob, perché può calcolare il cookie `Bob|H("Bob")`
- Serve una chiave segreta

Message Authentication Codes

- Un MAC è una primitiva a chiave simmetrica con due algoritmi
- Funzione **MAC(k, m) = t** genera un tag a partire da un messaggio
- Funzione **Verify(k, m, t) = 0/1** verifica se un tag è valido per m

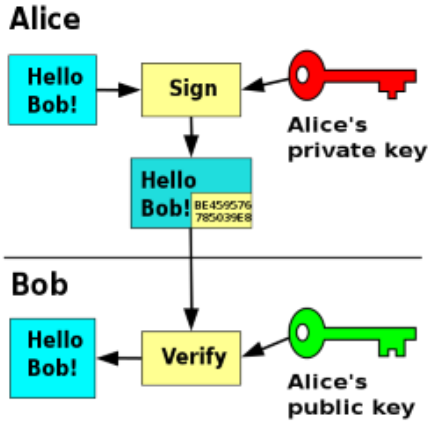
Message Authentication Codes

- Un MAC è una primitiva a chiave simmetrica con due algoritmi
- Funzione **MAC(k, m) = t** genera un tag a partire da un messaggio
- Funzione **Verify(k, m, t) = 0/1** verifica se un tag è valido per m

- Ora possiamo autenticare i cookies con m|MAC(k, m)
- I JWT hanno questa struttura
- Possiamo costruire un MAC usando una funzione di hash (**HMAC**)

Firme digitali

- Una firma digitale permette di verificare che uno specifico messaggio è stato "validato" da una specifica persona
- Costruzione a chiave pubblica; l'algoritmo di **firma** usa la chiave **privata**, l'algoritmo di **verifica** la chiave **pubblica**
- Se la firma verifica, siamo certi del contenuto del messaggio (**integrità**) e del mittente (**autenticazione**)
- Esempi: RSA, DSA, ECDSA, EdDSA



Hash vs MAC vs firme

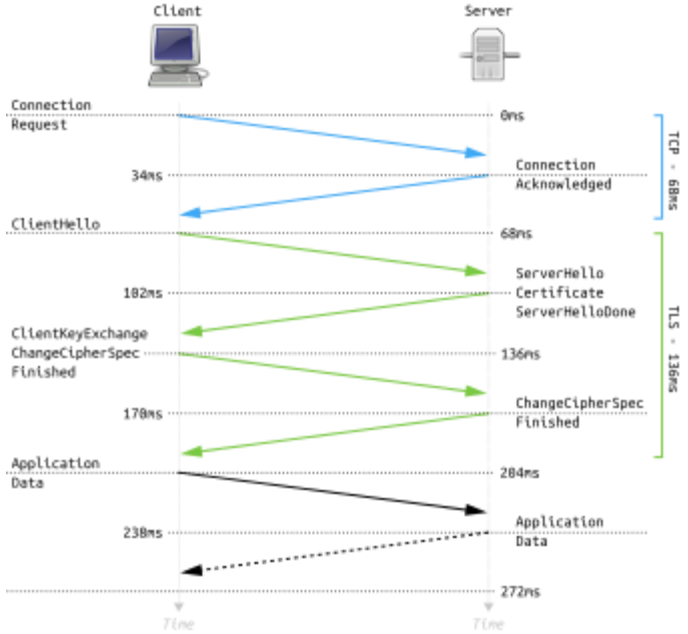
	Hash	MAC	Firma digitale
Integrità	✔	✔	✔
Autenticazione	✘	✔	✔
"Non ripudio"	✘	✘	"✔"
Tipo di chiavi	Nessuna	Simmetriche	Asimmetriche

Il problema del trust

- Le firme digitali possono risolvere l'attacco MITM contro un key exchange: Alice può firmare la propria chiave pubblica
- Come essere sicuri che la firma arrivi proprio da Alice? In generale **non possiamo.**
- Con le firme digitali possiamo solo **spostare** il problema del trust
- Per far funzionare HTTPS ci sono grandi aziende (certification authorities) che firmano i certificati web, e tutti i dispositivi trustano di default queste grandi aziende (**PKI**)

HTTPS

- Il server invia il **certificato**:
  - Hostname
  - Chiave pubblica
  - Firma di una CA
- Il client controlla la firma, ed esegue un key exchange (effimero) con la chiave pubblica del server
- Client e server cifrano in maniera simmetrica tutto il traffico
- TLS_AES_128_GCM_SHA256 (+ECDHE)



Dove fallisce la crittografia

"Cryptography is usually bypassed. I am not aware of any major world-class security system employing cryptography in which the hackers penetrated the system by actually going through the cryptanalysis […] usually there are much simpler ways of penetrating the security system"

- Shamir

Dove fallisce la crittografia

- Threat modeling: cosa un attaccante può e non può fare?
- Scelta del protocollo: trovare un protocollo esistente che protegge contro il threat model scelto
- Composizione dei protocolli: possiamo unire più protocolli per avere le proprietà di sicurezza di tutti... o no? (CRIME, KRACK)

Dove fallisce la crittografia

- Scelta della libreria: occorre trovare una libreria che implementi il protocollo cercato. Può essere implementato male, o peggio non esistere
- Abuso della libreria: una buona libreria dovrebbe astrarre tutto, e non permettere di riutilizzare nonces, ad esempio
- Interfaccia tra crittografia e utenti: bisogna mostrare in modo comprensibile se qualcosa non è verificato, senza però leakare informazioni (es. Bleichenbacher)

Conclusione

- Si definiscono alcune proprietà di sicurezza
- Si costruiscono/compongono protocolli che le forniscano
- Sia le definizioni che i protocolli sono molto precisi: comporre parti "sicure" non è detto che funzioni, se le proprietà di sicurezza sono incompatibili
- La crittografia a chiave pubblica si basa su "problemi difficili"; a volte ci sono istanze risolvibili; ogni tanto si scoprono algoritmi efficienti per problemi che si pensavano difficili

Materiale aggiuntivo

- https://cryptohack.org/
- *Real-World Cryptography*, D. Wong
- *Serious Cryptography*, J.P. Aumasson
- *Introduction to Modern Cryptography*, J. Katz e Y. Lindell

Crittografia 1
Introduzione alla crittografia

Riccardo Zanotto
CISPA Helmholtz Center
for Information Security

https://cybersecnatlab.it