Crittografia

Federico Matteoni

A.A. 2020/21

Indice

Introduzione

Prof.ssa: Anna Bernasconi.
Vedremo i cifrari da un punto di vista prettamente algoritmico. Vedremo anche i cifrari storici, ormai non più utilizzabili, perché hanno "aperto la strada", per poi passare ai cifrari perfetti (soluzione ideale ma con costo elevato).
Poi esamineremo i cifrari simmetrici, a chiave pubblica, curve ellittiche, firma digitale, SSL. Protocolli zero knowledge, blockchain e crittografia quantistica.

Libro di testo: Bernasconi, Ferragina, Luccio - Elementi di Crittografia.

Esame Orali nel caso di esami a distanza, scritto nel caso di esami in presenza, closed-book.

Capitolo 1

Introduzione alla Crittografia

1.1 Introduzione

Crittografia Significa "*scrittura nascosta*", si intendono tecniche matematiche per mascherare i messaggi per non renderli leggibili a terzi (**crittografia**) o tentare di svelarli quando non si è il legittimo destinatario (**crittoanalisi**). Quindi tecniche di protezione e viceversa.
Esiste per i due mondi in contrapposizione: persone che vogliono scambiarsi privatamente informazioni e gli *impiccioni* che desiderano ascoltare o intromettersi nelle conversazioni altrui (per curiosità, investigazione o altri scopi).

Due gruppi di persone Chi vuole proteggersi userà **metodi di cifratura**, gli altri useranno **metodi di crittoanalisi**

Crittografia Metodi di Cifratura

Crittoanalisi Metodi di . . . **crittologia** studio comunicazione canali non sicuri e relativi problemi

1.1.1 Lo scenario

Alice vuole comunicare con Bob su un canale insicuro, quindi adottano un metodo di cifratura per spedire il messaggio in chiaro `m` sottoforma di crittogramma `c` (testo cifrato) che deve essere: incomprensibile al crittoanalista Eve (eavesdropper) in ascolto sul canale, ma facilmente decifrabile da Bob.

MSG Insieme dei messaggi in chiaro

CRITTO Insieme dei crittogrammi

$$C : \text{MSG} \to \text{CRITTO}$$
$$D : \text{CRITTO} \to \text{MSG}$$

Sono operazioni da poter fare in tempo polinomiale. C e D sono una l'inversa dell'altra, ma C **deve essere iniettiva**.

1.1.2 Antichi esempi

Erodoto Nelle *Storie*, V secolo a.C.
Messaggi tatuati sulla testa, coperti dai capelli e riscoperti rasando la testa.

Scitale Spartani. Asta cilindrica in due esemplari identici. Si avvolgeva una striscia di carta attorno al cilindro e scritta. La chiave del cifrario è il diametro dello scitale.

Enea Tattico Un libro qualsiasi con un insieme di lettere segnate, o sostituire le vocali con simboli grafici.

Cifrario di Cesare Il più antico cifrario di concezione moderna. L'idea di base è che il crittogramma è ottenuto dal messaggio in chiaro `m` sostituendo ogni lettera con quella di tre posizioni più avanti nell'alfabeto.
Es. A $\to$ D, Z $\to$ C. La segretezza dipende interamente dalla conoscenza del metodo, era destinato all'uso ristretto da un piccolo gruppo di persone.

1.2 Livello di segretezza

Classificazione in base al livello di segretezza

> Cifrari per **uso ristretto**
> Le tecniche con cui si calcola e decifra il crittogramma sono tenute segrete in ogni loro aspetto. Impiegati per comunicazioni classificate (diplomatiche o militari), non adatti per uso di massa.
>
> Cifrari per **uso generale**
> Ogni codice segreto non può essere mantenuto tale per troppo a lungo. La parte segreta si limita alla chiave, nota solamente agli utenti che stanno comunicando.
> Vengono studiati dalla comunità, coinvolgendo tantissime persone. Solo la chiave deve essere segreta.
> **Il nemico conosce il sistema**.
> Quindi C e D sono note, la chiave **segreta** `k` è usata come input sia in C che in D:
> `c = C(m, k)`, `m = D(c, k)`
> Se non si conosce `k`, anche conoscendo C e D non si possono estrarre informazioni dal crittogramma.
> Tenere segreta una sola chiave è più facile che segretare l'intero metodo. Tutti possono usare C e D pubbliche con chiavi diverse, e se un crittoanalista entra in possesso di una chiave posso generarne semplicemente una nuova.

1.2.1 Chiavi segreta

Se la segretezza dipende unicamente dalla chiave bisogna proteggersi dagli attacchi a forza bruta, quindi avere un gran numero di chiavi, così da essere immuni da chi le prova tutte.
Inoltre la chiave deve essere scelta in modo casuale e non prevedibile, sennò il crittoanalista può provare le chiavi ovvie.

Attacco esauriente Il crittoanalisa potrebbe sferrare un attacco a forza bruta verificando la significatività delle sequenze `D(c, k)` $\forall$ `k`.
Se $|\text{Key}| = 10^{20}$ e con un calcolatore che impiega 10^{-6} per calcolare `D(c, k)` servirebbe in media più di un milione di anni per scoprire il messaggi provando tutte le chiavi. Però la segretezza può essere violata con altre tecniche: esistono cifrari più sicuri di altri pur con uno spazio di chiavi più piccoli.
Un cifrario complicato non è necessariamente più sicuro e **mai sottovalutare la bravura del crittoanalista**.

1.2.2 Crittoanalista

Comportamento Il comportamento di un crittoanalista può essere:

Passivo, quando si **limita ad ascoltare** la comunicazione

Attivo, quando **agisce sul canale** disturbando la comunicazione o modificando il contenuto dei messaggi.

Attacchi a un sistema crittografico Hanno l'obiettivo di forzare un sistema. Il metodo e il livello di pericolosità dipendono dalle informazioni in possesso del crittoanalista:

Cipher Text Attack: conosce una serie di crittogrammi

Known Plain-Text Attack: conosce una serie di coppie `(m, c)`

Chosen Plain-Text Attack: si procura coppie `(m, c)` relative a messaggi in chiaro da lui scelti.
Tutta la crittografia a chiave pubblica è soggetta a questo tipo di attacco (avendo la chiave pubblica, cifro dei messaggi che penso possano passare e ascolto finché non trovo nella comunicazione i crittogrammi in mio possesso).

Man in the Middle Il crittoanalista si installa sul canale di comunicazione:

Interrompe le comunicazioni dirette tra gli utenti Alice e Bob

le **sostituisce con messaggi propri**

e **convince** ciascun utente **che tali messaggi provengano leggitimamente dall'altro** utente.

Quindi il crittoanalista **Eve si finge Bob agli occhi di Alice e Alice agli occhi di Bob**.

Esiti

Successo pieno, si scopre completamente D o si ottiene la chiave

Successo limitato, si scopre solo qualche informazione ma sufficiente per comprendere il messaggio

1.2.3 Situazione attuale

Cifrari perfetti Inattaccabili, esistono ma richiedono operazioni complesse, **chiavi lunghe tanto quanto il messaggio e mai riutilizzabili**.
Shannon, 1945 (pubblicato nel 1949 per motivi di segretezza militare): `m` e `c` appaiono totalmente scorrelati, come se `c` fosse una stringa casuale di bit.
Nessuna informazione può filtrare dal crittogramma. Vedremo la teoria matematica.

One-Time Pad Anche detto blocco monouso, sicuro ma per essere usato bene richiede chiavi segrete totalmente casuali e lunghe quanto il messaggio. Come generarla e come scambiarla?

Cifrari attuali Nella crittografia di massa non si usano cifrari perfetti, ma **cifrari *dichiarati* sicuri**, inviolati dagli esperti e che usano algoritmi solo esponenziali per decrittare senza chiave. Il tempo per violare un cifrario è enorme e rende l'operazione insostenibile → impossibilità *pratica* di forzare il cifrario.

Dichiarati sicuri Non è noto se questi problemi matematici richiedano algoritmi *necessariamente* esponenziali o se sono dovuti all'incapacità nostra di trovare metodi più efficienti. Si riconduce a $P = NP$

1.2.4 Cifrari odierni

Advanced Encryption Standard AES, simmetrico a blocchi con chiavi di 128-256bit, pubblicamente noto e realizzabile su computer di ogni tipo. Il messaggio è diviso a blocchi lunghi quanto la chiave.

Le chiavi Sono stabilite dai mezzi elettronici (PC, smartphone, terminale...) e su Internet si scambia una chiave per ogni sessione.

Scambio delle chiavi La chiave va comunicata in sicurezza su un canale non ancora sicuro. Un'intercettazione nello scambio della chiave compromette il sistema.
Nel 1976 viene proposto un algoritmo per generare e scambiare una chiave segreta su un canale insicuro, senza necessità di scambiare informazioni o di incontrarsi in precedenza.
Si chiama **protocollo DH**, ancora largamente utilizzato nei protocolli crittografici su Internet.
Si scambiano pezzi di chiave tramite la rete e unendole a informazioni locali si costruisce la chiave.

Chiave pubblica Diffie ed Hellman hanno anche proposto la crittografia a chiave pubblica.

Cifrari simmetrici: stessa chiave per cifrare e decifrare, nota solo ai due utenti che comunicano. La scelgono di comune accordo e la tengono segreta.

Cifrari asimmetrici: chiavi pubbliche usate per cifrare e chiavi private per decifrare.
`c = C(m, k`$_{pub}$`)`
`m = D(m, k`$_{prv}$`)`
Si rende necessario che la C sia una one-way trapdoor: calcolare il crittogramma deve essere facile (polinomiale), ma decifrare `c` deve essere computazionalmente difficile (a meno di conoscere la trapdoor, la chiave privata).

RSA Rivest, Shamir, Adleman, 1977. Propongono un sistema a chiave pubblica facile da calcolare e difficile da invertire.

Vantaggi

Comunicazione molti a uno
Tutti possono inviare in modo sicuro allo stesso destinatario usando la sua chiave pubblica, ma solo lui può decifrarli. Un crittoanalista non può decifrare anche se conosce C, D e k_{pub}

Se n utenti vogliono comunicare servono solo $2n$ chiavi invece delle $n(n-1)/2$ necessarie nei cifrari simmetrici (una coppia per ogni coppia di utenti)

Non è richiesto nessun scambio

Svantaggi

Sono molto lenti rispetto ai cifrari simmetrici (polinomi di terzo grado)

Sono esposti ad attacchi di tipo chosen plain-text, perché conosco la chiave pubblica
Scelgo un numero qualsiasi di messaggi in chiaro, costruisce i crittogrammi relativi e ascolta sul canale confrontando i crittogrammi in transito e se trova un riscontro sa esattamente qual è il messaggio passato.

Come si usa Oggi si usa un cifrario a chiave segreta (AES) per le comunicazioni di massa, e un cifrario a chiave pubblica per scambiare le chiavi segrete relative al primo senza incontri fisici tra gli utenti.
Diventa lento solo lo scambio delle chiavi. Siamo anche al sicuro da attacchi chosen plain-text perché se la chiave è scelta bene risulta imprevedibile dal crittoanalista.

1.3 Rappresentazione matematica di oggetti

Per rappresentare gli oggetti scegliamo dei **caratteri** da un **insieme finito** detto **alfabeto**.
Un **oggetto** è **rappresentato da una sequenza ordinata di caratteri dell'alfabeto**. L'ordine dei caratteri è importante: a **oggetti diversi corrispondono sequenze diverse** e **il numero di oggetti che si possono rappresentare non ha limiti**. Significa che fissando un numero n arbitrariamente grande possiamo sempre creare un numero di oggetti $> n$, con sequenze via via più grande.

Alfabeto Γ con $|\Gamma| = s$ e N oggetti da rappresentare.
$d(s, N)$: lunghezza della sequenza più lunga che rappresenta un oggetto dell'insieme. A noi interessa la rappresentazione che minimizza $d(s, N)$, cioè $d_{min}(s, N)$
Una rappresentazione è tanto più efficiente quanto $d(s, N)$ si avvicina a $d_{min}(s, N)$

Esempio $s = 1, \Gamma = \{0\}$ l'unica possibilità è variare la lunghezza $\Rightarrow d_{min}(1, N) = N$, estremamente sfavorevole.
$s = 2, \Gamma = \{0, 1\}$, $\forall\, k \geq 1$ ho 2^k sequenze di lunghezza k. Il numero totale di sequenze lunghe da 1 a k è $2^{k+1} - 2$ (si esclude anche la sequenza nulla). Con N oggetti da rappresentare $\Rightarrow k \geq \log_2(N + 2) - 1 \Rightarrow N$ sequenze diverse tutte di $\log_2(N)$ caratteri.

Efficiente Codifica efficiente quando c'è questa riduzione logaritmica, **efficiente** quando . Sequenze della stessa lunghezza è vantaggioso perché non servono caratteri separatori. Per questo è necessario che l'alfabeto contenga almeno due caratteri.
La **notazione posizionale** è una rappresentazione efficiente indipendentemente dalla base $s \geq 2$ scelta. Un intero N è rappresentato con un numero d di cifre $\mid \log_s(N) \leq d \leq \log_s(N) + 1$

1.4 Richiamo della teoria della calcolabilità

Problemi computazionali Formulati matematicamente di cui cerchiamo una soluzione algoritmica: **decidibili** (e **trattabili** o **non trattabili**), o **non decidibili**.
Calcolabilità $\rightarrow$ **Algoritmo** e **problema non decidibile**
Complessità $\rightarrow$ **Algoritmo efficiente** e **problema intrattabile**.

Numerabilità Due insiemi A e B hanno lo stesso numero di elementi $\Leftrightarrow$ si può stabilire una **corrispondenza biunivoca** tra i loro elementi.
Questo porta alla definizione di **numerabile**: un insieme è numerabile $\Leftrightarrow$ i suoi elementi possono essere messi in **corrispondenza biunivoca con i numeri naturali**.
Numerabile significa che **possiede un'infinità numerabile di elementi**. Esempi: l'insieme dei numeri naturali N, l'insieme degli interi Z (avendo n in corrispondenza biunivoca con $2n + 1$ per $n \geq 0$ e $n \leftrightarrow 2|n|$ per $n < 0$, dando la sequenza $0, -1, 1, -2, 2 \ldots$) o anche l'insieme dei naturali pari ($2n \leftrightarrow n$)

Enumerazione delle sequenze Si vuole elencare in uno ordine ragionevole le sequenze di lunghezza finita costruite su un alfabeto finito. Le sequenze non sono in numero finito, quindi non si potrà completare l'elenco.
Lo scopo è **raggiungere qualsiasi sequenza σ arbitrariamente scelta in un numero finito di passi**. σ deve dunque trovarsi a **distanza finita** dall'inizio dell'elenco. Non va bene l'ordine del dizionario perché non saprei la posizione della prima stringa che inizia con b perché le stringhe composte da tutte a sono infinite.
Si stabilisce un ordine tra i caratteri. Si ordinano prima in lunghezza crescente e, a pari lunghezza, in ordine alfabetico.

Esempio $\Gamma = \{a, b, \ldots, z\}$, avrei
$a, b, \ldots, z$,
$aa, ab, \ldots, az, ba, bb, \ldots, bz, \ldots, zz, \ldots$
Ad una sequenza arbitraria corrisponde un numero intero, e la sequenza s arbitraria si troverà tra quelle di lunghezza $|s|$ in posizione alfabetica. Quindi ad una sequenza arbitraria $\leftrightarrow n$ che indica la posizione nell'elenco, e ad un numero naturale $n \leftrightarrow$ la sequenza che occupa l'n-esima posizione nell'elenco.

La **numerazione delle sequenze è fattibile perché sono di lunghezza finita**, anche se illimitata. Cioè per qualunque intero d scelto a priori, esistono sequenze di lunghezza maggiore di d. Per sequenze di lunghezza infinita la numerazione non è possibile

Insiemi non numerabili Insiemi non equivalenti a N come R, $(0, 1)$, l'insieme di tutte le linee del piano, insieme delle funzioni in una o più variabili... $\Rightarrow$ l'**insieme dei problemi computazionali non è numerabile**. Perché un problema computazionale è sempre visualizzabile come una funziona matematica, che associa ad ogni insieme di dati espressi da k numeri interi il corrispondente risultato espresso da j numeri interi

$$f : N^k \rightarrow N^j$$

Quindi l'insieme di queste f **non è numerabile**.

Diagonalizzazione $F = \{$ funzioni $f \mid f : N \rightarrow \{0, 1\}\}$, ogni $f \in F$ è rappresentata da una sequenza infinita

x 0 1 2 3 ... n ...

$f(x)$ 0 1 0 1 ... 0 ...

ma se è possibile è rappresentabile con una regola (f 0 se x pari 1 se x dispari)
Per assurdo, ipotizzo F numerabile. Si può assegnare ad ogni funzione un numero progressivo nella numerazione e costruire una tabella infinita con tutte le funzioni.

x	0	1	2	3	4	5	6	7	8	...
$f_0(x)$	1	0	1	0	1	0	0	0	1	...
$f_1(x)$	0	0	1	1	0	0	1	1	0	...
$f_2(x)$	1	1	0	1	0	1	0	0	1	...
$f_3(x)$	0	1	1	0	1	0	1	1	1	...
$f_4(x)$	1	1	0	0	1	0	0	0	1	...

Definisco $g(x) = \begin{cases} 0 & f_x(x) = 1 \\ 1 & f_x(x) = 0 \end{cases} \Rightarrow g$ non può corrispondere a nessuna delle f_i della tabella, perché differisce da tutte le funzioni almeno nella diagonale principale.
$g(x) \mid 0\,1\,1\,1\ldots$
Per assurdo $\exists\, j \mid g(x) = f_j(x) \Rightarrow g(j) = f_j(j)$ ma per la definizione $g(j)$ è il complemento di $f_j(j)$, quindi $g(j) \neq f_j(j)$ **contraddizione**.
Per qualunque numerazione scelta esiste sempre almeno una funzione esclusa, quindi F non è numerabile.

1.4.1 Algoritmi

Algoritmi la **formulazione di un algoritmo**, una sequenza finita di operazioni, completamente e univocamente determinate, **dipende dal modello di calcolo utilizzato**.
Qualunque modello si scelga, gli algoritmi devono essere descritti da sequenze finite di caratteri di un alfabeto finito $\Rightarrow$ sono **possibilmente infiniti ma numerabili**.

Problemi computazionali Sono **funzioni matematiche** che associano ad ogni insieme di dati il corrispondente risultato, e **non sono numerabili** come visto prima.

Problema della rappresentazione C'è una drastica perdita di potenza, perché gli algoritmi sono numerabili ma sono meno dei problemi computazionali

$$|\{Problemi\}| >> |\{Algoritmi\}|$$

$\Rightarrow$ **esistono problemi privi di un corrispondente algoritmo di calcolo**. Per esempio, il problema dell'arresto.

Lezione di Turing *Non esistono algoritmi che decidono il comportamento di altri algoritmi esaminandoli dall'esterno, cioè senza passare dalla loro simulazione.*

1.4.2 Modelli di calcolo

La teoria della calcolabilità dipende dal modello di calcolo?
Oppure
la decidibilità è una proprietà del problema?

I linguaggi di programmazione esistenti sono tutti equivalenti?
Ce ne sono di alcuni più potenti/più semplici di altri?
Ci sono algoritmi descrivibili in un linguaggio ma non in un altro?
È possibile che problemi oggi irrisolvibili possano essere risolti in futuro con altri linguaggi o altri calcolatori?
Le teorie della calcolabilità e della complessità dipendono dal modello di calcolo?

Tesi di Church-Turing Tutti i *ragionevoli* modelli di calcolo **risolvono esattamente la stessa classe di problemi**, quindi **si equivalgono nella possibilità di risolvere i problemi** pur operando con diversa efficienza.

Tesi C-H: la decidibilità è una proprietà del problema

Incrementi qualitativi sui calcolatori o sui linguaggi di programmazione servono **solo** ad abbassare i tempi di esecuzione o rendere più agevole la programmazione.

1.4.3 Decidibilità e trattabilità

Ci sono quindi problemi che non possono essere risolti da nessun calcolatore, indipendentemente dal tempo impiegato (**problemi indecidibili**).
Ci sono poi problemi decidibili che possono richiedere tempi di risoluzione esponenziali nella dimensione dell'istanza (**problemi intrattabili**).
Ci sono poi problemi che possono essere risolti con algoritmi di costo polinomiale nella dimensione dell'input (**problemi trattabili**).
Abbiamo poi una famiglia di problemi il cui stato non è noto: clique (cricca), cammino hamiltoniano... Sappiamo risolverli (decidibili) con algoritmi di costo esponenziale, ma non abbiamo limiti inferiori esponenziali. I migliori limiti inferiori sono polinomiali: c'è un gap fra il limite inferiore (polinomiale) e costo della migliore soluzione a disposizione (esponenziale) (**presumibilmente intrattabili**).

Notazione

Studiamo la dimensione dei dati trattabili in funzione dell'incremento della velocità del calcolatori.
Dati i calcolatori C_1, C_2 (k volte più veloce di C_1) e tempo di calcolo a disposizione t, avrò n_1 dati trattabili in tempo t su C_1 e n_2 trattabili in tempo t su C_2.

Si **osserva** che **usare C_2 per un tempo t equivale a usare C_1 per un tempo $k \cdot t$.**

Algoritmi polinomiali Un algoritmo polinomiale che risolve il problema in $c \cdot n^s$ secondi, con c ed s costanti.

C_1 $c \cdot n_1^s = t \Rightarrow n_1 = (t/c)^{1/s}$

C_2 $c \cdot n_2^s = t \Rightarrow n_2 = k^{1/s} \cdot (t/c)^{1/s}$

$\Rightarrow$ $n_2 = k^{1/s} \cdot n_1$, miglioramento di un fattore **moltiplicativo** $k^{1/s}$

Algoritmi esponenziali Un algoritmo polinomiale che risolve il problema in $c \cdot 2^n$ secondi, con c costante.

C_1 $c2^{n_1} = t \Rightarrow 2^{n_1} = t/c$

C_2 $c2^{n_2} = k \cdot t \Rightarrow 2^{n_1} = k \cdot t/c = k2^{n_1}$

$\Rightarrow$ $n_2 = n_1 + \log_2(k)$, miglioramento di un fattore **additivo** $\log_2(k)$

Di conseguenza **un algoritmo efficiente è di gran lunga più importante di un calcolatore più potente**.

1.4.4 Tipologie di problemi

Dato un problema Π su un insieme di istanze in ingresso I con un insieme di soluzioni S.

Problemi decisionali Richiedono una risposta binaria $S = \{0, 1\}$, quindi istanze positive $x \in I \,|\, \Pi(x) = 1$ o negative $x \in I \mid \Pi(x) = 0$. Esempio: verifica se un numero è primo, o se un grafo è connesso.
La teoria della complessità computazionale è definita principalmente in termini di problemi di decisione: risposta binaria, quindi il tempo per restituire la risposta è costante, e la complessità di un problema è già presente nella versione decisionale.

Problemi di ricerca Data un'istanza x, richiede di restituire una soluzione s.

Problemi di ottimizzazione Data un'istanza x, si vuole trovare la **migliore** soluzione s tra tutte quelle possibili. Esempio: clique di dimensione massima, cammino minimo...

1.4.5 Classi di complessità

Dato un problema **decisionale** Π ed un algoritmo A, diciamo che A **risolve** Π se, data un'istanza di input x, $A(x) = 1 \Leftrightarrow \Pi(x) = 1$
A risolve Π in **tempo** $t(n)$ **e spazio** $s(n)$.

Classi Time e Space

Time($f(n)$): insieme dei **problemi decisionali che possono essere risolti in tempo O($f(n)$)**

Space($f(n)$): insieme dei **problemi decisionali che possono essere risolti in spazio O($f(n)$)**

Classe P Classe dei problemi risolvibili in **tempo** polinomiale nella dimensione dell'istanza di input.
Algoritmo polinomiale nel tempo: $\exists c, n_0 > 0 \mid$ il numero di passi elementari è al più n^c per ogni input di dimensione $n > n_0$.

Classe PSPACE Classe dei problemi risolvibili in **spazio** polinomiale nella dimensione dell'istanza di input. Molto più grande di P.
Algoritmo polinomiale nello spazio: $\exists\, c, n_0 > 0 \mid$ il numero di celle di memoria è al più n^c per ogni input di dimensione $n > n_0$.

Classe EXPTIME Classe dei problemi risolvibili in tempo esponenziale nella dimensione dell'istanza di input.

$$P \subseteq PSPACE \subseteq EXPTIME$$

Non è noto se queste inclusioni siano note, ad oggi. L'unico risultato dimostrato finora riguarda P ed EXPTIME: esiste un problema che può essere risolto in tempo esponenziale ma per cui il tempo polinomiale non è sufficiente (es: torri di Hanoi).

1.4.6 Certificato

Per alcuni problemi, per le istanze accettabili (istanze in cui la risposta del problema decisionale è si), è possibile certificare che quell'istanza è accettabile con un certificato y che può convincerci dell'accettabilità.
Per clique, il certificato è il sottoinsieme di k vertici che forma la clique. Per il cammino hamiltoniano è la permutazione degli n vertici che formano il cammino. Per SAT, sono le assegnazioni che rendono vera la formula. Il certificato ha dimensione polinomiale (k, n) e la verifica del certificato è lineare.
Una volta che ho il certificato lo vado a verificare: attestato breve di esistenza di una soluzione con determinate proprietà. Si definisce solo per istanze accettabili, perché spesso la non accettabilità non è facile costruire un certificato.

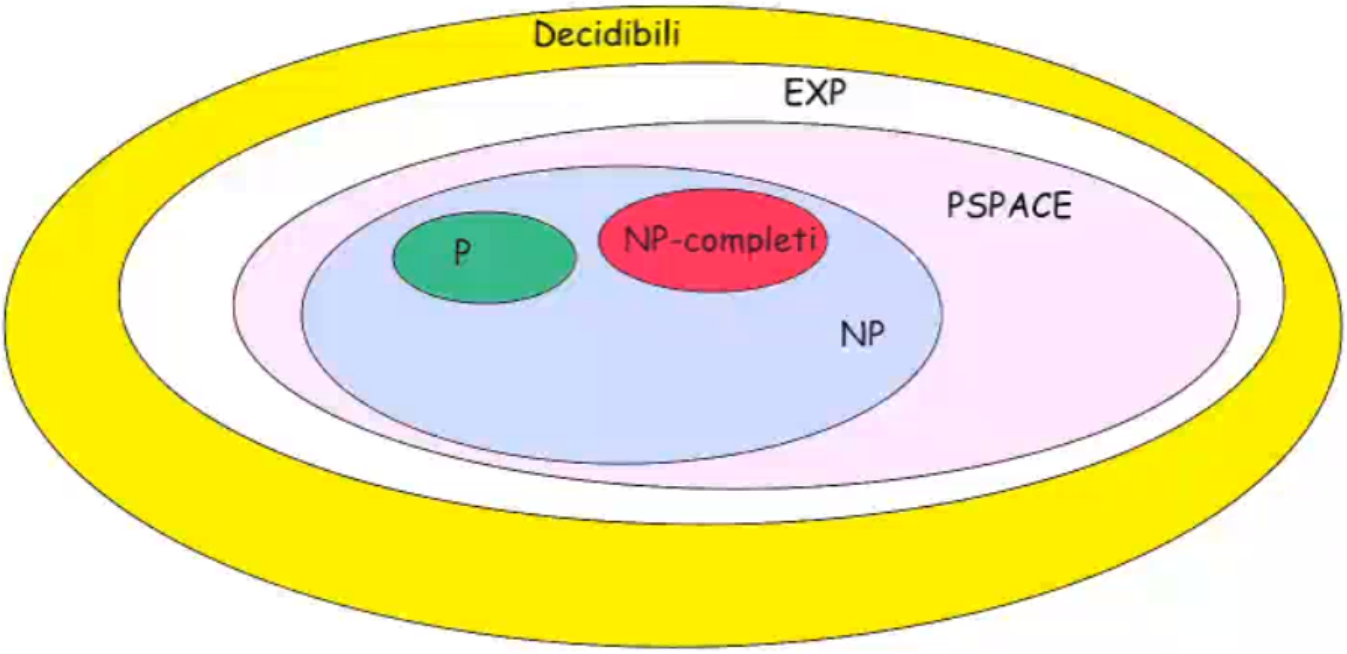
Idea Usare il costo della verifica di un certificato per un'istanza accettabile per **caratterizzare la complessità del problema** stesso.
Un problema è **verificabile in tempo polinomiale** se: tutte le istanze accettabili ammettono un certificato di lunghezza polinomiale ed esiste un algoritmo di verifica polinomiale in n.

Classe NP Classe dei problemi decisionali **verificabili in tempo polinomiale**. (NP = polinomiale su macchine non deterministiche)

P $\subset$ NP? Ovviamente si, ogni problema in P ammette un certificato verificabile in tempo polinomiale: eseguo l'algoritmo che risolve il problema per costruire il certificato.
Quello che non sappiamo è $P = NP$ oppure $P \neq NP$. Si pensa $P \neq NP$.
Si possono individuare i problemi più difficili in NP, ovvero quelli candidati ad appartenere ad NP se $P \neq NP$: sono i problemi NP-completi, quelli per cui se esiste un algoritmo polinomiale per risolvere un NP-completo allora tutti i problemi NP potrebbero essere risolti in tempo polinomiale e quindi P = NP.
Quindi tutti i problemi NP-completi sono risolvibili in tempo polinomiale oppure nessuno lo è.

Tutti i problemi NP-completi possono essere ridotti l'un l'altro, sono NP-equivalenti.

Gerarchia delle classi secondo le attuali congetture

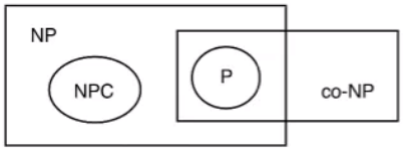


La fattorizzazione ad esempio $\in NP - (P \cup NPcompleti)$, infatti è risolto in tempo polinomiale su macchine quantistiche.

1.4.7 Classi co-P e co-NP

C'è molta differenza tra certificare l'esistenza e certificare la non esistenza di una soluzione. Dato un problema Π possiamo definire coΠ che accetta tutte e sole le istanze rifiutate da Π.
La classe coP è la classe per cui co$\Pi \in$ P. P = coP, i problemi complementari e i co-complementari (originali) si possono entrambi risolvere in tempo polinomiale: risolvo il problema complementare e complemento il risultato.

Questo non vale per coNP, la classe per cui co$\Pi \in$ NP. Si congettura che siano diverse, se la congettura è vera allora $P \neq NP$



Capitolo 2

Sequenze Casuali

2.1 Esempi di algoritmi numerici

Algoritmo di Euclide Algoritmo per il calcolo dell'MCD fra due interi.
Suppongo due interi a, b con $a \geq b$, $a > 0$ e $b \geq 0$
$\text{MCD}(a,b) = \begin{cases} a & b = 0 \\ MCD(a, a \bmod b) & else \end{cases}$

Valutazione complessità Data I istanza di input composta da a, b, vengono rappresentati ad esempio in base due. Quindi la dimensione n dell'istanza di input I $n = |I| = \Theta(\log a + \log b) = \Theta(\log a)$.
L'algoritmo è ricorsivo, quindi bisogna valutare il numero delle chiamate ricorsive, che dipenderanno dai dati. Ci saranno istanze in cui si termina subito (ad esempio se a multiplo di b, cioè $a \bmod b = 0$). In generale, **il numero di chiamate cresce con** $\log a$, perché $a \bmod b$ rimpiazza a.

Si osserva che $a \bmod b < \frac{a}{2}$
Questo perché $a = qb + (a \bmod b)$ e siccome per ipotesi $a \geq b \Rightarrow b \geq 1$ e lo è anche $q \Rightarrow a \geq b + (a \bmod b) > (a \bmod b) + (a \bmod b)$ perché $b > (a \bmod b)$.
$\Rightarrow 2(a \bmod b) < a \Rightarrow (a \bmod b) < \frac{a}{2}$.
Prima chiamata su a, b.
Seconda su $b, (a \bmod b)$.
Terza su $(a \bmod b), (b \bmod (a \bmod b))$.
Quindi ad ogni chiamata a si riduce almeno della metà, e lo possiamo fare al massimo $\log a$ volte
Quindi avrò $O(\log a)$ **ricorsioni**.

Il costo del calcolo del modulo è $O(\log a \cdot \log b) = O(\log^2 a)$
Complessivamente $\text{T}(n) = O(\log^3 a) = O(n^3)$ **polinomiale nella dimensione dell'istanza** $|I|$ (cioè nel numero di cifre), **polilogaritmico nel valore dei dati**

Test di primalità Versione inefficiente.
Primo(N): `for` $(i = 2, i \leq \sqrt{N}, i++)$
`if` $N\%i == 0$ `return false`
`else` *a fine ciclo* `return true`.
Uso la proprietà che se N non è primo allora ha almeno un divisore $\leq \sqrt{N}$.

Valutazione di complessità $I = N$, $|I| = \Theta(N) = n$
Ho $\sqrt{N}$ iterazioni, ciascuna di costo $\Theta(\log^2 N)$

$T(n) = O(\sqrt{N} \cdot \log^2 N) = O(2^{\frac{n}{2}} \cdot n^2)$ **pseudopolinomiale**, cioè **polinomiale nel valore di N ma esponenziale nella dimensione** $|I| = n$.

2.2 Casualità

Problema Data una sequenza binaria, vogliamo **capire se è una sequenza casuale** o meno. Le sequenze casuali sono importanti sia per la generazioni delle classi, sia perché **in crittografia spesso si ricorrono ad algoritmi randomizzati che usano sequenze casuali per funzionare**.

Significato algoritmico della casualità Vedendo la teoria di Kolmogorov. Prendiamo due sequenze

h: 1111…1 lunga n

h': 10110110101011010100101…0

La prima è molto facile da descrivere (*scrivi n "uni"*), mentre **descrivere la seconda è molto meno pratico**: l'intuizione è che **una sequenza casuale non si può descrivere in modo compatto**.
Ponendo $n = 20$, la probabilità di generare h è $P(h = (1/20)^{20}$, $P(h') = (1/20)^{20}$ (1/2 per generare 1, 1/2 per generare 0…).
A_h algoritmo che genera h. Formalizzabile semplicemente (*genera n uni*)
$|A_h| = \#$ bit di A_h codificato in binario $= \log n +$ const (la parte costante è la generazione e l'output, varia solo n) $\Rightarrow$ con $\log n$ bit ne abbiamo descritti n.
$A_{h'} =$ `print` h', $|A_{h'}| > |n| = |h'|$

L'intuizione è **una sequenza binaria è casuale se non ammette un algoritmo di generazione la cui rappresentazione binaria sia più corta di** h. Se posso usare meno bit vuol dire che la sequenza ha una qualche regolarità.

Sistemi di calcolo Sono un infinità numerabile $S_1 \dots S_i \dots$
Prendiamo S_i, p programma che genera la sequenza h nel sistema S_i, cioè $S_i(p) = h$
Def: **la complessità di Kolmogorov di** h **nel sistema** S_i **è** $K_{S_i}(h) = min\{\ |p|\ \mid S_i(p) = h\}$ cioè la minima lunghezza del programma p che in S_i genera h stessa.

Se la sequenza h non segue alcuna legge semplice di regolarità, allora **il più breve programma in grado di generarla dovrà contenerla al suo interno**, cioè sarà almeno lungo quanto la sequenza stessa e la genererà trasferendola in output. Quindi $K_{S_i}(h) = |h| +$ `const`$_i$. La costante è la parte di programma che trasferisce in output, dipende da S_i ma non da h.

Sistema di calcolo universale Tra tutti i sistemi di calcolo possibili ne esiste uno **universale in grado di simulare tutti gli altri**. Lo chiamiamo S_u e lo prendiamo in considerazione.
Supponiamo $p \mid S_i(p) = h$, allora $S_u(\langle i, p\rangle) = S_i(p) = h$. Ottengo $q = \langle i, p\rangle$ programma che genera h in S_u

$|q| = |\langle i, p\rangle| = |i| + |p| = \log_2 i + |p|$ quindi la lunghezza di q dipende da i ma non da h.
$\forall\, h\ \forall\, i\ \ K_{S_u}(h) \leq K_{S_i}(h) + C_i$
L'uguale vale per le sequenze generate per simulazione di S_i non essendoci per S_u algoritmi più "brevi".
Il minore vale per sequenze generabili con programmi più corti (ad esempio per simulazione su un altro sistema $S_j \neq S_i$).

Def La complessita di Kolmogorov di una sequenza h è $K(h) = K_{S_u}(h)$

2.2.1 Sequenze casuali

Sequenza casuale Una sequenza h è casuale se $K(h) \geq |h| - \lceil \log_2 h \rceil$
Non entra in gioco come genero la sequenza, la **casualità è una proprietà della sequenza**.

Conteggio delle sequenze $\forall\, n, \exists$ sequenze casuali (secondo Kolmogorov) di lunghezza n
Dim: n, $S = 2^n$ # sequenze binarie di lunghezza n
T = # sequenze di lunghezza n NON casuali. L'obiettivo è dimostrare che T < S.
Pongo $N = \#$ sequenze binarie di lunghezza $< n - \lceil \log_2 n \rceil = \sum_{i=0}^{n-\lceil \log n \rceil - 1} 2^i = 2^{n - \lceil \log n \rceil} - 1$
Tra queste N sequenze ci sono anche i programmi che generano le T sequenze non casuali di lunghezza n.
$\Rightarrow T \leq N < S \Rightarrow T < S$

Quindi non solo esistono ma sono anche la **maggioranza**, essendo enormemente più numerose di quelle non casuali. Lo vediamo studiando il rapporto

$$\frac{T}{S} \leq \frac{N}{S} = \frac{2^{n-\lceil \log n \rceil}}{2^n} - \frac{1}{2^n} < \frac{1}{2^{\lceil \log n \rceil}} \quad \lim_{n \to +\infty} \frac{T}{S} = 0$$

Stabilire la casualità Data una sequenza arbitraria di lunghezza n, stabilire se è casuale secondo Kolmogorov è un problema **indecidibile**.

Dim: per assurdo suppongo esista un algoritmo Random | $\text{Random}(h) = \begin{cases} 1 & h \textit{ casuale} \\ 0 & \textit{altrimenti} \end{cases}$

Possiamo costruire l'algoritmo Paradosso che enumera tutte le possibili sequenze binarie in ordine crescente di lunghezza.

Paradosso:
`for` (binary h = 1 $\rightarrow$ infty) `do`
`if` ($|h| - \lceil \log |h| \rceil > |P|$ && Random(h) == 1) `return` h

P è una sequenza binaria che rappresenta la codifica del programma complessivo Paradosso + Random, quindi $|P|$ = |Paradosso| + |Random|, costante che non dipende da h, perché la sequenza h non compare in P ma solo come nome di variabile. Il valore rimane registrato fuori dal programma.
Paradosso quindi restituisce come risultato la prima sequenza casuale | $|h| - \lceil \log_2 |h| \rceil > |P|$
Quindi se $\exists$ sequenze casuali di qualsiasi lunghezza, quindi certamente ne esisterà una che soddisfa entrambe le condizioni dell'`if`, che viene generata.
Ma la prima condizione mi dice che il programma rappresentato da P è breve e genera h, quindi h non è casuale perché prodotta con un programma breve.
Quindi $K(h) \leq |P|$, cioè P genera h, ma $|P| < |h| - \lceil \log_2 |h| \rceil$ quindi h non è casuale.
Ma la seconda condizione dice h casuale, giungendo ad un **paradosso** dato dall'assumere l'esistenza di Random.

2.2.2 Sorgente binaria casuale

Generatore Genera una sequenza di bit con queste proprietà:

1. P(0) = P(1) = 1/2, cioè genera 1 o 0 a pari probabilità.
   Si può indebolire richiedendo P(0) > 0, P(1) > 0 immutabili nel tempo della generazione.

2. La generazione di un bit è indipendente dalla generazione degli altri.
   $\Rightarrow$ non si può prevedere il valore di un bit osservando quelli già generati.

Perché possiamo indebolire la prima proprietà? Supponiamo di essere in un caso in cui P(0) > P(1), allora è **sempre possibile bilanciare la sequenza**.
Supponiamo di generare 001100111000010100 e si **dividono a coppie** 00 11 00 11 10 00 01 01 00 e si scartano le coppie uguali. Si associano le coppie miste, ad esempio 01 $\rightarrow$ 0 e 10 $\rightarrow$ 1. Si presentano in modo equiprobabile, quindi la sequenza si ribilancia ottenendo 100 (caso poco significativo perché sequenza corta).

Esistono queste sorgenti? Non si sa. Nella pratica non è possibile garantire la perfetta casualità o l'indipendenza. Quindi sfrutteremo le casualità presenti in processi fisici o processi software.

Generatore di sequenze brevi

Fenomeni casuali presenti in natura Ad esempio il rumore su un microfono o il tempo di decadimento di alcune particelle, sfruttabili come sorgenti di casualità.
Il problema di questo approccio è che bisogna non avere accesso fisico ai dispositivi usati (es: microfono manomesso), oltre alla difficoltà pratica di usare certe sorgenti.

Processi software Come la temperatura, la posizione della testina del disco fisico...

Pseudocasuale Si genera la casualità **mediante un algoritmo**, **cercandola all'interno di processi matematici**. **Generatore di numeri pseudo-casuali**: ad esempio `rand()` del C.
Perché pseudocasuali? Perché sono algoritmi deterministici e brevi, quindi non risultano casuali secondo Kolmogorov.

Come funzionano? Partono da un *seed* (seme), breve sequenza che viene amplificata per creare una sequenza più lunga. Quindi un generatore è un **amplificatore di casualità**.

Input: seme (sequenza o valore breve)

Output: flusso di bit arbitrariamente lungo e periodico.
Al suo interno contiene una sottosequenza che si **ripete**, quindi **un generatore è tanto migliore tanto più è lungo il suo periodo**

Avendo $s = \#$ bit nel seme e n lunghezza della sequenza ottenuta dal generatore, con $n >> s$, ho **una sequenza diversa per ogni seme**, con 2^s possibili semi.
$\#$ sequenze diverse $2^s << 2^n$ $\#$ sequenze possibili

Generatore lineare $x_i = (a \cdot x_{i-1} + b) \bmod n$ con a, b, n interi positivi.
Il seme è un valore intero iniziale casuale x_0, quindi quando $x_i = x_0$ la sequenza si ripete.
Dobbiamo avere $\text{MCD}(b, n) = 1$, $(a-1)$ divisibile per ogni fattore primo di n e $(a-1)$ dev'essere un multiplo di 4 se anche n lo è.
Servono per garantire che il generatore produca una permutazione degli interi da 0 a $n-1$

2.3 Test statistici

Per valutare le sequenze prodotte da un generatore pseudocasuale.
Si valuta se la sequenza presenta le proprietà tipiche di una sequenza casuale:

test di frequenza

poker test: se sottosequenze siano distribuite in modo equo

test di autocorrelazione: verifica che non ci siano regolarità nella sequenza ottenuta

run test: verifica se sottosequenze massimali di elementi tutti ripetuti abbiano una distribuzione esponenziale negativa, cioè più sono lunghe meno sono frequenti.

Per le applicazioni crittografiche si richiede anche il **test di prossimo bit**, molto severo che implica tutti gli altri 4 test statistici. Intuitivamente, verifica che sia impossibile prevedere gli elementi della sequenza prima di generarli.

Test di prossimo bit Un generatore binario **supera** il test di prossimo bit **se non esiste un algoritmo polinomiale in grado di prevedere l'$i+1$-esimo bit della sequenza a partire dalla conoscenza degli i bit precedentemente generati con probabilità maggiore di 1/2**.
Quindi se si hanno a disposizione risorse polinomiale non si può prevedere il prossimo bit. I generatori che superano questo test sono detti **generatori crittograficamente sicuri**.

Generatore polinomiale Non è crittograficamente sicuro.
$x_i = (a_1 x_{i-1}^t + a_2 x_{i-1}^{t-1} + \ldots + a_t x_{i-1} + a_{t+1}) \bmod n$
$r_i = \frac{x_i}{n}$
Prima cifra decimale pari $\to 0$, dispari $\to 1$

2.4 Generatori crittograficamente sicuri

Come costruire generatori crittograficamente sicuri Si ricorre alle **funzioni one-way**: funzioni facili da calcolare ma difficili da invertire, cioè **computabili in tempo polinomiale ($x \mapsto f(x)$)**, ma **computazionalmente difficile invertire la funzione ($y \mapsto x = f^{-1}(y)$)**. Come costruire queste funzioni?

Idea Con f one-way, scelgo il seme x_0. Genero S: x $f(x)$ $f(x_1) = f(f(x))$ $\ldots$ $x_i = f(f(\ldots(x_{i-1})\ldots))$
Cioè si itera l'applicazione della funzione one-way un numero arbitrario di volte. L'**idea è restituire la sequenza al contrario**, perché se conosco x_{i+1} non riesco a calcolare facilmente x_i.
Ogni elemento della sequenza S si può calcolare efficientemente con l'elemento precedente, ma non dai valori successivi perché f è one-way. Si calcola S per un certo numero di passi senza svelare il risultato e si comunicano gli elementi in ordine inverso. Ogni elemento non è prevedibile in tempo polinomiale pur conoscendo quelli comunicati.

Generatori binari crittograficamente sicuri Si usano i **"predicati *hard core*" delle funzioni one-way**.
Un predicato hard core di una funzione one-way $f(x)$ è $b(x)$ se $b(x)$ è facile da calcolare quando x è noto, ma è difficile da prevedere se si conosce $f(x)$.

Un esempio di funzione one-way è l'elevamento a quadrato in modulo ($f(x) = x^2 \bmod n$ con n non primo). Un predicato hard core è "$b(x) = x$ è dispari"

Generatore BBS Crittograficamente **sicuro**.
$n = p \cdot q$, con p e q primi **grandi**.

$p \bmod 4 = 3$

$q \bmod 4 = 3$

$2\lfloor \frac{p}{4} \rfloor + 1$ e $2\lfloor \frac{q}{4} \rfloor + 1$ **primi fra loro**

$\Rightarrow$ y coprimo con n, si calcola $x_0 = y^2 \bmod n$ e **si usa come seme** per **calcolare una successione di** $m \leq n$ **interi**

$x_i = (x_{i-1})^2 \bmod n$
$b_i = 1 \Leftrightarrow x_{n-i}$ è dispari

$b_1 = 1 \Leftrightarrow x_{n-1}$ dispari
$b_0 = 1 \Leftrightarrow x_n$ dispari

Quindi $x_0 = b_n$, ottengo una sequenza del tipo

$x_0 \to x_1 \to \ldots \to x_{n-1} \to x_n$

$\Rightarrow$ $b_n \to b_{n-1} \to \ldots \to b_1 \to b_0$
La **sequenza viene restituita in ordine inverso** $b_0 b_1 \ldots$

2.4.1 Generatori di numeri pseudocasuali basati su cifrari simmetrici

Idea Prendere un cifrario simmetrico e la sua chiave. Anziché usarlo per costruire un crittogramma, si sostituisce il messaggio da cifrare con un **seme iniziale** legato al generatore. Si comincia a cifrare in questo modo: produce una sequenza imprevedibile per le proprietà del cifrario.
Di seguito, un esempio approvato dal FIPS:

Usa il DES

r = # bit delle parole che vengono prodotte
($r = 64$ nel DES)

s = seme casuale di r bit

m = # parole da produrre

k = chiave segreta del cifrario

```
Generatore(s, m){ //flusso di output di m*r bit
        d = <rapp. su r bit di data e ora>;
        y = C(d, k);
        z = s;
        for (i = 1; i <= n; i++) {
                xi = C(y xor z, k);
                z = C(y xor xi, k);
                output(xi);
        }
}
```

2.5 Algoritmi randomizzati

Si dividono in due classi fondamentali:

Algoritmi **Las Vegas**
Generano un **risultato sicuramente corretto** in un **tempo probabilmente breve**.
Caso tipico: quick sort. Qualche passo a caso per cercare di evitare i casi sfavorevoli.

Algoritmi **Montecarlo**
Generano un **risultato probabilmente corretto** in un **tempo sicuramente breve**.
Probabilità di errore deve essere **arbitrariamente piccola e matematicamente misurabile**.
Caso tipico: test di primalità.

2.5.1 Test di primalità (Miller, Rabin)

N intero (dispari) da testare di n bit
$\Rightarrow$ N-1 è pari, $N-1 = 2^w \cdot z$ con z dispari, w esponente della potenza di 2 più grande che divide $N-1$
Es: $N = 21, N-1 = 20 = 2^2 \cdot 5$ quindi $z = 5, w = 2$

Da N, calcolo $N-1 \to \frac{N-1}{2} \to \frac{N-1}{4} \to \ldots \to z = 1$ quindi # divisioni per $2 \leq \log N = n$ volte
Quindi trovo w e z in maniera efficiente, $O(n)$ passi

Sia quindi N primo e $2 \leq y \leq N-1$ arbitrario detto *testimone*, allora

P1 MCD(N,y) = 1, per la primalità

P2 $(y^z \bmod N = 1) \vee (\exists\, i \;\; 0 \leq i \leq w-1 \mid y^{2^i \cdot z} \bmod N = -1)$

Se una delle due proposizione è falsa allora N non è primo, ma ci sono numeri composti che verificano P1 e P2 ma non sono primi (sono pochi).

Lemma 1 (Miller, Rabin) N è composto, allora il numero di testimoni y che soddisfano i predicati è basso.
Cioè N composto $\Rightarrow$ il numero di interi $y \mid 2 \leq y \leq N-1$ che soddisfano entrambi i predicati P1 e P2 è minore di $N/4$
La probabilità di scegliere un testimone che rende veri P1 e P2 $< \frac{N/4}{N-2} < \frac{1}{4}$

N y scelto a caso in $[2, N-1]$:

se uno dei due predicati è **falso** $\to N$ è **certamente composto**

se sono entrambi veri $\Rightarrow N$ è composto con probabilità $< \frac{1}{4}$, dunque N è primo con probabilità $> \frac{3}{4}$

Iterando il test k volte, la probabilità di errore diventa $< \left(\frac{1}{4}\right)^k$, con $k = 30$ diventa inferiore al 10^{-18}.
Di seguito l'algoritmo.

```
Verifica(N,y) { //controlla la validita del certificato y (certifica che N sia composto)
        if (P1 == false or P2 == false) return 1; //N certamente composto
        else return 0; //N probabilmente primo (prob errore < 1/4)
}

TestMR(N,k) {
        for (i = 0; i < k; i++) {
                //sceglie a caso y in [2, N-1]
                if (Verifica(N, y) == 1) return 0; //N non primo
        }
        return 1; //N probabilmente primo (prob errore < (1/4)^k)
}
```

Valutazione di complessità TestMR costa k volte Verifica, quindi valuteremo quest'ultimo.
Il calcolo dell'MCD, quindi di P1, è facile. Quindi indaghiamo la valutazione di P2: bisognerà calcolare

$y^z \bmod N == 1$ e, in caso non sia verificato

$y^{2^i \cdot z} \bmod N$, con $0 \leq i \leq w-1$

Nella seconda parte di P2 và calcolato $y^z \bmod N$, poi $y^{2z} \bmod N$ come quadrato del precedente.
L'esponente massimo per y è $i = w-1$ (da $N-1 = 2^w \cdot z$), perché $2^{w-1} \cdot z = \frac{N-1}{2}$
Quindi $y^{\frac{N-1}{2}} \bmod N$, al massimo voglio eseguire $\log N$ moltiplicazioni. La moltiplicazione la sappiamo fare polinomiale, e l'elevamento a potenza possiamo eseguirlo polinomiale con l'**algoritmo delle quadrature successive** o esponenziazione veloce.
w elementi al quadrato con $w = O(\log N)$.
Quindi l'algoritmo MR dà un test efficiente per la primalità.

Algoritmo delle quadrature successive Vogliamo calcolare $x = y^z \bmod s$, con x, z, s stesso ordine di grandezza.

Si scompone z in una somma di potenze di 2
$z = \sum_{i=0}^{t} k_i \cdot 2^i$ con $k_i \in \{0, 1\}$
Esempio $z = 45 = 32 + 8 + 4 + 1$
Il massimo t come visto è $t = \lfloor \log_2 z \rfloor = \Theta(\log z)$

Si calcolano tutte le potenze $y^{2^i} \bmod s$ per $1 \leq i \leq t = \lfloor \log_2 z \rfloor$, ciascuna come il quadrato della precedente.
$y^{2^{i+1}} \bmod s = \left(y^{2^i}\right)^2 \bmod s$
Esempio: $x = 9^{45} \bmod 11$ e $45 = 32 + 8 + 4 + 1$, $t = 5$

$y^2 \bmod s = 9^2 \bmod 11 = 4$
$y^4 \bmod s = 4^2 \bmod 11 = 5$
$y^8 \bmod s = 5^2 \bmod 11 = 3$
$y^{16} \bmod s = 3^2 \bmod 11 = 9$
$y^{32} \bmod s = 9^2 \bmod 11 = 4$

Calcoliamo $x = y^z \bmod s = \Pi_{(i \mid k_i \neq 0)} \left(y^{2^i}\right) \bmod s$
Nell'esempio:
$y^z \bmod s = 9^{45} \bmod 11 = 9^{32+8+4+1} \bmod 11 =$
$((9^{32} \bmod 11) \cdot (9^8 \bmod 11) \cdot (9^4 \bmod 11) \cdot (9^1 \bmod 11)) \bmod 11 =$
$(4 \cdot 3 \cdot 5 \cdot 9) \bmod 11 = 1$

Costo: $t = \Theta(\log_2 z)$ quadrature e al più t moltiplicazioni $\Rightarrow \Theta(\log_2 z)$ quadrature e $O(\log z)$ moltiplicazioni. Ogni moltiplicazione ha un costo al più quadratico nel numero di cifre.
Quindi l'algoritmo è **polinomiale nella dimensione dei dati**.

2.5.2 Generazione di numeri primi

In pratica è una **generazione di un numero casuale seguita da un test di primalità**. Se il test fallisce, lo si incrementa di due iterando fino a trovare un numero *dichiarato* primo.
Sono pochi i numeri da testare grazie alla densità: il numero di interi primi e minori di N tendono a $\frac{N}{\log_e N}$ per $N \to \infty$.
Quindi per N sufficientemente grande, in un suo intorno di ampiezza $\log_e N$ cade mediamente un numero primo.

```
Primo(n): { //n: # bit del numero generato
        //genera un numero primo di almeno n bit
        //probabilita errore < (1/4)^k
        S = seq di n-2 bit prodotti da un generatore binario pseudocasuale
        N = (1 S 1) //N ha n bit ed e dispari
        while(TestMR(N,k) == 0) { N = N + 2 } //O(n) = O(logN) volte
        //TestMR costo polinomiale in n = logN -> O(n^3)
        return N;
}
```

L'algoritmo è polinomiale, circa $O(n^4)$

2.6 Classe RP

Random Polinomial Classe dei problemi decisionali verificabili in tempo polinomiale randomizzato.
Dato un problema Π, e x istanza di input allora y è un **certificato probabilistico** per l'istanza x **se ha una lunghezza** $|y|$ **al più polinomiale** in $|x|$ **e** y **è estratto perfettamente a caso da un insieme associato a** x

$A(x, y)$ in tempo polinomiale **attesta con certezza che** x ***non*** **possiede la proprietà esaminata da** Π, cioè $\Pi(x) = 0$, **oppure attesta che** x **possiede la proprietà** esaminata da Π **con probabilità** $> \frac{1}{2}$.

Capitolo 3

Cifrari storici

Scopo Consentire comunicazioni **sicure** tra poche persone, ma **i cifrari storici sono stati tutti forzati**.
La **cifratura e decifrazione** erano tutte **realizzate a carta e penna**, mentre i **messaggi** da cifrare erano **frasi di senso compiuto** in linguaggio naturale, quindi con l'alfabeto classico di 26 lettere.

3.1 Principi di Bacone

XIII Secolo

C e D devono essere **funzioni facili da calcolare**

Impossibile ricavare D se C non è nota

Il **crittogramma** $c = \mathrm{C}(m)$ deve **apparire innocente**

3.2 Antichi esempi

3.2.1 Scitale

Metodo più antico di cui si ha notizia, inventato dagli spartani nel V secolo a.C.
Asta cilindrica costruita in due esemplari identici posseduti dai due corrispondenti.

3.2.2 Erodoto, Storie

Si tatuava il messaggio sulla testa rasata di un messaggero, si aspettava che ricrescessero i capelli e si portava a destinazione, rivelando il messaggio a seguito di una seconda rasatura.

3.2.3 Enea Tattico

Opera militare del IV secolo a.C. con un capitolo dedicato ai messaggi segreti. Consigliava di inviare un libro qualsiasi sottolineandovi un sottoinsieme di lettere che costituiscono il messaggio, oppure di sostituire le vocali di un testo con altri simboli grafici.

3.2.4 Cifrario di Cesare

Il più antico cifrario di concezione moderna. L'idea è che il crittogramma sia ottenuto dal messaggio in chiaro sostituendo ogni lettera con quella di 3 posizioni più avanti nell'alfabeto.
Non ha una chiave segreta, e la segretezza dipende dalla conoscenza del metodo: scoprire il metodo significa compromettere irrimediabilmente l'impiego. Il cifrario era quindi destinato all'uso ristretto di un gruppo di conoscenti.

Generalizzandolo a k posizioni più avanti, si rende più sicuro ($1 \leq k \leq 25$) e si ha k come chiave segreta.

Formulazione matematica

Con pos(X) indichiamo la **posizione nell'alfabeto della lettera X**: ad esempio pos(A) = 0, pos(Z) = 25...
La chiave k è quindi $1 \leq k \leq 25$, con $k = 26$ il cifrario lascerebbe invariato il messaggio.

Cifratura di X Lettera Y | pos(Y) = (pos(X) + k) mod 26

Decifrazione di Y Lettera X | pos(X) = (pos(Y) - k) mod 26

Un esempio con $k = 10$. Cifriamo R, cui pos(R) = 17. La sua cifratura è (17+10) mod 26 = 1 = pos(B)
Quindi R $\rightarrow$ B.

Crittoanalisi

Se si conosce la struttura del cifrario, in breve tempo **si possono applicare tutte le chiavi possibili**, che sono solo 25, ad un crittogramma: così viene **decifrato** e contemporaneamente si scopre la chiave segreta k. Come cifrario risulta quindi inutilizzabile a fini crittografici.

Gode della proprietà commutativa: data una sequenza di chiavi e di operazioni di cifratura e decifrazione, l'**ordine delle operazioni può essere permutato arbitrariamente** senza modificare il crittogramma finale.
Per esempio, date k_1 e k_2 chiavi e s sequenza,

$$\mathrm{C}(\mathrm{C}(s, k_2), k_1) = \mathrm{C}(s, k_1 + k_2)$$

$$\mathrm{D}(\mathrm{D}(s, k_2), k_1) = \mathrm{D}(s, k_1 + k_2)$$

Una sequenza di operazioni di cifratura e decifrazione può essere ridotta ad una sola operazione di cifratura o decifrazione.

Inoltre **comporre più chiavi non aumenta la sicurezza** del sistema.

3.3 Classificazione dei cifrari storici

3.3.1 Cifrari a sostituzione

Sostituiscono ogni lettera del messaggio in chiaro con una o più lettere dell'alfabeto secondo una regola prefissata.

Sostituzione monoalfabetica Es: Cifrario di Cesare.
Alla stessa lettera del messaggio corrisponde sempre una stessa lettera nel crittogramma.
Si possono impiegare funzioni di cifratura e decifrazione più complesse dell'addizione e della sottrazione in modulo, ottenendo uno spazio delle chiavi molto più ampio (ma sempre con una sicurezza molto modesta).

Per esempio, usare come $k = \langle a, b \rangle$, cifrare pos(Y)=(a·pos(X)+b) mod 26 e decifrare
pos(X) = $a^{-1} \cdot$ (pos(Y)$-b$) mod 26 con a^{-1} inverso di a mod 26 ($a \cdot a^{-1} = 1$ mod 26).
C'è quindi un **vincolo forte** sulla chiave: MCD(a, 26) $\neq 1 \Rightarrow$ la funzione di cifratura non è iniettiva e la decifrazione diventa impossibile. Ad esempio con $k = \langle 13, 0 \rangle$ tutte le lettere posizione pari sono trasformate in A, mentre tutte quelle dispari sono trasformate in N.
Le chiavi possibili sono quindi 12 (scelte per a) $\cdot$ 26 (scelte per b) = 312 chiavi che sono poche.

Se la segretezza dipende unicamente dalla chiave, allora **il numero di chiavi deve essere così grande da essere praticamente immune dal brute force** e deve essere **scelta in modo casuale**.

Cifrario completo prendendo una permutazione arbitraria dell'alfabeto come chiave:
lettera in chiaro in posizione i $\longrightarrow$ lettera di posizione i della permutazione.

La chiave è di 26 lettere, con uno spazio delle chiavi esteso a $26! - 1$, cioè circa $4 \cdot 10^{26}$ chiavi, il che lo rende molto vasto e inesplorabile.
Ma non è comunque sicuro: si può forzare senza ricorre a brute force, sfruttando la struttura logica dei messaggi in chiaro e l'occorrenza statistica delle lettere.

Sostituzione polialfabetica
Alla stessa lettera del messaggio corrisponde una lettera scelta in un insieme di lettere possibili, secondo una regola opportuna (a seconda della posizione o del contesto in cui appare la lettera nel messaggio). L'esempio più antico è l'archivio cifrato di Augusto.
I documenti in archivio erano scritti in numeri, invece che lettere. Augusto li scriveva in greco, poi metteva in corrispondenza la sequenza di lettere del documento con la sequenza di lettere del primo libro dell'Iliade. Sostituiva ogni lettera del documento con il numero che indicava la distanza, nell'alfabeto greco, di tale lettera con quella in pari posizione nell'Iliade.
Per esempio:

> Lettera in posizione i nel documento: α
>
> Lettera in posizione i nell'Iliade: ϵ
>
> Carattere in posizione i nel crittogramma: 4 (distanza tra α e ϵ)

Se la chiave è lunghissima, il cifrario diventa difficile da forzare. Venne usato anche della Seconda Guerra Mondiale, prendendo come chiave una pagina prefissata di un libro e cambiandola di giorno in giorno. Lo **svantaggio** è **registrare per iscritto la chiave**.

Cifrario di Alberti con due dischi, dove si cambia chiave ogni volta che si incontra un carattere speciale. Inserendo spesso caratteri speciali (scartati nel messaggio ricostruito) diventa difficile da attaccare e il continuo cambio di chiave rende inutili gli attacchi basati sulla frequenza dei caratteri. La **Macchina Enigma** è un'**estensione elettromeccanica del cifrario di Alberti**.

Cifrario di Vigenère, con una parola segreta k come chiave. La cifratura di un messaggio m avviene disponendo m e k su due righe adiacenti, allineando le lettere in verticale (se k è più corta di m la si ricopia più volte). Ogni lettera X di m risulta allineata ad una lettera Y della chiave. La X è sostituita nel crittogramma con la lettera che si trova nella cella T all'incrocio tra la riga che inizia con X e la colonna che inizia con Y

A	B	C	...	X	Y	Z
B	C	D	...	Y	Z	A
C	D	E	...	Z	A	B
...	...	...	...	...	...	...
X	Y	Z	...	U	V	W
Y	Z	A	...	V	W	X
Z	A	B	...	W	X	Y

Quindi le lettere allineate con A non subiscono modifiche. Quelle allineate con B sono traslate di una posizione in avanti, quelle con R di 17 posizioni...
Una stessa lettera in chiaro è cifrata in modi diversi a seconda della lettera con cui è allineata, mentre per la decifrazione si esegue il processo inverso.
La sicurezza del metodo è influenzata dalla lunghezza della chiave: se contiene h caratteri, le apparizioni della stessa lettera distanti un multiplo di h nel messaggio si sovrappongono alla stessa lettera della chiave, quindi sono trasformate nella stessa lettera cifrata.
I cifrari polialfabetici non sono molto più potenti dei monoalfabetici se le chiavi non sono molto lunghe.
Se si estende Vigenèere impiegando una chiave lunga quanto il testo, casuale e non riutilizzabile, il cifrario diventa inattaccabile: **one-time pad**.

3.3.2 Cifrari a trasposizione

Permutano le lettere del messaggio in chiaro secondo una regola prefissata.
L'idea di base è eliminare qualsiasi struttura linguistica presente nel crittogramma: permutando le lettere del messaggio in chiaro e inserendone eventualmente altre ignorate nella decifrazione.

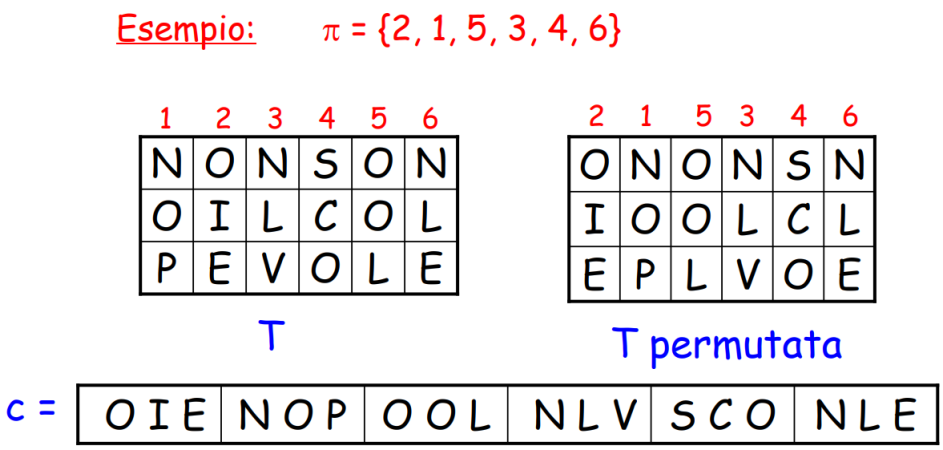
Semplice La chiave è un intero h e una permutazione π degli interi $\{1, 2, \dots, h\}$
Nella cifratura si suddivide il messaggio in m blocchi da h lettere e si permutano le lettere di ciascun blocco secondo π.
Se la lunghezza di m non è divisibile per h, si aggiungono alla fine delle lettere qualsiasi (padding): partecipano alla trasposizione, ma sono ignorate perché la decifrazione le riporta alla fine del messaggio.
Ci sono $h! - 1$ chiavi ed h non è fissato a priori: tanto è più grande tanto è più difficile impostare un brute force. Al crescere di h però cresce anche la difficoltà di ricordare π.

Permutazione di colonne $k = \langle c, r, \pi \rangle$ con:

c ed r denotano il numero di colonne e righe di una tabella di lavoro T

π è una permutazione degli interi $\{1, 2, \dots, c\}$

Il messaggio m è decomposto in blocchi $m_1, m_2 \dots$ di $c \cdot r$ caratteri.
Nella cifratura i caratteri di ogni blocco sono distribuiti tra le celle di T in modo regolare, scrivendoli per righe dall'alto verso il basso. Poi le colonne vengono permutate secondo π e si prendono le colonne dalla prima leggendole dall'alto verso il basso e da sinistra verso destra, ottenendo il crittogramma.



Per cifrare il prossimo blocco, si azzera T e si ripete il procedimento.
Le chiavi sono teoricamente esponenziali nella lunghezza del messaggio, non essendoci vincoli su r e c.

Cifrario a griglia Antenato: cifrario di Richelieu. Il crittogramma può essere celato in un libro qualsiasi. La chiave è data da una scheda perforata e dall'indicazione di una pagina del libro: la decifrzione consiste nel sovrapporre la scheda alla pagina, e le lettere visibili attraverso i fori costituiscono il messaggio in chiaro.

3.4 Crittoanalisi Statistica

La sicurezza di un cifrario è legata alla dimensione dello spazio delle chiavi. Ci sono altri metodi di attacco: i cifrari storici sono stati violati con un attacco statistico di tipo cipher text.
Nella crittoanalisi statistica si fanno delle ipotesi. Ci sono delle **informazioni note al crittoanalista**: il metodo usare per la cifratura e la decifrazione, il linguaggio naturale con cui è scritto il messaggio e si ammette che il messaggio sia sufficientemente lungo per poter rilevare alcuni dati statistici sui caratteri che compongono il crittogramma.

Attacco La frequenza con cui appaiono in media le varie lettere dell'alfabeto è ben studiata in ogni lingua. Dati simili sono noti per le frequenze di diagrammi (gruppi di due lettere consecutive), trigrammi (gruppi di tre lettere) e così via (**q-grammi**).

Decifrazione cifrari monoalfabetici Se un crittogramma è generato per sostituzione monoalfabetica allora la frequenza di Y nel crittogramma è circa la frequenza della corrispondente X del messaggio.
Nei **cifrari completi** si associano le lettere in base alle frequenze, provando le varie combinazioni. Con le combinazioni di prova, si studiano le frequenze dei possibili diagrammi, dei trigrammi e così via.

Decifrazione cifrari polialfabetici Più difficile. Per esempio in Vigenère ogni lettera Y del crittogramma dipende da una coppia di lettere $\langle$X, K$\rangle$ provenienti dal messaggio e dalla chiave.
Se la chiave è di h caratteri, il crittogramma è composto di h sottosequenze, ciascuna ottenuta per sostituzione monoalfabetica. Il problema è scoprire h per scomporre il crittogramma e continuare la decifrazione con il metodo monoalfabetico.
Quindi il messaggio contiene quasi sicuramente gruppi di lettere adiacenti ripetuti più volte. Le apparizioni della stessa sottosequenza allineate con la stessa porzione di chiave danno sottosequenze identiche. Si cercano nel crittogramma coppie di posizioni p_1, p_2 in cui indiziano sottosequenze identiche. La distanza $d = p_2 - p_1$ è *probabilmente* uguale ad h o ad un suo multiplo.

Decifrazione di cifrari a trasposizione Le lettere nel crittogramma sono le stesse del messaggio in chiaro, non ha senso un attacco statistico basato sulle frequenze: si studiano i q-grammi
Se si conosce la lunghezza h della chiave, si divide il crittogramma in porzioni di lunghezza h, in ciascuna si cercano gruppi di q lettere che formano i q-grammi più diffusi. Se un gruppo deriva effettivamente da un q-gramma, si scopre parte della permutazione.

Conclusione La rilevazione delle frequenze delle singole lettere del crittogramma è un potente indizio per discernere tra i vari tipi di cifrario:

nei cifrari a trasposizione, l'istogramma delle frequenze coincide circa con quello proprio del linguaggio

nei cifrari a sostituzione monoalfabetica, i due istogrammi coincidono a meno di una permutazione delle lettere

nei cifrari a sostituzione polialfabetica, l'istogramma del crittogramma è assai più appiattito di quello del linguaggio

3.5 La Macchina Enigma

Automatismo Prima evoluzione verso i sistemi automatizzati, la **macchina Enigma** ha avuto un ruolo fondamentale nella seconda guerra mondiale. Ci sono stati **molti studi dedicati a comprometterne la sicurezza**, che hanno posto le **fondamenta per la nascita dei calcolatori odierni**.
Fu inventata in Germania nel 1918 per applicazioni commerciali, come estensione elettromeccanica del cifrario di Alberti.

Rotori La macchina era composta da rotori, una tastiera, una griglia di luci sopra la tastiera corrispondenti alle lettere (lampboard) e una plugboard inferiore.
I rotori **non mantenevano la stessa posizione reciproca durante la cifratura**. Per ogni lettera battuta sulla tastiera:

Il primo rotore avanzava di un passo

Dopo 26 passi (# di lettere), il primo rotore era tornato sulla posizione iniziale e il secondo rotore avanzava di un passo

Dopo una rotazione completa del secondo rotore, il terzo rotore avanzava di un passo.

Questo fa si che la corrispondenza tra caratteri cambiasse ad ogni passo, cioè **la chiave cambiava ad ogni lettera premuta**: $26 \cdot 26 \cdot 26 = 17476$ chiavi diverse.

Debolezza I rotori erano immutabili, inoltre le 26^3 permutazioni sono sempre le stesse, applicate nello stesso ordine e note a tutti i proprietari di una macchina Enigma.
Alberti prevedeva, per ogni coppia di utenti, una coppia di dischi diversa da tutte le altre.

Modifiche Possibilità di permutare tra loro i tre rotori: # permutazioni $= 3! \cdot 26^3 > 10^5$
Inoltre fu aggiunta la plugboard fra la tastiera e il primo rotore: consente di **scambiare fra loro i caratteri di 6 coppie scelte arbitrariamente in ogni trasmissione**. Ogni cablaggio è descritto da una sequenza di 12 caratteri (le 6 coppie da scambiare): $\binom{26}{12} \simeq 10^7$ combinazioni possibili. Ogni gruppo di 12 caratteri può presentarsi in 12! permutazioni diverse, ma non tutte producono effetti diversi: ad esempio AB CD EF GH IJ KL e CD AB EF GH IJ KL producono lo stesso effetto, e con queste anche tutte le 6! permutazioni delle 6 coppie.
Infine vanno considerati i possibili scambi tra gli elementi delle coppie che producono lo stesso effetto:
AB CD EF GH IJ KL e BA CD EF HG IJ KL ad esempio, quindi dobbiamo dividere per un ulteriore fattore $2^6 = 64$

Numero di chiavi Il numero di chiavi dei rotori ($> 10^5$) si moltiplica per un fattore $\binom{26}{12} \frac{12!}{6! \cdot 64} > 10^{11}$
per un totale di più di 10^{16} chiavi (10 *milioni di miliardi* di combinazioni possibili)

Seconda Guerra Mondiale 8 rotori in dotazione, da cui sceglierne 3, e 10 coppie scambiabili tramite plugboard. Elenco di chiavi giornaliere in dotazione ai reparti militari, cioè l'assetto iniziale della macchina per quel giorno. Con quell'assetto, si trasmetteva una nuova chiave di messaggio, che indicava l'assetto da usare per quella particolare trasmissione.

Storia di Enigma Era lo standard militare tedesco durante la seconda guerra mondiale. Matematici polacchi e inglesi studiarono come rompere il cifrario (**Bletchley Park**): era necessaria una **rapida decifrazione** perché il sistema è continuamente variato.

> Costruzione di un simulatore di Enigma per studiarne il comportamento sotto possibili variazioni.
> Alan Turing e altri.
>
> Macchina COLOSSUS
> 1944
>
> Prototipo embrionale dei successivi calcolatori elettronici.

Capitolo 4

Cifrari perfetti

Shannon, 1949

Cifrario perfetto Sono cifrari che ci offrono una **sicurezza incondizionata**, proteggono le informazioni con certezza assoluta indipendentemente dalla potenza di calcolo. Si rende **essenziale avere la chiave**, un'attacco **brute force non permette di decifrare**.
Informalmente, un cifrario è perfetto se la **sicurezza è garantita qualunque sia l'informazione carpita dal canale di comunicazione**.
Abbiamo quindi due spazi:

MSG: lo **spazio dei messaggi**

CRITTO: lo **spazio dei crittogrammi**

e due variabili aleatorie:

M: descrive il **comportamento del mittente**, con valori in MSG

C: descrive il **processo di comunicazione**, con valori in CRITTO

Ho delle probabilità:

$P(M = m)$: probabilità che il mittente voglia spedire il messaggio m

$P(M = m \mid C = c)$: probabilità condizionata che il messaggio inviato sia m posto che sul canale stia transitando il crittogramma c

Con $\forall\ m \in$ MSG e $\forall\ c \in$ CRITTO.
Scenario: il crittoanalista conosce tutto del sistema tranne la chiave, quindi conosce:

la distribuzione di probabilità con cui il mittente invia i messaggi

cifrario utilizzato

lo spazio delle chiavi (KEY)

Definizione Un cifrario è perfetto se $\forall\ m \in$ MSG e $\forall\ c \in$ CRITTO, allora $P(M = m) = P(M = m \mid C = c)$, cioè la conoscenza di c non raffina la conoscenza. In un cifrario perfetto, la conoscenza del crittoanalista non cambia dopo che è stato osservato un crittogramma in transito. Quindi **il messaggio ed il crittogramma appaiono del tutto scorrelati, nessuna informazione sul messaggio filtra dal crittogramma**.

4.0.1 Teorema di Shannon

In un cifrario perfetto il numero delle chiavi deve essere maggiore o uguale del numero dei messaggi possibili (cioè con probabilità non nulla di essere inviati)

Dim Per assurdo

$N_m = \#$ dei messaggi possibili, cioè $m \in$ MSG $\mid P(M = m) > 0$

$N_k = \#$ delle chiavi

Suppongo per assurdo che $N_k < N_m$. Suppongo $c \in$ CRITTO con $\mathrm{P}(\mathrm{C} = c) > 0$. Conto i messaggi che possono corrispondere al crittogramma $\Rightarrow s$ messaggi potenzialmente corrispondenti a c. Questi s messaggi sono i messaggi che posso ottenere decifrando c con ogni chiave possibile: con la chiave k_1 ottengo m_1, k_2 ottengo m_2, ma anche possibile che con k_3 ottenga di nuovo m_2 dato che non so come funziona il cifrario. Posso dire con certezza che $s \leq N_k$ e per ipotesi $N_k < N_m$
Ottengo $s < N_m \Rightarrow \exists\, m \in \mathrm{MSG}$ con $\mathrm{P}(\mathrm{M} = m) > 0 \mid \mathrm{P}(\mathrm{M} = m \mid \mathrm{C} = c) = 0$, ottenendo una conoscenza maggiore (posso escludere m) quindi il cifrario non è perfetto, ottenendo una **contraddizione**.

4.0.2 One-Time Pad

1917, Mauborgne, Vernam
Usa l'alfabeto binario {0,1}, usato sia durante le guerre mondiali che durante la guerra fredda. L'idea è avere una chiave lunghissima consumata man mano e mai riutilizzata.
MSG, CRITTO e KEY sono spazi di sequenze binarie. La regola è pubblica e usa lo xor. Quindi sia C che D usano lo xor (somma in modulo 2) per trasformare il messaggio nel crittogramma e viceversa.
$m \in \{0,1\}^n$ e $k \in \{0,1\}^n$ con $n > 0$ fissato.
$c = C(m,k) = m \oplus k$ Con $n = 5$, $m = 10110$, $k = 01011$ allora $c = 11101$.
La decifrazione $m = D(c,k) = c \oplus k = (m \oplus k) \oplus k = m \oplus (k \oplus k) = m$ perché $k \oplus k = 0 \ldots 0$
Fondamentale non riutilizzare la chiave. Questo perché avendo c', c'' ottenuti con la stessa k da m', m'', faccio $c' \oplus c'' = (m' \oplus k) \oplus (m'' \oplus k) = m' \oplus m'' \oplus (k \oplus k) = m' \oplus m''$, acquisendo conoscenza (lunghi tratti di 0 significa tratti uguali nei messaggi in chiaro).

Teorema Quando il One-Time Pad è perfetto. Ipotesi:

1. Tutti i messaggi hanno lunghezza n, con del padding o divisione in blocchi all'occorrenza
2. Tutte le sequenze di n bit sono messaggi possibili (per quelle prive di senso probabilità molto bassa ma comunque > 0)
3. Chiave scelta **perfettamente a caso per ogni messaggio**

Il **teorema** dice che sotto 1, 2 e 3 One-Time Pad è **un cifrario perfetto e minimale** (usa un numero minimo di chiavi)

Dim Dimostro che è perfetto, cioè che $\forall\, m \in$ MSG e $\forall\, c \in$ CRITTO, allora $\mathrm{P}(\mathrm{M} = m) = \mathrm{P}(\mathrm{M} = m \mid \mathrm{C} = c)$
$\mathrm{P}(\mathrm{M} = m \mid \mathrm{C} = c) = \frac{P(M=m \wedge C=c)}{P(C=c)} = \frac{P(M=m)\cdot P(C=c)}{P(C=c)} = \mathrm{P}(\mathrm{M} = m)$
Per le proprietà dello xor, fissato m chiavi diverse producono crittogrammi diversi $\Rightarrow \exists!$ chiave $k \mid m \oplus k = c\ \forall\, c \in$ CRITTO, $\mathrm{P}(\mathrm{C} = c) = \mathrm{P}$(scegliere l'unica chiave $k \mid m \oplus k = c) = \frac{1}{2^n}$, questo perché $\{\mathrm{M} = m\}$ e $\{\mathrm{C} = c\}$ sono **eventi indipendenti**.
Per quanto riguarda la minimalità, cioè $N_k \geq N_m$, il One-Time Pad per ogni n ha $N_k = N_m = N_{CRITTO} = 2^n$, questo perché anche le chiavi sono sequenze di n bit come i messaggi e i crittogrammi.

Attacchi

Bruteforce: non ha senso, ogni chiave fa ricostruire un messaggio possibile quindi non aggiunge conoscenza

Rimane il problema di generare le sequenze di bit completamente casuali.

Osservazione Rimuoviamo la seconda ipotesi nella dimostrazione. Per esempio in inglese i messaggi significativi sono circa $\alpha^n << 2^n$ con $\alpha = 1.1$
Quindi $N_m = \alpha^n$, $N_k \geq N_m$ cioè $N_k \geq \alpha^n$, la chiave sarà una sequenza di t bit con $t \mid 2^t \geq \alpha^n$
Risolvo trovando $t \geq \log_w \alpha^n = n \log_2 \alpha = 0.12 \cdot n$

Per confondere l'avversario è opportuno che coppie diverse (m,k) producano lo stesso crittogramma $\Rightarrow$ # coppie $(m,k) >>$ # crittogrammi, cioè $\alpha^n 2^t >> 2^n$

4.1 Principi di Shannon

Principi per la resistenza agli attacchi di crittoanalisi statistica:

Diffusione: tutti i caratteri del testo in chiaro si devono spargere nel crittogramma.

Confusione: combinare testo in chiaro e chiave in modo complesso, per non permettere al crittoanalista di sapere le due sequenze analizzando il crittogramma.

Standard 1972 NBS (National Bureau of Standards) → 1973 NIST (National Institute for Security and Technology)

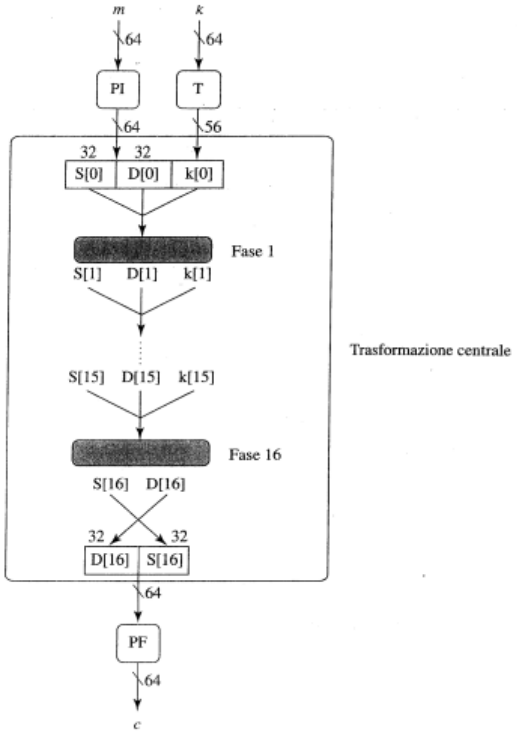
Sicurezza basata sulla segretezza della chiave e non sul processo di cifratura e decifrazione (che è **pubblico**)

Algoritmo efficiente in software e hardware

IBM propone **LUCIFER**, poi migliorato con l'NSA. Da 128 bit a 56 bit per la chiave: variazioni nella s-box.
Si arriva al 1977 con il **DES**, accettato e reso pubblicamente disponibile (licenza d'uso gratuita): **Data Encryption Stantard**. Rimane fino al 1999, quando si sconsiglia il DES in favore del 3DES. Dal 2005 anche il 3DES risulta superato, al suo posto **AES** (**Advanced Encryption Standard**)

4.2 Data Encryption Standard

Cifratura a blocchi di 64 bit e chiave segreta di 64 bit: 56 bit casuali e 8 bit di parità suddivisi in: 7 bit, bit parità, 7 bit, bit parità...
$r = 16$ fasi in cui si ripetono le stesse operazioni.



Trasformazione centrale

Struttura del DES

m: blocco del messaggio

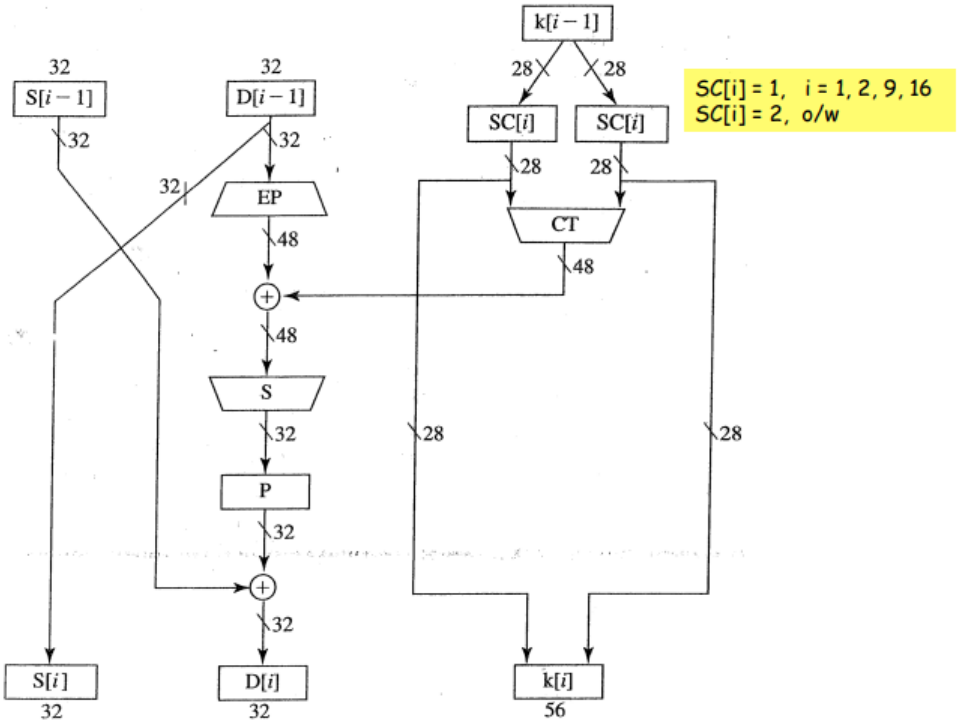
c: corrispondente blocco del crittogramma

k: chiave segreta con i bit di parità

$\forall\, i = 1, \ldots, 16$ ho

$S[i] = D[i-1]$
$D[i] = S[i-1] \oplus f(D[i-1], k[i-1])$
f: funzione **non** lineare

Fase i-esima del DES



Permutazioni

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Permutazione PI

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Permutazione PF

57	49	41	33	25	17	9	
1	58	50	42	34	26	18	
10	2	59	51	43	35	27	
19	11	3	60	52	44	36	
63	55	47	39	31	23	15	
7	52	54	46	38	30	22	
14	6	61	53	45	37	29	
21	13	5	28	20	12	4	

Trasposizione T

Le tabelle vanno lette per righe.

Permutazione PI: riordina i bit del messaggio $m = m_1 m_2 \dots m_{64}$ come $m_{58} m_{50} \dots m_7$: ad esempio porta in posizione 40 (numero della cella) il bit in posizione 1 (contenuto della cella)

Permutazione PF: è l'inversa di PI, cioè nell'esempio riporta in posizione 1 il bit in posizione 40

Trasposizione T: provvede anche a scartare dalla chiave $k = k_1 k_2 \dots k_{64}$ i bit di controllo $k_8, k_{16}, \dots k_{64}$, generando una sequenza di 56 bit che costituisce la prima sottochiave $k[0]$

Funzioni CT e EP

14	17	11	24	01	05
03	28	15	06	21	10
23	19	12	04	26	08
16	07	27	20	13	02
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Funzione CT

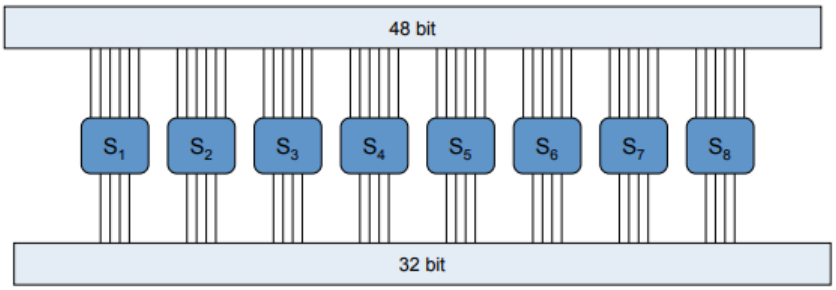
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Funzione EP

Funzione CT: 8 bit dell'ingresso (es: il bit 9) non sono presenti in uscita

Funzione EP: 16 bit di ingresso sono duplicati (es: il bit 32 è copiato nelle posizioni 1 e 47 dell'uscita)

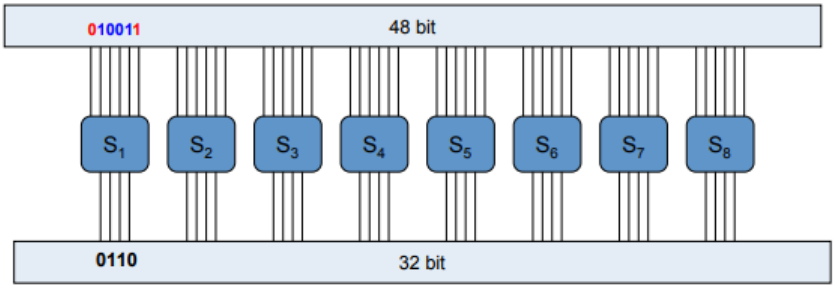
S-box



S_1

x \ y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Tabella che definisce la sottofunzione S_1. Le sottofunzioni $S_2, \ldots, S_8$ sono definite in modo simile.



S_1

x \ y	0	1	2	3	4	5	6	7	8	**9**	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
→ 1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

Permutazione P

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Permutazione di 32 bit che genera il blocco finale $D[i]$.

Attacchi Due metodi principali:

1. Architetture apposite progettate per attaccare il DES
2. Calcolo distribuito su più macchine

Attacchi esaurienti **Chosen Plain Text**: il crittoanalista si procura coppie $\langle m, c_1\rangle, \langle \overline{m}, c_2\rangle$
Da $C(m,k) = \begin{matrix} c_1 \Rightarrow k \textit{ probabile chiave} \\ c_2 \Rightarrow \overline{k} \textit{ probabile chiave} \end{matrix}$

Crittoanalisi differenziale 2^{47} coppie $\langle m, c\rangle$ scelti dal crittoanalista. L'attacco costa complessivamente 2^{55} con $r = 16$.

4.2.1 Varianti del DES

Scelta indipendente delle sottochiavi di fase # bit 56 $\rightarrow$ 16·48 = 768 bit
Crittoanalisi differenziale $\rightarrow 2^{61}$

Cifratura multipla $\forall\, k_1, k_2$ ho $C_D(C_D(n,k_1),k_2) \neq C_D(n,k_3)$, questo $\forall\, n, k_3$
spazio delle chiavi = 2^{112}, non sono 112 bit di sicurezza
Sicurezza pari a quella di una chiave da 57 bit

Attacco "Meet in the Middle" c = C(C(n, k_1), k_2), con k_1, k_2 chiavi di 56 bit $\rightarrow$ D(c, k_2) = C(n, k_1)
Si prende una coppia $\langle n, c\rangle$

$\forall\, k_1$ calcolo e salvo C(n, k_1) (2^{56})

$\forall\, k_2$ calcolo D(c, k_2) e lo cerco nella lista delle cifrature
Deve **esistere certamente almeno una corrispondenza** ($N = 2^{56}$)

N = 2^{56} cifrature + O(N) decifrazioni
Quindi costo 2N << N^2 (costo dell'enumerazione di tutte le coppie (k_1, k_2))

3DES TDEA = Triple Data Encryption Algorithm
2TDEA, 3TDEA: 2 e 3 sono # chiavi

2TDEA
c = C(D(C(n, k_1), k_2), k_1), con k_1, k_2 chiavi di 56 bit tra loro indipendenti.
$k_1 = k_2 \Rightarrow$ 2TDEA equivale ad una singola cifratura DES. CDC non è più robusto di CCC, 112 bit di sicurezza.

3TDEA
c = C(D(C(n, k_1), k_2), k_3), con k_1, k_2, k_3 chiavi di 56 bit tra loro indipendenti.
bit della chiave: 56·3 = 168 bit
Vulnerabile a meet in the middle $\rightarrow$ 112 bit di sicurezza

$m = D(C(D(c, k_3), k_2)k_1) \Rightarrow C(m, k_1) = C(D(c, k_3), k_2)$

1. Enumero le chiavi di 56 bit e salvo la lista $\mathrm{C}(m, k_1)\ \forall\, k_1 \in \{0,1\}^{56}$
2. $\forall\, k_2, k_3$ calcolo $C(D(c, k_3), k_2)$ e lo cerco nella lista
   $2^{56} + 2^{112}$, 2^{56} dal punto 1 e 2^{112} costo di enumerazione delle coppie k_2, k_3

AES 128 bit di chiave
Ogni sottochiave di fase, $w(i)$, è una sequenza di 4 byte che usa la s-box.

$k =$

			1byte
w(0)	w(1)	w(2)	w(3)

Chiave: $w(0), w(1), w(2), w(3)$

$\forall\, t \geq 4$ ho $w(t) = \begin{cases} w(t-1) \oplus w(t-4) & se\ 4 \not| t \\ T(w(t-1)) \oplus w(t-4) & se\ 4 \mid t \end{cases}$ con T non lineare e che usa la s-box

Chiave alla i-esima fase, $1 \leq i \leq 10$: $w(4i), w(4i+1), w(4i+2), w(4i+3)$

Cifratura Blocchi di 128 bit, riempita per colonne $b_{ij} \in \{0,1\}^8$

$$B = \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix}$$

Trasformazione iniziale: k matrice della chiave, con B costruisco $B \oplus k$
10 fasi da 4 operazioni:

01 `SUBISTITUTE BYTES`
Ogni byte di B è trasformato usando s-box: $b_{ij} \rightarrow \text{s-box}(b_{ij})$

02 `SHIFT ROWS`

03 `MIX COLUMNS`, non si applica nella fase 10

04 `ADD ROUND KEY`

Alla fine delle 10 fasi, il blocco B è il crittogramma

S-box Matrice 16×16 di interi $\in (0, 255) \Rightarrow$ contiene una permutazione
$b_{ij} \rightarrow \text{s-box}(b_{ij}) \in \{0,1\}^8$
$b_{ij} = b_1 b_2 b_3 b_4 | b_5 b_6 b_7 b_8$

Riga: $b_1 b_2 b_3 b_4$, $0 \leq x \leq 15$

Colonna: $b_5 b_6 b_7 b_8$, $0 \leq y \leq 15$

Es $b_{ij} = 1000|1011 = 8|11$, s-box$[8, 11] \rightarrow 61$ cioè 00111101

x byte $\longrightarrow_{s-box} x^{-1}$ inverso moltiplicativo + composizione lineare in GF(2^8)
Galois Field, campi finiti di Galois

02 `SHIFT ROWS`

$$\begin{matrix} \underline{b_{00}} & b_{01} & b_{02} & b_{03} \\ \underline{b_{10}} & b_{11} & b_{12} & b_{13} \\ \underline{b_{20}} & b_{21} & b_{22} & b_{23} \\ \underline{b_{30}} & b_{31} & b_{32} & b_{33} \end{matrix} \rightarrow \begin{matrix} \underline{b_{00}} & b_{01} & b_{02} & b_{03} \\ b_{11} & b_{12} & b_{13} & \underline{b_{10}} \\ b_{22} & b_{23} & \underline{b_{20}} & b_{21} \\ b_{33} & \underline{b_{30}} & b_{31} & b_{32} \end{matrix}$$

03 `MIX COLUMNS`
M matrice di 4×4 byte. Ogni colonna del blocco B $B_j \rightarrow M \cdot B_j$, con $0 \leq j \leq 3$
Il nuovo b_{ij} diventa un valore che dipende da tutti i byte della colonna $(b_{0j}, b_{1j}, b_{2j}, b_{3j})$
$b_{ij} \rightarrow b_{ij} \oplus k_{ij}$

Capitolo 5

Crittografia a Chiave Pubblica

Cifrari ibridi, usati per lo **scambio delle chiavi**.

One-Time Pad 1917
Non può essere decifrato senza conoscere la chiave. **Assolutamente sicuro**, ma richiede una nuova chiave segreta per ogni messaggio, che deve essere perfettamente casuale e lunga quanto il messaggio da scambiare. Diventa molto attraente per chi richiede una sicurezza assoluta ed è disposto a pagarne i costi.
Ma **come si genera** e **come si scambia la chiave**?

5.1 AES

Advanced Encryption System Standard per le comunicazioni riservate ma non-classificate. Pubblicamente noto e realizzabile su hardware di ogni tipo, con chiavi brevi (128/256 bit, qualche decina di caratteri)
Ma come scambiare in sicurezza una chiave segreta? La chiave serve per comunicare in sicurezza, ma deve essere stabilita comunicando *in sicurezza* senza poter ancora usare il cifrario.

5.1.1 Protocollo DH

Nel 1976 viene proposto un algoritmo per generare e scambiare una chiave segreta su un canale insicuro, senza necessità che le due parti si siano scambiate informazioni o incontrate in precedenza. Il **protocollo DH** è un algoritmo ancora usato nei protocolli crittografici su Internet. Diffie e Hellman propongono anche la definizione di crittografia a chiave pubblica, ma senza avere un'implementazione pratica.

5.1.2 Cifrari simmetrici

Nei **cifrari simmetrici**, la **chiave di cifratura è uguale a quella di decifrazione** (o l'una può essere facilmente calcolata dall'altra) ed è **nota solo ai due partner** che la scelgono di comune accordo e la mantengono **segreta**.

5.1.3 Cifrari asimmetrici

Nei **cifrari a chiave pubblica** l'**obiettivo** è **permettere a tutti di inviare messaggi cifrati ma abilitare solo il ricevente** (Bob) **a decifrarli**.
Le **operazioni di cifratura e decifrazione sono pubbliche** e usano **due chiavi diverse**:

k_{pub} per **cifrare**: **pubblica**

k_{priv} per **decifrare**: **privata**, nota solo a Bob

Esiste una coppia $\langle k_{pub}, k_{priv} \rangle$ per ogni utente del sistema, scelta da questi nella sua veste di possibile destinatario.

Cifratura La **cifratura di un messaggio da spedire** a Bob è eseguita da qualunque mittente come

$$c = C(m, k_{pub})$$

La chiave k_{pub} e la funzione di cifratura $C(m, k)$ sono note a tutti.

Decifrazione La **decifrazione di un messaggio ricevuto** da Bob è eseguita da Bob come

$$m = D(c, k_{priv})$$

La funzione di decifrazione $D(c, k)$ è nota a tutti **ma la chiave k_{priv} non è disponibile agli altri**.

Ruoli Ruoli completamente diversi svolti da mittente e destinatario di un messaggio, che hanno invece ruoli intercambiabili nei cifrari simmetrici dove condividono la solita informazione (chiave segreta)

Tre elementi

1. **Correttezza del processo di cifratura e decifrazione**
   Bob deve interpretare qualunque messaggio che gli altri utenti decidano di spedirgli. Quindi **per ogni possibile messaggio** $m \Rightarrow D(\, C(m, k_{pub}),\, k_{priv}) = m$
2. **Efficienza e sicurezza del sistema**

   Generazione casuale delle chiavi
   La **coppia di chiavi è facile da generare** e deve risultare praticamente impossibile che due utenti scelgano la stessa chiave.

   Adottabilità del sistema
   Dati m e k_{pub}, è **facile per Alice calcolare il crittogramma** $c = C(m, k_{pub})$
   Dati c e k_{priv}, è **facile per Bob calcolare il messaggio** $m = D(c, k_{priv})$
3. **Sicurezza del cifrario**
   Pur conoscendo il crittogramma c, la chiave pubblica e le funzioni C e D, è **difficile per il crittoanalista risalire al messaggio** m

Cifratura La funzione C deve essere una **one-way trapdoor**: calcolare $c = C(m, k_{pub})$ deve essere **computazionalmente facile**, ma **decifrare** c **computazionalmente difficile**. Questo a meno di un **meccanismo segreto** (**trapdoor**) rappresentato da k_{priv}

5.2 RSA

Rivest, Shamir, Adleman Proposto come un sistema a chiave pubblica basato su una funzione *facile* da calcolare e *difficile* da invertire: la moltiplicazione di due primi p e q.

Calcolare $n = p \cdot q$ è **facile**

Calcolare p, q da n è **difficile**, **a meno di non conoscere uno dei due fattori**.

Fa uso dell'**algebra modulare**:

Riduce lo spazio dei numeri su cui si opera, e quindi aumenta la velocità di calcolo

Rende **difficili** problemi computazionali che risultano facili o banali nell'algebra non modulare.

Le funzioni tendono a comportarsi in modo imprevedibile.

Funzione di Eulero Numero di interi minori di n e coprimi con esso

$$\phi(n) = |Z_n^*|$$

n primo $\Rightarrow \phi(n) = n - 1$
$Z_n = \{0, 1, 2, \ldots, n-1\}$, e $Z_n^* \subseteq Z_n$ è l'insieme degli elementi di Z_n coprimi con n. Se n è primo allora $Z_n^* = \{1, 2, \ldots, n-1\}$. Se non è primo, calcolare Z_n^* è computazionalmente difficile (direttamente proporzionale al valore di n.
Teorema n composto $\Rightarrow \phi(n) = n \cdot (1 - \frac{1}{p_1}) \cdot \ldots \cdot (1 - \frac{1}{p_k})$ con $p_1, \ldots, p_k$ fattori primi di n senza molteplicità.
Teorema n semiprimo (prodotto di due primi), cioè $n = p \cdot q \Rightarrow \phi(n) = (p-1) \cdot (q-1)$

Teorema di Eulero Per $n > 1$ e $\forall\, a$ primo con n si ha $a^{\phi(n)} \equiv 1 \bmod n$
Piccolo teorema di Fermat Per n primo e $\forall\, a \in Z_*$ si ha $a^{n-1} \equiv 1 \bmod n$

Come conseguenze si ha che per qualunque a primo con n $a \cdot a^{\phi(n)-1} \equiv 1 \ mod\ n$ e $a \cdot a^{-1} \equiv 1 \ mod\ n$ e quindi

$$a^{-1} \equiv a^{\phi(n)-1} \ mod\ n$$

L'inverso a^{-1} di a mod n si può calcolare per esponenziazione di a se si conosce $\phi(n)$. In generale, nell'algebra modulare **l'esistenza dell'inverso non è garantita** perché a^{-1} deve essere intero.

$ax \equiv b$ mod n ammette soluzione $\Leftrightarrow$ MCD$(a, n) \mid b$ con MCD(a, n) soluzioni distinte.
Quindi $ax \equiv b$ mod n ammette **unica soluzione** $\Leftrightarrow$ MCD$(a, n) = 1 \Leftrightarrow \exists\, a^{-1}$ inverso di a
$ax \equiv 1$ mod n ammette esattamente una soluzione $(a^{-1}) \Leftrightarrow a, n$ primi fra loro.

5.2.1 Generatori

$a \in Z_n^*$ è un **generatore di** Z_n^* se la funzione a^k mod n con $1 \leq k \leq \phi(n)$ **genera tutti e soli gli elementi di** Z_n^*.
Produce come risultati tutti gli elementi di Z_n^* ma in un ordine difficile da prevedere.
Teorema di Eulero $a^{\phi(n)}$ mod $n = 1 \Rightarrow 1 \in Z_n^*$ è un generatore per $k = \phi(n)$
Per ogni generatore $a^k \not\equiv 1$ mod n

Esempio 3 genera Z_7^*
$Z_7^* = \{1, 2, 3, 4, 5, 6\}$, $\phi(7) = 6$
3^k mod $7 = 3, 2, 6, 4, 5, 1$ con $1 \leq k \leq 6 = \phi(7)$

Esistono valori di n per cui Z_n^* non ha generatori. Ad esempio, Z_8^* non ammette generatori.
Teorema: se n è primo $\Rightarrow$ Z_n^* ha almeno un generatore.
Per n primo, non tutti gli elementi di Z_n^* sono suoi generatori. Ad esempio 1 non è mai generatore, e altri elementi possono non essere generatori. Per n primo sappiamo che i generatori di Z_n^* sono in totale $\phi(n-1)$

Problemi sui generatori rilevanti in crittografia

Determinare un generatori di Z_n^* con n primo Si possono provare tutti gli interi in $[2, n-1]$ fino a trovare il generatore: tempo **esponenziale** nella dimensione di n. Il problema è computazionalmente difficile, risolto con **algoritmi randomizzati**, con alta probabilità di successo.

Calcolo del logaritmo discreto Risolvere in x l'equazione $a^x \equiv b$ mod n con n primo.
Ammette una soluzione per ogni valore di $b \Leftrightarrow a$ è un generatore di Z_n^*. Tuttavia:

non è noto a priori in che ordine sono generati gli elementi di Z_n^*

quindi non è noto per quale valore di x si genera b mod n

un esame diretto della successione richiede tempo esponenziale nella dimensione di n

non è noto un algoritmo polinomiale di soluzione

Funzioni One-Way Trapdoor

Esistono funzioni matematiche che sembrano possedere i requisiti richiesti (proprietà di teoria dei numeri e di algebra modulare). Il loro calcolo risulta **incondizionatamente semplice** e la loro **inversione** è **semplice se si dispone di un'informazione aggiuntiva sui dati** (chiave privata). Senza questa informazione, l'inversione richiede la soluzione di un problema NP-hard, o comunque di un problema noto per cui non si conosce un algoritmo polinomiale.
Alcuni esempi:

Fattorizzazione $n = p \cdot q$ è facile, tempo quadratico nella lunghezza della rappresentazione.
Invertire, cioè ricostruire p e q da n richiede tempo esponenziale (per quanto noto fin'ora, anche se non vi è dimostrazione che sia NP-hard. L'esistenza di un algoritmo polinomiale è improbabile ma non da escludere)
Trapdoor: se si conosce p o q, la chiave segreta, ricostruire l'altro è facile.

Calcolo della radice in modulo Calcolare $y = x^z$ mod s con x, z, s interi richiede tempo polinomiale ($\Theta(\log_2 z)$ moltiplicazioni con esponenziazioni successive)
Se s non è primo, invertire e calcolare $x = y^{\frac{1}{z}}$ mod s richiede tempo esponenziale per quanto noto.
Se x è primo con s e si conosce $v = z^{-1}$ mod $\phi(s)$ (chiave segreta), x si può facilmente determinare con $x = y^v$ mod s per il teorema di Eulero.

Calcolo del logaritmo discreto Calcolare $y = x^z \bmod s$ è facile, ma invertire rispetto a z cioè calcolare z tale che $y = x^z \bmod s$ dati x, y, s è difficile.
Gli algoritmi noti hanno la stessa complessità della fattorizzazione, quindi si può introdurre una trapdoor.

5.2.2 Crittografia a chiave pubblica

Vantaggi

Se gli utenti di un sistema sono n, il numero complessivo di chiavi (pubbliche e private) è $2n$ invece di $n(n-1)/2$

Non è richiesto alcuno scambio segreto di chiavi

Svantaggi

Sono sistemi molto più lenti dei cifrari simmetrici

Sono esposti ad attacchi chosen plain-text

5.2.3 Cifrari ibridi

Si usa un cifrario a chiave segreta (AES) per le comunicazioni di massa

e un cifrario a chiave pubblica per scambiare le chiavi segrete relative al primo, senza incontri fisici tra gli utenti

La trasmissione dei messaggi lunghi avviene ad alta velocità, mentre è lento lo scambio delle chiavi

Le chiavi sono composte al massimo da qualche decina di byte

L'attacco chosen plain-text è risolto se l'informazione cifrata con la chiave pubblica (chiave segreta dell'AES) è scelta in modo da risultare imprevedibile al crittoanalista

La chiave pubblica deve essere estratta da un certificato digitale valido, per evitare attacchi man-in-the-middle (MITM)

5.2.4 RSA

Il cifrario è diviso in varie fasi.

Creazione della coppia di chiavi Il destinatario ha l'onere di creare le chiavi:

DEST sceglie p e q primi **molto grandi** (migliaia di bit): $n = p \cdot q$ deve avere 2048 bit, ancora più a lungo termine 3072 bit.
Si esegue in tempo polinomiale, primalità via Miller-Rabin

DEST calcola $n = p \cdot q$, $\phi(n) = (p-1) \cdot (q-1)$
Anche questo in tempo polinomiale

DEST sceglie $e < \phi(n) \mid \text{MCD}(e, \phi(n)) = 1$ (coprimo con $\phi(n)$)
Polinomiale

DEST calcola $d = e^{-1} \bmod \phi(n)$
$\exists!$ perché e è coprimo con $\phi(n)$
Polinomiale con algoritmo di Euclide esteso

DEST rende pubblica $k[pub] = \langle e, n \rangle$ **chiave pubblica**
Tiene private $k[priv] = \langle d \rangle$ **chiave privata**

Messaggio Sequenza binaria trattata come un intero $m < n$. Se $m \geq n$, m si divide in blocchi di $b = \lfloor \log_2 n \rfloor$ bit, cifrati indipendentemente, sennò ci sarebbero collisioni in fase di decifratura.
Nella pratica si fissa un limite inferiore comune per la dimensione dei blocchi $b \mid m < 2^b < n$

Cifratura Mittente si occupa di costruire $c = C(m, k[pub]) = m^e \bmod n \Rightarrow c < n$
Polinomiale via quadrature successive

Decifrazione Destinatario è l'unico che può ricavare $m = D(c, k[priv]) = c^d \bmod n$
Polinomiale via quadrature successive

Esempio $p = 5, q = 11 \Rightarrow n = 55, \phi(n) = (p-1)(q-1) = 40$
$e = 7$, MCD$(7, 40) = 1$ ok
$d = 7^{-1} \bmod 40 = -17 \bmod 40 = 23 \bmod 40 = 23$

EE(7, 40) $\uparrow \langle 1, \underline{-17} \rangle$

EE(40, 7) $\uparrow \langle 1, 3, -17 \rangle$

EE(7, 5) $\uparrow \langle 1, -2, +3 \rangle$

EE(5, 2) $\uparrow \langle 1, 1, 0 - \lfloor 5/2 \rfloor \cdot 1 \rangle = \langle 1, 1, -2 \rangle$

EE(2, 1) $\uparrow \langle 1, 0, 1 - \lfloor 2/1 \rfloor \cdot 0 \rangle = \langle 1, 0, 1 \rangle$

EE(1, 0) $\rightarrow \langle 1, 1, 0 \rangle$

Correttezza Dim che D(C$(m, k[pub])$, $k[priv]) = m$
$c^d \bmod n = (m^e \bmod n)^d \bmod n = m^{ed} \bmod n = m$
Teorema $\forall\, m < n$ si ha $m^{ed} \bmod n = m$, **dimostro** per casi:

1. p, q non dividono m
   $\Rightarrow$ MCD$(m, n) = 1$, sono coprimi
   $\Rightarrow m^{\phi(n)} \equiv 1 \bmod n$ (th. Eulero)
   $\Rightarrow e \cdot d \equiv 1 \bmod \phi(n)$, quindi $e \cdot d = 1 + r \cdot \phi(n)$ con $r \in N$ (def. di inverso)
   $\Rightarrow m^{ed} \bmod n = m^{1+r\phi(n)} \bmod n = m \cdot \left(m^{\phi(n)}\right)^r \bmod n = m \cdot 1^r \bmod n = m \bmod n = m$ perché $m < n$ ok
2. m, n non sono coprimi, suppongo $p \mid m$ e $q \nmid m$ (analogo invertendo p e q)
   $\Rightarrow p \mid m \Rightarrow m \equiv 0 \bmod p$
   $\Rightarrow \forall\, r \in N$ ho $m^r \equiv 0 \bmod p$
   $\Rightarrow \forall\, r \in N$ ho $m^r - m \equiv 0 \bmod p$
   $\Rightarrow$ con $r = e \cdot d$ ho $m^{ed} - m \equiv 0 \bmod p$
   $\Rightarrow$ MCD$(q, m) = 1$ sono coprimi
   $\Rightarrow m^{\phi(q)} \equiv 1 \bmod q$
   $\Rightarrow m^{ed} \bmod q = m^{1+r\phi(n)} \bmod q = m \cdot m^{r(p-1)(q-1)} \bmod q = m \left(m^{q-1}\right)^{r(p-1)} \bmod q =$
   $= m \left(m^{\phi(q)}\right)^{r(p-1)} \bmod q = m(1)^{r(p-1)} \bmod q = m \bmod q$
   $\Rightarrow m^{ed} - m$ è divisibile per p e per q, allora è divisibile anche per $n = p \cdot q$, quindi $m^{ed} - m \equiv 0 \bmod n$
   $\Rightarrow m^{ed} \equiv m \bmod n = m^{ed} \equiv m \bmod n$ e perché $m < n$ ho $m \bmod n = m$
3. p, q dividono m non si verifica perché $m < n$

Sicurezza Legata alla difficoltà di fattorizzare un numero arbitrario molto grande.
Fattorizzare $\Rightarrow$ forzare RSA ok, è sufficiente per forzarlo
Fattorizzare $\Leftarrow$ forzare RSA, non sappiamo se fattorizzare è necessario per forzarlo.
Il calcolo della radice in modulo $m = \sqrt[e]{c} \bmod n$ difficile almeno quanto la fattorizzazione (n composto)

Calcolare $\phi(n)$ direttamente da n è computazionalmente equivalente a fattorizzare n
Significa che un problema si trasforma nell'altro in tempo polinomiale.

$n = pq \longrightarrow \phi(n) = (p-1)(q-1)$, due riduzioni e un prodotto, polinomiale

$\phi(n) = (p-1)(q-1) \rightarrow \phi(n) = pq - (p+q) + 1 = n - (p+q) + 1$
$x_1 = p + q = n - \phi(n) + 1$
$\rightarrow (p-q)^2 = (p+q)^2 - 4pq = (p+q)^2 - 4n = x_1^2 - 4n$
$x_2 = p - q = \sqrt{x_1^2 - 4n}$
$\rightarrow p = \frac{x_1 + x_2}{2}, q = \frac{x_1 - x_2}{2}$, polinomiale

Ricavare d direttamente da $\langle n, e \rangle$ sembra costoso quanto fattorizzare n. Come fattorizzare n intero? È un problema difficile ma non più come un tempo: hardware più potente e algoritmi più raffinati, siamo in grado di fattorizzare in tempo subesponenziale $O(2^{\sqrt{b \cdot \log b}})$ con $b = \log_2 n + 1$, mentre il bruteforce richiede $O(n)$ cioè $O(2^b)$

Possiamo fattorizzare semiprimi fino a 768 bit, con l'algoritmo GNFS. Mentre per interi con strutture particolari ci sono algoritmi di fattorizzazione particolarmente efficienti.
La fattorizzazione e il logaritmo discreto non sono problemi NP-hard e si possono risolvere in tempo polinomiale su macchine quantistiche.
Per avere una sicurezza in RSA pari ad un AES 128 bit devo usare un RSA con 3072 bit, per un AES 256 bit ho bisogno di RSA da 15360 bit.

Scegliere p e q molto grandi, per resistere a bruteforce.
Sia $p-1$ che $q-1$ devono avere un fattore primo molto grande, sennò n si fattorizza in fretta.
MCD($p-1$, $q-1$) deve essere piccolo (idealmente 2). Conviene scegliere p e q tali che $(p-1)/2$ e $(q-1)/2$ siano coprimi.
Non bisogna anche mai riutilizzare uno dei due primi per altri moduli, né sceglierli troppo vicini fra loro, sennò n sarà circa p^2 o q^2 e $\sqrt{n}$ sarà vicino ai primi e basterà un bruteforce che cerca i fattori vicino a $\sqrt{n}$

Attacchi

Attacchi con esponenti bassi
Esponenti e e d bassi sono attraenti perché accelerano cifratura e decifrazione. Ovviamente d deve essere scelto sufficientemente grande per evitare bruteforce.
Se m ed e sono così piccoli che $m^e < n$, allora **risulta facile trovare $\sqrt[e]{c}$ poiché $c = m^e$ e non interviene il modulo**.

Attacchi a tempo
Si basano sul tempo di esecuzione dell'algoritmo di decifrazione. L'idea è determinare d analizzando il tempo impiegato per decifrare.
Quando si esegue l'esponenziazione modulare, si esegue una moltiplicazione ad ogni iterazione più un'**ulteriore moltiplicazione modulare per ciascun bit uguale a 1 in** d. Come rimedio, si introduce un ritardo casuale per confondere l'attaccante.

Attacco con e troppo piccolo
Se e utenti scelgono lo stesso e piccolo e tutti ed e ricevono lo stesso messaggio m ottengo
$$\begin{matrix} c_1 = m^e \ mod\ n_1 \\ c_2 = m^e \ mod\ n_2 \\ \vdots \\ c_e = m^e \ mod\ n_e \end{matrix} \quad \forall\, i \text{ con } 1 \le i \le e \text{ e } m < n_i.$$
Poiché $m \cdot m \cdot \ldots \cdot m < n_1 \cdot n_2 \cdot \ldots \cdot n_e = n$ so che $m^e < n$. Ipotizzo n_i coprimi fra loro, per il teorema cinese del resto $\exists$ e si può facilmente calcolare un unico m' tale che $m' < n$ e $m' \equiv m^e \bmod n$
Dato che $m' < n$ e $m^e < n$, $m' \bmod n = m^e \bmod n \Rightarrow m' = m^e \Rightarrow m = \sqrt[e]{m'}$

Attacco con lo stesso valore di n
$\langle e_1, n\rangle$, $\langle e_2, n\rangle$, MCD$(e_1, e_2) = 1$
$\exists\, r, s \in Z \mid e_1 r + e_2 s = 1 = \text{MCD}(e_1, e_2)$ e si trovano con l'algoritmo di Euclide esteso $ax + by = \text{MCD}(a, b)$
Pongo $r < 0, s > 0$, Eve intercetta $\begin{matrix} c_1 = m^{e_1} \ mod\ n \\ c_2 = m^{e_2} \ mod\ n \end{matrix} \Rightarrow m = m^1 =$
$= m^{re_1+se_2} = (m^{e_1} \bmod n)^r \cdot (m^{e_2} \bmod n)^s \bmod n = (c_1^r \cdot c_2^s) \bmod n = ((c_1^{-1})^{-r} \cdot c_2^s) \bmod n$
Per c_1^{-1} calcolo l'inverso di $c_1 \bmod n$. $(c_1^{-1})^{-r}$ e c_2^s con quadrature successive
$\Rightarrow m = (c_1^{-1})^{-r} \cdot (c_2)^s \bmod n$

5.3 Protocollo Diffie-Hellmann

Per esempio, RSA per scambiare la chiave e AES per proteggere la comunicazione.

Alice sceglie una chiave per AES (`k[session]`) e la cifra con la chiave pubblica RSA di Bob. Cifra il messaggio con `k[session]` e invia i due crittogrammi a Bob.
$\langle \text{C}_{RSA}(\texttt{k[session]}, \texttt{k}_{Bob}\texttt{[pub]}), \text{C}_{DES}(m, \texttt{k[session]})\rangle$

Bob decifra il primo crittogramma con $\texttt{k}_{Bob}\texttt{[prv]}$, trova `k[session]` e decifra il secondo crittogramma

L'onere di creare la chiave di sessione è lasciato esclusivamente al mittente, ma la chiave deve essere generate in modo più possibile casuale.

Protocollo Alice e Bob si accordano pubblicamente su un numero primo p molto grande (migliaia di bit) e su un generatore g di $Z_p^* = \{1, 2, \ldots, p-1\} = \{g^k \bmod p \mid 1 \leq k \leq p-1\}$. $\exists\, g$ perché p è primo.
Alice e Bob possono anche scegliere di usare una coppia $\langle p, g \rangle$ già disponibili. Non è richiesto che questa coppia sia mantenuta segreta.
Adesso il protocollo parte:

Alice	$\langle g, p \rangle$	Bob
Sceglie a caso $1 < x < p-1$ intero positivo, e calcola $A = g^x \bmod p$ Lo manda a Bob	$\rightarrow A \rightarrow$ $\leftarrow B \leftarrow$	Sceglie a caso $1 < y < p-1$ intero positivo, e calcola $B = g^y \bmod p$ Lo manda a Alice
Riceve B da Bob e calcola `k[session]` $= B^x \bmod p = g^{xy} \bmod p$		Riceve A da Alice e calcola `k[session]` $= A^y \bmod p = g^{xy} \bmod p$

Attacchi

Attacchi passivi, il protocollo è **resistente**.
Il crittoanalista conosce p, g, A, B. Per calcolare `k[session]` deve trovare x o y, ma $A = g^x \bmod p$ e $B = g^y \bmod p$.
Trovare x, y conoscendo A, B significa risolvere il logaritmo discreto che è un **problema difficile** quanto la fattorizzazione.

Attacchi attivi, il protocollo è **vulnerabile** ad attacchi del tipo MitM.

Alice	Eve	Bob
$A = g^x \bmod p \rightarrow$		$\leftarrow B = g^y \bmod p$
	Sottrae A, B dal canale e il sostituisce con un suo $E = g^z \bmod p$ con $1 < z < p-1$ $\leftarrow$La manda a entrambi$\rightarrow$	
`k[session]` $= E^x \bmod p = g^{xz} \bmod p$		`k[session]` $= E^y \bmod p = g^{yz} \bmod p$
	Conosce sia $K_A = A^z \bmod p$ sia $K_B = B^z \bmod p$ $K_A \neq K_B$	

5.4 Cifrario di El Gamal

Si basa sul logaritmo discreto come funzione one-way-trapdoor.
Alice $\rightarrow m \rightarrow$ Bob

Bob sceglie p numero primo grande e g generatore per Z_g^*
Sceglie $2 \leq x \leq p-2$ come **chiave privata**, `k[prv]` $= \langle x \rangle$
Calcola $y = g^x \bmod p$ e **pubblica** `k[pub]` $= \langle p, g, y \rangle$

Alice $0 \leq m < p$, si procura `k[pub]` $= \langle p, g, y \rangle$
Sceglie a caso $2 \leq r \leq p-2$ **segreto** e calcola $c = g^r \bmod p$, cioè un numero casuale $\in Z_p^*$ perché g generatore
Calcola $d = m \cdot (y)^r \bmod p$ e invia a Bob $\langle c, d \rangle$, coppia di crittogrammi: il primo contiene protetta l'informazione su r, e d contiene il messaggio m

Bob riceve $\langle c, d \rangle$ e decifra: $m = \frac{d}{c^x} \bmod p = d \cdot (c^x)^{-1} \bmod p$ (dove la frazione indica la moltiplicazione per l'inverso)

Correttezza $m = \frac{d}{c^x} \bmod p = \frac{y^r \cdot m}{c^x} \bmod p = \frac{(g^{xr}) \cdot m}{(g^r)^x} \bmod p = m \bmod p = m$ perché $m < p$

Attacchi Eve conosce p, g, y, c, d, cioè tutto tranne x e r.
Se conosce x calcola $m = \frac{d}{c^x} \bmod p$
Se conosce r calcola $m = \frac{d}{y^r} \bmod p = \frac{m \cdot y^r}{y^r} \bmod p = m$

Capitolo 6

Elliptic Curve Cryptography

Equivalenze di costo degli attacchi

AES	RSA,DH, El Gamal	ECC
128 bit	3072 bit (di n, p)	256 bit
256 bit	15360 bit	512 bit

Cosa sono Curve algebriche descritte da equazioni simili a quelle usate per il calcolo degli archi delle ellissi.
1985 da Miller e Koblitz, propongono di prendere la cifratura a chiave pubblica e modificarli sostituendo solo le operazioni: invece che algebra modulare usare punti delle curve ellittiche.

Definizione generale Campo K, insieme di punti $(x, y) \in K^2 \,|\, y^2 + axy + by = x^3 + cx^2 + dx + e$ con $a, b, c, d, e \in K$
Se la caratteristica di $K \neq 2, 3$ allora esiste la forma normale di Weierstrass $y^2 = x^3 + ax + b$ con $a, b \in K$

La **caratteristica** di un campo è il numero di volte che io devo sommare l'elemento neutro moltiplicativo (1) per ottere l'elemento neutro additivo (0). Ad esempio, negli interi modulo p primo, la caratteristica è p: $p \cdot 1 \bmod p = 0$
In R la caratteristica è 0, non esiste.

Con caratteristica $\neq 2, 3$, la curva ellittica può essere descritta da

$$E_K(a, b) = \{(x, y) \in K^2 \mid y^2 = x^3 + ax + b\} \cup \{O\}$$

Ora prenderemo $K = R$, cioè $E(a, b) = \{(x, y) \in R^2 \mid y^2 = x^3 + ax + b\} \cup \{O\}$.
La richiesta è che $x^3 + ax + b$ **non abbia radici multiple**, cioè **assumiamo che** $4a^3 + 27b^2 \neq 0$ (discriminante).
Questo garantisce l'esistenza della tangente in ogni punto della curva ellittica.

Simmetria orizzontale Dato $P = (x, y) \in E(a, b)$, quindi x, y soddisfano $y^2 = x^3 + ax + b \Rightarrow -P = (x, -y) \in E(a, b)$ infatti $(-y)^2 = y^2 = x^3 + ax + b$

Per il punto all'infinito si pone O = -O

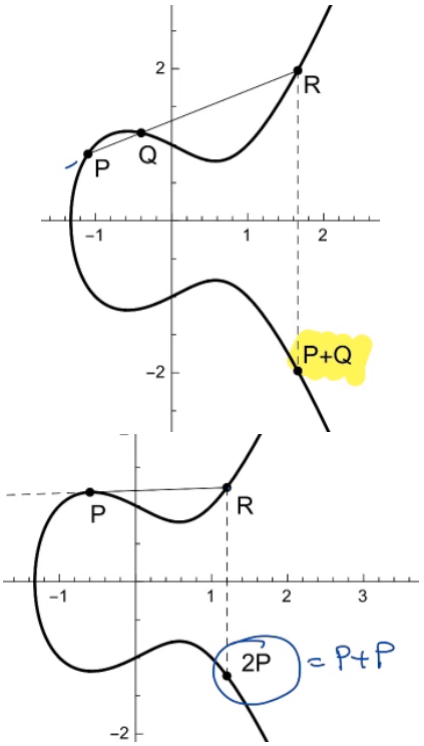
Idea Ogni retta interseca una curva in al più 3 punti:

3 punti di intersezione (tre soluzioni reali della cubica)

1 punto di intersezione (una soluzione reale e 2 complesse coniugate

Se una retta interseca $E(a, b)$ in 2 punti $\Rightarrow$ la interseca anche in un terzo punto: si usa per definire l'operazione "somma".
$P, Q, R \in E(a, b)$, se P, Q ed R sono su una retta si pone $P + Q + R = O \Rightarrow P + Q = -R$

Metodo per sommare due punti $P, Q \in E(a,b)$ con $Q \neq \pm P$
Si considera $\overline{PQ}$, la retta per P e Q, e si calcola il terzo punto di intersezione tra $E(a,b)$ e $\overline{PQ}$ chiamandolo R
Si pone $P + Q = -R$ $\begin{cases} R \in E(a,b) \\ -R \in E(a,b) \end{cases}$



Inverso Dato $Q = -P$, ho $P + (-P) = -O = O$
$P = (x, y) \Rightarrow -P = (x, -y)$

Somma con sé stesso Se $Q = P$, dal punto di vista algebrico ho due radici reali uguali. La retta considerata è la tangente alla cura nel punto P, sempre ben definita per costruzione della curva in quanto $4a^3 + 27b^2 \neq 0$.
Si prende poi l'opposto del punto di intersezione tra la tangente in P e la curva $E(a,b)$ (può essere O)

Proprietà della somma

Chiusura $\forall\, P, Q \in E(a,b)$ ho $P + Q \in E(a,b)$

Elemento neutro $\forall\, P \in E(a,b)$ ho $P + O = O + P = P$

Inverso $\forall\, P \in E(a,b)\ \exists!\, Q \in E(a,b) \mid P + Q = O = Q + P$ e $Q = -P$
$(P = (x,y) \Rightarrow -P = (x,-y))$

Proprietà commutativa $\forall\, P, Q \in E(a,b)$
ho $P + Q = Q + P$

Proprietà associativa $\forall\, P, Q, R \in E(a,b)$
ho $(P + Q) + R = P + (Q + R)$

Formulazione algebrica $P(x_p, y_p)$, $Q = (x_q, y_q)$, $S = P + Q$

$$Q \neq \pm P \;\; S = (x_s, y_s) \text{ con } \begin{cases} x_s = \lambda^2 - x_p - x_q \\ y_s = -y_p + \lambda(x_p - x_s) \\ \lambda = \frac{y_q - y_p}{x_q - x_p} \end{cases}$$

$Q = P$ Stesse formule, ma $\lambda = \frac{3x_p^2 + a}{2y_p}$. Se $y_p = 0$ allora $P + P = O$

$Q = -P$ $P + Q = O$

6.0.1 Curve Ellittiche su campi finiti

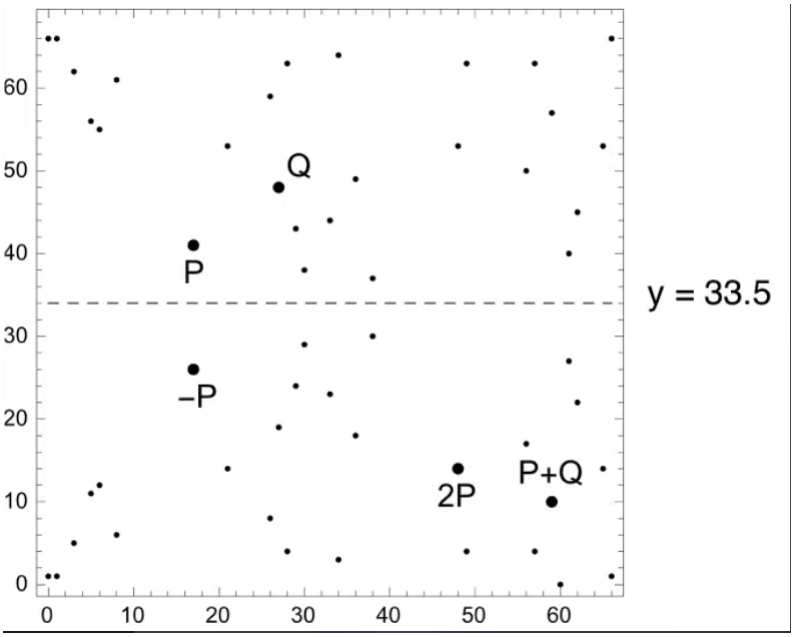
Due famiglie

Curve Prime $K = Z_p$ con p primo
Hanno caratteristica p, e sono quelle che considereremo. Imporremo $p > 3$

Curve Binarie $K = GR(2^m)$ con $m \in N$
Ha caratteristica pari a 2, quindi non si può usare la forma di Weierstrass

$$E_p(a,b) = \{(x,y) \in Z_p^2 \mid y^2 \bmod p = x^3 + ax + b \bmod p \cup \{O\}$$

Quindi in $[0, p-1]$ e ho simmetria...



$p = 67, a = -1, b = 1$

$P = (x, y) \in E_p(a, b) \Rightarrow -P = (x, p - y) \in E_p(a, b)$

Ordine # punti della curva. Non c'è una formula, dipende da com'è fatta la cubica $y^2 = x^3 + ax + b$ con $x \in Z_p$, quindi p valori per x. Ogni valore possibile di x da origine a 2 punti (y e $-y$ per il punto opposto).
Ci si aspettano quindi circa $2p$ punti per i valori di x, $+1$ per $\{O\}$, ma non è esattamente $2p + 1$ perché non tutti i numeri hanno una radice nel campo. $\frac{p-1}{2}$ sono residui quadratici.
Quindi indicativamente il numero di punti è un numero molto vicino a p. Il **teorema di Hasse** che dice dato $N =$ ordine di una certa curva $E_p(a, b)$, allora $|N - (p + 1)| \leq 2\sqrt{p}$

Esempio $y^2 \equiv x^3 + 4x + 4 \mod 5$

y	y^2
0	0
1	1
2	4
3	4
4	1

1 e 4 sono residui quadratici, ammettono una radice nel campo.

$x = 0 \;\; y^2 = 4 \Rightarrow (0, 2), (0, 3) \in E_5(4, 4)$

$x = 1 \;\; y^2 = 4 \Rightarrow (1, 2), (1, 3) \in E_5(4, 4)$

$x = 2 \;\; y^2 = 0 \Rightarrow (2, 0) \in E_5(4, 4)$

$x = 3 \;\; y^2 = 3 \Rightarrow$ nessuna soluzione

$x = 4 \;\; y^2 = 4 \Rightarrow (4, 2), (4, 3) \in E_5(4, 4)$

$\Rightarrow$ Ordine = 8, compreso O

Algebra modulare	Curve ellittiche
Moltiplicazione	Somma di punti
Fissato K: elevamento a potenza (one-way)	**Moltiplicazione scalare** di un punto P della curva per un intero k
$y^k = y \cdot y \cdot \ldots \cdot y$, k volte	$kP = P + P + \ldots + P$, k volte
Costo polinomiale	Costo polinomiale

$Q = kP$ in tempo polinomiale con il metodo dei **raddoppi ripetuti**. Il problema inverso, trovare k da Q e P è il logaritmo discreto sulle curve ellittiche, difficile.

6.0.2 Funzione one-way

Moltiplicazione scalare Verifichiamo che è one-way:

$Q = kP$ dati k e P è facile, con $\Theta(\log k)$ operazioni: **raddoppi ripetuti**

$k = \sum_{i=0}^{t} k_i \cdot 2^i$ con $(k_t k_{t-1} \dots k_1 k_0) = k$ e $t + 1 = \lfloor \log_2 k \rfloor + 1$ # bit

Si calcolano i punti $2P, 4P, \dots, 2^t P$, ciascuno come raddoppio del punto precedente: $t = \Theta(\log k)$ raddoppi

Si calcola $Q = \sum_{i \mid k_i = 1} (2^i P)$, $O(t) = O(\log k)$ somme

Avendo un algoritmo polinomiale.

L'**operazione inversa** della moltiplicazione scalare: dati $P, Q \in E_p(a, b)$ trovare il più piccolo $k \mid Q = kP$

$$k = \log_p Q$$

È il **problema del logaritmo discreto per le curve ellittiche**, difficile per cui non conosciamo algoritmi polinomiali e nemmeno subesponenziali, soltanto forza bruta.

6.0.3 Protocollo DH su curve ellittiche

Preparazione Alice e Bob scelgono una curva ellittica appropriata e un punto B della curva di ordine molto grande. (L'**ordine** n **di un punto B** è **il più piccolo intero** $n \mid nB = O$)
Il punto B gioca un po' il ruolo del generatore g del DH standard. Curva e punto B sono **pubblici**.

Alice	*Canale*	Bob
	$E_p(a, b)$ e $B \in E_p(a, b)$ pubblici con B di ordine n	
Estrae $n_A < n$ casuale **chiave privata**		**Estrae** $n_B < n$ casuale **chiave privata**
Calcola la **chiave pubblica** $P_A = n_A \cdot B$		**Calcola** la **chiave pubblica** $P_B = n_B \cdot B$
	In chiaro $\rightarrow P_A \rightarrow$ $\leftarrow P_B \leftarrow$	
Riceve P_B		**Riceve** P_A
Calcola $S = n_A \cdot P_B = n_A \cdot n_B \cdot B$ `k[sessione]` $= x_S \bmod 2^{256}$		**Calcola** $S = n_B \cdot P_B = n_B \cdot n_A \cdot B$ `k[sessione]` $= x_S \bmod 2^{256}$

Crittoanalista Eve conosce la curva, il punto B e intercetta P_A e P_B ma per calcolare S deve trovare uno tra

$n_A \mid n_A \cdot B = P_A$

$n_B \mid n_B \cdot B = P_B$

quindi deve **risolvere il problema del logaritmo discreto su curve ellittiche**.
Il protocollo è però soggetto ad attacchi attivi MitM.

6.1 Scambio di messaggi cifrati

Scambio di messaggi Si trasforma il messaggio m in P_m punto di una curva ellittica prima $E_p(a, b)$
$y^2 \equiv x^3 + ax + b \bmod p$, e possiamo pensare di sostituire m a x, il problema è che il risultato potrebbe non essere un residuo quadratico. $P(m^3 + am + b$ sia un residuo quadratico$) \simeq \frac{1}{2}$
Residuo quadratico: ha radice nel campo.
Tecnica con possibilità di successo così bassa non va bene, serve una tecnica migliore. Ci sono algoritmi randomizzati, **non sono noti algoritmi deterministici polinomiali**.

6.1.1 Algoritmo di Koblitz

Algoritmo polinomiale randomizzato che $m < p \mapsto P_m \in E_p(a,b)$
Si sceglie h intero $\mid (m+1)h < p$
$x = m \cdot h + i$ con $0 \leq i < h \Rightarrow h$ tentativi per inserire $m \cdot h + i$ come ascissa e vedere se è residuo quadratico o meno.

```
KOBLITZ(m,h,a,b,p){
        for(i = 0; i < h; i++){
                x = mh + i;
                z = (x^3 + ax + b) mod p;
                if (z residuo quadratico) { //Costo polinomiale
                        y = sqrt(z);
                        return Pm = (x,y);
                }
        }
        return "failure";
}
```

Probabilità di fallimento $\simeq \left(\frac{1}{2}\right)^h \Rightarrow$ probabilità di successo $\simeq 1 - \left(\frac{1}{2}\right)^h$
Per risalire a m da x, $\lfloor \frac{x}{h} \rfloor = \lfloor \frac{mh+i}{h} \rfloor = \lfloor m + \frac{i}{h} \rfloor = m$ perché $\frac{i}{h} < 1$

6.1.2 Scambio di messaggi

Viene fissata la curva $E_p(a,b)$ e il punto B di ordine elevato n, $B \in E_p(a,b)$
Ogni utente genera `k[pub]`, `k[priv]`

> `k[priv]` è $n_u < n$
>
> `k[pub]` è $P_u = n_u B$

Per Koblitz, è comune anche h.
Alice mittente vuole mandare un messaggio a Bob destinatario:

> Alice converte m messaggio in $P_m \in E_p(a,b)$, ad esempio con l'algoritmo di Koblitz
> Sceglie un intero casuale r e calcola $V = rB$ (V è quindi un punto a caso sulla curva $E_p(a,b)$)
> Calcola $W = P_m + rP_{Bob}$ e dato che r è casuale, rP_{Bob} è scelto a caso su $E_p(a,b)$ (P_{Bob} è la chiave pubblica di Bob)
> Invia a Bob $\langle V, W \rangle$
>
> Bob riceve $\langle V, W \rangle$
> Decifra $W - n_{Bob}V = (P_m + rP_{Bob}) - n_{Bob}V = P_m + rn_{Bob}B - n_{Bob}rB = P_m$
> Ricava $m = \lfloor \frac{x}{h} \rfloor$

La sicurezza è basata sulla difficoltà del logaritmo discreto su curve ellittiche.

Crittoanalista Potrebbe decifrare in questi modi, dallo stesso costo:

> Se trova r, decifra facendo $W - rP_{Bob} = (P_m + rP_{Bob}) - rP_{Bob}$
> Ma per trovare r da V, B, $V = rB$ bisogna risolvere il logaritmo discreto
> Consente di decifrare un solo crittogramma (r è one time).
>
> Altrimenti deve trovare n_{Bob} da P_{Bob}, B
> $P_{Bob} = n_{Bob}B$, logaritmo discreto
> Consentirebbe di decifrare tutti i crittogrammi successivi.

Sicurezza Per attaccare RSA, DH, El Gamal (algebra modulare) $O(2^{\sqrt{b \cdot \log b}})$, con $b = \#$ bit del modulo (basati su **index calculus**, sfruttano la proprietà algebrica dei campi Z_p)
Per attaccare su curve ellittiche (logaritmo discreto su curva ellittica) $O(2^{\frac{b}{2}})$, con $b = \#$ bit dell'ordine di B

Capitolo 7

Identificazione, Autenticazione e Firma Digitale

Identificazione Un sistema di elaborazione, isolato o in rete, deve **essere in grado di accertare l'identità di un utente** che richiede di accedere ai suoi servizi.

Autenticazione Il destinatario di un messaggio deve **essere in grado di accertare l'identità del mittente e l'integrità del crittogramma ricevuto**

Firma Digitale

1. Il **mittente non deve poter negare di aver inviato** un messaggio m
2. Il **destinatario deve essere in grado di autenticare** il messaggio
3. Il **destinatario non deve poter sostenere che** $m' \neq m$ è il messaggio inviato dal mittente

Tutto deve essere verificabile da una terza parte.

Relazioni tra le funzionalità Non sono indipendenti, ma **ciascuna estende le precedenti**:

L'autenticazione di un messaggio garantisce l'identificazione del mittente

L'apposizione della firma garantisce l'autenticazione del messaggio

Ogni funzionalità e utilizzata per contrastare gli attacchi attivi. Esistono **realizzazioni algoritmiche basate su cifrari asimmetrici e simmetrici**.

7.1 Funzioni hash

Funzione hash è una funzione $f : X \to Y$ tale che $n = |X| >> m = |Y|$
$\exists\, X_1, X_2, \ldots, X_m \subseteq X$ disgiunti tali che $X = X_1 \cup X_2 \cup \ldots \cup X_m \wedge \forall\, i, \forall\, x \in X_i$ ho $f(x) = y$
Comodo per gestire la rappresentazione compatta dei dati.

Una **buona funzione hash** assicura che:

I sottoinsiemi $X_1, \ldots, X_m$ hanno circa la stessa cardinalità.
Due elementi estratti a caso da X hanno probabilità circa $\frac{1}{m}$ di avere la stessa immagine in Y

Elementi di X molto "simili" tra loro appartengono a due sottoinsiemi diversi.
Ad esempio se X è un insieme di interi, due elementi con valori prossimi devono avere immagini diverse.

Gestione delle collisioni.
L'algoritmo che impiega la funzione hash dovrà affrontare la situazione in cui più elementi di X hanno la stessa immagine in Y

7.1.1 Funzioni hash one-way

Se la funzione è usata in crittografia, deve **soddisfare le seguenti proprietà**:

1. $\forall\, x \in X$ è **computazionalmente facile** calcolare $y = f(x)$
   (Ricordiamo che computazionalmente facile significa tempo polinomiale nella dimensione di x)
2. **Proprietà one-way**: per la maggior parte degli $y \in Y$ è **computazionalmente difficile** determinare $x \in X \mid f(x) = y$ cioè $x = f^{-1}(y)$
   (Ricordiamo che computazionalmente difficile significa tempo esponenziale)
3. **Proprietà claw-free**: è **computazionalmente difficile** determinare $x_1, x_2 \mid f(x_1) = f(x_2)$, cioè due elementi diversi che collidono sulla stessa immagine hash.

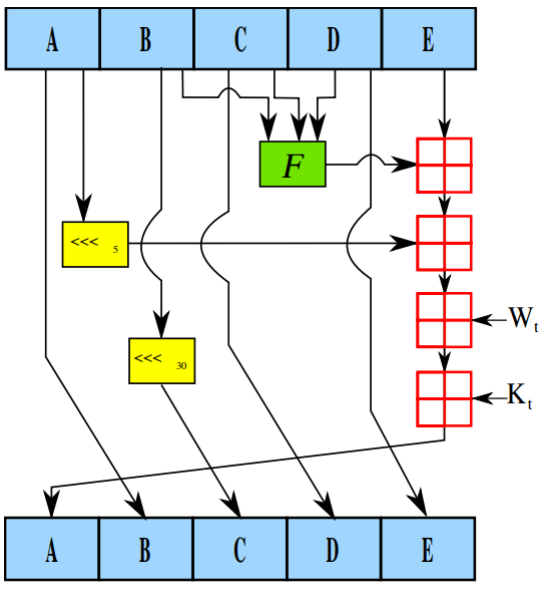
Funzioni hash usate in crittografia

MD5 Message Digest, versione 5
Famiglia di algoritmi, vennero pubblicati MD2 e MD4 ma avevano debolezze. MD5 proposto da Ron Rivest nel 1992.
Input: sequenza S di 512 bit
Output: **immagine di 128 bit**. La sequenza è *digerita*, riducendone la lunghezza ad un quarto.
Dal 2004 è considerato severamente compromesso, poiché è stato dimostrato che non resiste alle collisioni ed ha altre debolezze serie.

RIPEMD-160
Versione "matura" delle funzioni MD. Produce immagini di 160 bit ed è esente dai difetti di MD5.

SHA Secure Hash Algorithm
Viene adottato quando la proprietà claw-free è cruciale per la sicurezza del sistema.
Opera su sequenze lunghe fino a 2^{64} bit e produce immagini di 160 bit.
Crittograficamente sicura: soddisfa i requisiti delle funzioni hash one-wat e genere immagini molto diverse per sequenze molto simili.

SHA-1, opera su sequenze fino a $2^{64} - 1$ bit e produce immagini di 160 bit. Molto usato nei protocolli crittografici anche se non è più certificato come standard. Tutte le altre funzioni hanno una struttura simile a SHA-1.
Opera su blocchi da 160 bit, contenuti un buffer da 5 registri di 32 bit ciascuno, in cui sono caricati inizialmente dei valori pubblici. Il messaggio m viene concatenato con una sequenza di padding che ne rende la lunghezza multipla di 512 bit. Il contenuto dei registri varia nel corso dei cicli successivi, in cui questi valori si combinano tra loro e con blocchi di 32 bit provenienti da m.
A fine procedimento, i registri contengono SHA-1(m)



Un'iterazione all'interno di SHA-1.
A,B,C,D,E sono i registri a 32 bit.
F è una funzione non lineare che varia.
$<<<_n$ denota una rotazione del bit di sinistra di n posti.
n varia per ogni operazione.
$\boxplus$ denota l'addizione in modulo 2^{32}
K_t è una costante
W_t blocco di 32 bit ottenuto tagliando e rimescolando i blocchi del messaggio

Il contenuto dei registri, che partono con valori pubblici, varia nel corso dei cicli combinandosi tra loro e con blocchi da 32 bit provenienti dal messaggio W, nonché con alcuni parametri relativi al ciclo.
Alla fine del procedimento, quando è stato letto l'intero messaggio, i registri conterranno SHA-1(W)

Z

7.2 Identificazione su canali sicuri

Esempio L'accesso di un utente alla propria casella di posta elettronica, o a file personali memorizzati su un sistema con accesso riservato ai membri della sua organizzazione.
L'utente inizia il collegamento inviando in chiaro login e password. Se il canale è protetto in lettura e scrittura, un attacco può essere sferrato solo da un utente locale al sistema: ad esempio l'amministratore che ha accesso a tutti i file memorizzati (o un hacker). Il meccanismo di identificazione prevede una **cifratura delle password realizzata con funzioni hash one-way**.

7.2.1 Cifratura delle password nei sistemi UNIX

Quando un utente U fornisce per la prima volta la propria password P, il **sistema associa ad U due sequenze binarie** che memorizza nel file delle password al posto di P:

S, seme, prodotta da un generatore pseudocasuale

$Q = h(PS)$ con h funzione hash one-way e PS concatenazione semplice della password e del seme.

Per proteggere da attacchi interni è quindi necessario hashare le password: il seme è utile per non hashare password uguali di utenti diversi con stesso hash.

Ad ogni successiva connessione di U, il sistema:

Recupera S dal file delle password

Concatena S con la password fornita da U

Calcola l'immagine one-way della nuova sequenza: $h(PS)$

Se $h(PS) = Q$, l'identificazione ha successo.

Un accesso illecito al file delle password, quindi, non fornisce informazioni interessanti: è **computazionalmente difficile ricavare la password originale dalla sua immagine one-way**.

7.2.2 Protezione del canale

Se il canale è insicuro, la password può essere intercettata durante la trasmissione in chiaro. Il **sistema non dovrebbe mai maneggiare direttamente la password, ma una sua immagine inattaccabile**.

Canale insicuro: identificazione $\langle e, n\rangle, \langle d\rangle$: chiave pubblica e privata di un utente U che richiede l'accesso ai servizi offerti dal sistema S

1. S genera un numero casuale $r < n$ e lo invia in chiaro a U
2. U calcola $f = r^d \bmod n$, **firma di U su** r, con la sua chiave privata e lo spedisce ad S
3. S verifica la correttezza del valore ricevuto calcolando e verificando se $f^e \bmod n = r$
   Se ciò avviene, l'identificazione ha successo.

Le operazioni di cifratura e decifrazione sono invertite rispetto all'impiego standard dell'RSA, ma questo è possibile perché le due operazioni sono commutative:

$$(x^e \ mod \ n)^d \ mod \ n = (x^d \ mod \ n)^e \ mod \ n = x$$

f può essere generata solo da U che possiede $\langle d\rangle$. Se il passo 3 va a buon fine, il sistema ha la garanzia che l'utente che ha richiesto l'identificazione sia effettivamente U, anche se il canale è insicuro. **Problema**: S chiede ad U di applicare la chiave privata ad una sequenza r che S stesso ha generato, potrebbe essere stata **scelta di proposito per ricavare qualche informazioni sulla chiave privata di U**
Protocollo alternativo, a "**conoscenza zero**": impedisce che da una comunicazione si possa estrarre più di quanto sia nelle intenzioni del comunicatore.

Canale insicuro: autenticazione DEST deve **autenticare il messaggio**, accertando l'identità di MITT e l'integrità di m. MITT e DEST concordano una chiave segreta k.
MITT:

> Allega al messaggio un **MAC** (Message Authentication Code) $A(m, k)$, allo scopo di garantire la provenienza e l'integrità del messaggio.
>
> Spedisce la coppia $\langle m, A(m, k)\rangle$ in chiaro
> Oppure cifra m e spedisce $\langle C(m, k'), A(m, k)\rangle$ con C funzione di cifratura e k' chiave pubblica o segreta del cifrario scelto.

DEST:

> Entra in possesso di m, eventualmente dopo averlo decifrato
>
> Essendo a conoscenza di A e k, calcola $A(m, k)$
>
> Confronta il valore ottenuto con quello ricevuto da MITT per verificare che il MAC ricevuto corrisponda al messaggio a cui risulta allegato.
> Se la verifica ha successso, il messaggio è autenticato. Altrimenti, DEST scarta il messaggio.

Il **MAC** è un'immagine breve del messaggio, che può essere stata generata solo da un mittente conosciuto dal destinatario, previ opportuni accordi.
Ci sono varie proposte, basate su cifrari asimmetrici, simmetrici e funzioni hash one-way.

MAC con funzioni hash one-way $A(m, k) = h(mk)$ con h funzione hash one-way.
Risulta computazionalmente difficile per un crittoanalista scoprire la chiave segreta k. h è nota a tutti e m può viaggiare in chiaro o essere scoperto per altra via, ma k viaggia all'interno del MAC. Per recuperare k bisognerebbe invertire h.
Il crittoanalista non può sostituire facilmente m con un altro messaggio m', dovrebbe allegare alla comunicazione di m' il MAC $A(m', k) = h(m'k)$ che può produrre solo conoscendo k.

CBC+MAC Usando un cifrario a blocchi in modalità CBC (Cipher Block Chaining), si può usare il blocco finale del crittogramma come MAC. Il blocco finale, infatti, è funzione dell'intero messaggio.

7.2.3 Firma digitale

Una firma manuale ha i seguenti **requisiti**:

1. **Autentica e non falsificabile**
   Prova che chi l'ha prodotta è chi ha sottoscritto il documento
2. **Non è riutilizzabile**
   Legata strettamente al documento su cui è stata apposta
3. **Il documento firmato non è alterabile**
   Chi ha prodotto la firma è sicuro che questa si riferirà solo al documento sottoscritto nella sua forma originale
4. **Non può essere ripudiata da chi l'ha apposta**
   Costituisce prova legale di un accordo o dichiarazione

La **firma digitale** non può consistere semplicemente nella digitalizzazione del documento originale firmato manualmente. Un crittoanalista potrebbe "tagliare" dal documento digitale la parte contenente la firma e "copiarla" su un altro documento.
Deve avere una forma dipendente dal documento su cui è apposta, per essere inscindibile da questo. Per progettare firma digitali si possono usare sia cifrari simmetrici che asimmetrici.

Protocollo 1 Messaggio m in chiaro e firmato.

> U utente, $\texttt{k}_U\texttt{[priv]}$ e $\texttt{k}_U\texttt{[pub]}$ chiavi di U
>
> C, D funzioni di cifratura e decifrazione di un cifrario asimmetrico
>
> **Firma** U genera la firma $f = D(m, \texttt{k}_U\texttt{[priv]})$ per m e spedisce all'utente V la tripla $\langle U, m, f\rangle$

Verifica V riceve $\langle U, m, f\rangle$ e verifica l'autenticità della firma f controllando che $C(f, \texttt{k}_U\texttt{[pub]}) = m$
L'indicazione del mittente U consente a V di selezionare la chiave pubblica $\texttt{k}_U\texttt{[pub]}$ da utilizzare nel calcolo.
I processi di firma e verifica impiegano le funzioni C, D in ordine inverso a quello standard, quindi devono essere commutative $C(D(m)) = D(C(m)) = m$

Posso sostituire m con c crittogramma? La verifica richiede la conoscenza della chiave pubblica e restituisce m, quindi non avrebbe senso sostituire m con c.
Il protocollo soddisfa i requisiti della firma manuale:

f è autentica e non falsificabile (1)
$\texttt{k}_U\texttt{[priv]}$ è nota solo a U e per falsificare la firma occorre conoscere $\texttt{k}_U\texttt{[priv]}$ ma D è one-way

Il documento firmato $\langle U, m, f\rangle$ non può essere alterato se non da U, pena la non consistenza fra m e f (3)

Poiché solo U può aver prodotto f, U non può ripudiare la firma (4)

La firma f non è riutilizzabile su un altro documento $m' \neq m$ poiché è immagine di m (2)

Definito per un particolare utente U ma non per un particolare destinatario, quindi **chiunque può convincersi dell'autenticità della firma** facendo uso solo della chiave pubblica di U. Si tratta di uno **schema di principio**: comporta lo scambio di un messaggio di lunghezza doppia dell'originale, poiché la dimensione della firma è paragonabile alla dimensione del messaggio, ed **il messaggio non può essere cifrato** perché è **ricavabile pubblicamente** dalla firma attraverso la verifica di questa.

Protocollo 2 Messaggio m cifrato e firmato.
Firma e cifratura:

U genera la firma $f = D(m, \texttt{k}_U\texttt{[priv]})$ per m, calcola il crittogramma firmato $c = C(f, \texttt{k}_V\texttt{[pub]})$ con la chiave pubblica del destinatario $V \rightarrow$ si incapsula la firma nel documento cifrato.

Spedisce $\langle U, c\rangle$ a V

Decifrazione e verifica:

V riceve $\langle U, c\rangle$ e decifra il crittogramma $D(c, \texttt{k}_V\texttt{[priv]}) = f$

Cifra tale valore con la chiave pubblica di U ottenendo $C(f, \texttt{k}_U\texttt{[pub]}) = C(D(m, \texttt{k}_U\texttt{[priv]}), \texttt{k}_U\texttt{[pub]}) = m$

V ricostruisce m e se è significativo attesta l'identità di U
Un utente diverso da U ha probabilità praticamente nulla di generare un crittogramma di significato accettabile se cifrato con la chiave pubblica di U

Algoritmo Cifrario RSA con

$\langle d_U\rangle, \langle e_U, n_U\rangle$ chiavi di U

$\langle d_V\rangle, \langle e_V, n_V\rangle$ chiavi di V

Utente U

Genera la firma del messaggio m: $f = m^{d_U} \bmod n_U$

Cifra f con la chiave pubblica di V: $c = f^{e_V} \bmod n_V$

Spedisce a V $\langle U, c\rangle$

Utente V

Riceve $\langle U, c\rangle$ e decifra c: $c^{d_V} \bmod n_V = f$

Decifra f con la chiave pubblica di U: $F^{e_U} \bmod n_U = m$

Se m è significativo, conclude che è autentico

Perché sia corretto, è necessario che $n_U \leq n_V$ perché risulti $f < n_V$ e f possa essere cifrata correttamente e spedita a V. Questo impedirebbe a V di inviare messaggi firmati e cifrati a U.
Ogni utente stabilisce chiavi distinte per la firma e la cifratura. Si fissa pubblicamente H molto grande, es $H = 2^{1024}$, e le chiavi di firma $< H$ mentre quelle di cifratura $> H$.
Questo valore alto assicura di scegliere chiavi sufficientemente grandi. Il meccanismo di firma si presta però a diversi attacchi, basati sulla possibilità del crittoanalista di procurarsi la firma di un utente su messaggi apparentemente privi di senso.

Attacco 1 Supponiamo che un utente U invii una risposta automatica (ACK) a MITT ogni volta che riceve un messaggio m. Poniamo che l'ACK sia il crittogramma della firma di U su m.
Un crittoanalista attivo X può decifrare i messaggi inviati a U:

> X intercetta il crittogramma c firmato inviato da V a U, lo rimuove dal canale e lo rispedisce a U, facendogli credere che c sia stato inviato da lui.
>
> U spedisce automaticamente a X un ACK
>
> X usa l'ACK per risalire al messaggio originale m applicando le funzioni del cifrario con le chiavi pubbliche di V e U

1. V invia il crittogramma c a U
   $c = C(f, k_U[pub])$
   $f = D(m, k_V[priv])$
2. Il crittoanalista X intercetta c, lo rimuove dal canale e lo rispedisce a U (U crede che c arrivi da X)
3. U decifra c
   $f = D(c, k_U[priv])$
   e verifica la firma con la chiave pubblica di X ottenendo il messaggio
   $m' = C(f, k_X[pub])$
4. $m' \neq m$ è un messaggio privo di senso, ma il sistema manda ACK c' a X
   $f' = D(m', k_U[priv])$
   $c' = C(f', k_X[pub])$
5. X usa c' per risalire a m
   (a) Decifra c' e trova f'
   $D(c', k_X[priv]) = D(C(f', k_X[pub]), k_X[priv]) = f'$
   (b) Verifica f' e trova m'
   $C(f', k_U[pub]) = C(D(m', k_U[priv]), k_U[pub]) = m'$
   (c) Da m' ricava f
   $D(m', k_X[priv]) = D(C(f, k_X[pub]), k_X[priv]) = f$
   (d) Verifica f con la chiave pubblica di V e trova m
   $C(f, k_V[pub]) = C(D(m, k_V[priv]), k_V[pub]) = m$

Per evitare questo tipo di attacco, U deve bloccare l'ACK automatico che deve essere inviato solo dopo un esame preventivo di m e scartando i messaggi che non si desidera firmare.

Protocollo resistente agli attacchi: si **evita la firma diretta del messaggio**, si appone la firma digitale su una immagine del messaggio ottenuta con una funzione hash one-way e pubblica (MD5, SHA...).
Non si firma il messaggio ma l'hash del messaggio, il che **rompe tutti i giochi algebrici** che potevano essere eseguiti.

Protocollo 3 Messaggio m cifrato e firmato in hash.
Firma e cifratura

> Il mittente U calcola $h(m)$ e genera la firma $f = D(h(m), k_u[priv])$
>
> Calcola separatamente il crittogramma $c = C(m, k_V[pub])$
>
> Spedisce a V la tripla $\langle U, c, f \rangle$

Decifrazione e verifica

> V riceve $\langle U, c, f \rangle$
>
> Decifra il crittogramma c: $m = D(c, k_V[priv])$
>
> Calcola separatamente $h(m)$ e $C(f, k_U[pub])$
>
> Se $h(m) = C(f, k_U[pub])$, allora conclude che il messaggio è autentico.

Si scambiano una maggiore quantità di dati, ma l'incremento è trascurabile. La firma può essere calcolata più velocemente.

7.3 Attacchi man-in-the-middle

Debolezza dei protocolli Le chiavi di cifratura sono pubbliche e non richiedono un incontro diretto tra gli utenti per il loro scambio.
Un crittoanalista attivo può intromettersi proprio in questa fase iniziale del protocollo, pregiudicando il suo corretto svolgimento. Un attacco attivo chiamato **man-in-the-middle**:

Il crittoanalista si intromette nella comunicazione tra U e V

Si comporta come V agli occhi di U

Si comporta come U agli occhi di V

Intercetta e blocca le comunicazioni tra U e V

Attacchi MitM sulle chiavi pubbliche Il crittoanalista X devia le comunicazioni che provengono da U e V e le dirige verso sé stesso:

U richiede a V la chiave pubblica (tramite mail, pagina web...)

X intercetta la risposta che contiene $\texttt{k}_V\texttt{[pub]}$ e la sostituisce con la sua chiave pubblica $\texttt{k}_X\texttt{[pub]}$

X si pone in ascolto dei crittogrammi spediti da U a V, cifrati mediante $\texttt{k}_X\texttt{[pub]}$

X rimuove dal canale ciascuno di questi crittogrammi, li decritta, li cifra mediante $\texttt{k}_V\texttt{[pub]}$ e li rispedisce a V

U e V non si accorgeranno della presenza di X se il processo di intercettazione e rispedizione è sufficientemente veloce da rendere il relativo ritardo apparentemente attribuibile alla rete.

7.4 Certification Authority

Un algoritmo crittografico è tanto robusto quanto la sicurezza delle sue chiavi. Lo scambio/generazione della chiave è un passo cruciale. Gli attacchi MitM sono i principali problemi di sicurezza da affrontare.
Sono così nate le certification authority: sono **infrastrutture che garantiscono la validità delle chiavi pubbliche** e ne regolano l'uso, **gestendo la distribuzione** sicura delle chiavi delle due entità che vogliono comunicare.

Key Certification Authority (CA) Ente preposto alla certificaizone di validità delle chiavi pubbliche.
La CA autentica l'associazione ⟨utente, chiave pubblica⟩ emettendo un **certificato digitale**, che **consiste della chiave pubblica e di una lista di informazioni relative al proprietario**, opportunamente firmate dalla CA.
Per svolgere correttamente il suo ruolo, la CA mantiene un archivio di chiavi pubbliche sicuro, accessibile a tutti e protetto da attacchi in scrittura non autorizzati.
La chiave pubblica della CA è nota a tutti gli utenti, che la mantengono protetta da attacchi esterni e la utilizzano per verificare la firma della CA stessa sui certificati.

Certificato digitale Un certificato digitale contiene:

una indicazione del suo formato (numero di versione)

il nome della CA che lo ha rilasciato

un numero seriale che lo individua univocamente all'interno della CA emittente

la specifica dell'algoritmo usato dalla CA per creare la firma elettronica

il periodo di validità del certificato (data di inizio e data di fine)

il nome dell'utente a cui questo certificato si riferisce e una serie di informazioni a lui legate

un'indicazione del protocollo a chiave pubblica adottato da questo utente per la cifratura e la firma: nome dell'algoritmo, suoi parametri, e chiave pubblica dell'utente

firma della CA eseguita su tutte le informazioni precedenti

CA Se U vuole comunicare con V, richiede $\texttt{k}_V\texttt{[pub]}$ alla CA che risponde inviando a U il certificato cert_V di V.
Poiché U conosce $\texttt{k}_{CA}\texttt{[pub]}$, può controllare l'autenticità del certificato verificandone il periodo di validità e la firma prodotto dalla CA su di esso.
Se tutti i controlli vanno a buon fine, $\texttt{k}_V\texttt{[pub]}$ nel certificato è corretta e U avvia la comunicazione con V.
Un crittoanalista potrebbe intromettersi solo falsificando la certificazione, ma si assume che la CA sia fidata e il suo archivio di chiavi sia inattaccabile.

Organizzazione In ogni stato **esistono diverse CA organizzate ad albero**. La verifica di un certificato è in questo caso più complicata e si svolgi attraverso una catena di verifica hce vanno dalla CA di U alla CA di V.
V invia ad U una sequenza di caratteri ordinati in accordo alla CA che li ha firmati, da quella che ha certificato V a quella che è la radice dell'albero delle CA.

Ogni utente mantiene localmente al sistema una copia del proprio certificato cert_U e una copia di $\texttt{k}_{CA}\texttt{[pub]}$.
Interagisce con la CA una sola volta, e poi la gestione delle chiavi pubbliche diventa decentralizzata.

Protocollo 4 Messaggio m cifrato, firmato in hash e certificato.
Firma, cifratura e certificazione

Il mittente U si procura il certificato cert_V di V

Calcola $h(m)$ e genera la firma $f = D(h(m), k_U[priv])$

Calcola il crittogramma $c = C(m, k_V[pub])$

Spedisce a V la tripla $\langle \text{cert}_U, c, f \rangle$
cert_U contiene $k_U[pub]$ e la specificazione della funzione h usata

Verifica del certificato

Il destinatario V riceve $\langle \text{cert}_U, c, f \rangle$ e verifica l'autenticità di cert_U (e quindi di $k_U[pub]$) utilizzando la propria copia di $k_{CA}[pub]$

Decifrazione e verifica della firma

V decifra il crittogramma c con la propria chiave privata e ottiene $m = D(c, k_V[priv])$

V verifica l'autenticità della firma apposta da U su m controllando che risulti $C(f, k_U[pub]) = h(m)$

Conclusioni Il punto debole del meccanismo delle CA è rappresentato dai certificati revocati. La CA mettono a disposizione un archivi con i certificati non più validi.
La **frequenza di controllo di questi certificati** e le modalità della loro comunicazione agli utenti sono **cruciali per la sicurezza**.
Attacchi MitM possono essere evitati se si stabilisce un incontro diretto tra un utente e la CA nel momento in cui si istanza il sistema asimmetrico dell'utente.

7.5 Protocollo Zero Knowledge

Protocollo a conoscenza zero Lo vedremo all'opera per l'identificazione. Silvio Micali, premio Turing, e Shefi Goldwasser (MIT, 1989).

Obiettivo Due utenti: **prover** e **verifier**, anche chiamati Peggy/Victor.
Il protocollo è uno scambio tra P e V, composto da una serie di sfide che P deve superare per convincere V di essere in possesso di una certa informazione. Scambio di messaggi che permette a P di **dimostrare** a V, e a fine protocollo V deve ragionevolmente convincersi che P possiede l'informazione.

Esempio Peggy sostiene di poter contare i granelli di sabbia di una spiaggia solamente guardandola, e Victor vuole controllare se quanto dice Peggy è vero (**non con certezza**, ma con probabilità prossima a 1).
Victor sceglie la spiaggia e ci va con Peggy. **Fase di preparazione** dove Victor chiede a Peggy di comunicare la parità dei granelli di sabbia.

$$P \to^{b_0} V \quad b_0 = \textit{ parità dei granelli di sabbia}$$

Dopodiché, il protcollo funziona così:

```
for (i=1 to k) { //k scelto da V
        P si volta;
        V sceglie un e bit (0, 1) casuale;
        if (e==0) V toglie un granello di sabbia
        V chiede a P il nuovo valore bi //chiede nuovamente parita granelli
        if ((e == 0 && b[i] != b[i-1]) || //parita cambiata perche tolto granello
                (e == 1 && b[i] == b[i-1])) //parita non cambiata
                        continua alla prossima iterazione;
        else //P impostore
                        STOP
}
//P dice il vero con probabilita 1-(1/2)^k
```

La probabilità di vincere una sfida ingannando, se Peggy è disonesta, è la probabilità di indovinare il bit generato, $\frac{1}{2}$.
Quindi la probabilità di vincere ingannando è di farlo per k volte, cioè $\left(\frac{1}{2}\right)^k$.
Quindi, la probabilità di ingannare V è $\left(\frac{1}{2}\right)^k$.
La probabilità che P abbia la capacità assunta è almeno $1 - \left(\frac{1}{2}\right)^k$
Evidenza sperimentale significativa, ma **non dimostrazione rigorosa**, che P abbia la capacità/conoscenza.

7.5.1 Proprietà di un protocollo zero knowledge

Principi generali

1. **Completezza**
   Se P è onesto (la sua affermazione è vera), V **accetta sempre** la dimostrazione. Cioè P deve sempre poter superare le sfide di V.

2. **Correttezza**
   Se P è disonesto (la sua affermazione è falsa), V **può essere ingannato** con probabilità $\leq \left(\frac{1}{2}\right)^k$ con k scelto da V

3. **Conoscenza zero**
   Se l'affermazione di P è vera, il verificatore V (anche se disonesto) non può dedurre alcuna informazione se non la veridicità di questo fatto.

7.5.2 Protocollo di identificazione a conoscenza zero

Protocollo di Fiat-Shamir Uno dei primi nati. P dimostra a V la sua identità senza svelare alcun'altra informazione.
Basato sulla difficoltà di calcolo della radice quadrata in modulo n composto.
$t = s^2 \bmod n$, n composto. Victor conosce t, n, Peggy deve convincere V di conoscere $s = \sqrt{t}$. Deve succedere in tempo polinomiale, altrimenti in tempo esponenziale V può calcolarsela.

Fase di preparazione
P sceglie p, q primi molto grandi e calcola $n = p \cdot q$ e sceglie $s < n$ segreto. Calcola $t = s^2 \bmod n$.
Tutte attività eseguibili in tempo polinomiale.

P rende nota $\langle t, n \rangle$ (chiave **pubblica**) e mantiene segreta $\langle p, q, s \rangle$ (chiave **privata**)
p, q, s sono calcolabili in tempo polinomiale solo conoscendo t, n.
Se P è la proprietaria della chiave privata, sarà in grado di convincere qualunque verificatore con probabilità $1 - \left(\frac{1}{2}\right)^k$ senza che acquisica maggiori informazioni.

Protocollo, ripeti k volte:

1. V chiede a P di iniziare una iterazione
2. P genera un intero $r < n$ casuale, ne calcola $u = r^2 \bmod n$ e comunica u a V. Numero casuale usato per la sfida e buttato via.
3. V genera il **bit casuale** e e lo comunica a P
4. P calcola $z = r \cdot s^e \bmod n$ e comunica z a V
   Se $e = 0$, $z = r$. Se $e = 1$, $z = r \cdot s \bmod n$

5. V calcola $x = z^2 \bmod n$
   $(rs^e)^2 = r^2(s^e)^2 = u(s^2)^e = ut^e$
   Se $x = ut^e \bmod n$ allora sfida superata, si va alla successiva.
   Altrimenti STOP, P non è identificato.

r non viaggia mai in chiaro, tranne quando $e = 0$ cioè $z = r$. s non viaggia mai da solo, ma r è legato alla sola sfida quindi non è un problema.

Tutte le volte che $e = 0$ la sfida è facile da superare. Perché non si chiede sempre di usare $e = 1$, cioè di usare il segreto? Ad esempio, V manda sempre $e = 1$. P può imbrogliare e superare sempre. P si aspetta sempre $e = 1$, al passo 2. sceglie r a caso e invia a V non $u = r^2$ ma $u = \frac{r^2}{t} \bmod n = r^2 t^{-1} \bmod n$ (assumendo che t^{-1} esista e sia unico).
Al passo 4, P deve comunicare rs^e ma invia sempre $z = r$ perché non conosce s. Ma avendo imbrogliato al passo 2., V verifica in questo modo:

$x = z^2 \bmod n$ e verificare se $x = ut^e = ut$ perché $e = 1$
$x = z^2 \bmod n = r^2 \bmod n$
$ut = \frac{r^2}{t} \cdot \not{t} \bmod n = r^2 \bmod n$
Quindi la verifica è passata, $x = ut$

Quindi forzare a usare il segreto porta ad un facile imbroglio. Usare il bit casuale spinge P a cercare di prevedere le scelte: se prevede $e = 0$ non varia il protocollo, se prevede $e = 1$ prova la modifica appena vista.

Completezza Se $e = 0 \Rightarrow x = ut^e \bmod n = u \bmod n$
Se $e = 1 \Rightarrow x = z^2 \bmod n = (rs^e)^2 \bmod n = ut \bmod n$
Quindi P supera tutte le sfide, V accetta la dimostrazione.

Correttezza Dimostrare che si può imbrogliare con probabilità al massimo $1 - \left(\frac{1}{2}\right)^k$
P disonesto riesce a ingannare V se prevede correttamente il bit inviato da V ad ogni iterazione (visto sopra). Poiché e è generato casualmente, le previsioni di P sono corrette con probabilità $\frac{1}{2}$ ad ogni iterazione.
Per k iterazioni, diventa probabilità di inganno $= \left(\frac{1}{2}\right)^k$

7.6 Protocollo SSL

Sfrutta i concetti visti fin'ora nel corso. Oggi su usa il TLS, sviluppato sulla base dell'SSL.

Secure Socket Layer Alla base dei protocolli più diffusi nelle comunicazioni sicure. Garantisce **confidenzialità** e **affidabilità** delle comunicazioni su internet, proteggendole da intrusioni, modifiche o falsificazioni.
Sviluppato da Netscape per effettuare comunicazione sicure sul protocollo HTTP, la prima versione è del 1994: è **progettato in modo da permettere la comunicazione tra computer che non conoscono le reciproche funzionalità**.

L'utente U desidera accedere via internet ad un servizio offerto dal sistema S.

Confidenzialità La trasmissione è cifrata mediante un sistema ibrido:

Cifrario **asimmetrico** per **costruire e scambiare le chiavi di sessione**.

Cifrario **simmetrico** che **utilizza queste chiavi per criptare i dati** nelle comunicazioni successive.

Autenticazione SSL garantisce l'autenticazione dei messaggi accertanto l'identità dei due partner **attraverso un cifrario asimmetrico**, oppure **facendo uso di certificati digitali** e **inserendo nei messaggi stessi un apposito MAC** (che usa una funzione hash one-way crittograficamente sicura)

Posizione SSL si innestra tra un protocollo di trasporto affidabile (TCP/IP) e un protocollo applicativo (es. HTTP): è **completamente indipendente dal protocollo applicativo sovrastante**.
Protocollo **HTTPS**: combinazione tra SSL e HTTP, utilizzato da server web sicuri (`https://...`)

7.6.1 Struttura

Due livelli SSL è organizzato su due livelli:

SSL Handshake
Responsabile dell'instaurazione e del mantenimento dei parametri usati da SSL Record.
Permette all'utente e al sistema di **autenticarsi**, **negoziare gli algoritmi** di cifratura e firma e **stabilire le chiavi per i singoli algoritmi** crittografici e per il MAC
Crea un canale sicuro, affidabile e autenticato tra utente e sistema, entro il quale SSL Record fa viaggiare i messaggi divisi in blocchi opportunamente cifrati e autenticati.

La sessione di comunicazione è innescata da uno **scambio di messaggi preliminari** (handshake) indispensabili per la creazione del canale sicuro. Attraverso questi messaggi, S server e U client si identificano a vicenda e cooperano nella costruzioni delle chiavi segrete da impiegare nelle comunicazioni simmetriche successive. Il protocollo è **organizzato in passi**:

1. **U manda a S un messaggio di** `client hello`, con cui:

   Richiede la creazione della connessione SSL

   Specifica le prestazioni di sicurezza che desidera siano garantite durante la connessione (cifrari e meccanismi di autenticazione che U può supportare)

   Invia una sequenza di byte casuali

   Un esempio di cipher suite: `SSL_RSA_WITH_AES_CBC_SHA1`

   RSA per lo scambio di chiavi di sessione

   AES per la cifratura simmetrica

   CBC indica l'impiego di un cifrario di composizione a blocchi

   SHA1 funzione hash one-way per il MAC

2. Il sistema S riceve `client hello`, seleziona una cipher suite che anch'esso è in grado di supportare e **invia ad U un messaggio di** `server hello` che specifica la sua scelta e contiene una nuova sequenza di byte casuali.
   Se U non riceve il messaggio di `server hello`, interrompe la comunicazione.
3. **S si autentica con U inviandogli il proprio certificato digitale** e gli eventuali altri certificati della catena di certificazione dalla sua CA fino alla CA radice.
   Se i servizi offerti da S devono essere protetti dagli accessi, **S può richiedere a U di autenticarsi inviando il suo certificato digitale** (avviene raramente, la maggior parte degli utenti non ha un certificato personale quindi il sistema dovrà accertarsi dell'identità dell'utente in un secondo tempo)
4. **S invia il messaggio di** `server hello done` con cui sancisce la fine degli accordi della cipher suite e sui paramtetri crittografici a essa associati.
5. Per accertare l'autenticità del certificato ricevuto da S, l'utente U controlla che:

   La data corrente sia inclusa nel periodo di validità del certificato

   La CA che ha firmato il certificato sia tra quelle fidate

   La firma apposta dalla CA sia autentica

6. L'**utente U costruisce un pre-master secret** costituito da una sequenza di byte casuali (quindi genera un numero segreto). Cifra il pre-master secret con il cifrario a chiave pubblica su cui si è accordato con S e spedisce il crittogramma ottenuto a S.
   Ad esempio U cifra il pre-master secret con RSA usando la chiave pubblica presente nel certificato di S.
   Il pre-master secret viene combinato da U con alcune stringhe note e con i byte casuali presenti sia nel messaggio di `client hello` che in quello di `server hello`.
   A tutte le sequenze U applica delle funzioni hash one-way (SHA1 e MD5) secondo una combinazione opportuna.
   Ottiene così il **master secret**.
7. S decifra il crittogramma contenente il pre-master secret ricevuto da U, **calcola il master secret** mediante le stesse operazioni di U (possiede le stesse informazioni).
   Sia U che S conoscono: il numero casuale che U ha mandato a S, il numero casuale che S ha mandato a U (entrambi scambiati in chiaro) e il premaster secret (inviato cifrato). Quindi entrambi calcolano il master secret.

8. (Opzionale) Se all'utente U viene richiesto un certificato ed egli non lo possiede, il sistema interrompe l'esecuzione del protocollo.
   Altrimenti U invia il proprio certificato con allegate una serie di informazioni firmate con la sua chiave privata, tra cui: il master secret e tutti i messaggi scambiati fino a quel momento (**SSL history**)
   S controlla il certificato di U e verifica autenticità e correttezza della SSL history. Se ci sono anomalie, la comunicazione con U è interrotta.

9. U costruisce e spedisce a S `finished`, poi S costruisce e spedisce a U lo stesso messaggio. Si tratta del primo messaggio protetto mediante il master secret e la cipher suite: nei due invii la struttura del messaggio è la stessa, ma cambiano le informazioni in esso contenute.
   La **costruzione** avviene in due passi:

   I All'inizio si concatenano il master secret, tutti i messaggi di handshake scambiati fino a quel momento e l'identità del mittente (U o S)

   II La stringa ottenuta è trasformata applicando SHA1 e MD5: si ottiene una coppia di valori che costituisce il messaggio `finished`

   Il messaggio è diverso perché S aggiunge ai messaggi di handshake anche il messaggio `finished` appena ricevuto da U.
   Il destinatario della coppia, S o U, non può invertire la computazione precedente in quanto generata da funzioni one-way. Ricostruisce l'ingresso delle due funzioni SHA1 e MD5, le ricalcola e controlla che la coppia generata coincida con quella ricevuta, come dimostrazione che la comunicazione è avvenuta correttamente.

Il **master secret** viene usato da U e S per costruire una propria tripla contenente:

La chiave segreta da usare nel cifrario simmetrico

La chiave per l'autenticazione del messaggio (costruzione del MAC)

La sequenza di inizializzazione per cifrare in modo aperiodico messaggi molto lunghi

Le triple di U e S sono diverse tra loro ma note a entrambi i partner: ciascuno usa la propria, il che aumenta la sicurezza delle comunicazioni.

SSL Record
Livello più basso, connesso direttamente al protocollo di trasporto. Ha l'obiettivo di incapsulare i dati spediti dai protocolli dei livelli superiori, assicurando confidenzialità e integrità della comunicazione.
Realizza fisicamente il canale (meccanismo di rete): utilizza la cipher suite stabilita da SSL Handshake per cifrare e autenticare i blocchi di dati, prima di spedirli attraverso il protocollo di trasporto sottostante.
Il canale sicuro approntato da SSL handshake viene realizzato da SSL record: i **dati** sono **frammentati in blocchi** e **ciascun blocco** viene:

numerato, compresso e autenticato con l'aggiunta del MAC

cifrato con il cifrario asimmetrico concordato

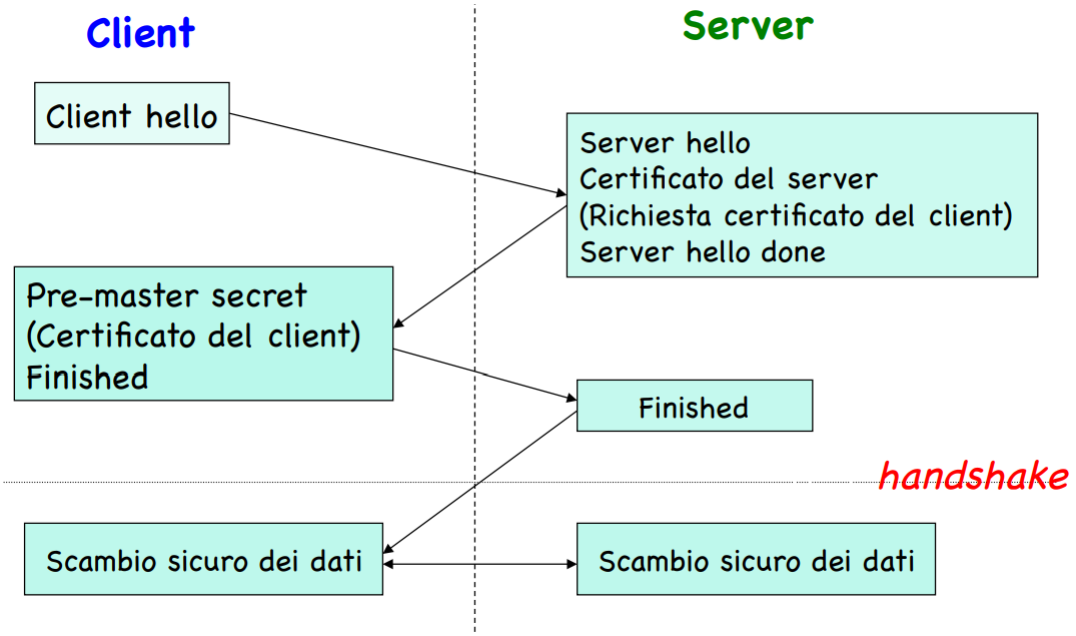
trasmesso dall'SSL record utilizzando il protocollo di trasporto sottostante

Il destinatario esegue un procedimento inverso sui blocchi:

decifra e verifica la loro integrità attraverso il MAC

decomprime e riassembla i blocchi in chiaro

li consegna all'applicazione sovrastante



Sicurezza Nei passi di `hello`, i due partner creano e si inviano due sequenze casuali per la costruzione del master secret, che risulta così diverso in ogni sessione SSL.
Il **crittoanalista non può riutilizzare i messaggi di handshake catturati sul canale per sostituirsi a S in una successiva comunicazione con U**.

MAC dei blocchi di dati SSL record numera in modo incrementale ogni blocco di dati e autentica il blocco attraverso un MAC.
Per prevenire la modifica del blocco da parte di un crittoanalista attivo, il MAC viene calcolato come immagine hash one-way di una stringa costruita concatenando:

- il contenuto del blocco
- il numero del blocco nella sequenza
- la chiave del MAC
- alcune stringhe note e fissate a priori

Dato che i MAC sono cifrati assieme al messaggio, un crittoanalista non può alterarli senza aver forzato la chiave simmetrica di cifratura: **un attacco volto a modificare la comunicazione tra i due partner è difficile almeno quanto quello volto alla sua decrittazione**.

Il sistema è autenticato Il canale definito da SSL handshake è immune da attacchi attivi MitM poiché il sistema S viene autenticato con un certificato digitale.
L'utente U può comunicare il pre-master secret al sistema S in modo sicuro attraverso la chiave pubblica presente nel certficato di S.
Solo S può decifrare quel crittogramma e costruire il master secret, su cui si fonda la costruzione di tutte le chiavi segrete adottate nelle comunicazioni successive.
Solo il sistema S, quello a cui si riferisce il certificato, potrà quindi entrare nella comunicazione con l'utente U.

L'utente può essere autenticato Il certificato di U, se richiesto, e la sua firma apposta sui messaggi scambiati nel protocollo (SSL history) consentono a S di verificare che U sia effettivamente quello che dichiara di essere e che i messaggi siano stati effettivamente spediti da esso.
Se ciò non si verifica, S deduce che il protocollo è stato alterato (casualmente o maliziosamente con un attacco MitM) e interrompe la comunicazione.
L'opzionalità della comunicazione ha reso l'SSL molto diffuso nelle transazioni commerciali via internet: per gli utenti, la necessità di certificazione può costituire un ostacolo pratico ed economico. L'utente può essere autenticato con altrim etodo (logi, pin...)

Generazione delle sequenze casuali Le tre sequenze casuali generate da U e da S e comunicate nei messaggi di `client hello`, `server hello` e pre-master secret sono usate per creare il master secret, quindi per generare le chiavi segrete di sessione.

La sequenza corrispondente al pre-master secret viene generata da U e comunicata per via cifrata a S.

La non predicibilità di questa sequenza è cruciale per la sicurezza del canale SSL:
una sua cattiva generazione renderebbe il protocollo molto debole.

Messaggio `finished` Contiene tutte le informazioni scambiate nel corso dell'handshake.
Scopo: consente un ulteriore controllo sulle comunicazioni precedenti per garantire che:

Queste siano avvenute correttamente

U e S dispongano dello stesso master secret

Che la comunicazione non sia stata oggetto di un attacco attivo

SSL Almeno sicuro quanto il più debole cipher suite supportato. Dal 2000 le norme internazionali non pongono alcuna limitazione sui cifrari impiegabili (se non il alcuni paesi).
Consigliato disabilitare i propri sistemi dall'impiego di cifrari ormai notoriamente insicuri e chiavi troppo corte.

Capitolo 8

Quantum Key Exchange

8.1 Meccanica Quantistica

Sovrapposizione Proprietà di un sistema quantistico di trovarsi in diversi stati contemporaneamente. Combinazione lineare a coefficienti complessi degli stati possibili. Il modulo del quadrato è la probabilità che misurando si trovi quello stato.

Decoerenza La misurazione di un sistema quantistico disturba il sistema: il sistema disturbato perde la sovrapposizione degli stati e collassa in uno stato singolo.

No-Cloning Impossibilità di duplicare un sistema conservando nella copia lo stato quantico dell'originale (senza misurarlo).
Impossibile copiare uno stato quantistico non noto.

Entanglement Possibilità che due o più elementi si trovino in stati quantici correlati tra loro in modo che, pur se portati a grande distanza, mantengano la correlazione.

8.2 Protcollo BB48

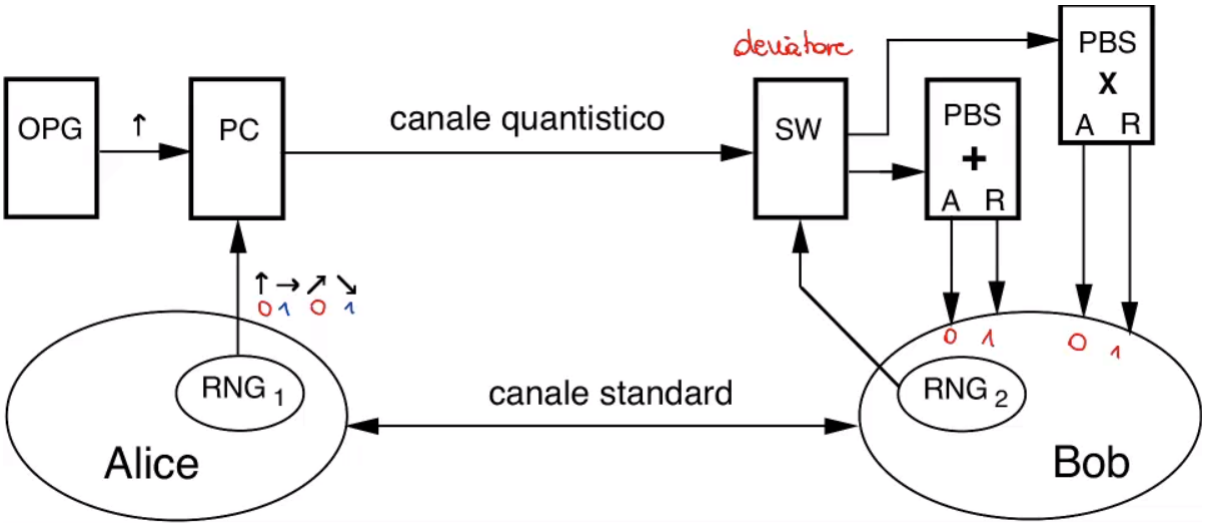
Bennet, Brassard
Scambio di chiave mediante l'invio di fotoni polarizzati. 4 stati di **polarizzazione** (piano di oscillazione del campo elettrico):

1. v Polarizzazione verticale $\uparrow$
2. h Polarizzazione orizzontale $\rightarrow$
3. $+45$ Polarizzazione +45 gradi $\nearrow$
4. -45 Polarizzazione -45 gradi $\searrow$

$\left.\begin{array}{ll} \uparrow & 0 \\ \rightarrow & 1 \end{array}\right\}$ Base di polarizzazione $+$

$\left.\begin{array}{ll} \nearrow & 0 \\ \searrow & 1 \end{array}\right\}$ Base di polarizzazione $\times$

Se non so in che base è stato polarizzato il fotone, non posso distinguere tra le quattro polarizzazioni, ma solo tra due possibili polarizzazioni nella stessa base. Se conosco la base, non è un problema, ma se non la conosco è difficile misurare.
Quindi **non è possibile distinguere fra i 4 casi**, l'**unica misura possibile è tra due stati ortogonali nella stessa base**.



Componenti

OPG: **One Photon Gun**
Un fotone alla volta con polarizzazione 90 gradi

PC: **Cella di Pokel**
Impone al fotone una certa polarizzazione tra quelle volute.

PBS: **Beam Splitter Polarizzante**
Determina la polarizzazione del fotone, risponde correttamente solamente se la base di riferimento dello strumento è la stessa del fotone (quindi scelta da Alice)

Alice sceglie casualmente (RNG_1) la polarizzazione del fotone tramite la cella di Pokel. Il fotone viaggia su fibra ottica. Bob dovrà scegliere casualmente (RNG_2) la base attraverso il deviatore.

PBS Ha un proprio asse di polarizzazione S, con un angolo θ con l'asse di polarizzazione F del fotone.
Devia il fotone verso una delle due uscite (Assorbimento, Riflessione):

A: il fotone viene inviato all'uscita A con probabilità $\cos^2\theta$, assume polarizzazione S

R: il fotone viene inviato a R con probabilità $\sin^2\theta$, e esce con polarizzazione ortogonale a S

Angolo:

$\theta = 0$, quindi fotone ha stessa polarizzazione dello strumento, il fotone esce da A con polarizzazione S (= F) ($\cos^2\theta = 1$).
Scelta la base corretta.

$\theta = 90$, il fotone esce da R con polarizzazione ortogonale a S (= F) ($\sin^2\theta = 1$)
Scelta la base corretta.

$\theta = \pm 45$ $\cos^2\theta = \sin^2\theta$
$\Rightarrow$ il fotone esce con pari probabilità da A o R, e la polarizzazione cambia (S o $S^{\perp}$), perdendo la polarizzazione iniziale F.

Fenomeno quantistico: la lettura attraverso il PBS ha distrutto lo stato quantistico precedente.

Tutti i bit incerti (i 50%) verranno ignorati per la costruzione della chiave. Useremo solo i bit misurati correttamente

8.2.1 Passi del protocollo

Sul canale standard si comunicano non i bit, ma si scambiano le basi: Bob dice le basi che ha usato, Alice risponde con le basi corrette. Dove hanno scelto la stessa base la comunicazione è avvenuta correttamente, dove hanno scelto basi diverse scarteranno i bit. Sceglieranno basi concordi più o meno la metà delle volte.

Protocollo

1. Alice invia una sequenza S_A sul canale quantistico.
   (S_A = sequenza di bit codificati nelle polarizzazioni dei fotoni)

2. Bob interpreta S_A con le sue basi, che ha scelto casualmente: ottiene una sequenza S_B.
   Coincideranno sicuramente su circa metà bit, dove le basi coincidono.

3. Bob, sul canale standard, comunica ad Alice non i bit ma soltanto la base (al cui interno, il bit può essere 0 o 1). Alice indica a Bob quali basi sono comuni alle sue.
   Adesso entrambi sanno come sono state fatte preparazione e misura.

⇒ In assenza di interferenze di Eve, Alice e Bob possiedono una sottosequenza $S'_A = S'_B$ identica formata dai bit codificati da Alice e decodificati da Bob con basi comuni.
$|S'_A| = |S'_B| \simeq \frac{|S_A|}{2}$

4. Alice e Bob sacrificano una porzione S''_A, S''_B di S'_A, S'_B in posizioni prestabilite, comunicandole sul canale standard (si passano quindi *alcuni* bit di S'_A,S'_B)
   La comunicazione sul canale standard può essere fatta anche chiaro ma deve essere almeno autenticata.
   A meno di errori sperimentali, se Eve non c'è stato dovrebbero essere sequenze identiche. Se c'è stato Eve, potrebbero non essere coincidenti: Eve ha rotto lo stato quantistico misurandone lo stato.
   Se $S''_A \neq S''_B$, allora Alice e Bob interrompono la comunicazione.
   Altrimenti usano $S'_A - S''_A = S'_B - S''_B$

Esempio Esempio di comunicazione e scelta

	S_A	1	0	1	1	1	0	0	…
Alice	Basi di A	+	×	+	×	×	+	×	…
	Fotoni di A	→	↗	→	↘	↘	↑	↗	…
	Basi di B	+	×	+	+	+	×	×	…
Bob	Fotoni di B	→	↗	→	↑	→	↘	↗	…
	S_B	**1**	**0**	**1**	0	1	1	**0**	…

Produrranno la sequenza che corrisponde alle basi scelte coincidenti (evidenziate in B) $S'_A = S'_B = 1010\ldots$

In caso di interferenza di Eve, deve fare una scelta di basi.

	S_A	1	0	1	1	1	0	0	…
Alice	Basi di A	+	×	+	×	×	+	×	…
	Fotoni di A	→	↗	→	↘	↘	↑	↗	…
	Basi di E	+	+	+	×	+	×	+	…
Eve	Fotoni di E	→	→	→	↘	↑	↗	→	…
	S_E	1	1	1	1	0	0	1	…
Che porterà Bob a produrre									
	Basi di B	+	×	+	+	+	×	×	…
Bob	Fotoni di B	→	↘	→	↑	↑	↗	↘	…
	S_B	**1**	**1**	**1**	0	0	0	**1**	…
	Variazione		⇑						

QBER Alice e Bob stabiliscono anche il **Quantum Bit Error Rate**: percentuale prevedibile di bit errati a causa degli errori sperimentiali/rumore…
Nel confronto fra S''_A e S''_B, se l'errore è $> QBER$ allora ci sono state interferenze di Eve e la chiave viene scartata. Se l'errore è $\leq$ QBER, allora concludono che non ci sono state intromissioni, ripuliscono gli errori con tecniche di correzione ricostruendo una chiave corretta in tutti i bit e usano le sequenze rimaste (identiche) come chiavi.

Attacco pericoloso Se Eve ne intercetta solo pochi di bit, l'attacco è pericoloso perché potrebbe provocare un errore inferiore a QBER e quindi l'intromissione potrebbe venire scambiata per errore sperimentale. Eve conoscerebbe qualche bit della chiave.
Nell'uso di questi protocolli non si usa mai infatti S', S'' ma una sorta di amplificazione di privacy prendendo la sequenza comune (di cui Eve può avere qualche bit) e usando come chiave l'immagine hash crittografica della funzione.

8.2.2 Protocollo

Canale Quantistico

```
SA[1,n] = sequenza iniziale di bit da cui verra estratta la chiave, rappresentata
con un codice a correzione di errori

for i=1 to n
        Alice:  sceglie una base a caso
                codifica SA[i]
                invia il fotone a Bob
        Eve (se presente):      intercetta il fotone
                                lo misura con una sua base
                                lo invia a Bob
                                calcola SE[i]
                        (questo anche solo per qualche i)
        Bob:    sceglie una base a caso
                interpreta il fotone ricevuto
                costruisce SB[i]
```

Canale Standard

```
QBER = percentuale di errori dovuti al rumore
h = funzione hash crittografica
Alice e Bob:
        1. Estraggono SA1 e SB1 corrispondenti alle basi
        comuni.
        Estraggono due sottosequenze di SA1 e SB1 in
        posizioni concordate, ottenendo SA2, SB2

        2. Si scambiano SA2 e SB2. Se la percentuale di
        bit diversi e maggiore di QBER, allora STOP

        3. Altrimenti, calcolano SA1–SA2 e SB1–SB2 e li
        decodificano con il codice di correzione di errori.
        Ottengono una sequenza comune SC
        (Attenzione: Eve potrebbe conoscere alcuni bit di SC)

        4. Alice e Bob calcolano k = h(SC) e usano k come chiave.
```

Capitolo 9

Bitcoin

2008: "Bitcoin: a peer to peer Electronic Cash System", Satoshi Nakamoto. Nel 2009 il "blocco genesi" (50 BTC) e la prima transazione (10 BTC). Nel 2010, la prima transazione commerciale.

Indirizzo Utente A:

> k_A`[pub]` Si trasforma in indirizzo.
> **Identificatore di A**, serve per i nuovi pagamenti ad A e per controllare la firma di A.
>
> k_A`[prv]` Per **firmare le transazioni**.

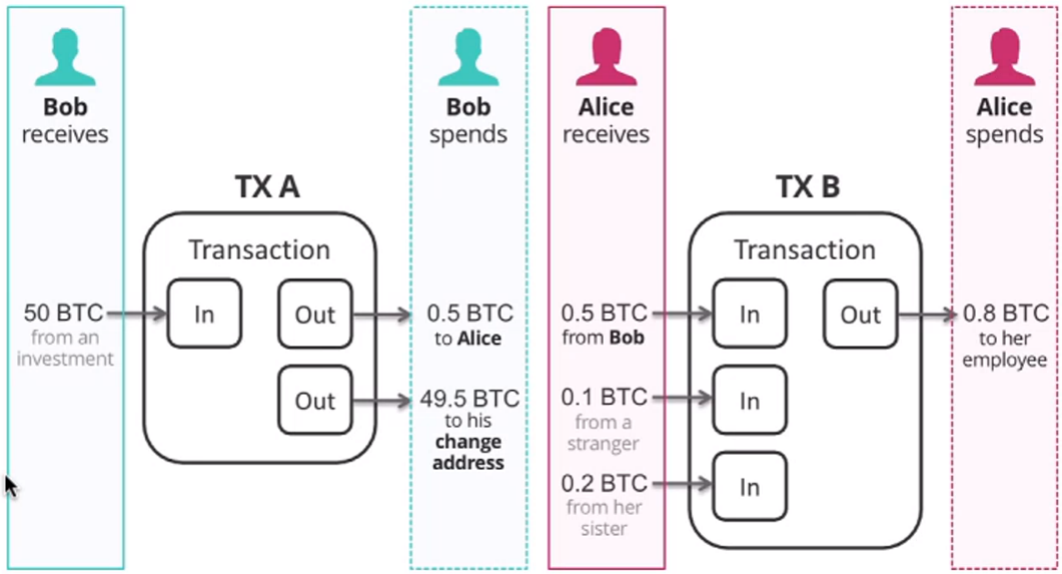
Tutto dalla crittografia su curve ellittiche.

Wallet **Insieme delle credenziali** che attestano la proprietà in BTC di un utente (il portafoglio). Diverse coppie indirizzo/chiave privata associata, oltre al software per la gestione.

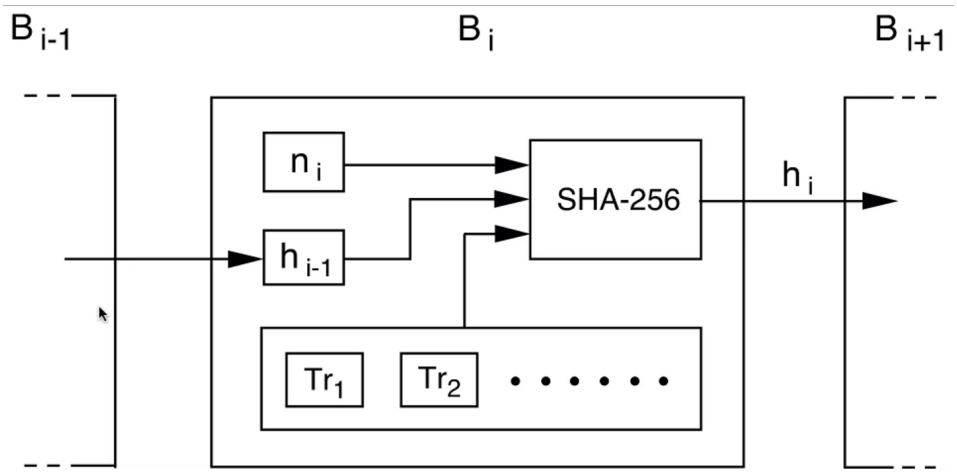
Transazione Una transazione è un movimento di BTC da un utente ad un altro.
Semplificando, Alice vuole mandare x BTC a Bob, cioè Alice prende un messaggio $m = \langle \text{adr}_A, x, \text{adr}_B \rangle$, esegue l'hash $h(m) = \text{SHA-256}(m)$ e lo firma $f = D(h, k_A[priv])$.
Fatto questo, Alice diffonde in broadcast sulla rete $\langle m, f \rangle$

Mancando un sistema centralizzato, il destinatario aspetta che la transazione venga convalidata. Cioè aspetta che il sistema distribuito raggiunga un consenso su quali sono le transazioni valide. Questo richiede intorno ai 10 minuti. Anche dopo i 10 minuti, quando se la transazione è valida viene aggiunta ad un blocco della blockchain, è bene aspettare ulteriore tempo perché al nuovo blocco ne vengano aggiunti altri per essere sicuri di essere sul ramo della blockchain ufficiale.
Alcuni utenti possono aggiungere blocchi in punti diversi, ma una sola catena è quella ufficiale ed è quella più lunga (con più numero di transazioni collegate su cui quindi è stato raggiunto il consenso). Le transazioni nelle altre catene (catene orfane) sono considerate non valide: chi ha pagato è come se non lo avesse fatto, e può rifare la transazione facendo in modo che finisca sulla blockchain ufficiale.

Quindi 10m per attendere che il blocco sia aggiunto alla blockchain, e bisogna attendere che altri 6 blocchi vengano aggiunti al blocco della mia transazione. Quindi circa un'ora.



I bitcoin in input devono finire tutti in output. Un po' come se Bob usasse una banconota da 50 BTC, ne dà 0.5 BTC ad Alice e "si manda" 49.5 BTC di resto (a sé stesso o comunque ad un indirizzo di cui possiede la chiave privata).
Il requisito è che la **somma degli input** $\geq$ **somma degli output**. Se è $>$, la differenza non va a chi ha pagato, ma viene trattenuta dal nodo computazionale che aggiunge il blocco alla blockchain.



Tr_i: la catena di transazioni da convalidare, con una transazione finale dove mi prendo il premio e la somma di tutte le differenze input-output

h_{i-1}: hash del contenuto del blocco precedente

n_i: **nonce**, è un intero ma non lo si conosce

Di tutte queste informazioni viene fatto l'hash SHA-256 in modo che h_i sia minore di una certa soglia fissata dal sistema. Quindi
devo trovare $n_i \mid h_i =$ **SHA-256**$(n_i, h_{i-1}, \{Tr_i\}) <$ **soglia**.
Non ci sono meccanismi, và eseguita una ricerca enumerativa. La proof of work è l'n_i che produce una h_i con t zeri iniziali. Il parametro t è fissato dal sistema ed è calibrato in base alla potenza computazionale dei nodi attivi, in modo tale che in media la ricerca di n_i richieda circa 10 minuti.
Trovare il nonce è **difficile**, **verificarlo** è **facile**.

Mining Un **miner** è un **nodo che valida le transazioni e aggiunge i blocchi alla blockchain**.
Ogni miner prepara il suo blocco e cerca di aggiungerlo. Chi risolve il nonce e lo diffonde per broadcast a tutti i nodi. I nodi lo controllano, controllano la validità delle transazioni ed esprimono il loro consenso cercando di creare nuovi blocchi da attaccare a questo. Si privilegia il blocco con il maggior numero di transazioni: ci possono essere più rami ma alcuni potrebbero diventare rami morti e solo quello con più transazioni diventa ufficiale.

Fallimento All'inizio tutti potevano contribuire minando, con la nascita dei mining pool (consorzi di tantissimo hardware progettato ad hoc per mining) questo non è più possibile e solo pochi minano e guadagnano dal mining.

Attacchi Attacco del 51%: devono avere maggior potenze computazionale del mining pool più potente.

Capitolo 10

Esercizi

10.1 Complessità e randomizzazione

Esercizio 1 Un sistema crittografico impiega chiavi private di 46 bit. Per decifrare un messaggio m data la chiave, un programma in assembler impiega un ciclo di 128 istruzioni ripetuto in media tante volte quanti sono i bit che costituiscono m. Impiegando un calcolatore che esegue un'operazione assembler in un tempo medio di 10 ns, indicare in ordine di grandezza quanti anni sarebbero necessari in media per condurre un attacco esaustivo sulle chiavi per un messaggio m di 1000 bit.
Indicare i calcoli eseguiti.

Svolgimento

$\#$ chiavi $= 2^{46}$

$\#$ operazioni per chiave $= 128 \cdot |m|$

$|m| = 1000$ bit $= 10^3$ bit

1 ns $= 10^{-9}$s

$\Rightarrow t = 2^{46} \cdot 128 \cdot 10^3 \cdot 10 \cdot 10^{-9}\text{s} = 2^{46} \cdot 2^7 \cdot 10^{-5}\text{s} = 2^{53} \cdot 10^{-5}\text{s} = 2^3 \cdot 2^{50} \cdot 10^{-5}\text{s}$ e dato che $2^{10} \simeq 10^3$
$\Rightarrow \simeq 2^8 \cdot (10^3)^5 \cdot 10^{-5}\text{s} = 8 \cdot 10^{10}\text{s} \simeq 2500$ anni

Esercizio 2 Si vuole generare una sequenza di bit pseudocasuali utilizzando il generatore BBS basato sulla legge:

$$x_i = x_{i-1}^2 \ mod\ n \quad b_i = 1 \Leftrightarrow x_{m-i} \text{ è } dispari$$

1. **Scegliere** $n = 11 \cdot 23$ e verificare che 11 e 23 soddisfano i requisiti del BBS
2. Sia M la propria matricola. Porre $y =$ M mod 100 e $x_0 = y^2$ mod n e **indicare** una sequenza di 10 bit generati, riportando i calcoli
3. **Discutere** se il generatore è crittograficamente sicuro

Svolgimento

1. Verificare che 11 mod 4 = 3 (ok) e 23 mod 4 = 3 (ok)
   Verificare che $2\lfloor\frac{11}{4}\rfloor + 1 = 2 \cdot 2 + 1 = 5$ e $2\lfloor\frac{23}{4}\rfloor + 1 = 11$ siano primi fra loro (sono primi entrambi, ok)
2. Usiamo M = 123456, M mod 100 = 56 e 11·23 = 253

   $x_0 = 56^2 \bmod 253 = 100 \to 0$
   $x_1 = 100^2 \bmod 253 = 133 \to 1$
   $x_2 = 133^2 \bmod 253 = 232 \to 0$
   $x_3 = 232^2 \bmod 253 = 188 \to 0$
   $x_4 = 188^2 \bmod 253 = 177 \to 1$
   $x_5 = 177^2 \bmod 253 = 210 \to 0$
   $x_6 = 210^2 \bmod 253 = 78 \to 0$
   $x_7 = 78^2 \bmod 253 = 12 \to 0$
   $x_8 = 12^2 \bmod 253 = 144 \to 0$
   $x_9 = 144^2 \bmod 253 = 243 \to 1$
   $\longrightarrow 1000010010$

Esercizio 3 Sia C una sequenza ottenuta rappresentando in binario ciascuna delle due cifre centrali del numero di matricola del candidato, prendendo per ciascuna di esse i tre bit meno significativi, concatenando questi due gruppi di bit e aggiungendo 1 in testa.

1. Eseguire $37^C \bmod 100$ per esponenziazioni successive **indicando** i calcoli eseguiti
2. **Spiegare** perché tale metodo di calcolo è considerato efficiente

Svolgimento Con M = 123456, 3 = 011 e 4 = 100
C = 1011100 = 64 + 16 + 8 + 4 = 92

1. $37^{92} \bmod 100 = 37^{64+16+8+4} \bmod 100$

   $37^4 \bmod 100 = 69^2 \bmod 100 = \underline{61}$
   $37^8 \bmod 100 = 61^2 \bmod 100 = \underline{21}$
   $37^{16} \bmod 100 = 21^2 \bmod 100 = \underline{41}$
   $37^{32} \bmod 100 = 41^2 \bmod 100 = 81$
   $37^{64} \bmod 100 = 81^2 \bmod 100 = \underline{61}$
   $\Rightarrow 37^{92} \bmod 100 = (61 \cdot 41 \cdot 21 \cdot 61) \bmod 100 = 81$

2. Perché esegue un numero di moltiplicazioni logaritmico nel valore dell'esponente (dunque lineare nella dimensione)

Esercizio 4 Applicando l'algoritmo Miller-Rabin, individuare un numero N primo di tre cifre decimali con probabilità di errore minore di $\frac{1}{50}$, spiegando il procedimento eseguito.

Svolgimento $N = 113$, ho $\frac{1}{4^k} < \frac{1}{50}$ per $k = 3$, quindi servono 3 testimoni y scelti a caso $\in [2, 112]$
Pongo $y = 3$

1. MCD(113, 3) = MCD(3, 113 mod 3) = MCD(2, 3 mod 2) = MCD(1, 0) = 1 ok
2. N = 113
   N-1 = 112 = $2^4 \cdot 7$, $w = 4$ e $z = 7$
   $3^7 \bmod 113 = 3^{1+2+4} \bmod 113 =$

   $3^2 \bmod 113 = 9$
   $3^4 \bmod 113 = 9^2 \bmod 113 = 81$

   $= (3 \cdot 9 \cdot 81) \bmod 113 = 40 \neq 1$, bisogna proseguire nella valutazione di P2
   $0 \leq i \leq w - 1 = 3$
   $i = 0$ $3^{2^0 \cdot 7} \bmod 113 = 3^7 \bmod 113 = 40 \neq -1$ (40 preso dal precedente risultato)
   $i = 1$ $3^{2^1 \cdot 7} \bmod 113 = (3^7)^2 \bmod 113 = 40^2 \bmod 113 = 18 \neq -1$ (40 preso dal precedente risultato)
   $i = 2$ $3^{2^2 \cdot 7} \bmod 113 = (18)^2 \bmod 113$ 98 $\neq -1$ (18 preso dal precedente risultato)
   $i = 3$ $3^{2^3 \cdot 7} \bmod 113 = (98)^2 \bmod 113$ -1 $= -1$ ok (98 preso dal precedente risultato)

Per gli altri 2 valori di y il procedimento è analogo

10.2 Cifrari Storici

Esercizio 1 **Decifrare** i seguenti crittogrammi

1. YMNXJCJWHNXJNXJFXD (Cifrario Cesare, chiave $k \neq 3$)
2. REXETSIH ONSICESI UCIFTFID REHTLIET (Cifrario a permutazione semplice, $h = 8$)

Svolgimento

1. Con $k = 21$ diventa THISEXERCISEISEASY
2. Con $\pi = \{5, 8, 7, 6, 4, 3, 2, 1\}$ diventa THISEXER CISEISNO TDIFFICU LTEITHER

Esercizio 2 Dato un cifrario affine (mod 26), si fa un attacco di tipo chosen plain-text usando il testo *hahaha*. Il testo cifrato è *nonono*. Determinare la funzione di cifratura.

Svolgimento Un cifrario affine è un cifrario che associa a pos(X) = a·pos(Y) + b
Bisogna trovare $a, b \mid \text{pos}(n) = a\cdot\text{pos}(h) + b \bmod 26 \wedge \text{pos}(o) = a\cdot\text{pos}(a) + b \bmod 26$
con pos(n) = 13, pos(h) = 7, pos(o) = 14 e pos(a) = 0
$\begin{cases} 13 = a \cdot 7 + b \ mod\ 26 \Leftrightarrow a = 11 \\ 14 = a \cdot 0 + b \ mod\ 26 \Leftrightarrow b = 14 \end{cases}$ ottentendo i due parametri $a = 11$ e $b = 14$ che danno il cifrario
pos(Y) = (11·pos(X) + 14) mod 26

Esercizio 3 Se nei cifrari affini si lavorasse in modulo 27 invece che modulo 26, quante sarebbero le chiavi possibili? E in modulo 29?

Svolgimento Con un alfabeto di 27 caratteri, a deve essere primo con $27 = 3^3$
$\rightarrow$ a può assumere i valori da 1 a 26 non multipli di 3
$\longrightarrow \phi(27) = \phi(3^3) = 2 \cdot 3^2 = 18$ (tramite la **funzione di Eulero** $\phi(p^k) = (p-1)p^{k-1}$ con p primo)
$\rightarrow$ b può assumere tutti i valori $\in [0, 26]$
$\Rightarrow$ # chiavi = 18·27 - 1 = 485 (il -1 esclude la coppia $(a, b) = (1, 0)$ che lascia invariato il messaggio)

Con un alfabeto di 29 caratteri: 29 è primo, quindi a può assumere tutti i valori $\in [1, 28]$ ($\rightarrow \phi(29) = 28$) e b tutti i valori $\in [0, 28]$
$\Rightarrow$ # chiavi $= 28 \cdot 29 - 1 = 811$

Esercizio 4 Questo esercizio ha lo scopo di dimostrare che un cifrario affine iterato ha la stessa sicurezza di un cifrario singolo.
Si considerino due cifrari affini:

$$C_1(x) = (a_1 \cdot x + b_1) \bmod 26$$

$$C_2(x) = (a_2 \cdot x + b_2) \bmod 26$$

Dimostrare che $\exists$ un cifrario affine $C_3 \mid C_3(x) = C_2(C_1(x))$

Svolgimento $C_2(C_1(x)) = (a_2 \cdot C_1(x) + b_2) \bmod 26 = (a_2 \cdot (a_1 + b_1) + b_2) \bmod 26$
$= (a_1a_2x + a_2b_1 + b_2) \bmod 26 = (a_3 \cdot x + b_3) \bmod 26$
Con $a_3 = a_1a_2 \bmod 26$, $b_3 = a_2b_1 + b_2 \bmod 26$

Esercizio 5 Il crittogramma c = MBR OJFGA SWNTE CNK QJDIL NURW MBR XHMR è ottenuto cifrando m = THE QUICK BROWN FOX JUMPS OVER THE GATE con un cifrario a sostituzione monoalfabetica completo.

1. Quanta informazione relativa alla chiave si può determinare conoscendo la coppia m, c?
2. Quante chiavi differenti potrebbero essere state usate per cifrare il messaggio m?
3. **Decifrare** il crittogramma MBR TRHLRP WHE HTHV CWND PNEYNE ZNN, che è stato cifrato usando la stessa chiave usata per cifrare m

Svolgimento

1. Parte della permutazione che definisce il cifrario.
   Più precisamente, si trova l'immagine cifrata di 22 caratteri su 26.
2. 24 chiavi
   Infatti mancano le corrispondenze per 4 caratteri, e se ne potrebbero costruire 4! = 24 differenti
3. THE WEASEL RUN AWAY FROM LONDON ZOO

Esercizio 6 Usando il metodo di Vigenère, **cifrare** il messaggio CRITTOGRAFIA impiegando come chiave le prime 4 lettere del proprio cognome. **Spiegare** inoltre come tale cifrario possa essere attaccano.

Svolgimento Si incrocia la lettera del messaggio con la lettera del crittogramma sulla tabella di Vigenère per trovare la lettera cifrata. Ad esempio, alla colonna C e riga M corrisponde la lettera O.

```
  C R I T T O G R A F I A
  M A T T M A T T M A T T
→ O A B M F O Z K M F B T
```

Per la spiegazione, vedi appunti.

Esercizio 7 Si deve cifrare il messaggio APPELLODIFEBBRAIO impiegando come chiave una permutazione arbitraria e segreta delle 26 lettere dell'alfabeto.

1. **Mostrare** la permutazione scelta e il crittogramma ottenuto
2. **Calcolare** il numero di prove necessarie per condurre un attacco esauriente sulle chiavi
3. **Discutere** la possibilità di un attacco più efficiente confrontandolo con quello del punto 2

Svolgimento

1. Ad esempio, DOAIIPFAPEREBBLOL
2. In generale 26! - 1
   Ma per decifrare il crittogramma ottenuto cifrando APPELLODIFEBBRAIO occorrono meno prove, perché il crittogramma contiene 10 lettere diverse tra loro da decifrare.
   $\Rightarrow$ # prove $= 26 \cdot 25 \cdot \ldots \cdot 17 = \frac{26!}{16!} \simeq 2 \cdot 10^{13}$
3. Crittoanalisi statistica

10.3 Cifrari Perfetti

Esercizio 1 Sia M il numero di matricola del candidato. Si converta M in una sequenza binaria B trasformando ordinatamente in binario ogni cifra decimale di M, prendendo per ciascuna di esse i tre bit meno significativi e concatenando tali gruppi di tre bit.

1. **Indicare** la sequenza B, **proporre** una chiave K di 18 bit ottenuta lanciando idealmente una moneta e **trasformare** B mediante One-Time Pad usando K
2. **Spiegare** se il cifrario può ritenersi sicuro per messaggi binari di lunghezza multipla di 18, utilizzando come chiave una ripetizione di K per il numero di volte necessario

Svolgimento

1. B = `101011000010101111`
   K = `010111001110111011`
   C $\Rightarrow$ `111100001100010100`
2. Il riutilizzo di K rende il procedimento indicato non sicuro quanto il one-time pad

Esercizio 2 In un cifrario A esistono un messaggio m e un crittogramma $c \mid \mathrm{P}(\mathrm{M} = m) = p \leq \frac{1}{4}$, $\mathrm{P}(\mathrm{C} = c) = 1 - p$ **Spiegare** se A può essere un cifrario perfetto e le conseguenze per un crittoanalista per la coppia m, c indicata.

Svolgimento

Esercizio 3 **Spiegare** con precisione matematica e proprietà di linguaggio perché il cifrario One-Time Pad su messaggi di n bit non può essere ritenuto perfetto se la chiave non è scelta perfettamente a caso.

Svolgimento Se ho una chiave con una qualche regolarità, lo xor farebbe filtrare tale proprietà sul crittogramma.

Esercizio 4 Nel codice one-time pad si sostituisca $\oplus$ con $\vee$ o con $\neg\oplus$.
Spiegare, nei due casi, se il protocollo funziona con le stesse proprietà del codice originale.

Svolgimento

Esercizio 5 Qual è lo svantaggio principale del cifrario one-time pad?

Svolgimento La chiave: deve essere perfettamente casuale e non venire riutilizzata.

Esercizio 6 Nel one-time pad si consideri una coppia arbitraria messaggio-crittogramma m, c di n bit.
Spiegare quanto vale P(M = m, C = c) (nota: è la probabilità dell'intersezione degli eventi, non quella condizionale)

Svolgimento

10.4 Cifrari Simmetrici

Esercizio 1 Sia C una sequenza ottenuta rappresentando in binario ciascuna delle due cifre centrali del numero di matricola del candidato, prendendo per ciascuna di esse i tre bit meno significativi e concatenando questi due gruppi di tre bit. Nella fase i-esima del DES, C costituisca la parte iniziale della sequenza in ingresso della S-box

1. **Indicare** la sequenza C
2. **Indicare**, con interi crescenti da 1 a 32, la sequenza POS di posizioni dei bit di D[i] influenzati da C, **spiegando** com'è stato ottenuto il risultato
3. Posto che la parte sinistra di S[$i-1$] del messaggio entrante nella fase i sia una sequenza di 1, **indicare** il valore dei bit di D[i] di cui al punto 2, **riportando** i calcoli eseguiti

Svolgimento

1. `000010`
2. Passando attraverso S_1, C dà in output `0100` (prendo i due bit esterni `00` per l'indice x e i rimanenti `0001` per l'indice y, trovando 04 nella tabella che in binario è `0100`)
   Questi bit andranno rispettivamente in posizione 9, 17, 23 e 30 di D[i], seguendo la tabella P (01 è in posizione 9 e così via). Quindi D[9] = 0, D[17] = 1, D[23] = 0 e D[30] = 0
3. Con S[$i-1$] eseguo lo $\oplus$ (xor) con i bit in uscita da P. Quindi D[9] = $1\oplus0 = 1$, D[17] = $1\oplus1 = 0$,
   D[23] = $1\oplus0 = 1$ e D[30] = $1\oplus0 = 1$

Esercizio 2 Si trasformi il DES complementando tutte le uscite della s-box.
Sia M il proprio numero di matricola. Si converta M in una sequenza binaria B trasformando ordinatamente in binario ogni cifra decimale di M e prendendo per ciascuna i tre bit meno significativi. Si estenda B fino ad avere 48 bit, aggiungendo zeri a destra: sia tale sequenza l'uscita del blocco EP del DES con chiave $k[0] = 1010\ldots10$

1. **Indicare** la sequenza B
2. Nella fase 1 del DES, **determinare** il valore dei primi 4 bit a sinistra in uscita dalla s-box, spiegando come è stato ottenuto il risultato
3. **Commentare** se il DES così modificato appaia o meno palesemente meno sicuro della versione standard

Svolgimento

1. $B =$ `101 011 000 010 101 111 000 000 000 000 000 000 000 000 000 000`
2. Nella S_i entrano 6 bit, quindi per vedere i primi 4 bit prodotti bisogna usare la S_1 con i primi 6 bit di B, cioè con `1 0101 1`. Con i bit esterni si indicizza x, in questo caso con valore 3 uso la quarta riga, e con i 4 bit interni si indicizza y, in questo caso con valore 5, ottenendo il valore 09, che quindi consiste nei bit `1001`. Per la modifica proposta, bisogna complementare ogni bit, ottenendo `0110`.
3.

Esercizio 3 Nella fase i-esima del DES, siano `000011` i primi 6 bit a sinistra in ingresso alla s-box.

1. **Indicare** (con interi crescenti tra 1 e 32) la sequenza POS di posizioni dei bit di S$[i+1]$ influenzati da questi 6 bit, spiegando come è stato ottenuto il risultato.

2. Siano $c_1c_2\ldots$ i bit di S$[i-1]$ nelle posizioni indicate in POS (le stesse quindi di S$[i+1]$). $c_1c_2\ldots$ rappresentino in binario il numero di consonanti presenti nel nome e cognome del candidato. **Indicare** il valore dei bit di S$[i+1]$ nelle posizioni POS spiegando come è stato ottenuto il risultato.

Svolgimento

1. Avendo `1 0001 1` come bit, ottengo 12 cioè `1100` come bit.
   Ogni bit andrà, secondo la tabella P, rispettivamente in posizione 9, 17, 23, 31 di D$[i]$, che diventerà S$[i+1]$.

2. La sequenza data da c è `1000`.
   Bisogna fare lo $\oplus$ tra S$[i-1]$ = `1000` e D attuale = `1100`. Quindi

   $$\text{S}[9] = 1 \oplus 1 = 0$$
   $$\text{S}[17] = 0 \oplus 1 = 1$$
   $$\text{S}[23] = 0 \oplus 0 = 0$$
   $$\text{S}[31] = 0 \oplus 0 = 0$$

   ottenendo `0 1 0 0` rispettivamente nelle posizioni 9, 17, 23, 31.

Esercizio 4 La s-box del cifrario AES è composta da 16 blocchi con 8 ingressi e 8 uscite ciascuno. Qual è il numero totale di possibili funzioni che si potrebbero scegliere per costruire ciascun blocco?

Svolgimento Per ognuna delle 2^8 configurazioni di input, posso scegliere 2^8 configurazioni in output. Quindi ho # funzioni = $\left(2^8\right)^{2^8} = 2^{8\cdot 2^8}$

Esercizio 6 La proprietà di non linearità di qualsiasi cifratura a blocchi è fondamentale per la sicurezza. Infatti, si supponga di avere una cifratura lineare a blocchi Cl che figra blocchi di 128 bit di testo in chiaro in 128 bit di testo cifrato, usando una chiave k la cui lunghezza è irrilevante. Dunque, ogni coppia di messaggi m_1 e m_2 risulta $Cl(m_1 \oplus m_2, k) = Cl(m_1, k) \oplus Cl(m_2, k)$
Descrivere come un avversario che abbia 128 testi cifrati scelti possa decifrare qualsiasi testo cifrato senza conoscere la chiave k.

Svolgimento Prendo c il crittogramma di un generico m, $c = Cl(m, k)$. Posso rappresentare $c = \bigoplus_{i:c_i=1} e^{(i)}$ dove $e_{(i)}$ ha un solo bit a 1 in posizione i, e gli altri 127 bit a 0.
Per esempio $c = 1011 = 1000 \oplus 0010 \oplus 0001 = e^{(1)} \oplus e^{(3)} \oplus e^{(4)}$
Sfruttiamo le proprietà delle funzioni di cifratura e decifrazione: $m = D(c, k) = D(\bigoplus_{i:c_i=1} e^{(i)}, k) = \bigoplus_{i:c_i=1} D(e^{(i)}, k)$
Si chiede la decifrazione dei 128 testi cifrati $e^{(i)}$, $i \in [1, 128]$, $f^{(i)} = D(e^{(i)}, k)$
Si può decifrare qualsiasi crittogramma c anche senza conoscere k: $m = \bigoplus_{i:c_i=1} f^{(i)}$

Esercizio 7 DESX è un cifrario proposto da Rivest per proteggere il DES dagli attacchi esaurienti. DESX usa una chiave segreta w di 64 bit, oltre alla chiave DES k di 56 bit, e opera così:

$$C_{DESX}(m, k, w) = w \oplus C_{DES}(m \oplus w, k)$$

Mostrare come eseguire la decifrazione.

Svolgimento $c = C_{DES}(m \oplus w, k) \Leftrightarrow c \oplus w = C_{DES}(m \oplus w, k)$
$\Leftrightarrow D_{DES}(c \oplus w, k) = D_{DES}(C_{DES}(m \oplus w, k))$ ma $D_{DES}(C_{DES}(m \oplus w, k)) = m \oplus w$ perché DES è un cifrario corretto
Quindi $\Leftrightarrow m \oplus w = D_{DES}(c \oplus w, k) \Leftrightarrow m = w \oplus D_{DES}(c \oplus w, k)$

10.5 Cifrari Asimmetrici

Esercizio

1. **Spiegare** in cosa consiste il cifrario RSA e **dimostrarne** la correttezza.
2. **Darne un esempio** di applicazione impiegando parametri numerici molto piccoli per cifrare il messaggio costituito dalle due cifra meno significative del proprio numero di matricola.

Svolgimento

1. Si basa sulla moltiplicazione di due numeri primi, facile da eseguire ma difficile da invertire, e fa uso dell'algebra modulare che rende difficili alcuni problemi facili nell'algebra non modulare.
   Scelti p, q primi, la cifratura è eseguita con la chiave pubblica del destinatario, k_{pub} composta da $e < (p-1)(q-1)$ coprimo con $n = pq$ e da n. La decifrazione è fatta dal destinatario con la proprio chiave privata $k_{priv} = d = e^{-1} \mod (p-1)(q-1)$. Vedi p. 43.
2. Usando $p = 3$, $q = 5$, cifro $m = 13$
   $n = 15$, $\phi(n) = (p-1)(q-1) = 2 \cdot 4 = 8$ quindi posso scegliere $e = 7$ che rispetta le condizioni.
   $d = 7^{-1} \mod 8 = 7$ perché $7 \cdot 7 \mod 8 = 49 \mod 8 = 1$
   $c = C(m, k_{pub}) = m^e \mod n = 13^7 \mod 15 = 7$
   $m = D(c, k_{priv}) = c^d \mod n = 7^7 \mod 15 = 13$

Esercizio 2 Poniamo che si scopra un algoritmo polinomiale per calcolare la funzione di Eulero. **Spiegare** in termini matematici quale influenza la scoperta avrebbe sul cifrario RSA.

Svolgimento Si potrebbe calcolare in tempo polinomiale la chiave privata $d = e^{-1} \mod \phi(n)$. Il cifrario sarebbe compromesso e non più utilizzabile.

Esercizio 3 Per la costruzione di una coppia di chiavi RSA si sceglie n come prodotto di due primi p, q considerando le seguenti possibilità:

1. $p = \Theta(n^{1/2}), q = \Theta(n^{1/2})$
2. $p = O(n^{1/3}), q = O(n^{2/3})$
3. $p = \Theta(n^{1/3}), q = \Theta(n^{1/3})$
4. $p = O(\log n), q = O(n/\log n)$

Per ciascuna di queste possibilità, **spiegare** con precisione se la scelta è corretta e consigliabile.

Svolgimento

1. No, p, q troppo vicini
2. Ok, p, q sufficientemente grandi e distanti tra loro
3. No, $p \cdot q$ non è $\Theta(n)$
4. p troppo piccolo, bruteforce avrebbe costo polinomiale

Esercizio 4 Si consideri un cifrario RSA con $p = 7, q = 11, e = 13$

1. **Determinare** il valore di d chiave privata
2. Qual'è la dimensione dei blocchi per la cifratura?
3. **Cifrare** `100011001010`

Svolgimento

1. $n = 77$, $\phi(n) = 60$, quindi $d = 13^{-1} \mod 60 = 37$
2. I blocchi saranno di $b = \lfloor \log_2 n \rfloor = 6$ bit
3. Diviso in blocchi, $m =$ `100011 001010`, cioè cifro $m_1 = 35$ e $m_2 = 10$
   $c_1 = m_1^e \mod n = 35^{13} \mod 77 = 63$
   $c_2 = m_2^e \mod n = 10^{13} \mod 77 = 10$

Esercizio 5 Si consideri un cifrario RSA con chiave pubblica $n = 55, e = 7$

1. **Cifrare** $m = 10$
2. **Forzare** il cifrario trovando p, q, d
3. **Decifrare** $c = 35$

Svolgimento

1. $c = m^e \bmod n = 10^7 \bmod 55 = 10$
2. $n = 55$ è il prodotto tra $p = 5$ e $q = 11$ quindi $\phi(55) = (p-1)(q-1) = 4 \cdot 10 = 40$ e $d = 7^{-1} \bmod 40 = 23$
3. $m = c^d \bmod n = 35^{23} \bmod 55 = 30$

Esercizio 6 Siano x, y, n tre interi positivi arbitrari con $x, y < n$. Poniamo che si scopra un algoritmo di algebra modulare di complessità $O(d^2)$ per calcolare, se esiste, il logaritmo discreto di y in base x modulo n, con $d = \Theta(n)$ oppure $d = \Theta(\log n)$.
Spiegare in termini matematici, per i due suddetti valori di d, quale influenza la scoperta potrebbe avere sull'algoritmo DH per lo scambio segreto delle chiavi.

Svolgimento Il crittoanalista potrebbe arrivare a conoscere $A = g^x \bmod p$ con g generatore di Z_p^*, o analogamente $B = g^y \bmod p$.
Avere un algoritmo efficiente per la risoluzione permetterebbe di trovare x o y in modo efficiente e renderebbe DH vulnerabile ad attacchi passivi.

$d = \Theta(n)$ L'algoritmo ha complessità esponenziale nella dimensione dell'input (polinomiale solo nel *valore* di n). Nessuna influenza su DH che si può sempre usare in sicurezza

$d = \Theta(\log n)$ L'algoritmo è effettivamente polinomiale, quindi il protocollo DH non è più utilizzabile.

Esercizio 7 Si consideri il protocollo basato sull'algoritmo DH con $g = 3, p = 353$ e siano $x = 97, y = 233$. Calcolare X, Y, k

Svolgimento

$X = g^x \bmod p = 3^{97} \bmod 353 = 40$

$Y = g^y \bmod p = 3^{233} \bmod 353 = 248$

`k[session]` $= g^{xy} \bmod p = 3^{97 \cdot 233} \bmod p = 160$

Esercizio 8 Due utenti A e B vogliono costruire una chiave segreta di sessione impiegano il protocollo basato sull'algoritmo DH. A tale scopo concordano su una coppia pubblica di interi $p = 11, g = 6$

1. **Dimostrare** che la coppia $\langle 11, 6 \rangle$ è adatta per il protocollo DH
2. Posto che A,B scelgano come numeri casuali segreti x, y la quarta e quinta cifra della matricola del candidato, **creare** `k[session]` **indicando i calcoli eseguiti da A e B**

Svolgimento

1. p, g proposti pari a $p = 11$ e $g = 6$, sono adatti perché p è primo e g è un generatore di $Z_{11}^* = \{1, 2, \ldots, 10\}$ $= \{6^k \bmod 11 \mid 1 \leq k \leq 10\}$

   $k = 1$ $6^1 \bmod 11 = 6$ $\quad$ $k = 6$ $6^6 \bmod 11 = 5$
   $k = 2$ $6^2 \bmod 11 = 3$ $\quad$ $k = 7$ $6^7 \bmod 11 = 8$
   $k = 3$ $6^3 \bmod 11 = 7$ $\quad$ $k = 8$ $6^8 \bmod 11 = 4$
   $k = 4$ $6^4 \bmod 11 = 9$ $\quad$ $k = 9$ $6^9 \bmod 11 = 2$
   $k = 5$ $6^5 \bmod 11 = 10$ $\quad$ $k = 10$ $6^{10} \bmod 11 = 1$

2. A sceglie $x = 2$ e B sceglie $y = 5$
   A calcola $X = 6^2 \bmod 11 = 3$, B calcola $Y = 6^5 \bmod 11 = 10$ e se li scambiano.
   A calcolerà `k[session]` $= Y^x \bmod p$ e B calcolerà `k[session]` $= X^y \bmod p$
   In entrambi i casi, `k[session]` $= g^{xy} \bmod p = 6^{10} \bmod 11 = 1$

Esercizio 9 Considerando il cifrario RSA:

1. Discutere se è possibile scegliere un valore pari di e

2. Siano e, e' due valori scelti per la chiave pubblica tali che e' è ottenuto da e cambiando un bit da 0 a 1. Dimostrare che $\text{MCD}(e, e') = 1$

Svolgimento

1. e pari significa $e = 2 \cdot f$ con f dispari.
   Dato che $\phi(n)$ è ottenuto con il prodotto $(p-1)(q-1)$ con p, q primi, ho che entrambi i fattori sono pari, diciamo $p - 1 = 2r, q - 1 = 2s$, di conseguenza avrò $\text{MCD}(2f, 4rs) = 2 \neq 1$

2. Sia $e' = e + 2^i$ (i-esimo bit di e è 0)
   $\forall\, t \mid \ \ t \mid e$ sappiamo che $t \neq 2$ (vedi punto 1), dunque $t \not| 2^i$
   $\Rightarrow \forall\, t \ \ t \mid e, t \not| e' \Rightarrow \text{MCD}(e, e') = 1$

Esercizio 10 Nonostante il cifrario RSA sia sicuro, alcune sue implementazioni possono rendere insicura la cifratura. Si consideri ad esempio la cifratura di un messaggio m di 64 bit con una chiave pubblica RSA $\langle e, n\rangle$ dove $e = 3$ e n è un numero da 512 bit.

1. **Spiegare** perché la cifratura di m è completamente insicura

2. **Decifrare** il crittogramma $c = 33076161$ con $n = 100082119$

Svolgimento

1. Il motivo principale è che n ha troppi pochi bit ed è attualmente fattorizzabile in tempi molto brevi.
   Inoltre, un messaggio m da 64 bit con $e = 3$ non viene ridotto in modulo per un n da 512 bit (m^e ha $64 \cdot 3 = 192$ bit). Avrò sempre $c = m^3$.

2. $d = 3^{-1} \bmod \phi(100082119)$, ma è inutile dato che il messaggio m non è ridotto in modulo, posso direttamente calcolare $m = \sqrt[e]{c} = \sqrt[3]{33076161} = 321$

Esercizio 11 Supponiamo che Eve intercetti il crittogramma $c = m^e \bmod n$ diretto ad Alice. Si supponga inoltre che Alice sia disposta a decifrare per Eve qualsiasi crittogramma c', a patto che $c' \neq c$.
Descrivere come Eve possa decifrare m in tempo polinomiale, richiedendo ad Alice la decifrazione del crittogramma $c' = cx^e$ dove $x < n$ è un intero casuale coprimo con n.

Svolgimento Avendo $c' = cx^e = m^e x^e \bmod n$ e $c = m^e \bmod n$, ho $m' = (m^e)^d (x^e)^d \bmod n = mx$,
da cui ottengo $m = m'x^{-1} \bmod n$

Esercizio 12 Alice vuole mandare un messaggio cifrato a Bob usando RSA, ma non conosce la chiave pubblica. Invia una mail a Bob chiedendogliela, che risponde inviando $K_{pub} = \langle e, n\rangle$.
Eve intercetta il messaggio, sostituisce e con un nuovo intero e' coprimo con e, e invia la chiave modificata $k'_{pub} = \langle e', n\rangle$ ad Alice.
Alice usa k'_{pub} per cifrare m, e invia il crittogramma $c' = m^{e'} \bmod n$ a Bob.
Dato che m è stato cifrato con la chiave sbagliata, Bob rimanda la propria chiave pubblica ad Alice che risponderà con il crittogramma corretto $c = m^e \bmod n$.
Mostrare come Eve, che conosce e, e', c, c', possa risalire a m.

Svolgimento e, e' sono coprimi. Applicando EE (Euclide Esteso) si possono calcolare in tempo polinomiale $r, s \mid e \cdot r + e' \cdot s = \text{MCD}(e, e') = 1$
$\Rightarrow m = m^{er+e's} \bmod n = (m^{er} \bmod n) \cdot (m^{e's} \bmod n) \bmod n = c^r (c')^s \bmod n$
Supponendo $r < 0, s > 0 \Rightarrow m = (c^{-1})^{-r} (c')^s \bmod n$
Eve calcola l'inverso di c, calcola $(c^{-1})^{-r}$ ($-r$ è positivo), calcola $(c')^s$ e ottiene m dal loro prodotto ridotto in modulo. Tutti i passaggi richiedono tempo polinomiale. L'inverso di c modulo n esiste ed è unico se c, n sono coprimi. Se non lo fossero, Eve potrebbe allora fattorizzare n e forzare il cifrario.

10.6 Funzioni hash, MAC e Firma digitale

Esercizio 1 Spiegare che proprietà devono possedere le funzioni hash one-way e perché tali funzioni sono importanti nei protocolli di autenticazione e firma.

Svolgimento Una funzione hash one-way deve essere **facile da calcolare**, **difficile da invertire** e **difficile trovare due elementi che collidono**.

Esercizio 2 Si scriva un messaggio a piacere in italiano $m = m_{20}m_{19}\dots m_0$ costituito da 21 caratteri alfabetici più lo spazio.
Si consideri il sottoinsieme dell'alfabeto $C_0 = \{A, B, \dots, L\}$.
Utilizzando la chiave $k = k_5k_4k_3k_2k_1k_0$ consistente nelle 6 cifre decimali della propria matricola, si autentichi m mediante il MAC di 6 bit $A(m,k) = h_5h_4h_3h_2h_1h_0$ costruito come segue:

```
j = 0
for (i = 0 to 5) do
        j = k[i] + j mod 21
        if (m[j] in C0) then h[i] = 0 else h[i] = 1
```

1. **Riportare** i valori di m e h_i per $i = 0, 1, \dots, 5$ indicando i calcoli

2. **Spiegare** se la funzione A definita sopra è adatta per l'applicazione considerata.

Svolgimento

1. $m =$ `messaggio celatissimo`, $k =$ `530257`

   $i = 0$ $j = 7 + 0 \bmod 21 = 7$, $m_7 = a \Rightarrow h_0 = 0$

   $i = 1$ $j = 5 + 7 \bmod 21 = 12$, $m_{12} = o \Rightarrow h_1 = 1$

   $i = 2$ $j = 2 + 12 \bmod 21 = 14$, $m_{14} = g \Rightarrow h_2 = 0$

   $i = 3$ $j = 14 + 0 \bmod 21 = 14$, $m_{14} = g \Rightarrow h_3 = 0$

   $i = 4$ $j = 3 + 14 \bmod 21 = 17$, $m_{17} = s \Rightarrow h_4 = 1$

   $i = 5$ $j = 5 + 17 \bmod 21 = 1$, $m_1 = m \Rightarrow h_5 = 1$

   $\Rightarrow$ $A(m,k) =$ `110010`

2.

Esercizio 3 Sia S la somma delle sei cifre decimali del numero di matricola del candidato. Sia ponga $M = S + 10$. Si convertano le cifre di M in binario su 4 bit, se le calcoli lo XOR e si riconverta il valore ottenuto in un numero decimale H che sarà usato come hash di M: $h(M) = H$
Per due utenti Alice e Bob di un sistema RSA, si considerino i seguenti parametri:

Alice: $p = 5, q = 11, e = 7, d = 23$

Bob: $p = 7, q = 13, e = 5, d = 29$

Alice deve spedire a Bob il messaggio M cifrato e firmato in hash, impiegando le chiavi RSA e la funzione hash di cui sopra.

1. **Spiegare** se i parametri RSA indicati sopra sono scelte in modo consistente con le regole del cifrario (a parte le dimensioni)

2. **Indicare** esplicitamente tutte le operazioni aritmetiche fatte da Alice e Bob nella trasmissione e verifica del messaggio M e della firma.

Svolgimento

1. Per entrambi, abbiamo p, q primi ed e coprimo con $\phi(pq) = (p-1)(q-1)$:

   $\text{MCD}(7, 40) = 1$

   $\text{MCD}(5, 72) = 1$

   Inoltre $23 = 7^{-1} \bmod 40$ e $29 = 5^{-1} \bmod 72$
   Quindi i valori sono appropriati a meno delle dimensioni.

2. Usando il protocollo 3

   Alice
   $M = 22 + 10 = 32 \to H = \texttt{0011} \oplus \texttt{0010} = \texttt{0001} \to h(32) = 1$
   Calcola il crittogramma $c_m = 32^5 \bmod 91 = 2$ con la chiave pubblica di Bob
   Calcola il crittogramma $c_H = 1^{23} \bmod 55 = 1$ con la propria chiave privata
   Spedisce c_m e c_H a Bob

   Bob
   Riceve c_m e calcola $m = 2^{29} \bmod 91 = 32$ con la propria chiave privata
   Riceve c_H e calcola $H_{Alice} = 1^7 \bmod 55 = 1$ con la chiave pubblica di Alice
   Calcola H con il precedente calcolo, lo verifica con H_{Alice} ricevuto e conclude che il messaggio è autenticato.

Esercizio 4 Posto che si scopra un algoritmo polinomiale per calcolare la funzione di Eulero, **spiegare** in termini matematici quale influenza la scoperta avrebbe sui protocolli di firma.

Svolgimento Come per l'RSA, la scoperta di un algoritmo polinomiale per calcolare $\phi(n)$ si potrebbe calcolare in tempo polinomiale $d = e^{-1} \bmod \phi(n)$, rendendo chiunque capace di firmare al posto dell'utente.

Esercizio 5 Sia $n = pq$ con p, q primi, e sia e un intero coprimo con $\phi(n)$. Si discuta se la funzione $h(m_1, m_2) = m_1^e m_2^e \bmod n$ è resistente alle collisioni.

Svolgimento Non è resistente. Per dimostrarlo si prende $(m_1', m_2') = (m_2, m_1)$ e si ottiene una collisione con la coppia (m_1, m_2), per la transitività della moltiplicazione.

Esercizio 6 Due utenti A e B di un sistema RSA hanno scelto le seguenti chiavi: `k[pubA] =` $\langle 7, 341\rangle$, `k[privA] =` $\langle 43\rangle$, `k[pubB] =` $\langle 5, 299\rangle$, `k[privB] =` $\langle 53\rangle$.
A deve mandare a B il seguento messaggio m cifrato e firmato in hash, impiegando RSA e la seguente funzione hash h:

m = numero di matricola del candidato, diviso in 3 blocchi m_1, m_2, m_3 di due cifra ciascuno

$h(m) = (m_1 + m_2 + m_3) \bmod 100$

1. Spiegare se le chiavi di A e B sono scelte in modo consistente con le regole del cifrario (a parte le loro dimensioni), elencando i calcoli eseguiti.
2. Indicare esplicitamente tutte le operazioni aritmetiche eseguite da A e B nella trasmissione e verifica del messaggio m e della firma.

Svolgimento

1. A $n = 341 = pq = 11 \cdot 31 \Rightarrow \phi(n) = 300$
   $\text{MCD}(7, 300) = 1$ OK
   $d = 7^{-1} \bmod 300 = 43$ OK

   B $n = 299 = pq = 13 \cdot 23 \Rightarrow \phi(n) = 264$
   $\text{MCD}(5, 264) = 1$ OK
   $d = 5^{-1} \bmod 264 = 53$ OK

2. $m_1 = 53, m_2 = 02, m_3 = 57$
   Firma e Cifratura
   A calcola $h(m) = 53 + 2 + 57 \bmod 100 = 14$
   A calcola la firma $f = D(h(m),$ `k[privA]`$) = 14^{43} \bmod 341 = 214$
   A calcola i crittogrammi $c_i = C(m_i,$ `k[pubB]`$) \Rightarrow$ $c_1 = m_1^e \ mod\ 299 = 53^5 \ mod\ 299 = 40$; $c_2 = m_2^e \ mod\ 299 = 2^5 \ mod\ 299 = 32$; $c_3 = m_3^e \ mod\ 299 = 57^5 \ mod\ 299 = 5$
   Decifrazione e verifica
   B decifra i crittogrammi $m_i = D(c_i$ `k[privB]`$) \Rightarrow$ $m_1 = c_1^d \ mod\ 299 = 40^{53} \ mod\ 299 = 53$; $m_2 = c_2^d \ mod\ 299 = 32^{53} \ mod\ 299 = 2$; $m_3 = c_3^d \ mod\ 299 = 5^{53} \ mod\ 299 = 57$
   B calcola $h(m) = (m_1 + m_2 + m_3) \bmod 100 = 14$
   B verifica che $14 = C(f,$`k[pubA]`$) = 214^7 \bmod 341 = 14$

Esercizio 8 Si presenta un primo tentativo di firma elettronica basato su curve ellittiche. Si ha una curva ellittica globale, un numero primo p e un *generatore* B. Alice sceglie una chiave di firma privata x_A e crea la chiave pubblica di verifica $Y_A = x_A B$. Per firmare il messaggio M:

Alice sceglie un valore k

Alice invia a Bob M, k e la firma $F = M - kx_A B$

Bob verifica che $M = F + kY_A$

1. Dimostrare che questo schema funziona correttamente. Ovvero, che il processo di verifica produce un'uguaglianza quando la firma è valida.
2. Dimostrare che lo schema è inaccettabile descrivendo una semplice tecnica per creare la firma falsa di un utente su un qualsiasi messaggio.

Svolgimento

1. $M = F + kY_A = M - kx_A B + kY_A = M - \cancel{k}x_A B + \cancel{k}(x_A B) = M$
2. $F = M - kx_A B = M - kY_A$
   k viene scelto da chi firma, e Y_A è la chiave **pubblica** di chi firma

Esercizio 9 Si presenta un tentativo di firma elettronica basato su curve ellittiche. Si ha una curva ellittica globale, un numero primo p e un *generatore* B. Alice sceglie una chiave di firma privata x_A e crea la chiave pubblica $Y_A = x_A B$. Per firmare un messaggio M:

Bob sceglie un valore k

Bob invia ad alice $C = kB$

Alice invia a Bob M e la firma $F = M - x_A C$

Bob verifica che $M = F + kY_A$

1. Dimostrare che questo schema funziona correttamente. Ovvero, che il processo di verifica produce un'uguaglianza quando la firma è valida.

2. Dimostrare che falsificare una firma con questo schema è difficile quanto forzare la crittografia a curva ellittica El Gamal

Svolgimento

1. $M = F + kY_A = M - x_A C + kY_A = M - \not{x_A} kB + \not{k} x_A B = M$

2. Per falsificare la firma bisogna calcolare $x_A C = x_A kB = kY_A$
   Y_A pubblico, ma k non è noto. Per ricavarlo da $C = kB$ occorre risolvere il logaritmo discreto su curve ellittiche. L'alternativa consiste nel calcolare la chiave privata x_A da Y_A, per cui occorre di nuovo risolvere il problema del logaritmo discreto su curve ellittiche.

10.7 Curve Ellittiche

Esercizio 1 Il punto P = (4,7) appartiene alla curva ellittica $y^2 = x^3 - 5x + 5$ sui numeri reali?

Svolgimento Verificare se $7^2 = 4^3 - 5 \cdot 4 + 5 \Leftrightarrow 49 = 64 - 20 + 5 \Leftrightarrow 49 = 49$, quindi P appartiene alla curva

Esercizio 2 Nella curva ellittica sui reali $y^2 = x^3 - 36x$, siano P = (-3,9) e Q = (-2,8).
Trovare P + Q, 2P.

Svolgimento Formule a p.48

$$P + Q = S = (x_s, y_s) \text{ con } \begin{cases} x_s = \left(\frac{9-8}{-3+2}\right)^2 + 3 + 2 = 6 \\ y_s = -9 + \left(\frac{9-8}{-3+2}\right)(-3 - x_s) = -9 + (-1)(-9) = 0 \end{cases} = (6, 0)$$

$$2P = T = (x_t, y_t) \text{ con } \begin{cases} x_t = \left(\frac{27-36}{18}\right)^2 + 3 + 3 = \frac{25}{4} \\ y_t = -9 + \left(\frac{27-36}{18}\right)(-3 - x_t) = -\frac{35}{8} \end{cases} = \left(\frac{25}{4}, -\frac{35}{8}\right)$$

Esercizio 3 La curva ellittica di equazione $y^2 = x^3 + 10x + 5$ definisce un gruppo su Z_{17}?

Svolgimento Bisogna verificare che $4a^3 + 27b^2 \bmod p \neq 0$
In questo caso, $4 \cdot 10^3 + 27 \cdot 5^2 \bmod 17 = 5 + 12 \bmod 17 = 0$, quindi non definisce un gruppo su Z_{17}

Esercizio 4 Determinare i punti appartenenti alla curva ellittica $E_{11}(1,6)$

Svolgimento La curva è $E_{11}(1,6)$ cioè data dall'equazione $y^2 = x^3 + x + 6 \bmod 11$
Partiamo identificando i residui quadratici, cioè i valori di y e $y^2 \bmod 11$

$y = 0 \Rightarrow y^2 = 0$

$y = 1 \Rightarrow y^2 = 1$

$y = 2 \Rightarrow y^2 = 4$

$y = 3 \Rightarrow y^2 = 9$

$y = 4 \Rightarrow y^2 = 5$

$y = 5 \Rightarrow y^2 = 3$

Simmetria

$y = 6 \Rightarrow y^2 = 3$

$y = 7 \Rightarrow y^2 = 5$

$y = 8 \Rightarrow y^2 = 9$

$y = 9 \Rightarrow y^2 = 4$

$y = 10 \Rightarrow y^2 = 1$

Quindi i residui quadratici sono $0, 1, 3, 4, 5, 9$ Ora possiamo calcolare i valori, le soluzioni esistono solo se esistono i residui quadratici di cui sopra

$x = 0 \longrightarrow y^2 = 0 + 0 + 6 = 6 \Rightarrow$ nessuna soluzione

$x = 1 \longrightarrow y^2 = 1 + 1 + 6 = 8 \Rightarrow$ nessuna soluzione

$x = 2 \longrightarrow y^2 = 8{+}2{+}6 = 16 \bmod 11 = 5 \Rightarrow y = 4, 7$
$\Longrightarrow (2,4), (2,7)$

$x = 3 \longrightarrow y^2 = 27 + 3 + 6 = 3 \Rightarrow y = 5, 6$
$\Longrightarrow (3,5), (3,6)$

$x = 4 \longrightarrow y^2 = 8 \Rightarrow$ nessuna soluzione

$x = 5 \longrightarrow y^2 = 4 \Rightarrow y = 2, 9$
$\Longrightarrow (5,2), (5,9)$

$x = 6 \longrightarrow y^2 = 8 \Rightarrow$ nessuna soluzione

$x = 7 \longrightarrow y^2 = 4 \Rightarrow y = 2, 9$
$\Longrightarrow (7,2), (7,9)$

$x = 8 \longrightarrow y^2 = 9 \Rightarrow y = 3, 8$
$\Longrightarrow (8,3), (8,8)$

$x = 9 \longrightarrow y^2 = 7 \Rightarrow$ nessuna soluzione

$x = 10 \longrightarrow y^2 = 4 \Rightarrow y = 2, 9$
$\Longrightarrow (10,2), (10,9)$

Quindi $E_{11}(1,6) = \{(2,4), (2,7), (3,5), (3,6), (5,2), (5,9), (7,2), (7,9), (8,3), (8,8), (10,2), (10,9)\} \cup O$
Sono 13 elementi, quindi **l'ordine della curva è 13**.

Esercizio 5 Calcolare gli opposti dei seguenti punti su curva ellittica Z_{17}: $P = (5,8), Q = (3,0), R = (0,6)$

Svolgimento $-P = (5, -8 \bmod 17) = (5,9)$, $-Q = (3,0)$, $-R = (0, -6 \bmod 17) = (0,11)$

Esercizio 6 Nella curva ellittica $E_{17}(1,7)$ siano $P = (1,3)$ e $Q = (2,0)$. Trovare $P + Q$ e $2P$

Svolgimento La curva è di equazione $y^2 = x^3 + x + 7$ con $a = 1, b = 7$ e $p = 17$ Formule a p. 48

$$P + Q = S = (x_s, y_s) \text{ con } \begin{cases} x_s = \lambda^2 - 1 - 2 \ mod\ 17 = 9 - 3 \ mod\ 17 = 6 \\ y_s = -3 + \lambda(1 - x_s) \ mod\ 17 = -3 + 14 \cdot (-5) = -3 + 15 \ mod\ 17 = 12 \\ \lambda = \frac{0-3}{2-1} \ mod\ 17 = -3 \ mod\ 17 = 14 \end{cases} = (6,12)$$

Ricordando $6^{-1} \bmod 17 = x \mid 6 \cdot x \bmod 17 = 1 \Leftrightarrow x = 3$

$$2P = T = (x_t, y_t) \text{ con } \begin{cases} x_t = \lambda^2 - 1 - 1 \ mod\ 17 = 8 - 2 = 6 \\ y_t = -3 + 12(1 - x_s) \ mod\ 17 = -3 + 12 \cdot (-5) \ mod\ 17 = -3 + 8 = 5 \\ \lambda = \frac{3 \cdot (1)^2 + 1}{2 \cdot 3} \ mod\ 17 = \frac{4}{6} \ mod\ 17 = 4 \cdot 6^{-1} \ mod\ 17 = 4 \cdot 3 \ mod\ 17 = 12 \end{cases} = (6,5)$$

Esercizio 7 Nella curva ellittica $E_{23}(14, 12)$, sia $P = (1, 2)$. Calcolare $11P$

Svolgimento $11P = (8 + 2 + 1)P = 8P + 2P + P$, quindi calcolo $2P, 4P = 2(2P)$ e $8P = 2(4P)$

$$2P = \begin{cases} \lambda = \frac{3 \cdot 1^2 + 14}{2 \cdot 2} \ mod\ 23 = \frac{17}{4} \ mod\ 23 = 17 \cdot 4^{-1} \ mod\ 23 = 17 * 6 \ mod\ 23 = 10 \\ x_{2P} = \lambda^2 - 2 \cdot 1 \ mod\ 23 = 8 - 2 = 6 \\ y_{2P} = \lambda(1 - x_{2P}) - 2 \ mod\ 23 = 10 \cdot (-5) - 2 \ mod\ 23 = 19 - 2 = 17 \end{cases} = (6, 17)$$

$$4P = \begin{cases} \lambda = 9 \\ x_{4P} = 0 \\ y_{4P} = 14 \end{cases} = (0, 14)$$

$$8P = \begin{cases} \lambda = 12 \\ x_{8P} = 6 \\ y_{8P} = 6 \end{cases} = (6, 6)$$

$11P = P + 2P + 8P = (1, 2) + (6, 17) + (6, 6)$

$$(1, 2) + (6, 17) = \begin{cases} \lambda = \frac{2-17}{1-6} \ mod\ 23 = 3 \\ x_{P+2P} = 9 - 1 - 6 \ mod\ 23 = 2 \\ y_{P+2P} = -2 + 3(1 - 2) \ mod\ 23 = -2 - 3 \ mod\ 23 = -5 \ mod\ 23 = 18 \end{cases} = (2, 18)$$

$$(2, 18) + (6, 6) = \begin{cases} \lambda = \frac{18-6}{2-6} \ mod\ 23 = -3 \\ x_{11P} = 9 - 2 - 6 \ mod\ 23 = 1 \\ y_{11P} = -18 + (-3)(2 - 1) \ mod\ 23 = 2 \end{cases} = (1, 2)$$

Quindi $11P = (1, 2)$

Esercizio 8 Impiegando una curva ellittica $E_p(a, b)$ su un campo finito:

1. Spiegare come si esegue in modo efficiente la moltiplicazione di un punto P per una costante intera k
2. Spiegare cosa si intende per "logaritmo discreto", se esiste, di un punto R in base P
3. Descrivere un algoritmo di scambio di chiavi basato sulla crittografia ellittica e spiegare perché può ritenersi sicuro

Svolgimento

1. La moltiplicazione kP si può eseguire efficientemente con la tecnica dei raddoppi ripetuti: si calcolano $2P, 4P, \ldots, 2^t P$ ciascuno come raddoppio del punto precedente, ottenendo t raddoppi cioè $\Theta(\log k)$ perché $t = \lfloor \log_2 k \rfloor$
2. Il logaritmo discreto è il più piccolo intero k, se esiste, tale che $R = kP$
3. DH su curve ellittiche (p. 50)

Esercizio 9 Impiegando una curva ellittica prima su un campo finito:

1. Spiegare come trasformare un numero intero in un punto della curva
2. Descrivere un algoritmo di scambio di messaggi cifrati e spiegare perché può ritenersi sicuro
3. Trasformare il messaggio $m = 5$ in un punto della curva prima $E_{23}(1, 1)$, usando il parametro $h = 3$

Svolgimento

1. Algoritmo di Koblitz, p. 51
2. Scambio di messaggi, p. 51
3. Troviamo prima i residui quadratici di Z_{23}

$x = 1 \Rightarrow x^2 \bmod 23 = 1$
$x = 2 \Rightarrow x^2 \bmod 23 = 4$
$x = 3 \Rightarrow x^2 \bmod 23 = 9$
$x = 4 \Rightarrow x^2 \bmod 23 = 16$
$x = 5 \Rightarrow x^2 \bmod 23 = 2$
$x = 6 \Rightarrow x^2 \bmod 23 = 13$
$x = 7 \Rightarrow x^2 \bmod 23 = 3$
$x = 8 \Rightarrow x^2 \bmod 23 = 18$
$x = 9 \Rightarrow x^2 \bmod 23 = 12$
$x = 10 \Rightarrow x^2 \bmod 23 = 8$
$x = 11 \Rightarrow x^2 \bmod 23 = 6$
Simmetria
$x = 12 \Rightarrow x^2 \bmod 23 = 6$
$x = 13 \Rightarrow x^2 \bmod 23 = 8$
$x = 14 \Rightarrow x^2 \bmod 23 = 12$
$x = 15 \Rightarrow x^2 \bmod 23 = 18$
$x = 16 \Rightarrow x^2 \bmod 23 = 3$
$x = 17 \Rightarrow x^2 \bmod 23 = 13$
$x = 18 \Rightarrow x^2 \bmod 23 = 2$
$x = 19 \Rightarrow x^2 \bmod 23 = 16$
$x = 20 \Rightarrow x^2 \bmod 23 = 9$
$x = 21 \Rightarrow x^2 \bmod 23 = 4$
$x = 22 \Rightarrow x^2 \bmod 23 = 1$

Quindi i residui quadratici sono $1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18$
Simuliamo l'algoritmo di Koblitz: $0 \leq i \leq h - 1$

$i = 0$ $x = mh + i = 5 \cdot 3 + 0 = 15$
$y^2 \bmod 23 = (15^3 + 15 + 1) \bmod 23 = 10$ non è residuo quadratico

$i = 1$ $x = 15 + 1 = 16$
$y^2 \bmod 23 = (16^3 + 16 + 1) \bmod 23 = 19$ non è residuo quadratico

$i = 2$ $x = 15 + 2 = 17$
$y^2 \bmod 23 = (17^3 + 17 + 1) \bmod 23 = 9 = 3^2$ è un residuo quadratico

$\Rightarrow m = 5 \longrightarrow P_m = (17, 3)$