I CIFRARI PERFETTI

Fabiola Genevois
Alessio Nunzi
Federico Russo

INDICE

Introduzione

Sembra, dopo la “rottura” del sistema DES, che l'ambizione secolare dei crittografi, (scoprire un cifrario *perfetto,* in modo tale che il crittogramma, in mancanza della chiave, non fornisca *nessuna* informazione sul messaggio in chiaro), sia destinata a restare insoddisfatta. Ebbene, uno dei risultati più sorprendenti della crittografia moderna, è che un tale cifrario, in realtà *esiste,* come si può *dimostrare*, con rigorose argomentazioni matematiche, che vedremo in seguito.
Per descrivere il funzionamento di questo cifrario, introduciamo, prima, i possibili attacchi che il crittoanalista può attuare.

1 – Strategie di attacco

Per descrivere i possibili modi in cui un crittanalista possa studiare i punti deboli dei crittosistemi, assumiamo che esso, conosca il crittogramma e non le chiavi e il testo in chiaro.
Quindi, i vari tipi di attacco possono essere qualificati come segue:

- *Ciphertext-only*: il crittoanalista, conosce i testi cifrati e da questi cerca di risalire ai testi in chiaro o alle chiavi
- *Know-plaintext*: l'attaccante, conosce uno o più testi in chiaro e le rispettive cifrature; da questi cerca di risalire alle chiavi o di decodificare altri testi cifrati
- *Chosen-plaintext*: l'attaccante, è in grado di cifrare senza conoscere le chiavi. Cerca di risalire alle chiavi o di decodificare altri testi cifrati
- *Adaptative chosen-plaintext*: l'attaccante, è in grado di cifrare ed è in grado di variare il testo in chiaro, in conseguenza dei testi cifrati ottenuti. Non conosce le chiavi, cerca di risalire alle chiavi o di decodificare altri testi cifrati
- *Chosen-ciphertext*: il crittoanalista, è in grado di decifrare, ma non conosce le chiavi e cerca di risalire ad esse.
- *Brute force*: il crittoanalista, tenta di decifrare il testo, provando tutte le possibili chiavi di cifratura. Questo attacco, ha efficacia maggiore con l'aumento della potenza di calcolo.

Dati tutti questi possibili casi di rottura del sistema crittografico, si sono cercati dei sistemi sempre più sicuri, anche dal punto di vista computazionale.

2 – Sicurezza dei sistemi crittografici

E' possibile valutare la sicurezza di un crittosistema nel seguenti modi:

- *Sicurezza computazionale*: un crittosistema, è considerato *computazionalmente sicuro*, se il miglior algoritmo noto che ne consente la violazione, ha complessità computazionale superiore ad un certo limite N sufficientemente grande. Allo stato attuale delle conoscenze, nessun crittosistema utilizzato in pratica, è stato dimostrato essere computazionalmente sicuro.
- *Sicurezza dimostrabile*: si cerca di fornire la prova, che la sicurezza di un crittosistema, è equivalente a quella di un problema che si ritiene difficile da risolvere; in tal caso il crittosistema è detto *dimostrabilmente sicuro*. La sicurezza del crittosistema, in questo caso, non è quindi assoluta ma relativa a quella di un altro problema.
- *Sicurezza incondizionale*: un crittosistema, è detto *incondizionalmente sicuro*, se non è violabile, anche utilizzando una potenza di calcolo illimitata.

La Teoria di Shannon, consente di studiare quest'ultima tipologia di sicurezza, mediante l'utilizzo delle probabilità.

3 – Il cifrario perfetto

La definizione formale di cifrario perfetto, fu data da Shannon. Esso affermava che:

" Un cifrario è perfetto se per ogni m appartenente a Msg e per ogni c appartenente Critto vale la relazione $\mathcal{P}(\mathcal{M}=m \mid \mathcal{C}=c) = \mathcal{P}(\mathcal{M}=m)$ *"*

Dove, Msg è l'insieme dei messaggi in chiaro e Critto è l'insieme dei crittogrammi.
In altre parole, un cifrario è perfetto, se la probabilità a priori che il mittente invii un messaggio *m*, è pari alla probabilità a posteriori che il messaggio inviato, sia effettivamente *m*, se il crittogramma *c* transita sul canale.
Una immediata conseguenza della definizione, è che, impiegando un cifrario perfetto, la conoscenza complessiva del crittoanalista, non cambia dopo che egli ha osservato un crittogramma arbitrario *c,* transitare sul canale.

Per comprendere meglio la precedente affermazione, assumiamo che:

- Il cifrario non sia perfetto
- $\mathcal{P}(\mathcal{M}=m) = p$ con $0 < p < 1$

Queste assunzioni implicano $\mathcal{P}(\mathcal{M}=m \mid C=c) \neq \mathcal{P}(\mathcal{M}=m)$, quindi si possono verificare tre casi:

1) Se $\mathcal{P}(\mathcal{M}=m \mid C=c) = 0$ il crittoanalista, osservando *c*, deduce che il messaggio spedito non è *m*, mentre prima sapeva che sarebbe potuto transitare *m* con probabilità *p*.
2) Se $\mathcal{P}(\mathcal{M}=m \mid C=c) = 1$ si può addirittura dedurre che il messaggio spedito è proprio *m*.
3) In tutti i casi intermedi, il crittoanalista raffina la sua conoscenza sul possibile messaggio spedito.

La conoscenza del crittoanalista rimane quindi inalterata se e solo se

$$\mathcal{P}(\mathcal{M}=m \mid C=c) = \mathcal{P}(\mathcal{M}=m)$$

Da questa definizione discende un importante teorema, anch'esso dovuto a Shannon, che sottolinea la impraticità dei cifrari perfetti.

3.1 – Enunciato di Shannon

"In un cifrario perfetto il numero delle chiavi deve essere maggiore o uguale al numero dei messaggi possibili "

Indicando con N_m il numero dei messaggi possibili e N_k il numero di chiavi con cui cifrare tali messaggi, dimostriamo che $N_k \geqslant N_m$

Dimostrazione

In un qualsiasi cifrario, il numero dei messaggi possibili deve essere inferiore alla cardinalità dell'insieme dei crittogrammi:

$N_m \leqslant |Critto|$ con *Critto* insieme dei crittogrammi

Se cosi non fosse, esisterebbero messaggi diversi, a cui corrisponde lo stesso crittogramma e sarebbe impossibile per il destinatario, decifrare il messaggio inviatogli!

Supponiamo ora, per assurdo, che in un cifrario perfetto valga la relazione:

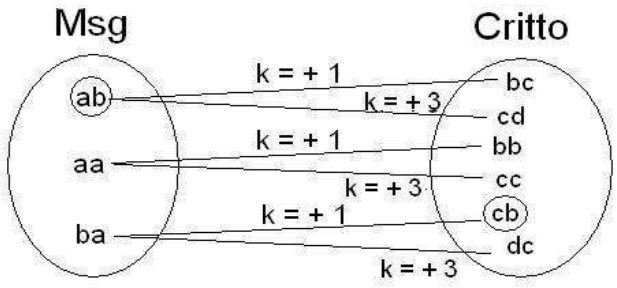
$$N_k < N_m$$

Si ha quindi che:

$$N_k < N_m \leqslant |Critto|$$

Ora, fissando un messaggio *m* con $\mathcal{P}(\mathcal{M}=m) \neq 0$, da esso, si possono generare al massimo N_k crittogrammi diversi, applicando tutte le chiavi.

Dato che $N_k \leqslant |Critto|$, esisterà in *Msg*, almeno un crittogramma *c*' che non è immagine di *m*.



Se “ab ” è il nostro m, “cb” è il c' che non è immagine di m!

Da questo ne segue che:

$$\mathcal{P}(\mathcal{M}=m \mid C=c) \neq \mathcal{P}(\mathcal{M}=m)$$

Abbiamo trovato l'assurdo poiché questa conclusione contrasta con la definizione di cifrario perfetto.

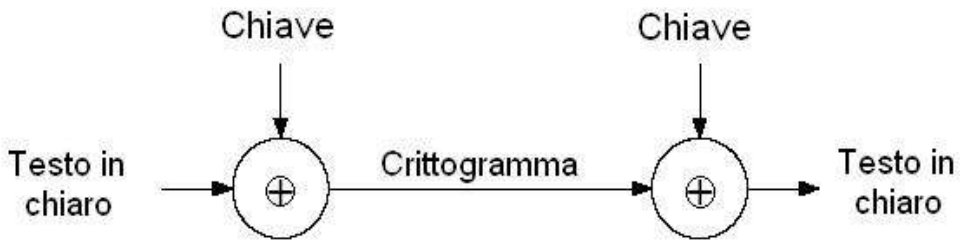
Quindi un cifrario perfetto non è assolutamente pratico da realizzare vista l'enorme dimensione dell'insieme delle chiavi.

4 – Il cifrario di Vernam

Claude Shannon formalizzò, quindi, il processo crittografico perfetto, mediante un modello matematico e concluse, che il cifrario inventato da Vernam, possedeva le proprietà per essere perfetto!
Il cifrario di Vernam, è detto anche *One-time Pad*. Questo nome, si riferisce alla sequenza della chiave, come se fosse scitta su un blocco di appunti (*pad*), e indica come tale sequenza, venga progressivamente utilizzata, per la cifratura e non sia riutilizzabile (*one-time*).
Fu usato, intensivamente, durante la Guerra Fredda, da spie della CIA e del KGB, ma anche dalla cosiddetta Linea Rossa, il famoso collegamento telefonico diretto Washington-Mosca.

Come funziona?

Nel 1917, Vernam costruì un dispositivo, in grado di leggere, contemporaneamente, due nastri d'ingresso e produrre, a partire da questi, un nastro di uscita, tale che ciascun foro, fosse generato mediante un'operazione di *OR- ESCUSIVO* (o `XOR`), dei corrispondenti due fori sui nastri d'ingresso. Ossia: in ciascuna posizione del nastro di uscita, viene praticato un foro, se e solo se, le corrispondenti posizioni nei 2 nastri d'ingresso, sono differenti fra loro (una ha un foro e l'altra no), nessun foro, in caso contrario (le posizioni originali sono entrambe forate o entrambe non forate).
Il funzionamento del cifrario di Vernam, quindi, è il seguente: si prende un nastro, su cui è perforata una sequenza casuale di caratteri, lunga almeno, quanto il testo che si intende cifrare e lo si inserisce sul primo lettore, mentre il nastro, su cui è perforato il testo in chiaro, va nel secondo lettore, attivando la macchina, si ottiene un nuovo nastro, completamente inintelligibile, ovvero il cifrato.
Per decifrare, si utilizzava la stessa macchina, inserendo in input i nastri contenenti, rispettivamente, il cifrato e la chiave. Effettuando la stessa operazione svolta per la cifratura, si ottiene sul nastro di output, il testo in chiaro.



Trent'anni dopo Shannon, per dimostrare la perfezione di tale cifrario, generalizzò l'idea di Vernam, prendendo in esame, *sequenze binarie* arbitrariamente lunghe, rappresentate in qualunque modo.

Proprietà

Il sistema di crittografia di Vernam è, teoricamente, immune da qualsiasi tipo di attacco.
Lo schema del cifrario perfetto di Vernam, si basa sull'impiego di una chiave comune, con le seguenti proprietà:

1. ***La chiave deve essere lunga almeno quanto il messaggio*** e deve anche essere disponibile con un certo anticipo. Questo, non è praticabile col *throughput* delle telecomunicazioni ad alta velocità. Inoltre, vista la lunghezza della chiave, il problema si sposta da comunicare un messaggio in tutta sicurezza, nella distribuzione sicura della chiave, grande quanto il messaggio stesso.

2. ***La chiave deve essere una sequenza completamente casuale di caratteri***.
   Un computer, non potrà mai utilizzare questo sistema, se non interfacciato con un generatore casuale esterno, dato che esso può generare solo sequenze pseudo-casuali. Ricorrendo però a generatori pseudo-casuali, si deve essere consci, che le proprietà del cifrario saranno valide, solo con una certa approssimazione, non semplice da definire o da qualificare.

3. ***La chiave non deve essere riutilizzata***.
   Ad ogni nuovo testo cifrato, deve corrispondere una differente chiave. Se così non fosse, verrebbe eliminata la casualità della chiave ed il cifrario sarebbe esposto ad attacchi di tipo *chipher-text* , che andrebbero facilmente a buon fine.
   Ad esempio poniamo due messaggi m' e m'' di n bit cifrati con la stessa chiave k da cui generiamo i due crittogrammi c' e c''.
   Abbiamo quindi $c_i' = m_i' \oplus k_i$ e $c_i'' = m_i'' \oplus k_i$, segue che $c_i' \oplus c_i'' = m_i' \oplus m_i''$ con $0 \leqslant i \leqslant n$
   Ad un crittoanalista, questo fornisce importanti informazioni: degli zeri consecutivi nello `XOR` tra c' e c'' corrispondono a tratti identici in m' e m'' ; inoltre per trovare m_i basterebbe tentare tutte le combinazioni di lettere, la cui distanza, è indicata dalla differenza delle due cifrature.

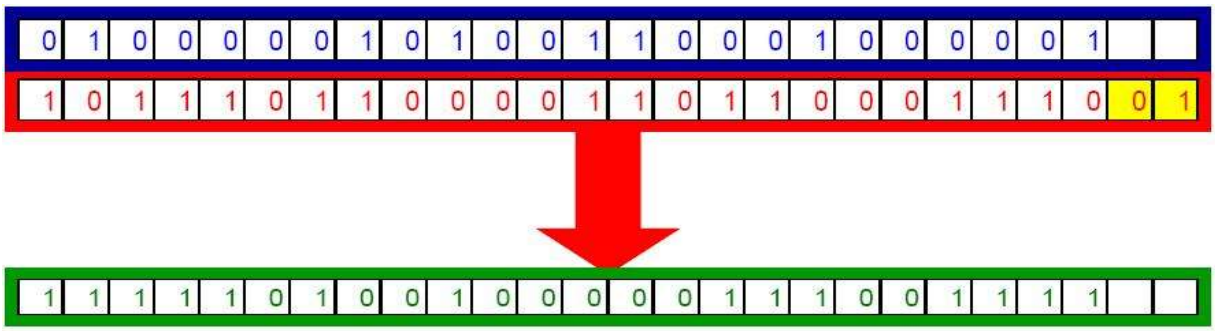
Uso di un cifrario perfetto

Iniziamo, illustrando un'applicazione del cifrario, sul linguaggio naturale, senza perdere, in alcun modo di generalità, utilizzando, come funzione di crittografia, la somma e la sottrazione.
Supponiamo, di dover trasmettere la parola "RANA". A ogni carattere della parola, si associa un numero, in base alla sua posizione sull'alfabeto tra 1 e 26. (a=1 ... z=26)
Utilizziamo poi, la chiave casuale *K* "AMID" (che ha lo stesso numero di caratteri della parola), ecco che, se utilizziamo la funzione somma, otteniamo una cosa di questo tipo:

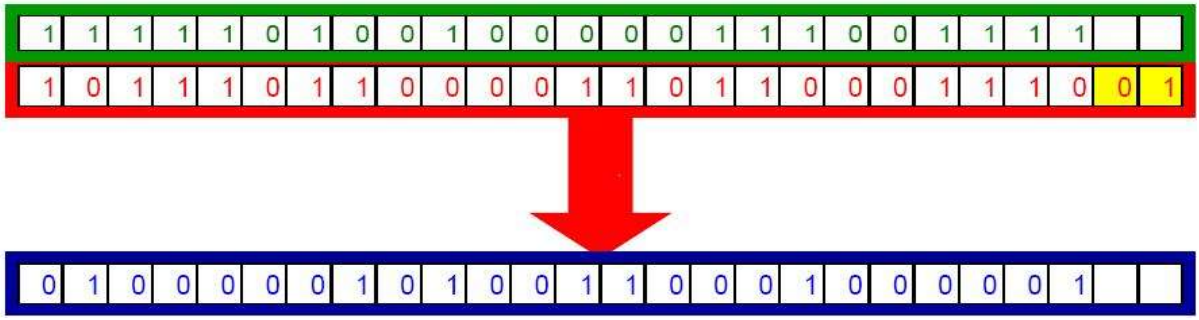
```
R   A   N   A
18  1   14  1  +
A   M   I   D
1   13  9   4  =
----------------
S   N   W   E
```

Il destinatario, dovrà utilizzare il processo inverso, andando a sottrarre a "SNWE", "AMID" ed otterrà, finalmente, "RANA".

Analizziamo un altro esempio, stavolta utilizzando sequenze binarie.
Volendo codificare la parola “ALA”, che ha una codifica Ascii 65-76-65, si esegue lo `XOR`, tra i bit delle singole lettere e i bit casuali, che costituiscono la chiave (lunga quanto il testo del messaggio).



La decodifica, avviene eseguendo sempre lo `XOR`, ma, in questo caso, tra i bit del crittogramma e i bit della chiave utilizzata.



Notiamo un interessante effetto di questo algoritmo, cioè anche se, il messaggio “in chiaro”, risulta periodico, il crittogramma non lo è. Infatti, uno stesso bit (lettera), viene in genere cifrato, in modo diverso, a seconda del bit (lettera), corrispondente nella chiave, rompendo cosi, eventuali pattern, esistenti nel messaggio in chiaro.

5 – One-Time Pad come cifrario perfetto

Prima di dimostrare la perfezione del One-Time Pad, dobbiamo fare due ipotesi:

1. Tutti i messaggi hanno la stessa lunghezza *n*
2. Tutte le sequenze di *n* bit sono messaggi possibili

La prima ipotesi, non è una grave forzatura, infatti, se i messaggi avessero lunghezze diverse tra loro, la lunghezza di uno di essi, darebbe informazione al crittoanalista, che escluderebbe automaticamente i messaggi più lunghi. E' quindi, sempre opportuno, adottare uno schema in cui i messaggi più corti di *n*, vengano completati con bit inutili, mentre quelli più lunghi, vengano divisi, in blocchi di lunghezza *n*.
La seconda ipotesi, verrà discussa, quando tratteremo i problemi legati all'uso del cifrario di Vernam.

5.1 Teorema di Shannon

Riprendendo la definizione di cifrario perfetto $\mathcal{P}(\mathcal{M}=m \mid \mathcal{C}=c) = \mathcal{P}(\mathcal{M}=m)$ dimostriamo che il One-Time Pad verifica questa relazione sotto le ipotesi precedenti.

Dimostrare che il cifrario è perfetto significa provare la validità della relazione

$$\mathcal{P}(\mathcal{M}=m \mid \mathcal{C}=c) = \mathcal{P}(\mathcal{M}=m,\mathcal{C}=c) / \mathcal{P}(\mathcal{C}=c)$$

Osserviamo che l'evento {M=m,C=c} descrive la situazione in cui il mittente ha genarato il testo in chiaro e lo ha cifrato come crittogramma *c.* Il crittogramma viene trasmesso su un canale non sicuro utilizzando k' per cui $k'_i \oplus m_i = c_i$.

Si ha cosi $k'_i = k'_i \oplus m_i \oplus m_i = c_i \oplus m_i$ quindi la chiave k' è unica.

Indicando con K un variabile aleatoria che modella il processo di generazione della chiave,il numeratore del secondo membro dell'equazione può essere riscritto come:

$$\mathcal{P}(\mathcal{M}=m,\mathcal{C}=c)=\mathcal{P}(\mathcal{M}=m,\mathcal{C}=c,\mathcal{K}=k')$$

Utilizzando a ritroso la formula della probabilità condizionale possiamo riscrivere questa equazione come:

$$\mathcal{P}(\mathcal{M}=m,\mathcal{C}=c)=\mathcal{P}(\mathcal{C}=c \mid \mathcal{M}=m,\mathcal{K}=k') \times \mathcal{P}(\mathcal{M}=m \mid \mathcal{K}=k') \times \mathcal{P}(\mathcal{K}=k') \qquad [1]$$

Ora abbiamo che $P(C=c \mid M=m, K=k') = 1$ per la definizione di k' ;

$P(M=m \mid K=k') = P(M=m)$ perchè la generazione della chiave sono due processi indipendenti; e $P(K=k')=(1/2)^n$ poichè la chiave è stata generata utilizzando una sorgente perfettamente casuale. Possiamo quindi concludere :

$$P(M=m; C=c) = (1/2)^n \times P(M=m) \qquad [2]$$

Combinando le equazioni [1] e [2] otteniamo infine:

$$P(M=m \mid C=c) = (1/2)^n \times P(M=m) / P(C=c) \qquad [3]$$

Rimane da calcalore $P(C=c)$. Per questo scopo consideriamo l'insieme degli eventi F_m ={M=n} con m appartente all'insieme dei Messaggi in chiaro. Poiche tutti questi eventi sono disgiunti abbiamo

$$P(C=c) = \sum_m P(M=m, C=c)$$

e poiche $P(U_m F_m)$=1 Ricaviamodall'equazione [2] :

$$P(C=c) = \sum_m P(M=m, C=c) = \sum_m (1/2)^n \times P(M=m) = (1/2)^n$$

Sostituendo questo risultato nell'equazione [3] otteniamo finalmente

$$P(M=m \mid C=c) = P(M=m)$$

cioè il *cifrario è perfetto*.

E' infine interesnte osservare che l'equazione [2] puo essere riscirtta come

$$P(M=m, C=c) = P(M=m) \times P(C=c)$$

il che indice che M e C sono variabili indipendenti. Quindi, per il crittoanalista, messaggioin chiaro e crittogramma risultano del tutto scorrelati fra loro, a sottolineare come in un cifrario perfetto nessuna informazione puo “filtrare” dal crittogramma.

5.2 - Limiti del One-Time Pad

Di fronte, a questo tipo di cifratura, un Hacker, che entra in possesso del crittogramma, non ha possibilità di capire il messaggio, a meno che, conosca anche lui la chiave!
Anche con un attacco Brute Force, avrà, le stesse possibilità di ricavare la parola corretta, come una non corretta.
Questo sistema di codifica, è usato per ottenere i più alti gradi di sicurezza militare, ma ha il grosso difetto, di non essere impiegabile su vasta scala, o per messaggi particolarmente lunghi. Alcuni algoritmi, sono stati costruiti, approssimando il sistema a chiave infinita, con l'espansione di una chiave, relativamente breve. Il sistema "Vernam", ad esempio, usato nelle trasmissioni telegrafiche, usava lunghi nastri di carta, contenenti dei bit casuali, i quali, erano aggiunti, ai bit del messaggio originario.
Il sistema One Time Pad, ha delle grandi **limitazioni**,perchè, rende necessario, scambiare ogni volta, una chiave, tramite un canale sicuro e inoltre, tale chiave deve essere, completamente casuale.

Scambio della chiave

Con un one-time pad, il messaggio segreto, non viene condensato. È difficile trasmettere il “pad” al destinatario, nella stessa misura in cui, è difficile trasmettere il messaggio stesso. La crittografia moderna, cripta grandi entità (connessioni Internet, audio e video digitali, conversazioni telefoniche, ecc..) ed avere a che fare, con i one-time pad, per questo tipo di applicazioni, è assolutamente impraticabile.

Generazione della chiave

Un generatore pseudo-casuale, è un algoritmo che, partendo da un valore numerico iniziale, scelto arbitrariamente (il *seme*), genera una sequenza di bit, apparentemente casuali (il *periodo*), che si ripete indefinitivamente.
All'interno del cifrario, è indispensabile che il mittente e il destinatario, utilizzino due generatori crittograficamente sicuri e identici, e si sincronizzino, sullo stesso punto iniziale della sequenza.
Se il generatore, è pubblicamente noto, il suo seme, costituisce, l'informazione segreta del cifrario. Questa soluzione, rispetta i principi guida della crittografia (pubblicità degli algoritmi), ma espone, il cifrario ad un banale attacco esaustivo, sull'insieme dei semi possibili. Pertanto, i semi, devono essere espressi, con un numero sempre maggiore di

cifre, cosa non permessa, dai generatori pseudo-casuali più comuni.
Anche il periodo del generatore, costituisce un punto debole, in quanto, pubblicamente noto. Quindi, si dovrebbero utilizzare, periodi molto lunghi oppure un seme diverso per ogni messaggio.
In alternativa, la chiave, può essere costituita, dal generatore e dal suo seme. Questa soluzione, è in aperto contrasto, col principio che gli algoritmi impiegati in un cifrario, siano pubblici. Non è poi così assurda, se si considera, che i generatori pseudo-casuali, sono in genere realizzati, con programmi brevissimi, per cui, non ne è difficile lo scambio segreto.
Un compromesso, tra i due metodi, sopra descritti, è quello di prendere come chiave, un file binario arbitrario, lungo e disponibile a tutti, su cui il mittente e il destinatario, si possono accordare, semplicemente, scambiandosi il nome del file e il punto da cui iniziare a scorrerlo. In tal modo, il cifrario è difficilissimo da attaccare, se il file chiave, è ragionevolmente casuale ed è stato acquisito in un periodo precedente e sicuro.

Tutte le sequenze di n bit sono messaggi possibili

Da quanto appena detto, risulta evidente, che il One-Time Pad, è una soluzione computazionalmente molto costosa. Pertanto, cerchiamo di rilassare la seconda ipotesi, su cui si basa, il teorema di perfezione.
Considerando che la comunicazione, si svolga in un linguaggio naturale, non tutti i 2^n possibili messaggi di n bit, sono legittimi.
Si parte dall'ipotesi che, il mittente generi sequenze di caratteri, in accordo con le frequenze, ricavate empiricamente, con cui le lettere dell'alfabeto, appaiono nella lingua considerata, e si raffina poi, il modello imponendo che, ogni carattere sia una funzione dei $q-1$ caratteri che lo precedono, in accordo, alle frequenze dei q-grammi del linguaggio.
Si ottiene, una “fedeltà” del modello, tanto maggiore quanto più grande è q, ed in particolare, non si generano messaggi, che contengono sequenze di lettere inesistenti nel linguaggio. Si tratta, sempre, di una grossolana approssimazione, perchè, ogni frase reale è costruita, secondo leggi semantiche del tutto diverse. Il modello risultante è in genere, sufficiente per uno studio statistico, ragionevolmente significativo.
E' stato stimato, che nei linguaggi naturali, il numero di messaggi di n bit semanticamente validi, sia approssimativamente espresso dalla funzione α^n, dove $\alpha < 2$ è una costante propria del linguaggio.
Ma, se il numero di messaggi è α^n, anzichè 2^n, anche il numero complessivo di chiavi, potrebbe essere ridotto, pur mantenendo per esse, una lunghezza n, pari a quella del messaggio.

Perchè un crittoanalista, possa decifrare un messaggio, con un attacco esaustivo sulle chiavi, è necessario, che solo una di esse, permetta di costruire il crittogramma *c*, da un messaggio significativo *m*. Se, infatti, utilizzando chiavi diverse, si ricostruissero da *c* messaggi significativi diversi, il crittoanalista, non potrebbe stabilire, quale sia effettivamente spedito.

Ci si protegge, quindi, contro questi attacchi, facendo sì che, da molti messaggi diversi, cifrati con chiavi divese, si generi, lo stesso crittogramma. Il numero totale $2^t\alpha^n$ dei crittogrammi generati, da tutti messaggi con tutte le chiavi, deve essere molto maggiore del numero 2^n dei crittogrammi possibili, dove *t*, indica il numero minimo di bit casuali necessari, per mantenere la sicurezza del cifrario. Dalla relazione $2^t\alpha^n >> 2^n$, si ricava $t + n\log_2\alpha >> n$, ovvero $t >> 0{,}88n$. Dunque, la lunghezza della chiave, deve rimanere, necessariamente, assai prossima a *n*, anche sotto queste ipotesi restrittive.

6 - Generatori pseudo-casuali

Rivolgiamo, la nostra attenzione, al problema delle sorgenti di sequenze. Nel caso binario, una sorgente casuale, genera una sequenza di *bit,* tali che, ognuno di essi, può assumere il valore 0, con probabilità P(0)= ½ o il valore 1 con probabilità P(1)=½ . La generazione di un bit, è completamente indipendente, da quella degli altri. Ciò implica, inimediatarnente, che il valore di un bit, non può essere previsto, osservando i precedenti. In particolare, è difficile garantire che la generazione di un bit, avvenga in modo indipendente dalla generazione degli altri, poiché, nella realtà, ogni esperimento, modifica l'ambiente circostante e può influenzare l'esperimento successivo. Nella pratica, la perfetta casualità di una sorgente, non potrà essere garantita e dovremo accettare alcuni compromessi.
Per la generazione di sequenze *brevi*, esistono due approcci, che presentano sufficienti garanzie di casualità.
Uno, sfrutta la presunta aleatorietà di alcuni fenomeni fisici (ad esempio: l'intervallo di tempo tra l'emissione di particelle, durante il decadimento di un materiale radioattivo, o i valori campionati, del segnale proveniente da un microfono).
L'altro, impiega, alcuni processi software, rilevando, per esempio, come valori casuali, Io stato dell'orologio interno di un calcolatore o la posizione della testina su un disco rigido.
I due approcci, producono risultati ragionevolmente imprevedibili se, non si ha accesso fisico ai dispositivi utilizzati. Il loro impiego, è però, problematico per difficoltà di realizzazione, e perché le sequenze generate, si considerano tanto più casuali, quanto più sono brevi. D'altra parte, poiché, il numero di bit casuali richiesti nelle applicazioni crittografiche, può essere molto elevato, si ricorre in genere, a un diverso meccanismo algoritmico, che ricerca, la casualità all'interno di alcuni processi matematici.

Un *generatore di numeri pseudo-casuali*, è un algoritmo, che parte da un valore iniziale detto *seme,* solitamente fornito, come dato di ingresso e genera una sequenza arbitrariamente lunga di numeri; questa, a sua volta, contiene una sottosequenza, che si ripete indefinitamente, detta *periodo*.
In linea di principio, un generatore, è tanto migliore, quanto più lungo è il periodo, perché *è* questo, che dovrebbe essere successivamente utilizzato come sequenza casuale. Per motivi di efficienza, i generatori, sono in genere realizzati con programmi molto brevi.
Questi generatori, possono, però, essere considerati *amplificatori di casualità*, perché, se innescati da un seme casuale di lunghezza *m,* fornito dall'utente o prodotto con uno dei metodi visti sopra, generano, una sequenza apparentemente casuale di lunghezza *n >> m.*

Una inerente limitazione, è il numero di sequenze diverse, che possono essere così generate è (nel caso binario) al massimo 2^m, enormemente minore del numero complessivo 2^n delle sequenze lunghe *n.* Tali caratteristiche, motivano per questi generatori, l'appellativo di pseudo-casuali.

Un generatore, si valuta, in base alla «similarità» tra la sequenza S da esso prodotta e una sequenza, perfettamente casuale. Può essere o meno accettato, a seconda dell'impiego previsto per la S.

Se, essa è destinata alla simulazione di un processo arbitrario, o alla scelta dei passi aleatori, all'interno di un algoritmo randomizzato, è sufficiente, che la sequenza superi una serie di test statistici standard, sulla frequenza e la distribuzione dei suoi elementi.

Se, invece, la S è destinata ad applicazioni crittografiche, in particolare alla generazione di chiavi segrete, essa, deve superare anche un severo test addizionale, che assicuri, che sia impossibile fare previsioni sui suoi elementi, prima che siano generati.

I test standard di valutazione, verificano, dunque, se una sequenza pseudo-casuale S, rispetta alcune proprietà delle sequenze casuali.

I principali sono:

- Il *test di frequenza,* verifica se i diversi elementi appaiono in S, approssimativamente, lo stesso numero di volte;
- Il *poker test*, che verifica la equidistribuzione di sottosequenze di lunghezza arbitraria, ma prefissata;
- Il *test di autocorrelazione*, che verifica il numero di elementi ripetuti a distanza prefissata;
- Il *run test* che verifica, se le sottosequenze massimali, contenenti elementi tutti uguali, hanno una distribuzione esponenziale negativa.

Un generatore pseudo-casuale, molto semplice, che supera con successo i quattro test citati è il *generatore lineare,* che produce, una sequenza di interi positivi $x_{1....}x_n$ a partire da un seme casuale x_0 secondo la relazione:

$$x_i = (ax_{i-1} + b) \bmod m$$

dove *a*, *b*, *m* sono interi positivi. Affinché, il generatore abbia periodo lungo m, e quindi induca una permutazione deglì interi 0,1,...,m-1, i suoi parametri devono essere scelti in modo tale che *rncd(b, m*) = 1, (a -1) sia divisibile per ogni fattore primo di m, e (a -1) sia un multiplo di 4, se anche m è un multiplo di 4.

Un'ovvia generalizzazione è il generatore polinomiale, che impiega una legge del tipo:

$$x_i=(a_1 x_{i-1}^t + a_2 x_{i-1}^{t-1} + \ldots + a_t x_{i-1} + a_{t+1}) \bmod m$$

e che supera anch'esso i test statistici. Questi generatori, possono essere facilmente trasformati, per produrre sequenze binarie pseudo-casuali. Si calcola il valore $r = x_i/m$: se la prima cifra decimale di r è dispari il bit generato è 1, altrimenti è 0.
I generatori lineari e polinomiali, sono particolarmente efficienti, ma non impediscono di fare previsioni sugli elementi generati, neanche quando il seme impiegato, è strettamente casuale. Esistono, infatti, algoritmi che permettono di scoprire, in tempo polinomiale, i parametri del generatore, partendo dall'osservazione di alcune sequenze prodotte, e questo, ne svela completamente il funzionamento. Così, se per esempio, il generatore, fosse impiegato per la creazione di una sequenza di chiavi, un crittoanalista potrebbe prevedere le chiavi future di un utente, dalla conoscenza di alcune sue chiavi passate. Affinché, questa condizione non sia verificata, occorre rafforzare definizioni e test.

Il progetto dei nuovi generatori, si fonda sull'esistenza di funzioni dette *one-way,* che sono computazionalmente facili da calcolare e difficili da invertire. Si conosce, cioè, un algoritmo polinomiale per il calcolo di y = *f(x),* ma si conoscono, solo algoritmi esponenziali per il calcolo di x = $f^{-1}(y)$.
Notiamo, che si opera su numeri, quindi, la complessità degli algoritmi, deve essere riferita alla lunghezza della rappresentazione di x : un algoritmo, che richiede un numero di operazioni proporzionale al valore di x, sarà dunque esponenziale. Consideriamo la sequenza S = x *f(x) f(f(x))...* ottenuta, iterando l'applicazione della *f*, un numero arbitrario di volte. Ogni elemento della S, si può calcolare, in modo efficiente dal precedente, ma, non dai successivi, perché f è *one-way*. Se, dunque, si calcola la S, per un certo numero di passi, senza svelare il risultato, e si comunicano poi gli elementi, uno dopo l'altro, in ordine inverso, ciascun elemento, non è prevedibile in tempo polinomiale, pur conoscendo quelli comunicati prima di esso.
Nelle applicazioni crittografiche, si impiegano generatori binari, che superano il *test di prossimo bit*, se non esiste un algoritmo polinomiale in grado di predire l'*(i* + 1)-esimo bit della sequenza pseudo-casuale a partire dalla conoscenza degli *i* bit precedenti, con probabilità significativamente maggiore di ½. Quindi, per un crittoanalista, che dispone di risorse computazionali ragionevoli, cioè polinomiali, la sequenza prodotta dal generatore, risulta indistinguibile da una sequenza casuale. I generatori, che superano il test di prossimo bit, sono detti *crittograficamente sicuri,* e si può dimostrare che superano,

anche, i quattro test standard, discussi in precedenza. Essi, vengono costruiti, impiegando particolari *predicati* delle funzioni one-way, cioè, proprietà, che possono essere vere o false, per ogni valore di *x.*

Un predicato *b(x),* è detto *hard-core,* per una funzione one-way *f(x)*, se *b(x) è* facile da calcolare conoscendo il valore di *x,* ma è difficile da calcolare (o anche solo da prevedere con probabilità di successo maggiore di ½), conoscendo solo il valore di *f(x)*. In sostanza la proprietà «hard-core», concentra in un bit, la difficoltà computazionale della *f*.

Un esempio di funzione one-way è la $f(x) = x^2 \bmod n$ se *n* non è primo.

Su questi risultati, si fonda il generatore *BBS,* introdotto nel 1986 da Blum, Blum e Shub. Sia $n = p * q$ il prodotto di due numeri primi, grandi tali che $p \bmod 4 = 3$ e $q \bmod 4 = 3$, e sia $x_0 = y^2 \bmod n$ per un qualche y. Il generatore *BBS* impiega x_0 come seme, calcola una successione x_i di $m \leqslant n$ interi e genera in corrispondenza una sequenza binaria *b* secondo la legge:

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

Il generatore parte da x_0 cui corrisponde b_m, e procede al calcolo degli interi $x_i,\ldots,x_m$ memorizzando i corrispondenti bit $b_{m-1},\ldots,b_0$ che comunica poi all'esterno in ordine inverso.

Dobbiamo dimostrare, che il generatore *BBS* ,supera il test di prossimo bit, ma questo, è in effetti un caso particolare di un risultato generale. Poniamo che *f(x),* sia una arbitraria funzione one-way e *b(x)* sia un suo predicato hard-core. Indichiamo con $f^{(i)}(x)$, l'applicazione iterata della funzione per *i* volte consecutive, e consideriamo la sequenza binaria, che si ottiene partendo da un valore arbitrario di *x,* calcolando *b(x)* e *f(x),* iterando il calcolo della *f,* un numero arbitrario di volte, e ordinando in senso inverso i valori del predicato così ottenuti:

$$b(f^{(i)}(x))\ b(f^{(i-1)}(x))\ \ldots\ b(f^{(2)}(x))\ b(f(x))\ b(x)$$

Per quanto osservato in precedenza, sulle funzioni one-way, e ricordando che il predicato scelto è hard-core, segue che ciascun elemento della sequenza, supera il test di prossimo bit. La precedente sequenza, è generata dal *BBS,* se si sceglie $f(x) = x^2 \bmod n$, e $b(x)$ è il corrispondente predicato di disparità.

Nonostante alcuni accorgimenti di calcolo, per incrementarne l'efficienza, il generatore *BBS,* è piuttosto lento, poiché, il calcolo di ogni bit, richiede l'esecuzione di un elevamento al quadrato, di un numero di grandi dimensioni nell'algebra modulare. La lentezza è, comunque, una prerogativa di tutti i generatori progettati, a partire da problemi computazionalmente difficili. Essi, sono pertanto, impiegati nella creazione di chiavi segrete corte, ma, non quando si richiedono lunghe sequenze casuali. Poiché, questo caso, si presenta in molte applicazioni crittografiche, si preferiscono generatori teoricamente meno sicuri, ma efficientissimi e comunque a tutt'oggi inviolati.

È interessante notare, che questi ultimi generatori, possono essere costruiti, utilizzando le funzioni di cifratura $\mathcal{F}(m, k)$ sviluppate per i cifrari simmetrici. Si utilizza la chiave segreta *k* del cifrario e si sostituisce il messaggio *m,* con un opportuno valore relativo al generatore. In tal modo l'imprevedibilità dei risultati, è garantita dalla struttura stessa dei cifrari, e per questi, esistono realizzazioni estremamente efficienti. Poiché, i cifrari generano parole (i crittogrammi), composte da molti bit, ogni parola prodotta, potrebbe essere interpretata come numero pseudo-casuale o come sequenza di bit pseudo-casuali. Vediamo un generatore di questo tipo, approvato come *Federal Inforniation Processing Standard (FIPS)* negli Stati Uniti: il cifrario impiegato al suo interno, è in genere, una versione del *DES* di amplissima diffusione. Detto *r* il numero di bit delle parole prodotte (nel *DES* si ha *r* = 64), *s* un seme casuale di *r* bit, *m* il numero di parole che si desidera produrre, e ricordando che *k* è la chiave del cifrario, abbiamo:

```
FunctiOn Generatore (s, m):
    d <-- rappresentazione su r bit di data e ora attuale;
    y <-- F (d, k);
    z <-- s;
    for i <— 1 to m do
        x_i <-- F (y Xor z, k);
        z <-- F (y Xor x_i , k);
        comunica all'esterno x_i
```