valv0's

cryptography workshop

6 may 2k2 – ZO' – Catania

" un computer sicuro e’ un computer a cui e’ stata tolta la corrente. "

cavallo/s0ftpj 2k1

Costantino Pistagna (valvoline@vrlteam.org) − crypto workshop − 6May 2k2 − CT

1.0 − Introduzione

La rete è NON sicura. Chiunque si accosti alla rete, deve tenere conto di questa variabile non indifferente e condizionante. Ricordo la famosa parabola delle lettere sigillate e delle cartoline: i pacchetti transitanti su internet possono essere paragonati a cartoline. Chiunque, con un po' di esperienza, può dargli un'occhiata e farne proprie le esperienze riportate.

La codifica delle stringhe (nella ragione piu' generale di bit) ha una sua motivazione di esistenza: la manutenzione della segretezza. La crittologia, cioe' lo studio dei sistemi per comunicare segretamente, e' costituita da due settori che si completano tra loro: la crittografia, intesa come la progettazione di sistemi per comunicazioni segrete, e l'analisi cifrata o crittologia, intesa come lo studio dei modi di compromettere una comunicazione segreta.

Bisogna fare un'assunto alla base di tutto il lavoro che tratteremo in seguito: la crittogralogia è di interesse militare, delle multinazionali e dei potenti. Nel mondo di oggi, loro sono i "buoni"; voi siete i "cattivi". Senza scendere troppo in ambito filosofico, si suppone che i crittografi siano i "buoni" e gli analisti i "cattivi": lo scopo che ci si prefigge è quello di proteggere i dati codificati e quant'altro risulti utile dai criminali. Qualora, non sembri ancora chiaro quello che voglio dire, basti notare che nel momento stesso in cui si ricorre alla crittografia, si presuppone l'esistenza di un qualche nemico. Se il mondo fosse fatto solo di buoni, non ci sarebbero molti problemi, senza dover arrivare a quello delle comunicazioni cifrate. Molto probabilmente non ci sarebbe neanke un manipolo di persone che si riuniscono per parlare di crittografia e segreti di stato, ma questa è un'altra storia.

Esistono attualmente due fazioni che ritengono di classificare la crittografia come "scienza" o come "arte". Personalmente ritengo che catalogare sia sempre una brutta parola, limitativa e che lascia troppo poco alla libera fantasia ed interpretazione, pero' diciamo che sarei piu' propenso a classificare questa branca dell'informatica nello schedario: "arte".

Costantino Pistagna (valvoline@vrlteam.org) − crypto workshop − 6May 2k2 − CT

Come tutto quello che si basa su algoritmi, la crittografia esisteva da molto prima della nascita dei primi elaboratori. La progettazione di sistemi di segretezza e gli studi sulle loro interpretazioni hanno un filo conduttore unico. Ritengo che non sia possibile dire quale delle due branche abbia trovato piu' giovamento dall'avvento dei computer e dei super−elaboratori: la crittografia (con maggiore potenza di calcolo, per le proprie elaborazioni) o l'analisi crittografica (con i propri algoritmi di attacco agli stessi ?).

ALAN TURING (1946)

ENIGMA

Quasi tutte le comunicazioni tedesche venivano criptate con una macchina di cifratura chiamata Enigma.

Questa macchina è una rappresentante niente affatto indegna di una classe di cifrari a rotore, utilizzati fino all'introduzione di cifrari elettronici e microelettronici che hanno sconvolto e trasformato il mondo della crittografia. Per forzare l'Enigma (alcuni dettagli della soluzione sono tenuti segreti fino ad oggi) Turing, per conto del governo inglese, si servì di gigantesche macchine chiamate appunto Colossi, che possono considerarsi i precursori dei moderni calcolatori elettronici.

Costantino Pistagna (valvoline@vrlteam.org) − crypto workshop − 6May 2k2 − CT

Provare a tracciare una storia della crittografia non e' impresa facile, e penso esuli dagli scopi di questo corso, quindi non vi allietero' con le varie strutture che si vedono su milioni di libri di crittografia. Direi, pero', che si possa tracciare una netta linea di confine, interessante per i nostri studi, dal 1970 in poi.

Intorno quegli anni, infatti un tale di nome: Feistel, termino' un lavoro per conto della IBM, che all'epoca per tutti era considerata (a torto o ragione), alla stregua del Grande Fratello. Il lavoro in questione era quello che oggi va' sotto il nome di DES (Data Encryption Standard). In quegli anni, la california e la florida erano un fervore di idee e rivoluzioni in tutti i campi della scienza digitale, binaria e numerica. Nel 1976 si apri' una nuova strada alla crittografia, con il lavoro di due, allora sconosciuti, ricercatori: Diffie e Hellman.

Il loro lavoro introduceva e stabiliva le basi per una nuova teoria della crittografia, che da' li a qualche anno sarebbe diventato uno standard di importanza mondiale: la crittografia a chiavi pubbliche. Agli albori degli anni '80, tre ricercatori: Rivest, Shamir ed Adleman, diedero vita ad un protocollo che ripercorreva quello che Diffie ed Helman avevano soltanto accennato.

L'algoritmo, basato sulla fattorizzazione dei numeri primi, prese il nome di RSA e di fatto, mise le basi per diventare lo standard mondiale a cui tutti da li' a poco avrebbero fatto capo.

Costantino Pistagna (valvoline@vrlteam.org) − crypto workshop − 6May 2k2 − CT

Comunicare significa condividere, emozioni, concetti, informazioni, dati. Negli ultimi 10 anni l'incremento delle comunicazioni voce e dati, è quadruplicato, ed e' destinato a crescere ancora di piu'. Tutto questo gran bisogno di comunicazione, ha messo in luce alcuni aspetti, che fino ad ora erano rimasti celati alla maggior parte dell'utenza: la sicurezza delle trasmissioni; quando si comunica via telefono, fax, email, su un canale di trasmissione, chiunque puo' ascoltarci e fare proprie le nostre esperienze.

Consideriamo le principali reti pubbliche di comunicazioni:

Trasmissioni radio e satellitari:
Molti sistemi di telefonia analogica cellulare non presentano meccanismi di codifica cifrata tali da evitare intercettazioni da parte di terzi. La maggior parte delle trasmissioni digitali, anche se non sono facili da decodificare, non usano meccanismi di cifratura. Le nuove trasmissioni satellitari digitali utilizzano delle semplici tecniche di cifratura che vengono facilmente eluse (SECA, SECA2, IRDETO, VIACCESS, etc.).

Telecomunicazioni terrestri:
Le reti pubbliche di telefonia terrestre presentano anche loro delle vulnerabilita'. In alcune reti (con codifica CCITT7 o inferiore) ci si puo' inserire nelle centrali numeriche e ridirigere, chiudere, o ascoltare qualsiasi conversazione o trasmissione dati.
Questo fenomeno chiamato Phreaking preventivato in bilancio dalle compagnie telefoniche, non viene ammesso ufficialmente perche' condurrebbe ad una maggiore perdita economica.

Reti Digitali ad alta velocita':
Anche questo tipo di reti non e' immune da tali problemi. Su una rete tipo internet, infatti non e' troppo difficile poter ascoltare e controllare i pacchetti transitanti non indirizzati espressamente alla nostra macchina. Questo tipo di tecnica e' spesso chiamata "sniffing".

Chi puo' volere od esigere sicurezza nella trasmissione dei propri dati ?

- Transazioni finanziarie
- Informazioni commerciali di vario tipo
- Varie ed eventuali (anche gli illegali usano la crittografia!)

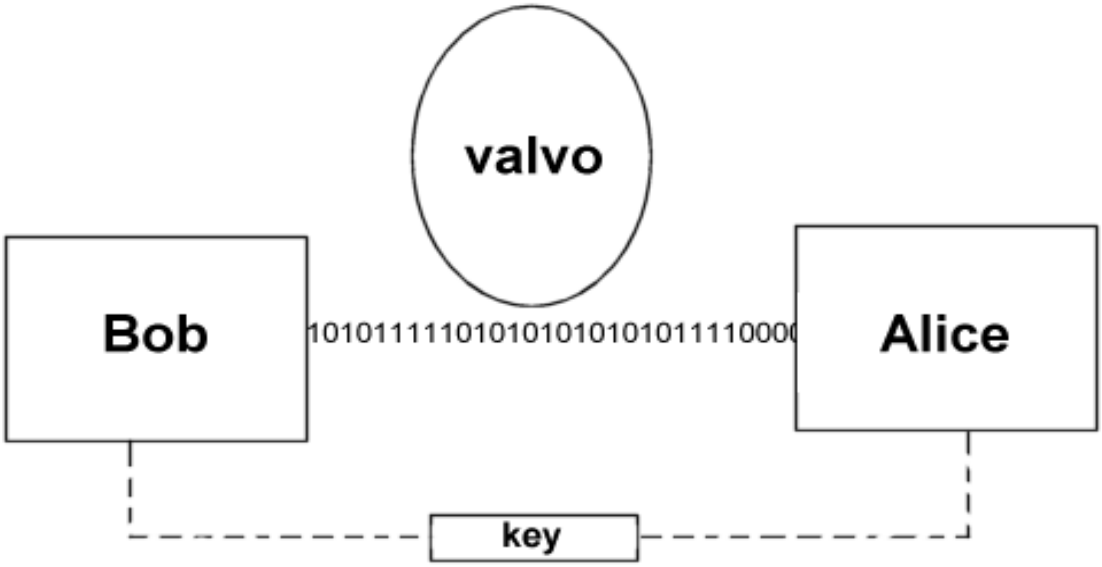
Non sono solo queste, tuttavia, le motivazioni che spingono il settore della crittografia. L'autenticazione dell'utente, la certificazione e la firma digitale, sono altri esempi di applicazioni possibili della crittografia al mondo moderno.

- Essere sicuri che su di una macchina accessibile al pubblico, soltanto gli utenti autorizzati abbiano accesso, e' un fattore determinante dell'informatica moderna.

- La certezza che un documento firmato da Marco, sia in effetti stato scritto ed inviato da Marco, apre nuove frontiere al mondo dell'informatica e del commercio elettronico.

1.1 − Uno schema Generale

L'insieme degli elementi che consentono una comunicazione sicura tra due individui sono generalmente detti sistemi di cifratura. La letteratura indica un insieme di questo tipo nella seguente forma:



Il sistema è di facile comprensione: Bob, il mittente, comunica con Alice, il ricevente, in maniera "sicura", grazie all'utilizzo di una funzione/chiave (algoritmo) applicata al testo originale. Ovviamente, non ci sono solo loro: valvo, qui' sta' ad indicare il "cattivo" di turno, quello, cioe', che provera' a violare la comunicazione tra B. ed A. Quello sopra è, nella forma più generale, il problema di tutta la crittografia:

Rendere sicura una comunicazione, supponendo che il canale di comunicazione possa essere violato, cioe' sia NON−SICURO.

(nota: se il canale, fosse sicuro non ci sarebbe bisogno di usare un sistema crittografico. Basterebbe mettere i dati sul canale.)

(nota2: se si potesse trovare il modo di vivere senza lavorare, forse non staremmo qui' ad allietarci con queste simpatiche allegorie matematiche).

Costantino Pistagna (valvoline@vrlteam.org) − crypto workshop − 6May 2k2 − CT

1.2 − basi matematiche necessarie

Come avevo accennato sopra, tutta la crittografia si basa sull'utilizzo di chiavi e funzioni (algoritmi). Senza di esse, lo studio e la creazioni di algoritmi crittografici, sarebbe praticamente non proponibile. Diamo quindi un'occhiata a questi due nuovi termini.

Per funzioni, nel mondo crittografico, si intende qualunque funzione matematica.

$$F(x) = y$$

F(X) e' una funziona matematica. X, e' la sua variabile. Al variare di x, si ottengono output diversi. Applicando questo paradigma al mondo della crittografia: F() diventera' la nostra funzione/chiave ed X, sara' il testo in chiaro da cifrare. Y, sara' il nostro output.

Ovviamente, altrimenti non sarei qui', le cose non sono cosi' semplici. Le funzioni, ad esempio, devono rispettare un principio fondamentale:

La funzione deve essere BIETTIVA.

Indicando con:

- X, il dominio della funzione ($x \in X$)
- Y, il codominio della funzione ($y \in Y$)

*Richiediamo alla nostra funzione, che sia BIETTIVA se: per qualunque elemento in X, esiste **uno ed un solo** elemento in Y. La condizione e' necessarie e sufficiente (vale anche il contrario)*

Quest'assunto, per quanto possa sembrare banale, e' di fondamentale importanza per la crittografia: pensate per un attimo, se il vostro algoritmo non possa assicurare UNIVOCITA' in un senso. In pratica, due file (DIVERSI), saranno cifrati allo stesso modo.

e il decrypt ?, come diavolo lo faccio ?

Non ci basta!. Esiste un altro tipo di funzioni in matematica, che giocano un ruolo importantissimo nella crittografia; sono chiamate funzioni "one−way".

Una funzione viene detta "one−way", se per qualunque $x \in X$, si ha uno ed un solo elemento $y \in Y$ **FACILMENTE** computabile. Dato $y \in Y$, invece, e' **IMPOSSIBILE** calcolare $x \in X$, in maniera **EFFICENTE**.

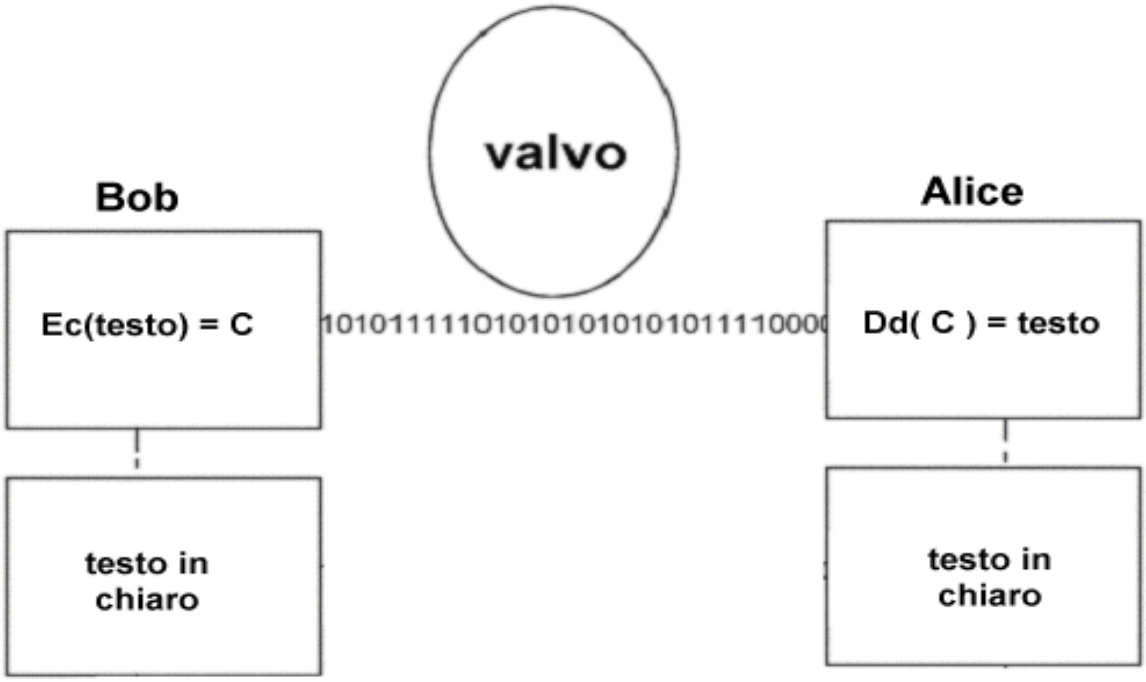
Vorrei porre la vostra attenzione sul fatto che allo stato attuale delle cose non c'e' molta chiarezza su quale possa essere un candidato di funzione che rispetti quanto detto sin'ora:

FACILITA' COMPUTAZIONALE
IRREVERSIBILITA' IMPOSSIBILE
EFFICENZA

Volendo vedere la cosa da un punto di vista matematico, non esiste nessuna dimostrazione, che un tale tipo di funzione esista, e che sia computabile. Comunque, esistono molte funzioni, che assomigliano e si avvicinano ad una funzione one−way. Possiamo computarle facilmente, e non riusciamo a reversarle. (x ^ 2, ad esempio e' molto facile da implementare. La stessa cosa non possiamo dire di x ^ ½) ⁕
Ad ogni modo, comunque la si vuole intendere, la funzione one−way gioca un ruolo chiave in tutta la moderna crittografia. Vedremo che la si utilizza per sistemi di checksum, di fingerprint, di Controllo Manipolazione di Codice, etc.etc.

Quello che faremo e' accostarci a quanto detto, con cautela e scetticismo: analizzando le nostre funzioni attentamente, stando attenti a vagliarne pro' e contro.

Alla luce di quanto sopra esposto, possiamo riguardare con occhio piu' critico alla parabola di Alice, Bob e Valvo:



La terminologia utilizzata è quella standard della crittografia tradizionale:

- **La funzione di Encrypt viene indicata con E_c dove "c", è la chiave.**
- **La funzione di Decrypt viene indicata con con D_d dove "d", è la chiave.**

1.3 − Crittografia Simmetrica

Con il termine crittografia simmetrica, si intende quella branca della crittografia, dedita allo studio di sistemi di cifratura a singola chiave. In pratica mittente e destinatario sono in possesso della stessa chiave, utilizzata per la cifratura/decifratura del testo. Ritornando al grafico della pagina precedente: c = d.

Si suppone che lo scambio di chiave venga fatto servendosi di canali sicuri. inviolabili.

"Se una lettera del testo in chiato e' la N−esima lettera la si sostituisce con la lettera dell'alfabeto di posto N+K, dove K e' un qualsiasi intero fissato."

```
char *encoding(char *input, int len, int K) {
   char *output;
   output=(char *)malloc(len*sizeof(char));
   for(int i=0;i<len;i++) output[i] = input[i] + K;
   return output;
}

char *decoding(char *input, int len, int K) {
   char *output;
   output=(char *)malloc(len*sizeof(char));
   for(int i=0;i<len;i++) output[i] = input[i] − K;
   return output;
}
```

Non ci vuole molto a capire che questo metodo e' molto debole!. L'analista non deve far altro, che indovinare il valore di K. Tentando ciascuna delle 26 lettere dell'alfabeto, sicuramente riuscirà a leggere il messaggio!.

"Impieghiamo una tabella (chiave) per definire le sostituzioni: per ciascuna lettera del testo in chiaro, la tabella dice quale lettera mettere nel testo cifrato."

```
char *encoding(char *input, char *KEY, int len) {
   char *output;
   output=(char *)malloc(len*sizeof(char));
   for (int i=0; i<len; i++ ) output[i] =
KEY[input[i]%97];
   return output;
}

char *decoding(char *input, char *KEY, int len) {
   char *output;
   output=(char *)malloc(len*sizeof(char));
   for (int i=0; i<len; i++ ) {
      for (int j=0; j<strlen(KEY); j++ )
         if ( input[i]==KEY[j] ) break;
      output[i]=j+97;
   }
   return output;
}
```

Questo metodo è molto più potente: un possibile attaccante, dovrebbe provare molte più combinazioni (tabelle), **circa 27!**

Benvenuti nel mondo della Crittografia!

1.3.1 − una prima analisi crittografica

Gli algoritmi a sostituzione semplice, sopra elencati soffrono di un grave problema, che sicuramente tutti voi avete già individuato: lo schema di cifratura, è facilmente individuabile a causa della frequenza che le lettere ereditano dal linguaggio. Ad esempio, visto che la A è la lettera più frequente della lingua italiana, l'analista potrebbe partire sostituendo con questa lettera la lettera più frequente del messaggio cifrato.

La situazione migliora ulteriormente (per il cattivo), quando si tiene conto anche delle combinazioni di due lettere (bigrammi), e tre lettere (trigammi). Certi bigrammi, non possono mai comparire perché privi di senso nelle lingue inglese ed italiano, ad esempio. QJ non ha nessun senso!

Analizzando le frequenze delle lettere e delle loro combinazioni, è possibile risalire molto facilmente ad un algoritmo di questo genere.

Benvenuti nel mondo dei "CATTIVI"!

"il valore di K, del nostro primo algoritmo, per ciascuna lettera è determinato attraverso una piccola chiave ripetuta. Ad ogni passo si aggiunge l'indice della lettera corrente della chiave all'indice della lettera del testo, determinando l'indice della lettera del testo cifrato."

```
char *encoding(char *input, int len, char *K, int K_len) {
   char *output;
   output=(char *)malloc(len*sizeof(char));
   for(int i=0;i<len;i++) output[i] = input[i] + K[i%K_len];
   return output;
}

char *decoding(char *input, int len, char *K, int K_len) {
   char *output;
   output=(char *)malloc(len*sizeof(char));
   for(int i=0;i<len;i++) output[i] = input[i] − K[i%K_len];
   return output;
}
```

Possiamo continuare a complicare il nostro algoritmo base, impiegando ad esempio, tabelle generali differenti per ciascuna lettera del testo in chiaro, invece di semplici offset di posizione.

Più lunga sarà la chiave, più sicuro sarà il nostro algoritmo

Se la chiave è lunga come il testo da cifrare, si ottiene un sistema *one−time−pad*

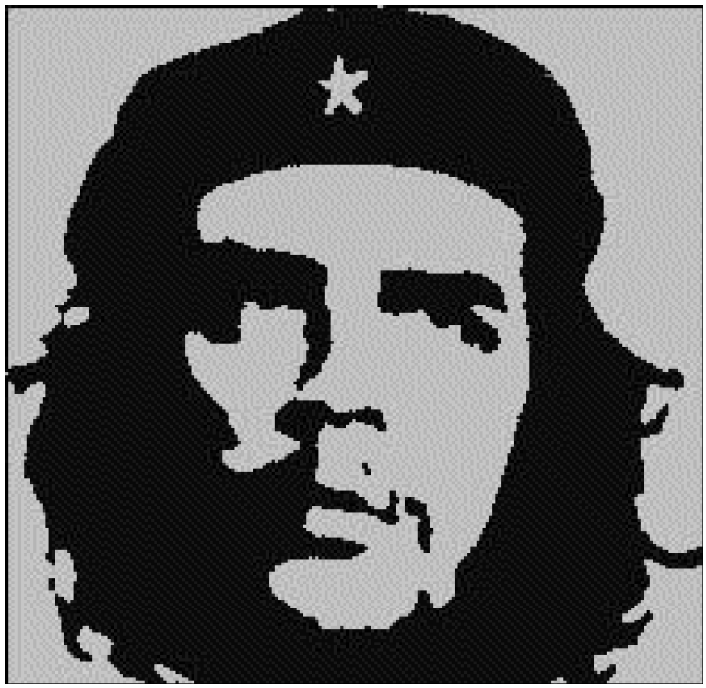
In realta' il sistema One−Time−Pad, per essere dimostrato realmente sicuro, deve soddisfare le seguenti 3 regole:

- **H(K) >= H(mex)**
- **| K | >= | mex |**
- **la chiave deve essere cambiato dopo ogni cifratura/decifratura**

Costantino Pistagna (valvoline@vrlteam.org) − crypto workshop − 6May 2k2 − CT

Il sistema one−time−pad è l'unico sistema dimostrabilmente sicuro.
i sistemi di comunicazione vitali, utilizzano sistemi su questa base.

¢



1.4 − Digitale è BELLO (ed efficiente)

Fin'ora abbiamo parlato di sistemi basati su elaborazioni del testo in chiaro di tipo "banale". La somma, la sostituzione, etc. Se si interpreta il nostro testo come una sequenza di bit, è possibile utilizzare un comune operatore noto a tutti gli esperti del settore crittografia:

lo XOR (OR−esclusivo).

A	B	A^B
0	0	1
0	1	0
1	0	0
1	1	1

"Per cifrare il messaggio, lo si sottopone ad un "or−esclusivo" bit a bit con una chiave."

```
char *encoding(char *input, int len, char *K, int K_len) {
   char *output;
      output=(char *)malloc(len*sizeof(char));
         for(int i=0;i<len;i++) output[i] = input[i] ^
K[i%K_len];
            return output;
}
char *decoding(char *input, int len, char *K, int K_len) {
   char *output;
   output=(char *)malloc(len*sizeof(char));
   for(int i=0;i<len;i++) output[i] = input[i] ^ K[i%K_len];
   return output;
}
```

E' molto interessante notare, una caratteristica speculare di questo metodo:
il decifrare coincide con il cifrare: il testo cifrato è lo XOR del testo in chiaro e della chiave. Iterando ancora una volta, allora, con un altro XOR con la chiave, si riottiene il messaggio iniziale!

Lo XOR tra il testo in chiaro ed il testo cifrato, fornisce la chiave utilizzata!

In realtà molti sistemi (anche commercialmente in distribuzione attualmente), hanno la non desiderabile proprietà di rivelare la chiave una volta noto il messaggio in chiaro!.

1.4 schemi a blocchi e stream

In realta' gli algormito simmetrici, sono divisi in due sottocategorie. Algoritmi a blocchi (block cipher) ed algoritmi streaming.

"...un algoritmo **crittografico a blocchi**, consiste di un algoritmo che divide il testo in input in blocchi, e prosegue nella cifratura di ognuno dei blocchi singolarmente."

Un algoritmo crittografico di tipo stream, e' un sistema a blocchi (n), con n = 1

I sistemi e gli esempi, di cui ci siamo occupati sin'ora sono sistemi classificati di tipo stream, perche' prendono in input un carattere (byte) alla volta, e ne ritornano l'output cifrato. In realtà tutti gli esempi trattati, possono essere ricondotti a sistemi di tipo a blocchi, senza troppa fatica.

"E' importante chiarire che i due sottosistemi si differenziano solo per le modalità di interazione con il testo sorgente."

I sistemi stream sono importanti per la **velocita'** computazionale richiesta e per le poche risorse stimate. In realtà i sistemi simmetrici, in commercio oggi, adottano un approccio di tipo a blocchi, per svariate ragioni, che proverò a riassumere qui:

- **sicurezza intrinseca del sistema**
- **resistenza stimata ad attacchi di tipo lessicale**
- **migliore capacità di interazione con dati binari (files ed affiliati)**
- **migliore gestione dell'input (efficienza)**

1.5 Crittografia Asimmetrica

Ritengo, che il problema della distribuzione delle chiavi, sia il problema cruciale ed il punto debole della maggior parte dei criptosistemi a chiave simmetrica.

Non importa quanto sicuro sia un cripto−sistema. Se un intruso riesce ad entrare in possesso di una chiave, il sistema viene compromesso irrimediabilmente, perdendo di valore intrinseco ed applicativo.

Alla fine degli anni ’70 due ricercatori, Diffie ed Hellman proposero un tipo di cripto−sistema radicalmente nuovo, nel quale le chiavi di cifratura e decifratura fossero differenti, e la chiave di decifratura non potesse essere derivata da quella di cifratura. Nella loro proposta, l’algoritmo di cifratura (E), e l’algoritmo di decifratura (D), devono rispettare insieme i seguenti tre requisiti:

- **D(E(P)) = P**
- **E’ estremamente difficile dedurre D da E.**
- **E non puo’ venir violato da un attacco con testo in chiaro selezionato**

Sotto queste condizioni non esiste ragione, per cui la chiave di cifratura non posso essere pubblica. In pratica, nella crittografia a chiave pubblica ogni utente possiede due chiavi: una chiave pubblica, usata da tutto il mondo per cifrare i messaggi da inviare a quell’utente, e una chiave privata, che l’utente utilizza per decifrare i messaggi. Ci riferiremo d’ora in avanti a queste chiavi come chiavi **pubbliche** e **private**.

Sebbene le chiavi pubbliche e private siano correlate matematicamente, la chiave privata non puo’ essere facilmente ricavata dalla chiave pubblica e per questo motivo rendere nota la chiave pubblica non costituisce una perdita di sicurezza.

1.6 RSA

Nel 1978, un gruppo di ricercatori del MIT, trovo' un buon metodo, per rendere in pratica quello che Diffie ed Hellman avevano teorizzato soltanto due anni prima. I tre ricercatori: Rivest, Shamir ed Adleman idearono l'algoritmo che ad oggi e' noto come **RSA**. Il sistema si basa sull'enorme differenza tra la facilità di trovare numeri primi grandi e la difficoltà di scomporre in fattori il prodotto di due numeri primi grandi. Come abbiamo già accennato, in un sistema a chiavi pubbliche ogni partecipante ha una **chiave pubblica** ed una **chiave privata**.

Ogni chiave contiene un pezzo di informazione

Nel cripto−sistema RSA, ogni chiave consiste di una coppia di interi. I partecipanti "Alice" e "Bob" hanno rispettivamente le chiavi pubbliche e segrete seguenti: P_A S_A e P_B S_B .

E' fondamentale mantenere sicura la chiave segreta, ma la chiave pubblica può essere rivelata senza particolari accorgimenti a chiunque.

Le chiavi pubblica e segreta per qualunque partecipante sono una coppia corrispondente, nel senso che specificano funzioni che sono una l'inversa dell'altra. Cioe':

$$\mathbf{M = S_A(P_A(M))}$$
$$\mathbf{M = P_A(S_A(M))}$$

Quest'idea venne sintetizzata e trasformata in RSA, secondo le seguenti regole:

- **selezionare due numeri primi random P e Q grandi.**
- **Calcolare n=P*Q.**
- **Selezionare un intero piccolo random dispari E, tale che:**
  E > 1
  E < PQ
  E ed (P−1)*(Q−1) siano primi tra loro (nessun fattore primo in comune)
- **Calcolare D, in modo che: E*D = 1 mod ((P−1)*(Q−1))**
  In pratica (DE −1) sia divisibile per (P−1)*(Q−1)
  D è l'inverso moltiplicativo di E.
  Questo non è difficile da fare:
  basta trovare X, tale che: D=(X*(P−1)*(Q−1) + 1)/E sia intero.

- **Pubblicare la coppia P=(E, n)**
- **Mantenere segreta la coppia S=(D,n)**

Sotto queste regole, la trasformazione di un messaggio M associato ad una chiave pubblica **P=(E,n)** è dato dall'equazione:

$$P(M) = M^E \pmod n$$

Viceversa, la trasformazione di un testo cifrato C, associato alla chiave segreta **S=(D,n)** è:

$$S(C) = C^D \pmod n$$

La sicurezza del sistema di crittografia RSA, si basa in gran parte sulla difficoltà della scomposizione in fattori di interi grandi.

Se un attaccante, è in grado di scomporre il modulo n di una chiave pubblica, allora può derivare la chiave segreta dalla chiave pubblica, usando la conoscenza dei fattori P e Q nello stesso modo in cui li usa il creatore della chiave pubblica.

In questo modo, se la scomposizione in fattori di interi grandi è facile, allora anche violare il sistema di crittografia RSA è facile.

L'opposto, cioè, se la scomposizione in fattori di interi grandi è difficile allora anche violare il sistema RSA è difficile **non è stata mai provata**. Attualmente, l'unico metodo conteplato per attaccare RSA è quello di scomporre in fattori il modulo n. Operazione, purtroppo, troppo complessa anche per le macchine più veloci, odierne.

Per raggiungere sicurezza con il sistema RSA, è necessario lavorare con interi che siano lunghi almeno 300−500 cifre, poiché la scomposizione di interi piu' piccoli non porta a buoni risultati.

Secondo gli ultimi studi fatti, fattorizzare un numero di 200cifre richiede circa 4.000.000.000 di anni di tempo macchina; fattorizzare un numero di 500 cifre, richide circa 10^{25} anni.

In entrambi gli esempi, si utilizzano il **miglior algoritmo** noto ed un computer con un tempo di **1ns** per istruzione. Cosa ancora piu' importante, è quella di riuscire ad essere in grado di trovare primi grandi in modo efficiente, per creare le chiavi necessarie.

La complessità e la difficolta' computazionale di RSA si basa su questi problemi, appena elencati.

Vediamo un esempio di codice di quanto appena detto:

```
char *RSA_encrypt(char *input, verylong D, verylong n) {
   char *output=(char *)malloc(sizeof(char)*strlen(input));
   for(int i=0;i<strlen(input);i++) output[i]=my_pow(input[i], D)%n;
   return output;
}

char *RSA_decrypt(char *input, verylong E, verylong n) {
   char *output=(char *)malloc(sizeof(char)*strlen(input));
   for(int i=0;i<strlen(input);i++) output[i]=my_pow(input[i], E)%n;
   return output;
}
```

Vediamo un esempio su tabella di quello che fa' il nostro algoritmo:

Stringa	Intero	M^E	M^E mod n
V	22	10648	22
A	01	1	1
L	13	2197	19
V	22	10648	22
O	16	4069	4

Per la conversione:

Testo Cifrato	M^D	M^D mod n	Stringa
22	2494357888	22	V
1	1	1	A
19	893871739	13	L
22	2494357888	22	V
4	16384	16	O

Per questo esempio abbiamo scelto:

- p = 3
- q = 11
- n = 33
- e = 3
- d = 7

Come mostra il codice dell'algoritmo e la nostra tabella di esempio, in ogni blocco di testo in chiaro utilizziamo un solo carattere (limitazione pedagogica: imposta dai numeri primi scelti, troppo piccoli, e dal codice, che passa al setaccio un solo carattere alla volta...brp). Il risultato, come si puo' vedere è un cifrario a sostituziona alfabetica, e peraltro poco efficiente. Scegliendo numeri interi molto piu' grandi, potremmo rivedere il nostro algoritmo in modo da prendere in input **blocchi** di caratteri e non caratteri singoli.

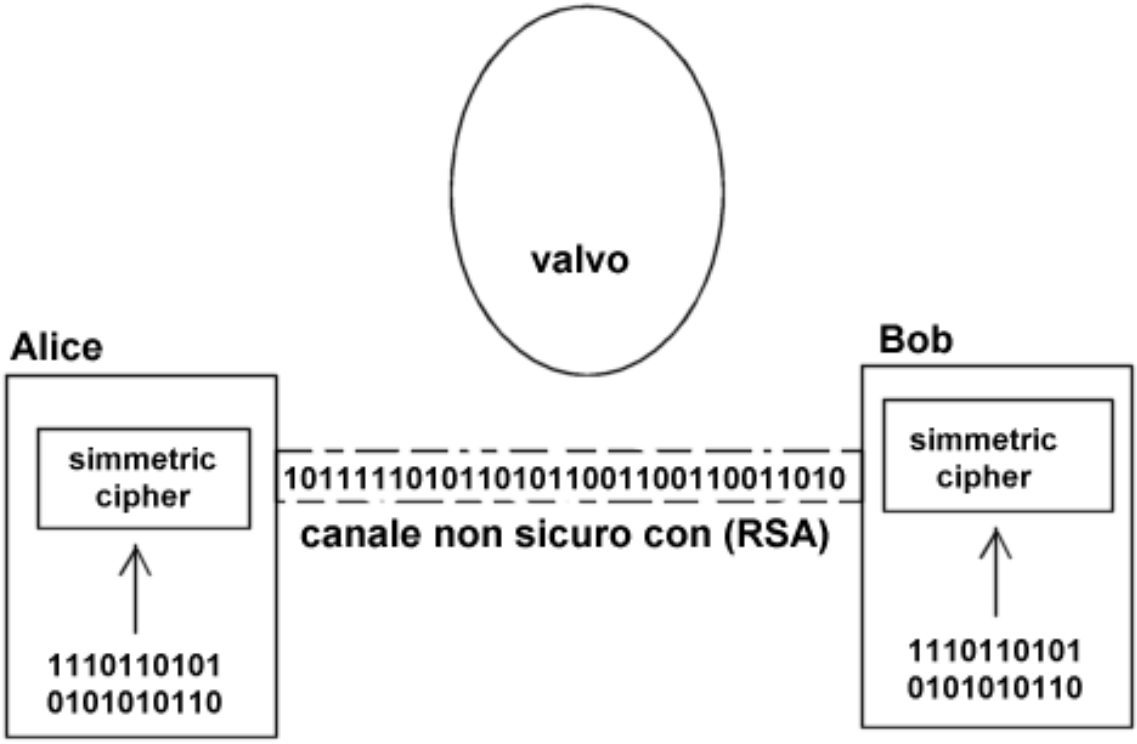
Ad esempio scegliendo **p** e **q** ~= 10^{100} avremmo n ~= 10^{200}, quindi ogni blocco potrebbe contenere fino a 664bit (2^{664} ~= 10^{200}), oppure 83caratteri di 8bit.

1.6.1 RSA : ibrido o non ?

In realtà, per motivi di efficienza, RSA nelle implementazioni commerciali è spesso usato in modo "ibrido" o "a gestione di chiavi" con veloci sistemi di crittografia simmetrica.

viene utilizzato RSA per garantire la sicurezza e l'affidabilità del canale, ed un algoritmo a chiavi simmetriche per realizzare la comunicazione cifrata.

Proviamo a rivedere il nostro prima schema grafico di sistema di crittografia, che avevamo introdotto all'inizio:



Costantino Pistagna (valvoline@vrlteam.org) − crypto workshop − 6May 2k2 − CT

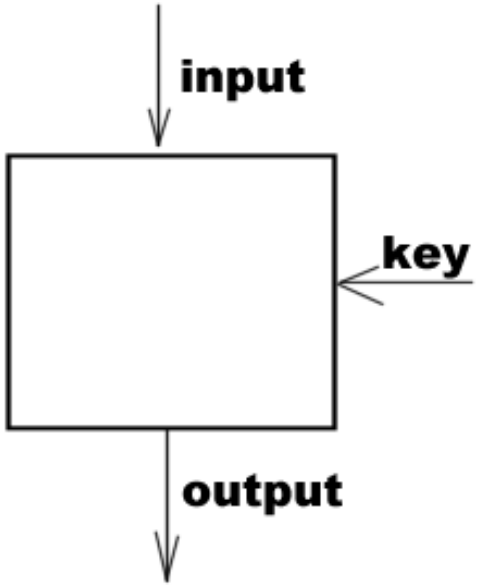
2.1 Ancora sulla crittografia simmetrica: il DES

Alla fine degli anni ’70, il governo degli stati uniti, adottò come standard per tutte le informazioni non classificabili un cifrario composto sviluppato dalla IBM, il **DES**. Allo stato attuale delle cose, l’algoritmo originale di DES, non è più sicuro. Ne è stata dimostrata la facile attaccabilità nel 1995 da **Wayner**. Attualmente se ne usano due forme modificate, **triple DES** e **double DES**.

"L’algoritmo nella sua forma standard e’ un cifrario simmetrico monoalfabetico a blocchi e trasposizione multipla"

Il testo in chiaro viene codificato in blocchi di 64bit, generando 64bit di testo cifrato. L’algoritmo è parametrizzato da una chiave di 56bit, ed ogni cifratura richiede 18iterazione distinte. Il primo passo è una trasposizione. L’ultimo passo è l’esatto contrario della trasposizione iniziale. Nel mezzo, stanno 16passi di iterazione parametrizzati da valori/funzioni differenti della chiave.

"L’algoritmo è stato progettato per permettere di decodificare con la stessa chiave di codifica"



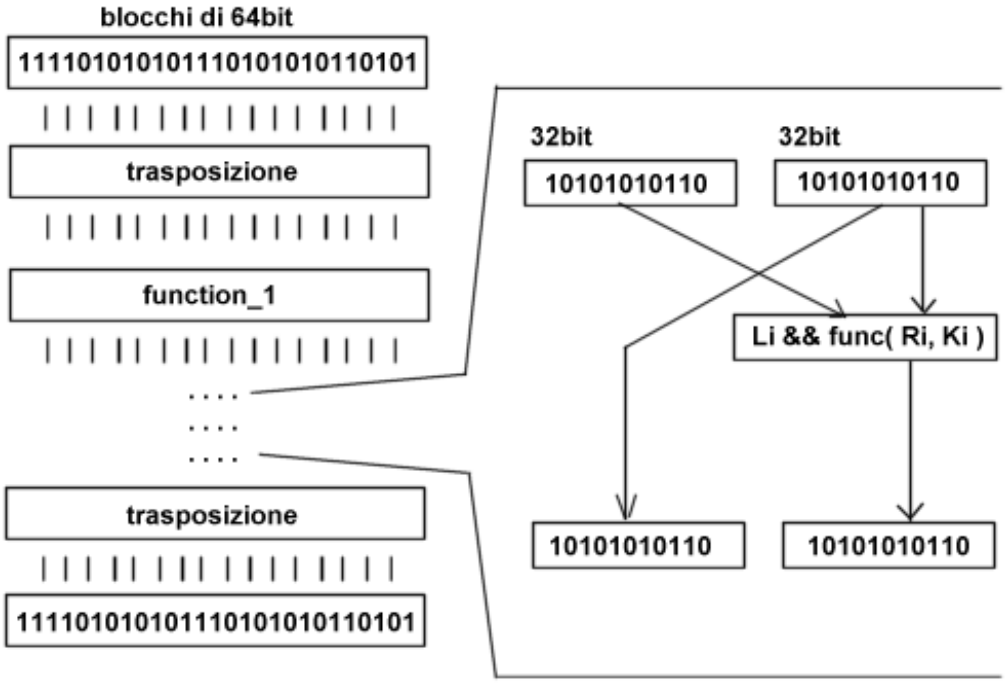
La complessità dell'algoritmo risiede nei 16passi intermedi, tutti computazionalmente identici (cambia solo la chiave. Parametro).

Ogni passo, richiede due ingressi di 32bit, e produce due uscite di 32bit.

- L'uscita a sinistra è una copia dell'ingresso a destra.
- L'uscita a destra è lo XOR dell'ingresso a sinistra, una **funzione** dell'ingresso di destra e la chiave per questo passo, K_i .

La funzione è l'elemento chiave del sistema. L'algoritmo prevede cinque passi distinti svolti in sequenza, per ottenere l'output di destra:

- viene costruito un numero di 48bit, E, espandendo i 32bit di R_i, secondo una permutazione e una regola di duplicazione fissata. Possiamo immaginare che vengano prima duplicati 16bit dei 32bit iniziali, e poi venga permutata la stringa ottenuta.
- Viene eseguito lo XOR tra E e K_i
- L'output viene suddiviso in 8 gruppi di 6bit ciascuno, ognuno viene inviato ad una differente S−Box.
- L'S−Box prende in input un blocco di 6bit e da in ouput una sequenza a 4bit.
- Gli 8 *4 ouput (32bit) passano attraverso un'ulteriore permutazione.



In effetti non e' noto, perche' le funzioni d permutazione e le funzioni di espansione siano affidate ad alcune tabelle fisse.

IP							
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

IP^{-1}							
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

E bit-selection table					
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

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

L'unico punto in cui il DES non utilizza funzioni di tipo lineare (xor, permutazioni, espansioni, etc.etc.) e' nelle S−Box. Che costituiscono, quindi, il punto centrale dell'algoritmo.

Le S−Box, appaiono a molti come qualcosa di misterioso: alcuni asseriscono, addirittura, che esse nascondano in se' delle trapdoor. I criteri di costruzione delle stesse, infatti, non sono a tutt'oggi completamente note.

S_1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Il senso di lettura e' il seguente: Dato X (l'ouput di cui al punto quattro nella descrizione dell'algoritmo), il primo e l'ultimo bit di X, vengono usati come indice di riga, mentre i bit interni come indice della colonna.

vediamo per ultima cosa, come vengono generate le chiavi durante i passaggi di cui abbiamo parlato sopra:

- Data una chiave da 64bit, ignoriamo gli 8 bit di parita', e con i restanti 56bit effettuiamo una permutazione, tramite la tabella:

PC-1						
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	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

  sia PC−1(K) = C_0 D_0 dove C_0 D_0 sono rispettivamente i primi e gli ultimi 28bits della chiave.

- Per 1 <= i <= 16, computiamo:
  C_i=Ls$_i$(C_{i-1})
  D_i=Ls$_i$(D_{i-1})
  Ki=PC−2(C_i D_i)

Dove LS e' uno shift ciclico a sinistra di una posizione a seconda del valore di i (lo shift riguarda solo una posizione alle iterazioni 1, 2, 9, 16, e due posizioni a tutte le altre iterazioni).
PC−2 rappresenta la tabella seguente, ed e' una tabella di compressione di una stringa da 56bits in una stringa da 48bits.

PC-2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

I Bits, che vengono soppressi dalla compressione sono quelli in posizione 9, 18, 22, 25, 35, 38, 43 e 54.

2.1.2 il DES o Cesare ?

A dispetto, di tutta la complessità computazionale appena vista, il DES è essenzialmente un cifrario a sostituzione monoalfabetico, che utilizza in effetti un carattere da 64bit. In pratica, ad ogni testo in chiaro, corrisponde sempre lo stesso testo cifrato.

riuscite ad immaginare cosa possa fare il nostro, attaccante ?

iniziamo con il codificare il messaggio, suddividendolo in blocchi da 64bit e codificando ogni singolo blocco. Una volta codificati tutti i blocchi, il nostro attaccante, entra in possesso del testo crittografato. In che modo, puo' comprometterne la sicurezza ?

Supponendo che il nostro attaccante conosca il formato del testo in ingresso, puo' facilmente RI−suddividere i blocchi cifrati, e scambiarli in maniera opportuna, senza per questo compromettere la validita' del testo cifrato!

Quello sopra esposto è un primo attacco che può essere effettuato su DES.

In realtà DES, lavora in due modalità differenti ECB (Electronic Code Book mode) e CBC (Chained Block Cypher mode).

La modalità di cifratura esposta sopra, sofferente all'attacco spiegato, viene classificata come ECB. Il motivo per cui esiste CBC, credo sia evidente...

Esiste un ulteriore problema che riguarda DES. Sebbene DES possa apparire complicato come metodo per cifrare un messaggio in chiaro, nasconde una caratteristica molto importante:

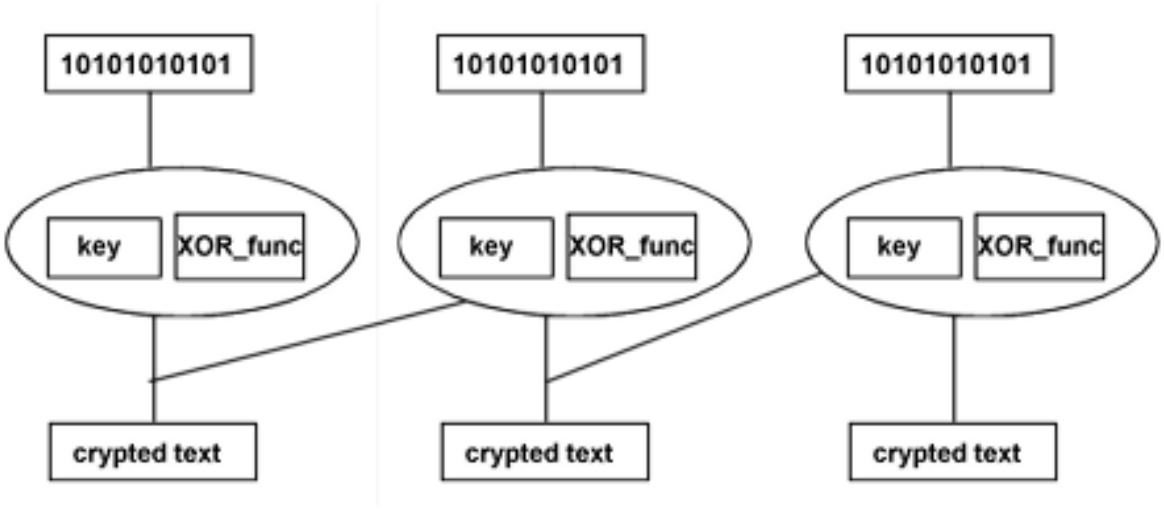
DES^−1 è lo stesso algoritmo DES, fatta eccezione per la generazione delle chiavi, che viene invertita.

Invertiremo allora, gli shift da sinistri a destri, nei passi di generazione delle chiavi, e lavorevemo con coppie in ordine inverso per riottenere il messaggio originale.

2.2 DES CBC

Per prevenire il tipo di attacchi sopra esposti, il DES (come tutti i cifrari di un certa rilevanza a blocchi), può venire concatenato in vari modi così che sostituire un blocco nella maniera usata dal nostro attaccante, possa provocare soltanto la cancellazione del testo in chiaro a partire dal blocco rimpiazzato. Nel sistema usato da DES (**CBC**), ogni blocco di testo in chiaro viene messo in XOR con il precedente blocco di testo cifrato prima di essere codificato. Di conseguenza lo stesso blocco di testo in chiaro non corrisponde allo stesso blocco di testo cifrato, e la codifica non è piu' un grosso cifrario per sostituzione monoalfabetico!.

Per la prima iterazione, il blocco di testo in input viene messo in XOR con un vettore di inizializzazione, scelto in maniera random e trasmesso insieme al testo cifrato.



3.1.1 − Forza, compattezza e proprietà di DES

Le caratteristiche auspicabili per un sistema di cifratura a blocchi sono svariate; DES riesce a soddisfarne la maggior parte, rendendolo in effetti un sistema abbastanza sicuro (sotto certe restrizioni, che vedremo in seguito). Vediamone alcune in dettaglio:

- Ogni bit del testo cifrato deve dipendere esclusivamente da tutti i bits della chiave e tutti i bit del testi in chiaro; in altre parole, non deve esistere una relazione predicibile tra il testo in chiaro ed il testo cifrato

- Alterando un singolo bit del testo in chiaro/chiave, dovrebbe essere alterato un qualunque bit del testo cifrato con probabilità 1.

- Alterando un singolo bit, l'output risultante dovrebbe essere impredicibile.

Alcune caratteristiche al DES, riguardano la sua sicurezza, ed in particolare le S−Box.

- È difficile (se non impossibile), dimostrare che ci siano delle trapdoor dietro la costituzione delle S−Box.
- C'e' gente che spera di trovare delle proprieta' sulle S−Box.
- Perche' un solo shift alle posizioni 1, 2, 9, 16, ?
- Perche' solo 16 iterazioni ? (Biham e Shamir, dimostrarono nel 90, che utilizzando l'analisi differenziale con meno di 16 iterazioni e' facile crackare DES con un attacco di testo conosciuto in chiaro. Con 16 iterazioni il brute−force e' equivalente all'attacco di Biham e Shamir).
- La lunghezza della chiave e' troppo piccola. Sono noti attacchi su chiavi di 64bit! (anche statistici, sulle probabilita' del testo).
- Le weak−key o chiavi di involuzione

Costantino Pistagna (valvoline@vrlteam.org) − crypto workshop − 6May 2k2 − CT

Attacchi a DES

Un primo possibile attacco a DES, e' quello di forza bruta. Questo tipo di attacco e' universale, nel senso che puo' essere applicato a qualunque algoritmo crittografico.

Innanzitutto, a dispetto delle 2^{56} possibili combinazioni, in media bastano 2^{55}(circa la meta') prove!

Il DES nella sua modalita' operativa standard, codifica 64bits con chiavi a 56bits e produce un output di 64bits. Un attacco esaustivo richiederebbe di provare tutte le 2^{56} possibili chiavi. Con :

O(2^{56}) complessita' in tempo
K spazio costante

Costruire una macchina che riesca a svogere il lavoro in maniera parallela su 56 processori in tempo accettabile, propone problemi del tipo: costi, grandezza, tempo, il gioco vale la candela ?

Cambiando strategia di attacco.. Su una macchina ad un solo processore, i costi computazionali asintotici sono sempre dello stesso ordine:

O(2^{56}) complessita' in tempo
K spazio costante

Proviamo, allora, a risparmiare sul tempo, aumentando lo spazio, nella maniera seguente:

Per un testo dato in chiaro:

- Precomputiamo $y_k = DES_k(x)$ per ogni chiave K tra le 2^{56} possibili.
- Manteniamo una tabella di coppie (y_k, K) ordinate per testo cifrato.
- Dopo aver ottenuto y (il testo cifrato con chiave sconosciuta), cerchiamo y nella tabella e teniamo memoria della chiave k associata. Questa e' la chiave cercata.

Se la ricerca e' fatta in modo binario, il tempo diventa logaritmico: **O(lg n)**
Se la tabella per la ricerca e' mantenuta con una tabella HASH, il tempo diventa, addirittura costante: **O(k)** !!

Ovviamente, costruire una macchina ed un sistema di ricerca per un spazio cosi' esteso e' cosa gravosa almeno quanto costruire una macchina con 56 processori in parallelo.

- ## Attacco Distribuito

Verser il 13 Marzo del 1997, porto' a compimento un'opera che rivedette tutte le affermazioni fatte sino ad allora dall'NSA, riguardo la crittografia ed il DES. In soli 39 giorni, grazie ad un sistema distribuito, che partizionava lo spazio delle chiavi possibili, tra tutti i volontari collegati tramite internet riusci' a crackare il DES.

- ## Attacco tramite CHIP

DES ha fatto il suo tempo. L'ultima sfida ha crackato in un tempo record di 22ore il sistema!

3.1.2 − CryptoAnalisi − Forzare le regole è bello

Supponendo che l'attaccante possa solo ascoltare i messaggi e non modificarli, i metodi di attacco ai crittosistemi sono universalmente riconosciuti come:

- **Cyphertext−only**: ascoltare i messaggi in transito, cercando di crackare il sistema
- **Known plaintext**: l'attaccante conosce alcune coppie (x,y) rispettivamente messaggio in chiaro e messaggio cifrato.
- **Chosen plaintext**: l'attaccante oltre a conoscere le coppie (x,y), puo' scegliere i messaggi in chiaro di cui vuole ottenere i testi cifrati.
- **Chosen ciphertext**: e' simile al chonse plaintext, stavolta pero' l'attaccante ha a disposizione una machina che decifra i messaggi.

Costantino Pistagna (valvoline@vrlteam.org) − crypto workshop − 6May 2k2 − CT

3.1.3 − CryptoAnalisi − L'analisi statistica

La struttura implicita di qualunque linguaggio, permette ad un cryptoanalista di controllare patterns e lessemi che derivano da messaggi in chiaro. Semplicemente, controllando questo paragrafo, ad esempio, ci si accorge, che la vocale piu' frequente e' la E, la consonante piu' frequente e' la c, e via dicendo...è possibile stilare una tabella statistica, su campioni di testo molto grossi, in maniera da avere una sorta di costante per ogni linguaggio, a cui riferirsi durante le proprie cryptoanalisi.

Scrivere un analizzatore di frequenze, non è un'impresa difficilissima:

```
char *frequency(char *input, int ilen, char *table) {
  register int I, j; char a,b;
  for(i=0;i<ilen;i++) table[ input[i] ]++;
  return table;
}
```

Ovviamente, la tabella, deve contenere un'indicizzazione propria delle lettere passate, cosi' dovremo avere almeno 255 elementi nella nostra tabella (i char, vanno da 0 a 255), in maniera da comprendere ogni carattere possibile. La tabella, andra' prima azzerata, in maniera da avere in output solo le occorrenze presenti.

Lavoro piu' difficile, diventa, quello di costruire un analizzatore di n−grammi (bigrammi, trigrammi, etc.) Stavolta non bastera' piu' un semplice array indicizzato per contenere le nostre occorrenze. Dovremmo considerare le possibili disposizioni di due lettere: 255^2 ~= 64k. Ovviamente, è possibile, inizializzare solo particolari combinazioni di n−grammi. Un metodo sarebbe quello di creare una lista linkata o una tabella hash, per ogni bigramma incontrato, cosi' da scartare tutte le combinazioni di possibili bigrammi inutili (nella lingua italiana, i comuni possibili bigrammi sono solo il 2% dei bigrammi presenti. Nella lingua inglese sono solo il 3%). Conti alla mano l'impresa diventa fattibile. Se poi si aggiunge un qualkosa ke lascia in memoria solo le occorrenze maggiori di un certo k. Allora, diventa impresa facile stilare un analizzatore di questo tipo.

Ad ogni modo esistono software appositi, che si preoccupano di analizzare un linguaggio dato in un file, ed estrapolare tutte le informazioni del caso.

Una volta avuto il numero e l'entita' dei bigrammi/trigammi piu' importanti, possiamo costruire un nostro analizzatore di frequenze, utilizzando una array di array o meglio ancora, una qualunque struttura dati dinamica, in maniera da potere indicizzare per ogni possibile bigramma/trigramma la nostra frequenza di occorrenza.

Costantino Pistagna (valvoline@vrlteam.org) − crypto workshop − 6May 2k2 − CT

3.1.3 − String Matching Approssimato

Abbiamo parlato di cryptoanalisi sul testo, derivando il tipo di lessemi/n−grammi utilizzati, per poi utilizzare quest'analisi statistica sul testo cifrato. Esiste un'altro metodo, molto efficente, per poter riconoscere il tipo di linguaggio ed interpretare statisticamente quello che si ha di fronte. Il nostro nuovo obiettivo sara' quello di classificare parole simili ortograficamente in classi.

Ovviamente abbiamo bisogno di un metodo, per poter associare parole simili ortograficamente. Ci rifaremo ad un lavoro di due ricercatori: Adamson e Boreham. Il metodo originale inventato dai due, consiste nella seguente formula:

$$\frac{2 \times |\text{bigrammi}(x) \cap \text{bigrammi}(y)|}{|\text{bigrammi}(x)| + |\text{bigrammi}(y)|}$$

La funzione bigrammi è una funzione che riduce una parola in un insieme di bigrammi. Ad esempio:

bigrammi(finestra) = {fi, in, ne, es, st, tr, ra} = 7
bigrammi(modifica) = {mo, od, di, if, fi, ic, ca } = 7

- bigrammi(finestra) ∩ bigrammi(modifica) = { fi }

il risultato allora sarà: (2 x 1) / (7 + 7) ~= 0.143

utilizzando questa tecnica, e' possibile costruire un analizzatore, che ci dice il grado di parole ortograficamente simili in un testo. Rendendo di fatto l'analisi del testo cifrato molto piu' semplice.

3.1.4 CryptoAnalisi di cifrari a sostituzione.

In questo attacco si suppone che il testo in chiaro $X=(x_1,x_2,...,x_n)$ sia in linguaggio naturale. Nella cifratura del nostro secondo sorgente, ad esempio, il testo cifrato $Y= (y_1,y_2,...,y_n)$ è ottenuto tramite la chiave K, dividendo il testo in chiaro in blocchi di m caratteri ed effettuando la somma modulo 26 (o 97) carattere per carattere.

- Il messaggio in chiaro deve essere >> della lunghezza della chiave, in maniera da poter mettere in risalto particolari regolarita' statistiche dovute alla divisione e cifratura in blocchi.
- Nella fattispecie, se si utilizza un sistema con blocchi di lunghezza = 1 (un sistema pseudo−stream), l'analisi statistica diventa molto semplice.

Per calcolare la chiave dobbiamo prima calcolare la sua lunghezza e poi i valori numerici assunti da ciascuna componente della chiave.

Kasiski, risolse il problema della lunghezza della chiave, fornendo un metodo che va' alla ricerca di alcune regolarita' che si ripetono all'interno del testo cifrato.

Il test di kasiski e' basato sull'osservazione che due segmenti identici del testo in chiaro vengono cifrati allo stesso modo solo se la porzione di chiave atta a cifrarli e' la stessa!

Se troviamo due segmenti identici del testo cifrato, ciascuno di lunghezza almeno 3, allora ci sono buone probabilita' che il testo in chiaro sia lo stesso.

Il test di kasiski cerca le coppie di segmenti identici nel testo cifrato di lunghezza almeno 3 e memorizza le distanze tra le posizioni iniziali dei due segmenti. Se si ottengono le distanze d1, d2, ..., dn, allora si puo' dedurre che m (lunghezza della chiave), divida il GCD dei di.

In questo modo possiamo solo ottenere informazioni sulla lunghezza della password.

Costantino Pistagna (valvoline@vrlteam.org) − crypto workshop − 6May 2k2 − CT

Un secondo attacco ai cifrari per sostituzione, si basa sul calcolo delle statistiche relative al testo cifrato. Esso consiste di due passi:

- Calcolo della lunghezza della password
- Calcolo del suo valore

Ricordiamo che la probabilita' si puo' calcolare come:

(numero eventi favorevoli) / (numero di eventi possibili)

Data una stringa X=x1, x2, ..., xn di n caratteri, il numero di modi di scegliere due caratteri e' (n 2).

Se indichiamo con f0, f1, ..., fn il numero di occorrenze dei caratteri da A a Z in X, allora il numero di modi di scegliere due caratteri in X, entrambi uguali ad A è (f0 2), entrambi uguali a B, sara' (f1 2), etc.etc.

Otteniamo il numero di casi favorevoli:

(f0 2) + (f1 2) + ... + (fn 2). Per cui avremo che:

$$IC(X) = i=0...25 \quad \sum (fi(fi-1)) \ / \ n(n-1)$$

Supponendo di conoscere la distribuzione tipica dei caratteri grazie al nostro codice sorgente, si puo' calcolare il valore medio di IC(X).

Dette p0, p1, , pn le probabilita' delle occorrenze delle lettere 'A', 'B', , 'Z', avremo che la probabilita' che scelte a caso due posizioni in un testo qualunque, le lettere occupanti tali posizioni siano due A è $p0^2$, etc.etc.

Si ha che:

$$IC(X) \sim= i=0..25 \quad \sum (pi^2) \quad \sim= 0.075$$

Se supponiamo che il testo cifrato, venga cifrato sempre con la stessa chiave, l'indice di coincidenza del testo cifrato sara' uguale a quello in chiaro.

Se in un testo i caratteri sono scelti a caso, tutte le Pi, saranno pari a 1/26. e quindi l'indice di coincidenza per questo testo sara:

$$IC(X) \sim= i=0..25 \quad \sum ((1/26)^2) \quad \sim= 0.038$$

Se l'indice di coincidenza del messaggio, cioe' e' diverso da 0.075 e vicino a 0.038 allora la chiave usata per cifrare il messaggio in chiaro non e' la stessa. Si puo' in questo modo determinare correttamente la lunghezza della chiave.

3.1.5 − CryptoAnalisi Differenziale

Nel 1991 Biham e Shamir pubblicarono un articolo, che in effetti era il primo metodo nella letteratura in grado di crackare il DES senza effettuare una ricerca esaustiva sulle chiavi. Questo metodo, chiamato CriptoAnalisi Differenziale risulto' (e risulta) ad oggi essere un valido metodo per la ricerca su sistemi crittografici in qualche modo legati a DES.

La CryptoAnalisi Differenziale, si basa sull'attacco che abbiamo classificato come Known −Plaintext . Il senso, è di prendere blocchi di testo in chiaro, che il cryptoanalista conosce e provare a determinare la natura dell'algoritmo di cifratura, cifrandoli con chiavi random.

In effetti, su larghe basi di testi in chiaro iniziali, è possibile determinare dipendenze o legami statistici tra le chiavi usate ed il testo cifrato. In questo modo, saremo agevolati nel capire in che modo l'algoritmo agisce, ed in ke modo un cambio di alcuni bits nella chiave, altera l'output.

Alla fine, dopo aver indovinato un accettabile numero di Bits della chiave, e' possibile lasciare il resto ad un attacco di forza bruta.

Una nota, riguardo questo metodo, riguarda l'esigenza di avere COPPIE di input in chiaro conosciute in maniera che possano essere comparate.

Visto che abbiamo a che fare con coppie di testo in chiaro, dobbiamo avere un modo per correlare questi input ed il loro equivalente cifrato. Usiamo la seguente:

E(X) .xor. E(X*) = E(X .xor. X*)

Se X è un testo in chiaro, allora X* è l'altro testo in chiaro associato per le comparazioni. Denotiamo, inoltre X .xor. X*, come la differenza tra X ed X*.
La sottochiave, inoltre, viene XORata con l'output della funzione E.

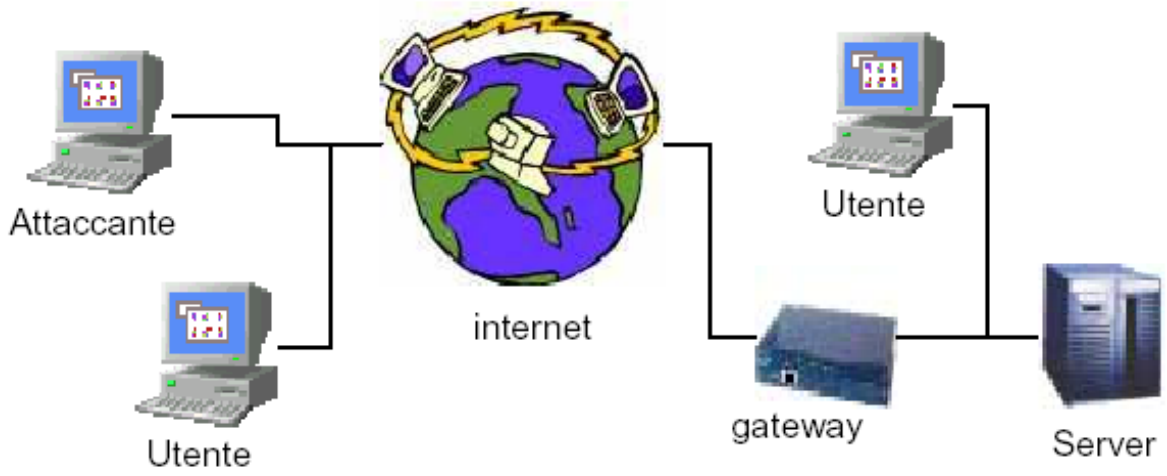
Dobbiamo anche avere una maniera per mettere in correlazione due testi in chiaro, che vengono XORati con la stessa chiave:

(X .xor. K) .xor. (X* .xor. K) = X .xor. X*

Un attacco di analisi differenziale, inizia costruendo una tabella di distribuzione. Questa tabella, e' un grafico/tracciato degli XOR di input e gli XOR di output, e quante possibili coppie esistono con quello stato. (0 = identici, etc.etc.). Utilizzando questa tabella ad ogni passo di iterazione, e' possibile ottenere informazioni riguardo la chiave utilizzata.

Lo XOR della chaive con il testo in chiaro, LASCIA INFORMAZIONI SENSIBILI SUGLI INPUT UTILIZZATI!.

Altri Attacchi possibili

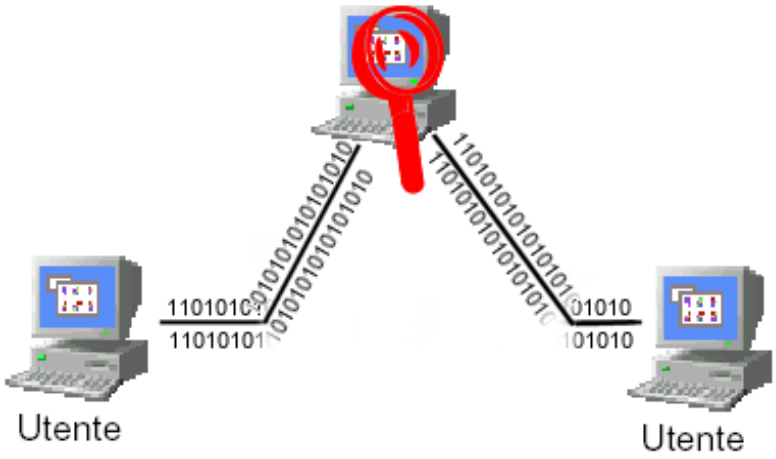


Oltre agli attacchi puramente matematici che abbiamo presentato sopra, il modo reale e gli attaccanti, si basano su risorse e materiale, non propriamente dimostrato "matematicamente" !

- Chiunque su internet e' sottoposto ad attacchi di tipo passivo sulla propria sottorete (basta stare li', ed osservare i pacchetti che transitano).

- **Man in The Middle** (l'uomo che visse due volte).

In effetti, la situazione e' piu' disperata...



Costantino Pistagna (valvoline@vrlteam.org) − crypto workshop − 6May 2k2 − CT

3.1.5 RNG, PRNG, TRNG − l'arte della confusione quantistica

Un buon sistema crittografico, richiede un buon generatore di numeri casuali. Qualunque sistema di cifratura, richiede la generazione e l'uso di una porzione di codice sconosciuto all'attaccante. Numeri random, sono utilizzati per la costituzioni di chiavi pubbliche, di chiavi private.

Numeri random, sono utilizzati per operazioni di salting, di noising, per inizializzazione di sfide. I sistemi onetime, richiedono che le chiavi vengano generate da un sistema REALMENTE random.

Dato che gran parte della sicurezza di un sistema si basa su questi generatori, essi devono assicurare alcuni standards, che spesso non riescono a garantire! ✷ Cerchiamo di vedere prima in linea teorica, e poi in linea di un attaccante come funzionano e cosa sono i generatori di numeri primi.

In accordo con la teoria di Shannon, l'entropia **H** di qualunque messaggio è data da:

$$H = -K \sum_{i=1}^{n} p_i \log p_i$$

dove Pi e' la possibilià dello stato i su n possibili stati e K è una costante, che serve per rapportare il valore risultate dalla sommatoria al numeri di bits.

Si noti che necessariamente 0 < Pi < 1, e quindi logP < 0.

Nel caso di un sistema di generazione random la H di sopra produce un risultato binario di k−bit, e Pi è la probabilità che un output sarà uguale ad i. dove 0 <= i <= 2^k.

Nel contesto di un sistema di cifratura, valutando l'incertezza prima e dopo la cifratura di un messaggio è possibile stimare la quantità di informazione ad esso associabile in termini di differenza fra l'entropia a priori e l'entropia a posteriori.

Per comprendere meglio assimiliamo l'invio di un messaggio ad un "gioco" con più risultati mutuamente esclusivi {Mi}, con una distribuzione di probabilità {pi}. Il generico messaggio verrà scelto all'interno di un certo insieme di messaggi ammissibili: ad esempio, un messaggio di un "bit" verrà scelto nell'insieme {0, 1}, una lettera dell'alfabeto verrà scelta nell'insieme {a, b, ..., z}, una parola di lunghezza fissata sarà costituita di lettere scelte all'interno di un certo alfabeto.

Possiamo affermare allora che l'informazione convogliata da quella particolare realizzazione del messaggio che ha probabilità pi di presentarsi è numericamente uguale alla sua incertezza prima della realizzazione, in quanto l'incertezza può anche venire interpretata come potenziale sorpresa: sorpresa relativa, appunto, al realizzarsi di quella possibilità.

In altre parole, il contenuto informativo (la sorpresa) del singolo messaggio sarà tanto maggiore quanto maggiore era l'ignoranza (l'incertezza) preventiva su di esso.

In parole povere, nel caso di un perfetto sistema random Pi = 2^(−n), e l'entropia H dell'output è uguale a k bits. Cioe' tutti i possibili output sono equamente probabili e l'informazione di output non puo' essere rappresentata con meno di k−bits. In contrasto a questo, l'entropia tipica di un testo italiano e' di circa 1.5bits/chars e quella di un testo inglese circa 1.8bits/chars.

Costantino Pistagna (valvoline@vrlteam.org) − crypto workshop − 6May 2k2 − CT

I sistemi di generazione random dei calcolatori, utilizzano un generatore pseudorandom (PRNG). Questi ultimi, utilizzano processi deterministici per generare una serie di outputs da uno stato iniziale (seed). Poiche' l'ouput è semplicemente una funzione del seed iniziale, l'entropia dell'output non puo' mai superare l'entropia dell'input!

Ad esempio, un PRNG inizializzato con un seed da 256bits, non potra' generare piu' di 256bits di output realmente random. Un attaccante che indovina il seed iniziale, puo' indovinare tutte le successive sequenze, senza sforzo computazionale eccessivo. Ovviamente indovinare una sequenza iniziale di 256bits, è uno sforzo computazionale esagerato, quindi possiamo considerare ragionevolmente sicuri PRNG basati su seed iniziali con un numero di bits >= di un certo N.

Un sistema TRNG (True Random Number Generator), utilizza una sorgente non−deterministica per produrre un output. E' possibile, ad esempio, usare sorgenti quali (rumore sonoro, rumore visivo, osservazioni atmosferiche, etc.etc.). Un sistema PRNG, senza alla base un vero TRNG, puo' essere considerato insicuro.

Ci sono varie tecniche, utilizzate in maniera piu' o meno efficiente per inizializzare un generatore random. PGP, ad esempio, richiede di inserire per circa 15secs numeri random e movimenti del mouse. Questo tipo di sistemi, diventano insicuri quando affidati a scripts di automazione.

Un altro metodo consiste nell'utilizzare caratteristiche intrinseche della macchina ad un dato periodo (numero di settori vuoti del disco, cilindri del disco principale, numero di serie della macchina, etc.etc.).

A dispetto, di tutta questa complessita' il problema piu' generale dei sistemi di crittografia, e' la generazione di numeri random realmente casuali.

Bruce Schneier: *"Good Random Generators, are hard to design, because their security often depends on the particulars of the hardware and software. Many products we examine use bad ones."*

Il sistema di generazione di numeri casuali di Netscape, si basa su tre elementi principali: La data attuale, l'ID del processo e l'ID del processo padre.

Un avversario, che vuole predire l'insieme di numeri casuali generati ad una dato istante, puo' utilizzare l'algoritmo di MD5 per computare l'esatta sequenza generata.

La misurazione della casualita' di un input è in effetti un dibattito filosofico ed un argomento di divulgazione matematico.

E' impossibile provare se un insieme finito di elementi e' realmente random.

Il valore a 128bit "0" è tanto probabile quanto il valore esadecimale: "7eh12418hh182377a"!!

Costantino Pistagna (valvoline@vrlteam.org) − crypto workshop − 6May 2k2 − CT

3.1.6 − Visual Cryptography − L’arte di nascondersi

Si tratta di una tecnica millenaria, che a Sparta ed Atene usava tavolette di legno incise e poi ricoperte di cera: a prima vista sembrava non ci fosse scritto niente ma poi, una volta rimossa la copertura, il messaggio veniva alla luce. Dopo essere stata utilizzata anche durante la seconda guerra mondiale oggi e’ ritornata in voga con Bin Laden.

A differenza della crittografia, la steganografia non cifra le informazioni. Le nasconde solamente. Potremmo dire che la steganografia e’ l’ arte di nascondere le informazioni all’ interno di altri dati. Per esempio, le informazioni possono essere nascoste all’ interno di immagini, file sonori o video.

Tutte le immagini computerizzate (digitali) sono basate su una successione di punti, chiamati pixels, che costituiscono una griglia molto fine in grado di disegnare un’immagine con notevole accuratezza. Ciascuno di questi pixels ha un proprio colore, rappresentato come quantità separate di rosso, verde e blu. Ciascun colore , per esempio, può variare tra 256 livelli compresi tra 0 (nessun colore e 255 un colore saturo).

Un pixel con una terna RGB di 0, 0, 0 è nero; uno con una terna di 255, 255, 255 è bianco.

Un sistema a 24 bit (3 byte) divide questi in 3 gruppi di 8 bit (1 byte) per definire il rosso (Red), verde (Green) e blu (Blue). Il risultato è 28= 256 possibili variazioni (per saturazione e luminosità) di rosso ed altrettante di verde e blu. La loro combinazione fornisce 256x256x256=16.777.216 colori teorici differenti.

In realtà, la maggior parte delle immagini non necessita di un così elevato numero di colori e già 40.000 colori sono un valore considerevolmente alto. Questo significa che la stessa immagine a 24 bit potrebbe essere memorizzata con 16 bit (per ogni pixel) senza differenze percepibili. Gli 8 bit (24−16) che risparmiamo su ogni pixel (codificati in modo opportuno) possono essere utilizzati per qualcosa di differente: occultare un testo in un’immagine!

Per un’immagine a 24 bit, l’occultamento di un file è semplice perchè 24 bit sono immagazzinati internamente come terne RGB, le quali − come abbiamo visto − sono sovrabbondanti. Così, tutto ciò che è necessario è distribuire i nostri bit da occultare nell’immagine scelta e salvare il tutto in un nuovo file.

E’ notevolmente più difficile nascondere qualche cosa in un’immagine a 256 colori (8 bit). Questo perchè l’immagine già di per sé può avere 200 colori, che la nostra intromissione porterebbe al di sopra del massimo assoluto di 256 colori rendendo impossibile un’operazione di occultamento. Tuttavia, si può verificare facilmente che la perdita di qualità conseguente alla riduzione di colori di un’immagine da 16,7 milioni di colori (24 bit) a 256 colori (8 bit) è molto maggiore di quanto sia lo scadimento da 256 (8 bit) a 32 (5 bit) colori.

Per contattarmi...

valvoline@vrlteam.org
valvoline@tiscalinet.it
valvoline@s0ftpj.org

http://www.vrlteam.org

http://www.freaknet.org