

**POLITECNICO DI MILANO**  
Dipartimento di Elettronica ed Informazione

---

ELABORATO DI LAUREA

# **CRITTOLOGIA ED A.R.C.C**

*Storia della crittologia passata e contemporanea e  
sviluppo di un algoritmo di cifratura simmetrica*

**RELATORE**  
**Prof. Fabrizio Ferrandi**

**AUTORI**  
Angelo Rosiello – Roberto Carrozzo

# CONTENUTI

.....	1
<b>AUTORI.....</b>	<b>1</b>
<b>CONTENUTI.....</b>	<b>2</b>
<b>1 INTRODUZIONE.....</b>	<b>3</b>
1.1 La crittologia.....	3
1.2 Perché Cifrare e cosa Cifrare.....	4
1.3 Cifratura Reversibile & Irreversibile.....	5
1.4 L'algoritmo Perfetto.....	5
1.5 Sviluppo di A.R.C.C: Idea, Scopi ed Utilizzi.....	6
<b>2 ACCENNI STORICI SULLA CRITTOGRAFIA.....</b>	<b>7</b>
2.1 Definizioni utili.....	7
2.1.1 Cosa è la Crittografia.....	7
2.1.2 Cosa è la Crittanalisi.....	9
2.2 Il codice di Atbash.....	12
2.3 La scacchiera di Polibio.....	12
2.4 Cifrario di Cesare.....	13
2.5 Codici Medioevali.....	14
2.6 Il Disco di Leon Battista Alberti.....	15
2.7 Giovan Battista della Porta.....	16
2.8 Giovanbattista Bellaso.....	17
2.9 Il Cifrario di Vigénère.....	18
2.10 Codice di Jefferson.....	20
2.11 Il Cifrario di Playfair.....	21
2.12 Il Cifrario bifido di Delastelle.....	23
2.13 Il Cifrario di Vernam.....	24

2.14 Enigma.....	25
<b>3ALGORITMI DI CIFRATURA ODIERNI.....</b>	<b>27</b>
<b>3.1Cifrari a Blocchi.....</b>	<b>27</b>
3.1.1 Concetti Generali.....	27
3.1.2 DES.....	28
3.1.3 IDEA.....	30
3.1.4 AES.....	31
3.1.5 Blowfish.....	32
<b>3.2 Cifrari a Flusso.....</b>	<b>32</b>
3.2.1 Concetti Generali.....	32
3.2.2 RC4.....	34
3.2.3 SEAL.....	34
3.2.4 ORIX.....	34
<b>3.1Cifratura a Chiave Pubblica.....</b>	<b>35</b>
3.3.1 Concetti generali.....	35
3.3.2 RSA.....	36
<b>4 SVILUPPO DI UN NUOVO ALGORITMO DI CIFRATURA SIMMETRICA: A.R.C.C... 37</b>	<b>37</b>
<b>4.1 Scelta del linguaggio di programmazione.....</b>	<b>37</b>
<b>4.2 Implementazione.....</b>	<b>38</b>
4.2.1 Numeri casuali.....	38
4.2.2 Approccio alla funzione Polialfabetica.....	40
4.2.3 Approccio allo “XOR”.....	44
<b>4.3 Crittanalisi.....</b>	<b>49</b>
4.3.1 Sicurezza del Livello Polialfabetico.....	49
4.3.2 Sicurezza dello Strato “XOR”.....	51
4.3.3 Sicurezza Complessiva dell’Algoritmo.....	52
<b>5CONFRONTI CON ALTRI ALGORITMI.....</b>	<b>53</b>
<b>5.1 Confronti e Analisi delle Prestazioni.....</b>	<b>53</b>
<b>6 CONCLUSIONI E SVILUPPI FUTURI.....</b>	<b>59</b>
<b>BIBLIOGRAFIA.....</b>	<b>63</b>

## 1 Introduzione

### 1.1 La crittologia

La scienza delle scritture segrete si chiama **Crittologia** e si divide in due grandi branche la **Crittografia**(1.1.1), cioè l'arte di scrivere messaggi che siano segreti e quindi indecifrabili e la **Crittanalisi**(1.1.2) che è viceversa l'arte di decrittare i messaggi segreti.

All'origine di questa scienza vi è la necessità di nascondere, rendere incomprensibili ed inusabili dei messaggi e delle informazioni importanti. Nell'era moderna dominata dai computer, questa necessità è maggiore di quella avvertita una volta, quando la crittografia veniva utilizzata solo per la corrispondenza militare ed in tempo di guerra. Oggi, anche chi crede di non avere niente da difendere, deve comunque difendersi da uno spionaggio continuo, dalle minacce provenienti da Internet, da reti locali, dai colleghi di lavoro, anche semplicemente per la corrispondenza elettronica privata. Per non parlare di chi utilizza il computer per documenti importanti, per segreti industriali e numeri di carte di credito. Difficilmente un computer non contiene informazioni riservate, a partire da informazioni contabili, fino ad arrivare a informazioni che riguardano le proprie preferenze. Non sono poi da trascurare i danni che potrebbero derivare, ad esempio, dalla trafuga della propria password di posta elettronica, o dai danni recati a terzi per non curanza delle informazioni ricevute. Chi crede di non avere niente da nascondere, si arrabbierebbe comunque moltissimo scoprendo che all'ufficio postale hanno aperto una sua lettera. In Internet succede continuamente.

## **1.2 Perché Cifrare e cosa Cifrare**

Tutti noi abbiamo qualcosa di personale, privato e non sono affari di nessuno tranne nostri. Vorremmo poter pianificare una campagna politica, discutere delle nostre tasse, dati molto importanti da preservare, qualunque cosa sia, non vogliamo che i nostri documenti confidenziali vengano letti da altri. Non c'è niente di sbagliato nel voler affermare il proprio diritto alla riservatezza.

La riservatezza è parte della nostra vita ed è garantita dalla Costituzione.

Oggi giorno i documenti digitali hanno sostituito la maggior parte di quelli cartacei, ma quanto tutto ciò è un bene? Quanto è sicuro avere sul proprio computer documenti importanti?

Tutti i giorni i giornali ci tartassano di storie di “hackers”, che penetrano sistemi informatici “rubando” informazioni private. C'è poco da stupirsi, i livelli di sicurezza attuali sono ancora molto bassi. Basti pensare che perfino i log di tutti i sistemi operativi odierni, unico baluardo dell'amministratore, che consentirebbero di comprendere quanto accade al proprio computer, quando lavora, sono ancora in chiaro e potrebbero essere modificati da qualunque “smanettone” a proprio arbitrio. Appare quanto mai evidente la necessità di cifrare i dati importanti.

Le attività di ricerca nel settore della crittologia rappresentano il futuro dell'informatica, poiché è indispensabile garantire agli utenti e/o clienti la libertà di utilizzare la tecnologia, senza incorrere in situazioni quanto mai irreali e sgradevoli!

### 1.3 Cifratura Reversibile & Irreversibile

La cifratura di un messaggio può avvenire in due modi: *in maniera reversibile*, cioè dal messaggio cifrato è possibile tornare al messaggio in chiaro, o in maniera *irreversibile*, cioè dal messaggio cifrato non è più possibile tornare al messaggio in chiaro. Scegliere tra i due tipi di cifratura dipende in ogni caso da ciò che si ha intenzione di fare: se per esempio bisogna spedire un messaggio (tipo un messaggio di posta elettronica) in modo segreto sulla rete, esso deve essere cifrato dal mittente, spedito e il destinatario deve avere la possibilità leggere il file in chiaro, decifrando il file cifrato; in questo caso appare evidente che verrà utilizzato una cifratura reversibile. Un altro esempio di cifratura reversibile è quello di criptare propri dati personali, tipo la carta di credito o documenti privati.

Contrariamente ai casi precedenti, si potrebbe voler cifrare dei dati escludendo la possibilità di decifrarli in qualsiasi modo. Tale meccanismo consente di proteggere i dati in maniera assolutamente sicura da eventuali attacchi. Evidentemente questi dati sono informazioni di solito molto brevi che l'utente avrà con sé. Un esempio pratico di questo problema è la cifratura delle password dei sistemi operativi, in cui non occorre effettuare un'operazione di decifratura, bensì è sufficiente effettuare un confronto run-time tra la stringa cifrata, presente sul computer, e quella generata dall'apposito programma di cifratura.

### 1.4 L'algoritmo Perfetto

L'algoritmo perfetto non è altro che un cifrario di **Vernam**.

Il sistema **Vernam** è un caso particolare della cifratura "**One time-Pad**"<sup>1</sup>. La cifratura detta a chiave infinita o "one-time pad" consiste nel generare una chiave composta il più possibile da lettere, numeri e simboli disposti in modo casuale, e di una lunghezza almeno pari alla lunghezza del testo da criptare. Quindi per ogni carattere del testo in chiaro viene aggiunto (come somma di valori ASCII ad esempio) un carattere della chiave, ottenendo il testo criptato. È facile intuire che un sistema del genere è praticamente impossibile da decifrare, in quanto si basa su una chiave ottenuta in modo assolutamente casuale (entropia massima) e senza la possibilità di analizzare delle ripetizioni nel testo, in quanto la chiave ha appunto la lunghezza stessa del testo. Questo meccanismo 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

---

<sup>1</sup> La chiave è utilizzata una sola volta.

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 Vernam può comportare degli svantaggi dovuti all'espansione della chiave, se questa ha lunghezza molto inferiore a quella del messaggio da cifrare. Risulta dunque conveniente usare chiavi molto lunghe e, soprattutto, più casuali possibili. Si tratta di un algoritmo efficientissimo, ma, essendo a chiave privata, comporta non pochi limiti.

### **1.5 Sviluppo di A.R.C.C: Idea, Scopi ed Utilizzi**

L'algoritmo di cifratura **A.R.C.C.**<sup>2</sup>, nasce per puri scopi di ricerca e muove i suoi primi passi nell'estate del 2002.

La prima versione era costituita esclusivamente dal livello Polialfabetico, ma molto più semplificato della versione odierna; essa eseguiva in effetti una semplice sostituzione dei caratteri dell'alfabeto con delle stringhe di lunghezza predeterminata.

Nel corso dei mesi sono state apportate moltissime modifiche ed aggiunte, cercando di rendere il progetto quanto più originale e robusto possibile.

Un solo livello di cifratura era inadeguato, è per questa ragione che è stato realizzato lo strato XOR. La chiave dello XOR iniziale era troppo breve; una situazione troppo lontana dall'algoritmo perfetto di Vernam, ecco che è stato adottato un meccanismo di "accrescimento" della chiave che attualmente simula, con buona approssimazione, una chiave infinita o, il che è lo stesso, di lunghezza pari al messaggio in chiaro.

L'obiettivo principale è tutt'ora quello di codificare un cifrario fortissimo, quasi indecifrabile, sacrificando per certi versi la velocità di esecuzione dell'implementazione pratica.

Non sono mancate le delusioni, gli affanni e i ripensamenti, ma "chi la dura la vince" e dopo un anno di intenso lavoro l'algoritmo è stato completato.

Gli ambiti in cui **ARCC** può essere impiegato sono fra i più disparati.

**ARCC** può essere infatti interpretato per certi versi come un cifrario a sostituzione e per altri come un cifrario a flusso, dunque può essere utilizzato per cifrare dati "statici", ossia presenti su memorie di massa, ma anche "dinamici" nelle trasmissioni a distanza, considerando esclusivamente il livello XOR.

---

<sup>2</sup> ARCC è l'acronimo di Angelo Rosiello, Roberto Carozzo, Cryptography

Si potrebbe utilizzare **ARCC** per cifrare dati privati come il numero della propria carta di credito, documenti importanti o perfino file; allo stesso modo è possibile criptare connessioni riservate “peer-to-peer” su internet o su altre reti di trasmissione.

Sicuramente **ARCC**, essendo molto giovane, non può essere considerato fra i migliori cifrari, ma siamo certi che nel corso del tempo avrà modo di dimostrare tutta la sua “grinta” e il suo carattere.

## **2 Accenni Storici sulla Crittografia**

### ***2.1 Definizioni utili***

#### **2.1.1 Cosa è la Crittografia**

La crittografia<sup>3</sup> è una vera e propria scienza matematica, il cui oggetto consiste nella realizzazione di una metodologia in grado di cifrare dati, al fine di renderli incomprensibili ad occhi "indiscreti". In generale i due processi principali che vengono applicati in crittografia si dividono in "cifratura" e "codifica"<sup>4</sup>. La cifratura lavora sulle lettere "individuali" di un alfabeto, mentre una codifica lavora ad un livello semantico più elevato, come può essere una parola o una frase. I sistemi di cifratura possono lavorare per trasposizione (mescolando i caratteri di un messaggio in un nuovo ordine), o per sostituzione (scambiando un carattere con un altro carattere in accordo con una regola specifica), o una combinazione di entrambi. Nel linguaggio corrente la trasposizione è anche chiamata permutazione. Una cifratura che implementa entrambe le tecniche (trasposizione e sostituzione) è chiamata "cifratura composta". In linea di massima una cifratura composta è più sicura di una cifratura basata solo su sostituzione o su trasposizione. Claude Shannon<sup>5</sup>, il padre della teoria dell'informazione, paragona la sostituzione alla "confusione", perchè l'output è una funzione non lineare dell'input. Egli inoltre ha paragonato la trasposizione alla "diffusione" perchè estende la dipendenza dell'output da un piccolo numero di posizioni dell'input a un grande numero.

Ogni sistema di crittografia ha due parti essenziali: un algoritmo (per codificare e decodificare) e una "chiave", la quale consiste di informazioni che, combinate con il testo "in chiaro", passato attraverso l'algoritmo, vi darà poi il testo codificato. In ogni moderno sistema di crittografia si assume che l'algoritmo sia conosciuto dai potenziali "attaccanti", quindi la sicurezza di un sistema risiede solo ed esclusivamente nella segretezza della chiave.

Il nostro obiettivo è quello di tradurre il linguaggio del testo in chiaro in un nuovo "linguaggio" che non può essere compreso e/o tradotto senza le informazioni supplementari fornite dalla chiave. A chi è familiare il concetto di entropia in fisica potrà essere sorpreso di scoprire che questo concetto è utile anche in crittografia. L'entropia è la misura della quantità di disordine in un sistema fisico, o della relativa assenza di informazione in un sistema di comunicazione. In un linguaggio naturale, come potrebbe essere l'italiano, l'entropia è bassa, in quanto vi è ridondanza e una certa regolarità

---

<sup>3</sup> Dizionario Zanichelli: "Sistema segreto in cifra o codice"; termine che deriva dal greco *Kruptòs*, che significa nascosto e *graphia*, che significa scrittura.

<sup>4</sup> L.SACCO, Manuale di crittografia, Roma, 1947, p.3.

<sup>5</sup> Nasce a Petoskey, Michigan (USA), si laurea (bachelor's degree) in Matematica e Ingegneria elettrica alla University of Michigan. Ottiene il master's degree al MIT con una tesi sull'algebra di Boole; lavora con il fisico Vannevar Bush, ricordato oggi come l'inventore dell'ipertesto, alle prime macchine calcolatrici; entra alla Bell Telephones come ricercatore di Matematica. Pubblica "A Mathematical Theory of Communication", un testo fondamentale per la teoria dell'Informazione. Pubblica "Communication theory of secret systems" nel quale definisce i fondamenti teorici della crittologia e le caratteristiche del cifrario perfetto consistenti nel fatto che la chiave abbia lunghezza infinita o che abbia la stessa lunghezza del testo in chiaro.

statistica nei termini e nei caratteri. Se alcuni dei caratteri di una frase sono persi o incomprensibili noi di solito riusciamo a ricostruire la frase, con una buona approssimazione, nella grande maggioranza dei casi. Inversamente, noi vogliamo che il nostro testo criptato abbia la più alta entropia possibile, al fine di renderne il più difficile possibile la decodifica. Idealmente il nostro testo dovrebbe apparire come una serie di lettere o simboli casuali (random). Il nostro principio guida sarà quello di aumentare l'incertezza del criptoanalista il più possibile.

La prima considerazione nella sicurezza di un sistema di crittografia riguarda la lunghezza della chiave. Se si assume una chiave troppo corta (se comparata alla lunghezza del testo in chiaro) molto probabilmente l'algoritmo usato, arrivato ad un certo punto della codifica, dovrà ripetere dei caratteri, o delle sequenze di caratteri, fornendo così uno schema che il criptoanalista potrebbe sfruttare per compiere il suo lavoro. Se la cosa dovesse ripetersi molte volte il criptoanalista potrebbe avere abbastanza materiale in mano per poter ottenere la nostra chiave. Un altro fattore importante da tenere in considerazione è il numero di chiavi che l'algoritmo ammette. Se lo algoritmo ammettesse, ad esempio, 10000 chiavi, un "nemico" dotato di mezzi di calcolo anche modesti potrebbe provarle tutte in un lasso di tempo accettabile, vanificando quindi il sistema di crittografia.

Questo approccio (ricerca esaustiva su tutte le chiavi possibili) viene chiamato approccio di "forza bruta". Questo introduce il concetto del "fattore lavoro", necessario per rompere un sistema di crittografia. In linea di principio un sistema sicuro in assoluto **NON** esiste. Ma se il fattore lavoro necessario per rompere l'algoritmo è molto alto, l'attaccante prima di lanciarsi nel tentativo di decodifica eseguirà un bilancio costi-benefici. Se il beneficio che esso potrà avere dalla decodifica del messaggio è inferiore allo sforzo (anche economico) che deve sostenere per decodificarlo (o per tentare di farlo) molto probabilmente lascerà perdere.

### **2.1.2 Cosa è la Crittanalisi**

La crittanalisi è l'arte di forzare un testo cifrato, ed è quindi uno dei rami più importanti della crittologia. Nel progettare un metodo crittografico si deve ovviamente tenere conto delle possibilità di crittanalisi del medesimo. I tentativi di forzare un testo cifrato da parte del crittanalista sono chiamati "*attacchi*".

I metodi crittanalitici classici sono di tipo statistico e si basano sul fatto che tutte le lingue hanno una distribuzione delle frequenze delle lettere, dei bigrammi e dei trigrammi che è facilmente riconoscibile.

Esistono molti parametri statistici<sup>6</sup> in grado di aiutare il crittanalista già nella prima fase, che è quella dell'individuazione del metodo di cifratura usato.

Per il successo della crittanalisi sono utili anche molte altre cose: la conoscenza della lingua del crittogramma è ovviamente fondamentale, avere indicazioni certe su qualche parola che deve comparire sicuramente nel testo è importante e dal momento che "errare humanum est" spesso sono decisivi gli errori o le ingenuità dei cifratori. Infatti è usuale che gli addetti alle operazioni di cifratura dei messaggi, siano spesso militari o impiegati con limitate conoscenze crittografiche.

Con l'avvento del computer anche lo scenario della crittanalisi è molto cambiato e sono stati provati nuovi tipi di attacchi.

In generale si possono individuare diverse famiglie di attacchi:

- Attacco esaustivo: consiste nel provare tutte le possibili chiavi, fino a trovare quella corretta; la complessità è enorme e per i metodi più sicuri come Triple-DES o RSA, un attacco del genere potrebbe richiedere molti anni di elaborazione anche per computer con grande potenza di calcolo.
- Attacco basato su uno o più testi cifrati: il crittanalista dispone di uno o più testi cifrati intercettati in qualche modo e sulla base di questi cerca di ricostruire il testo chiaro; è il caso della *crittanalisi statistica*.
- Attacco basato su testi chiari e cifrati; il crittanalista dispone per esempio del programma cifrante, e può provare a cifrare più testi in chiaro con la stessa chiave ed esaminare i testi cifrati ottenuti. È il caso della *crittanalisi differenziale* e la più recente *crittanalisi lineare*.

Risulta indispensabile approfondire le metodologie di crittanalisi suddette.

### **Crittanalisi statistica**

Il cifrario per sostituzione monoalfabetica è forse il più classico dei cifrari; la sua sicurezza è però accettabile solo per messaggi molto brevi ed alla irrealistica condizione di cambiare chiave quasi ad ogni messaggio.

La crittanalisi di questi sistemi si fonda infatti, su metodi statistici tanto più efficienti quanto più lunghi e numerosi sono i testi cifrati che si hanno a disposizione.

---

<sup>6</sup> Il primo problema per il crittanalista che si trova davanti a un testo cifrato, è di stabilire quale metodo di cifratura è stato usato, p.es. se si tratti di un cifrario monoalfabetico o polialfabetico, o di una trasposizione.

Sono stati definiti alcuni parametri statistici che sono di valido aiuto a questo scopo:

- Numero di presenze: è il numero  $p$  di lettere diverse che compaiono nel testo cifrato; ovviamente è sempre minore o uguale al numero  $A$  di lettere dell'alfabeto usato (21 per l'italiano, 26 per l'internazionale ...).
- Media frequenza quadratica: altro parametro statistico che misura l'uniformità della distribuzione delle varie lettere.

Il fatto è che ogni lingua ha una distribuzione delle frequenze dei caratteri molto peculiare. In italiano per esempio le lettere più frequenti sono le vocali E, A, I; in francese la E è di gran lunga la lettera più frequente.

Ancor più evidente è la frequenza dei bigrammi e dei trigrammi. In molte lingue la lettera Q è sempre seguita dalla U, in italiano la lettera H è quasi sempre preceduta dalla C o dalla G.

Per forzare un testo cifrato monoalfabetico, si esegue una statistica della frequenza delle lettere presenti e si costruisce un grafico disponendo i caratteri in ordine decrescente. È molto utile rilevare anche i bigrammi e trigrammi più frequenti.

I caratteri più frequenti nel crittogramma, saranno le lettere più frequenti nella lingua; e così i bigrammi e i trigrammi. In genere sono sufficienti discreti tentativi per riconoscere qualche parola e a questo punto il lavoro è di gran lunga meno complesso.

### **Crittanalisi differenziale**

La sicurezza di un messaggio cifrato non può mai basarsi sulla presunzione che il nemico non conosca il metodo di cifratura; in altre parole bisogna partire dal presupposto che il nemico conosca bene tale meccanismo. Oggi poi molti sistemi di cifratura per computer sono reperibili in formato elettronico e chiunque può procurarseli e testarli.

Questo fatto offre un'opportunità in più al crittanalista ossia quella di provare a cifrare un certo numero di messaggi in chiaro, di esaminare i testi cifrati ottenuti con una medesima chiave e dall'esame delle differenze, cercare di calcolare la probabilità di ogni possibile chiave, fino ad individuare le chiavi più probabili e presumibilmente la vera chiave.

È questa l'idea che sta alla base della crittanalisi differenziale, ideata negli anni '70 da Murphy e poi da Shamir e Biham, che la usarono nel tentativo di forzare il DES.

I moderni metodi di cifratura devono tenere in considerazione non solo la tradizionale crittanalisi statistica, ma anche quella differenziale e lineare.

### **Crittanalisi lineare**

La crittanalisi lineare è uno dei frutti delle innovazioni crittanalitiche del XX secolo. Si basa sulla possibilità di disporre di una coppia, messaggio chiaro  $\leftrightarrow$  messaggio cifrato, cosa non impossibile nel campo delle telecomunicazioni.

Partendo da un testo in chiaro, questo sistema usa un'approssimazione lineare per descrivere il comportamento dell'algoritmo di cifratura.

Fondamentalmente viene confrontato il testo in chiaro e il corrispondente testo cifrato. In base ai dati così raccolti ed attentamente esaminati, ci si può fare un'idea della chiave. La probabilità di successo è abbastanza alta.

## 2.2 Il codice di Atbash

Il libro di Geremia nella Bibbia usa un semplicissimo codice monoalfabetico<sup>7</sup> per cifrare la parola Babele; la prima lettera dell'alfabeto ebraico (Aleph) viene cifrata con l'ultima (Taw), la seconda (Beth) viene cifrata con la penultima (Shin) e così via; da queste quattro lettere è derivato il nome di Atbash (A con T, B con SH) per questo codice.

Usando per comodità l'alfabeto inglese, l'*Atbash* può essere riassunto mediante la seguente tabella di cifratura:

CHIARO	a b c d e f g h i j k l m
CIFRATO	Z Y X W V U T S R Q P O N
CHIARO	n o p q r s t u v w x y z
CIFRATO	M L K J I H G F E D C B A

Utilizzando la parola in chiaro “atbash”, il risultato cifrato sarà “ZGYZHS”.

## 2.3 La scacchiera di Polibio

Lo storico greco Polibio (~200-118AC), descrive nelle sue Storie (Libro X) il più antico esempio di codice poligrafico<sup>8</sup>, che attribuisce ai suoi contemporanei Cleoxeno e Democleito.

L'idea è quella di cifrare una lettera con una coppia di numeri compresi tra 1 e 5, in base ad una scacchiera 5x5, convertendo quindi lettere con numeri.

In tal modo il messaggio può essere trasmesso con due gruppi di cinque torce (e.g. 1,5 = una torcia accesa a destra, cinque a sinistra). In effetti più che di un codice segreto, si tratta di un sistema di telecomunicazione, di fatto un telegrafo ottico. Telegrafi a torce esistevano da molti secoli ed erano stati destritti da Enea il tattico intorno al 350AC, ma erano basati su un limitato elenco di messaggi possibili; quello di Polibio si basa invece sulla scomposizione del messaggio nelle singole lettere ed è quindi in grado di trasmettere qualsiasi messaggio.

---

<sup>7</sup> I cifrari monoalfabetici sono cifrari di sostituzione, cioè si sostituisce ogni carattere del testo con un altro carattere (o numero) secondo una tabella prestabilita, ottenendo il testo cifrato.

<sup>8</sup> I cifrari poligrafici sono così denominati poichè ogni lettera del testo viene dapprima scomposta in gruppi di due o più lettere o cifre che vengono poi a loro volta cifrate mediante meccanismi più o meno complessi.

Per comodità negli esempi seguenti si utilizzerà, al posto di quello greco, l'alfabeto inglese il quale però ha il difetto di essere formato da 26 caratteri, così per poter costruire il quadrato necessario per la cifratura bisognerà, come in questo caso per la k e la q, "fondere" due lettere rare ma non foneticamente differenti nella stessa casella. In questo modo si otterrà la seguente tabella:

#	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I	J
3	KQ	L	M	N	O
4	P	R	S	T	U
5	V	W	X	Y	Z

**Tabella 1:** Scacchiera di Polibio costituita da lettere dell'alfabeto inglese

Ogni lettera è rappresentata da due numeri, guardando la riga e la colonna in cui la lettera si trova, allo stile della battaglia navale.

Per esempio, A=11 e U=45.

Se dovessimo cifrare "Il nemico ci attacca" avremmo il seguente risultato:

2432341533241335132411444411131311

## 2.4 Cifrario di Cesare

Mediante gli scritti di Svetonio siamo in grado di affermare che Giulio Cesare era solito utilizzare un codice di sostituzione molto semplice per le sue corrispondenze riservate, nel quale la lettera in chiaro veniva sostituita dalla lettera collocata tre posizioni in avanti nell'alfabeto.

La lettera A è sostituita dalla D, la B dalla E e così via fino alle ultime lettere che sono cifrate con le prime (come le odierne funzioni modulari).

La tabella che segue esemplifica il cifrario di Cesare utilizzando l'alfabeto inglese.

Chiaro	a b c d e f g h i j k l m n o p q r s t u v w x y z
Cifrato	D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

Prendendo come esempio la frase "Attacchiamo i Galli" si otterrà il seguente messaggio cifrato:

Chiaro	Attacchiamo i Galli
Cifrato	Dwwdffkldpr l Jdool

Più in generale si dice codice di Cesare un codice nel quale la lettera del messaggio in chiaro viene spostata di un numero fisso di posizioni ma non necessariamente tre; un esempio è il codice che sempre secondo Svetonio era usato da Augusto, dove la A era sostituita dalla B, la B dalla C e così via.

Poiché l'alfabeto inglese è costituito da 26 caratteri sono possibili 26 codici di Cesare diversi dei quali solo uno darà un testo cifrato uguale al messaggio in chiaro iniziale.

## **2.5 Codici Medioevali**

In questo periodo la crittografia viene usata solo per celare i nomi propri, sostituendo una lettera con quella successiva dell'alfabeto regolare (A con B, B con C ecc.), ma limitando tale sistema alle vocali, cifrate a volte con gruppi di punti secondo il sistema di Enea il tattico<sup>9</sup>.

Verso l'anno mille compaiono i primi alfabeti cifranti<sup>10</sup> o monografici. Essi sono usati successivamente soprattutto nelle missioni diplomatiche tra i vari staterelli europei, particolarmente da parte delle repubbliche marinare e dalla corte papale di Roma e a partire dal XIV° secolo.

Si usano le cosiddette nomenclature, ossia liste di parole chiave del gergo diplomatico abbreviate con un solo segno; ne troviamo molti esempi tra i secoli XIV° e XVIII°. Un altro sistema è quello usato dall'Arcivescovo di Napoli, Pietro di Grazia, tra il 1363 e il 1365 in cui le vocali sono sostituite da semplici segni e le vocali scritte in chiaro funzionano da nulle<sup>11</sup>; nelle ultime lettere il procedimento è applicato anche alle consonanti più frequenti (l,r,s,m,n), che a volte erano cifrate anche con altre lettere alfabetiche.

Nel 1378, dopo lo scisma di Avignone, l'antipapa Clemente VII° decise di unificare i sistemi di cifratura dell'Italia Settentrionale ed affidò tale compito a **Gabriele Lavinde**, in Vaticano è conservato un suo manuale del 1379. In esso ogni lettera è cifrata con un segno di fantasia, in alcuni casi vi sono delle nulle, in altri vi sono delle nomenclature; le vocali sono trattate come le altre lettere, come in una cifra del 1395 di Mantova.

---

<sup>9</sup> Tra il 360 e il 390 venne compilato da Enea il tattico, generale della lega arcadica, il primo trattato di cifre il cui XXI capitolo tratta appunto di messaggi segreti.

In questo viene descritto un disco sulla zona esterna del quale erano contenuti 24 fori, ciascuno corrispondente ad una lettera dell'alfabeto. Un filo, partendo da un foro centrale, si avvolgeva passando per i fori delle successive lettere del testo: all'arrivo, riportate le lettere sul disco, si svolgeva il filo segnando le lettere da esso indicate: il testo si doveva poi leggere a rovescio. Le vocali spesso erano sostituite da gruppi di puntini.

<sup>10</sup> La cifratura tramite alfabeto monografico viene effettuata, nel caso più semplice, dando ad ogni lettera dell'alfabeto, compresi a volte lo spazio e i vari segni di interpunzione, come corrispondente un segno dello stesso alfabeto, di un altro alfabeto o addirittura inventato dall'ideatore della cifra al momento.

Si ottiene quindi una tabella a due colonne dove ogni segno alfabetico del testo in chiaro corrisponde biunivocamente ad uno dell'alfabeto cifrante.

<sup>11</sup> Termine usato nel campo della crittografia per denominare un segno o un gruppo cifrante che non ha valore di lettera alfabetica né di segno d'interpunzione, ma che si usa solo per alterare le frequenze relative delle lettere del testo.

Dagli inizi del XIV° secolo, per depistare i tentativi di analisi statistica delle frequenze, si iniziano ad usare più segni per cifrare una stessa vocale come possiamo leggere in una cifra con più di tre segni diversi per ogni vocale, ma senza nulle e senza omofoni<sup>12</sup> conservata sempre a Mantova del 1401.

In seguito viene ampliato il nomenclatore e, a parte la diversità dei segni cifranti, tutte le cifre italiane dei tre secoli successivi seguirono questo modello. Ne abbiamo esempi anche alla corte Francese del XVII° secolo e perfino da parte dei nobili francesi in esilio nel 1793. Tale sistema fu in uso anche nella telegrafia segreta attorno alla seconda metà del 1800.

Eccezioni a questo canone si debbono al Cardinale Richelieu attorno al 1640 per consiglio di **Antonio Rossignol**; si tratta di repertori invertiti con gruppi cifranti variabili, con due documenti per cifrare e decifrare con omofoni per le singole lettere. Possiamo trovarne altri esempi nelle corrispondenze tra Luigi XIV° e il suo maresciallo alla fine del '600 . La loro corrispondenza, con 11.125 gruppi cifranti diversi, veniva considerata "sicura", ed infatti fu sempre cifrata con lo stesso repertorio, mentre era già stata violata nel 1689 da Wallis. Dopo Luigi XIV° la crittografia francese declinò tanto che sotto Napoleone si usava un repertorio di soli 200 gruppi quasi privo di omofoni ed applicato solo a parti dei dispacci. Sembra che anche questa inferiorità nella cifratura contribuì al disastro russo del 1812-13 .

Altre cifre papali del XVI° secolo utilizzano un sistema assai diverso, ossia la cifratura con polifoni. La prima di queste cifre appare attorno al 1540; l'ultima nel 1585. Il nomenclatore di tali cifre è costituito da circa 300 voci, tutte cifrate con gruppi di tre cifre.

Un altro esempio di polifonia si trova nel sistema usato dal langravio d'Assia nei primissimi anni del '600, nella quale spesso un gruppo di due numeri indica o una lettera ed una parola vuota oppure una sillaba. Tuttavia è probabilmente a distinguere le funzioni del gruppo era la collocazione di segni ausiliari, che poi il tempo ha cancellato.

Secondo il Meister, uno studioso di crittografia, il sistema polifonico era usato spesso per ridurre la lunghezza del testo cifrato. Egli riporta anche istruzioni per la composizione di simili cifre che sono all'avanguardia per i suoi tempi.

## **2.6 Il Disco di Leon Battista Alberti**

La prima macchina per cifrare è il **disco cifrante**, inventato nel XV secolo dall'architetto italiano **Leon Battista Alberti**. Egli partì da due dischi di rame, uno di diametro leggermente maggiore rispetto all'altro. Lungo la circonferenza di ciascun disco, è riportato un alfabeto. Collocando il disco più grande sul più piccolo, e infilando entrambi su di un perno, egli realizzò un congegno in

---

<sup>12</sup> Segni corrispondenti nell'alfabeto cifrante aventi tutti lo stesso significato chiaro.

grado di cifrare. I dischi sono liberi di ruotare l'uno rispetto all'altro, cosicché i due alfabeti vengono ad assumere differenti posizioni relative. Quindi, il disco cifrante può essere usato per crittografare un messaggio col sistema di Cesare. Per esempio, per effettuare una cifratura di Cesare con spostamento pari a uno basta collocare la A interna in corrispondenza della B esterna; il disco interno rappresentava l'alfabeto normale, quello esterno l'alfabeto cifrante. Poi, senza spostare i dischi si cercano le lettere del testo chiaro sul disco interno, una per volta; in ciascun caso, la lettera corrispondente situata sul disco esterno è quella da inserire nel crittogramma.

Pur essendo un dispositivo molto semplice, il disco cifrante facilita in modo significativo la produzione di scritture segrete, ed è rimasto in uso per cinque secoli.

Il funzionamento descritto fin qui è immediato e il crittogramma che ne deriva può essere decifrato con facilità, ma il disco può essere utilizzato in modo più sofisticato. L'Alberti, che lo inventò, suggeriva di cambiare l'assetto del disco durante la codifica del messaggio, realizzando in sostanza una cifratura polialfabetica, anziché monoalfabetica.

L'aspetto importante di questo modo di procedere è che lo strumento che sostituisce gli elementi chiari con quelli in cifra cambia modo di funzionare durante la sostituzione.

Tuttavia, pur essendosi imbattuto nella più importante scoperta dell'ultimo millennio nel campo delle scritture segrete, egli non riuscì a trasformare la sua idea, appena abbozzata, in una tecnica ben definita.

## **2.7 Giovan Battista della Porta**

**G.B. Porta** (o Della Porta), pubblicò nel 1563 a Napoli un trattato di crittografia (*De Furtivis literarum notis - vulgo de ziferis*) molto vasto e di ottimo livello.

Tra le cifre proposte dal Porta è nota soprattutto la “*tavola*”, che non è certo la migliore tra quelle presenti nel trattato e che è peraltro più debole di quella dell'Alberti.

La tavola del Porta usa 11 alfabeti e introduce il cosiddetto verme<sup>13</sup> letterale, poi adottato in varie cifre, e che ha il grave inconveniente di produrre un periodo di cifratura relativamente corto, perché comprendente tante lettere quante ne ha il verme nel quale le liste cifranti si susseguono tutte nello stesso ordine: particolarità su cui si basa la decrittazione del sistema, facilitata in questo caso dalla conoscenza degli alfabeti usati.

In realtà il Porta consiglia di usare 11 alfabeti involuttori arbitrari, ma dà come esempio la tavola con l'alfabeto base regolare: sotto questa sola forma la sua cifra è stata poi da tutti divulgata. Seguendo le indicazioni del Porta si scriverà la parola, o verme, lettera per lettera sotto ciascuna

---

<sup>13</sup> Svolge una funzione simile a quella della chiave, ma dovrebbe avere una lunghezza molto maggiore aumentando così la sicurezza dell'algoritmo.

lettera del testo chiaro, ripetendola quante volte occorre: la cifratura si farà usando per ciascuna lettera del testo chiaro la lista individuata dalla corrispondente lettera chiave.

## 2.8 Giovanbattista Bellaso

G.B.Bellaso pubblicò nel 1553 un opuscolo, "Il vero modo di scrivere in cifra" contenente alcuni suoi cifrari polialfabetici. L'idea è quella di ricavare diversi alfabeti (tutti disordinati) da una parola convenuta, versetto o motto.

Un esempio dell'autore: data la parola chiave sia IOVE, il primo alfabeto derivato (con V=U) è:

I O A B C D F G H L  
V E M N P Q R S T X

Il secondo si ottiene spostando circolarmente la seconda riga:

I O A B C D F G H L  
X V E M N P Q R S T

e così via fino ad ottenere cinque alfabeti; ognuno di questi sarà identificato da un gruppo di quattro lettere. Per esempio:

I D V Q | I O A B C D F G H L  
          | V E M N P Q R S T X

O F E R | I O A B C D F G H L  
          | X V E M N P Q R S T

A G M S | I O A B C D F G H L  
          | T X V E M N P Q R S

B H N T | I O A B C D F G H L  
          | S T X V E M N P Q R

C L P X | I O A B C D F G H L  
          | R S T X V E M N P Q

A questo punto si deve convenire un altro motto, per esempio OPTARE MELIORA; le lettere di quest'ultimo servono a selezionare l'alfabeto da usare.

Volendo allora cifrare la frase "Inviare truppe domani" si ottiene:

Verme            O                    P                    T

Chiaro  I N V I A R E    T R U P P E    D O M A N I

Cifrato  X C O X E G A    A I C H H D    M T D X F S

Le cifre del Bellaso sono più deboli di quella dell'Alberti perchè usano pochi alfabeti ed il cambio di lista non è segreto.

## **2.9 Il Cifrario di Vigénère**

Blaise de Vigénère pubblicò nel 1586 un trattato di cifre nel quale proponeva tra gli altri un codice che ebbe grande fortuna e che è ricordato con il suo nome. Si tratta del più semplice codice di sostituzione polialfabetica e proprio per la sua semplicità ha goduto per secoli di una grande fama, sebbene fosse molto più debole di altri codici polialfabetici precedenti come il disco dell'Alberti, o le cifre del Bellaso. Tale fortuna è durata fino a molti decenni dopo che era stato pubblicato anche il primo metodo di decrittazione del Kasiski, descritto di seguito. In un crittogramma<sup>14</sup> alla Vigénère si trovano spesso sequenze identiche di caratteri a una certa distanza l'una dell'altra, questo avviene nel momento in cui si cripta la medesima lettera che si trova in corrispondenza della stessa lettera del verme; se per esempio usando la chiave VERME come sopra si scrive due volte la preposizione DEL a 30 caratteri di distanza questa sarà cifrata in modo identico essendo 30 un multiplo della lunghezza del verme che è 5.

Dal cifrario di Vigenere deriva il cifrario di Vernam, considerato il cifrario teoricamente perfetto.

Il metodo si può considerare una generalizzazione del codice di Cesare; invece di spostare sempre dello stesso numero di posti la lettera da cifrare, questa viene spostata di un numero di posti variabile, determinato in base ad una parola chiave, da concordarsi tra mittente e destinatario, e da scriversi sotto il messaggio, carattere per carattere. La parola è detta verme, per il motivo che, essendo in genere molto più corta del messaggio, deve essere ripetuta molte volte sotto questo, come nel seguente esempio:

Testo chiaro    -  ARRIVANOIRINFORZI  
Verme            -  VERMEVERMEVERMEVE

---

<sup>14</sup> Messaggio cifrato.

Testo cifrato - VVIUZVRFUVDRWAVUM

Il testo cifrato si ottiene spostando la lettera chiara di un numero fisso di caratteri, pari al numero ordinale della lettera corrispondente del verme. Di fatto si esegue una somma aritmetica tra l'ordinale del chiaro (A = 0, B = 1, C = 2 ...) e quello del verme; se si supera l'ultima lettera, la Z, si ricomincia dalla A, secondo la logica dell'aritmetica modulare.

Per semplificare questa operazione il Vigénère propose l'uso della seguente tavola quadrata, composta da alfabeti ordinati spostati. Volendo ad esempio cifrare la prima **R** di ARRIVANO si individuerà la colonna della R, quindi si scenderà lungo la colonna fino alla riga corrispondente della corrispondente lettera del verme (qui E); la lettera trovata all'incrocio è la lettera cifrata (qui V); la seconda **R** invece sarà cifrata con la lettera trovata sulla riga della R di VERME, e cioè con la I

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	S
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U

W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

**Figura 1:** Tavola quadrata di Vigenère

Il vantaggio rispetto ai codici mono alfabetici è evidente: la stessa lettera del testo chiaro non è sempre cifrata con la stessa lettera e questo rende più difficile l'analisi statistica del testo cifrato e la decrittazione.

Chi riceve il messaggio per decifrarlo deve semplicemente usare il metodo inverso (sottrarre invece che sommare), riferendosi all'esempio di sopra si avrà:

Testo cifrato - VVIUZVRFUVDRAWVUM  
 Verme - VERMEVERMEVERMEVE  
 Testo chiaro - ARRIVANOIRINFORZI

si potrà decifrare la seconda **V** ricercandola nella riga della corrispondente lettera del verme, la **E**; la colonna dove si trova la **V** ha al primo posto in alto la lettera chiara, la **R**.

## 2.10 Codice di Jefferson

Il codice di Jefferson prende il nome dal suo inventore Thomas Jefferson (1743-1826), autore della Dichiarazione d'Indipendenza e presidente degli USA nel mandato del 1801.

Il codice è di facile utilizzo e tuttora rimane abbastanza sicuro, secondo gli standard di oggi. Non venne però adottato dagli Stati Uniti anche se in teoria, avrebbe potuto sopportare qualsiasi attacco crittoanalitico contemporaneo.

Jefferson lo archiviò e il suo codice rimase nel "dimenticatoio" fino al 1922, quando fu riscoperto e utilizzato fino agli anni '50 dall'esercito statunitense. Fino a questo momento il codice di Jefferson era stato talmente ignorato che nel 1890 Etienne Bazeries, lo "*Indecifrabile*", reinventò indipendentemente lo stesso metodo di cifratura.

Il codice di Jefferson era un metodo di cifratura meccanico e cioè basato su di una macchina; questa macchina consiste in un cilindro di circa 15 cm di lunghezza e 4 cm di larghezza montato su un asse e sezionato in 36 dischi uguali (25 nella versione poi utilizzata dagli Americani). Sul bordo di ciascuna ruota sono scritte le 26 lettere dell'alfabeto, equidistanti l'una dall'altra. L'ordine in cui sono disposte le varie lettere non corrisponde a quello naturale e varia da ruota a ruota.

Il messaggio in chiaro deve essere cifrato a blocchi di 36 lettere ciascuno (qualora l'ultimo blocco presenti meno di 36 lettere, esso deve essere completato con lettere nulle); la chiave di cifra è un

numero che va da 1 a 25. Supponendo che il primo blocco in chiaro sia “*La missione in Polinesia è fallita*”, e la chiave sia il numero 6, in una certa riga, non importa quale, si comporrà il messaggio in chiaro (omettendo naturalmente gli spazi); il crittogramma corrispondente andrà letto sulla sesta riga dopo quella che contiene il blocco in chiaro. Questo metodo di cifratura verrà in parte riutilizzato dai Tedeschi nella Seconda Guerra Mondiale nella cosiddetta Macchina Enigma. Come quasi tutti i metodi di cifratura anche il cilindro di Jefferson ha un grave difetto che ricorda quello che “minava” il codice di Cesare: poichè le chiavi sono solo venticinque se il cilindro cade nelle mani del nemico il crittogramma può essere facilmente risolto.

### 2.11 Il Cifrario di Playfair

Il cifrario di Playfair prende il nome non dal suo inventore, il fisico Sir Charles Wheatstone, ma da colui che lo ha fatto conoscere cioè Lyon Playfair Barone di St. Andrews.

Lyon Playfair, mostrò per la prima volta questo sistema nel 1854 durante una cena organizzata da Lord Granville alla presenza di Lord Palmerston (1784-1865) allora ministro degli Esteri.

La speranza di Playfair era quella di far utilizzare il cifrario durante la guerra di Crimea, ma esso fu effettivamente utilizzato dall'esercito britannico solamente a partire dalla guerra Boera.

Questo sistema deriva direttamente dalla Scacchiera di Polibio, ma ne differisce nel metodo di cifratura che questa volta non avviene lettera per lettera ma a coppie di due lettere, cioè a diagrammi, e in modo diverso a seconda di come sono distribuite le due lettere da cifrare nella scacchiera.

La scacchiera viene costruita in modo del tutto analogo a quella di Polibio, si usa una matrice di 25 lettere che viene riempita nelle prime caselle con la parola chiave, abolendo le eventuali lettere ripetute, ed è completata con le rimanenti lettere nel loro ordine alfabetico. Si omette la W che, se necessario, potrà essere cifrata come una doppia V. Così, con la chiave “*computer*”, si otterrà la seguente tabella:

#	1	2	3	4	5
1	C	O	M	P	U
2	T	E	R	A	B
3	D	F	G	H	I
4	J	KQ	L	N	S
5	V	W	X	Y	Z

**Tabella 2:** Esempio di scacchiera di Playfair

A questo punto il messaggio chiaro viene spezzato in coppie di lettere ognuna delle quali viene cifrata tenendo conto delle seguenti regole:

1. Se le due lettere della stessa coppia stanno sulla stessa riga vanno sostituite con quelle che le seguono sulla stessa riga (tenendo conto che la tabella è prolungata per circolarità, cioè se una delle lettere sta sulla quinta colonna verrà sostituita dalla lettera che sta sulla prima colonna).
2. Se le due lettere stanno sulla stessa colonna vanno sostituite con quelle che stanno sulla riga sottostante anche questa volta con il prolungamento per circolarità (se una lettera sta nell'ultima riga viene sostituita da quella nella prima riga).
3. Se le due lettere non stanno nè sulla stessa riga nè sulla stessa colonna, che è il caso più frequente, vanno comunque ad individuare una matrice come segue:

#	1	2	3	4	5
1	C	O	M	P	U
2	T	E	R	A	B
3	D	F	G	H	I
4	J	K	L	N	S
5	V	W	X	Y	Z

**Tabella 3:** Esempio di scacchiera di Playfair

Le due lettere verranno sostituite da quelle che contraddistinguono gli altri due vertici aventi in prima posizione quella che sta sulla riga della prima lettera chiara, per esempio partendo dalla coppia KA si ottiene NE, mentre partendo da AK si avrebbe EN.

4. Se le due lettere formanti la coppia sono uguali non è possibile cifrarle ma bisogna interromperle inserendo delle nulle fra di loro come X, K, W cioè lettere "rare" e poi proseguire con l'applicazione delle altre regole.

L'inserimento delle nulle fra le doppie è anche un' importante strategia per cercare di rendere più complessa possibile l'analisi delle frequenze, infatti le doppie sono costituite solo da alcune lettere.

Prendendo la frase "*Inviare subito nuove truppe*" si otterrà la seguente successione di bigrammi (si noti che il raddoppio della lettera -p- è stato spezzato inserendo fra le due lettere la -y-):

**INVIARESUBITONUOVETRUPPE**

Quindi, seguendo le regole succitate, il messaggio cifrato risultante sarà il seguente:

**HQZFABTKBIDBPKCMOFEACUAYE**

Il Playfair cypher è dunque un buon sistema di cifratura la cui unica limitazione è che la tabella contiene nelle sue prime due righe le lettere più frequenti della lingua, mentre quelle rare si trovano normalmente nell'ultima linea.

D'altro canto non sarebbe consigliabile riempire la tabella in modo del tutto disordinato per la difficoltà di memorizzazione, infatti non è opportuno che un "corriere" che porta la chiave da mittente a destinatario scriva la tabella per non incorrere nel rischio di dimenticarla.

### **2.12 Il Cifrario bifido di Delastelle**

Il cifrario di Delastelle è un cifrario poligrafico che si basa su una matrice 5x5, ed è stato inventato nel XIX secolo da uno dei più importanti crittologi francesi, nonostante fosse un dilettante, ossia Félix-Marie Delastelle.

Il metodo si articola in quattro passi:

- Il messaggio in chiaro viene spezzato in blocchi di cinque caratteri sotto ciascuno dei quali viene inserito il numero di colonna e ancora più sotto il numero di riga che la contraddistingue; se l'ultimo blocco non è esattamente di cinque caratteri, gli ultimi posti sono riempiti con X.
- Ogni lettera del blocco viene cifrata con due cifre e cioè con l'indice di riga e l'indice di colonna, che vengono scritte in verticale sotto la lettera chiara.
- Le cifre vengono ora riscritte in orizzontale riga dopo riga ottenendo un messaggio con un numero di cifre doppio dell'originale.
- A questo punto ogni coppia di numeri viene ritrasformata in lettera sempre secondo la matrice. Ne risulta il messaggio cifrato da trasmettere.

La matrice può essere quella semplice con le lettere dell'alfabeto ordinate (senza la W che può cifrarsi con una doppia V), oppure può essere ottenuta con una parola chiave come nel cifrario di Playfair.

Esempio di cifratura:

TESTO IN CHIARO: Un geniale crittografo

PAROLA CHIAVE: ARANCIA

Matrice:

#	1	2	3	4	5
1	A	R	N	C	I
2	B	D	E	F	G
3	H	J	KQ	L	M
4	O	P	S	T	U
5	V	W	X	Y	Z

Punto 1:

U	N	G	E	N	I	A	L	E	C	R	I	T	T	O	G	R	A	F	O
5	3	5	3	3	5	1	4	3	4	2	5	4	4	1	5	2	1	4	1
1	1	2	2	1	1	1	3	2	1	1	1	4	4	4	2	1	1	2	4

Punto 2:

53	53	31	12	21	51	43	41	13	21	25	44	11	14	44	52	14	12	11	24
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

Punto 3:

M	M	N	B	R	I	L	C	H	R	W	T	A	O	T	G	O	B	A	P
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Il Delastelle propose anche un cifrario trifido, che fa uso di una matrice tridimensionale 3x3x3, con 27 caratteri dunque inserito l'intero alfabeto di ventisei caratteri avanza ancora una cella che si può utilizzare con un carattere qualsiasi, per esempio lo spazio.

### 2.13 Il Cifrario di Vernam

L'idea proposta da G.S. **Vernam** nel 1926 per il cifrario che porta il suo nome è quella di generare una chiave del tutto casuale, e dunque imprevedibile, lunga come il testo; a questo punto il testo in chiaro e la chiave vengono "sommati" proprio come nel cifrario di Vigenere. L'unica differenza è che nel Vernam si sommano non tanto gli ordinali delle lettere da cifrare ma i singoli bit che codificano la lettera con l'operazione logica XOR. Questa è simile all'addizione, ma ha il vantaggio di essere reversibile, e quindi verrà usata anche per decifrare.

In tal modo la debolezza del Vigenere è superata e anzi **Claude Shannon**, il padre della Teoria dell'Informazione, ha dimostrato nel 1949 che ogni cifrario "teoricamente sicuro" è un cifrario di Vernam (e viceversa). Infatti se la chiave è totalmente casuale e lunga come il testo allora il testo cifrato non contiene alcuna informazione sul testo chiaro, ed è del tutto al sicuro dagli attacchi della crittanalisi statistica.

Per avere una sicurezza assoluta non si dovrebbe mai riutilizzare la stessa chiave; se si utilizza più volte la stessa chiave infatti questa torna ad essere più breve del messaggio, o meglio della somma di tutti i messaggi e il cifrario non è più perfetto. Per questo motivo questo tipo di cifrario viene detto a "chiave non riutilizzabile".

Un esempio di cifratura di Vernam è il seguente:

Chiario	A	T	T	E	N	Z	I	O	
C	11000	00001	00001	10000	00110	10001	01100	00011	
	N	E							
	00110	10000							

Verme V	W I A P F I L K 11001 01100 11000 01101 10110 01100 01001 11110 M S 00111 10100
Cifrato C XOR V	00001 01101 11001 1101 10000 11101 00101 11101 T P W Q E Q H Q 00001 00100 T {sp}

**Tabella 4:** Esempio di cifrario di Vernam

## 2.14 Enigma

Nel 1918 l'inventore tedesco Arthur Scherbius mise a punto un dispositivo crittografico che, in sostanza, era una versione elettromeccanica del disco cifrante di Alberti.

La sua invenzione fu chiamata **Enigma**.

La macchina Enigma consisteva di diversi ingegnosi elementi, da lui combinati in un potente e sofisticato dispositivo per la produzione di scritture segrete.

La versione base del congegno di Scherbius, consisteva in tre componenti collegati da fili elettrici: una tastiera per immettere le lettere del testo chiaro; un'unità scambiatrice, che cifra la lettera trasformandola nel corrispondente elemento del crittogramma; e un visore con varie lampadine, che, accendendosi, indicano la lettera da inserire nel crittogramma.

Per generare il crittogramma, l'operatore preme il tasto corrispondente alla lettera da crittare; l'impulso elettrico raggiunge l'unità scambiatrice, e dopo essere stato elaborato va ad illuminare il visore in modo corrispondente alla lettera crittata.

Lo **scambiatore**, uno spesso disco di gomma attraversato da una complessa rete di fili, è la parte più importante della macchina. I fili elettrici, provenienti dalla tastiera, entrano nello scambiatore in 26 punti, seguono un percorso caratterizzato da vari gomiti, e infine emergono dalla parte opposta in altri 26 punti. I circuiti interni dello scambiatore determinano il modo in cui un elemento del testo chiaro è crittato. Lo scambiatore, in sostanza, definisce un alfabeto cifrante, e la macchina può essere usata per realizzare una semplice cifratura per sostituzione monoalfabetica.

Il passo successivo dell'idea di Scherbius, consiste nel far ruotare automaticamente il disco scambiatore di un ventiseiesimo di giro dopo la cifratura di ogni lettera. In questo modo l'alfabeto cifrante cambia per ogni lettera. Vengono così definiti 26 alfabeti cifranti ed Enigma può essere usata per effettuare una cifratura polialfabetica. Tuttavia, il congegno ha un punto debole: dopo 26 pressioni consecutive di un tasto, il disco torna alla posizione iniziale e lo schema di cifratura si ripeterebbe tale e quale.

Il problema può essere risolto in misura notevole introducendo un altro scambiatore.

Ogni volta che una lettera è cifrata, il primo disco ruota di un carattere. L'altro disco, invece, resta immobile per gran parte del tempo. Esso compie una parziale rotazione solo quando il primo scambiatore ha completato un giro; questo è infatti dotato di un dente che, raggiunta una certa posizione, fa avanzare di un posto il secondo scambiatore.

L'aggiunta del secondo scambiatore, comporta il vantaggio che lo schema della cifratura non si ripete finché il secondo scambiatore non è tornato al punto di partenza; il che richiede 26 giri completi del primo scambiatore, ovvero la cifratura di  $26^2 = 676$  lettere.

Enigma è quindi capace di utilizzare un enorme numero di alfabeti cifranti passando continuamente da uno all'altro di essi. L'operatore batte una lettera particolare e, in conformità con l'assetto dei rotori, la lettera è crittata in base a uno delle centinaia di alfabeti cifranti disponibili. Subito dopo, l'assetto dei rotori cambia, cosicché la lettera successiva sarà crittata in base ad un altro alfabeto cifrante.

Per una sicurezza maggiore, il modello base di questa macchina per cifrare impiegava un terzo rotore. Quindi essa disponeva di  $26^3 = 17.576$  procedure di sostituzione diverse.

L'inventore aggiunse anche un **riflessore**. Esso era simile allo scambiatore, consistendo in un disco di gomma con circuiti interni, ma era diverso, sia perché non ruotava, sia perché i fili che entravano da un lato riemergevano dallo stesso lato. Col riflesore installato, quando l'operatore digitava una lettera, il segnale elettrico attraversava i tre scambiatori, raggiungeva il riflesore ed era rimandato indietro. Perciò, esso passava di nuovo dagli scambiatori, ma lungo un percorso differente.

Il riflesore, essendo statico, non fa aumentare il numero di alfabeti cifranti. La sua funzione è quella di permettere la decifrazione di un messaggio crittato. Supponiamo che un operatore volesse inviare una comunicazione cifrata. Prima di cominciare egli doveva regolare gli scambiatori, in modo che assumessero la posizione iniziale voluta. Quest'ultima era contenuta in un cifrario, che elencava le chiavi da usare di giorno in giorno ed era a disposizione di tutti gli operatori della rete. Una volta regolati i rotori, l'operatore poteva iniziare la codifica del messaggio, digitando il testo in chiaro per generare il crittogramma. Per decifrare il crittogramma, il destinatario doveva possedere un'altra macchina Enigma e un cifrario con l'assetto dei rotori da utilizzare giorno per giorno. Egli regolava la macchina come spiegato nel cifrario e digitava il crittogramma per ripristinare il testo chiaro.

Scherbius decise di accrescere ulteriormente l'affidabilità della sua invenzione aumentando il numero di assetti, cioè di possibili chiavi. Rese i **rotori removibili e sostituibili**. Siccome i rotori erano diversi, e diversi erano i movimenti che compivano durante la cifratura, quest'ultima era influenzata dalla posizione reciproca dei rotori. Questo accorgimento aumentava il numero di chiavi di un fattore pari a 6.

Inoltre egli inserì un **pannello a prese multiple** tra la tastiera e il primo rotore. Il pannello permetteva al mittente di inserire alcuni cavi muniti di spinotti, che avevano l'effetto di scambiare due lettere prima della loro immissione nel rotore.

Il numero di chiavi che Enigma poteva impiegare, dipendeva dalle caratteristiche del pannello, dei singoli rotori, e dell'unità cifratrice formata dai tre rotori. Gli scambiatori generavano 17.576 combinazioni diverse di orientamenti. L'unità cifratrice (i tre scambiatori potevano assumere diverse posizioni reciproche) ne generava 6. Il pannello a prese multiple generava 12 ( $6 * 2$ ) possibili abbinamenti di lettere su 26. Quindi il numero totale di chiavi era di circa 10 milioni di miliardi. Una volta presi accordi sui collegamenti del pannello, sull'ordine dei rotori e sul loro orientamento, mittente e destinatario potevano crittare e decrittare i messaggi con facilità.

Questa enorme quantità di chiavi può far pensare che i crittogrammi generati da Enigma siano inviolabili. Invece non è così. Il matematico Alan **Turing** riuscì a decrittarli, anche per mezzo di una apparecchiatura, per la decifrazione automatica, che egli stesso ideò. Essa venne chiamata **Bomba**.

Oltre alle Bombe di Turing, usate per tradurre i messaggi Enigma, i Britannici inventarono un altro dispositivo di decrittazione: si chiamava **Colossus**. Esso era stato progettato per debellare un sistema crittografico ancora più resistente: la **cifratura Lorenz**. Quest'ultima era usata per crittare le comunicazioni tra Hitler e i suoi capi di Stato Maggiore. La codifica era effettuata dalla **macchina Lorenz SZ40**, simile a Enigma nei principi di funzionamento, ma molto più complicata. Violare la cifratura Lorenz richiedeva indagini, confronti, analisi statistiche e capacità deduttiva – molto più di quanto le Bombe potessero garantire. Colossus, invece, era in grado di farlo. Infatti oltre a possedere un'elevata velocità, era anche programmabile. Esso è, quindi, uno dei precursori dei moderni elaboratori digitali.

### **3 Algoritmi di Cifratura Odierni**

#### **3.1 Cifrari a Blocchi**

##### **3.1.1 Concetti Generali**

I cifrari a blocchi rappresentano di questi tempi i modelli più usati per la cifratura simmetrica, soprattutto per quanto riguarda la trasmissione di dati via rete. Si parla di cifrario a blocchi quando il messaggio originale viene suddiviso in blocchi di carattere, per poi applicare su ciascun blocco la trasformazione crittografica.

Un cifrario a blocchi divide il testo in chiaro in blocchi usualmente di una dimensione fissata e opera su ogni blocco in modo indipendente. Un particolare blocco di testo in chiaro viene “mappato” nello stesso blocco di testo cifrato ogni volta che appare nel testo. I cifrari a blocchi sono quindi cifrari a semplice sostituzione e devono disporre di larghi alfabeti per contrastare l'analisi frequentistica. I blocchi a 64 bit del DES realizzano un alfabeto di  $2^{64}$  caratteri.

La grande popolarità dei cifrari a blocchi è dovuta al successo di DES, che è stato forzato la prima volta solo nel 1997, con un attacco esaustivo.

I concetti su cui si fonda un cifrario a blocchi sono due:

- Nessun bit del testo in chiaro dovrebbe mai apparire direttamente nel testo cifrato; piuttosto ogni bit del testo cifrato dovrebbe essere una funzione di tutti i bit della chiave e del testo in chiaro.
- Il cifrario dovrebbe essere progettato in modo tale che, cambiando anche un solo bit del testo in chiaro, o della chiave, cambino approssimativamente la metà dei bit del testo cifrato<sup>15</sup>.

### 3.1.2 DES

Il DES è il sistema ufficiale di cifratura utilizzato dal governo degli Stati Uniti, a partire dal 1977. Negli anni 1972-73, il governo americano indice una gara per la scelta di un algoritmo ufficiale. La proposta della IBM arriva con l'algoritmo DEA (Data Encryption Algorithm) derivato dall'algoritmo Lucifer di fattura precedente. La NSA sceglie il DEA come standard ed è annunciato sul documento N.46 di Federal Information, col nome di **DES**.

Questo metodo di cifratura è in realtà l'unione di 16 trasformazioni tramite trasposizione e sostituzione in successione. Più in dettaglio, il testo (o file) viene diviso in blocchi da 64 bit (8 Bytes), quindi viene operata una trasposizione, una serie di cifrature ed una nuova trasposizione inversa alla prima.

La chiave di crittografia è lunga 64 bit, ma 8 bit sono di controllo, quindi la chiave effettiva è 56 bit. Questo porta ad avere  $2^{56}$  possibili chiavi in un tentativo di attacco “brute force”<sup>16</sup>.

Tuttavia, anche senza basarsi sull'attacco di forza bruta, gli studiosi Biham e Shamir e Matsui hanno provato tecniche di forzatura differenziale, la crittoanalisi differenziale e la crittoanalisi lineare rispettivamente. I tempi di decrittazione sono comunque lunghi: il primo esperimento di Matsui richiese 9735 postazioni di lavoro e 50 giorni e 12 ore di tempo.

---

<sup>15</sup> Questa proprietà è nota sotto il nome di propagazione dell'errore (*error propagation*) ed è utile nei problemi di autenticità, poiché rende improbabile che un nemico possa alterare il messaggio in una maniera che non possa essere scoperta, a meno che non sia in possesso della chiave.

<sup>16</sup> Attacco a “Forza bruta”, cioè provando tutte le possibili combinazioni.

Il DES non viene più certificato dal National Institute of Standards and Technology (NIST, una divisione del dipartimento del commercio USA), ha tuttavia ancora larghissimo impiego, data anche l'elevata velocità di cifratura rispetto al suo rivale **RSA**<sup>17</sup> a chiave pubblica. Non può essere tuttavia utilizzato in quei casi dove il valore dell'informazione da proteggere superi certe cifre, dato che già tempo fa è stato dimostrato che in circa 3 ore, qualsiasi testo cifrato con DES può essere riportato in chiaro tramite un computer dal costo di 1 milione di dollari.

E' comunque stata costruita una versione del DES, chiamata Triplo DES, che utilizza tre chiavi diverse ad ogni passaggio di sovracifratura. Nonostante i risultati siano inferiori alle aspettative, è stato innalzato moltissimo il tempo necessario per un attacco di tipo brute force; di contro anche i tempi di cifratura sono aumentati, dovendo iterare tre volte il DES.

Le modalità di funzionamento del DES sono quattro:

- *ECB (Electronic Code Book)*: Costituisce la modalità di funzionamento predefinita. I dati vengono suddivisi in blocchi di 64 bits ed ogni blocco viene cifrato singolarmente indipendentemente dal precedente. In questo modo, nell'eventualità in cui si verificasse un errore, questo sarebbe limitato al blocco contenente l'errore stesso. Il principale rischio è dato dal fatto che, vista la totale assenza di ulteriori misure di sicurezza, se non quelle proprie dell'algoritmo, sarebbe possibile scambiare l'ordine dei blocchi cifrati risultanti senza che questo possa essere rilevato. Comunque, anche se effettivamente è la meno sicura, per le sue caratteristiche di velocità e di semplicità di implementazione costituisce la modalità più utilizzata.
- *CBC (Cipher Block Chaining)*: In questa modalità, ogni blocco di testo, dopo essere stato cifrato in modalità ECB, viene sottoposto ad una ulteriore operazione di OR esclusivo (XOR) con il successivo blocco ancora da cifrare. In questo modo, tutti i blocchi verranno resi dipendenti dai blocchi che li precedono così che, per trovarne il testo in chiaro, si renda necessario non solo conoscerne il relativo testo cifrato ma anche la chiave, ed il testo cifrato del blocco precedente. Da notare che, il primo blocco, non avendo un blocco precedente, verrà sottoposto all'operazione di XOR con un numero di 64 bits chiamato "Vettore di Inizializzazione" (Initialization Vector). Anche se più sicura ma al tempo stesso più lenta della modalità ECB, la logica seguita fa sì che, se durante la trasmissione si introduce un errore, questo verrà trascinato lungo tutta la sequenza dei blocchi rimanenti, visto che ogni blocco è dipendente dal precedente.

---

<sup>17</sup> Paragrafo 3.3.2

- *CFB (Cipher Feedback)*: Questa modalità permette di cifrare blocchi in chiaro inferiori a 64 bits così da eliminare la necessità dell'ulteriore elaborazione che normalmente si rende necessaria con i files aventi grandezza non multipla di 8 bytes. Il testo in chiaro non viene sottoposto all'algoritmo DES ma, ad un'operazione di OR esclusivo (XOR) con i dati uscenti dall'algoritmo, generati utilizzando inizialmente un blocco casuale di 64 bits, chiamato Shift Register, ed in uscita, un componente aggiuntivo, M-Box, che selezionerà gli M bits più a sinistra del blocco risultante così da farlo corrispondere alla grandezza del blocco che vogliamo cifrare. Il testo cifrato ottenuto al termine dell'operazione, diverrà il nuovo dato in ingresso all'interno dello Shift Register, da utilizzare con il successivo blocco da cifrare. Presenta tutte le caratteristiche sia positive che negative della modalità CBC.
- *OFB (Output Feedback)*: E' simile alla modalità CFB da cui si differenzia per il fatto che è il blocco in uscita dal ciclo DES che viene riutilizzato all'interno dello Shift Register, piuttosto che il risultato finale di tutta la procedura. A differenza delle modalità CFB e CBC, un eventuale errore di trasmissione non compromette tutta la procedura, visto che, una volta in possesso del valore iniziale casuale dello Shift Register, tutti i successivi concatenamenti vengono derivati da quest'ultimo. Allo stesso tempo però, questa modalità è meno sicura della CFB perchè, anche non conoscendo la chiave, basta il testo cifrato ed i dati in uscita dal ciclo DES per risalire al testo in chiaro dell'ultimo blocco.

### 3.1.3 IDEA

L'algoritmo **IDEA** (International Data Encryption Algorithm) è nato nel 1991 sotto il nome di IPES (Improved Proposed Encryption Standard), ideato da Xuejia Lai e James L. Massey.

Come il DES è un codice cifrato a blocchi, di lunghezza 64 bit, con una chiave però di 128 bit, che limita moltissimo la possibilità di riuscita della ricerca esaustiva nello spazio della chiave.

Anche questo come il DES può essere usato nei quattro modelli possibili: ECB, CBC, CFB e OFB. A differenza del DES, che era stato progettato per implementazioni hardware, IDEA è stato creato per il software.

La cifratura con IDEA comporta una divisione del blocco di 64 bit del testo normale in quattro sottoblocchi di 16 bit. Ogni sottoblocco subisce 8 round in cui sono coinvolte 52 sottochiavi diverse a 16 bit ottenute dalla chiave a 128 bit.

Le sottochiavi sono generate nel seguente modo:

- 1) la chiave a 128 bit è divisa in 8 stringhe di 16 che rappresentano le prime 8 sottochiavi;

2) le cifre della chiave a 128 sono shiftate di 25 bit a sinistra in modo da generare una nuova combinazione, il cui raggruppamento ad 8 bit fornisce le prossime 8 sottochiavi;

3) il secondo passo è ripetuto finché tutte le 52 sottochiavi sono generate.

Ogni round comporta calcoli abbastanza semplici come XOR, addizione modulare e moltiplicazioni modulari (addizione modulare significa che una somma non può superare i 16 bit quindi quelli in overflow vengono scartati; moltiplicazione modulare è in modulo ( $2^{16}$ )).

Durante i round il secondo e il terzo blocco si scambiano di posto mentre al round finale i quattro sottoblocchi vengono concatenati per produrre un blocco di testo cifrato a 64 bit.

L'operazione di decrittazione è identica eccetto per il fatto che le sottochiavi sono ottenute in maniera diversa dalla chiave principale a 128.

Non è chiaro se IDEA sia o meno migliore del Triple DES, ma alcuni crittologi pensano che il TDES sia più sicuro.

### 3.1.4 AES

**AES** è un algoritmo sviluppato da Joan Daemen e Vincent Rijmen sotto richiesta del NIST<sup>18</sup>. Il cifrario utilizza chiavi di lunghezza variabile (gli autori hanno dimostrato come è possibile variare le dimensioni delle chiavi con multipli di 32 bit). Lo schema del Rijndael è stato influenzato dall'algoritmo SQUARE<sup>19</sup>. Questo algoritmo ha vinto la selezione per l'Advanced Encryption Standard (AES) il 2 Ottobre 2000. Ufficialmente il Rijndael è diventato lo standard per la cifratura del XXI secolo.

La specifica originale di tale algoritmo prevedeva blocchi di 192 o 256 bit, ma il NIST ha previsto uno standard di 128 bit.

Il funzionamento di AES segue la falsariga di molti cifrari a blocchi:

I 128 bit del blocco vengono inseriti in una matrice 4 x 4 in cui ogni cella occupa 8 bit.

Ogni volta sono svolte quattro operazioni:

1) Ogni byte viene immagazzinato in memoria.

2) Ogni riga viene ruotata un numero di volte dipendenti dalla riga stessa; ad esempio la prima riga non è ruotata, la seconda è ruotata di un byte, la terza di due e così via.

3) Ogni colonna è moltiplicata con una matrice fissata di questo tipo:

---

<sup>18</sup> National Institute of standards and technology.

<sup>19</sup> È un cifrario a blocchi di tipo iterativo su insiemi di 128 bit con l'utilizzo di chiavi sempre di 128 bit. La funzione di arrotondamento di questo algoritmo è composta da quattro trasformazioni: una trasformazione lineare, una trasformazione non-lineare, una permutazione a livello di byte e una addizione sui bit con la chiave.

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

4) Viene fatto lo XOR di ogni cella con una chiave detta “round-key” solitamente di 128 bit, che dipende dalla chiave inserita dall’utente.

AES è stato scelto come standard dal NIST per le sue caratteristiche di velocità e sicurezza di cifratura.

### 3.1.5 Blowfish

Il Blowfish è un algoritmo crittografico con chiave variabile da 32 bit fino a 448 bit progettato nel 1993 da Bruce Schneier come sostituto di DES o IDEA, ma senza brevetto, con licenza libera e senza diritti di sfruttamento.

L’algoritmo è diviso in due parti: espansione della chiave e cifratura dei dati.

La sezione di espansione della chiave prende la chiave (che al massimo è 448 bit) e la suddivide in tante sotto-chiavi, per un totale di 4168 byte.

La sezione di cifratura dei dati è costituita da una semplice funzione iterata 16 volte. A ogni iterazione vengono eseguite le seguenti operazioni: permutazioni dipendenti dalla chiave, sostituzioni dipendenti dalla chiave e dai dati. Tutte le operazioni effettuate sono addizioni e XOR su parole di 32 bit.

Blowfish fa largo uso di sotto-chiavi, che devono essere generate necessariamente prima della cifratura dei dati.

Ancora non sono stati trovati metodi per attaccare l’algoritmo, a parte l’attacco esaustivo, che comunque risulta improponibile per la lunghezza della chiave, che può arrivare anche a 448 bit.

Il suo punto di forza è la velocità, e in più è libero e gratuito.

## 3.2 Cifrari a Flusso

### 3.2.1 Concetti Generali

Il cifrario a flusso è un un approccio alternativo a quello dei cifrari a blocchi. A differenza dei cifrari a blocchi nei quali la stessa chiave  $k$  viene riutilizzata per la cifratura di ogni blocco, l'idea di base degli "stream cipher"<sup>20</sup> consiste nel generare una sequenza, a partire da  $k$ , detta keystream:  $z = z_1z_2\dots$  e nell'utilizzarla per cifrare la stringa  $x = x_1x_2\dots$  del testo in chiaro.

Un metodo per generare la keystream a partire da  $m$  valori  $k_1, \dots, k_m$ , consiste nel fissare i valori per  $z_1, \dots, z_m$  con  $z_i = k_j$  per ogni  $i = 1, 2, \dots, m$  e nel calcolare i valori successivi in base ad una relazione di ricorrenza lineare di grado  $m$ :

$$z(i+m) = \sum_{j=0}^{m-1} (c_j * z(i+j)) \pmod{2}$$

dove  $c_0, \dots, c_{m-1}$  sono costanti predeterminate che individuano univocamente un polinomio chiamato polinomio delle connessioni che ha la forma seguente:

$$c_0z^m + c_1z^{m-1} + \dots + c_{m-1}z + 1$$

Questa ricorrenza è detta lineare perché  $z_{i+m}$  è funzione lineare dei termini precedenti, ed ha grado  $m$  dato che ciascun termine dipende dagli  $m$  precedenti. In tal caso la chiave consiste dei  $2m$  valori

$$(k_1, \dots, k_m, c_0, \dots, c_{m-1}),$$

dove  $k_i = z_i$  per  $i = 1, \dots, m$ .

La keystream in alcuni casi può essere generata anche in hardware utilizzando un linear feedback shift register (LFSR), un registro che genera un bit di output in base ai suoi stati precedenti e ad un polinomio di feedback. Tali registri sono caratterizzati da polinomi, denominati polinomi delle connessioni.

Rispetto ai block cipher, gli stream cipher sono generalmente più veloci. Normalmente sono anche più appropriati e, in qualche caso, obbligatori (tipo nelle applicazioni di telecomunicazioni), dove la lunghezza del buffer è limitata o dove i caratteri devono essere processati così come sono ricevuti. Avendo una propagazione di errore scarsa o nulla, essi possono essere usati in circostanze in cui gli errori di trasmissione sono molto probabili.

Ci sono ancora pochi algoritmi di cifrari a flusso, e quei pochi presenti spesso sono proprietari; al contrario la maggior parte degli algoritmi di cifrari a blocchi sono ormai diventati di pubblico dominio. Nonostante questo, gli stream cipher sono largamente usati per via indiretta<sup>21</sup> oggi giorno, e si può prevedere un sicuro aumento del loro impiego negli anni futuri.

<sup>20</sup> Cifrari a flusso.

<sup>21</sup> PGP ed RSA utilizzano RC4.

### **3.2.2 RC4**

RC4 è un cifrario a flusso basato su chiave simmetrica. È stato sviluppato nel 1987 da Ronald Rivest e mantenuto da RSA Data Security. Nel settembre del 1994 l'algoritmo fu mandato su Internet in forma anonima a "Cyberpunks' anonymous remailers list".

RC4 usa una chiave lunga da 1 a 256 byte per inizializzare una tabella di stato da 256 byte. Questa tabella è utilizzata per la generazione di byte pseudo-casuali e quindi per generare flussi pseudo-casuali su cui è fatta l'operazione logica XOR con il testo in chiaro per ottenere il testo cifrato. Ogni elemento nella tabella di stato viene scambiato almeno una volta.

La chiave del RC4 è spesso limitata a 40 bit (anche se è già stata violata con un brute force), e a volte viene presa anche a 128 bit. A livello teorico, la chiave può andare da 1 a 2048 bit. RC4 è usato in software tipo Lotus Notes e Oracle Secure SQL, e fa anche parte della "Cellular Specification".

### **3.2.3 SEAL**

Sviluppato da Don Coppersmith dell'IBM Corp, il SEAL rappresenta probabilmente il più veloce algoritmo di crittografia attualmente disponibile. Le chiavi che utilizza richiedono diversi kilobyte di spazio, ma bastano solo 5 operazioni per byte per la generazione della keystream. Questo algoritmo è particolarmente appropriato per le applicazioni di crittografia su dischi e in tutte le applicazioni nelle quali è necessario cifrare un blocco di dati con letture variabili dal centro.

Il SEAL è registrato dall'IBM che ne detiene la licenza.

### **3.2.4 ORIX**

ORYX è un algoritmo utilizzato per cifrare i dati delle comunicazioni cellulari. È un cifrario basato su tre funzioni di Galois LFSR a 32 bit. Si distingue dall'algoritmo CMEA, un cifrario a blocchi utilizzato per la protezione dei dati del canale di controllo delle comunicazioni cellulari. Il team crittografico della Counterpane Systems (composto da David Wagner, John Kelsey e Bruce Schneier) è riuscito a sviluppare un attacco all'ORYX basato sulla conoscenza di circa 24 byte di un testo cifrato con circa 216 parametri iniziali.

## **3.1 Cifratura a Chiave Pubblica**

### **3.3.1 Concetti generali**

Il problema principale che affliggeva tutti i metodi di crittografia consisteva nello scambio della chiave, ossia nel prendere accordi con il destinatario del messaggio su quale codice usare. Supponiamo ad esempio che i due corrispondenti scelgano di scambiarsi messaggi tramite una corda con alcuni nodi particolari. È necessario che i due si incontrino almeno una volta e in gran segreto per concordare la sequenza e il significato dei nodi, altrimenti il messaggio risulterebbe assolutamente incomprensibile. È ovvio che se lo "scambio della chiave" fosse spiato da qualcun'altro quest'ultimo potrebbe a sua volta decifrare i messaggi una volta entrato in possesso della corda annodata, invalidando in questo modo l'intero sistema. Se dover far incontrare i due corrispondenti poteva essere problematico un tempo, quando magari i due abitavano a parecchi giorni di cammino l'uno dall'altro, è praticamente impossibile ai giorni nostri, almeno su vasta scala. Con l'avvento delle reti telematiche è possibile corrispondere con persone ai quattro angoli del globo, e ovviamente diventa impossibile andare a visitarle tutte ... Questo problema ha impedito il diffondersi di sistemi di crittografia come ad esempio il DES che usano una sola chiave per codificare e decodificare il messaggio. Il problema è stato risolto in modo brillante da Whitfield Diffie, in collaborazione con il professor Martin Hellmann della Stanford University, che creò un sistema di cifratura detto "a chiave pubblica". Il concetto è abbastanza facile da capire; nei sistemi di crittografia usati fino ad allora la "password" era unica, sia per codificare che per decodificare il messaggio. Con i sistemi a chiave pubblica le cose non stanno più così le chiavi sono due, una per codificare e una per decodificare. Entrambe sono generate dal programma quando l'utente glielo chiede, e sono univoche, cioè non possono essere utilizzate per lo scopo inverso, ed è praticamente impossibile ottenere l'una quando si è in possesso solo dell'altra. Fatto questo basta rendere pubblica (appunto) la chiave per codificare e tenere ben nascosta quella per decodificare. In questo modo chiunque voglia mandarmi un messaggio potrà farlo, stando certo che io e solo io potrò leggerlo. Anzi, potrà a sua volta mandarmi, magari nella stessa mail, la sua chiave pubblica, così che io possa rispondergli in modo assolutamente sicuro. Ecco, è proprio partendo da questo principio che è nato PGP, il quale consente appunto di generare una coppia di chiavi, in congiunzione biunivoca tra loro, e di rendere pubblica ove si vuole la chiave per decodificare (anzi più viene pubblicizzata e meglio è) trattenendo per se la chiave privata. Anzi, mi permette di fare anche molte altre cose, ad esempio

di lasciare un messaggio "in chiaro" e di "firmarlo" elettronicamente con la mia chiave privata, in gergo si dice che si effettua "**l'hash**" di un messaggio. Chiunque posseda la mia chiave pubblica potrà poi chiedere al programma di verificare che il messaggio riportato nel testo sia effettivamente quello che ho scritto io e che non sia stato modificato da altri.

### 3.3.2 RSA

Il codice RSA permette di cifrare un messaggio attraverso un procedimento che richiede l'utilizzo dei numeri primi

- Si determini la prima chiave  $n$ , prodotto di  $p$  e  $q$ , due numeri primi molto elevati, tali che la fattorizzazione di  $n$  sia difficile o perlomeno  $n$  risulti una funzione unidirezionale rispetto al tempo d'uso del codice.  $N$  viene infatti resa pubblica.

Esempio:  $n=pq=5*7=35$

- Si calcoli dunque il valore della funzione di Eulero in  $n$ :  $b=f(n)=(p-1)*(q-1)$  il cui valore rimane segreto; si scelga ancora un intero  $d$  tale che  $d$  e  $f(n)$  siano primi tra loro, infine il suo inverso  $h$ , ovvero il più piccolo  $x$  per cui  $(dx-1)/f(n)$  è un intero, il numero  $h$  è la seconda chiave, e viene reso pubblico, mentre  $d$  resta segreto.

Esempio:

$$b=f(n)=(5-1)*(7-1)=4*6=24$$

$$d=7$$

$$k=7x-1/24$$

$$7x-1=k*24$$

$$7x=k*24+1$$

$$x=k*24+1/7$$

Sostituisco a  $k$  2 per far risultare  $x$  un numero intero ed ottengo  $x=7$ , dunque  $h=x=7$

- Per trasmettere il messaggio lo si traduce inizialmente in un vettore di numeri (in precedenza ci si è accordati riguardo alla modalità di "traduzione"). Stabilita dunque la sequenza numerica  $m_1, m_2, \dots, m_r$  si trasmettono gli  $m$  uno alla volta. Il crittogramma corrispondente a  $m$  è allora  $c=m^n \bmod n$ .

Esempio:

prendiamo  $m=3$

$$c=m^n \bmod n=3^7 \bmod 35=2187 \bmod 35=17$$

- La chiave di decifrazione è costituita dall'intero  $b$ , segreto, nella formula  $m=c^b \bmod n$ .

Esempio:  $m=c^b \bmod n=17^{24} \bmod 35=3$

In sintesi: per cifrare un messaggio dunque il trasmettitore deve prendere le diverse cifre pubbliche del ricevente e costruire un messaggio cifrato, quest'ultimo a sua volta utilizza la parte segreta del suo codice per decifrarlo.

Utente	Parte pubblica		Parte segreta		
Ricevente	N	n	n=p*q	b=f(n)	d

Il codice RSA viene considerato sicuro perchè, essendo la formula di decifrazione basata su  $f(n)$  calcolabile solo se a conoscenza di  $p$  e  $q$ , non esiste un algoritmo efficiente per scomporre  $n$  in  $p$  e  $q$ , perlomeno in tempi accettabili.

Potrebbe sorgere il dubbio che esista un modo di calcolare  $f(n)$  senza passare per  $p$  e  $q$ : questa ipotesi in effetti è verificabile ma ha lo stesso grado di complessità di fattorizzare  $n$ . Osserviamo che per un crittoanalista rompere l'RSA equivale a calcolare  $f(n)$ . Infatti, se  $n$  e  $f(n)$  sono conosciuti ed  $n$  è il prodotto di due primi  $p$  e  $q$ ,  $n$  può essere facilmente fattorizzato risolvendo le due seguenti equazioni:  $n = p \times q$  e  $f(n) = (p - 1)(q - 1)$ . Nelle due incognite  $p$  e  $q$ . Sostituendo  $q = n / p$  nella seconda equazione se ne ottiene un'unica di secondo grado nella sola incognita  $p$ :

$$p^2 - (n - f(n) + 1)p + n = 0.$$

Le due radici di questa equazione sono i fattori  $p$  e  $q$ . Quindi se un crittoanalista conosce il valore di  $f(n)$  può fattorizzare  $n$  e rompere il sistema.

## 4 Sviluppo di un nuovo algoritmo di cifratura simmetrica: A.R.C.C.

### 4.1 Scelta del linguaggio di programmazione

La scelta del linguaggio di programmazione rappresenta un passo importante nel processo di sviluppo del software. Uno sbaglio nella scelta del linguaggio, comporta un'implementazione pratica del programma completamente inefficiente, che andrebbe a "demonizzare" tutti i buoni propositi duramente "partoriti" nella fase di progettazione.

Nel caso particolare del progetto ARCC, questo discorso assume una piega diversa poichè l'algoritmo non individua alcuna complessità di alto rilievo; il linguaggio di programmazione, per questa motivazione, non rappresenta dunque una peculiarità di fondamentale importanza.

Il linguaggio adottato è stato il piu' che celeberrimo C. Non c'è infatti una particolare ragione per cui si sarebbero dovuti preferire linguaggi di programmazione basati e/o orientati agli oggetti (i.e. C++ / Java ecc.) a linguaggi procedurali (i.e. C / Turbo Pascal ecc.).

La scelta del C è principalmente legata alla grande "popolarità" acquisita da questo linguaggio e alla disponibilità del relativo compilatore che accompagna di default i sistemi operativi più in voga al momento.

## 4.2 Implementazione

ARCC consta di due livelli: Polialfabetico e XOR. Nei prossimi paragrafi è analizzato il loro sviluppo. Inoltre, vista l'importanza che hanno i numeri casuali nel livello polialfabetico, si è deciso di dedicare sull'argomento un intero paragrafo.

### 4.2.1 Numeri casuali

Il progetto ARCC fa ampio uso di numeri casuali nella parte inerente il Polialfabetico.

I numeri casuali utilizzati sono stati generati dalla funzione standard rand() fornita con il C. Essa però, se presa a sè ed iterata, dà come risultato sempre la stessa sequenza di numeri casuali. La funzione rand() del C infatti, per generare numeri pseudo-casuali usa dei procedimenti matematici basati su di una radice (detta "seed"). Ad ogni seed corrisponde una certa sequenza di numeri casuali. Tale seed ha un valore di default pari a 1, che genera sempre la stessa sequenza di numeri casuali, è importante quindi cambiare il seed per avere un buon numero di diverse sequenze casuali. Nel caso del rand() del C, si può cambiare il seed con la funzione "srand(unsigned int seed)". Il seed è un unsigned int; questo vuol dire avere la possibilità di scelta tra  $2^{32}$  seed, cioè  $2^{32}$  diverse sequenze di numeri casuali. E' inoltre importante che il seed non sia generato in maniera deterministica bensì completamente casuale. Per far ciò si è utilizzato il seguente algoritmo.

```
float casuale()          //genera numero casuale
{
    int i, risultato[NUM_BIT], j, fd, k, f    ;
    float risultato_casuale = 1.0          ;
    float a                                  ;

    char nomefile[]="qazwsxedcrfvgtgbyhnujmiko.tmp";
    struct timeval start_time, stop_time    ;
    int stime, etime, eltime                ;
    memset( risultato, 0, NUM_BIT )        ;

    for( j=0; j<NUM_BIT-1; j++ )
    {
        gettimeofday(&start_time, NULL)    ;
        i = 0                               ;
```

```

while( i<10000 )
{
    f = open( nomefile, O_CREAT, 0644 ) ;
    close(f) ;
    i++ ;
}

gettimeofday(&stop_time, NULL);
stime = start_time.tv_usec; //+ 0.000001*start_time.tv_usec;
etime = stop_time.tv_usec; // + 0.000001*stop_time.tv_usec;
etime = (etime - stime);
risultato[j] = etime % 2;

}

risultato_casuale =
1.0*trasformatoreottobit(risultato)/(potenza(2,NUM_BIT)-1);

remove(nomefile);
return risultato_casuale;

}

```

Come si può vedere, esso calcola il tempo in millisecondi per effettuare 10.000 open()/close() di un file; essendo questa operazione tutt'altro che banale (al suo posto può essere usata qualunque altra operazione che non sia eccessivamente semplice, o possono essere effettuate delle operazioni ad-hoc in assembler) è stata introdotta un'incertezza nel calcolo del tempo di esecuzione. L'incertezza riguarda ovviamente solo i millisecondi del tempo di esecuzione, il tempo esatto varierà, anche se di pochissimo, di volta in volta di pochi millisecondi. L'incertezza così determinata, è stata sfruttata per eseguire la seguente operazione:  $\text{tempo\_esecuzione} \% 2$ , che restituisce come risultato 0 o 1. Si può affermare a questo punto di aver ottenuto un bit casuale. Si può iterare questo procedimento un numero arbitrario di k volte (nel nostro caso 10 volte) e ottenere quindi un numero binario di k bit, che facilmente può essere convertito in formato decimale.

Per verificare la bontà della metodologia sudescritta, si è effettuato un test simulando 10000 “lanci di una moneta a due facce non truccata”, con le seguenti corrispondenze: testa=0, croce=1. Ecco i

risultati ottenuti:

n\_teste=51.3%

n\_teste=49.5%

n\_teste=47.2%

n\_teste=51.0%

n\_teste=52.3%

n\_teste=51.8%

n\_teste=49.8%

Si vede che la frequenza di uscita della testa è molto vicina al 50% ed essendo il numero di lanci molto alti, tale frequenza rappresenta con buona approssimazione la probabilità di uscita di una singola faccia. Questa probabilità(frequenza) è molto vicina alla probabilità di uscita teorica, che è del 50%, quindi il test si è concluso con risultati positivi e si è dimostrato empiricamente che i numeri generati dall'algorithm sopra citato, sono effettivamente casuali.

Visti gli ottimi risultati ottenuti per la generazione dei numeri casuali, si era pensato di utilizzarlo per la generazione di tutti i numeri casuali, non solo per il seed. In realtà questo meccanismo è ottimo se devono essere generati relativamente “pochi” numeri casuali, ma diventa eccessivamente lento se devono essere generati tantissimi numeri casuali. Questo è stato il motivo per cui si è usato tale stratagemma solo per la generazione del seed, che offre comunque una sufficiente “casualità”.

#### **4.2.2 Approccio alla funzione Polialfabetica**

Nel corso della storia la tecnica della sostituzione è stata ampiamente utilizzata.

Questo sistema, nonostante le varie sfaccettature, prevede un collegamento biunivoco tra il carattere in chiaro e il carattere cifrato, per esempio se il carattere da cifrare è ‘A’ e si sceglie di cifrarlo con ‘W’, esso sarà cifrato per tutta l’elaborazione del testo con ‘W’<sup>22</sup>, quindi è abbastanza facile elaborare un algoritmo che, basandosi sulla frequenza<sup>23</sup> di utilizzo di alcune lettere, in una certa lingua(e.g. in italiano ‘a’ è molto più frequente di ‘z’), riesca a decifrare il testo cifrato.

Lo scenario su descritto potrebbe essere reso più complesso se un carattere venisse sostituito da una stringa<sup>24</sup> di caratteri di lunghezza determinata. In realtà anche una cifratura di questo genere

---

<sup>22</sup> Cifrario monoalfabetico.

<sup>23</sup> Attacco frequentistico.

<sup>24</sup> Cifrari polialfabetici.

risulterebbe ancora facilmente attaccabile, è sufficiente infatti ricercare le ripetizioni delle stringhe nel testo cifrato e riapplicare l'attacco frequentistico.

Nell'algoritmo **ARCC**, si è pensato di usare una sostituzione, di seguito descritta, un po' più complessa ma in compenso molto più efficace, per evitare di incombere in situazioni spiacevoli.

In **ARCC**, un carattere viene sempre sostituito da una stringa di più caratteri, ma questa volta non esclusivamente da una stringa di lunghezza fissa, bensì essa è scelta tra più stringhe di varia lunghezza. Si è scelto di optare convenzionalmente tra stringhe di lunghezza variabile fra i cinque e i venti caratteri.

Le stringhe, che vengono usate per sostituire un carattere, sono generate da un primo programma e memorizzate in una libreria, svolgente funzioni di database. Questa libreria viene successivamente inclusa nella parte di programma che si occupa della sostituzione.

La soluzione, ottima relativa alla scelta delle stringhe per la sostituzione dei caratteri, sarebbe quella di adottare un meccanismo di associazione puramente casuale, dando origine però ad un ulteriore problema: come può sapere, colui che deve decifrare, quale stringa è stata scelta per sostituire il carattere?

Adottando una soluzione di assegnamento casuale, sarebbe impossibile.

L'unico modo per far fronte a questo inconveniente è quello di inserire delle lettere di controllo all'inizio di ogni carattere sostituito, al fine di indicare la lunghezza della stringa sostituyente. Inoltre è indispensabile dare delle regole a queste lettere di controllo, ad esempio si potrebbe scegliere, per convenzione, di usare la 'a' per indicare stringhe di cinque caratteri, la 'b' per stringhe di sei, e così via. Questo meccanismo produce tre fastidiosi inconvenienti:

- 1) il sorgente non potrebbe essere libero(open source) poiché queste regole sarebbero conosciute da tutti e i caratteri verrebbero quindi facilmente scoperti.
- 2) il concetto di password e di personalizzazione del cifrario sarebbe del tutto inesistente.
- 3) il sistema di cifratura sarebbe legato esclusivamente al binario.

Al fine di risolvere i suddetti "intoppi", si è cercato di utilizzare delle regole che legassero il cifrario in maniera diretta alla password, come segue.

Per quanto concerne il carattere di controllo, ad ogni carattere ASCII viene associato un numero da cinque a venti, che indica la lunghezza della stringa relativa al carattere in chiaro(la stringa sostituyente). Per ottenere un numero in questo intervallo viene eseguita la seguente operazione aritmetica:

$$X \% 16 + 5$$

X deve dipendere dalla password e dal carattere di controllo, in questo caso sono stati presi due caratteri (chpwd1 e chpwd2), dipendenti dalla password, trovati facendo delle operazioni logiche di and, or e XOR fra i bit della password.

Una buona operazione per trovare la X è:  $((\text{chPwd1}+1)*(\text{chPwd2}+1)*256^4/((\text{carattere di controllo in ASCII}+1)*(\text{carattere di controllo in ASCII}+1)))$ .

Come si può notare X dipende sia dalla password sia dal carattere di controllo.

Ecco un semplice esempio pratico di cifratura Polialfabetica **ARCC**:

#### Encrypt:

carattere da cifrare: 'A'

supponiamo che il numero casuale generato sia 100, corrispondente a 'd', e che  $\text{chpwd1}='a' (=97)$  e  $\text{chpwd2}='b' (=98)$ .

Alla 'd' viene quindi associato il numero:

$$(\text{int})((97+1)*(98+1)*256^4/((100+1)*(100+1))) \bmod 16 + 5 = 17$$

ad 'A' vengono associate le stringhe : (5) "qwert", (6) "asdfgg", (7) "zxfgrte", (8) "lkjhpoi", (9) "wsxedcrfv"....."dsafwrqplokijuhyg" (17) ..... (20)

'A' quindi viene cifrato come:

**ddsafwrqplokijuhyg**

#### Decrypt:

stringa da decifrare sopra definita = "ddsafwrqplokijuhyg"

Si legge il primo carattere, in questo caso 'd' (=100)

Si fa la stessa operazione di prima e, se la password è corretta ( $\text{chpwd1}='a'=97$ ,  $\text{chpwd2}='b'=98$ ), si arriva alla stessa soluzione:

$$(\text{int})((97+1)*(98+1)*256*256/((100+1)*(100+1))*256*256) \bmod 61 + 5 = 17$$

A questo punto si sa che dopo la lettera 'd' le successive 17 rappresentano il vero carattere, è sufficiente attuare una ricerca tra le stringhe di 17 caratteri e trovare quella uguale a quella data, a cui corrisponderà il carattere giusto 'A'.

Le operazioni suddette devono essere iterate per tutte le lettere dei file.

La parte della funzione encrypt che determina la complessità temporale è la seguente:

do

{

    a = read( fd, &i, 1)           ;

    if(a == 0) break           ;

    rd1 = rand0\_1()\*256       ;

    control = (char)rd1       ;

    nlet = controllo[control]   ;

```

write( new_fd, &control ,1);
for( j=0; j<nlet; j++ )
{
    write( new_fd, &stringhe[nlet-NMIN][i][j] , 1);
    //NMIN è il numero minimo di lettere per sostituire un carattere
}
} while( a!= 0);

```

Per ogni lettera del file in chiaro l'operazione più lenta da eseguire è la scrittura su file. In particolare vengono scritti sul file cifrato un carattere di controllo e la stringa con cui sostituire il carattere del file in chiaro, di lunghezza massima venti e media tredici. In totale per ogni lettera del file in chiaro al massimo vengono eseguite ventuno scritture su file, in media quattordici.

Per quanto riguarda il decrypt, ecco la parte che pesa maggiormente sulla complessità temporale:

```

do
{
    a = read( fd, &ch ,1 ) ;
    if( a == 0 ) break ;

    nlet = ctrl[i];
    for( j=0; j<nlet; j++ )
    {
        a = read( fd, &controllo[j] ,1 ) ;
        if( a == 0 ) break ;
    }
    if( a == 0 ) break ;
    trovato = 0 ;
    for( k=0; k<NUM_LETT; k++ )
    {
        j = 0 ;
        while( j<nlet )
        {
            stringa[j] = stringhe[nlet-NMIN][k][j];
            //NMIN è il numero minimo di lettere per sostituire
            j++ ;
        }
        stringa[j] = "\0";
        if( strcmp( stringa, controllo, j ) == 0 )
        {
            new_ch =(char)k ;
            write( new_fd, &new_ch, 1 ) ;
            trovato = 1 ;
        }
    }
}

```

```
} while( a != 0 ) ;
```

Le operazioni più “dure” da eseguire sono la lettura dal file cifrato e la ricerca tra le stringhe da sostituire, attraverso la funzione `strcompare()`<sup>25</sup> così definita:

```
int strcompare( unsigned char s1[], unsigned char s2[], int dim)
{
    int i ;
    for( i=0; i<dim; i++ )
    {
        if( s1[i]!=s2[i] ) return 1;
    }
    return 0;
}
```

Per ogni coppia, carattere di controllo  $\leftrightarrow$  stringa sostituita, vengono effettuate al più  $(1 + 20)$  letture da file (media  $13+1$ ), e 256 ricerche (media 128), quindi altrettante chiamate alla funzione `strcompare()`. Essa è costituita da un ciclo `for()`, che esegue operazioni di confronto, eseguito un numero di volte pari alla lunghezza delle stringhe da confrontare, quindi al massimo venti. In tutto sono effettuate al massimo  $20*256 = 5120$  (media  $13*128=1664$ ) operazioni per la ricerca.

In totale, per ogni carattere del file in chiaro che viene calcolato, sono eseguite ventuno letture dal file cifrato (media quattordici) e 5120 (media 1664) operazioni per la ricerca, che comunque sono solo operazioni di confronto quindi aventi complessità asintotica  $\theta(1)$ .

L’algoritmo globalmente è lento sia in fase di cifratura che di decifratura, perciò è consigliato il suo utilizzo solo nei casi in cui si cercasse un’estrema sicurezza, a discapito della velocità.

### 4.2.3 Approccio allo “XOR”

Il secondo livello di criptazione da noi utilizzato è fondato sull’uso dell’operatore logico XOR. Esso si basa appunto sull’operazione di XOR tra i caratteri del file in chiaro e una certa chiave.

La tabella di verità dell’operatore XOR è riportata in basso.

A	B	AXORB
0	0	0
0	1	1
1	0	1
1	1	0

**Tabella 5:** tavola della verità dell’operatore XOR

<sup>25</sup> `Strcompare()` è come `strcmp()`, ma prende come argomento anche la lunghezza delle stringhe.

Se si vuole effettuare lo XOR tra un carattere  $x$  e un carattere  $y$  si ottiene  $z$ ; per tornare al carattere di partenza  $x$  si fa uno XOR tra  $z$  e  $y$ , infatti questo operatore è di tipo “simmetrico”<sup>26</sup>, quindi si presenta come un ottimo metodo per la cifratura reversibile.

Nella fase di progettazione dell’algoritmo **ARCC**, sono state idealizzate varie tecniche di utilizzo dello XOR, per cifrare i dati in uscita dallo strato Polialfabetico, cercando di eliminare tutte le possibili tecniche miranti a spezzare la robustezza dell’algoritmo.

In linee teoriche si è cercato di avvicinarsi il più possibile alle caratteristiche ideali della cifratura perfetta (è il teorema di Shannon, vedi paragrafo “L’algoritmo perfetto”).

Chiaramente tali presupposti continuano a rimanere più vicini ad un’approccio teorico del problema piuttosto che all’implementazione vera e propria, ma è ingegneristicamente sufficiente trovare una soluzione che soddisfi, entro certi canoni, i requisiti suscitati, cercando di renderli per lo meno applicabili nella realtà.

Un primo approccio è stato quello di prendere una chiave di 8 bit e fare lo XOR di tutti i caratteri del file con tale chiave. La chiave sarebbe stata ricavata dalla password facendo ad esempio uno XOR tra i vari caratteri della password stessa. La chiave dipendente della password è fondamentale, perché deve apparire casuale agli occhi di un utente che non conosce la password stessa.

La chiave di 8 bit si dimostra però troppo debole. Bastano infatti 256 ( $2^8$ ) tentativi per trovare la combinazione giusta.

Un approccio molto migliore è quello di costruire una chiave più lunga; ma come è possibile ottenere ciò?

Per questo scopo viene in soccorso la funzione di hashing MD5.

MD5 è un algoritmo per calcolare la funzione di hash di un file: permette di generare un identificativo univoco di un file o di un flusso di dati o semplicemente di una stringa. (Gli algoritmi di hash sono utilizzati solitamente per verificare l’integrità dei dati trasmessi e rivestono un ruolo fondamentale per la firma digitale).

In poche parole MD5 riceve una stringa qualsiasi in ingresso e restituisce una stringa di lunghezza fissa (16 caratteri – 128 bit) dipendente dalla stringa in ingresso. Variando la stringa d’ingresso (anche di pochissimo, è sufficiente un carattere) si ottengono risultati completamente diversi. La stringa in ingresso può avere qualunque lunghezza, provocando delle “collisioni” (a due stringhe diverse viene assegnata da MD5 la stessa stringa), che sono comunque imprevedibili e difficili da scovare. MD5 viene usato da quasi tutti i sistemi “Unix-like” per conservare le password in modo sicuro.

---

<sup>26</sup> Altrettanto non sono gli altri operatori logici AND e OR, in quanto dal carattere finale non sarebbe più possibile tornare al carattere di partenza.

In definitiva si è scelto di dare come ingresso alla funzione MD5 la password definita dall'utente, ottenendo una stringa di 32 caratteri da noi denominata "verme".

Il "verme" è impiegato per operazioni logiche tra i caratteri: viene eseguito uno XOR fra il primo carattere del file in chiaro ed il primo carattere della stringa, il secondo carattere del file in chiaro ed il secondo della stringa, e così via fino a che non si raggiunge la fine del verme.

Arrivati a questo punto, si è scelto di usare nuovamente il verme partendo dal primo carattere, mentre i caratteri del file continuano a scorrere, e fare le stesse operazioni precedenti. Ogni volta che viene raggiunta la fine della stringa generata da MD5, essa viene riavvolta; questo comporta una cifratura simile a quella di Vigénère (2.8), in cui il "verme" è rappresentato dalla stringa generata dalla funzione MD5. Essa è vulnerabile ad attacchi di tipo frequentistico (anche se in questo caso sono molto più difficili da attuare per la presenza del livello Polialfabetico).

A causa di questa vulnerabilità, un verme ciclico non è sufficiente, ma è necessaria l'esistenza di un unico verme lungo quanto tutto il messaggio (cifratura perfetta di Vernam). Per realizzare questo obiettivo è stato realizzato un'anagramma della password secondo il seguente schema:

```
strcpy( genString, pwd )           ;
genString[0]=pwd[strlen(pwd)-3]    ;
genString[1]=pwd[strlen(pwd)-1]    ;
genString[2]=pwd[strlen(pwd)-2]    ;
genString[strlen(genString)-1]=pwd[2] ;
genString[strlen(genString)-2]=pwd[3] ;
genString[strlen(genString)-3]=pwd[0] ;
```

Una volta generato l'anagramma, si è eseguita la funzione MD5 sui suoi caratteri, ottenendo una stringa denominata "ausiliaria<sup>27</sup>".

Si è presa una stringa temporanea contenente:

- la prima metà del verme precedentemente discusso (MD5 della password) (8 caratteri)
- la seconda metà della stringa ausiliaria (8 caratteri)
- MD5 della password (16 caratteri)
- Stringa ausiliaria (16 caratteri)

Su di essa è eseguita la funzione MD5, la cui stringa in uscita costituirà la prima parte del verme.

Da questo momento viene impiegato il metodo iterativo. E' presa una stringa temporanea contenente:

- 16 caratteri fra gli ultimi 32 del verme attraverso un rimescolamento (vedasi algoritmo)
- MD5 della password (16 caratteri)
- Stringa ausiliaria (16 caratteri)

su cui ne viene lanciata la funzione MD5 e la cui uscita sarà il proseguimento del verme.

---

<sup>27</sup> Nome da noi associato.

Questo processo è eseguito ciclicamente per tutta la lunghezza del messaggio.

Sul “verme” viene cancellata ogni traccia della password in MD5 e gli vengono aggiunte le stringhe sopra citate; per sapere come vengono aggiunte le stringhe, bisogna conoscere necessariamente o direttamente la password, o MD5 della password e della stringa ausiliaria. Questo è il motivo per cui l’attaccante, anche se riuscisse a scoprire un pezzo di “verme”, non potrebbe decifrare l’intero messaggio, non sapendo come tale “verme” si allunga.

Ecco riportato un esempio che può chiarire il suddetto metodo:

Si supponga per semplicità di avere la funzione MD5 che restituisce stringhe di 4 caratteri anziché 32.

Si indichi con MD5(x) la stringa in uscita da MD5 con ingresso x.

Password : password

Password anagrammata: pwd\_anag

Variabile temporanea = temp

MD5(password)=”asdf” e MD5(pwd\_anag)=”qwer” ← stringa ausiliaria

Si prenda la prima metà di MD5(password) e la seconda metà di MD5(pwd\_anag) ottenendo: “aser”.

Temp=”aser” + “asdf” + “qwer” =”aserasdfqwer”

La prima parte del verme è MD5(temp), si supponga uguale a “zxcv”.

Da qui in poi viene applicato il metodo iterativo:

Temp=”zxcv” + “asdf” + “qwer” = “zxcvasdfqwer”

La seconda parte del verme è MD5(temp) = “mnbv”

Temp=”mnbv” + “asdf” + “qwer” = “mnbvasdfqwer”

La terza parte del verme è MD5(temp)= “wert”

E così via per tutto il messaggio

Il numero di combinazioni che bisogna effettuare per ricavare tutto il verme dipende dalla fase di inizializzazione. Essa a sua volta è basata su:

- 1) funzione MD5 sulla password;
- 2) stringa ausiliaria.

Per la funzione MD5 sulla password ci sono  $2^{128}$  combinazioni possibili e anche per la stringa ausiliaria, infatti quest’ultima è stata generata a partire da un’anagramma della password, quindi non conoscendo la password, non si può conoscere neppure il suo anagramma. In tutto si ottengono  $2^{128} * 2^{128} = 2^{256}$  combinazioni che è un numero molto grande e, come si vedrà nella sezione di crittoanalisi, molto difficile da forzare in maniera esaustiva.

La complessità temporale di questo strato è determinato dal seguente ciclo:

```

do
{
    x=0          ;
    while (x < BLOCK_LENGTH)
    {
        plainBlock [x] = getc ( sourceFile );
        if(feof(sourceFile)) //eventualmente aggiungere controllo di errore
        {
            len = x          ;
            filefinito = 1   ;
            break            ;
        }
        x = x + 1          ;
    }
    for( x=0; x<len; x++ )
    {

        crypted[x] = plainBlock[x] ^ MD5pwd[scorriMD5Pwd];
        scorriMD5Pwd++      ;
        if( scorriMD5Pwd == MD5_LENGTH )
        {
            for( contatore=0; contatore<MD5_LENGTH; contatore++ )
            {
                if( contatore % 2 == 0 )
                    MD5temp[contatore]=MD5pwd[contatore] ;
                if( contatore % 2 == 1 )
                    MD5temp[contatore]=MD5pwdUno[contatore] ;
            }
            MDString(MD5temp, MD5temp2 )          ;
            for( contatore=0; contatore<MD5_LENGTH; contatore++ )
            {
                MD5pwdUno[contatore]=MD5pwd[contatore] ;
            }
            for( contatore=0; contatore<MD5_LENGTH; contatore++ )
            {

```

```

        MD5pwd[contatore]=MD5temp2[contatore]    ;
        }
        scorriMD5Pwd = 0                        ;
    }
}
fwrite( crypted, len, 1, destinationFile )    ;
}

```

I punti cruciali di tale ciclo sono tre:

- Lettura del file
- Chiamata alla funzione MD5
- Scrittura su file

#### **Lettura del file**

Il file viene letto a blocchi, e ogni blocco riempito mediante una lettura carattere per carattere, con la funzione `getc(file)`. I blocchi hanno dimensione `BLOCK_LENGTH(=2048)`. Viene effettuata un'operazione di lettura per ogni carattere del file.

#### *Chiamata alla funzione MD5*

Essa è chiamata ogni volta che il verme deve essere allungato, quindi ogni 32 caratteri del file da cifrare (se ad esempio il file fosse lungo 1000 byte, verrebbe chiamato  $1000/32=32$  volte, approssimando per eccesso).

#### *Scrittura su file*

La scrittura su file avviene completamente a blocchi; ogni blocco ha lunghezza `BLOCK_LENGTH(=2048)`. Tale operazione viene eseguita un numero di volte pari alla dimensione del file / 2048 (se ad esempio il file fosse di 1000000 di byte, avverrebbero  $1000000/2048 \sim 489$  scritture su file).

Per aumentare le prestazioni bisognerebbe migliorare almeno uno dei tre suddetti fattori:

- scrittura e lettura dei file usando operazioni più veloci, magari anche di basso livello;
- uso di algoritmo di digest più veloce e/o in grado di fornire una stringa in uscita più lunga di MD5.

### **4.3 Crittanalisi**

E' analizzata nei paragrafi seguenti la sicurezza dei due livelli, sia singolarmente sia insieme.

#### **4.3.1 Sicurezza del Livello Polialfabetico**

Il livello Polialfabetico, anche preso singolarmente, non è facile da decifrare. Se l'attaccante disponesse del file cifrato, del sorgente, ma non del file in chiaro e del binario si comporterebbe nel seguente modo:

Conoscendo il sorgente, egli sa che il primo carattere del file indica la lunghezza della stringa con la quale è stato sostituito il carattere in chiaro. Basandosi su questo fatto potrebbe riuscire a trovare nel file stringhe che si ripetono più frequentemente, per poi riuscire a trovare tutte le stringhe usate per sostituire i caratteri. Va poi fatta un'analisi statistica sulle stringhe per trovare i caratteri giusti.

L'analisi statistica deve basarsi su enormi quantità di dati, molto maggiori rispetto a qualsiasi altro algoritmo basato sulla sostituzione, perché ogni carattere è sostituito di volta in volta con stringhe differenti.

Se l'attaccante non disponesse di grosse quantità di dati, allora non potrebbe riuscire a decifrare alcunchè.

Se l'attaccante disponesse del file cifrato, del file in chiaro relativo, del sorgente ma non fosse in possesso del binario allora questa volta il file in chiaro fornisce informazioni sul numero di caratteri presenti, quindi l'algoritmo per ricavare le stringhe risulterebbe leggermente più facile da implementare e la complessità dipenderebbe da come è stato generato il file cifrato. Se l'attaccante riuscisse a codificare tale algoritmo, ritroverebbe una parte di stringhe che sostituiscono i caratteri e sarebbe in grado di decifrare (almeno in parte) i file. Si sta comunque ipotizzando uno scenario un po' strano; file in chiaro e file cifrato contemporaneamente in possesso dell'attaccante, che è improbabile che si verifichi, a meno che l'attaccante non abbia "rubato" in qualche altro modo il file in chiaro. Si ricorda che anche se l'attaccante usasse un altro binario(caso in cui verrebbe tranquillamente in possesso di file in chiaro e cifrato), la sua cifratura sarebbe comunque diversa da quella dell'attaccato.

Ultima situazione che si va ad analizzare, è quella in cui l'attaccante si trovasse in possesso del file cifrato, del sorgente, del binario e non del file in chiaro. Disporre del binario dà all'attaccante un'arma potentissima, potrebbe infatti conoscere le stringhe attraverso le quali vengono sostituiti i caratteri. Si tratta di un'operazione comunque non facile da effettuare (dovrebbe "disassemblare" il binario), ma comunque possibile. In questo caso sarebbe relativamente facile trovare le stringhe nel file cifrato e costruire un algoritmo che decifra il file. Da qui si evince quello che sembra essere l'unico difetto del Polialfabetico: **dipende strettamente dal file binario, esso non può cadere in mano all'attaccante.**

Nonostante questi difetti, **ARCC**, considerato nella sua globalità, rimane comunque molto forte anche se il binario fosse nelle mani dell'attaccante, come si evince dal paragrafo 4.4.4 "Sicurezza Complessiva dell'Algoritmo".

### 4.3.2 Sicurezza dello Strato “XOR”

Lo strato “XOR” presenta il più alto livello di sicurezza.

Esso, al contrario del Polialfabetico, è completamente indipendente dal binario.

Se l’attaccante fosse in possesso del file cifrato, non del file in chiaro e conoscesse il sorgente, non ci sarebbero altri modi di decifrare il file se non per mezzo di un attacco esaustivo.

Gli altri tipi di attacco, tipo quello frequentistico, sarebbero inefficaci, essendo il “verme” lungo quanto il messaggio.

L’attacco di tipo esaustivo mira ad attuare tutte le combinazioni possibili per trovare la chiave.

Come visto nella sezione “*Approccio allo XOR*”, si dovrebbero scoprire la stringa ottenuta dalla funzione MD5, avente come ingresso la password inserita dall’utente, e la stringa ottenuta dalla funzione MD5 da un’anagramma della password; si arriva così ad un totale di  $2^{256}$  combinazioni possibili.

Per dare una dimensione più “concreta” a tale numero, supponiamo di avere  $10^9$  chip che ogni secondo riescono a scandire  $10^{10}$  chiavi. In questo caso essi impiegherebbero ben  $1,16 \cdot 10^{58}$  secondi, pari a circa  $10^{50}$  anni.

Se l’attaccante fosse in possesso sia del file in chiaro che del file cifrato, allora riuscirebbe a trovare il “verme”. Gli basterebbe infatti fare uno XOR tra i caratteri del file in chiaro e quelli del file cifrato, il risultato ottenuto è il “verme”. L’attaccante potrebbe usare tale “verme” per decifrare altri file lunghi al massimo quanto il file che possedeva, perchè non conosce in che modo si “allunga” il verme. Esso si “allunga” secondo una funzione<sup>28</sup> dipendente dalla password, quindi l’attaccante, non conoscendola, non può “allungare” il verme (a meno di un attacco esaustivo ovviamente).

Parliamo comunque di una situazione assurda, in quanto l’attaccante è difficile che si trovi “gratuitamente” in possesso del file in chiaro.

Una situazione più reale è il caso in cui l’attaccante conoscesse parte del file in chiaro. In tal caso l’attaccante potrebbe venire a conoscenza solo di una parte del verme, lunga quanto è lunga la sezione del file in chiaro conosciuto (questo però viene a crollare nell’ARCC grazie al livello Polialfabetico, vedi paragrafo successivo).

Il livello XOR potrebbe essere utilizzato eventualmente anche da solo, offrendo comunque un altissimo livello di sicurezza. L’importante è che l’attaccante non entri mai in possesso del file cifrato e del relativo file in chiaro. Nel caso in cui ciò accadesse, basterebbe cifrare gli altri messaggi con una password differente, in modo tale che l’attaccante non sia più in grado di decifrarli.

---

<sup>28</sup> La funzione usata dipende da una stringa data in ingresso in MD5 che dipende dalla password. MD5 può generare collisioni, ma sono poche e imprevedibili, quindi poco rilevanti.

### 4.3.3 Sicurezza Complessiva dell'Algoritmo

Si è già visto nelle sezioni precedenti che i due livelli di cifratura, se presi singolarmente, presentano alcuni difetti. Essi vengono a cadere quando i due livelli funzionano insieme.

In particolar modo, devono essere eseguiti per la fase di cifratura, prima il livello Polialfabetico e poi lo XOR, mentre per la fase di decifratura l'inverso.

Se fosse eseguito prima lo XOR e poi il Polialfabetico otterremmo che, nel momento in cui l'attaccante possedesse il binario potrebbe decifrare il livello Polialfabetico, rimarrebbe quindi solo la parte dello XOR che da sola presenta i difetti evidenziati nel paragrafo 4.4.3.

Se l'attaccante fosse in possesso del file cifrato, del sorgente, ma non del file in chiaro e del binario, allora gli sarebbe impossibile decifrare il file, a meno di un attacco esaustivo sullo XOR (analizzato nel paragrafo precedente) per poi decifrare molto più facilmente il solo strato Polialfabetico.

Qualora l'attaccante disponesse del file cifrato, del sorgente, del binario, ma non del file in chiaro, allora potrebbe venire a conoscenza delle stringhe usate per la sostituzione, ma non potrebbe sapere quali e come esse sono state usate, quindi alla fine bisognerebbe comunque condurre un attacco esaustivo al livello XOR, per poi risolvere molto facilmente il livello Polialfabetico.

Se egli avesse il file cifrato, il file in chiaro, il sorgente ma non il binario, non saprebbe per ogni carattere del file in chiaro, con quali stringhe è stato sostituito, e non riuscirebbe dunque a ricavare neppure informazioni utili sulla chiave dello XOR.

Se l'attaccante avesse il file cifrato, il file in chiaro, il sorgente ed il binario, potrebbe scrivere un algoritmo che a partire dal file in chiaro tenta tutte le combinazioni possibili applicando solo il livello Polialfabetico, compatibilmente al numero di caratteri finali del file, e poi farci lo XOR col file cifrato, ottenendo la chiave dello XOR. Questa è l'unica situazione in cui l'algoritmo appare vulnerabile, ma a questo si può sempre porre rimedio cambiando la password di cifratura.

Si rammenta al lettore che, se l'attaccante possedesse il binario, potrebbe attuare un attacco di tipo "forza bruta" alla password, per questo è stata inserita nel binario una seconda password senza la quale non è possibile eseguire il programma. In questo modo l'attacco è ancora più difficile da effettuare, perché il numero di combinazioni possibili è veramente elevato.

Al buon senso dell'utente è lasciata l'accortezza di usare password non troppo semplici e possibilmente differenti una dall'altra.

## 5 Confronti con altri Algoritmi

### 5.1 Confronti e Analisi delle Prestazioni

Gli algoritmi di cifratura odierni presentano delle velocità molto elevate, anche su dati in ingresso molto grandi.

Si sono testati gli algoritmi più importanti al momento, mediante il seguente meccanismo; sono stati generati dieci file di lunghezza rispettivamente 100 kb, 500 kb, 1000 kb, 2000 kb, 3000 kb, 5000 kb, 7000 kb, 9000 kb, 10000 kb e dati in pasto agli algoritmi:

(Block Cipher)

- DES
- Blowfish
- AES (Rijndael)

(Stream Cipher)

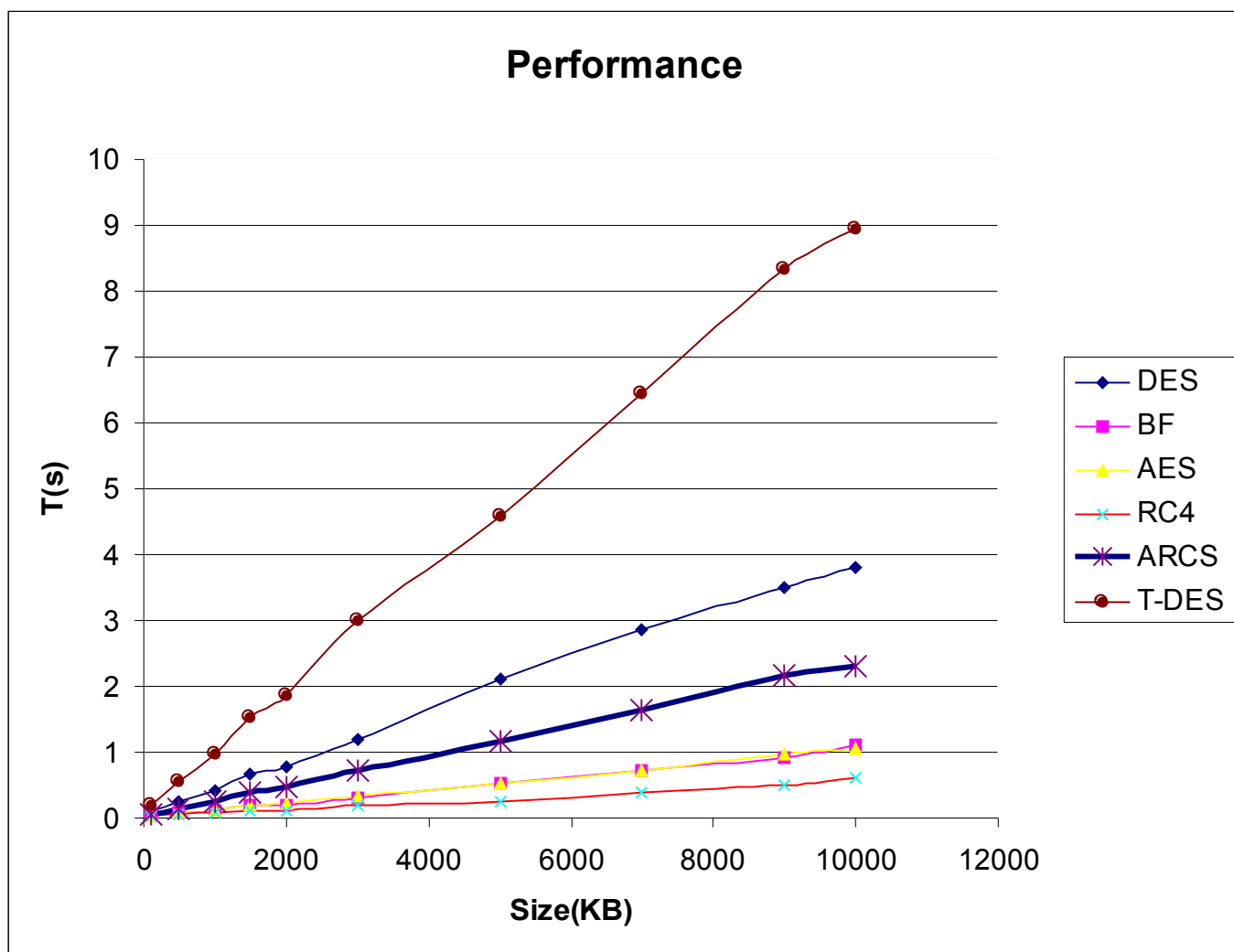
- RC4

Dalla cifratura dei file di prova si sono ottenuti i risultati riportati nella tabella. A seconda della dimensione dei file (da 100 kB a 10000 kB) sono stati trascritti i relativi tempi di esecuzione e alla fine è stato calcolato il throughput di ogni cifrario.

KB	DES	BF	AES	RC4	ARCS	T-DES
100	0,1	0,044	0,03	0,03	<b>0,053</b>	0,185
500	0,25	0,08	0,07	0,05	<b>0,15</b>	0,558
1000	0,42	0,115	0,12	0,075	<b>0,25</b>	0,975
1500	0,67	0,195	0,19	0,125	<b>0,4</b>	1,533
2000	0,773	0,2	0,21	0,12	<b>0,47</b>	1,865
3000	1,193	0,315	0,33	0,195	<b>0,72</b>	3,01
5000	2,1	0,52	0,52	0,26	<b>1,16</b>	4,59
7000	2,873	0,72	0,73	0,38	<b>1,63</b>	6,455
9000	3,512	0,915	0,96	0,505	<b>2,16</b>	8,33
10000	3,8	1,1	1,05	0,6	<b>2,3</b>	8,95
Throughput	2273,34	8269,908	8016,69	13966,14	<b>3666,025</b>	934,9895

**Tabella 6:** Risultati sperimentali del test di cifratura

Da cui si evince il grafico qui sotto.



**Figura 2:** Grafico delle prestazioni dei cifrari analizzati

Dal grafico e dai dati si denota che tutti gli algoritmi presentano un andamento pressocchè lineare, che aumenta proporzionalmente alla dimensione del file di ingresso.

L'algoritmo che risulta essere più lento è il DES, comunque ormai considerato obsoleto.

AES e Blowfish hanno un andamento quasi identico, circa il 25% del DES.

RC4 presenta i tempi migliori fra i quattro, circa il 50% di AES e Blowfish e il 12%-13% di DES.

ARCS ha dei risultati migliori del DES, ma comunque per adesso ancora più lento (il doppio circa) di AES, Blowfish ed RC4.

Il grafico sopra riportato tiene conto solamente della velocità dei singoli algoritmi, ma non tiene conto di un altro fattore importantissimo: la lunghezza della chiave.

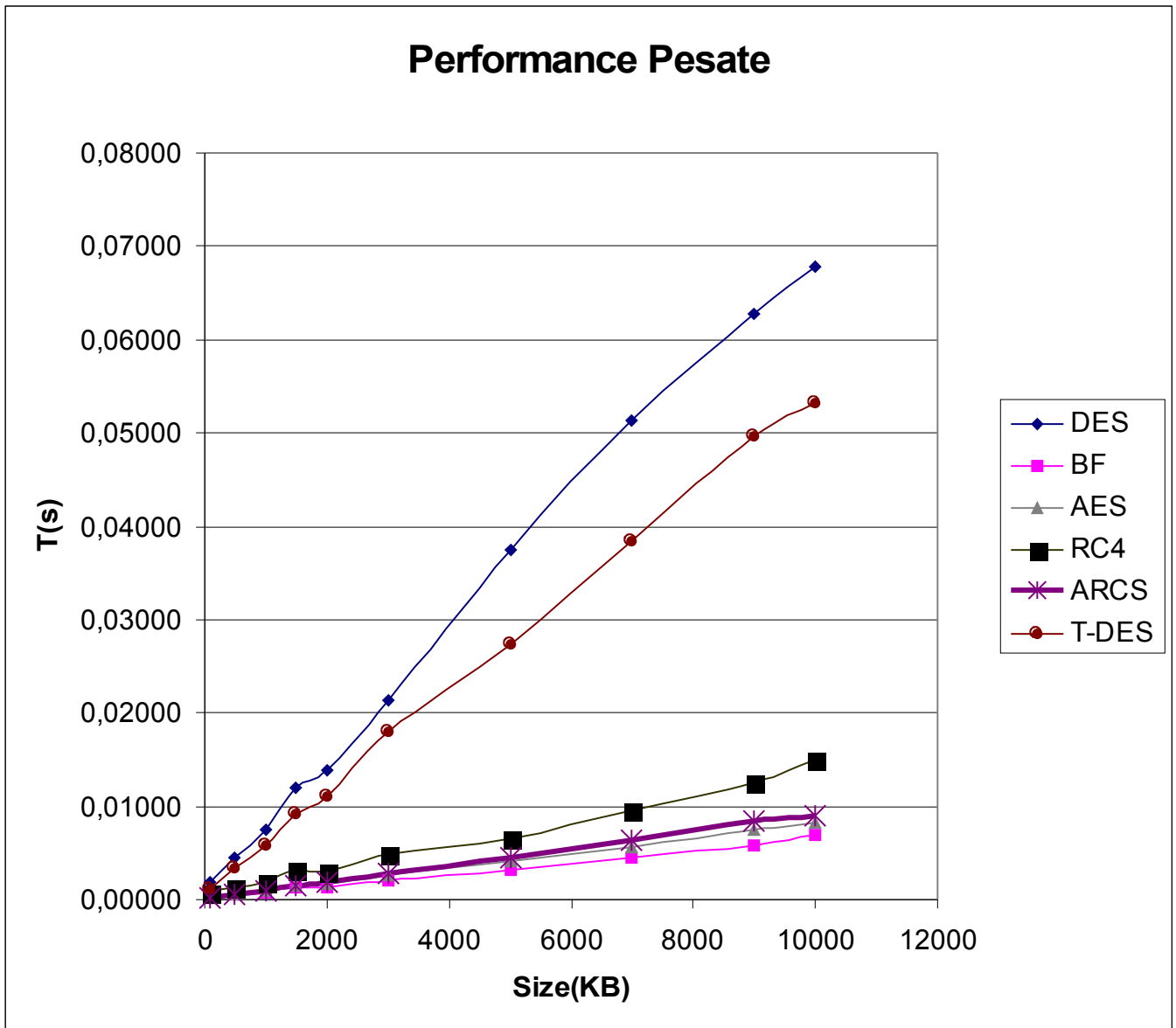
E' importante, affinché l'analisi sia il più accurata possibile, che l'indice delle prestazioni non sia funzione solo del tempo(come nel grafico precedente), ma anche della lunghezza della chiave.

Il criterio adottato consta nel dividere i tempi per la lunghezza della chiave, ottenendo i dati riportati nella seguente tabella.

KB	DES	BF	AES	RC4	ARCS	T-DES
100	0,00179	0,00028	0,00023	0,00075	<b>0,00021</b>	0,00110
500	0,00446	0,00050	0,00055	0,00125	<b>0,00059</b>	0,00332
1000	0,00750	0,00072	0,00094	0,00188	<b>0,00098</b>	0,00580
1500	0,01196	0,00122	0,00148	0,00313	<b>0,00156</b>	0,00913
2000	0,01380	0,00125	0,00164	0,00300	<b>0,00184</b>	0,01110
3000	0,02130	0,00197	0,00258	0,00488	<b>0,00281</b>	0,01792
5000	0,03750	0,00325	0,00406	0,00650	<b>0,00453</b>	0,02732
7000	0,05130	0,00450	0,00570	0,00950	<b>0,00637</b>	0,03842
9000	0,06271	0,00572	0,00750	0,01263	<b>0,00844</b>	0,04958
10000	0,06786	0,00688	0,00820	0,01500	<b>0,00898</b>	0,05327
	bit chiave	bit chiave	bit chiave	bit chiave	<b>bit chiave</b>	bit chiave
	56	160	128	40	<b>256</b>	168

**Tabella 7:** Risultati sperimentali del test di cifratura "pesati" con la lunghezza della chiave

Da qui se ne trae il grafico sotto riportato.



**Figura 3:** Grafico delle prestazioni dei cifrari analizzati “pesati” con la lunghezza della chiave

Dal grafico si può notare come ARCS recuperi in confronto agli altri cifrari, soprattutto rispetto al DES, che ha una chiave di soli 56 bit, e RC4, che nella versione utilizzata nella simulazione, aveva una chiave di 40 bit.

Per le simulazioni è stato usato un Pentium 1700 Mhz con 128 MB di RAM(DDR) e i programmi sono stati eseguiti con il simulatore linux per windows “cygwin”.

## **6 Conclusioni e sviluppi futuri**

**ARCC**, acronimo di “Angelo Rosiello & Roberto Carrozzo Cryptography”, è nato nell’estate del 2002 con lo scopo di poter cifrare dati riservati molto importanti, come il numero di carta di credito o documenti di estrema delicatezza. La prima versione, come già detto, era un semplice algoritmo di sostituzione polialfabetica, nel corso del tempo completamente rivoluzionato, fino a diventare quasi del tutto indipendente dal suddetto strato.

L’idealizzazione dell’algoritmo e lo studio crittanalitico, sono risultati in fin dei conti la parte più complessa della sua intera elaborazione.

Un algoritmo di crittografia che si rispetti deve garantire, nei tempi che corrono, un livello molto alto di sicurezza e non può essere vulnerabile ad attacchi che non siano di tipo esaustivo. Per dirla in parole povere, un cifrario deve essere quanto più forte e robusto possibile.

Una stima sui tempi per un attacco di tipo esaustivo ad **ARCC**, evidenzia che occorrono all'incirca  $10^{40}$  anni per forzare l'algoritmo, ovviamente senza considerare il livello Polialfabetico.

Lo strato Polialfabetico aggiunge un'incredibile forza al cifrario che, grazie alla sicurezza aggiunta dalla singolarità del binario, potrebbe essere considerato addirittura quasi "indecifrabile", perfino agli occhi di un attacco esaustivo. Oltre ai pregi, il suddetto strato aggiunge però anche dei difetti, quali l'accrescimento della dimensione dei file e la lentezza nelle operazioni di cifratura e decifratura. Tuttavia, grazie alla forza dello strato XOR, è possibile fare a meno del livello Polialfabetico aumentando esponenzialmente le prestazioni, pur conservando un altissimo livello di sicurezza. **ARCC** se privato del livello Polialfabetico, è uno stream-cipher in piena regola, da noi "battezzato" **ARCS**, proprio come RC4, che diventa in questo momento, il principale cifrario concorrente. Unico obiettivo futuro diventa dunque quello di cercare di battere in velocità RC4, impresa al quanto ardua, considerata la sua rapidità.

Ecco alcuni dati indicativi che modellizzano la situazione.

KB	RC4	ARCS	T-DES
100	0,03	<b>0,053</b>	0,185
500	0,05	<b>0,15</b>	0,558
1000	0,075	<b>0,25</b>	0,975
1500	0,125	<b>0,4</b>	1,533
2000	0,12	<b>0,47</b>	1,865
3000	0,195	<b>0,72</b>	3,01
5000	0,26	<b>1,16</b>	4,59
7000	0,38	<b>1,63</b>	6,455
9000	0,505	<b>2,16</b>	8,33
10000	0,6	<b>2,3</b>	8,95

**Tabella 3:** Risultati sperimentali del test di cifratura

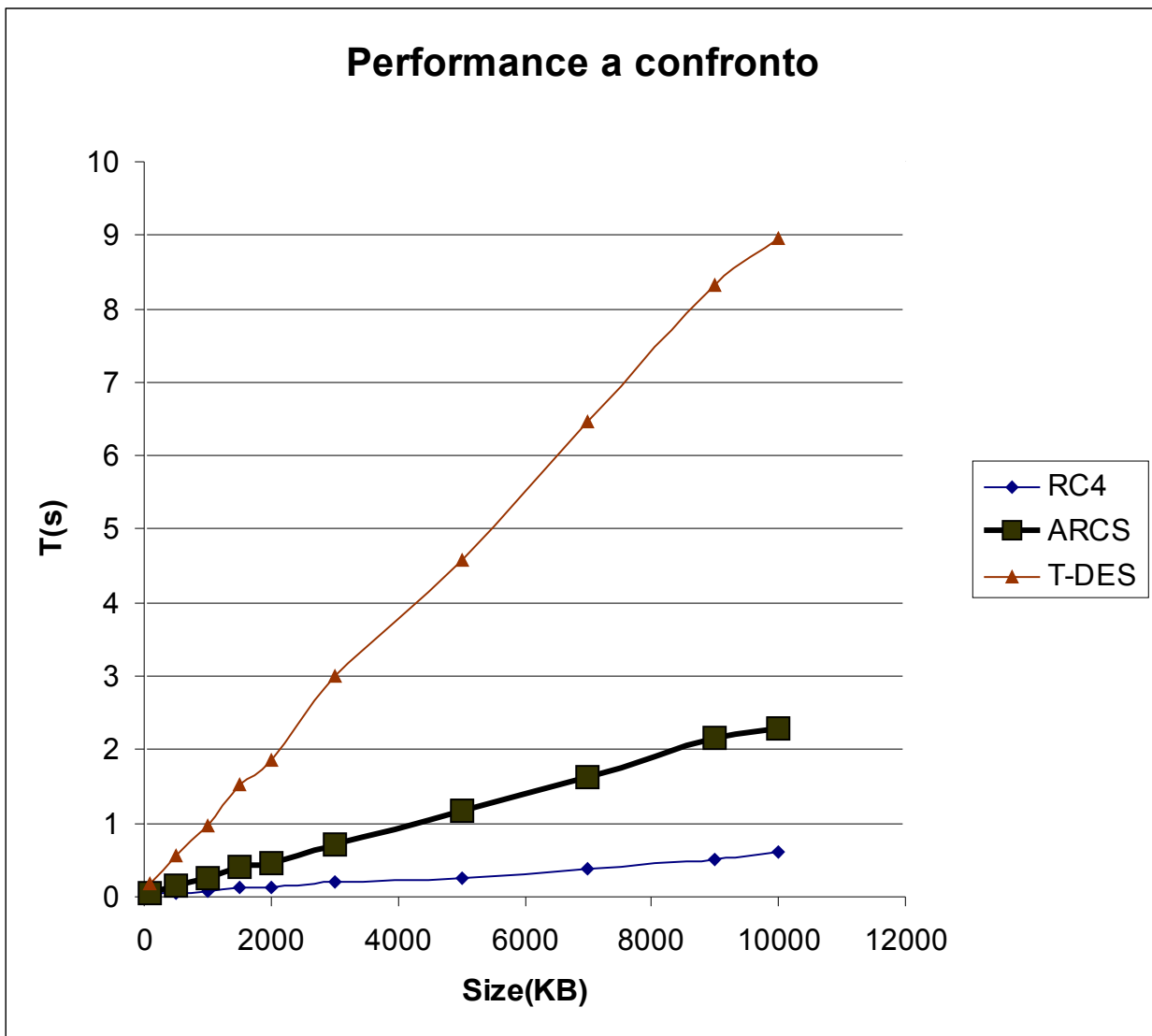


Figura 3: Grafico delle prestazioni dei cifrari analizzati

Si evince dal grafico come ARCC nella sua versione stream cipher (ARCS) risulta essere una soluzione intermedia tra Triple-DES ed RC4.

Bisogna altresì tenere presente che in questo caso rappresenta la migliore soluzione a livello di sicurezza. Triple-DES fa uso di una chiave di 168 bit, la versione di RC4 considerata invece di una chiave di 40 bit, mentre ARCS ha una chiave di ben 256 bit.

La chiave di  $2^{256}$  bit, questo rappresenta un ottimo risultato, ma può essere ulteriormente migliorato o modificato. La chiave viene generata dall'unione della password "trasformata" con la funzione di hashing MD5 e di un suo anagramma a cui è ancora applicata la stessa funzione.

MD5 dà in uscita una stringa di 16 caratteri, per un totale di  $256^{16}(=2^{128})$  combinazioni disponibili. Se fosse messo a disposizione un algoritmo di digest migliore, che ritornasse una stringa di  $x > 16$  caratteri, in minor tempo, avremmo  $256^x$  combinazioni differenti, senz'altro maggiori di  $256^{16}$ , ottenendo così una chiave più lunga. E' stato provato un altro algoritmo di

digest molto famoso, SHA-1, ma, nonostante ritornasse in uscita una stringa di 20 caratteri, più di MD5, è risultato essere troppo lento.

Altri modi per espandere la chiave potrebbero essere realizzati effettuando differenti anagrammi della password, a cui applicare la funzione di hashing. In questo modo è possibile allungare la chiave di 128 bit per anagramma.

In alcuni casi, quando ad esempio bisogna lavorare con dei buffer di lunghezza limitata, la chiave potrebbe anche essere accorciata, senza che ciò vada ad influenzare troppo il livello di sicurezza, considerando che di questi tempi anche una chiave di 128 bit può essere considerata più che accettabile.

Per velocizzare ARCS, si potrebbe usare un algoritmo di digest più veloce. Un esempio potrebbe essere MD versione 4, molto più veloce del suo successivo, ma purtroppo con qualche bug risolto poi in MD 5, ma ha scapito del tempo di esecuzione.

Questo meccanismo renderebbe la vita più semplice a chi ha a che fare, per lavoro o altro, con dati importanti da tenere riservati.

Implementare un buon algoritmo di cifratura richiede tempo e molta pazienza; più di qualche volta è necessario cancellare tutto e ricominciare da zero, sempre se ne resta la voglia e la lucidità necessaria; ma questi sono i rischi a cui va incontro un crittologo.

Grazie agli imprevisti riscontrati durante l'implementazione di **ARCC**, siamo certamente riusciti a comprendere a fondo gli obiettivi e le metodologie utilizzate dai cifrari codificati nel corso della storia, apprendendone pregi e difetti, a cui potersi ispirare durante la stesura di un nuovo algoritmo.

La riservatezza dei dati e la sicurezza delle trasmissioni rappresentano il futuro ed **ARCC**, nella versione stream, ha ottime prospettive visto che ambisce a contrastare, in un singolo anno di vita, l'esclusività di RC4, un cifrario che ha ormai un'esperienza più che decennale.

Il messaggio che gli autori vogliono trasmettere non ha la pretesa, né la sfrontatezza di "convincere" il lettore circa la "perfezione" dell'algoritmo anzi, una delle peculiarità di **ARCC** è quella di essere "free-ware" ed "open source", ossia aperto e disponibile per tutti.

Non è altruista tenere riservato qualcosa che potrebbe giovare alla società e per dirla tutta non è nemmeno, ingegneristicamente parlando, la scelta ottima. I software crittografici, per essere sicuri devono essere aperti, dando ai crittanalisti la possibilità di testare tutti i possibili attacchi effettuabili al cifrario.

Speriamo vivamente che le analisi effettuate non abbiano annoiato il lettore, bensì stimolato l'interesse per la crittologia nella sua interezza.

## Bibliografia

1. L.Sacco, *Manuale di crittografia*, Roma, 1947.
2. Sito ufficiale del NIST, Home page <http://csrc.nist.gov>
3. Sito ufficiale dell'università olandese "Katholieke Universiteit Leuven", Home page <http://www.esat.kuleuven.ac.be/index.en.shtml>
4. Crypto: sito informativo su Crittografia e altre forme di comunicazione sicura, Home page <http://alt255.virtualave.net>
5. Articolo su Blowfish, Home page <http://r.empstudios.com>
6. MediaMacros Inc., Home page <http://www.mediamacros.com>
7. Riksoft - Software-Protezione-Servizi, Home page <http://www.riksoft.com>
8. Sito ufficiale di RSA Security, Home page <http://www.rsasecurity.com>
9. Rubberhose cryptographically deniable transparent disk encryption system, Home page <http://www.rubberhose.org>
10. Bruce Schneier, *Applied Cryptography : Protocols, Algorithms, and Source Code in C*, 2nd Edition, 1997
11. Appunti di Crittografia, Home page <http://www.rcvr.org>
12. EnricoZimuel.net - crittografia e crittoanalisi: teorie, algoritmi ed applicazioni, Home page <http://www.enricozimuel.net>