

I cifrari simmetrici possono essere di 2 categorie:

- **Cifrari a blocchi:** permettono la trasformazione di un blocco di testo in chiaro in un blocco di testo cifrato, più sicuri e più utilizzati.
- **Stream Cipher:** trasformazione, dipendente dallo spazio/tempo, di singoli caratteri del testo in chiaro. Le modalità di cifratura qui non esistono.

5.0 STREAM CIPHER

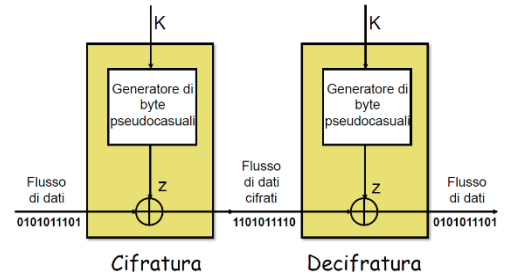
Testo in chiaro	M_0	M_1	M_2	M_3	...	M_i	...
Keystream	z_0	z_1	z_2	z_3	...	z_i	...
Testo cifrato	C_0	C_1	C_2	C_3	...	C_i	...

Su alfabeto binario: $C_i = M_i \text{ XOR } z_i$

Dato un testo in chiaro che viene diviso in tanti bit, viene generata una Keystream dallo Stream Cipher mediante una generazione casuale. La Keystream viene utilizzata per cifrare il testo in chiaro facendo lo XOR bit a bit. Questa è una tecnica simile a quella del cifrario perfetto, che usa come modalità di cifratura "output feedback mode". Nello Stream Cipher lo step più importante è quello della generazione della Keystream, nella maniera più opportuna e veloce possibile.

L'idea descritta in precedenza può essere mostrata in maniera più tecnica:

Il generatore di bit pseudocasuali corrisponde allo Stream Cipher, produce una chiave z che fa lo XOR bit a bit con il flusso in chiaro dato in input. Una volta ottenuta la chiave cifrata, per il processo di decifrazione, si genera esattamente nello stesso modo la sequenza di bit pseudocasuali e si fa l'operazione inversa della cifratura, che in questo caso è lo XOR (ricorda, l'inverso dello XOR è lo XOR stesso).



La domanda quindi è come si fa a produrre questa chiave pseudocasuale, detta anche Stream Cipher? In generale l'operazione di Stream Cipher comprende poche linee di codice che vengono ripetute più volte, le operazioni sono semplici ed efficienti sulle varie architetture. Poiché la chiave è piccola e l'output del generatore è molto lungo, quest'ultimo comincerà a ciclare, dunque questo è il periodo dell'output. È importante che il periodo sia lungo, poiché sarà più difficile effettuare la crittoanalisi. Il generatore pseudocasuali ha 2 caratteristiche:

- Dovrebbe avere lo stesso numero di 0 e 1
- Se vista come sequenza di byte, ciascuno dei 256 valori possibili dovrebbe apparire lo stesso numero di volte.

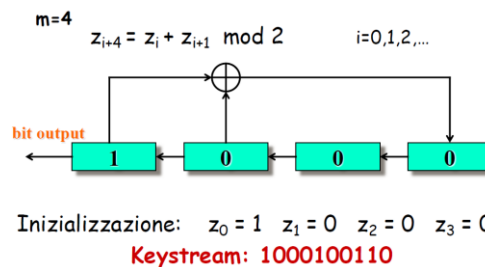
Verranno descritti vari Stream Cipher, quali: **LSFR (Linear Feedback Shift Register)**, **A5(e GSM)**, **RC4**, **Salsa20**, **ChaCha20**.

5.0.1 LINEAR FEEDBACK SHIFT REGISTER (LSFR)

Come suggerisce il nome, il processo si basa su registri lineari a shift con feedback.

Essenzialmente l'idea è che questi registri vengono descritti con un'equazione ricorrente di un certo grado. Ad esempio: $z_{i+m} = c_0 z_i + c_1 z_{i+1} + \dots + c_{m-1} z_{i+m-1} \text{ mod } 2$ $i=0,1,2,\dots$ in cui il valore di un certo bit nella sequenza, in questo esempio z_{i+m} , è una combinazione lineare dei precedenti m bit. I valori c_0, c_1, \dots sono dei coefficienti fissati con valore 0-1, l'operazione viene effettuata in modulo 2. Per inizializzare un registro di questo tipo ho bisogno ho chiaramente bisogno degli m bit iniziali. L'implementazione hardware risulta essere efficiente.

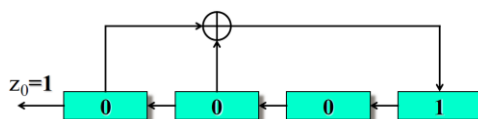
Vediamo con un esempio pratico il suo funzionamento:



L'equazione è fissata ed è $z_{i+4} = z_i + z_{i+1} \text{ mod } 2$. Questo è un registro in cui ogni singolo valore dipende dai 4 bit precedenti. Avendo gli m bit precedenti, posso calcolare il corrente. Questo è un particolare caso dell'equazione che è stata descritta all'inizio del paragrafo, poiché erano presenti i coefficienti c_0, c_1 che sono dei bit. In questo caso z_i e z_{i+1} hanno coefficiente 1, mentre z_{i+2} e z_{i+3} hanno coefficiente 0 e non sono stati dunque messi nell'equazione.

Con $m=4$ ho bisogno di 4 bit per inizializzare la sequenza dei valori z .

Si supponga l'inizializzazione con i valori illustrati in foto: $z_0=1$; $z_1=0$; $z_2=0$; $z_3=0$. Avendo fissato questi valori posso costruire i z successivi, che possono essere calcolati mediante il registro a scorrimento. Posiziono i valori di z all'interno di 4 registri. Al tempo iniziale la sequenza di 4 bit, calcola il valore successivo con uno XOR tra il primo registro da sinistra e il secondo e contemporaneamente i 4 registri effettuano uno shift a sinistra in modo tale che l'1 va in output, i 3 zeri shiftano a sinistra, in modo tale che l'ultimo registro sia disponibile ad accettare il risultato dello XOR (tutto quello che è stato appena spiegato viene applicato sulla figura sopra. La risultante appare nella figura qui sotto).



Le iterazioni successive seguono lo stesso ragionamento proposto in precedenza, e più in generale, ciclare all'infinito.

Da questo ragionamento deriva il nome shift, perché i registri shiftano a sinistra, c'è un feedback perché il valore dello XOR viene calcolato e rientra nell'ultimo registro, ed è lineare in quanto l'equazione che definisce questo registro è lineare.

Una domanda che sorge spontanea è perché solo i primo 2 registri da sinistra effettuano lo XOR, questo perché corrispondono a z_i e z_{i+1} . Più in generale, a seconda della relazione di ricorrenza, vengono stabiliti gli input allo XOR.

Se ciclassi infinitamente, ad un certo punto, la sequenza in output diventa periodica, in questo caso, è una Keystream di periodo 15: 100010011010111... significa che dopo 15 bit, i successivi sono uguali ai 15 precedenti, perché cicla. In questo caso avendo 4 registri, si hanno 2^4 possibilità, cioè 16 possibilità. Abbiamo però detto in precedenza che la Keystream di periodo è 15, nonostante 2^4 sia 16, perché chiaramente non si tiene conto della sequenza di bit 0000 poiché si rimarrebbe su 4 zeri senza possibilità di cambiare.

Il periodo non è detto che debba essere sempre 15, che rappresenta il massimale, sarà in genere comunque un suo divisore. Questo dipende dal polinomio associato alla relazione. Nel nostro esempio il polinomio associato è: $P(X) = x^3 + x + 1$, il periodo è massimale se vengono soddisfatte alcune proprietà del polinomio che non vengono attualmente trattate.

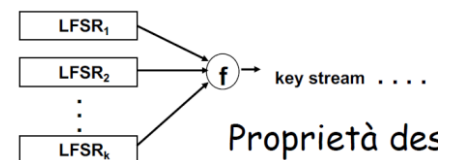
Riguardo la crittoanalisi, generalmente tutto ciò che è lineare, prima o poi verrà rotto. Assumiamo un attacco di tipo *Known Plaintext Attack* in cui l'avversario conosce il testo in chiaro: $M_0...M_n$. Conoscendo il testo cifrato: $y_0...y_n$, può calcolare la sequenza in output del registro a scorrimento $z_0...z_n$, naturalmente può farlo poiché il valore $z_i = M_i \oplus Y_i$. Avendo una Keystream abbastanza lunga, si possono ottenere i valori z , poiché se andassi a scrivere l'equazione precedente in forma matriciale:

$$\begin{bmatrix} z_{m+1} & z_{m+2} & \dots & z_{2m} \end{bmatrix} = \begin{bmatrix} c_0 & c_1 & \dots & c_{m-1} \end{bmatrix} \begin{bmatrix} z_1 & z_2 & \dots & z_m \\ z_2 & z_3 & \dots & z_{m+1} \\ \dots & \dots & \dots & \dots \\ z_m & z_{m+1} & \dots & z_{2m-1} \end{bmatrix}$$

Ad esempio il valore z_{m+1} lo ottengo moltiplicando i coefficienti per la prima colonna, ossia gli m elementi che precedono $m+1$ e così via tutti gli altri. Una volta scritta in forma matriciale, i valori z posso essere ottenuti. Nel caso in cui non si ha conoscenza dei valori c , si può risolvere considerando la matrice inversa dei z .

$$\begin{bmatrix} z_{m+1} & z_{m+2} & \dots & z_{2m} \end{bmatrix} \begin{bmatrix} z_1 & z_2 & \dots & z_m \\ z_2 & z_3 & \dots & z_{m+1} \\ \dots & \dots & \dots & \dots \\ z_m & z_{m+1} & \dots & z_{2m-1} \end{bmatrix}^{-1} = \begin{bmatrix} c_0 & c_1 & \dots & c_{m-1} \end{bmatrix}$$

Abbiamo visto il concetto generale degli LFSR, ma in realtà non vengono mai utilizzati da soli, bensì considerando diversi registri con diverse lunghezze, e gli output di questi ultimi, vengono combinati in maniera non lineare tra di loro per ottenere la key stream. Avendo un elemento non lineare, il periodo lungo risulta essere più alto. Con questo sistema, si superano le problematiche precedenti.



Un esempio di questo sistema appena descritto, è presente nell'algoritmo **A5**, legato al GSM, che è un vecchio standard legato alla telefonia mobile.

5.0.2 A5

Il seguente algoritmo si trova nella comunicazione tra la parte mobile, i nostri cellulari, e una Base station.



Analizziamo nel dettaglio la configurazione del sistema GSM:

MS: Sistema radio dotato di SIM.

BTS: Stazione di base con cui la MS comunica

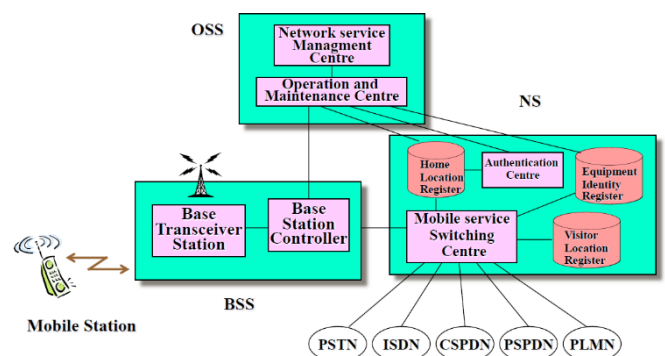
BSC: nodo comune tra le varie BTS

NS: Sottosistema di rete, realizza la connessione tra l'utente della rete mobile e gli utenti delle altre reti, che comprende il **centro di autenticazione**, fondamentale per poter decidere se si può effettuare la telefonata, o no.

OSS: Sottosistema di esercizio e manutenzione, sovrintende al corretto funzionamento della rete.

MSC: Nodo che controlla tutte le BSC in una zona e fornisce la connessione con le reti fisse.

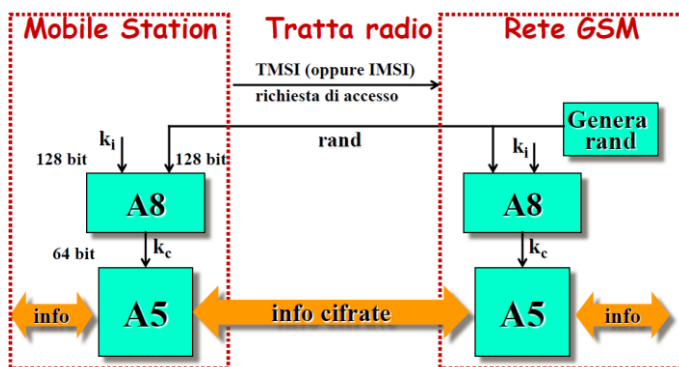
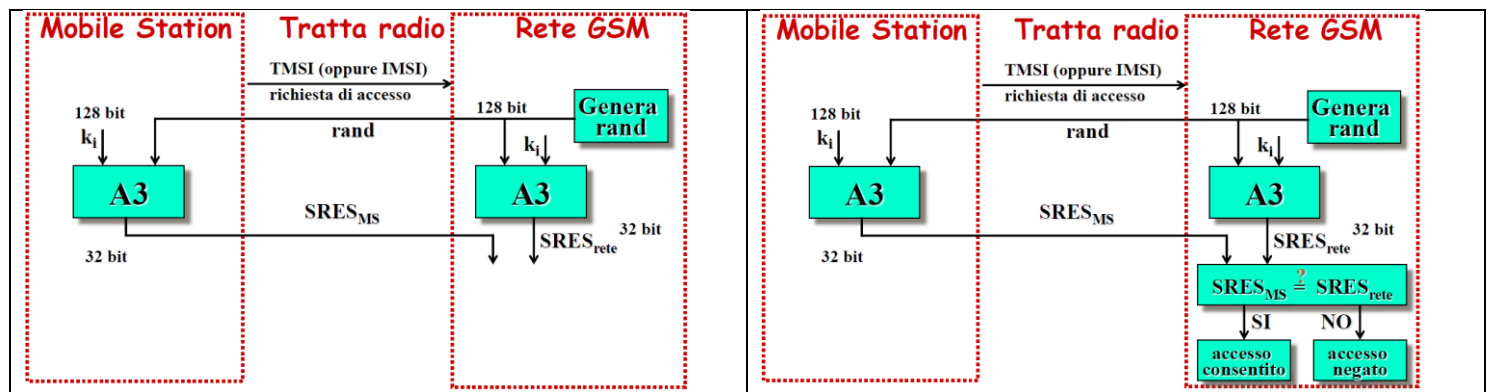
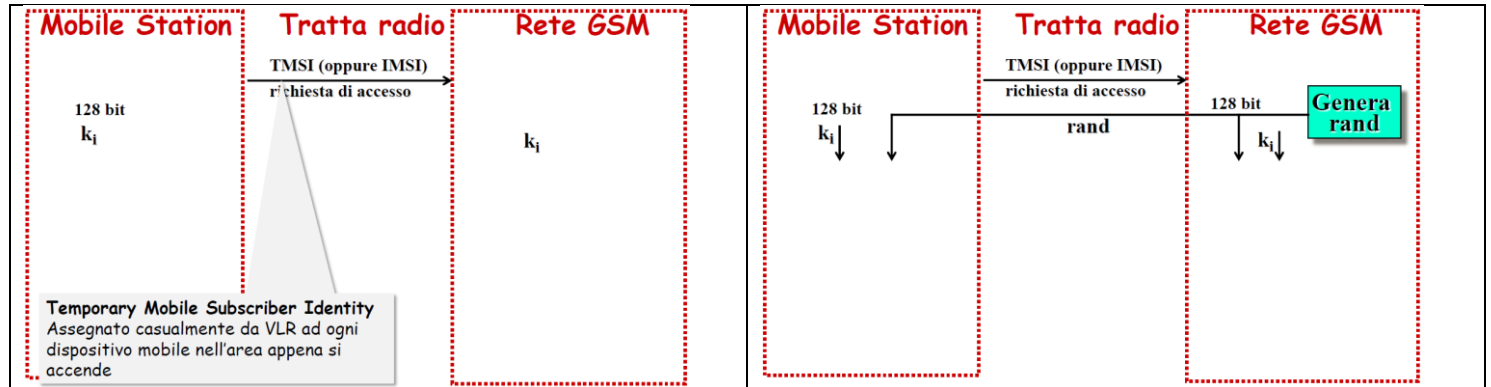
VLR: Database temporaneo contenente info sulle MS in transito della zona, per riconoscere a chi reindirizzare la telefonata e a dove si trova l'utente.



Dal dispositivo mobile, alla stazione base (BTS), la comunicazione è cifrata. Poi, tutta la parte descritta nell'OSS e NS è la parte interna al provider, in cui la comunicazione diventa in chiaro, e ci sarà una parte che ha il compito di smistarla agli altri provider, questo accade nel caso in cui un utente decide di chiamare un altro utente che ha un operatore telefonico differente, quest'ultimo utente nel momento in cui risponderà, invierà un segnale ad un altro segnale mobile. Poiché la struttura interna è in chiaro, ad esempio un giudice può chiedere all'operatore di risalire a determinate conversazioni che avvengono sul cellulare, chiaramente può farla, a meno di cifrature end-to-end, poiché in questo caso, il gestore non può decifrare la conversazione. Dell'A5 esistono varie versioni ed è stato scoperto l'algoritmo che permettesse l'autenticazione e la cifratura. Il modo con cui funziona è il seguente: ci sono 3 registri a scorrimento lineare, di grado 19,22,23 che venivano tra loro combinati in maniera non lineare. La chiave utilizzata per questo sistema è memorizzata nella SIM presente nel telefono, usata per autenticazione e decifratura.

Una SIM si compone un PIN, IMSI (International mobile subscriber identification) e una Chiave k_i di 128 bit, utilizzata per l'autenticazione e la cifratura.

L'autenticazione è uno step utilizzato per accertarsi che sia realmente l'utente che ha sottoscritto un abbonamento ad un operatore telefonico, ad effettuare un'operazione di chiamata. L'autenticazione avviene in contemporanea alla cifratura ed è descritta in questo modo:



Nella prima didascalia viene divisa nella parte sinistra il dispositivo mobile, e nella parte di destra la Rete GSM col suo punto d'accesso che è la BTS. La prima cosa che fa la stazione mobile è inviare una richiesta d'accesso o con l'IMSI o col TMSI (valore temporaneo associato casualmente nel momento in cui si entra in contatto ad una stazione BTS). La Rete GSM condivide con la stazione mobile una chiave k_i per autenticarlo, come si osserva nella seconda didascalia, genera un valore casuale di 128 bit e lo manda alla stazione mobile e chiaramente lo conserva per sé. Si nota molto semplicemente che entrambi hanno le stesse conoscenze: la stessa chiave privata e lo stesso valore di 128 bit generato casualmente.

Nella terza didascalia, la stazione mobile usa un algoritmo (A3) che prende in input la chiave privata e il numero random, generando un output a 32 bit detto $SRES_{MS}$ e lo invia al GSM. La Rete GSM esegue la stessa operazione fatta dalla stazione mobile, avendo in input gli stessi valori e di conseguenza oltre a $SRES_{MS}$ si trova anche con $SRES_{rete}$ elaborato al suo interno. Con questi 2 valori, per sapere se la stazione mobile è corretta o meno, deve verificare l'uguaglianza di questi 2 valori, come mostrato dalla quarta didascalia. Se i valori sono uguali, l'accesso è consentito, altrimenti l'accesso è negato. Si osservi che naturalmente la Rete GSM conosce a priori la chiave delle stazioni mobili in quanto fornitrice.

Tornando a ritroso, se la Rete GSM non generasse ogni volta un valore random, bensì fornisse sempre lo stesso valore, un protocollo del genere naturalmente non sarebbe buono, in quanto un mal intenzionato risale facilmente al valore inviato dalla Rete GSM. Poiché il valore è random, probabilità di risalire al valore precedente è 2^{128} , una probabilità impossibile.

Una volta effettuato l'accesso, come mostrato nella quinta e ultima figura, avviene la cifratura con l'algoritmo A5, che prende in input una chiave di sessione k_c a 64 bit, generata casualmente mediante l'algoritmo A8 che prende in input la chiave di 128 bit e il valore random dato dalla Rete GSM di

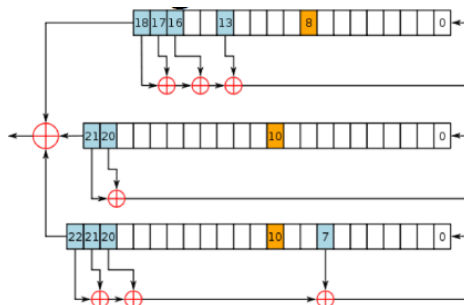
volta in volta in maniera sincronizzata tra le due parti. Si osserva quindi che per le informazioni cifrate viene generata la chiave di sessione k_c , mentre la Master-Key di 128 bit viene usata per generare la k_c .

Veniamo ora all'effettivo algoritmo A5: viene utilizzato con 3 polinomi a scorrimento e sono:

- $x^{19} + x^5 + x^2 + x + 1$
- $x^{22} + x + 1$
- $x^{23} + x^{15} + x^2 + x + 1$

e vengono utilizzati, in maniera non lineare per la cifratura.

La combinazione avviene nel seguente modo:



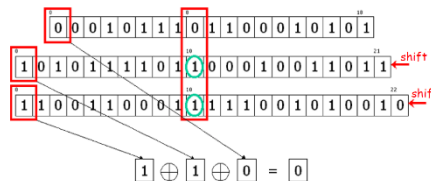
I 3 registri si trovano come si vede nella figura con i rispettivi gradi. Per combinarli tra di loro, una volta prodotto l'output, che si calcola come XOR dei 3 output, si deve considerare l'elemento non lineare che viene identificato al bit centrale di ogni registro che è stato colorato di arancione per visualizzarlo meglio. Si osservano i valori che si trovano in questi 3 registri e si vede il bit di maggioranza, per esempio:

- Se i tre bit sono 0,1,1, il bit di maggioranza è 1 e i 2 registri contenenti l'1 effettuano lo shift, chi contiene lo 0 sta fermo.
- Se i tre bit sono 0,1,0, il bit di maggioranza è 0 e i 2 registri contenenti lo 0 effettuano lo shift, chi contiene l'1 sta fermo.

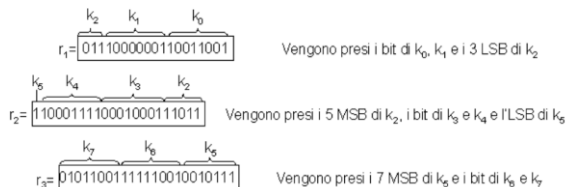
Almeno due registri vengono shiftati ad ogni round. L'output poi è il valore che viene utilizzato come la Key-Stream e viene utilizzato facendo lo XOR con i bit del testo in chiaro.

Questo è il modo in cui vengono combinati i 3 registri e sono facilmente implementabili in hardware.

Viene mostrato un ulteriore esempio riguardante l'elemento non lineare:



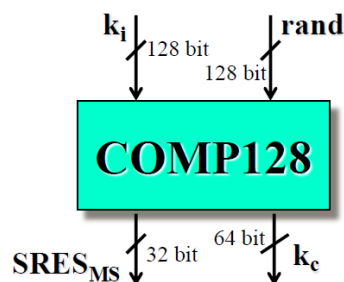
L'inizializzazione dei registri avviene nel seguente modo: la chiave k_c è di 64 bit e viene divisa in 8 byte (k_0, k_1, \dots, k_7) che vengono messi nei registri nel seguente modo:



Successivamente i registri a scorrimento subiscono delle modifiche in base alcuni valori che dipendono dal frame di riferimento.

L'algoritmo A3 viene utilizzato per l'autenticazione, A8 per il calcolo della chiave di sessione. In realtà, in diverse implementazioni si ha un algoritmo che sostituisce A3 e A8 che si chiama **COMP128** che prende in input la chiave k_i a 128 bit, il valore casuale mandato dalla BTS a 128 bit, utilizzato per autenticazione e cifratura. Questi due valori una volta computati nel COMP128 daranno vita a 2 output: un valore $SRES_{MS}$ usato per l'autenticazione e la chiave di sessione a 64 bit usata per la cifratura. Il COMP128 combina la fase di autenticazione e cifratura mediante un solo valore random.

Abbiamo detto che la chiave k_c è una chiave di sessione a 64 bit, di cui però si deve precisare che gli ultimi 10 bit sono tutti 0. In realtà sono quindi 54.



Sono stati fatti vari attacchi all'algoritmo COMP128, ad esempio, avendo possesso della SIM si potrebbero mandare valori casuali ed avere in input $SRES_{MS,1}$ che viene restituito dall'autenticazione. Ne fa diversi e alla fine si ottiene la chiave k_i . Per calcolare la chiave ci sono algoritmi che computano $2^{17,5}$ valori dati alla SIM, e, considerando la poca velocità al secondo, si possono fare 6.25 query al secondo, per un tempo di attacco di circa 8 ore.

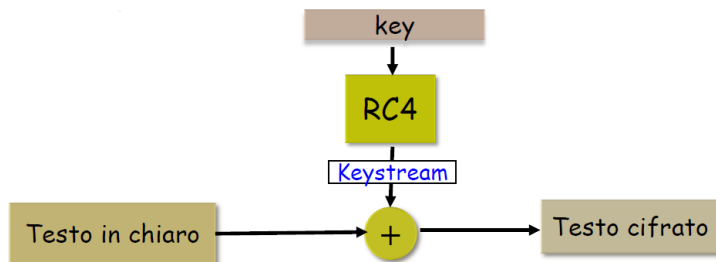
Un algoritmo di forza bruta invece, impiega 2^{54} , poiché considera tutti i tentativi per trovare la chiave giusta.

Per l'esecuzione dell'algoritmo è possibile usare macchine parallele, come ad esempio la già citata COPACOBANA in cui i singoli elementi che effettuano lo stesso calcolo in parallelo, possono essere riprogrammati, per effettuare un attacco su algoritmi A5.

Da un documento emerso da Edward Snowden nel 2013, si legge che l'MSA può rompere l'algoritmo A5/1 anche quando la chiave è sconosciuta.

5.0.3 RC4

In questo caso l'idea non si basa su registri a scorrimento, bensì algoritmi particolari, semplici ed efficaci. Il 4 indica la versione dell'algoritmo, è stato tenuto segreto finché non comparve in forma anonima nel 1994. Viene anche chiamato come Allerged RC4. in questo non è sicuro che sia l'effettivo RC4, ma è l'idea venuta fuori dalla pubblicazione precedentemente detta.



L'idea dell'algoritmo viene mostrata in foto: l'RC4 prende in input una chiave, la processa e ottiene una Keystream. La Keystream effettua un'operazione di XOR con il testo in chiaro per ottenere il testo cifrato.

La chiave è di lunghezza variabile (da 1 a 256 byte), è semplice da implementare e da analizzare, la Keystream ottenuta ha un periodo molto grande, maggiore di 10^{100} e vengono usate operazioni orientate ai byte. Viene utilizzato in vari prodotti, ad esempio nella comunicazione SSL/TLS per la comunicazione sicura sul web, nel protocollo HTTPS e poi nell'IEEE 802.11 wireless LAN: WEP.

Analizziamo ora l'algoritmo dell'RC4: c'è un array di stato S i quali sono una permutazione dei 256 byte. I tre step eseguiti sono:

1. Inizializzazione di S in maniera indipendente dalla chiave.
2. Aggiorna S con la chiave K ottenendo una nuova permutazione
3. Genera la Keystream byte per byte da S, cambiando ancora la permutazione.

La chiave è la seguente: naturalmente è un valore che può essere al massimo di 256 byte

$K[0] \dots K[h-1]$ vettore della chiave
di h byte, $1 \leq h \leq 256$



Inizializzazione di S: banalmente il vettore S che va da 0 a 255 viene inizializzato in ogni singola entrata, mettendo il valore i-esimo all'interno della cella i:



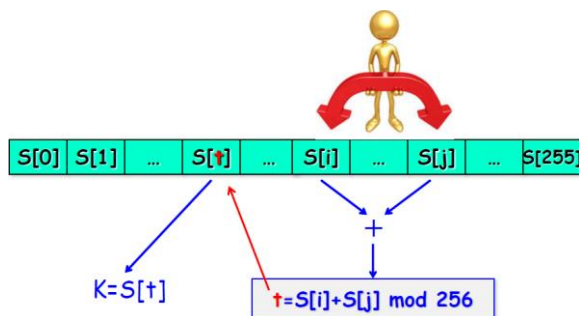
Inizializzazione
for i=0 to 255 do S[i]=i

Aggiornamento di S con la chiave K: l'idea si basa su 4 linee di codice in cui mediante indice i si effettua un ciclo da 0 a 255. Per ognuno di questi valori viene calcolato j e scambia (effettua dunque uno Swap) il valore i-esimo col valore j-esimo. I 256 valori ottenuti nell'inizializzazione non vengono mai modificati, solo scambiati tra di loro. Il calcolo di j è il seguente: $j = (j + S[i] + K[i \bmod h]) \bmod 256$, cioè viene preso il valore precedente di j, viene addizionato al valore della cella i-esima dell'array, sommo il valore della chiave al posto i modulo h, che rappresenta la lunghezza della chiave. Di questa somma ne faccio il modulo 256 per assicurarmi che j diventi un byte compreso tra 0 e 255.

j=0
for i=0 to 255 do
j=(j+S[i]+K[i mod h]) mod 256;
Swap(S[i],S[j]);



Generazione della Keystream da S: l'idea è che vengono fissati due indici i e j, vengono sommati i valori corrispondenti nelle celle in modulo 256, ottengo quindi un valore t, che viene interpretato come indice nell'array di stato. Di conseguenza, la chiave K corrisponde al valore S[t]. Contemporaneamente alla selezione della Keystream, effettuo lo scambio dei valori negli indici i e j per garantire di non ottenere valori uguali.

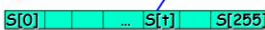


L'algoritmo risultante è il seguente:

```

i, j = 0
while (true)
    i = i+1 mod 256;
    j = (j+S[i]) mod 256;
    Swap(S[i], S[j]);
    t = (S[i]+S[j]) mod 256;
    k = S[t]
    Effettua lo XOR del byte k con il prossimo byte del
    testo in chiaro

```



Inizializzo i e j uguali a zero. Entro nel while(true) e calcolo i come $i+1 \bmod 256$, j come $j + S[i] \bmod 256$, effettuo lo swap tra le celle che hanno indice i e j ottenute dai precedenti calcoli, calcolo t come $S[i] + S[j] \bmod 256$ e il valore corrispondente all'indice t, è la mia chiave. Il byte ottenuto, tenendo a mente la struttura dell'R4, viene utilizzato per lo XOR con il testo in chiaro. Questo ciclo continua ogni qualvolta è necessario un valore per poter effettuare lo XOR.

Vedi slide pag 62 per esempio Wired Equivalent Privacy (WEP/WEP2)...

La problematica del WEP è stata risolta mediante WPA(Wi-Fi Protected Access), la quale è una versione temporanea, che ha permesso in parallelo un protocollo più sicuro: lo standard WPA(2), diventato obbligatorio per chi usufruisse di wifi dal 2006. Generalmente è quello che viene usato da tutti gli utenti. Successivamente, nel 2018, è stato annunciato il WPA3, che non è ancora implementato ancora da tutti i servizi, dipende dalla data di acquisto dei dispositivi. Le caratteristiche di ognuno di essi verranno descritti brevemente:

- WPA: usa Temporal Key Integrity Protocol (TKIP) per mischiare chiave e IV prima di darla ad RC4.
- WPA2: AES e Counter Mode CBC-MAC (Message Authentication Code) Protocol (CCMP).
- WPA3: diviso in 2 versioni:
 - WPA3-Personal: AES-128 in Counter with CBC-MAN (CCM) mode;
 - WPA3-Enterprise (utilizzato dalle aziende): AES-256 in Galois/Counter Mode (GCM) mode ed HMAC con SHA-384.

Analizziamo ora uno standard più attuale di Stream Cipher: **Salsa20**.

5.0.4 SALSA20

In questo Stream Cipher, la chiave è di 256 bit (ma anche 128) e utilizza un vettore di inizializzazione (Nonce/IV) di 64 bit. L'idea di questo Stream Cipher insieme al ChaCha20 è quello di utilizzare delle operazioni che sono facilmente eseguite su architetture a 32 bit, e le operazioni che vengono utilizzate sono 3 e sono dette add-rotate-xor (ARX):

- **Addizione mod 2^{32}** indicato con \boxplus : la seguente operazione viene eseguita facilmente con architettura a 32 bit, partendo dai bit meno significativi, analizzando eventuali riporti ed arrivando ai bit significativi. Interviene il mod 2^{32} poiché quando addiziono due parole a 32 bit, si può avere un riporto, quindi la somma non va in 32 bit. Per rimediare a ciò, viene effettuata la normale addizione fermandomi quando arrivo a 32 bit. I successivi bit di riporto non vengono considerati.
- **XOR** indicato con \oplus : operazione banale e veloce.
- **Rotazioni** indicato con \lll : la parola di 32 bit viene shiftata in maniera circolare di un certo numero di bit.

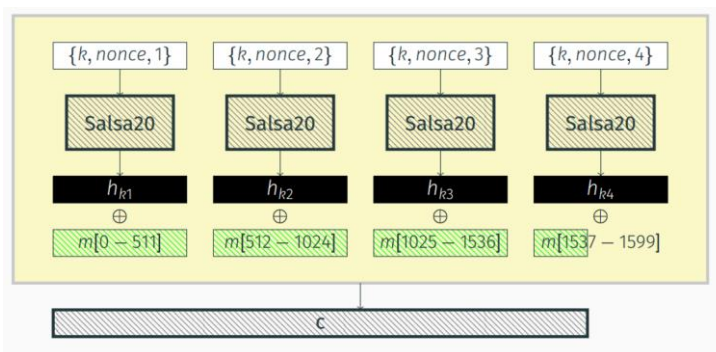
L'idea del Salsa20 è di utilizzare una particolare funzione primitiva detta **R(a,b,c,k)** che è la seguente:

$$b \oplus= (a \boxplus c) \lll k$$

La variabile *a* e la variabile *c* vengono sommate in mod 2^{32} , il risultato viene shiftato a sinistra in maniera circolare così da non perdere informazione di *k* posizioni. Questo valore viene fatto XOR bit a bit con *b*. Il risultato va in *b*.

Il numero di round che viene eseguito è 20, da cui deriva il nome Salsa20. Ciò non toglie che se si vuole risparmiare tempo computazionale, si può usare una versione a 12(Salsa20/12) o 8 round(Salsa20/8)

Analizziamo più in dettaglio lo standard Salsa20:



L'obiettivo di Salsa20, essendo uno Stream Cipher, è di produrre uno stream di bit ($h_{k1}, h_{k2}, h_{k3}, h_{k4}$). Ogni stream di bit va in XOR bit a bit con il messaggio diviso in 512 bit per blocco. I risultati degli XOR combinati, mi danno C che rappresenta il messaggio cifrato. Chiaramente se il messaggio in chiaro non è multiplo di 512, l'ultimo blocco non sarà di 512 ma di meno, come in foto.

Lo stream di output di Salsa20 si ottiene dando in input al Salsa20 la chiave k, il vettore di inizializzazione nonce, e un indice. La terna va in input a Salsa20 per produrre lo stream di bit.

La decifratura, banalmente, si ottiene facendo lo XOR tra il testo cifrato e l'output dello stream di bit.

Adesso esaminiamo in dettaglio l'algoritmo Salsa20 vero e proprio:

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

Opera su uno stato di 512 bit. Questi 512 bit vengono organizzati in una matrice (da 0 a 15) in cui ogni cella contiene 32 bit (infatti $32 \cdot 16 = 512$).

L'algoritmo viene inizializzato in questo modo:

- La chiave viene posizionata 3 volte nella prima riga, 1 volta nella seconda riga, 1 volta nella terza riga e 3 volte nella quarta riga ($32 \cdot 8 = 256$).
- Il vettore di inizializzazione nonce viene posto nella seconda riga ($32 \cdot 2 = 64$).
- Position, ossia l'indice preso in input dall'algoritmo, che viene scritto nella terza riga.
- C'è una costante nella diagonale "expand 32-byte k" in ASCII, il quale è un valore fissato, che è stato correttamente spiegato nella sua costruzione, per evitare obiezioni. Per poterla scrivere in ASCII, avendo 32 bit a disposizione per ogni cella, ogni carattere prende 8 bit.

expa	key	key	key
key	nd 3	nonce	nonce
position	position	2-by	key
key	key	key	te k

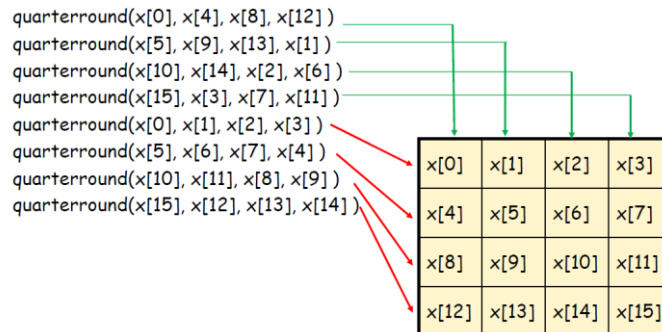
Costante: "expand 32-byte k" in ASCII

Un esempio pratico dello stato iniziale è il seguente:

0x61707865	k[0,31]	k[32,63]	k[64,95]
k[96,127]	0x3320646e	nonce[0,31]	nonce[32,63]
ctr[0,31]	ctr[32,63]	0x79622d32	k[128,159]
k[160,191]	k[192,223]	k[224,255]	0x6b206574

Inseriti correttamente i valori nella matrice, le operazioni vengono effettuate prima su ogni colonna e poi su ogni riga. L'idea dell'algoritmo è questa:

For round = 1 to 10



Viene effettuati 20 round, di cui le prime quattro operazioni sono sulle colonne, successivamente sulle righe. Di conseguenza sono 2 round che vengono ripetuti per 10 volte ($10 \cdot 2 = 20$).

La funzione quarterround, descritta precedentemente come $b \oplus = (a \boxplus c) \lll k$ viene specificata nel seguente modo, senza entrare troppo nel dettaglio:

```

x[ 4] ⊕= (x[ 0] ⊞ x[12])<<7;
x[14] ⊕= (x[10] ⊞ x[ 6])<<7;
x[ 8] ⊕= (x[ 4] ⊞ x[ 0])<<9;
x[ 2] ⊕= (x[14] ⊞ x[10])<<9;
x[12] ⊕= (x[ 8] ⊞ x[ 4])<<13;
x[ 6] ⊕= (x[ 2] ⊞ x[14])<<13;
x[ 0] ⊕= (x[12] ⊞ x[ 8])<<18;
x[10] ⊕= (x[ 6] ⊞ x[ 2])<<18;

x[ 9] ⊕= (x[ 5] ⊞ x[ 1])<<7;
x[ 3] ⊕= (x[15] ⊞ x[11])<<7;
x[13] ⊕= (x[ 9] ⊞ x[ 5])<<9;
x[ 7] ⊕= (x[ 3] ⊞ x[15])<<9;
x[ 1] ⊕= (x[13] ⊞ x[ 9])<<13;
x[11] ⊕= (x[ 7] ⊞ x[ 3])<<13;
x[ 5] ⊕= (x[ 1] ⊞ x[13])<<18;
x[15] ⊕= (x[11] ⊞ x[ 7])<<18;

x[ 1] ⊕= (x[ 0] ⊞ x[ 3])<<7;
x[11] ⊕= (x[10] ⊞ x[ 9])<<7;
x[ 2] ⊕= (x[ 1] ⊞ x[ 0])<<9;
x[ 8] ⊕= (x[11] ⊞ x[10])<<9;
x[ 3] ⊕= (x[ 2] ⊞ x[ 1])<<13;
x[ 9] ⊕= (x[ 8] ⊞ x[11])<<13;
x[ 0] ⊕= (x[ 3] ⊞ x[ 2])<<18;
x[10] ⊕= (x[ 9] ⊞ x[ 8])<<18;

x[ 6] ⊕= (x[ 5] ⊞ x[ 4])<<7;
x[12] ⊕= (x[15] ⊞ x[14])<<7;
x[ 7] ⊕= (x[ 6] ⊞ x[ 5])<<9;
x[13] ⊕= (x[12] ⊞ x[15])<<9;
x[ 4] ⊕= (x[ 7] ⊞ x[ 6])<<13;
x[14] ⊕= (x[13] ⊞ x[12])<<13;
x[ 5] ⊕= (x[ 4] ⊞ x[ 7])<<18;
x[15] ⊕= (x[14] ⊞ x[13])<<18;

```

Ora vediamo l'ultimo Stream Cipher, il **ChaCha20**

5.0.5 CHACHA20

Molto simile a Salsa20, solo che cambia la funzione primitiva rispetto a Salsa20.

La funzione primitiva in ChaCha è la sequenza di queste 3 istruzioni:

- $b \boxplus = c;$
- $a \oplus = b;$
- $a \lll = k;$

Quarter-round QR (a,b,c,d) diventa (non lo chiede all'orale):

- $a \boxplus = b; d \oplus = a; d \lll = 16;$
- $c \boxplus = d; b \oplus = c; b \lll = 12;$
- $a \boxplus = b; d \oplus = a; d \lll = 8;$
- $c \boxplus = d; b \oplus = c; b \lll = 7;$

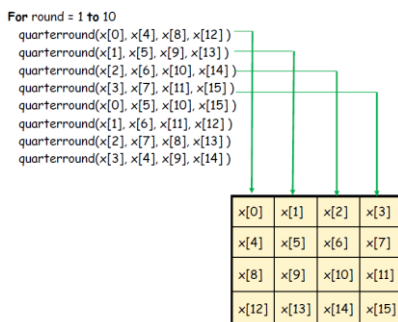
Lo stato iniziale è simile:

expa	nd 3	2-by	te k
key	key	key	key
key	key	key	key
position	position	nonce	nonce

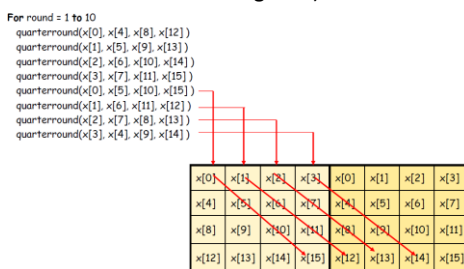
Costante: "expand 32-byte k» in ASCII

In questo caso la chiave viene inserita nella seconda e la terza riga, il vettore nonce viene inserito nelle ultime due celle, position nella quarta riga e la costante nella prima riga. Un po' diverso dal caso precedente ma la filosofia è la stessa.

Le operazioni sono eseguite in questo modo:



Come si vede, le operazioni sulle colonne sono uguali a Salsa20, mentre la seconda parte che in Salsa20 si basava sulle righe, qui si basa sulle diagonal (si inserisce una matrice di copia alla destra per permettere di ottenere le diagonal):



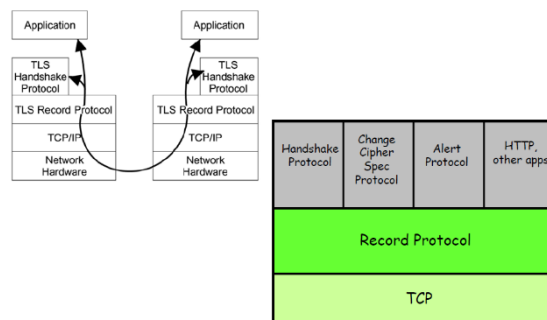
SSL è un sistema creato per garantire una comunicazione sicura tra client e server. Da questo nasce poi TLS, di cui sono uscite varie versioni, fino alla 1.3 in cui è stato aggiunto ChaCha20.

Il TLS si trova sopra il livello TCP/IP e sopra il livello applicazione e ci sono 2 parti: TLS Record Protocol e TLS Handshake Protocol. Poiché è collocato sopra a TCP/IP e sotto Application, il suo obiettivo è di garantire autenticazione e privacy.

Il TLS si compone di una prima parte di Handshake Protocol, dove il client e il server devono negoziare la Ciphersuite, cioè scegliere gli algoritmi della cifratura. Client e server posseggono un insieme di algoritmi, si intersecano per scegliere algoritmi comuni, garantendo sicurezza.

C'è poi una fase di autenticazione in cui il server si autentica al client (mediante certificato rilasciato da un ente, che possiede anche la corrispettiva chiave privata). Una volta stabilita l'autenticazione, si devono stabilire le chiavi da usare (Record Protocol).

L'Handshake protocol avviene mediante interazioni tra client e server. Naturalmente chi inizia la comunicazione è sempre il client per formare un insieme di operazioni. Analizziamo con più dettaglio quelli che sono i passi per stabilire una sessione fra client e server:



Nell’inizializzazione tra client e server, il client manda al server un insieme di algoritmi, che servono per la confidenzialità ed integrità. Una volta che il Server riceve questa lista di algoritmi, nell’intersezione, sceglie gli algoritmi che verranno utilizzati. Viene dunque scelta la suite di algoritmi che verranno utilizzati per lo scambio di informazioni.

A questo punto il Server invia un Certificato che contiene una chiave pubblica del Server firmato da una Autorità di Certificazione.

Il Client a questo punto valida il certificato, cioè ne verifica la correttezza, se così non fosse il Client riceve un Warning di sicurezza prima dell’ingresso al sito.

Se il lavoro del Server si fermasse all’invio di questo certificato, allora chiunque potrebbe copiare questa parte descritta fino ad ora.

Il Protocollo SSL, quindi, calcola, ogni qualvolta si crea una sessione, una chiave di sessione (Pre Master Key), la quale viene cifrata con la chiave pubblica presente nel certificato e inviata al Server. Se il Server riesce a decifrare allora ottiene la Pre Master Key del Client, rispetto alla quale possono comunicare in maniera confidenziale e autenticata, poiché sia Client che Server posseggono la Pre Master Key, o chiamata lato Server Shared Secret Key.

