Tesina per il corso di "Elementi di Crittografia"
A.A. 2004/2005

La crittografia nella telefonia mobile

Cimaglia Lorenzo

Conti Emanuele

Gerardi Francesco

1 Introduzione

Nella evoluzione della telefonia possiamo individuare tre principali tecnologie che segnano i punti cardine di questo sviluppo:

- **Telefoni Tacs ed ETacs**. E' la prima generazione di telefoni cellulari, basata su una comunicazione di tipo analogico.
- **GSM e GPRS**. La prima tecnologia rappresenta la cosiddetta seconda generazione, mentre la seconda è un'evoluzione che porta delle estensioni ma che non segna una vera svolta.
- **UMTS**. E' la terza generazione di telefonia mobile e fa parte dello standard IMT-2000. Questo è un tentativo di standardizzazione delle telecomunicazioni da quelle satellitari alle reti senza fili terrestri (UTRA). UMTS è solo una parte, seppur molto importante, di questo standard.

1.1 Breve cronologia sull'evoluzione della telefonia mobile

1979 i primi telefoni cellulari analogici vengono sperimentati in USA e nel 1981 nei paesi scandinavi ma si è ancora lontani dalla concezione commerciale di tali tecnologie.

1982 il CEPT istituisce il gruppo di ricerca GSM per la formazione di uno standard che permettesse il roaming (la possibilità di comunicazione tra operatori diversi) internazionale.

1985 in Inghilterra compaiono i primi Total Access Communication System (TACS) per scopo commerciale. A 6 anni dalle prime sperimentazioni il sistema di comunicazione inizia ad essere funzionante e accessibile

1990 ad Aprile in Italia viene attivata la rete Etacs a 900 MHz. In breve però la SIP diviene il più grande operatore di telefonia mobile europeo. Nello stesso periodo (o pochi anni dopo) in alcuni paesi europei viene introdotta la telefonia digitale attraverso lo standard GSM.

1992 viene attivata la copertura GSM di alcuni dei tratti autostradali Italiani.

1995 sempre in ritardo rispetto ai paesi del nord dell'Europa inizia il servizio commerciale GSM della TIM. Viene quindi introdotta la telefonia digitale.

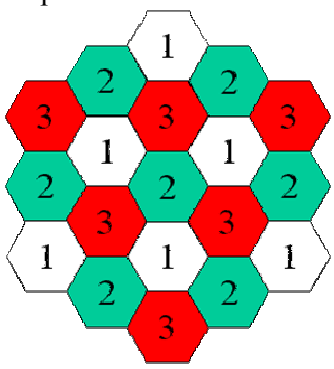
1996 dopo oltre un anno di incomunicabilità, è attivato il roaming nazionale tra Omnitel e TIM (ricordiamo che il GSM doveva essere lo standard che permetteva il roaming internazionale).

Verso la fine degli anni '90 vengono introdotti i servizi a pacchetto GPRS.

Attualmente la tecnologia UMTS sta prendendo sempre più piede e in prospettiva futura sarà la tecnologia dominante. I lavori sull'IMT-2000 iniziarono già nel 1992, dopo 10 anni dall'istituzione del gruppo di ricerca GSM.

1.2 I concetti principali

- **MS mobile station**. E' l'apparecchio telefonico o meglio di comunicazione.
- **Cella**. E' la zona più piccola di copertura del territorio.



- **BTS base tranciever station**. E' il primo referente della MS ed in pratica è un trasmettitore che scambia informazioni con la MS e che permette la ricezione del segnale. Ogni BS permette l'identificazione di una cella.
- **BS base station**. Ha associati più BTS. Determina la disposizione delle celle nell'area tramite la disposizione delle varie BTS.
- **VLR visitor location register**. Contiene le informazioni temporanee e locali sulle MS che sono nell'area. Tra le informazioni è presente anche la dislocazione spaziale dell'apparecchio (la cella di "residenza").
- **HLR home location register**. Contiene le informazioni di tutti gli abbonati. Ce n'è di solito uno per ogni operatore.
- **MSC mobile services switching center**. E' la parte centrale della rete e realizza la connessione tra l'utente della rete mobile e le altre reti (terrestri o mobili di altri operatori oltre che quella dell'operatore di riferimento).
- **AuC autentication center**. Realizza le funzioni di autenticazione e applica gli algoritmi necessari.

Molti altri "attori" intervengono nelle comunicazioni mobili ma, in questo caso, è opportuno citare solo quelli fondamentali alla comprensione di ciò che segue, per non distogliere l'attenzione dall'argomento principale.

2 Tacs ed ETacs

Tacs (*Total Access Communication System*) ed *ExtendedTacs* sono le prime forme di comunicazione mobile e, in entrambi i casi, sono di tipo analogico. Il primo sfruttava le frequenze tra 890 e 960 MHz collocandovi 1000 canali a distanza di 25 KHz disponendo di una portante di ampiezza 45KHz ((960-890Mhz/1000)-25KHz). Il secondo invece è una estensione del primo, le frequenze sono spostate tra 872 e 950 e i canali diventano 1320 (la larghezza di banda si portava quindi a 32Khz).

Ad ogni utente veniva assegnata una portante intera e che rimaneva la stessa per tutta la conversazione. L'unica modulazione usata era quella di frequenza (**FDMA**). Nessun tipo di sistema per la protezione della conversazione veniva utilizzato lasciando le informazioni trasmesse in chiaro.

Evidentemente i problemi erano molti. L'intercettazione, in particolare, era semplice. Bastava uno scanner di frequenze per ascoltare le conversazioni nell'area coperta dall'apparecchio.

Anche il sistema di autenticazione, basato solo sulla trasmissione di un codice di 15 cifre (il codice IMEI) da parte della MS alla Base Station, era particolarmente debole. Era sufficiente, infatti, oltre ad avere il numero di telefono, intercettare la prima parte della comunicazione, durante la quale era trasmesso il codice, per entrarne in possesso e poter così "clonare" il telefono.

2.1 Il codice IMEI

Il codice **IMEI** è costituito da 15 cifre e può avere due diversi formati a seconda della data di fabbricazione dell'apparecchio:

- Per apparecchi fabbricati prima del 1° aprile 2004: `aabbbb-cc-dddddd-e`
  - `aabbbb` è il *Type Approval Code* (**TAC**) in cui le prime due cifre rappresentano il paese

    - `cc` è il *Final Assembly Code* (**FAC**) e identifica il produttore
    - `dddddd` è il *codice seriale della MS* (**SNR**)
    - `e` è una cifra di controllo solitamente settata a 0
- Per apparecchi fabbricati dopo questa data: `xxxxxxxx-dddddd-e`
    - `xxxxxxxx` è il Type Allocation Code
    - `dddddd` ed `e` hanno lo stesso significato del precedente formato
    - Non c'è più l'identificazione del costruttore

Per maggiore chiarezza è utile riportare due esempi e alcuni FAC di costruttori:

350601-10-845702-0	NOKIA 8210 (primo formato)
35366700-111867-8	SIEMENS M65 (secondo formato)

01,02 =AEG	60 =Alcatel
07,40 =Motorola	61 =Ericsson
10,20 =Nokia	65 =AEG
30=Ericsson	70 =Sagem
40,41,44=Siemens	75 =Dancall
50=Bosch	80=Philips
51=Sony, Siemens, Ericsson	85 =Panasonic

Nel caso in cui ci siano due cifre aggiuntive alla fine del codice queste indicano la versione del software installato.
Per visualizzare il codice è sufficiente digitare *#06# sul proprio telefonino.

2.2 L'introduzione delle SIM card

Le **SIM** (*Subscriber Identity Module*) sono dei supporti che contengono il microchip in cui sono cablati diversi dati fondamentali e in cui possono essere conservate informazioni private dell'utente (messaggi, voci della rubrica, PIN, PIN2, PUK, etc.).
Le informazioni più importanti che sono all'interno della SIM card sono:

- Algoritmo di identificazione e quello di generazione della chiave di sessione **A3** e **A8**
- **Ki** chiave necessaria al funzionamento degli algoritmi.
- **IMSI**

Nonostante il senso comune faccia credere che sia così, il numero telefonico non è un dato memorizzato nella SIM card (motivo per cui è possibile cambiare numero senza cambiare scheda) ma è conservato nelle basi di dati degli operatori.

2.3 Il codice IMSI

Il codice IMSI è costituito da 15 cifre memorizzato all'interno della SIM card. I campi in cui il codice è suddiviso sono:

- Mobile Country Code (**MCC**)
- Mobile Network Code (**MNC**)
- Mobile Subscriber Identity Number (**MSIN**)

Ad esempio:

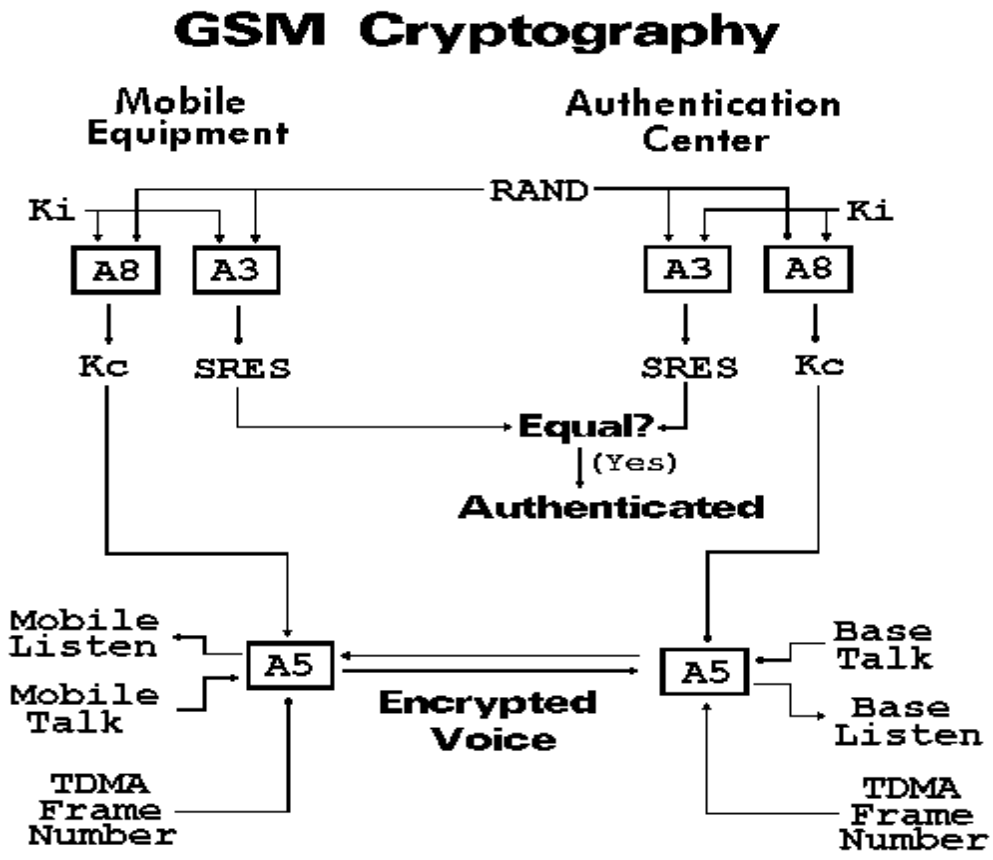
- 222 MCC Italia
- 88 MNC TIM
- 1651310747 MSIN

Ogni IMSI corrisponde ad una sola SIM card e ogni operatore, tramite questo codice, è in grado di identificare all'interno dei propri database (HLR o VLR a livello locale) il numero telefonico associato alla SIM card. Questo è uno dei cambiamenti principali portati dall'introduzione di questi supporti. Prima infatti il numero telefonico era indissolubilmente legato all'apparecchio. Era l'IMEI a permettere la corrispondenza all'interno dei database tra MS e numero telefonico. Per utilizzare un numero diverso quindi era obbligatorio cambiare tipo di contratto, contattando l'operatore.

3 GSM

3.1 Concetti generali

Il GSM nasce dal lavoro del gruppo di ricerca GSM (a partire dal 1982) istituito dal CEPT con l'esigenza di trovare uno standard sicuro e soprattutto unico per tutte le nazioni e permettere così anche il roaming internazionale. Vengono finalmente introdotte le multiplazioni di frequenza e di tempo e si dà la possibilità di accesso al servizio ad una quantità di utenti nettamente superiore rispetto a quello dell'ETacs. Inoltre con l'introduzione delle SIM card e del GSM inizia l'epoca delle comunicazioni mobili digitali. Questa innovazione permette non solo l'introduzione di servizi innovativi quali gli SMS, ma anche un notevole passo avanti nella sicurezza della comunicazione. Gli algoritmi contenuti nella card, lo sfruttamento della segmentazione delle informazioni e l'introduzione di molti canali ausiliari di comunicazione, danno la possibilità di implementare algoritmi di cifratura vari e complessi.



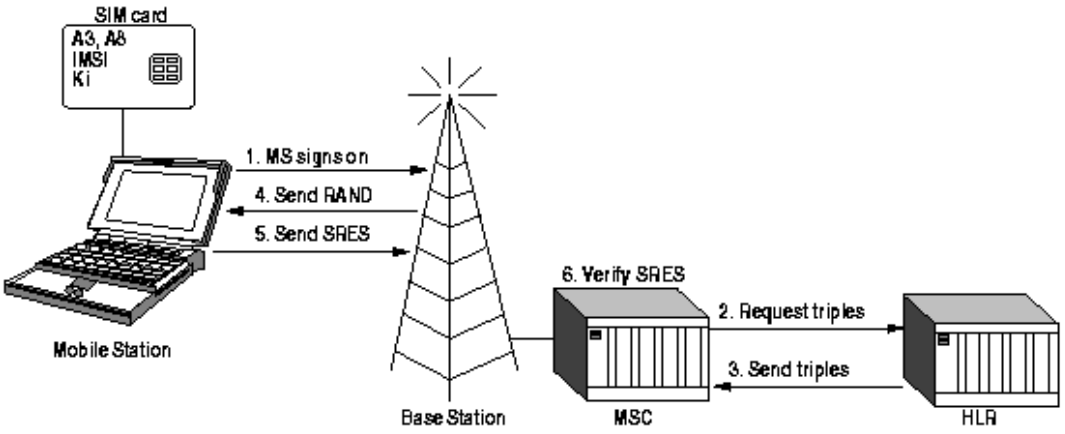
Il sistema di crittografia del GSM comprende tre algoritmi fondamentali che si articolano in due fasi: autenticazione e cifratura della comunicazione.
Nella prima fase intervengono gli algoritmi **A3** e **A8** mentre nella seconda il più complesso algoritmo **A5** nelle sue molte varianti.

3.2 A3 e A8

Come visto i primi due algoritmi sono memorizzati nella SIM card e devono essere conosciuti (nella stessa implementazione) dalle strutture del gestore (BS e Autentication Center).

L'autenticazione di un terminale avviene in 3 fasi sulla base di un protocollo "sfida - risposta":

1. Richiesta di autenticazione da parte della MS
2. Spedizione della "sfida" RAND di 16 byte
3. Calcolo e spedizione della "risposta" SRES tramite l'algoritmo A3
4. Confronto della correttezza di SRES e abilitazione MS



L'algoritmo A8 è necessario alla generazione della chiave di sessione Kc, che verrà usata successivamente dall'algoritmo A5 per cifrare la comunicazione.
In quasi tutti i casi reali A3 e A8 vengono implementati insieme dall'algoritmo **COMP128**.

3.3 COMP128

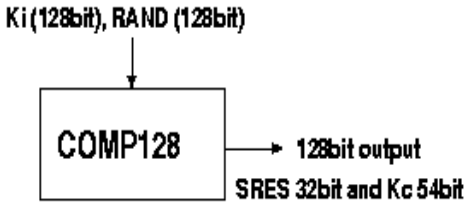
3.3.1 L'algoritmo

Questo algoritmo prende in input:

- La chiave segreta **Ki** contenuta nella SIM card
- La sfida **RAND** della BS

L'output a 96 bit consiste di:

- **SRES** 4 byte
- **Kc** necessaria ad altri algoritmi 8 byte



E' stato abbastanza semplice reperire su internet una implementazione in C dell'algoritmo e testarla (nonostante presentasse qualche piccolo errore molto semplice da individuare).
Parte di questa implementazione (secondo l'autore) è stata reperita da materiale ufficiale e parte da un reverse engeneering.
Per prima cosa Ki e RAND vengono inseriti in un vettore X[] di 32 byte.
Nei primi 16 viene messa Ki e nei byte da 16 a 31 RAND.
Quindi viene eseguito un loop per 8 volte ricaricando ad ogni iterazione Ki nei primi 16 byte.

```
/* ( Load RAND into last 16 bytes of input ) */
     for (i=16; i<32; i++)
          x[i] = rand[i-16];
     /* ( Loop eight times ) */
     for (i=1; i<9; i++) {
          /* ( Load key into first 16 bytes of input ) */
          for (j=0; j<16; j++)
               x[j] = key[j];
```

In ogni loop inizia con una struttura di **compressione detta "a farfalla"** che utilizza delle tavole di compressione standard che sono in realtà matrici: T0[512],T1[256],T2[128],T3[64],T4[32].
Ogni Ti contiene valori compresi tra 0 e $2^{(8-i)}$-1.
Per chiarezza e completezza è interessante riportare almeno una di queste tavole ed in particolare quella da 128 valori (compresi tra 0 e $2^{(8-2)}$-1=63)

```
table_2[128] = {
         52, 50, 44,  6, 21, 49, 41, 59, 39, 51, 25, 32, 51, 47, 52, 43,
         37,  4, 40, 34, 61, 12, 28,  4, 58, 23,  8, 15, 12, 22,  9, 18,
         55, 10, 33, 35, 50,  1, 43,  3, 57, 13, 62, 14,  7, 42, 44, 59,
         62, 57, 27,  6,  8, 31, 26, 54, 41, 22, 45, 20, 39,  3, 16, 56,
         48,  2, 21, 28, 36, 42, 60, 33, 34, 18,  0, 11, 24, 10, 17, 61,
         29, 14, 45, 26, 55, 46, 11, 17, 54, 46,  9, 24, 30, 60, 32,  0,
         20, 38,  2, 30, 58, 35,  1, 16, 56, 40, 23, 48, 13, 19, 19, 27,
         31, 53, 47, 38, 63, 15, 49,  5, 37, 53, 25, 36, 63, 29,  5,  7
    }
```

La compressione così restituisce 32 valori di 4 bit che verranno assemblati in 16 byte (in un array) prima di una nuova iterazione. Solo successivamente i 16 byte saranno compressi in 12 byte per formare l'output del programma.
Il ciclo principale dell'algoritmo è costituito come si vede nel codice seguente.

```
/* ( Perform substitutions ) */
            for (j=0; j<5; j++)
                  for (k=0; k<(1<<j); k++)
                        for (l=0; l<(1<<(4-j)); l++) {
                              m = l + k*(1<<(5-j));
                              n = m + (1<<(4-j));
                              y = (x[m]+2*x[n]) % (1<<(9-j));
```

```
            z = (2*x[m]+x[n]) % (1<<(9-j));
            x[m] = table[j][y];
            x[n] = table[j][z];
    }
```

Gli shift a sinistra, rappresentati dai caratteri <<, implementano gli elevamenti a potenza dei byte.
I primo ciclo (quello su j) indica il livello di compressione dell'algoritmo e quindi determina quale delle tabelle utilizzare. Gli altri due invece servono ad identificare l'elemento del vettore che viene scelto per il passo di compressione.

Volendo considerare il vettore di lavoro x[] diviso in due sottovettori di 16 byte (e successivamente in 4 da 8 byte e così via) è possibile vedere come l'algoritmo, ad ogni iterazione, combini un elemento del sottovettore a destra con quello corrispondente a sinistra dando vita alla già citata compressione "a farfalla". Con un paio di esempi, anche solo parziali, è possibile notare quali siano gli elementi dei vettori coinvolte nel calcolo in relazione al livello di compressione e capire meglio quanto detto.

Supponiamo per semplicità che i=0, j=0, k=0 cioè il primo livello di compressione e i primi elementi coinvolti. Allora avremo il vettore X[] diviso in due sottovettori di 16 elementi e, effettuando i calcoli:

$m = l + k*2^{(5-j)} = 0 + 0 * 32 = 0$
$n = m + 2^{(4-j)} = 0 + 16$
$y = (X[0] + 2*X[16]) \bmod 512$
$z = (2*X[0] + X[16]) \bmod 512$

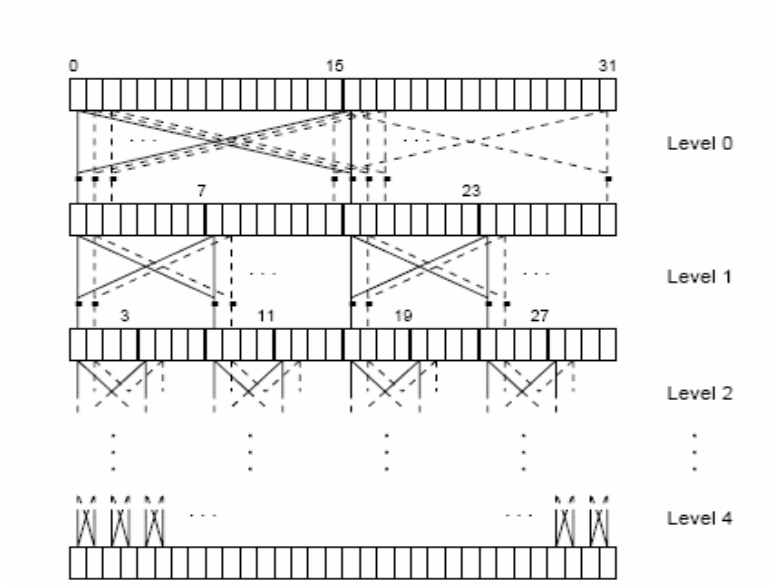
Gli elementi del vettore coinvolti sono il primo del sottovettore di sinistra x[0] e il primo del sottovettore di destra x[16].

Nel secondo esempio supponiamo il terzo livello di compressione j=2 e i primi elementi coinvolti. Il vettore può essere pensato come diviso in 8 "frame" ognuno di 4 elementi. Svolgendo i calcoli si ha:

l=0 j=2 k=0
m = 0
n = m + 2(4-2) = 4
$y = (X[0] + 2*X[8]) \bmod 512$
$z = (2*X[0] + X[8]) \bmod 512$

Vengono coinvolti il primo elemento del primo sottovettore e il primo del sottovettore immediatamente alla sua destra.

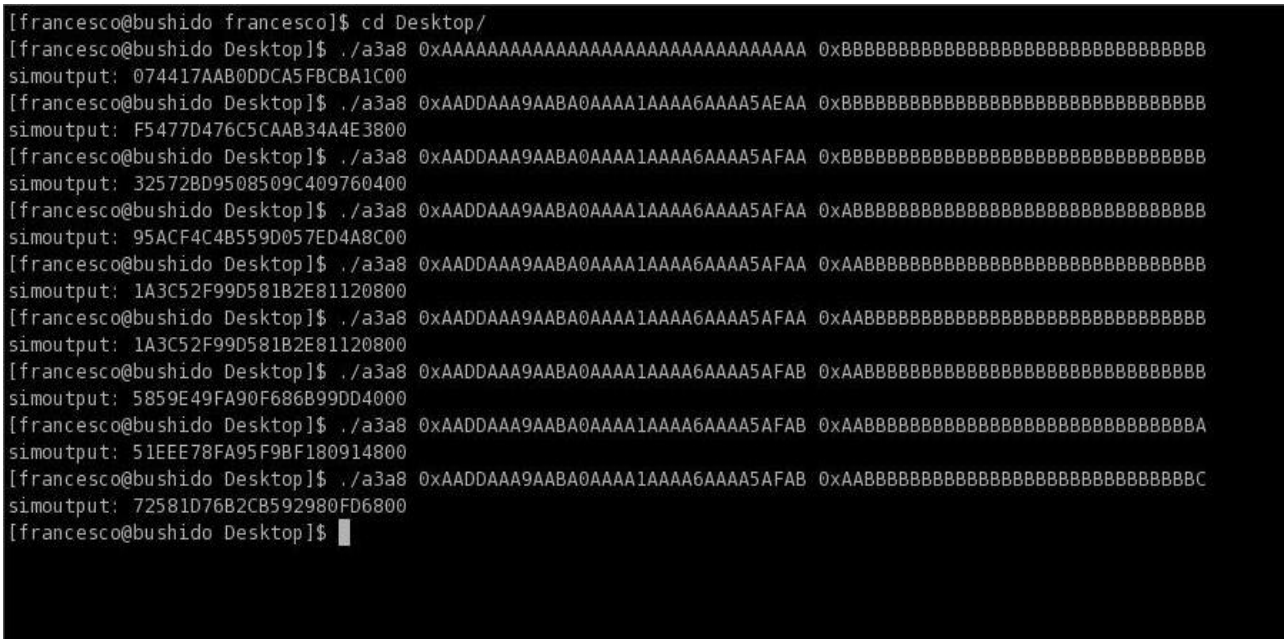
La caratteristica di questo tipo di compressione risulta comunque più chiara tenendo presente la figura che è stata riportata sotto.



Inoltre, ad ogni iterazione tranne che nell'ultima, vengono effettuate delle permutazioni all'interno dell'array x[].

```
/* ( Permutation but not on the last loop ) */
if (i < 8)
      for (j=0; j<16; j++) {
                  x[j+16] = 0;
                  for (k=0; k<8; k++) {
                        next_bit = ((8*j + k)*17) % 128;
                        x[j+16] |= bit[next_bit] << (7-k);
```

Alla fine di questa parte dell'algoritmo, come già detto viene effettuata una ulteriore compressione che, porta il vettore da 16 a 12 byte. Questa parte di codice è dedotta dal riverse engeneering e quindi non viene riportata.
Il programma C è stato compilato e sono stati fatti alcuni test esemplificativi di cui è possibile vedere i risultati nelle figure seguenti. Gli input e l'output sono in formato esadecimale.



E' possibile notare che un cambiamento anche minimo di Ki o della sfida porta ad un cambiamento molto evidente dell'output generato.

E' importante ricordare che la Kc generata è di 64 bit ma in realtà gli ultimi 10 sono solo zeri (la chiave si riduce a 54 bit).
Nell'esempio si può notare che le ultime due cifre esadecimali sono sempre 00 e che la terzultima cifra, nel sistema binario ha sempre almeno due zeri come cifre più a destra (8=1000, C=1100, etc…).

3.3.2 Gli attacchi a COMP 128 (A3/A8)

L'algoritmo COMP 128 ha vari difetti che ne hanno compromesso la stabilità. In particolare sono troppe le informazioni che l'algoritmo esplicita nei propri risultati. Molti attacchi sono stati provati a tale algoritmo. Ne riportiamo due che sono sembrati essere significativi (anche se probabilmente ve ne sono di più recenti).

Primo attacco
La Smartcard Developer Association e la ISAAC (gruppo di ricerca sulla sicurezza) sono stati in grado di trovare una debolezza in COMP 128 e sono stati in grado di reperire, da una SIM card a cui avevano accesso, il segreto Ki.
L'attacco è stato del tipo chosen-challenge, cioè un attacco del tipo brute force, ma solo su valori scelti. Sono stati sottoposti alla SIM card 150.000 valori differenti di RAND e, collezionate le risposte, è stata applicata una crittoanalisi differenziale. Le macchine che danno accesso alla SIM card e che hanno permesso di realizzare questo attacco sono facilmente reperibili (su internet ce ne sono a migliaia). Per generare i valori invece è stato usato un calcolatore in grado di generare 6.25 query al secondo. Con un breve calcolo si può quindi vedere che sono state sufficienti 6 ore e 40 minuti (a cui va però sommato il tempo necessario alla crittoanalisi differenziale stimati in circa un'ora e mezza) per rompere l'algoritmo.

E' importante ricordare che in questo modo in realtà gli algoritmi rotti sono due A3 (per l'autenticazione) e A8 (per la generazione della chiave di sessione).
Il chiaro limite di questo attacco è la necessità di possedere fisicamente la SIM card per più di 6 ore ma i vantaggi che derivano dalla successiva clonazione sono moltissimi (basti pensare che è possibile autenticarsi al posto di qualcun'altro).

Secondo attacco

Un attacco migliore è stato teorizzato ma non realizzato dagli stesso gruppi che hanno prodotto il primo. Il miglioramento consiste nell'eliminare la debolezza del primo e reperire il segreto senza impossessarsi della SIM card fisicamente.
Il funzionamento è lo stesso del primo attacco ma questa volta, tramite specifiche apparecchiature ci si sostituisce alla BS generando un segnale più potente, magari in zone in cui di solito c'è assenza di copertura (come la metropolitana). La stazione mobile viene allora "bombardata" di query esattamente come veniva fatto nel primo attacco.
Il tempo di azione è stato stimato intorno alle 8-13 ore che però possono essere "frammentate" in brevi intervalli di tempo (altrimenti sarebbe praticamente impossibile attuare realmente questo attacco). Le apparecchiature necessarie però sono illegali e l'attacco non è stato attuato…almeno alla luce del giorno.

3.4 A5 : cifratura sul canale radio

3.4.1 Parametri

Come si è già accennato, la chiave K_i viene utilizzata sia per l'autenticazione alla rete GSM, sia per la generazione della **chiave di sessione a 64 bit K_c**, usata per la codifica sul canale via-etere. E' interessante notare che la chiave di sessione generata dalla network giunge dalla network fino alla BS ma non viene mai trasmessa sul canale radio, per maggiore sicurezza.

Il flusso di dati trasmesso tra la il terminale utente (MS, mobile station) e la stazione base (BTS, base-tranceiver station) viene diviso in **frame**. Ciascun frame contiene **114 bit** e rappresenta una parte della comunicazione tra MS e BTS o viceversa. Un nuovo frame viene inviato ogni 4.6 ms ed è identificato con un numero sequenziale di 22 bit chiamato F_n. Ogni volta che viene inviato un frame, F_n viene incrementato di 1. All'inizio di una trasmissione è necessario che MS e BTS si accordino sul valore iniziale di F_n. Per l'intera durata della comunicazione, vengono inviati due frame alla volta: uno da MS a BS ed uno da BS a MS.

Quando la MS vuole comunicare con la rete GSM, invia una richiesta di accesso alla BTS che la inoltra alla MSC. Alla richiesta di identificazione della MS alla rete, l'HLR fornisce alla MSC cinque triple contenenti:

- **RAND**, un numero casuale a **128 bit**;
- **SRES (32 bit)**, calcolata su RAND e sulla chiave K_i, utilizzando l'algoritmo A3;
- **K_c (64 bit)**, la chiave di sessione. Essa viene calcolata mediante l'algoritmo A8 che prende in input **RAND** e K_i.

Ogni tripla viene utilizzata per una autenticazione della MS confrontando la SRES inviata dalla MS con la SRES della tripla. Quando tutte le triple sono state utilizzate l'HLR fornisce un nuovo insieme di cinque triple per l'MSC.

3.4.2 Le versioni di A5

L'algoritmo addetto alla cifratura dei singoli frame da 114 bit è A5: un algoritmo a 64 bit custodito nell'hardware del terminale e nella BS. Di questo algoritmo ne esistono diverse versioni di cui si riportano solo quelle utilizzate in pratica:

- **A5/0**: questa sigla corrisponde a non avere alcun algoritmo di cifratura in etere.
- **A5/1**: questa versione viene utilizzata in Europa. L'algoritmo A5/1 non fu reso pubblico dopo la sua definizione. Solo nel 1994 il disegno generale fu scoperto e nel 1999 Marc Briceno riuscì a fare il reverse engeneering degli algoritmi, rendendo pubblico il codice. Questo algoritmo usa una chiave di 64 bit derivata da quella di autenticazione a 128 bit.
- **A5/2**: il resto del mondo utilizza questa versione. Questo algoritmo è notevolmente più debole ed è stato progettato unicamente per scopi di esportazione. Può essere rotto in meno di un secondo con un normale PC.
- **A5/3**: è basato sul cifrario a blocchi **KASUMI** e fu introdotto nel 2002 per rispondere alle debolezze riscontrate nelle versioni precedenti. Questo algoritmo utilizza una chiave a 128 bit. Tuttavia quando la chiave in input ha una lunghezza minore di 128, la chiave di 128 bit è semplicemente ottenuta ripetendo i primi valori di quella in input. Nel caso della chiave di sessione di 64 bit, dunque, la chiave è ottenuta mediante duplicazione.

3.4.3 Gli stream cipher (A5/1, A5/2)

Gli algoritmi A5/1 e A5/2 appartengono alla famiglia degli **stream cipher**. Questo tipo di cifrari costituiscono un approccio alternativo rispetto a quello dei cifrari a blocchi. Gli stream cipher operano su unità molto piccole dei dati, solitamente su singoli bit. Inoltre rispetto ai cifrari a blocchi nei quali la stessa chiave k viene riutilizzata per la cifratura di ogni blocco, l'idea di base degli stream cipher consiste nel generare una sequenza, a partire dala chiave k, detta **keystream**:

$$z = z_1 z_2 \ldots$$

e nell'utilizzarla per cifrare la stringa del testo in chiaro :

$$x = x_1 x_2 \ldots$$

nel modo seguente:

$$y = y_1 y_2 \ldots = e_{z1}(x1)\, e_{z2}(x2) \ldots$$

dove y rappresenta il testo cifrato e inoltre :

$$z_i = f_i(k, x_1, x_2, \ldots, x_i\text{-}1)$$

è una qualche funzione della chiave *k* e dei caratteri precedenti x_i del testo in chiaro. Per cifrare la stringa x occorrerà calcolare z1 per poi ottenere y1, z2 per poi ottenere y2 e così via.

Gli stream cipher sono spesso definiti in Z_2. In tal caso le funzioni di cifratura e decifratura potrebbero ad esempio essere le seguenti:

$$\text{encryption: } e_{zi}(x_i) = x_i + z_i \bmod 2$$

$$\text{decryption: } d_{zi}(y_i) = y_i - z_i \bmod 2$$

Associando a "0" il valore booleano "falso", e ad "1" il valore booleano "vero" allora l'addizione e la sottrazione modulo 2 corrispondono allo XOR, pertanto la cifratura e la decifratura possono

essere implementate molto efficientemente in hardware. Questa è in effetti l'implementazione tipica.

Un metodo per generare la keystream a partire da m valori della chiave $k_1, \ldots, k_m$, consiste nel porre i primi m valori della keystream uguali ai primi m valori della chiave. Formalmente:

$$\text{per } z_1,\ldots,z_m \text{ con } z_i = k_i \text{ per ogni } i = 1,2,\ldots,m$$

e nel calcolare i valori successivi in base ad una relazione di ricorrenza lineare di grado m:

$$z_{i+m} = \sum_{j=0}^{m-1} c_j z_{i+j} \bmod 2$$

dove $c_0, \ldots, c_{m-1} \in Z_2$ sono costanti predeterminate che individuano univocamente un polinomio chiamato *polinomio delle connessioni*.

Esempio 1
Sia $m = 4$ e sia la keystream generata nel modo seguente:

$$z_{i+4} = z_i + z_{i+1} \bmod 2$$

dove $i > 1$. Se ad esempio si considerano i seguenti valori di k: *1, 0, 0, 0* la keystream corrispondente è:

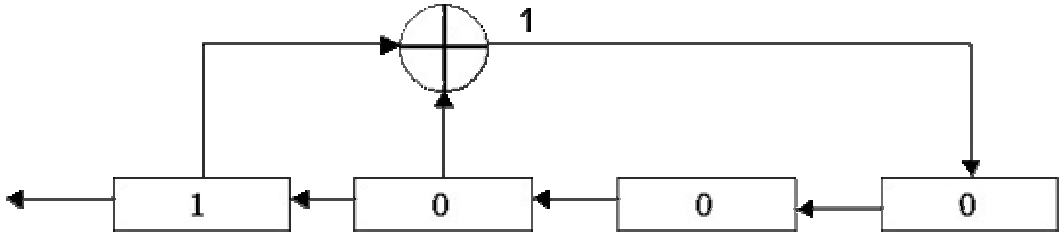
1 0 0 0 **1 0 0 1 1 0 . . .**

Un altro aspetto interessante del metodo appena esposto è che la keystream può essere generata in hardware in modo molto efficiente utilizzando un **linear feedback shift register (LFSR)**.
Si consideri un registro con m stadi. Il vettore $(k_1, \ldots\ldots, k_m)$ viene usato per l'inizializzazione. Ad ogni istante vengono effettuate contemporaneamente le seguenti operazioni:

1) k_1 è il bit di output del LFSR
2) valori $k_2, \ldots\ldots, k_m$ vengono shiftati a sinistra di una posizione
3) viene calcolato il nuovo valore di k_m, che è

$$k_m = \sum_{j=0}^{m-1} c_j z_{i+1} \bmod 2$$

Esempio 2
Consideriamo il linear feedback shift register relativo all'esempio precedente.



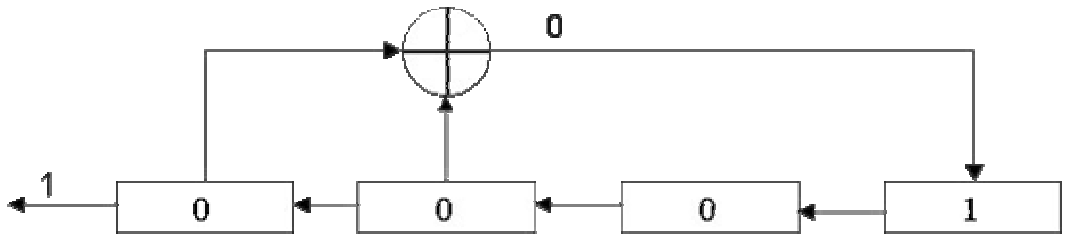
L'inizializzazione dei registri viene fatta in questo modo:

$$z_0 = 1, \quad z_1 = 0, \quad z_2 = 0, \quad z_3 = 0$$

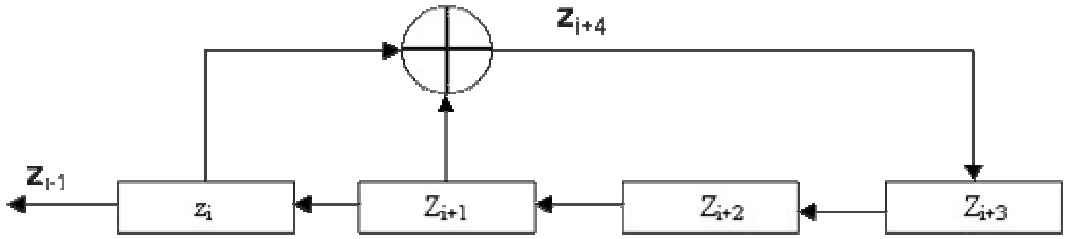
Eseguendo i vari passi:

1) il contenuto del primo registro k_1 (cioè 1) sarà dato in output come primo bit della keystream
2) il contenuto degli altri registri sarà shiftato di una posizione a sinistra
3) il nuovo valore di k_m è calcolato come XOR dei primi due registri verrà inserito nell'ultimo registro (1+0=1)

Il risultato complessivo per questa prima serie di passi è riportato nella figura seguente :



In generale il funzionamento del linear feedback shift register è illustrato nella seguente figura:



Quando si utilizza uno stream cipher, da un certo istante in poi, i bit della keystream cominciano a ripetersi. L'ampiezza dell'intervallo di ripetizione dei bit si chiama *periodo*. Il periodo dipende dal polinomio delle connessioni e tanto più è grande, tanto più il cifrario è sicuro. La keystream relativa a questo esempio ha periodo *15* (al massimo possiamo avere 16 combinazioni dei 4 bit, ed inoltre quella con tutti 0 non viene considerata, poiché darebbe luogo ad una keystream composta da soli 0) ed il polinomio delle connessioni ad essa associato è:

$$x^4 + x^3 + 1$$

3.4.4 A5/1

L'algoritmo A5/1 è lo *stream cipher* utilizzato per cifrare i frame che vengono inviati via-etere. Lo stream cipher viene inizializzato ogni volta che un frame viene inviato con la chiave di sessione K_c e con il numero di frame da cifrare. La stessa K_c viene utilizzata per tutta la sessione, durante la quale il numero del frame di 22 bit cambia. In questo modo viene generata un' unica keystream per ciascun frame in quanto funzione sia della chiave di sessione K_c che del numero di frame F_n.
L'algoritmo A5, utilizzato nei paesi europei, consiste di tre LFSR di lunghezze differenti, organizzati in modo non lineare, che corrispondono ai tre polinomi delle connessioni:

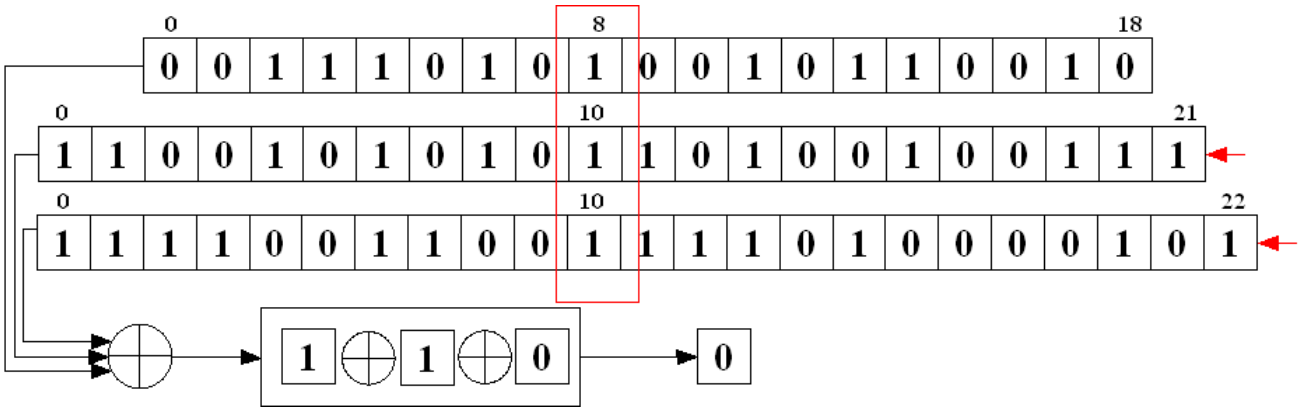
$$x^{19} + x^5 + x^2 + x + 1$$
$$x^{22} + x + 1$$
$$x^{23} + x^{15} + x^2 + x + 1$$

Tali polinomi sono stati scelti primitivi in modo che i corrispondenti registri abbiano periodo massimale. Il periodo massimo di un registro a k bit è $2^k - 1$, che è la cardinalità dell'insieme delle sequenze di k bit, tranne quella composta da tutti 0 (esclusa in quanto darebbe una keystream composta da soli zeri). Quindi i loro periodo sono rispettivamente pari a 2^{19} -1, 2^{22} - 1, and 2^{23} -1.

La lunghezza totale, cioè la somma delle lunghezze dei tre LFSR, è di 64 bit. Lo XOR dei bit meno significativi dei tre registri rappresenta un bit della keystream. Gli LFSR sono lunghi rispettivamente 19, 22 e 23 bit e le funzioni utilizzate sono polinomi di feedback sparsi. Ad ogni passo ciascun registro viene shiftato se il suo bit centrale concorda con il valore di maggioranza dei bit centrali dei tre registri. Per esempio:

- se i bit centrali dei tre registri sono rispettivamente 0,1 e 1, gli ultimi due registri vengono shiftati (in quanto abbiamo due presenze di 1 e una presenza di 0 quindi la maggioranza è rappresentata dai bit pari ad 1)
- se i bit centrali sono rispettivamente 0, 1 e 0, allora vengono shiftati il primo ed il terzo registro (in quanto abbiamo due presenze di 0 e una presenza di 1 quindi la maggioranza è rappresentata dai bit pari ad 0)

In definitiva, ad ogni round, vengono shiftati almeno due registri. E' anche possibile che siano shiftati tre registri: questo avviene nel caso in cui i tre bit centrali presentino tutti lo stesso valore (indipendentemente dal fatto che questo sia pari a 0 o a 1).



E' evidente che l'attaccante non è in grado di individuare quali registri siano stati shiftati e quali no, dato che si effettua lo shift di due o tre registri, ma non è possibile individuare quali (lo shift di tre registri avviene con probabilità pari ad ¼)

L'inizializzazione dei registri avviene attraverso i seguenti passi:

1) I tre registri sono resettati con tutti i bit pari a 0.
2) ciascun bit della chiave K_c (dal bit meno significativo a quello più significativo) viene posto in XOR con il bit meno significativo dei tre registri in parallelo. I registri sono poi shiftati ignorando la regola di maggioranza. Questo procedimento viene ripetuto 64 volte, ovvero per ciascun bit della chiave K_c.
3) I tre registri sono shiftati per altre 22 volte (sempre ignorando la regola di maggioranza). Questa volta viene effettuato lo XOR tra i bit successivi del numero di frame F_n (dal bit meno significativo a quello più significativo) in parallelo sui tre registri con il loro bit meno significativo.

Il contenuto dei tre registri alla fine di questo passo è lo stato iniziale del frame.

L'inizializzazione viene ripetuta per ogni burst. Successivamente l'algoritmo genera 114 bit per ognuna delle due direzioni di comunicazione (dalla MS alla BS e dalla BS alla MS), cioè in totale 228 bit. La **cifratura** avviene attraverso i seguenti passi:

- i primi 100 bit di output della keystream vengono scartati allo scopo di distribuire i bit del numero di frame in modo casuale nei tre LFSR
- vengono prodotti 114 bit di output della keystream, che vengono utilizzati per cifrare il frame dalla BS alla MS
- vengono scartati altri 100 bit di output dell keystream per nascondere la relazione tra i primi 114 bit ed i successivi 114 bit della keystream

- vengono prodotti 114 bit di output della keystream, che vengono utilizzati per decifrare il frame successivo ricevuto dalla BS

Ad ogni passo, viene prodotto un bit della keystream. L'algoritmo restituisce una keystream di 228 bit. La cifratura effettiva avviene mettendo in XOR 114 bit della keystream e 114 bit del messaggio in chiaro. Successivamente, l'algoritmo A5 viene reinizializzato con la stessa chiave K_c ed il numero del frame successivo.

3.4.5 Gli attacchi ad A5/1 e A5/2

1) Marc Briceno scoprì che nelle versioni installate dell'algoritmo A5/1 i 10 bit meno significativi dei 64, venivano impostati a 0. La complessità quindi di una ricerca esaustiva fu ridotta a $O(2^{54})$
2) Anderson and Roe proposero un attacco di tipo *divide and conquer*. Gli attacchi di questo tipo cercano di indovinare una parte della chiave segreta per poi individuare la restante parte, imponendo la prima parte come vincolo. In questo caso l'attacco si basava sul cercare di indovinare i 41 bit nei registri più corti R1 e R2 e derivare i 23 bit del registro R3 dall'output. Complessità : $O(2^{45})$
3) Golic utilizzò un attacco che richiedeva $O(2^{40})$ passi. In ogni caso ogni operazione risultava molto complicata in quanto richiedeva la risoluzione di un sistema di equazioni lineare
4) Golic propose un attacco *time-memory trade-off*. Questo tipo di attacchi presuppone di sferrare l'attacco stesso in due fasi : una fase di preprocessing in cui il crittoanalista riassume in una struttura dati ciò che egli sa del comportamento dell'algoritmo di cifratura, e una seconda fase che sferra l'attacco utilizzando la struttura dati calcolata. Golic concluse che era possibile trovare la chiave in $O(2^{22})$ prove in locazioni casuali di una tabella precedentemente calcolata con 2^{42} entry di 128 bit (64 Tb) oppure $O(2^{28})$ riducendo lo spazio richiesto a 862 Gb (richiedeva però tre ore di conversazione e quindi era irrealistico)
5) A. Biryukov, A. Shamir e D. Wagner proposero alla fine del 1999 altri attacchi con parametri accettabili (1 secondo o pochi minuti di calcolo e 2 minuti di conversazione utilizzando al massimo 4 dischi da 73 Gb per memorizzare il risultato dei passi di preprocessing)

Per quanto riguarda A5/2, in una sessione di Crypto 99, Ian Goldberg e David Wagner proposero un attacco su A5/2 che mediante la generazione di pochi bit pseudo casuali e $O(2^{16})$ passi, riusciva a rompere la cifratura. Questa versione dell'algoritmo A5, come si è già specificato, è stata progettata unicamente a scopi di esportazione. Molti studiosi, osservando il comportamento di A5/2, sono d'accordo nell'attribuire alla sua debolezza una precisa volontà politica di indebolire la sicurezza di GSM nei paesi non europei.

Del resto anche l'indebolimento della chiave di sessione individuato da Briceno in A5/1, sembrerebbe avere legami con insistenze, da parte di agenzie governative come la CIA e la NSA, a lasciare una sorta di porta aperta a scopi investigativi.

3.4.6 A5/3

Come è stato precedentemente detto, questo algoritmo è basato sul cifrario a blocchi *KASUMI* che è utilizzato anche dall'algoritmo GEA3 adottato dal GPRS e dagli algoritmi f8/f9 in UMTS. Il cifrario KASUMI si basa a sua volta su un cifrario a blocchi precedente chiamato *MISTY1* (Ohta & Matsui, RFC 2994 e, infatti, la parola KASUMI è la traduzione in giapponese della parola *misty*.
La funzione KASUMI opera su un input di 64 bit usando una chiave segreta K di 128 bit e produce un output di 64 bit (par. 4.8 per maggiori dettagli)
Il cuore dell'algoritmo A5/3 è l'applicazione di una funzione **KGCORE** che utilizza il cifrario KASUMI.

I parametri di ingresso della funzione KGCORE sono i seguenti vettori di bit :

INPUT	Commento
CA	8 bit CA[0]…CA[7]
CB	5 bit CB[0]…CB[4]
CC	32 bit CC[0]…CC[31]
CD	Un singolo bit CD[0]
CE	16 bit CE[0]…CE[15]
CK	128 bit CK[0]….CK[127]
CL	Un intero nell'intervallo $1…2^{19}$ (estremi inclusi) che specifica il numero di bit da restituire in output

Da notare il il parametro CL che identifica il numero di bit da produrre in output. Il parametro in output è un singolo vettore di bit :

OUTPUT	Commento
CO	Il numero di bit dipende da CL

L'inizializzazione della funzione viene fatta mediante queste assegnazioni :

- registro di 64 bit A = CC[0]…CC[31] CB[0]…CB[4] CD[0] 0 0 CA[0]…CA[7] CE[0]…CE[15]
- key modifier KM = 0x55555555555555555555555555555555
- il primo blocco di 64 bit della keystream KSB0 = 0

Viene poi applicata una singola operazione di KASUMI al registro A, usando come chiave CK ⊕ KM (lo XOR tra il vettore CK e il key modifier, entrambi di 128 bit):

$$A = KASUMI[\ A\ ]_{CK \oplus KM}$$

Una volta eseguita l'inizializzazione, l'algoritmo della funzione KGCORE è il seguente:

```
For n with 1 ≤ n ≤ BLOCKS {
          BLKCNT = n – 1
          KSB_n = KASUMI[ A ⊕ BLKCNT ⊕ KSB_n-1 ]_CK
}
```

Dove BLOCKS = CL/64, arrotondato all'intero superiore, indica il numero di blocchi da 64 bit che sarà necessario avere in output. BLKCNT è semplicemente un contatore.

I singoli bit da produrre in output sono estratti dai blocchi da KSB_1 a KSB_{BLOCKS} applicando il seguente ciclo indentato, che semplicemente copia i bit dai vari blocchi KSB al vettore CO :

```
For n = 1 to BLOCKS
     For i =0 to 63
          CO[((n–1)*64)+i] = KSB_n[i]
```

Naturalmente poichè questa funziona così definita produce in output solo multipli di 64 bit, nel caso in cui CL non sia multiplo di 64 bit, verranno scartati i bit meno significativi in eccesso dell'ultimo KSB.

A questo punto per definire A5/3 non resta che eseguire il mapping i parametri di input/output di A5/3 in quelli di input/output di KGCORE. Una volta ottenuto questo mapping, l'algoritmo A5/3 risulta essere una singola applicazione della funzione KGCORE.

I parametri di A5/3 di input/output sono i seguenti (quella che segue è una semplice rinominazione):

INPUT	Size (bits)	Comment
COUNT	22	Il numero di frame di 22 bit F_n COUNT[0]…COUNT[21]
K_C	KLEN	Chiave di sessione K_C[0]… K_C[KLEN-1], dove KLEN è nell'intervallo 64…128

OUPUT	Size (bits)	Comment
BLOCK1	114	Keystream bits BLOCK1[0]…BLOCK1[113]
BLOCK2	114	Keystream bits BLOCK2[0]…BLOCK2[113]

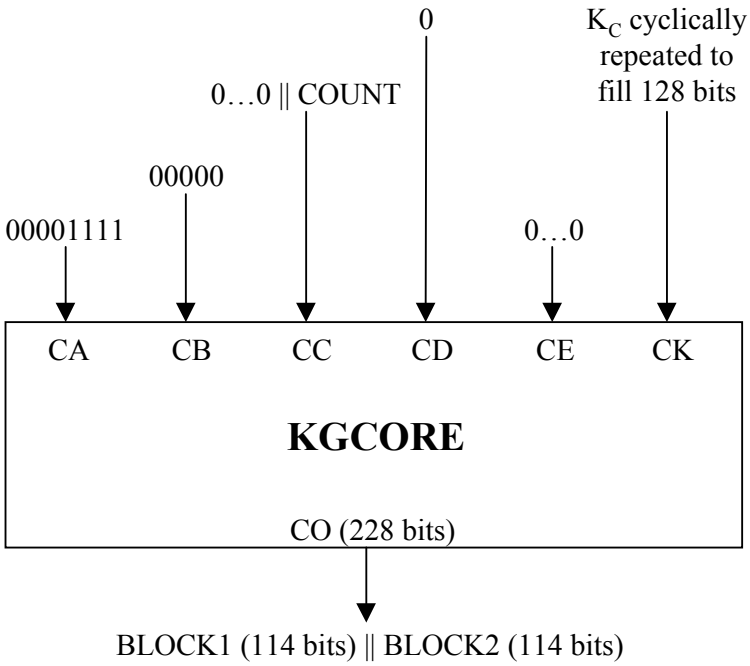
Da notare che l'algoritmo A5/3 prenderà in input (come A5/1) una chiave di sessione di 64 bit e, quindi, in realtà KLEN può valere solo 64. A partire dai parametri di input dell'algoritmo A5/3, iI parametri di input della funzione KGCORE vengono calcolati come segue:

- `CA[0]…CA[7] = 0 0 0 0 1 1 1 1`
- `CB[0]…CB[4] = 0 0 0 0 0`
- `CC[0]…CC[9] = 0 0 0 0 0 0 0 0 0 0`
- `CC[10]…CC[31] = COUNT[0]…COUNT[21]`
- `CD[0] = 0`
- `CE[0]…CE[15] = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0`
- `CK[0]…CK[KLEN-1] =` K_C`[0]…`K_C`[KLEN-1]`
- `If KLEN < 128 then`
  `CK[KLEN]…CK[127] = KC[0]…KC[127 – KLEN]`
  `(in particolare if KLEN = 64 then CK = KC || KC)`
- `CL = 228`

Il valore specificato dal parametro CL è proprio 228, ovvero il numero di bit della keystream che è necessario avere in GSM. Assegnando questi valori e applicando la funzione KGCORE ai parametri così definiti, otteniamo in output il vettore CO con lunghezza pari a 228 bit. Le due keystream di 114 bit vengono ottenute semplicemente mediante uno split in due parti uguali del parametro CO, ovvero con le seguenti assegnazioni :

- `BLOCK1[0]…BLOCK1[113] = CO[0]…CO[113]`
- `BLOCK2[0]…BLOCK2[113] = CO[114]…CO[227]`

La figura seguente visualizza il mapping dei parametri:



3.5 Considerazioni su GSM

L'algoritmo A5/3 progettato dal 3GPP dovrebbe garantire una maggiore sicurezza, in quanto fonda la sua complessità sul cifrario KASUMI che è stato ampiamente testato. Tuttavia si possono notare almeno due difetti :

- la chiave Kc viene trasformata in una chiave a 128 bit semplicemente mediante duplicazione
- il parametro CE viene posto completamente a 0

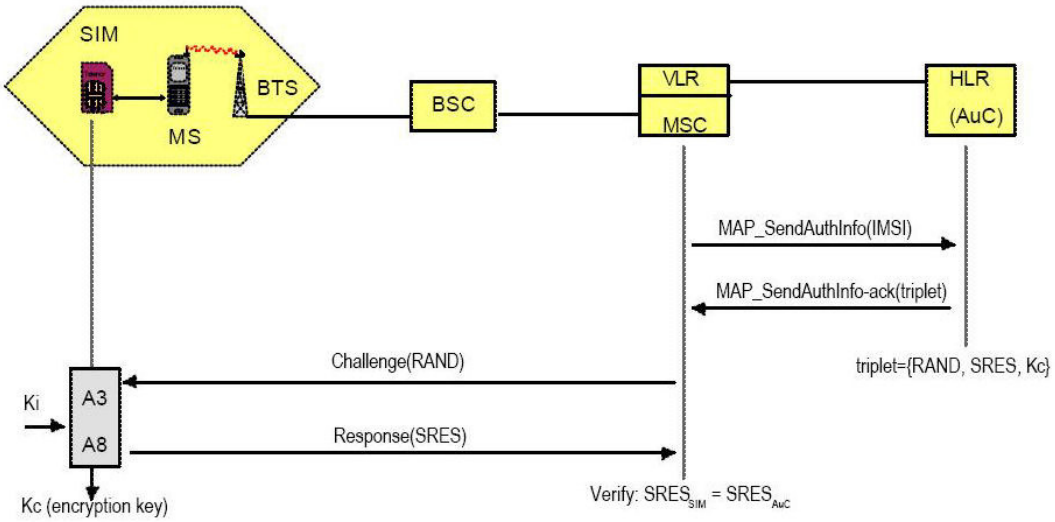
Appare sconcertante, tuttavia, il fatto che, indipendentemente dall'algoritmo utilizzato, il sistema GSM presenti almeno due falle per quanto riguarda la sicurezza dei flussi di dati vocali. Innanzitutto sarebbe possibile, utilizzando un certo tipo strumenti adeguati, impersonare una BS e imporre A5/2 come algoritmo di cifratura che come abbiamo visto è molto facile da rompere. C'è da dire anche che queste apparecchiature sono illegali, ingombranti e piuttosto costose. Inoltre è possibile attaccare dopo la BS dove i dati solitamente viaggiano in chiaro su cavi o canali satellitari. L'attaccante potrebbe non essere scoperto per un tempo molto lungo e quindi ottenere dati privati in modo tutto sommato semplice.
Infine si può osservare come decifrare una chiamata in GSM comporta non soltanto l'ascolto dei dati vocali, ma anche la localizzazione geografica dell'utente stesso.

4 UMTS

4.1 Evoluzione dal 2G…

La struttura della sicurezza nell'UMTS evoluzione dei sistemi GSM/GPRS nasce da una scelta di progetto ben precisa: individuare funzioni e opzioni per la sicurezza già presenti (legacy), punti di provata *necessità* e *robustezza* per apportare miglioramenti a tali caratteristiche ed introdurre elementi innovativi.

Viene presentato uno schema riassuntivo dell'architettura della sicurezza nel GSM (di cui si è parlato nelle sezioni precedenti)



Dallo schema vengono palesate alcune debolezze in tale architettura: solo la rete può iniziare la procedura di autenticazione, che è in un certo senso opzionale e solo l'utente viene autenticato (non c'è mutua autenticazione); l'operatore dell'HLR deve fidarsi dell'operatore del VLR/MSC (o SGSN) (nel caso di roaming, ad esempio, si potrebbe trattare di entità differenti); la protezione della confidenzialità è implementata solo al link layer (non è presente dalla bts in poi), inizializzata solo dalla rete e non verificabile dal terminale. Inoltre le chiavi di cifratura sono inerentemente limitate a 64bit (all'inizio il massimo effettivo era di 54bit) che si sono dimostrati troppo pochi per le debolezze di algoritmi come A5/1.

4.2 … al 3G

Caratteristiche fondamentali dell'architettura di sicurezza 3G/UMTS sono dunque le seguenti: un algoritmo di cifratura a 128bit (Kasumi) (anche per le keystream), una mutua autenticazione in singolo-passo (utente↔rete), un set di algoritmi di esempio per la cifratura dell'AKA e un nuovo meccanismo di integrità per i messaggi di segnalazione

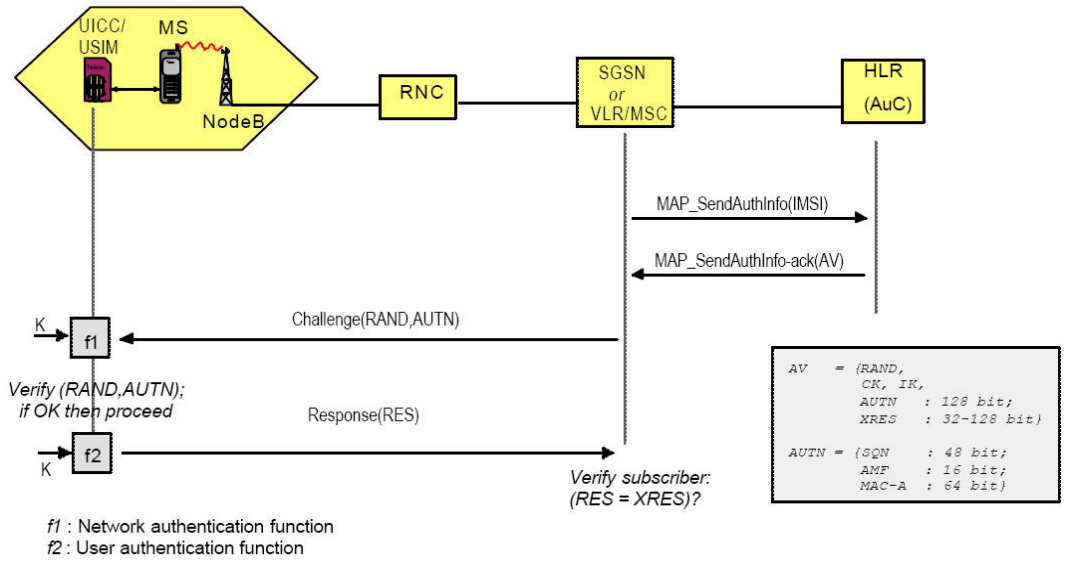
Identificazione dell'utente
Identificare l'utente vuol dire determinare il suo identificatore IMSI (International Mobile Subscriber Identity). La rete identifica l'utente o chiedendo l'IMSI direttamente all'utente o determinandolo tramite l'identificatore temporaneo TMSI (Temporary Mobile Subscriber Identity). Per motivi di sicurezza l'IMSI viene chiesto all'utente solo quando quest'ultimo non può essere identificato tramite il suo codice temporaneo TMSI. Questo per evitare che un intruso possa intercettare l'IMSI lungo la tratta radio mentre l'utente è collegato alla rete. Anche perchè l'intruso è agevolato dal fatto che l'IMSI non viene cifrato lungo la tratta radio, per cui durante la sua trasmissione, il codice IMSI è particolarmente esposto ad attacchi di intercettazione e quindi la sua mancata cifratura rappresenta un punto debole nell'ambito della riservatezza dell'identità dell'utente. Motivo per cui di norma per identificare l'utente e quindi per risalire al suo codice IMSI, si usa l'identificatore temporaneo TMSI. Il codice TMSI, contrariamente all'IMSI, viene protetto in due modi; viene cifrato mediante l'algoritmo f8 ed il suo valore viene cambiato almeno ogni volta che l'utente passa da una Location Area (LA) (o Routing Area (RA)) all'altra.

La rete distingue due identificatori temporanei, il TMSI ed il P-TMSI. L'identificatore TMSI viene generato ed assegnato all'utente dal dominio CS (Circuit Switched), cioè dal registro VLR, invece l'identificatore P-TMSI viene generato ed assegnato all'utente dal dominio PS (Packet Switched), cioè dal registro SGSN.
Il TMSI (risp. P-TMSI) viene usato dalla rete per identificare l'utente quando quest'ultimo chiede una connessione alla rete, un distacco dalla rete, un aggiornamento della locazione, ecc.
Per evitare equivoci l'identificatore TMSI (risp. P-TMSI) viene associato al numero LAI (risp. RAI) che identifica la Location Area (LA) (risp. Routing Area (RA)) nella quale l'utente è stato registrato, cioè nella quale l'utente stazionava quando il registro VLR (risp. SGSN) ha generato per lui il codice TMSI (risp. P-TMSI). L'associazione di TMSI (risp. P-TMSI) con LAI (risp. RAI) permette di identificare univocamente l'utente mediante il solo identificatore TMSI (risp. P-TMSI) quando l'utente si trova nella LA (risp. RA) identificata dal numero LAI (risp. RAI) al quale TMSI (risp. P-TMSI) è associato; al di fuori di quella LA (risp. RA) per identificare l'utente non è più sufficiente il solo TMSI (risp. P-TMSI) ma è necessario anche il numero LAI (risp. RAI) ossia è necessaria la coppia TMSI/LAI (risp. P-TMSI/RAI).

Ogni volta che l'utente passa in un'altra LA (risp. RA), il suo codice TMSI (risp. P-TMSI) viene aggiornato insieme all'identificatore LAI (risp. RAI) della locazione. Così mediante la coppia TMSI/LAI (risp. P-TMSI/RAI) è possibile identificare l'utente senza possibilità di equivoci, su tutto il territorio ed è possibile conoscere istante per istante la posizione dell'utente sul territorio.
L'aggiornamento della coppia TMSI/LAI (risp. P-TMSI/RAI) e quindi l'aggiornamento della posizione dell'utente viene ottenuto mediante l'esecuzione di un'apposita procedura di aggiornamento della locazione dell'utente.
Se la LA (risp. RA) dalla quale parte e quella nella quale arriva l'utente, sono sotto il controllo dello stesso registro VLR (risp. SGSN), la procedura di aggiornamento della locazione è la seguente (in essa, per evitare equivoci, il "vecchio" TMSI cioè quello accoppiato all'identificatore LAI della LA dalla quale l'utente proviene, è indicato con TMSIo dove "o" sta per "old" mentre il "nuovo" TMSI cioè quello accoppiato all'identificatore LAI della LA nella quale l'utente attualmente staziona, è indicato con TMSIn dove "n" sta per "new"; in maniera anologa vanno letti anche P-TMSIo, P-TMSIn, LAIo, LAIn, RAIo e RAIn).

1) Il registro VLR (risp. SGSN) manda all'UE il messaggio "user identity request" di richiesta dell'identità.
2) L'UE invia la coppia TMSIo/LAIo (risp. P-TMSIo/RAIo) al registro VLR (risp. SGSN).
3) Il registro VLR (risp. SGSN);
dalla coppia TMSIo/LAIo (risp. P-TMSIo/RAIo) ricevuta, rintraccia nel suo database il codice IMSI dell'utente;
genera un nuovo codice TMSIn (risp. P-TMSIn);
memorizza nel suo database il nuovo codice TMSIn (risp. P-TMSIn) in corrispondenza dell'IMSI dell'utente associando così i due identificatori;
invia la coppia TMSIn/LAIn (risp. P-TMSIn/RAIn) all'UE;
invia il suo *VLR number* (risp. *SGSN number*) al registro centrale HLR per comunicare a quest'ultimo presso quale VLR (risp. SGSN) è ora registrato l'UE.
4) L'UE ricevuta la coppia TMSIn/LAIn (risp. P-TMSIn/RAIn);
associa, in memoria, TMSIn (risp. P-TMSIn) al codice IMSI e rimuove l'associazione tra TMSIo (risp. P-TMSIo) e IMSI;
manda un segnale al registro VLR (risp. SGSN) per avvertirlo che la sostituzione di TMSIo (risp. P-TMSIo) con TMSIn (risp. P-TMSIn) si è conclusa con successo.
5) Il registro VLR (risp. SGSN) ricevuto il segnale dall'UE, rimuove nel database l'associazione tra TMSIo (risp. P-TMSIo) e IMSI.

Se al punto 3) il registro VLR (risp. SGSN) non trova nel suo database il codice IMSI dell'utente, lo richiede direttamente all'UE e poi la procedura prosegue normalmente.
Se invece, alla fine della procedura, il registro VLR (risp. SGSN) non riceve il segnale di successo dall'UE, il registro VLR (risp. SGSN) mantiene nel suo database l'associazione sia tra TMSIo (risp. P-TMSIo) e IMSI che tra TMSIn (risp. P-TMSIn) e IMSI e nel successivo collegamento con l'UE cercherà di capire quale tra TMSIo (risp. P-TMSIo) e TMSIn (risp. P-TMSIn) è memorizzato nell'UE per decidere così quale cancellare dal suo database. Ovviamente cancellerà il TMSI (risp. P-TMSI) che non è memorizzato nell'UE. Deciderà poi se rieseguire la procedura.
La procedura appena vista può essere considerata costituita da due procedure; la procedura di identificazione dell'utente costituita dai punti 1), 2) e 3)_1 (cioè, il passo 1 del punto 3) e la procedura di aggiornamento della coppia TMSI/LAI (risp. P-TMSI/RAI) costituita dai passi successivi.
Invece, nel caso in cui la LA (risp. RA) dalla quale parte e quella nella quale arriva l'utente, sono sotto il controllo di due diversi registri VLR (risp. SGSN), la procedura di aggiornamento della locazione è la seguente (in essa, per evitare equivoci, il "vecchio" VLR, cioè il VLR visitato precedentemente dall'utente, è indicato con VLRo dove "o" sta per "old" mentre il "nuovo" VLR, cioè il VLR che l'utente sta attualmente visitando, è indicato con VLRn dove "n" sta per "new"; in maniera analoga vanno letti anche SGSNo, SGSNn, TMSIo, TMSIn, P-TMSIo, P-TMSIn, LAIo, LAIn, RAIo e RAIn).
1) Il registro VLRn (risp. SGSNn) manda all'UE il messaggio "user identity request" di richiesta dell'identità.
2) L'UE invia la coppia TMSIo/LAIo (risp. P-TMSIo/RAIo) al registro VLRn (risp. SGSNn).
3) Il registro VLRn (risp. SGSNn) invia, al registro VLRo (risp. SGSNo), la coppia TMSIo/LAIo (risp. P-TMSIo/RAIo) con il messaggio "user identity request" di richiesta dell'identità dell'utente.
4) Il registro VLRo (risp. SGSNo) cerca nel suo database i dati dell'utente.
Se li trova, manda al registro VLRn (risp. SGSNn), insieme al messaggio di *user identity response*, il codice IMSI ed alcuni parametri riguardanti l'utente (come ad esempio alcuni vettori di autenticazione AV non usati, le chiavi CK ed IK, ecc.). Poi il registro VLRo (risp. SGSNo) cancella dal suo database alcuni di tali parametri, quali ad esempio i vettori di autenticazione AV.
Se invece l'utente non è presente nel database del registro VLRo (risp. SGSNo), quest'ultimo manda al registro VLRn (risp. SGSNn) il messaggio di *user identity response* con l'indicazione che l'utente non è stato ritrovato.
5) Il registro VLRn (risp. SGSNn);
se riceve il messaggio di *user identity response* con il codice IMSI ed eventualmente i parametri, memorizza l'IMSI ed i parametri nel suo database dopodiché la procedura prosegue con il punto 3)_2 della precedente procedura, cioè con la generazione del codice TMSIn (risp. P-TMSIn) da parte del registro VLR (risp. SGSN) e si procede fino alla fine di quella procedura con l'accortezza però di interpretare il registro VLR (risp. SGSN) come registro VLRn (risp. SGSNn);
se riceve il messaggio di *user identity response* con l'indicazione che l'utente non è stato trovato, chiede direttamente all'utente il codice IMSI dopodiché la procedura prosegue con il punto 3)_2 della precedente procedura, cioè con la generazione del codice TMSIn (risp. P-TMSIn) da parte del registro VLR (risp. SGSN) e si procede fino alla fine di quella procedura con l'accortezza però di interpretare il registro VLR (risp. SGSN) come registro VLRn (risp. SGSNn).
Se il registro VLRo (risp. SGSNo) precedentemente visitato dall'utente, non può essere contattato o non trova nel suo database il codice IMSI dell'utente, allora quest'ultimo si identifica da sé inviando al registro VLRn (risp. SGSNn) il codice IMSI.
Viene presentato di seguito uno schema riassuntivo di tale procedura di autenticazione.



4.3 Funzioni crittografiche

Il 3GPP ha ritenuto necessario fornire il sistema UMTS di alcune caratteristiche di sicurezza, realizzate mediante l'uso di funzioni ed algoritmi crittografici. Le funzioni crittografiche che vengono usate nella procedura di autenticazione descritta successivamente, sono le seguenti;

- **f0**: Funzione per la generazione del numero (pseudo) casuale RAND.
- **f1**: Funzione per l'autenticazione della rete. Genera un codice MAC (Message Authentication Code) o XMAC (Expected MAC).
- **f1***: Funzione per l'autenticazione del massaggio di ri-sincronizzazione. Genera un codice MAC-S (Message Authentication Code - Synchronisation) o XMAC-S (Expected MAC-S).
- **f2**: Funzione per l'autenticazione dell'utente. Genera un codice RES (User Response) o XRES (Expected RES).
- **f3**: Funzione per la generazione della chiave CK (Cipher Key) che viene data in input alla funzione f8
- **f4**: Funzione per la generazione della chiave IK (Integrity Key) che viene data in input alla funzione f9.
- **f5**: Funzione per la generazione della chiave AK (Anonymity Key) usata nelle normali operazioni.
- **f5***: Funzione per la generazione della chiave AK (Anonymity Key) usata nei messaggi di ri-sincronizzazione.

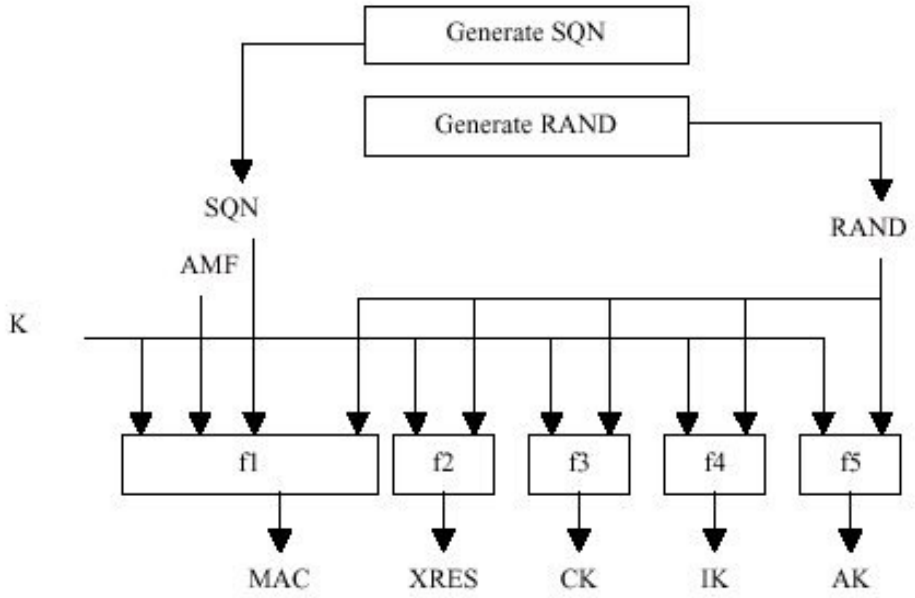
Le funzioni crittografiche che vengono usate per la cifratura del traffico utente e di alcuni messaggi di segnalazione, sono le seguenti;

- **f8**: Funzione per la riservatezza dei dati. Genera una keystream.
- **f9**: Funzione per l'integrità dei dati. Genera un codice MAC-I (Message Authentication Code - Integrity).

Il 3GPP ha stabilito la standardizzazione delle sole funzioni f8 ed f9.
La f8 è la funzione per la riservatezza dei dati (*data confidentiality*) mentre la f9 è la funzione per l'integrità dei dati (*data integrity*). Sono state appositamente progettate per l'UMTS, dai partecipanti al 3GPP. Il lavoro per la realizzazione di tali funzioni è iniziato nell'agosto 1999 ed è terminato nel

novembre dello stesso anno. Sono state pubblicate dall'ETSI, nella loro prima versione, il 23 dicembre 1999. L'ETSI ha provveduto anche alla loro standardizzazione.

Generazione dei parametri nell'AuC
Come mostra la seguente figura:



i parametri che per la procedura di autenticazione vengono generati nell'AuC, sono i seguenti;

- **RAND** (Random challenge) è un numero casuale di 128 bit generato dalla funzione f0 nel seguente modo;
  f0(stato interno) = RAND
- **SQN** (Sequence Number) è un numero di 48 bit. Come vedremo dopo, in ogni vettore di autenticazione AV creato per la procedura di autenticazione, viene inserito un numero SQN. I numeri SQN vengono usati principalmente come numeri sequenziali rispetto ai quali vengono tenuti in ordine i vettori di autenticazione AV di un certo utente, identificano perciò in maniera univoca ogni vettore AV dell'utente. L'AuC mantiene un contatore SQN_{HE} per ogni utente, perciò l'SQN_{HE} è un contatore individuale. Nell'AuC, il valore di tale contatore viene utilizzato da un'apposita procedura per generare i numeri SQN. Anche l'USIM mantiene un suo contatore. E' il contatore SQN_{MS} che contiene il più grande numero SQN che l'USIM ha accettato fino a quel momento.
- **AMF** (Authentication Management Field) è un parametro di 16 bit il cui uso non è stato standardizzato. La decisione circa il suo impiego è stata lasciata agli operatori della rete PLMN.
- **MAC** (Message Authentication Code) è un codice di 64 bit generato dalla funzione f1 nel seguente modo;
  $f1_K(SQN \,||\, RAND \,||\, AMF) = MAC$
  Cioè, la f1 con in input la chiave K ed i parametri SQN, RAND e AMF concatenati nella maniera indicata, dà MAC come output.
- **XRES** (Expected Response) è un parametro MAC (Message Authentication Code) che ha una lunghezza compresa tra 32 e 128 bit. Viene generato dalla funzione f2 nel seguente modo;
  $f2_K(RAND) = XRES$
  Cioè, la f2 con in input K e RAND dà XRES come output.
- **CK** (Cipher Key) è la chiave di 128 bit per la cifratura dei dati generata dalla funzione f3 nel seguente modo;
  $f3_K(RAND) = CK$

Cioè, la f3 con in input K e RAND dà CK come output.
La chiave CK viene data in input alla funzione per la riservatezza dei dati, f8.

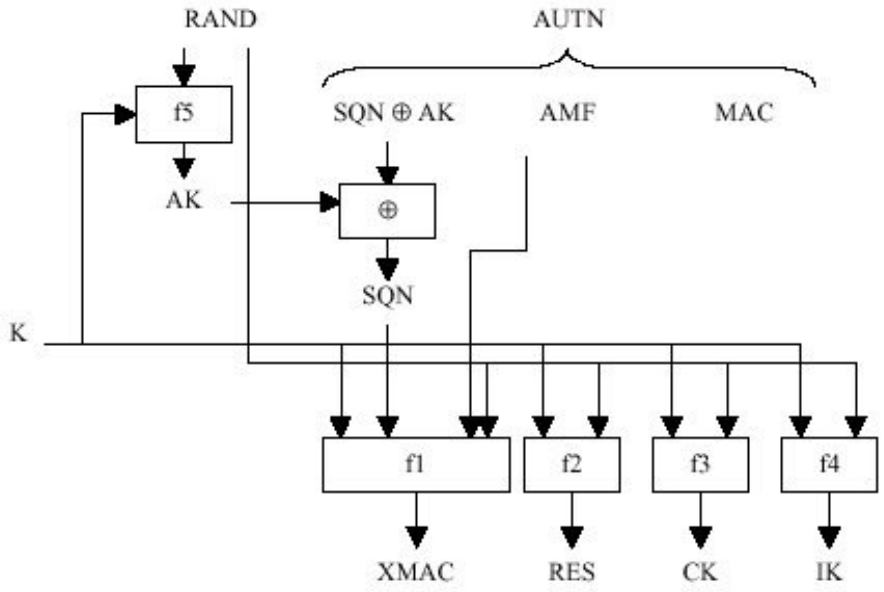
- **IK** (Integrity Key) è la chiave di 128 bit per l'integrità dei dati generata dalla funzione f4 nel seguente modo;
  $f4_K(RAND) = IK$
  Cioè, la f4 con in input K e RAND dà IK come output.
  La chiave IK viene data in input alla funzione per l'integrità dei dati, f9.
- **AK** (Anonymity Key) è una chiave di 48 bit generata dalla funzione f5 nel seguente modo;
  $f5_K(RAND) = AK$
  Cioè, f5 con in input K e RAND dà AK come output.

La chiave AK viene usata per tenere nascosto il valore del numero SQN. Infatti, all'inizio della procedura di autenticazione, l'AuC manda all'USIM un numero SQN inserendolo in un parametro chiamato AUTN, però in tale parametro AUTN per ragioni di sicurezza, non viene inserito direttamente il numero SQN ma il risultato dell'operazione di xor tra SQN ed AK.
Dopo la loro generazione, i parametri SQN, AK, AMF, MAC vengono concatenati nella seguente maniera;
AUTN = SQN xor AK || AMF || MAC
e viene così ottenuto il parametro AUTN.
Infine, dalla concatenazione dei parametri RAND, XRES, CK, IK, AUTN viene ottenuto il vettore di autenticazione AV (Authentication Vector);
AV = RAND || XRES || CK || IK || AUTN

4.4 Procedura di autenticazione

E' la procedura mediante la quale:
L'utente e la rete raggiungono la reciproca autenticazione sulla base di un segreto condiviso dall'AuC e dall'USIM. Il segreto condiviso è una chiave K lunga 128 bit che è conosciuta e disponibile solo nell'AuC e nell'USIM. Ad ogni utente viene assegnata una chiave K diversa.
Viene stabilita una nuova coppia di chiavi CK (Cipher Key) ed IK (Integrity Key) per gli algoritmi f8 e f9, rispettivamente.
E' il registro VLR/SGSN (VLR è il registro per il dominio CS (Circuit Switched) mentre SGSN è il registro per il dominio PS (Packet Switched)) ad iniziare la procedura di autenticazione dopo la prima registrazione dell'utente, dopo una richiesta di servizio (una telefonata, un collegamento ad Internet, ecc.) da parte dell'utente, dopo una richiesta di aggiornamento della locazione dell'utente, dopo una richiesta di connessione alla rete, dopo una richiesta di sconnessione dalla rete, ecc.
Un utente che staziona in una Location Area/Routing Area controllata da un registro VLR/SGSN, viene identificato da quest'ultimo nella maniera descritta successivamente. Dall'identità dell'utente, cioè dal suo identificatore IMSI, il registro VLR/SGSN individua nel suo database, l'array ordinato (rispetto ai numeri sequenziali SQN) che contiene i vettori di autenticazione AV di quel dato utente. Da tale array, il registro VLR/SGSN preleva il successivo vettore AV e manda all'utente (cioè, all'USIM) i parametri RAND e AUTN contenuti in tale vettore. Infatti il registro VLR/SGSN mantiene, per ogni utente, un array contenente i vettori di autenticazione AV e distingue tali array in base al codice IMSI dell'utente. In ogni array, i vettori di autenticazione AV sono ordinati rispetto ai numeri SQN.
Il registro VLR/SGSN, così come ogni altra entità della rete, gestisce i vettori di autenticazione AV secondo la logica FIFO (First-In / First-Out) per cui tra i vettori AV memorizzati per un dato utente nel suo database, sceglierà ed utilizzerà per la prossima procedura di autenticazione il primo che ha ricevuto dall'AuC.
Se il registro VLR/SGSN ha terminato i vettori di autenticazione AV di un dato utente, ne richiede **n** all'AuC inviando a tale entità il codice IMSI che identifica in maniera univoca l'utente. L'AuC dal

codice IMSI rintraccia nel suo database i vettori AV generati per quel dato utente e manda al registro VLR/SGSN un array ordinato (rispetto ai numeri sequenziali SQN) di **n** vettori AV. Se nel database non vi sono vettori AV per quell'utente, l'AuC ne genera **n** e poi li manda in un array ordinato (rispetto ai numeri sequenziali SQN) al registro VLR/SGSN il quale li memorizza e li utilizza per le prossime **n** procedure di autenticazione che riguardano quell'utente.
Nell'AuC i numeri SQN che fanno parte dei vettori AV, vengono generati da un'apposita procedura a partire dal valore del contatore SQN_{HE}. L'AuC mantiene un contatore SQN_{HE} per ogni utente e dal codice IMSI rintraccia il contatore SQN_{HE} dell'utente per il quale la procedura deve calcolare i numeri SQN.
L'USIM, ricevuti i parametri RAND e AUTN (con AUTN = SQN xor AK || AMF || MAC), esegue le operazioni descritte nella seguente;

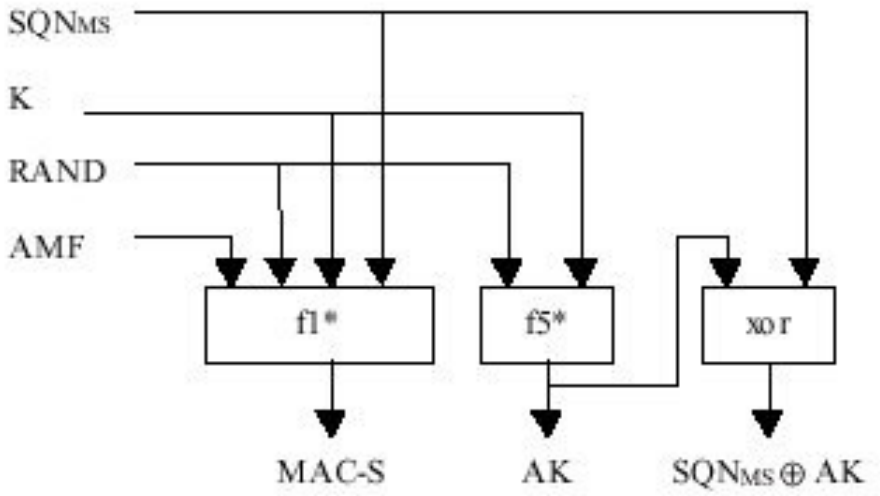


Cioè esegue i seguenti passi:
1) Con il parametro RAND che ha ricevuto e la chiave K che possiede, l'USIM determina, usando la funzione f5, la chiave AK;
$f5_K(RAND) = AK$
La chiave K è conosciuta sia dall'USIM che dall'AuC, è infatti il segreto condiviso da queste due entità.
2) Utilizzando il valore AK calcolato al passo precedente, l'USIM determina il numero SQN effettuando la seguente operazione;
SQN = (SQN xor AK) xor AK
3) L'USIM dando in input alla funzione f1 la chiave K ed i parametri SQN, RAND e AMF concatenati (il parametro AMF è contenuto nel parametro AUTN che l'USIM ha ricevuto dal registro VLR/SGSN insieme al parametro RAND), determina il codice XMAC;
$f1_K(SQN \,||\, RAND \,||\, AMF) = XMAC$
4) L'USIM confronta il codice XMAC calcolato al passo precedente, con il codice MAC contenuto nel parametro AUTN; se sono differenti, manda un messaggio di rigetto dell'autenticazione dell'utente (user authentication reject) al registro VLR/SGSN con una indicazione della causa ed abbandona la procedura di autenticazione.
Infatti se i codici XMAC e MAC sono differenti vuol dire che almeno uno dei parametri SQN, RAND, AMF calcolati nell'AuC, è diverso dal corrispondente parametro ricevuto o calcolato dall'USIM e ciò può accadere per vari motivi, ad esempio può darsi che qualcuno di quei parametri

sia stato modificato da qualche intruso o che il registro VLR/SGSN abbia mandato all'utente i parametri sbagliati a causa di una errata identificazione dell'utente.
Il registro VLR/SGSN quando riceve il messaggio di rigetto dell'autenticazione dell'utente, inizia una procedura per la fallita autenticazione inviando al registro HLR (che conserva in maniera permanente i dati dell'utente) il codice IMSI che identifica l'utente ed un codice che indica la causa del fallimento. Il registro VLR/SGSN decide poi se iniziare una nuova procedura di identificazione ed autenticazione con l'utente che l'ha abbandonata.
5) Se al passo precedente i codici XMAC e MAC sono risultati uguali, l'USIM controlla se il valore del numero SQN che ha ricevuto dal registro VLR/SGSN, non oltrepassa il limite stabilito. Vediamo come effettua tale verifica.
Nell'USIM (e nell'AuC) ogni numero SQN viene considerato costituito da due sottostringhe, SEQ e IND, concatenate;
$SQN = SEQ \,||\, IND$
dove SEQ sono i bit più significativi ed IND sono i bit meno significativi del numero SQN.
Inoltre, l'USIM mantiene un array di dimensione **a** i cui elementi;
$SEQ_{MS}(0), SEQ_{MS}(1), \ldots , SEQ_{MS}(\mathbf{a}\text{-}1)$
sono le parti più significative SEQ di **a** numeri SQN che l'USIM ha accettato in precedenza. In principio, tale array viene inizializzato a zero.
Per l'esattezza, il primo elemento $SEQ_{MS}(0)$ dell'array è la parte più significativa SEQ dell'ultimo numero SQN del tipo;
$SQN = SEQ \,||\, IND = SEQ_{MS}(0) \,||\, 0$
(cioè, del tipo SQN = SEQ || IND con IND = 0) che l'USIM ha accettato fino a quel momento.
Il secondo elemento $SEQ_{MS}(1)$ dell'array è la parte più significativa SEQ dell'ultimo numero SQN del tipo;
$SQN = SEQ \,||\, IND = SEQ_{MS}(1) \,||\, 1$
(cioè, del tipo SQN = SEQ || IND con IND = 1) che l'USIM ha accettato fino a quel momento.
Pertanto, l'i-mo elemento $SEQ_{MS}(i)$ dell'array è la parte più significativa SEQ dell'ultimo numero SQN del tipo;
$SQN = SEQ \,||\, IND = SEQ_{MS}(i) \,||\, i \qquad \text{con } i = 0, 1, \ldots , \mathbf{a}\text{-}1$
che l'USIM ha accettato fino a quel momento.
Perciò ognuno degli **a** numeri $SEQ_{MS}(i)$ dell'array proviene da un certo numero SQN accettato dall'USIM; per evitare equivoci chiameremo questi **a** numeri con SQN^*. Cioè indicheremo con;
$SQN^* = SEQ_{MS}(i) \,||\, i \qquad \text{con } i = 0, 1, \ldots , \mathbf{a}\text{-}1$
gli **a** numeri che l'USIM ha accettato e dei quali le parti più significative $SEQ_{MS}(i)$ sono memorizzate nell'array precedentemente definito.
Ovviamente, dalle parti più significative $SEQ_{MS}(i)$ memorizzate nell'array è possibile risalire facilmente ai relativi numeri SQN^* da cui provengono; infatti per ottenere l'i-mo numero SQN^* basta eseguire un'operazione di concatenazione tra l'i-mo elemento $SEQ_{MS}(i)$ dell'array e l'indice **i** di quell'elemento. Per tale motivo è possibile trattare l'array come se contenesse proprio i numeri SQN^*.
Indichiamo con;
$SQN_{MS} = SEQ_{MS} \,||\, IND_{MS}$
il più grande numero SQN che l'USIM ha accettato fino a quel momento, cioè il più grande dei numeri SQN^* dell'array. Il numero SQN_{MS} è il cosiddetto **contatore** dell'USIM.
Osserviamo inoltre che nell'AuC i numeri SQN di un dato utente non vengono generati mediante una semplice operazione del tipo;
$SQN_{j+1} = SQN_j + 1 \quad \text{con } \mathbf{j} = 1, 2, \ldots$
ma viene applicata una procedura che può generare due numeri SQN consecutivi che differiscono per più dell'unità, cioè può risultare;
$SQN_{j+1} - SQN_j > 1 \quad \text{con } \mathbf{j} \text{ intero positivo}$

Si può pertanto verificare un salto di valori tra due numeri SQN generati consecutivamente; un tale salto può verificarsi anche per altri motivi, ad esempio a seguito di un'operazione di reset del contatore SQN_{HE} (mantenuto nell'AuC) è facile che il numero SQN generato subito prima e quello generato subito dopo tale operazione differiscano per più dell'unità.
Ebbene, poichè possono verificarsi tali salti, anche l'USIM li accetta sempre che non oltrepassino un certo limite, cioè sempre che non siano arbitrariamente grandi. Infatti, l'USIM rifiuta il numero SQN = SEQ || IND che ha ricevuto dal registro VLR/SGSN e quindi abbandona la procedura di autenticazione, quando la differenza tra la parte più significativa SEQ di tale numero e la parte più significativa SEQ_{MS} del contatore SQN_{MS} risulta maggiore o uguale di un limite **d** opportunamente stabilito in precedenza, cioè quando risulta;
$SEQ - SEQ_{MS} \geq \mathbf{d}$
Il limite **d** è stato stabilito per proteggere il contatore SQN_{MS} dell'USIM da attacchi diretti a forzarlo, cioè da attacchi che costringono il contatore dell'USIM ad assumere dei valori che lo fanno avanzare in maniera "circolare" (wrap around) nel suo intervallo dei valori (con il termine "circolare" si intende che il contatore dopo aver raggiunto il valore massimo riassume il valore minimo, cioè lo zero, per poi eventualmente riavanzare verso il valore massimo; per cui il valore minimo ricopre il ruolo di successore del valore massimo). Per evitare che questo accada è stato introdotto, come detto, il parametro **d**.
L'USIM effettua un'ulteriore verifica sul numero SQN che ha ricevuto dal registro VLR/SGSN, infatti verifica anche se quel numero SQN è "fresh"; cioè si accerta se quel dato numero non è mai stato usato in precedenza.
Prima di vedere come viene effettuata tale verifica ricordiamo che ogni vettore di autenticazione AV generato nell'AuC contiene un numero SQN = SEQ || IND e precisiamo che due vettori AV consecutivi, cioè generati nell'AuC uno dopo l'altro e per uno stesso utente, contengono valori IND che sono consecutivi in $Z_a = \{0, 1, \ldots , \mathbf{a}\text{-}1\}$, così se il primo contiene il valore IND, il successivo conterrà il valore IND + 1 mod **a** , dove **a** è la dimensione dell'array precedentemente definito.
Infatti la regola generale prevede che per un dato utente i valori consecutivi degli indici IND vengano generati mediante l'operazione;
$IND_{j+1} = IND_j + 1 \bmod \mathbf{a} \quad \text{con } \mathbf{j} = 1, 2, \ldots$
cioè, l'indice IND_{j+1} successivo viene ottenuto incrementando di 1 modulo **a** l'indice IND_j precedente. Tuttavia, quando sono disponibili informazioni aggiuntive è possibile stabilire delle eccezioni a tale regola generale, come ad esempio la possibilità che in ogni vettore di autenticazione che l'AuC genera per uno stesso lotto, l'indice IND abbia lo stesso valore o la possibilità di definire due insiemi di valori, uno per gli indici IND da inserire nei vettori AV generati per i servizi PS (Packet Switched) e l'altro per gli indici IND da inserire nei vettori AV generati per i servizi CS (Circuit Switched).
In ogni caso, il valore dell'indice IND varia sempre da 0 ad **a**-1 (così, ad esempio se è stato fissato **a** = 32, l'indice IND varierà da 0 a 31 ed allora IND avrà una lunghezza di 5 bit mentre i restanti 43 bit più significativi del numero SQN = SEQ || IND conterranno il valore di SEQ).
L'AuC genera gli indici IND a partire dal valore IND_{HE} che è la parte meno significativa del contatore $SQN_{HE} = SEQ_{HE} \,||\, IND_{HE}$ che l'AuC rintraccia nella sua memoria mediante l'identificatore individuale IMSI dell'utente.
L'indice IND che fa parte di ogni numero SQN = SEQ || IND viene usato come puntatore all'array;
$SEQ_{MS}(0), SEQ_{MS}(1), \ldots , SEQ_{MS}(\mathbf{a}\text{-}1)$
precedentemente definito, ossia come puntatore all'elemento $SEQ_{MS}(IND)$ di tale array.
L'USIM per verificare se il numero SQN = SEQ || IND che ha ricevuto dal registro VLR/SGSN è "fresh", lo confronta con il numero $SQN^* = SEQ_{MS}(i) \,||\, i$ che ha lo stesso indice **i** = IND, cioè confronta i due numeri;
$SQN = SEQ \,||\, IND \quad e \quad SQN^* = SEQ_{MS}(i) \,||\, i \quad \text{con } \mathbf{i} = IND$
In realtà, dato che le parti meno significative di questi due numeri sono uguali (infatti, come detto è **i** = IND), l'USIM si limita a confrontare solo le loro parti più significative SEQ e $SEQ_{MS}(i)$.

L'USIM non ha difficoltà ad eseguire tale confronto perchè la seconda di tali parti più significative, cioè $SEQ_{MS}(i)$, è l'elemento dell'array puntato dall'indice **i** = IND appartenente al numero SQN = SEQ || IND che l'USIM ha ricevuto dal registro VLR/SGSN. Perciò l'USIM è in grado di individuare facilmente, mediante tale indice **i** = IND, la parte più significativa $SEQ_{MS}(i) = SEQ_{MS}(IND)$ ed è quindi in grado di eseguire facilmente il predetto confronto.
Dal confronto tra SEQ e $SEQ_{MS}(i)$;
se SEQ risulta maggiore di $SEQ_{MS}(i)$, l'USIM accetta il numero SQN come "fresh" e memorizza SEQ in $SEQ_{MS}(i)$;
se SEQ risulta minore o uguale di $SEQ_{MS}(i)$, l'USIM manda al registro VLR/SGSN un messaggio di fallita sincronizzazione (synchronisation failure) contenente il numero SQN_{MS}, cioè il più grande numero dell'array e poi abbandona la procedura di autenticazione.
Inoltre, l'USIM può utilizzare un parametro L per controllare la differenza tra il contatore SQN_{MS} ed il numero SQN in esame. In tal caso, in aggiunta alle condizioni precedenti, il numero SQN sarà accettato dall'USIM solo se risulterà;
$SQN_{MS} - SQN < L$.
dove SQN_{MS} è come sempre, il più grande numero dell'array precedentemente definito.
Il parametro L può essere anche usato per risparmiare spazio. Infatti nell'array precedentemente definito, invece dei numeri $SEQ_{MS}(i)$ possono essere memorizzate le differenze $SQN_{MS} - SQN^*$ (dove $SQN^* = SEQ_{MS}(i)$ || i è un numero che l'USIM ha accettato). L'unico numero che deve essere necessariamente memorizzato integralmente nell'array è SQN_{MS}, cioè il più grande dei numeri SQN che l'USIM ha accettato fino a quel momento.
Il compito di stabilire il valore del parametro L è stato lasciato all'operatore della rete PLMN questo perchè il valore scelto per L ha effetto solo nell'USIM e non influisce sulla scelta dei valori per gli altri parametri. Perciò l'operatore che intende utilizzare il parametro L può fissare il suo valore in base alla strategia di sicurezza che intende adottare.
In ogni caso, prima che l'USIM abbandoni la procedura (o perchè il numero SQN che ha ricevuto dal registro VLR/SGSN supera il limite **d** o perchè tale numero non è "fresh") manda un messaggio di fallita sincronizzazione (synchronisation failure) al registro VLR/SGSN. Tale messaggio contiene il parametro AUTS costruito come indicato nella seguente figura:



e così definito;

AUTS = SQN_{MS} xor AK || MAC-S

dove SQN_{MS} è come sempre, il più grande dei numeri SQN che l'USIM ha accettato fino a quel momento e dove i parametri AK e MAC-S vengono calcolati dall'USIM, rispettivamente, con le funzioni $f5^*$ ed $f1^*$, nel seguente modo;
$f5^*_K(RAND) = AK$
$f1^*_K(SQN_{MS} \| RAND \| AMF) = MAC\text{-}S$
dove RAND è il parametro calcolato nell'AuC mentre il valore del parametro AMF non è quello che l'USIM ha ricevuto dal registro VLR/SGSN ma è semplicemente il valore nullo. Viene usato questo valore fittizio per il parametro AMF perchè così non è necessario trasmetterlo con il messaggio di fallita sincronizzazione al registro VLR/SGSN.
Il registro VLR/SGSN quando riceve dall'USIM il messaggio di fallita sincronizzazione (synchronisation failure), inizia con l'AuC la procedura di ri-sincronizzazione.
6) Se l'USIM ritiene valido il numero SQN, calcola con la funzione f2 il parametro RES;
$f2_K(RAND) = RES$
(dove RAND è il parametro che l'USIM ha ricevuto dal registro VLR/SGSN) e poi manda al registro VLR/SGSN una risposta sull'autenticazione dell'utente (user authentication response) contenente anche il parametro RES appena calcolato.
7) Infine l'USIM calcola le chiavi CK (Cipher Key) ed IK (Integrity Key) rispettivamente con le funzioni f3 ed f4, nel seguente modo;
$f3_K(RAND) = CK$; $f4_K(RAND) = IK$
dove RAND è il parametro che l'USIM ha ricevuto dal registro VLR/SGSN.
Il registro VLR/SGSN quando riceve la risposta sull'autenticazione dell'utente (user authentication response) mandatagli dall'USIM, confronta il codice RES contenuto in tale messaggio, con il codice XRES che fa parte del vettore di autenticazione AV che contiene anche i parametri RAND e AUTN che il registro VLR/SGSN ha mandato all'USIM quando ha avuto inizio la procedura di autenticazione. Se RES ed XRES risultano uguali, allora l'autenticazione dell'utente viene accettata ed il registro VLR/SGSN adotta le chiavi CK ed IK contenute nel vettore di autenticazione AV, come chiavi per gli algoritmi f8 ed f9.
Perciò, l'AuC e l'USIM determinano, ognuno per proprio conto, le chiavi CK ed IK utilizzando la chiave segreta K da essi condivisa. Infatti, dopo la reciproca autenticazione, la rete (cioè, il registro VLR/SGSN) e l'utente sono sicuri che entrambi useranno la stessa chiave K e quindi che entrambi otterranno come output delle funzioni f3 ed f4, le stesse chiavi CK ed IK. Il trasferimento di tali chiavi CK ed IK da un'entità all'altra, viene così evitato.
Se invece RES e XRES risultano differenti, il registro VLR/SGSN inizia una procedura per la fallita autenticazione inviando al registro HLR (che conserva in maniera permanente i dati dell'utente) il codice IMSI che identifica l'utente ed un codice che indica la causa del fallimento. Il registro VLR/SGSN decide poi se iniziare una nuova procedura di identificazione ed autenticazione con l'utente che l'ha abbandonata.
Quando la procedura di autenticazione si conclude con esito positivo, le chiavi CK ed IK concordate durante tale procedura tra la rete e l'utente, vengono date in input alle funzioni f8 ed f9 che così possono essere impiegate per proteggere il traffico utente (telefonate, posta elettronica, ecc.) e diversi messaggi di segnalazione.

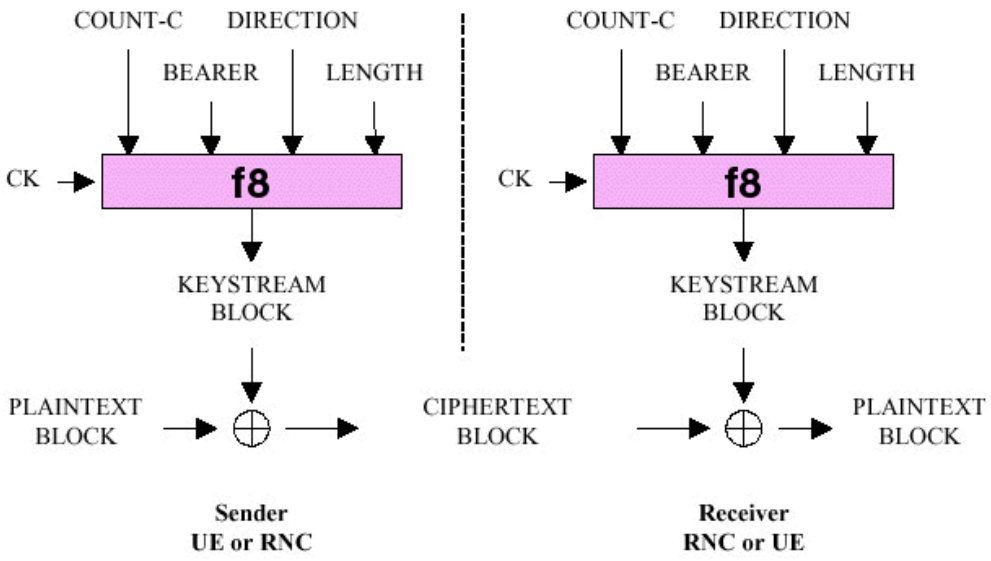
4.5 Procedura di ri-sincronizzazione

Mediante questa procedura vengono sincronizzati i contatori SQN_{HE} e SQN_{MS} (riferiti allo stesso utente) mantenuti rispettivamente nell'AuC e nell'USIM.
Il registro VLR/SGSN manda all'AuC, insieme al codice IMSI che identifica l'utente, i parametri AUTS e RAND.
Dove;
$AUTS = SQN_{MS}$ xor $AK \| MAC\text{-}S$

è il parametro calcolato ed inviato dall'USIM al registro VLR/SGSN mentre RAND è il parametro che all'inizio della procedura di autenticazione il registro VLR/SGSN spedì all'USIM insieme al parametro AUTN.
L'AuC dal codice IMSI risale al contatore SQN_{HE} dell'utente ed esegue i seguenti passi.
1) L'AuC, utilizzando la funzione $f5^*$, il parametro RAND e la chiave segreta K, con le seguenti due operazioni;
$f5^*_K(RAND) = AK$
$(SQN_{MS}$ xor $AK)$ xor $AK = SQN_{MS}$
estrae il numero SQN_{MS} dal parametro AUTS che ha ricevuto.
2) L'AuC utilizza il valore del contatore SQN_{HE} per generare un nuovo numero SQN e verifica poi se tale numero SQN ha un valore che l'USIM accetterebbe. Infatti, se il numero venisse accettato dall'USIM vorrebbe dire che il valore del contatore SQN_{HE} è corretto perchè da esso si è ottenuto un numero SQN valido.
3) Se il valore del contatore SQN_{HE} è corretto, l'AuC salta al passo 6, altrimenti continua con il passo 4.
4) L'AuC, utilizzando la funzione $f1^*$, la chiave segreta K ed i parametri SQN_{MS} e RAND ricevuti dal registro VLR/SGSN, calcola il codice XMAC-S;
$f1^*_K(SQN_{MS} \,||\, RAND \,||\, AMF) = XMAC\text{-}S$
dove AMF contiene un valore fittizio, lo zero. Confronta poi, per controllare l'integrità dei parametri SQN_{MS} e RAND, il codice XMAC-S con il codice MAC-S contenuto nel parametro AUTS che ha ricevuto dal registro VLR/SGSN. Se i due codici risultano uguali vuol dire che i parametri SQN_{MS} e RAND sono integri, cioè non sono stati modificati ad esempio da qualche intruso.
5) Se i due codici XMAC-S e MAC-S sono risultati uguali, vuol dire che a contenere il valore sbagliato è proprio il contatore SQN_{HE}. L'AuC, allora, resetta il contatore SQN_{HE} al valore del contatore SQN_{MS} cioè copia il valore del contatore SQN_{MS} in SQN_{HE}.
6) L'AuC risponde al registro VLR/SGSN inviandogli un nuovo blocco di vettori di autenticazione AV. Se il contatore SQN_{HE} non è stato resettato (perchè sono stati saltati i passi 4 e 5), l'AuC prende i vettori di autenticazione AV da quelli già memorizzati nel suo database, se invece il contatore è stato resettato (al passo 5), l'AuC genera con il nuovo valore del contatore, dei nuovi vettori di autenticazione AV e manda questi al registro VLR/SGSN.
Il registro VLR/SGSN insieme ai vettori di autenticazione AV, riceve dall'AuC un messaggio dal quale capisce se quei vettori sono di nuova generazione. Se lo sono, cancella dalla sua memoria i vecchi vettori AV ed utilizza quelli nuovi nelle successive procedure di autenticazione.

4.6 f8 - algoritmo per la riservatezza dei dati (confidentiality algorithm)

La riservatezza (confidentiality) è la proprietà dell'informazione di non essere disponibile ad entità non autorizzate.
I dati dell'utente ed alcuni dati di segnalazione sono considerati sensibili, è necessario perciò renderli riservati. Nell'UMTS il meccanismo per la riservatezza dei dati impiega la funzione di cifratura f8.
Il compito di tale funzione è quello di cifrare il traffico utente (traffico telefonico, e-mail, ecc.) e diversi dati di segnalazione (quali, ad esempio, i dati che l'UE ed il registro VLR/SGSN si scambiano quando deve essere verificata l'identità dell'utente). Perciò la f8 viene impiegata in quei protocolli in cui è necessario proteggere i dati dell'utente e/o i dati di segnalazione.
Mediante tale funzione viene generata una keystream che viene poi utilizzata per cifrare i dati. La sottostante figura mostra l'uso dell'algoritmo f8 per cifrare un testo in chiaro mediante l'impiego della keystream.



Viene eseguita un'operazione di xor tra la keystream ed il testo in chiaro; il risultato è un testo cifrato. Dal testo cifrato si può riottenere il testo in chiaro generando la stessa keystream mediante l'utilizzo degli stessi parametri di input ed applicando un'operazione di xor tra la keystream ed il testo cifrato.
In realtà l'algoritmo f8 genera piccoli blocchi che concatenati costituiscono la keystream. Ogni blocco della keystream viene usato per cifrare un solo frame. Perciò la sequenza dei dati viene cifrata mediante una concatenazione di piccoli blocchi della keystream.
Dalla figura notiamo che se l'operazione di cifratura avviene nell'UE (User Equipment) allora l'operazione di decifratura avviene nell'RNC (Radio Network Controller), altrimenti accade il contrario.
La funzione f8 è memorizzata nell'UE (User Equipment) e nell'RNC (Radio Network Controller).

4.6.1 Parametri di input

I parametri in input all'algoritmo f8 sono i seguenti.
CK (Cipher Key)
La chiave di cifratura CK è lunga 128 bit.
Nel caso in cui è necessario ridurre la lunghezza k della chiave a meno di 128 bit, i bit più significativi costituiranno la chiave CK mentre i restanti bit meno significativi ripeteranno il valore della chiave secondo la formula;
CK[n] = CK[n mod k] per ogni **n** = k, k+1, ... ,127
Vi può essere una chiave CK per le connessioni di tipo CS (cioè una chiave CK_{CS}) stabilite tra il dominio del servizio CS e l'utente, e una chiave CK per le connessioni di tipo PS (cioè una chiave CK_{PS}) stabilite tra il dominio del servizio PS e l'utente.
La chiave CK viene generata nell'HLR/AuC e viene memorizzata nell'USIM. Una copia su richiesta dell'ME (Mobile Equipment), viene mandata dall'USIM all'ME dove viene memorizzata. Un meccanismo garantisce l'utilizzo di una particolare chiave CK solo per un periodo di tempo limitato; questo per evitare che il valore della chiave CK possa essere determinato e quindi per evitare possibili attacchi. Tale meccanismo contenuto nell'USIM limita la quantità di dati che una particolare chiave CK può proteggere. Il meccanismo è semplice; all'USIM vengono comunicati, durante una connessione, i valori $START_{CS}$ e $START_{PS}$ (il primo riferito alle connessioni di tipo CS, il secondo a quelle di tipo PS). L'USIM memorizza questi due valori e li incrementa durante le successive connessioni. Quando uno di tali valori raggiunge un valore limite THRESHOLD, una nuova chiave CK viene generata.

L'ME (Mobile Equipment) cancella la chiave CK dalla sua memoria quando viene spento o quando la CK viene rimossa dall'USIM.

BEARER

L'identificatore BEARER è lungo 5 bit.

E' l'identificatore della portante radio (radio bearer) usata per la connessione. (Una portante radio, radio bearer, è un canale logico).

La stessa chiave CK può essere usata per le differenti portanti radio che sono simultaneamente associate allo stesso utente. Allora per evitare che la stessa keystream venga usata per cifrare su più di una portante, è stato posto BEARER come parametro di input, in tal modo la keystream viene generata in base all'identità della portante radio.

DIRECTION

L'identificatore DIRECTION è lungo 1 bit.

Viene posto uguale a 0 per i messaggi inviati dall'UE (User Equipment) all'RNC (Radio Network Controller) e viene posto uguale ad 1 per i messaggi inviati dall'RNC all'UE.

La stessa chiave CK può essere usata per i differenti canali simultaneamente associati allo stesso UE (User Equipment); alcuni di tali canali trasmettono i dati in una direzione mentre altri trasmettono nella direzione opposta. Allora per evitare che la stessa keystream venga usata sia per cifrare i messaggi trasmessi dall'UE verso l'RNC che per cifrare i messaggi trasmessi nella direzione opposta, è stato introdotto il parametro DIRECTION. In tal modo la keystream viene generata in base alla direzione della trasmissione.

LENGTH

L'indicatore LENGTH è lungo 16 bit.

Per un fissato canale ed una fissata direzione di trasmissione, la lunghezza del blocco del testo in chiaro (da cifrare) che viene trasmesso, può variare. E' necessario perciò generare di volta in volta, una keystream di lunghezza appropriata. Il parametro LENGTH indica proprio quanto lunga deve essere generata la keystream.

E' stato stabilito che LENGTH contenga un valore compreso tra 1 e 5114.

COUNT-C

Il numero della sequenza di cifratura COUNT-C è lungo 32 bit.

E' un contatore di frame composto di due parti; una sequenza "corta" ed una sequenza "lunga". La sequenza "corta" è costituita dai bit meno significativi mentre la sequenza "lunga" è costituita dai bit più significativi di COUNT-C. Le dimensioni della sequenza "corta" e "lunga", il valore e l'aggiornamento di COUNT-C dipendono dal tipo di trasmissione.

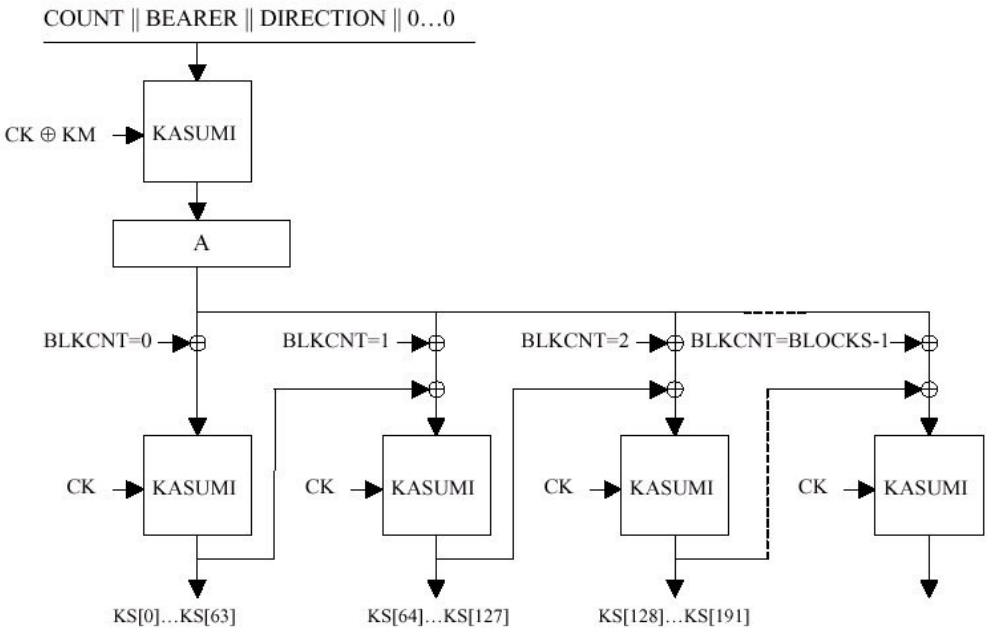
[Ad esempio, se la trasmissione è di tipo RLC AM (Radio Link Control Acknowledged Mode), la sequenza "corta" è di 12 bit e contiene l'RLC SN (RLC Sequence Number) che fa parte della testa dell'RLC AM PDU (Radio Link Control Acknowledged Mode Protocol Data Unit), invece la sequenza "lunga" è di 20 bit e contiene l'RLC AM HFN (Radio Link Control Acknowledged Mode Hyper Frame Number)].

4.6.2 Algoritmo f8

L'algoritmo f8 è basato sull'algoritmo KASUMI (che a sua volta deriva dall'algoritmo MISTY che è un algoritmo block cipher, giapponese).

f8 è uno stream cipher usato per cifrare/decifrare blocchi di dati lunghi tra 1 e 5114 bit. Usa KASUMI per generare una keystream costituita da un numero di bit pari ad un multiplo di 64.

Nella seguente figura viene mostrato lo schema dell'algoritmo f8. Da tale figura si può notare che f8 è una variante dello standard Output Feedback Mode (OFB).



Inizializzazione

I parametri COUNT, BEARER e DIRECTION vengono concatenati;
COUNT || BEARER || DIRECTION || 00 ... 0
la stringa risultante viene memorizzata in un registro A lungo 64 bit, cioè;
A = COUNT[0] ... COUNT[31] BEARER[0] ... BEARER[4] DIRECTION[0]00 ... 0
Essendo A lungo 64 bit, gli ultimi 26 bit (i bit meno significativi) vengono posti uguali a zero.
Un contatore di blocchi BLKCNT lungo 64 bit viene inizializzato a zero, cioè;
BLKCNT = 00 ... 0
(dove 00 ... 0 è la stringa binaria nulla di 64 bit).
Una costante KM lunga 128 bit viene inizializzata con la stringa binaria 010101 ...01 (lunga 128 bit), cioè;
KM = 010101 ... 01
KM viene usata per modificare la chiave CK anche essa lunga 128 bit. Si ottiene tale modifica effettuando un'operazione di xor tra CK e KM, cioè eseguendo l'operazione;
CK xor KM
Il primo blocco KSB_0 di 64 bit della keystream, viene inizializzato a zero, cioè;
KSB_0 = 00 ... 0
(dove 00 ... 0 è la stringa binaria nulla di 64 bit).
L'algoritmo KASUMI con in input la chiave CK xor KM (cioè la chiave CK modificata da KM tramite l'operazione di xor), viene applicato al registro A; l'output viene memorizzato nel registro A, cioè;
$A = KASUMI\ [A]_{CK\ xor\ KM}$

Generazione della keystream

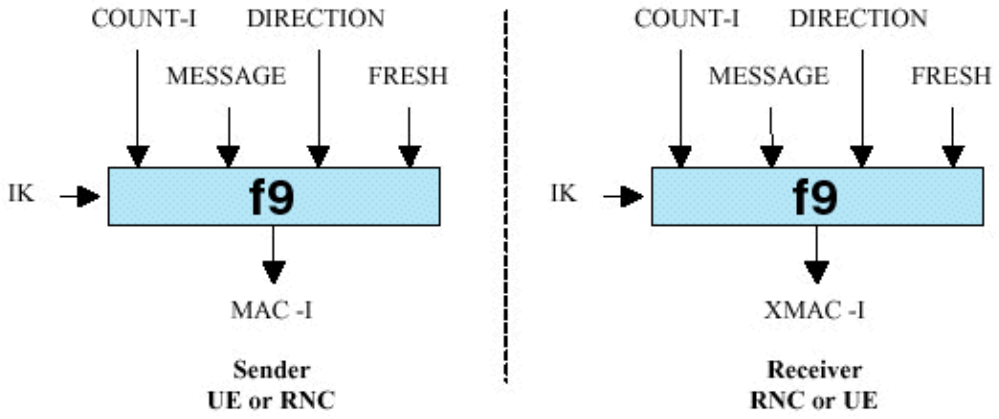
La lunghezza del testo-in-chiaro/testo-cifrato che deve essere cifrato/decifrato è uguale a LENGTH bit (cioè varia tra 1 e 5114 bit) mentre il generatore della keystream produce una keystram costituita da un numero di bit multiplo di 64. Allora per fare in modo che la keystream abbia lunghezza LENGTH, dall'ultimo blocco della keystream viene scartato, a seconda di quanto è lunga la lunghezza LENGTH del testo-in-chiaro/testo-cifrato da cifrare/decifrare, un numero di bit (i meno significativi) che varia tra 0 e 63.
Nella variabile intera BLOCKS viene memorizzato il numero LENGTH/64 arrotondato al più piccolo intero che è maggiore o uguale di LENGTH/64 (cioè, la parte alta di LENGTH/64).

Per esempio, se è LENGTH = 128 allora è BLOCKS = 2; se è LENGTH = 129 allora è BLOCKS = 3.
Per generare ogni blocco KSB_n di 64 bit della keystream, viene eseguita la seguente operazione:
Per ogni intero **n** = 1, 2, ... , BLOCKS si calcola;
$KSB_n = KASUMI[A \text{ xor } BLKCNT \text{ xor } KSB_{n-1}]_{CK}$
dove; BLKCNT = n-1.
Cioè, l'algoritmo KASUMI prende come input, sotto il controllo della chiave CK, la stringa di 64 bit ottenuta mediante un'operazione di xor tra A, BLKCNT e KSB_{n-1} ; l'output è l'n-mo blocco KSB_n di 64 bit della keystream.
Da tale operazione si vede che l'algoritmo KASUMI viene utilizzato nella modalità output-feedback, infatti l'output, costituito dal blocco KSB_{n-1} (della keystream), prodotto da KASUMI nel precedente passo n-1, diventa un input per KASUMI al passo successivo e viene perciò utilizzato per calcolare il successivo blocco KSB_n della keystream. Viene poi eseguita la seguente operazione:
Per **n** = 1, 2, ... , BLOCKS e per ogni intero **i** = 0, 1, ... , 63 si calcola;
$KS[((n-1) \cdot 64) + i] = KSB_n[i]$
Mediante tale operazione vengono estratti i bit dai blocchi KSB_1, KSB_2, ... , KSB_{BLOCKS} (ciascuno di 64 bit) e tali bit vengono memorizzati nelle variabili KS[0], KS[1], ... , KS[63], KS[64], ...
Da ciascun blocco KSB_n i bit vengono estratti a partire da quello più significativo.
I bit KS[0], KS[1], ... , KS[63], KS[64], ... ottenuti, costituiscono la keystream.

Cifratura / Decifratura
Le operazioni di cifratura e decifratura sono identiche e consistono in un'operazione di xor tra il testo da cifrare o il testo da decifrare e la keystream KS.
Cioè, per ogni **i** con **i** = 0, 1, ... LENGTH-1 si definisce;
OBS[i] = IBS[i] xor KS[i]
dove LENGTH è la lunghezza del testo da cifrare o del testo da decifrare, IBS[i] è l'i-mo bit del testo da cifrare o del testo da decifrare, KS[i] è l'i-mo bit della keystream, OBS[i] è l'i-mo bit che si ottiene come risultato dell'operazione di xor tra i bit IBS[i] e KS[i].

4.7 f9 - algoritmo per l'integrità dei dati (data integrity algorithm)

L'integrità dei dati è la proprietà dei dati di non poter essere alterati in maniera non autorizzata.
La maggior parte dei dati di segnalazione che viaggiano tra l'UE (User Equipment) e l'RNC (Radio Network Controller) sono sensibili ed è necessario perciò proteggere la loro integrità.
L'algoritmo f9 viene usato per autenticare l'integrità di un messaggio di segnalazione. Il suo impiego è perciò, in tutti quei protocolli nei quali è necessario proteggere l'integrità dei dati di segnalazione che si scambiano l'UE e l'RNC.
Non viene invece protetta, sulle portanti radio (cioè lungo la tratta radio), l'integrità dei dati dell'utente.
La figura seguente mostra che la funzione f9 genera un codice di autenticazione dei massaggi denominato MAC-I (Message Authentication Code - Integrity).



Il codice MAC-I viene aggiunto al messaggio (o dato) di segnalazione MESSAGE quando quest'ultimo viene inviato sui collegamenti radio. Il ricevente calcola il codice XMAC-I utilizzando gli stessi parametri che sono stati utilizzati per calcolare MAC-I da chi ha mandato il messaggio MESSAGE; in particolare il ricevente utilizza come parametro di input anche il messaggio MESSAGE che ha ricevuto. Una volta calcolato XMAC-I, il ricevente lo confronta con il codice MAC-I che ha ricevuto insieme al messaggio; se i due codici coincidono vuol dire che il messaggio MESSAGE non ha subito alterazioni.
Dalla figura precedente notiamo che se MAC-I viene calcolato nell'UE (User Equipment), allora il codice XMAC-I viene calcolato nell'RNC (Radio Network Controller) e viceversa.
La funzione f9 è memorizzata nell'UE (User Equipment) e nell'RNC (Radio Network Controller).

4.7.1 Parametri di input

I parametri di input all'algoritmo f9 sono i seguenti.
IK (Integrity Key)
La chiave IK per l'integrità dei dati è lunga 128 bit.
Vi può essere una chiave IK per le connessioni di tipo CS (cioè una chiave IK_{CS}) stabilite tra il dominio del servizio CS e l'utente, e una chiave IK per le connessioni di tipo PS (cioè una chiave IK_{PS}) stabilite tra il dominio del servizio PS e l'utente.
La chiave IK viene generata nell'HLR/AuC e viene memorizzata nell'USIM. Una copia su richiesta dell'ME (Mobile Equipment), viene mandata dall'USIM all'ME dove viene memorizzata. Un meccanismo garantisce l'utilizzo di una particolare chiave IK solo per un periodo di tempo limitato; questo per evitare che il valore della chiave IK possa essere determinato e quindi per evitare possibili attacchi. Tale meccanismo contenuto nell'USIM limita la quantità di dati che una particolare chiave IK può proteggere. Il meccanismo è semplice; all'USIM vengono comunicati, durante una connessione, i valori $START_{CS}$ e $START_{PS}$ (il primo riferito alle connessioni di tipo CS, il secondo a quelle di tipo PS). L'USIM memorizza questi due valori e li incrementa durante le successive connessioni. Quando uno di tali valori raggiunge un valore limite THRESHOLD, una nuova chiave IK viene generata.
L'ME (Mobile Equipment) cancella la chiave IK dalla sua memoria quando viene spento o quando la IK viene rimossa dall'USIM.
FRESH
E' un parametro lungo 32 bit.
Per ogni utente vi è un parametro FRESH.
La stessa chiave IK può essere usata per diverse connessioni consecutive. Questo parametro protegge la rete contro i tentativi da parte dell'utente di replicare i messaggi di segnalazione. Quando la connessione è attivata, l'RNC (Radio Network Controller) genera un valore casuale FRESH e lo manda all'utente. Il parametro FRESH viene poi usato sia dalla rete che dall'utente per

tutta la durata di una singola connessione. Questo meccanismo assicura la rete che l'utente non ha replicato qualche vecchio codice MAC-I.

DIRECTION

L'identificatore DIRECTION è lungo 1 bit.

Viene posto uguale a 0 per i messaggi inviati dall'UE (User Equipment) all'RNC (Radio Network Controller) e viene posto uguale ad 1 per i messaggi inviati dall'RNC all'UE.

La stessa chiave IK può essere usata per i differenti canali simultaneamente associati allo stesso UE (User Equipment), alcuni di tali canali trasmettono i dati in una direzione mentre altri li trasmettono nella direzione opposta. Allora per evitare che l'algoritmo f9 usi lo stesso insieme di valori dei parametri di input sia per i messaggi diretti dall'UE all'RNC che per quelli diretti nel verso opposto, è stato introdotto il parametro DIRECTION.

MESSAGE

E' il messaggio di segnalazione la cui integrità l'algoritmo f9 deve proteggere.

COUNT-I

La lunghezza di COUNT-I è di 32 bit.

E' un contatore di frame introdotto come protezione contro i tentativi di replicare i messaggi di segnalazione durante una connessione. E' composto da due parti; una sequenza "corta" ed una sequenza "lunga". La sequenza "corta" è costituita dai 4 bit meno significativi mentre la sequenza "lunga" è costituita dai 28 bit più significativi. COUNT-I contiene un valore che viene incrementato ogni volta che un messaggio viene protetto.

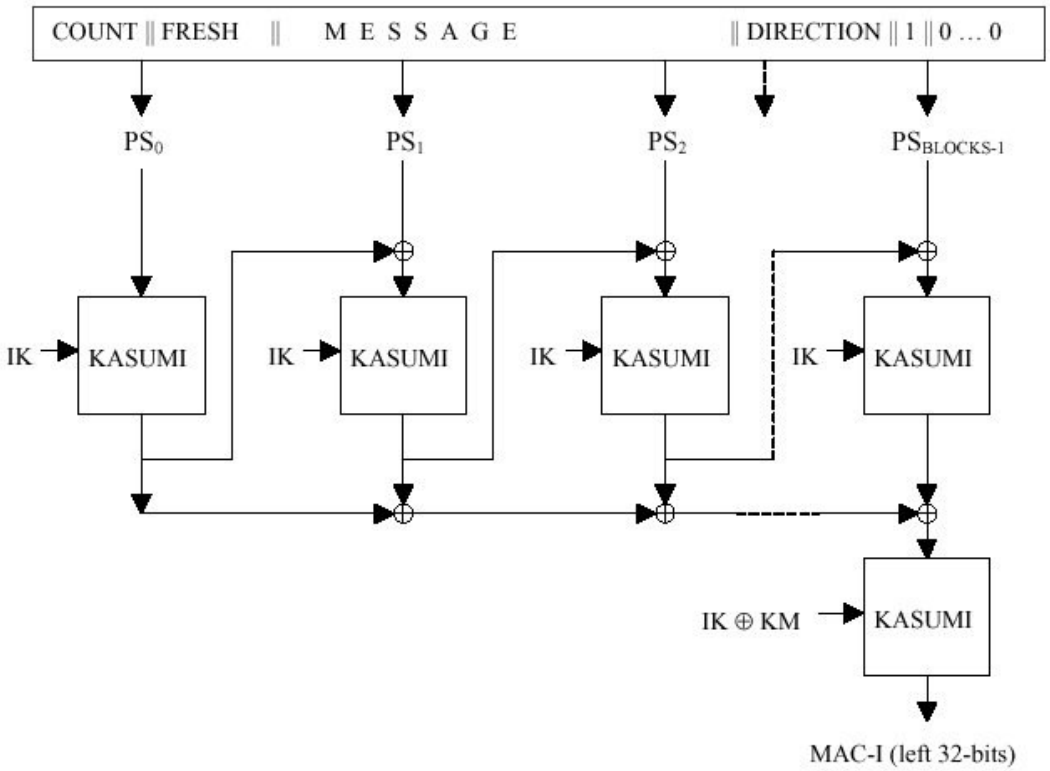
[Per l'esattezza, la parte "lunga" contiene l'RRC HFN (Radio Resource Control Hyper Frame Number) mentre la parte "corta" contiene l'RRC SN (Radio Resource Control Sequence Number) che è disponibile in ogni frame RRC PDU (Radio Resource Control Protocol Data Unit)].

4.7.2 Algoritmo f9

L'algoritmo f9 è basato sull'algoritmo KASUMI (che a sua volta deriva dall'algoritmo MISTY che è un algoritmo block cipher, giapponese).

f9 genera un codice di autenticazione del messaggio detto MAC-I (Message Authentication Code - Integrity) lungo 32 bit, per un messaggio che ha una lunghezza compresa tra 1 e 5114 bit.

Nella seguente figura è riportato lo schema dell'algoritmo f9. Da tale figura si può notare che f9 è una variante dello standard Cipher Block Chaining (CBC).



Inizializzazione

Due registri A e B lunghi 64 bit vengono inizializzati a zero, cioè;

A = 00 ... 0

B = 00 ... 0

(dove 00 ... 0 è la stringa binaria nulla lunga 64 bit).

Una costante KM lunga 128 bit viene inizializzata con la stringa binaria 101010 ... 10 (lunga 128 bit), cioè;

KM = 101010 ... 10

KM viene usata per modificare la chiave IK anche essa lunga 128 bit. Tale modifica viene ottenuta mediante un'operazione di xor tra IK e KM, cioè mediante l'operazione;

IK xor KM

I parametri COUNT, FRESH, MESSAGE e DIRECTION vengono concatenati e viene aggiunto un bit 1 seguito da tanti bit 0 quanti ne sono necessari per ottenere una stringa che abbia un numero totale di bit uguale ad un multiplo di 64 (perciò il numero di bit 0 che vengono aggiunti è compreso tra 0 e 63). La stringa che si ottiene, costituita da un numero di bit multiplo di 64, viene memorizzata in PS, cioè;

PS = COUNT || FRESH || MESSAGE || DIRECTION || 1 || 0^*

cioè;

PS = COUNT[0] ... COUNT[31] FRESH[0] ... FRESH[31] MESSAGE[0] ... MESSAGE[LENGTH-1] DIRECTION[0] 1 0^*

dove 0^* indica una stringa costituita da un numero di bit 0 compreso tra 0 e 63 mentre LENGTH è la lunghezza, in bit, del messaggio.

Elaborazione

La stringa PS viene divisa in tanti blocchi PS_i concatenati ed ognuno costituito di 64 bit, cioè;

PS = PS_0 || PS_1 || PS_2 || ... || $PS_{BLOCKS-1}$

dove BLOCKS è il numero dei blocchi PS_i.

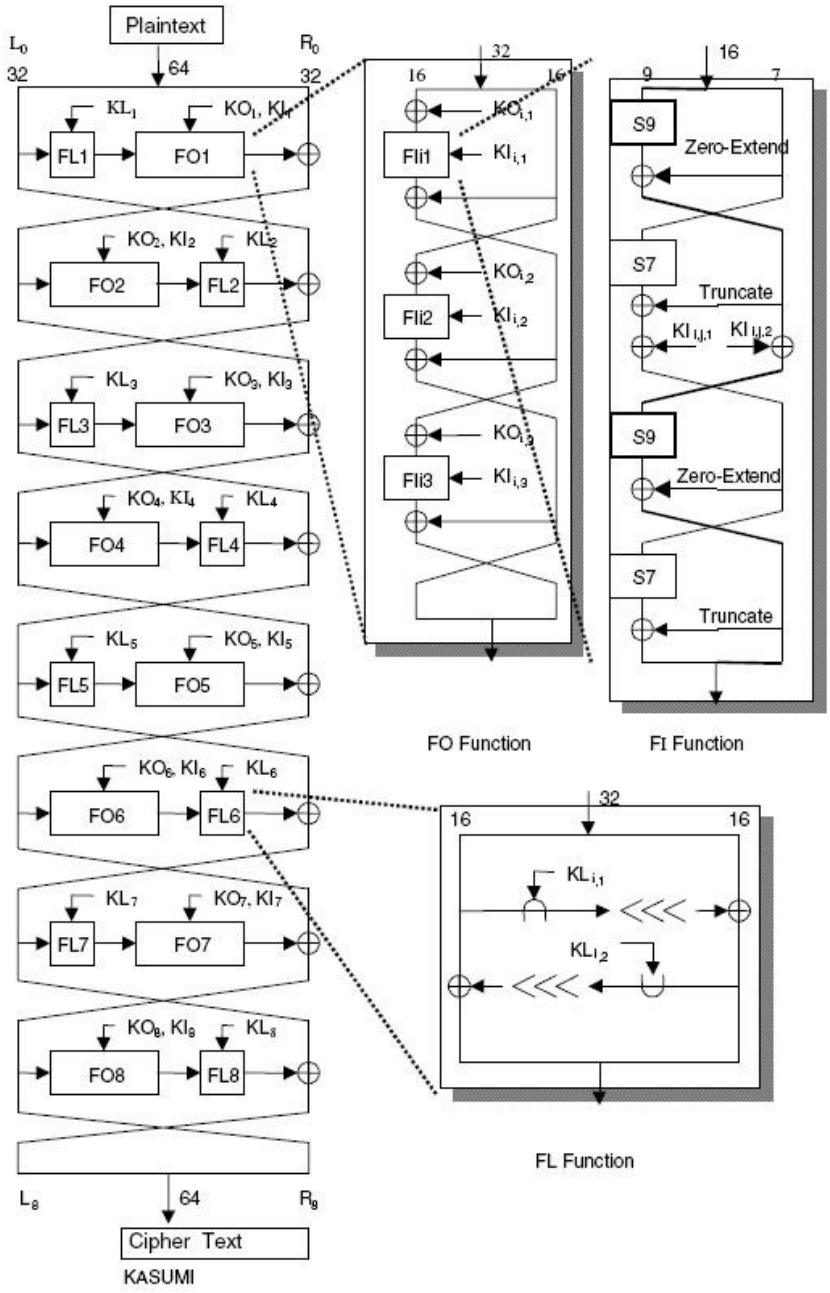
Vengono poi eseguite le seguenti operazioni:

Per ogni intero **n** con **n** = 0, 1, 2, ... , BLOCKS-1 si calcola;
A = KASUMI[A xor PS_n]$_{IK}$
B = B xor A
L'algoritmo perciò consiste di un numero di iterazioni pari a BLOCKS. Ad ogni iterazione vengono eseguite le due precedenti operazioni. Cioè ad ogni iterazione, sotto il controllo della chiave IK, l'algoritmo KASUMI prende come input la stringa A xor PS_n di 64 bit. L'output, consistente in una stringa di 64 bit, viene messo nel registro A. Poi tra l'aggiornato registro A ed il registro B viene eseguita un'operazione di xor ed il risultato viene messo nel registro B.
Dopo che sono state eseguite tutte le iterazioni, viene eseguita un'ulteriore operazione, la seguente;
B = KASUMI[B]$_{IK \text{ xor } KM}$
KASUMI, sotto il controllo della chiave modificata IK xor KM, prende come input il valore contenuto nel registro B di 64 bit. L'output viene memorizzato nello stesso registro B.
I 32 bit più significativi del registro B vengono messi in MAC-I, cioè:
Per ogni intero **i** con **i** = 0, 1, 2, ... , 31 viene eseguita;
MAC-I[i] = B[i]
Così i bit B[32] ... B[63] vengono scartati e MAC-I, il codice di autenticazione del massaggio, è l'output definitivo dell'algoritmo f9.

4.8 Algoritmo KASUMI

Poichè fu stabilito un tempo piuttosto breve per il progetto di un algoritmo riguardante la sicurezza dell'UMTS si decise di partire da un algoritmo già esistente che fosse già stato sottoposto a qualche criterio di valutazione; fu scelto l'algoritmo giapponese, di tipo block cipher, MISTY. La scelta cadde su tale algoritmo per le caratteristiche di sicurezza che fornisce e per la convenienza riguardo una sua implementazione hardware.
Perciò, l'algoritmo KASUMI deriva dall'algoritmo MISTY (ha partecipato al progetto KASUMI anche Matsui, il crittografo della Mitsubishi Electric Corporation, che ha ideato l'algoritmo MISTY).
KASUMI è un algoritmo block cipher (per l'esattezza è un Feistel block cipher) che produce un output di 64 bit da un input di 64 bit sotto il controllo di una chiave di 128 bit. E' composto da otto round.

KASUMI opera su un input I di 64 bit usando una chiave segreta K di 128 bit e produce un output di 64 bit.
L'input I viene diviso in due stringhe L0 e R0 di 32 bit ciascuna, concatenate, cioè;
I = L0 || R0
Nell'i-mo round di KASUMI vengono eseguite le seguenti due operazioni:
Per ogni intero i con i = 1, 2, ... , 8 si calcola;
Ri = Li-1
Li = Ri-1 xor fi(Li-1 , RKi)
Come si vede la funzione fi dell'i-mo round ha come input Li-1 e la chiave RKi.
La chiave RKi è composta dalle tre sottochiavi KLi , KOi , KIi dell'i-mo round.
Dopo otto round, l'algoritmo KASUMI, dà come output la stringa L8 || R8 di 64 bit.



La precedente figura mostra uno schema di implementazione della Kasumi.

4.8.1 Le componenti di KASUMI

La funzione fi

La funzione fi prende un input I di 32 bit e ritorna un output di 32 bit sotto il controllo di una chiave RKi dell'i-mo round.

La chiave RKi È composta dalle tre sottochiavi KLi , KOi , KIi.

La funzione fi È composta dalle due sottofunzioni FL ed FO. Alla FL È associata la sottochiave KLi mentre alla FO sono associate le sottochiavi KOi e KIi.

La funzione fi opera in due modi diversi a seconda se si tratta di un round pari o di un round dispari.

Nei round dispari 1, 3, 5 e 7 si ha;

fi(I , RKi) = FO(FL(I , KLi) , KOi , KIi)

mentre, nei round pari 2, 4, 6 e 8 si ha;
fi(I , RKi) = FL(FO(I , KOi , KIi) , KLi)
Cioè nei round dispari i dati passano prima per FL e poi per FO, mentre nei round pari passano prima per FO e poi per FL.

La funzione FL

La funzione FL prende un input I di 32 bit e una sottochiave KLi di 32 bit.
La sottochiave KLi viene divisa in due sottochiavi KLi,1 e KLi,2 di 16 bit ciascuna, concatenate, cioè;
KLi = KLi,1 || KLi,2
L'input I viene diviso in due stringhe L ed R di 16 bit ciascuna, concatenate, cioè;
I = L || R
Poi vengono eseguite le seguenti operazioni;
R' = R xor ROL(L and KLi,1)
L' = L xor ROL(R' or KLi,2)
nelle quali, oltre alle operazioni xor, and, or, viene eseguita l'operazione ROL di rotazione circolare a sinistra di 1 bit.
L'output di 32 bit è il valore; L' || R'.

La funzione FO

La funzione FO prende come input una stringa I di 32 bit e due sottochiavi KOi e KIi di 48 bit ciascuna.
La stringa I di 32 bit viene divisa in due sottostringhe L0 e R0 di 16 bit ciascuna, concatenate, cioè;
I = L0 || R0
La sottochiave KOi di 48 bit viene suddivisa in tre sottochiavi di 16 bit ciascuna, cioè;
KOi = KOi,1 || KOi,2 || KOi,3
La sottochiave KIi di 48 bit viene suddivisa in tre sottochiavi di 16 bit ciascuna, cioè;
KIi = KIi,1 || KIi,2 || KIi,3
Vengono poi eseguite le seguenti operazioni:
Per ogni intero j con j = 1, 2, 3 si calcola;
Rj = FI(Lj-1 xor KOi,j , KIi,j) xor Rj-1
Lj = Rj-1
L'output è il valore L3 || R3 di 32 bit.

La funzione FI

La funzione FI prende come input una stringa I di 16 bit e una sottochiave $KI_{i,j}$ di 16 bit. La stringa I viene divisa in due sottostringhe L_0 e R_0 concatenate, con L_0 costituita dai 9 bit più significativi di I ed R_0 costituita dai 7 bit meno significativi di I. Cioè;
$I = L_0 \,||\, R_0$
Analogamente la chiave $KI_{i,j}$ viene divisa in una sottochiave $KI_{i,j,1}$ di 7 bit ed una sottochiave $KI_{i,j,2}$ di 9 bit, dove;
$KI_{i,j} = KI_{i,j,1} \,||\, KI_{i,j,2}$
La funzione FI usa due S-box; la S7 che trasforma un input di 7 bit in un output di 7 bit e la S9 che trasforma un input di 9 bit in un output di 9 bit. Inoltre usa due funzioni chiamate ZE e TR, così definite:

- ZE(x) Prende il valore x di 7 bit e lo converte in un valore di 9 bit aggiungendo due bit 0 come bit più significativi.
- TR(x) Prende il valore x di 9 bit e lo converte in un valore di 7 bit eliminando i due bit più significativi.

La FI è implementata dalle seguenti operazioni;

La funzione FI ritorna il valore $L_4 \| R_4$ di 16 bit.

Le S-box S7 ed S9
Le due S-box sono state progettate in maniera tale da poter essere facilmente implementate nella logica combinatoria e in una tabella decimale.
L'input x e l'output y della S-box S7 sono costituiti da 7 bit. L'input x e l'output y della S-box S9 sono costituiti da 9 bit.
Cioè, abbiamo;
$x = x_8 \| x_7 \| x_6 \| x_5 \| x_4 \| x_3 \| x_2 \| x_1 \| x_0$
e
$y = y_8 \| y_7 \| y_6 \| y_5 \| y_4 \| y_3 \| y_2 \| y_1 \| y_0$
dove || è l'operatore di concatenazione ed i bit più significativi x_8 e x_7 dell'input x ed i bit più significativi y_8 e y_7 dell'output y vengono applicati solo alla S-box S9.
Le implementazioni delle S-box S7 ed S9 sia sotto forma di equazioni logiche che di tabelle decimali, sono note ma non riportate qui.

Schedulazione della chiave
KASUMI usa una chiave segreta K di 128 bit.
Prima di ogni round di KASUMI, dalla chiave K di 128 bit vengono ricavate le tre sottochiavi KL_i , KO_i , KI_i , la prima di 32 bit e le altre due di 48 bit ciascuna. Dalla chiave KL_i di 32 bit vengono poi ricavate le due sottochiavi $KL_{i,1}$ e $KL_{i,2}$ ciascuna di 16 bit; dalla chiave KO_i di 48 bit vengono ricavate le tre sottochiavi $KO_{i,1}$, $KO_{i,2}$, $KO_{i,3}$ ciascuna di 16 bit; dalla chiave KI_i di 48 bit vengono ricavate le tre sottochiavi $KI_{i,1}$, $KI_{i,2}$, $KI_{i,3}$ ciascuna di 16 bit.
Perciò prima di ogni round di KASUMI, dai 128 bit della chiave K vengono ricavate le otto sottochiavi;
$KL_{i,1}$, $KL_{i,2}$, $KO_{i,1}$, $KO_{i,2}$, $KO_{i,3}$, $KI_{i,1}$, $KI_{i,2}$, $KI_{i,3}$
ciascuna di 16 bit.
Vediamo come vengono ottenute.
La chiave K di 128 bit viene suddivisa in otto sottostringhe K_1 , K_2 , ... , K_8 di 16 bit ciascuna, concatenate, cioè;
$K = K_1 \| K_2 \| ... \| K_8$
Poi un secondo array di sottochiavi K_j' viene ottenuto da K_j calcolando:
Per ogni intero **j** con **j** = 1, 2, ... , 8
$K_j' = K_j$ xor C_j
dove C_j è il valore costante definito nella sottostante Tab.2.
Le otto sottochiavi di ogni round vengono ottenute da K_j e K_j' operando come descritto in una tabella nota.

4.8.2 Analisi di KASUMI

Ogni componente funzionale di KASUMI è stata attentamente studiata per vedere se rivelasse qualche debolezza che potesse essere sfruttata per portare un attacco all'intero algoritmo.

Funzione FL
La funzione FL è una funzione lineare dalla quale non dipende la sicurezza dell'algoritmo. E' una funzione di disturbo, infatti il suo scopo è di rendere difficile l'individuazione del percorso seguito da un determinato bit attraverso i round di KASUMI.
La funzione FL ha la proprietà che, con una fissata chiave KL, con un input $0^{16}1^{16}$ dà come output 1^{32}. Questo implica che in un round, con certi valori come input, il cambiamento di qualcuno dei bit della chiave KL non ha nessun effetto sull'output di quel round. Questa proprietà può essere usata per ottenere una differenza pari a zero tra l'input e l'output del primo round e quindi in pratica per eliminare il primo round. In generale, perciò, piccoli cambiamenti sull'input della FL producono

solo piccoli cambiamenti sull'output; questo può tornare utile quando si attraversano i round della FL in avanti e all'indietro.
Tutto ciò potrebbe essere sfruttato per attaccare l'algoritmo, anche se finora pare che nessun attacco che sfrutti questa proprietà e che si estende per più di 5 round di KASUMI, sia stato ideato.

Funzione FI
E' la funzione di base per la randomizzazione nell'algoritmo KASUMI.
Prende 16 bit di input e dà 16 bit di output. E' composta da una struttura a 4 round che usa due S-box, la S7 e la S9, che sono due funzioni non lineari. Infatti la S7 e la S9 sono state progettate proprio per evitare la presenza di strutture lineari in FI.
E' stato dimostrato che con una distribuzione uniforme delle sottochiavi in uso, la probabilità differenziale e lineare media della FI è minore di 2^{-14}.

Funzione FO
La funzione FO costituisce la parte non lineare di ciascun round di KASUMI.
E' stato dimostrato che con una distribuzione uniforme delle sottochiavi in uso, la probabilità differenziale e lineare media della FO è minore di 2^{-28}.
FO, con in input una fissata chiave, è una permutazione dei blocchi di 32 bit, ma a causa della sua struttura costituita da 3 round, può essere distinta, mediante l'impiego di quattro plaintext opportunamente scelti, da una permutazione scelta in modo casuale.
E' stato proposto di aggiungere un ulteriore round alla FO per aumentare la sicurezza dell'intero algoritmo KASUMI, ma dato che non sono state trovate indicazioni che le proprietà di una funzione FO con tre round potessero essere usate in un attacco contro tutti gli otto round di KASUMI, è stata lasciata la FO con soli tre round.

Schedulazione della chiave
La schedulazione della chiave di KASUMI anche se molto semplice, non costituisce un punto di debolezza dell'algoritmo anche perchè sembra che non si guadagni nulla dal renderla più complicata.
Ognuno dei 128 bit della chiave segreta K viene usato soltanto una volta in ogni round. Nei differenti round, ogni bit viene usato in differenti modi ed anche in parti differenti. A volte poi, i valori vengono alterati mediante l'uso di costanti che modificano la chiave. Vengono usate le costanti C_1 , C_2 , ... , C_8 nella schedulazione della chiave per evitare che si instauri una relazione di ricorrenza tra le chiavi consecutive di un certo round. Questa proprietà è utile ad impedire attacchi chosen plaintext che sono più veloci della ricerca esaustiva.
Anche se la regolarità e la simmetria nella schedulazione della chiave non è un punto di debolezza per l'algoritmo, è bene evitare di allungare chiavi corte, ad esempio chiavi di 64 bit, in modo estremamente simmetrico. In particolare, è bene evitare l'allungamento di una chiave con la semplice aggiunta di bit nulli, per non agevolare possibili attacchi. Questa, comunque, è solo una misura precauzionale dato che non è stata trovata nessuna chiave debole.
Per il recupero della chiave K sono possibili vari tipi di attacchi. La loro analisi ha portato alla conclusione che è possibile recuperare una chiave in un algoritmo di KASUMI costituito da soli 5 round (a volte anche 6) mentre sembra impossibile il recupero della chiave quando i round di KASUMI sono otto.
In particolare, è stato descritto un attacco differenziale di tipo chosen plaintext per il recupero della chiave su 5 round di KASUMI. L'attacco richiede circa 2^{38} plaintext opportunamente scelti e 2^{80} operazioni. Tale attacco potrebbe essere esteso a 6 round ma non a tutti gli 8 round dell'algoritmo KASUMI. Sono stati presi in considerazione anche gli attacchi differenziali sulla chiave ed è stato concluso che questi tipi di attacchi sono possibili solo su un algoritmo di KASUMI ridotto a quattro o cinque round. Un attacco di tale tipo, su KASUMI ridotto a quattro round, richiede la cifratura di circa 2^9 coppie di plaintext X e X^* scelti opportunamente e con in input rispettivamente le chiavi K

e K^* che differiscono di un solo bit. La complessità media di questo attacco è approssimativamente 2^{41}. Invece, se i round di KASUMI sono cinque, l'attacco richiede la cifratura in media di $3 \cdot 2^{17}$ coppie di plaintext opportunamente scelti con una complessità media di circa 2^{36}. E' stato calcolato che se la chiave K fosse di soli 64 bit, l'algoritmo di KASUMI con soli cinque round, diventerebbe vulnerabile ad un attacco differenziale sulla chiave. Infatti, in tal caso basta la cifratura di circa 10 plaintext sotto due chiavi, con un attacco che ha una complessità di circa 2^{25}.

4.9 Analisi di f8 ed f9

Nella funzione f8 viene calcolata inizialmente la seguente costante;
$W = KASUMI_{CK \text{ xor } KM}$(COUNT || BEARER || DIRECTION || 00 ... 0)
questo essenzialmente per due motivi.
Protezione contro attacchi di tipo chosen plaintext; se la costante W non venisse calcolata e quindi se il vettore IV (Initialisation Vector);
IV = COUNT || BEARER || DIRECTION || 00 ... 0
fosse usato direttamente come input nella struttura a catena che costituisce la f8, si avrebbe che la conoscenza di un plaintext implicherebbe la conoscenza anche dell'input e dell'output di ogni operazione di KASUMI. Invece con il precalcolo della costante W non è possibile conoscere gli input plaintext all'algoritmo KASUMI.
Protezione contro attacchi basati sulle collisioni; l'analisi mostra che in teoria dovrebbe essere possibile distinguere il generatore di una funzione pseudo-random associato ad una funzione f8 mancante del precalcolo della costante W, da un generatore veramente random basato sull'osservazione di, ad esempio, 2^{33} blocchi della keystream. Questo sembra che non si verifichi con la f8 che si è deciso di implementare (cioè con la f8 provvista del precalcolo della W).
In pratica:
Per distinti valori iniziali IV ed IV' (IV ed IV' sono due diversi valori del vettore di input COUNT || BEARER || DIRECTION || 00 ... 0) le rispettive costanti W e W' sono diverse per cui è difficile predire quando certi blocchi della keystream associati a IV ed IV' sono uguali. D'altronde, dall'osservazione di due blocchi uguali della keystream associati a due distinti valori IV, un avversario può trarre informazioni mediante il calcolo del valore W xor W', questa però da sola non rappresenta un'informazione rilevante; essa permette solo di predire che diverse coppie di blocchi associati a IV ed IV' sono uguali, ma poiché ogni keystream è costituita al massimo di 80 blocchi (infatti è stato stabilito che la f8 produrrà una keystream costituita da non più di 80 blocchi, cioè 5114 bit), la probabilità di ottenere un'informazione rilevante risulta essere minore di $80^2/2.2^{64}$ che è circa 2^{-52}.
Un valore iniziale IV possiede le seguenti proprietà caratteristiche; le collisioni sui blocchi della keystream sono possibili ma la probabilità che tali collisioni capitino tra gli 80 blocchi della keystream associata a IV è meno di $80^2/2.2^{64}$ che è circa 2^{-52}. Così anche se uno sfidante è in possesso delle sequenze della keystream associate a tutti i possibili valori, la probabilità che capiti una collisione resta comunque bassa.

f9
Se l'algoritmo f9 fosse stato progettato con uno stato interno lungo 64 bit, un attacco basato sul "messaggio del compleanno" portato con 2^{33} valori (messaggi) potrebbe probabilmente produrre una collisione nello stato interno. Per cui, una volta individuati due messaggi m_i ed m_j che inducono una collisione, si avrebbe che m_i || x ed m_j || x , per una qualche estensione x, darebbero lo stesso MAC. Cioè una volta ottenuto il MAC per m_i || x si potrebbe ottenere il MAC anche per m_j || x.
Come precauzione contro l'attacco del compleanno, è stato previsto per la funzione f9 uno stato interno di 128 bit in modo che possa resistere ad un attacco del compleanno con 2^{33} valori. Infatti con uno stato interno di 128 bit sono necessari, affinchè l'attacco del compleanno abbia successo,

2^{65} valori (cioè stringhe di testo) opportunamente scelti, che è un numero estremamente elevato e quindi impossibile da raggiungere.

Valutazioni
L'obiettivo della valutazione statistica è verificare che KASUMI, f8 ed f9 si comportino come funzioni random.
KASUMI è stato sottoposto al test del cosiddetto "effetto valanga" (avalanche effect) e ad un test che verificasse se fosse capace di distruggere le ridondanze nell'input. L'algoritmo KASUMI ha superato entrambi i test. In particolare con il test "effetto valanga" è stato verificato che, con una fissata chiave, quando cambia uno dei 64 bit dell'input di KASUMI, cambia anche circa la metà dei bit dell'output, cioè cambiano circa 32 bit dell'output. Il test è stato effettuato sia rispetto alla coppia "testo in chiaro-testo cifrato" che rispetto alla coppia "chiave-testo cifrato"; in entrambi i casi è stato superato, confermando così il comportamento random di KASUMI.
Dato che la funzione f8 è un generatore di keystream, sono stati eseguiti vari tipi di test stream cipher (frequency test, gap test, poker test, ...) sulle keystream da esse generate. La f8 è stata sottoposta anche al test "effetto valanga". Tutti i test non hanno evidenziato debolezze statistiche della f8 confermando così il suo comportamento random.
Per la f9 è essenziale che il MAC da essa generato, dipenda da ogni bit dell'input. Per questo la f9 è stata sottoposta al test "effetto valanga" che è stato effettuato sia rispetto alla coppia "chiave-testo cifrato" che rispetto alla coppia "testo in chiaro-testo cifrato"; in entrambi i casi il test è stato superato confermando così il comportamento random della funzione f9.

4.10 Conclusioni

Gli algoritmi KASUMI, f8 ed f9 sono stati sottoposti alla valutazione di vari gruppi di studiosi. Fondamentalmente sono stati giudicati sicuri contro possibili attacchi. Alcune critiche comunque sono state avanzate.
KASUMI: Non sono chiari i motivi di alcuni cambiamenti effettuati sull'algoritmo MISTY e per alcuni cambiamenti non si è sufficientemente sicuri che non inducano qualche tipo di debolezza.
E' ritenuta, invece, ottima la scelta di utilizzare una chiave di 128 bit, in quanto si ritiene che offra sufficiente protezione contro attacchi esaustivi per almeno i prossimi 20 anni. E' stata avanzata però qualche preoccupazione riguardo la possibilità che la lunghezza della chiave possa essere ridotta ponendo i bit meno significativi uguali a zero; il 3GPP ha comunque assicurato che laddove sarà necessario ridurre la lunghezza della chiave, verrà adottato un algoritmo per la costruzione dell'intera chiave di 128 bit richiesta da KASUMI.
Alcuni studiosi hanno inoltre suggerito di aggiungere, per elevare il margine di sicurezza, un ulteriore round alla funzione FO e di progettare una schedulazione della chiave meno semplice. Tali suggerimenti non sono stati accolti dal 3GPP, anche se ritenuti fondati, perchè vorrebbe dire aumentare la complessità dell'algoritmo KASUMI e soprattutto complicare la sua implementazione hardware in cambio di un aumento del margine di sicurezza non ritenuto necessario.
f8 ed f9: Sono state avanzate critiche circa l'uso delle chiavi modificate. Infatti è stato notato che nella f8 il primo blocco della keystream viene ottenuto eseguendo una doppia cifratura KASUMI sotto il controllo di sottochiavi molto simili. Analogamente, in f9 è stato notato che il rapporto esistente tra le sottochiavi di IK ed IK xor KM potrebbe essere causa di debolezza della doppia cifratura.

Le migliorie e le aggiunte rispetto al 2G/GSM hanno reso il 3G/UMTS molto più sicuro, ma sicuramente ci saranno sviluppi e aggiunte futuri per ottenere ulteriori miglioramenti e nuove funzionalità; attualmente non sono presenti nello standard elaborato dal 3GPP un meccanismo di protezione per i protocolli basati su SS7 e per quelli basati su IP; probabilmente guidato da nuove e future “killer application”, verrà inserito anche esplicito supporto per applicazioni di e-commerce, pagamenti via cellulare, flussi multimediali a pagamento, ecc.

Bibliografia

www.dia.unisa.it/~ads/

"Partitioning Attacks: Or How to Rapidly Clone Some GSM Cards"
di Josyula R. Rao, Pankaj Rohatgi and Helmut Scherzer, IBM Watson Research Center

www.gsm-security.net

www.cellular.co.za

www.newlc.com

www.echelon.cx

http://www.dia.unisa.it/~ads/corso-security/www/CORSO-9900/a5/testo.htm

http://www.dia.unisa.it/~ads/corso-security/www/CORSO-9900/umts/testo_umts.htm

"Global System for Mobile Communications"
http://www.absoluteastronomy.com/encyclopedia/G/Gl/Global_system_for_mobile_communications.htm

"A5/1"
http://www.absoluteastronomy.com/encyclopedia/A/A5/A51.htm

"Comparison of Airlink Encryption"
http://www.qualcomm.com/technology/1xev-do/webpapers/wp_Airlink_Encryption.pdf

"La teoria delle comunicazioni su rete GSM"
www.electronetmodena.it/gsm.pdf

"A5/3 and GEA3 Specifications"
http://www.gsmworld.com/using/algorithms/docs/a5_3_and_gea3_specifications.doc

"Real Time Cryptanalysis of A5/1 on a PC"
http://cryptome.org/a51-bsw.htm di Alex Biryukov, Adi Shamir, David Wagner

"Security in the GSM system"
http://www.ausmobile.com/downloads/technical/Security%20in%20the%20GSM%20system%2001052004.pdf

"3GPP TS 55.216 V6.2.0" (A5/3 specifications)
http://www.gsmworld.com/using/algorithms/docs/a5_3_and_gea3_specifications.pdf