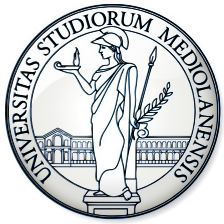

UNIVERSITÀ DEGLI
STUDI DI MILANO

Crittografia
AA 2011-12

Algoritmi Crittografici nello standard UMTS

Davide Gerhard
matricola: 707705
davide.gerhard@studenti.unimi.it

Abstract

Il documento presenterà l'algoritmo di crittografia KASUMI presente nello standard UMTS e ne analizzerà le due diverse modalità di funzionamento; la prima rivolta alla confidenzialità dei dati e la seconda a preservarne l'integrità. In conclusione verranno discussi i punti deboli di tale algoritmo e i possibili attacchi scoperti durante questi anni.

Indice

1 Introduzione

Negli gli anni '90 ci si accorse che la sicurezza dello standard GSM era stata violata, grazie anche a degli studi intrapresi dall'Università di Berkley, e quindi si pensò a disegnare un nuovo schema di cifratura che potesse essere inglobato nella nascente tecnologia UMTS. Si decise che la strada corretta da intraprendere per garantirne una maggiore sicurezza fosse quella di pubblicare le specifiche degli algoritmi dando quindi la possibilità a chiunque di esaminarli. All'epoca si trattò di un grande cambiamento perchè non vi era consuetudine nel rilasciare documenti pubblici, si pensi a GSM, e gli unici ad averne diritto erano i consorziati al progetto. Difatti, le conoscenze sullo standard GSM derivano principalmente da ricercatori che, con la tecnica del reverse engineering, ne hanno svelato il funzionamento. A questo scopo, nel 1999, ETSI creò un progetto, chiamato SAGE TF 3GPP , che servisse a definire il nuovo standard e le relative specifiche implementative. Questo gruppo, composto dal SAGE[1], da Mitsuru Matsui[2], e tre componenti provenienti dal gruppo 3GPP, produsse nei primi mesi del 2000 la prima versione dei documenti qui analizzati. Si decise, inoltre, che lo sviluppo degli algoritmi crittografici dovesse avvenire in due fasi. La prima, quella di definizione degli algoritmi e delle specifiche da parte del gruppo 3GPP[3]. La seconda, di far analizzare i documenti a tre università indipendenti le quali avrebbero dovuto evidenziarne eventuali problematiche. Tale progetto, sviluppato all'interno dell'Unione Europea, si contrappose allo standard prodotto dalla 3GPP2, chiamato CDMA2000, e sviluppato negli Stati Uniti. Nei prossimi capitoli vedremo come gli algoritmi di confidenzialità e di integrità siano stati implementati e come KASUMI, una variante di MISTY1, sia stata utilizzata all'interno di questi algoritimi.
Il documento verterà su un approccio bottom-up, che a differenza della documentazione proposta da 3GPP, partirà con l'analisi dell'algoritmo KASUMI e delle sue componenti fondamentali, per poi analizzare come quest'ultimo venga usato all'interno delle due funzioni, $f8$ e $f9$, con cui lo standard UMTS implementa i requisiti crittografici richiesti, confidenzialità e integrità. Per concludere, verrà presentato un breve excursus sulle possibili problematiche di sicurezza riscontrate in KASUMI durante questo ultimo decennio e come tali requisiti possano essere implementati, con maggiore sicurezza, attraverso l'uso della crittografia asimmet-

[1] ETSI Security Algorithms Group of Experts; gruppo di supervisionamento degli algoritmi crittografici dell'ETSI

[2] Ricercatorie della Mitsubishi Electrical Corporation e padre dell'algoritmo MISTY (versione 1 e 2) e del relativo documento che ne dimostra la sicurezza da attacchi di crittoanalisi lineare e differenziale

[3] organo composto da ETSI (European Telecommunications Standards Institute), ARIB (Association of Radio Industries and Businesses) e TTA (Telecommunications Technology Association)

rica.
Gli algoritmi proposti si baseranno, se non meglio precisato, sui documenti pubblicati da 3GPP in versione 10.0 e in data 2011-03.
In particolare, gli algoritmi che verranno presentati in questa relazione sono tratti dai seguenti documenti:

- 3GPP TS33.105 : cryptography algorithm requirements
- 3GPP TS35:201[2] : f8 and f9 specification
- 3GPP TS35.202[3] : KASUMI specification
- 3GPP TS35.203 : implementations
- 3GPP TS35.204 : design conformance test data

2 KASUMI

2.1 Specifiche

Lo standard UMTS venne pensato tenendo sempre conto della piattaforma finale su cui sarà implementato. Questa verterà principalmente sui telefoni cellulari la cui potenza, all'inizio del nuovo secolo, era davvero ridotta. Si pensi che il valore massimo fissato dai requisiti era di 10000 gate per l'implementazione hardware di tutto l'algoritmo crittografico. Il risultato fù poi alquanto sorprendente visto che si riuscì ad integrare tutto l'algoritmo in soli 3000 gate. Questo aspetto verrà più e più volte trattato all'interno di questa relazione perchè permise sia di ridurre la complessità progettuale sia perchè permise una maggiore efficienza, e di conseguenza maggiore banda disponibile al traffico dati. Inoltre, si tratto della motivazione principale per cui si decise di modificare l'algoritmo MISTY1 e di non usarlo nella sua versione originale. Ovviamente tale aspetto non dovrà mai incidere sulla sicurezza dello stesso. Alla problematica dell'ottimizzazione si aggiunsero anche l'eterogeneità della piattaforma e le problematiche elettriche e fisiche dell'ambiente in cui dovrà operare. Se per la seconda il problema è più di competenza di chi sviluppa la parte radio e i protocolli di trasmissione, per la prima si dovrà prevedere un sistema per cui l'operatore riesca facilmente ad integrare nella propria infrastruttura il nuovo standard. Tale argomento verrà trattato alla fine del prossimo capitolo in cui si presenterà la divisione tra algoritmi crittografici, KASUMI e funzioni $f8$ e $f9$, e quelli di generazione e gestione della chiave. In questo capitolo, invece, analizzeremo nei particolari KASUMI, un algoritmo di crittografia simmetrica a blocchi, derivato, come più volte detto, da MISTY1. La scelta di basarsi su un algoritmo già sviluppato è da attribuirsi al fatto che quest'ultimo avesse già una storia di pubblicazioni e di studi di crittoanalisi che ne garantissero una certa sicurezza. In particolare, si scelse un algoritmo a blocchi e precisamente MISTY1 per i seguenti motivi:

- si predilesse un algoritmo a blocchi perchè permise una facile definizione delle strutture ed una sua valutazione e perchè normalmente i protocolli di integrità sono tipicamente algoritmi MAC e quindi facilmente definibili con un sistema a blocchi.
- doveva essere possibile implementarlo sia in hardware che in software.
- era già stato esaminato accuratamente da molti ricercatori.
- erano stati definiti dei valori quantitativi sulla sua sicurezza[13].
- la lunghezza della chiave era di 128 bit.

Nei prossimi paragrafi si vedrà l'algoritmo KASUMI, una versione lievemente modificata di MISTY1, tale da rispettare le specifiche precedentemente elencate.

2.2 KASUMI

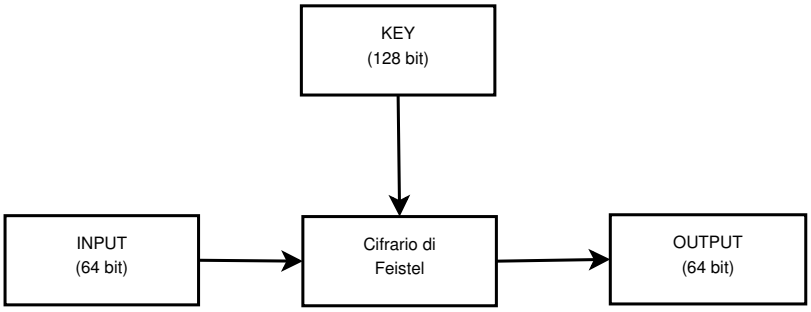

Figura 1: KASUMI a blocchi

KASUMI è un algoritmo di crittografia a blocchi che prevede un blocco lungo 64 bit sia in input che in output e una chiave da 128 bit, come si evince dalla figura 1. La parte crittografica è implementata con l'uso del cifrario di Feistel in otto round. Introduciamo brevemente il funzionamento di KASUMI per poi addentrarci nella spiegazione delle sue componenti fondamentali. Ogni round si compone di due sotto-funzioni, chiamate FL e FO, che si alternano in base al numero del ciclo. In particolare, si avrà che la funzione FL precede FO nei casi in cui il ciclo sia un numero dispari, e viceversa nel caso in cui il numero del ciclo sia pari. Si avrà, quindi, che il cifrario di Feistel sarà così composto:

per i cicli numero 1,3,5,7:

$$f_i(I, RK_i) = FO(FL(I, KL_i), KO_i, KI_i)$$

per i cicli numero 2,4,6,8:

$$f_i(I, RK_i) = FL(FO(I, KO_i, KI_i), KL_i)$$

dove i rappresenta il numero di round a cui si trova l'esecuzione. Nei prossimi paragrafi verrà illustrato il funzionamento di ogni singola funzione per poi ritornare al cifrario di Feistel e analizzarne il funzionamento.

2.3 Funzione FL

La funzione FL riceve in input un blocco dati lungo 32 bit e una chiave KL_i di pari lunghezza. Quest'ultima sarà divisa in due parti, tale che

$$KL_i = KL_{i,1} \| KL_{i,2}$$

Lo stesso procedimento sarà applicato al blocco dati in input il quale verrà suddiviso in due blocchi avente ciascuno lunghezza di 16 bit e chiamati rispettivamente L ed R. Questa suddivisione simmetrica ci permette di definire due funzioni separate, una per la parte sinistra (left=L) ed una per la parte destra (right=R) le quali saranno calcolate nella seguente maniera:

$$R' = R \oplus ROL(L \cap KL_{i,1})$$

$$L' = L \oplus ROL(R' \cup KL_{i,2})$$

dove ROL è una rotazione circolare sinistra di un bit, $\cap$ rappresenta l'AND logico, $\cup$ rappresenta l'OR logico e $\oplus$ rappresenta lo XOR logico. Il risultato di tale procedimento è visibile in figura 2 e avrà come blocco in output, di lunghezza pari a 32 bit, il seguente valore:

$$OUTPUT = L' || R'$$

Analisi
Come si è potuto vedere, la funzione FL è una funzione lineare e lo scopo principale è quello di avere un metodo addizionale di codifica senza avere una complessità troppo elevata. Infatti, un piccolo cambiamento dell'input genera soltanto un piccolo cambiamento nell'output. Un'altra proprietà, individuabile nel funzionamento di FL, è che se noi prendessimo una chiave arbitraria KL e avessimo in input una sequenza di $0^{16}1^{16}$ il risultato sarebbe sempre costante e pari a 1^{32}.

2.4 Funzione FO

La funzione FO riceve in input un blocco dati lungo 32 bit e due chiavi KO_i e KI_i lunghe 48 bit ciascuna. Quest'ultime saranno suddivise in blocchi di 16 bit nella seguente maniera:

$$KO_i = KO_{i,1} \| KO_{i,2} || KO_{i,3}$$

$$KI_i = KI_{i,1} \| KI_{i,2} || KI_{i,3}$$

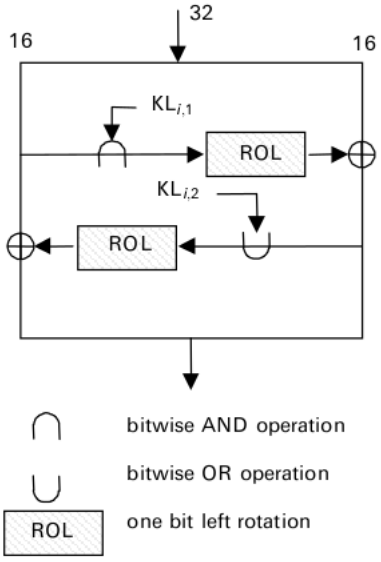

Figura 2: funzione FL_i

La funzione verrà suddivisa, come nella precedente funzione, in due rami, quello destro, denominato R, e quello sinistro, denominato L, tale che

$$R_j = FI(L_{j-1} \oplus KO_{i,j}, KI_{i,j}) \oplus R_{j-1}$$

$$L_j = R_{j-1}$$

dove j e' un numero intero definito come $1 \leq j \leq 3$ e FI una funzione che verrà definita nel prossimo paragrafo. Il risultato, come visibile in figura 3, sarà un blocco di 32 bit e il suo valore sarà così composto:

$$OUTPUT = L_3||R_3$$

Analisi
La funzione FO costituisce la parte non lineare di ogni round. Tale aspetto viene implementato attraverso l'uso della funzione FI che al suo interno implementa quattro round in cui vengono eseguite due distinte S-box. Ma vediamo in dettaglio come la funzione FI è implementata.

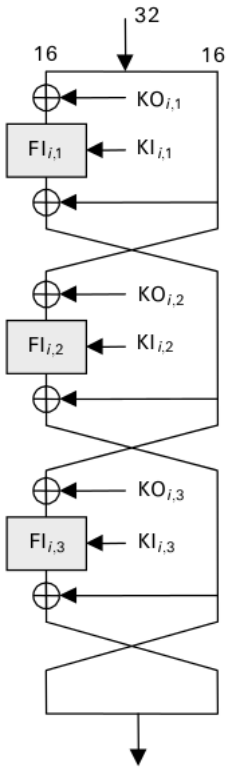

Figura 3: funzione FO_i

2.5 Funzione FI

La funzione FI viene usata all'interno della funzione FO e riceve in input un blocco di 16 bit e una chiave $KI_{i,j}$ lunga, anch'essa, 16 bit. Il blocco in input, come definito nell'equazione del paragrafo precedente, sarà composto da $L_{j-1} \oplus KO_{i,j}$. A questo punto l'input verrà suddiviso in due blocchi aventi lunghezza differente; la parte sinistra L_0 avrà una lunghezza pari a 9 bit, mentre la parte destra R_0 avrà una lunghezza di 7 bit. Una simile suddivisione asimmetrica si avrà anche per la chiave dove però la parte sinistra $KI_{i,j,1}$ avrà una lunghezza di 7 bit e invece la parte destra $KI_{i,j,2}$ avrà una lunghezza pari a 9 bit. Quindi, l'input e la chiave saranno così rappresentate:

$$I = L_0 \| R_0$$

$$KI_{i,j} = KI_{i,j,1} \| KI_{i,j,2}$$

Come preannunciato, la funzione FI verterà su due S-box e su due funzioni di trasformazione. Le prime, in base al principio di confusione enunciato da Claude

E. Shannon[4], permettono di complicare le relazioni tra il testo in chiaro e il testo cifrato in maniera non lineare. Le seconde, chiamate $ZE()$ e $TR()$, permettono di trasformare un input avente una certa lunghezza in un output avente una lunghezza differente. Vediamo brevemente quest'ultime per poi ritornare alle S-box.
Le funzioni di trasformazione saranno così definite:
$ZE()$: prende in input 7 bit e li converte in un output avente 9 bit aggiungendo due bit con valore zero nella parte più significativa del blocco.
$TR()$: prende in input 9 bit e li converte in un output avente 7 bit togliendo i due bit più significativi dal blocco.
Ritornando alle S-box, abbiamo che la funzione FI userà due distinte mappe, la prima, chiamata $S7$, che trasla un input di 7 bit in un output avente stessa lunghezza, e la seconda, chiamata $S9$, che trasla un input di 9 bit in un output avente stessa lunghezza. Definiti i componenti della funzione, possiamo vedere nello specifico come verranno implementati i quattro round che la compongono. Si ha quindi:

$$\begin{array}{ll} L_1 = R_0 & R_1 = S9[L_0] \oplus ZE(R_0) \\ L_2 = R_1 \oplus KI_{i,j,2} & R_2 = S7[L_1] \oplus TR(R_1) \oplus KI_{i,j,1} \\ L_3 = R_2 & R_3 = S9[L_2] \oplus ZE(R_2) \\ L_4 = S7[L_3] \oplus TR(R_3) & R_4 = R_3 \end{array}$$

Alla conclusione di queste operazioni si avrà in output un blocco lungo 16 bit e costituito dai risultati dei due rami e sarà così composto:

$$OUTPUT = L_4 \| R_4$$

Una visione grafica della funzione la si ha in figura 4.

Analisi
La funzione FI permette all'algoritmo KASUMI di mantenere un alto grado di randomizzazione tra i 16 bit in input e i 16 bit in output. È composto da una struttura a quattro round e usa due funzioni di sostituzione non lineare, dette S-box, che vedremo in dettaglio nel prossimo paragrafo, e che permettono di evitare, con un buon tasso di probabilità (2^{-14}), il legame tra il blocco in ingresso e quello in uscita. Questa proprietà è stata anche confermata da test statistici.

2.6 S-box

Le due S-box, usate nella funzione FI, possono essere implementate in due modi diversi: nel primo, attraverso l'uso della logica combinatoria, nel secondo attraverso

[4] http://netlab.cs.ucla.edu/wiki/files/shannon1949.pdf

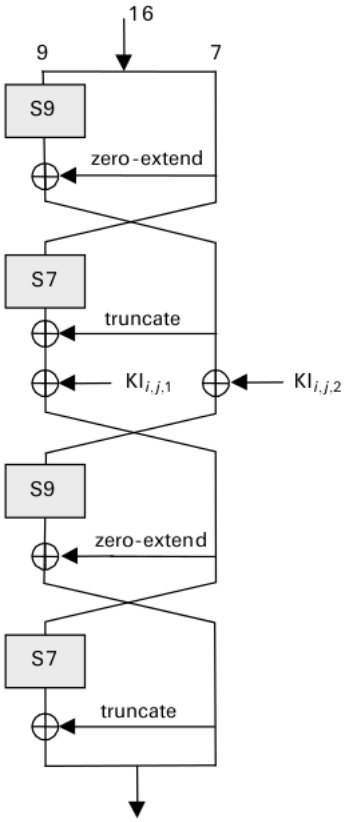

Figura 4: funzione $FI_{i,j}$

l'uso di tabelle di look-up. Si tenga presente che queste funzioni devono essere implementate in hardware con scarse disponibilità computazionali. In questi anni, seguendo la legge di Moore, si è però passati dall'uso di funzioni di logica combinatoria, implementata direttamente in hardware, all'uso di tabelle.
In questo documento verrà analizzata soltanto la prima via, quella delle funzioni di logica combinatoria, perchè chiarisce meglio l'idea del loro funzionamento, ad ogni modo sul documento [3] si trova anche la spiegazione del funzionamento delle tabelle di look-up.
Iniziamo col definire una rappresentazione univoca del blocco in input:

$$x = x8||x7||x6||x5||x4||x3||x2||x1||x0$$

e del blocco in output:

$$y = y8||y7||y6||y5||y4||y3||y2||y1||y0$$

dove $x8, x7$ e $y8, y7$ verranno usati soltanto nella S-box $S9$ visto che l'input e l'output hanno lunghezza pari a 9 bit. Definiamo la sequenza $x0x1x2$ equivalente a $x1 \in x2 \in x3$ e che l'operatore $\oplus$ equivale all'operatore binario XOR. Quindi, le S-box $S7$ e $S9$ saranno così formate:

S7 - funzioni logiche

$$y0 = x1x3 \oplus x4 \oplus x0x1x4 \oplus x5 \oplus x2x5 \oplus x3x4x5 \oplus x6 \oplus x0x6 \oplus x1x6 \oplus x3x6$$
$$\oplus x2x4x6 \oplus x1x5x6 \oplus x4x5x6$$

$$y1 = x0x1 \oplus x0x4 \oplus x2x4 \oplus x5 \oplus x1x2x5 \oplus x0x3x5 \oplus x6 \oplus x0x2x6 \oplus x3x6 \oplus x4x5x6 \oplus 1$$

$$y2 = x0 \oplus x0x3 \oplus x2x3 \oplus x1x2x4 \oplus x0x3x4 \oplus x1x5 \oplus x0x2x5 \oplus x0x6 \oplus x0x1x6 \oplus x2x6 \oplus x4x6 \oplus 1$$

$$y3 = x1 \oplus x0x1x2 \oplus x1x4 \oplus x3x4 \oplus x0x5 \oplus x0x1x5 \oplus x2x3x5 \oplus x1x4x5 \oplus x2x6 \oplus x1x3x6$$

$$y4 = x0x2 \oplus x3 \oplus x1x3 \oplus x1x4 \oplus x0x1x4 \oplus x2x3x4 \oplus x0x5 \oplus x1x3x5 \oplus x0x4x5 \oplus x1x6$$
$$\oplus x3x6 \oplus x0x3x6 \oplus x5x6 \oplus 1$$

$$y5 = x2 \oplus x0x2 \oplus x0x3 \oplus x1x2x3 \oplus x0x2x4 \oplus x0x5 \oplus x2x5 \oplus x4x5 \oplus x1x6 \oplus x1x2x6$$
$$\oplus x0x3x6 \oplus x3x4x6 \oplus x2x5x6 \oplus 1$$

$$y6 = x1x2 \oplus x0x1x3 \oplus x0x4 \oplus x1x5 \oplus x3x5 \oplus x6 \oplus x0x1x6 \oplus x2x3x6 \oplus x1x4x6 \oplus x0x5x6$$

S9 - funzioni logiche

$$y0 = x0x2 \oplus x3 \oplus x2x5 \oplus x5x6 \oplus x0x7 \oplus x1x7 \oplus x2x7 \oplus x4x8 \oplus x5x8 \oplus x7x8 \oplus 1$$

$$y1 = x1 \oplus x0x1 \oplus x2x3 \oplus x0x4 \oplus x1x4 \oplus x0x5 \oplus x3x5 \oplus x6 \oplus x1x7 \oplus x2x7 \oplus x5x8 \oplus 1$$

$$y2 = x1 \oplus x0x3 \oplus x3x4 \oplus x0x5 \oplus x2x6 \oplus x3x6 \oplus x5x6 \oplus x4x7 \oplus x5x7 \oplus x6x7 \oplus x8 \oplus x0x8 \oplus 1$$

$$y3 = x0 \oplus x1x2 \oplus x0x3 \oplus x2x4 \oplus x5 \oplus x0x6 \oplus x1x6 \oplus x4x7 \oplus x0x8 \oplus x1x8 \oplus x7x8$$

$$y4 = x0x1 \oplus x1x3 \oplus x4 \oplus x0x5 \oplus x3x6 \oplus x0x7 \oplus x6x7 \oplus x1x8 \oplus x2x8 \oplus x3x8$$

$$y5 = x2 \oplus x1x4 \oplus x4x5 \oplus x0x6 \oplus x1x6 \oplus x3x7 \oplus x4x7 \oplus x6x7 \oplus x5x8 \oplus x6x8 \oplus x7x8 \oplus 1$$

$$y6 = x0 \oplus x2x3 \oplus x1x5 \oplus x2x5 \oplus x4x5 \oplus x3x6 \oplus x4x6 \oplus x5x6 \oplus x7 \oplus x1x8 \oplus x3x8 \oplus x5x8 \oplus x7x8$$

$$y7 = x0x1 \oplus x0x2 \oplus x1x2 \oplus x3 \oplus x0x3 \oplus x2x3 \oplus x4x5 \oplus x2x6 \oplus x3x6 \oplus x2x7 \oplus x5x7 \oplus x8 \oplus 1$$

$$y8 = x0x1 \oplus x2 \oplus x1x2 \oplus x3x4 \oplus x1x5 \oplus x2x5 \oplus x1x6 \oplus x4x6 \oplus x7 \oplus x2x8 \oplus x3x8$$

Analisi
La $S7$ box è essenzialmente uguale a quella presente in MISTY1 e soltanto l'ordine di alcuni bit è stato cambiato. Inoltre, tale S-box è stata pensata per essere facilmente implementata in hardware usando la logica combinatoria e ha un ottima proprietà non lineare. Di contro la funzione $S9$ non è derivata da quella di MISTY1 perchè quest'ultima era troppo complessa da implementare in hardware. Si è inoltre dimostrato che tale S-box raggiunge quasi una perfetta non linearità.

2.7 Schedulazione della chiave

KASUMI, come detto nei paragrafi precedenti, prevede che vengano usati otto round del cifrario di Feistel e ogni round necessiterà di una chiave. Per generare le otto chiavi necessarie bisognerà prevedere una procedura per cui, partendo dalla singola chiave iniziale, lunga 128 bit, si possano ricavare otto chiavi, il più possibili indipendenti una col'altra, e aventi sempre lunghezza pari a 128 bit. Vediamo come questa operazione possa essere effettuata automaticamente, così da poter ricavare i valori che abbiamo definito nelle funzioni precedenti, KL_i, KO_i, KI_i. Iniziamo col suddividere la chiave K (128 bit) in otto parti lunghe ciascuna 16 bit, tale che

$$K = K1\|K2\|K3\|K4\|K5\|K6\|K7\|K8$$

dove Kj e' uno dei sotto-prodotti della chiave. Quindi, per ogni j con $1 \leq j \leq 8$, calcoliamo

$$K'_j = K_j \oplus C_j$$

dove C_j è uguale al valore presente nella seguente tabella:

C1	0x0123
C2	0x4567
C3	0x89AB
C4	0xCDEF
C5	0xFEDC
C6	0xBA98
C7	0x7654
C8	0x3210

Per conclusione, si avrà che le sotto chiavi delle funzioni FO, FI, FL saranno generate come definito nella seguente tabella 5.

	1	2	3	4	5	6	7	8
$KL_{i,1}$	K1<<<1	K2<<<1	K3<<<1	K4<<<1	K5<<<1	K6<<<1	K7<<<1	K8<<<1
$KL_{i,2}$	K3′	K4′	K5′	K6′	K7′	K8′	K1′	K2′
$KO_{i,1}$	K2<<<5	K3<<<5	K4<<<5	K5<<<5	K6<<<5	K7<<<5	K8<<<5	K1<<<5
$KO_{i,2}$	K6<<<8	K7<<<8	K8<<<8	K1<<<8	K2<<<8	K3<<<8	K4<<<8	K5<<<8
$KO_{i,3}$	K7<<<13	K8<<<13	K1<<<13	K2<<<13	K3<<<13	K4<<<13	K5<<<13	K6<<<13
$KI_{i,1}$	K5′	K6′	K7′	K8′	K1′	K2′	K3′	K4′
$KI_{i,2}$	K4′	K5′	K6′	K7′	K8′	K1′	K2′	K3′
$KI_{i,3}$	K8′	K1′	K2′	K3′	K4′	K5′	K6′	K7′

Figura 5: Tabella di schedulazione delle chiavi

Analisi
Si è visto come le chiavi vengano generate per gli otto round necessari al cifrario di Feistel e anche se questo meccanismo all'apparenza pare semplice non ne pregiudica la sicurezza. L'uso dei valori C_j, elencati nella precedente tabella, permettono di non avere correlazione tra una chiave successiva e l'altra. Questo punto previene l'attacco di tipo chosen plaintext. A questo si aggiunge, nel caso non fosse già chiaro, che una chiave, derivata dal processo di generazione, verrà usata una sola volta nel arco di tutto l'algoritmo e sarà sempre lunga 128 bit.

2.8 KASUMI - L'algoritmo

Visti tutti i blocchi costituenti il cifrario di Feistel possiamo vedere e comprendere come questi vengano ricombinati insieme e rispondano alle specifiche richieste. In figura 6 possiamo osservare la ricostruzione a blocchi dell'algoritmo KASUMI. Partiamo rivedendo i dati in ingresso; si avrà un blocco in input lungo 64 bit e una chiave lunga 128 bit. In output, invece, avremo un solo oggetto lungo 64 bit. L'input I verrà suddiviso in due parti simmetriche, ciascuna lunga 32 bit, tale che

$$I = L_0 \| R_0$$

e per ogni round i, dove i è $1 \leq i \leq 8$, verranno eseguite le seguenti operazioni:

$$R_i = L_{i-1}$$

$$L_i = R_{i-1} \oplus f_i(L_{i-1}, RK_i)$$

dove f_i è la funzione definita per quel round.
Come presentato all'inizio di questo capito si avrà che f_i sarà uguale per i cicli numero 1,3,5,7 a

$$f_i(I, RK_i) = FO(FL(I, KL_i), KO_i, KI_i)$$

e per i cicli numero 2,4,6,8 a

$$f_i(I, RK_i) = FL(FO(I, KO_i, KI_i), KL_i)$$

Alla conclusione degli otto round previsti, si avrà in output un blocco lungo 64 bit e così formato

$$OUTPUT = L_8 \| R_8$$

Questo risultato, verrà presentato nel prossimo capito con la seguente dicitura

$$OUTPUT = KASUMI[I]_K$$

dove I è il blocco di input e K è la chiave simmetrica.

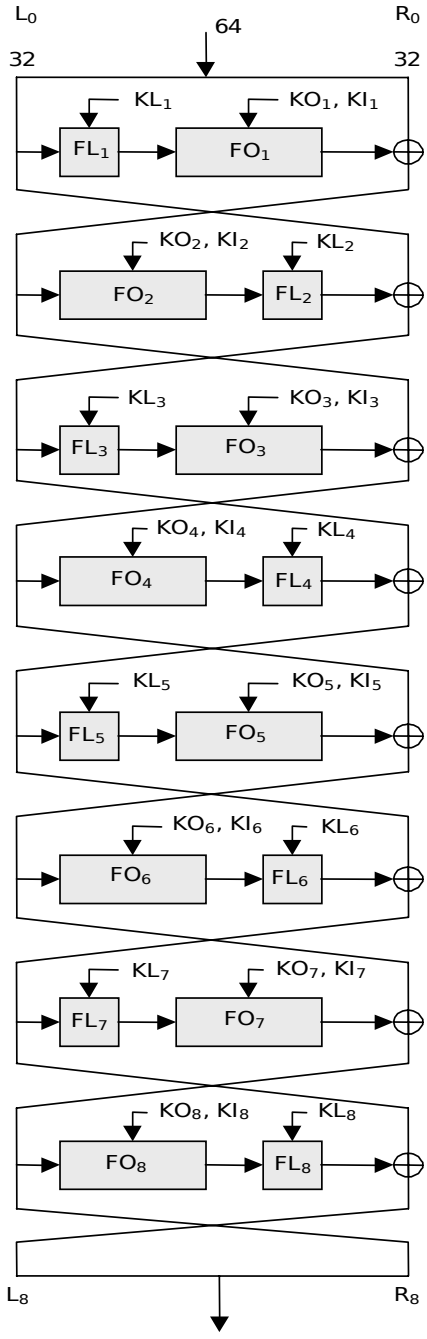

Figura 6: Algoritmo KASUMI

2.9 Vantaggi Hardware di KASUMI

Come si aveva già accennato all'inizio di questo capito, durante la fase di definizione delle specifiche, il team di sviluppo dell'algoritmo KASUMI ha sempre tenuto in gran considerazione l'ambiente in cui tali funzioni dovranno essere implementate. A questo proposito vediamo brevemente quali siano state le accortezze e i risultati ottenuti per minimizzare il dispendio di cpu e memoria e massimizzare la larghezza di banda criptata, sia in upload che in download, con un clock di 20 MHz.

- le S-box possono essere facilmente implementate in hardware, constano di soli due operatori booleani: AND e XOR.
- si ha la possibilità di parallelizzare il calcolo delle S-box quindi, mentre calcoliamo la $S7$ nel frattempo possiamo calcolare $S9$.
- le operazione di generazione delle chiavi sono facilmente implementabili in hardware e constano di sole due operazioni: shifting e XOR.
- le funzioni $FI_{i,1}$ e $FI_{i,2}$ possono essere eseguite in parallelo.

Queste ed altre funzionalità hanno permesso di implementare l'algoritmo KASUMI all'interno dello standard UMTS e di poterlo sviluppare anche in presenza di cellulari non molto performanti, com'erano quelli presenti nei primi anni 2000.

3 UMTS

Lo standard UMTS prevede che siano identificate due entità, la prima, chiamata *user equipment* (UE), e la seconda, chiamata *radio network controller* (RNC), che identificando, rispettivamente, il telefono o l'apparato in dotazione all'utente e la stazione di trasmissione del segnale UMTS gestito dall'operatore (BTS[5]). Essendo tale canale trasmissivo basato su onde radio, qualsiasi malintenzionato potrebbe, attraverso l'uso di uno scanner o di un apparato con capacità pari a quelle di un telefono, ricevere il segnale inviato e ricevuto dagli utenti presenti nelle vicinanze. Prima dell'avvento della tecnologia GSM era presente una tecnologia chiama TACS che non essendo criptata permetteva a qualsiasi persona, dotata anche di un semplice televisore con funzionalità di scanner manuale, di intercettare le chiamate presenti nella zona. Per questo motivo si pensò di definire un nuovo standard, chiamato GSM, che prevedesse l'uso della crittografia per preservare la confidenzialità delle comunicazioni. Questo aspetto doveva essere integrato a diversi livelli, il primo, all'interno del *medium access control* (MAC) e il secondo all'interno del *radio link control* (RLC). Per questo motivo si dovette, come nello studio dell'algoritmo KASUMI, tenere ben presente le difficoltà implementative e la necessità di poter realizzare tali specifiche sia a livello hardware, in particolare per lo user equipment, sia a livello software, per il radio network control. Oltre a questi requisiti, il gruppo di lavoro per l'UMTS, facente capo al 3GPP, dovette produrre, partendo sempre dall'algoritmo KASUMI, due funzionalità distinte. La prima che supplisse alla confidenzialità delle trasmissioni, siano essi dati, voce o traffico di controllo, per esempio SS7[6], e la seconda che permettesse l'integrità dei dei inviati. La prima funzionalità verrà chiamata, nella documentazione 3GPP, funzione $f8$ e verrà vista nel prossimo paragrafo. La seconda, chiamata funzione $f9$, verrà trattata in quello successivo. Nel paragrafo conclusivo si accennerà all'infrastruttura e alle possibilità che gli operatori hanno per gestire le chiavi di cifratura.

3.1 Funzione f8

La funzione $f8$, come accennato nell'introduzione, si prefigge l'obiettivo di mantenere confidenziali i dati tra i due punti terminali. La funzione $f8$ è un algoritmo *stream cipher* che permette di cifrare e decifrare blocchi di dati attraverso una chiave confidenziale CK, lunga 128 bit. Il blocco, invece, potrà avere una dimensione variabile, da 1 a 20000 bit, e verrà elaborato dall'algoritmo KASUMI in modalità auto-feedback. In particolare, la funzione $f8$ può essere vista come una combi-

[5] Base transceiver station

[6] Signalling System No. 7 - http://www.ss7-training.net/

nazione tra due dei standard definiti dal NIST[7], l'OFB[8] e il CTR[9]. In figura 7 può essere osservato come questi due tipi di cifrari a blocchi siano combinati. Ma vediamo come tale funzione sia implementata.

Iniziamo col definire cosa l'algoritmo riceva in input e cosa generi in output.

INPUT

- **COUNT** (32-bit): evita l'attacco replay perchè viene incrementato di 1 ad ogni nuovo messaggio. Composto da due parti: Hyper Frame Number (HFN) e dal Radio Resource Control (RRC) sequence number (SQN).
- **BEARER** (5-bit): identità del canale radio.
- **DIRECTION** (1-bit): direzione della trasmissione.
- **CK** (128-bit): chiave.
- **LENGTH** (intero): numero di bit che dovranno essere cifrati/decifrati.
- **IBS** (1-20000 bit): dato da cifrare/decifrare.

OUTPUT

- **OBS** (1-20000): dato cifrato/decifrato.

Proseguiamo col definire l'insieme delle procedure necessarie per il funzionamento della funzione $f8$. Tale insieme è composto, in ordine, dall'inizializzazione, dalla generazione del keystream e dalla fase di cifratura e decifratura.

3.1.1 Inizializzazione

In questa fase si inizializza il generatore di keystream e si definisce il valore di alcuni registri necessari al funzionamento, A e $BLKCNT$.

1. definiamo il registro $A = COUNT||BEARER||DIRECTION||0...0$. Aggiungiamo nella parte finale della parola 26 zeri.
2. definiamo il counter $BLKCNT$ a zero
3. definiamo la chiave modificatrice $KM = 0x55555555555555555555555555555555$

[7] http://www.itl.nist.gov/fipspubs/fip81.htm
[8] Output Feedback
[9] Counter

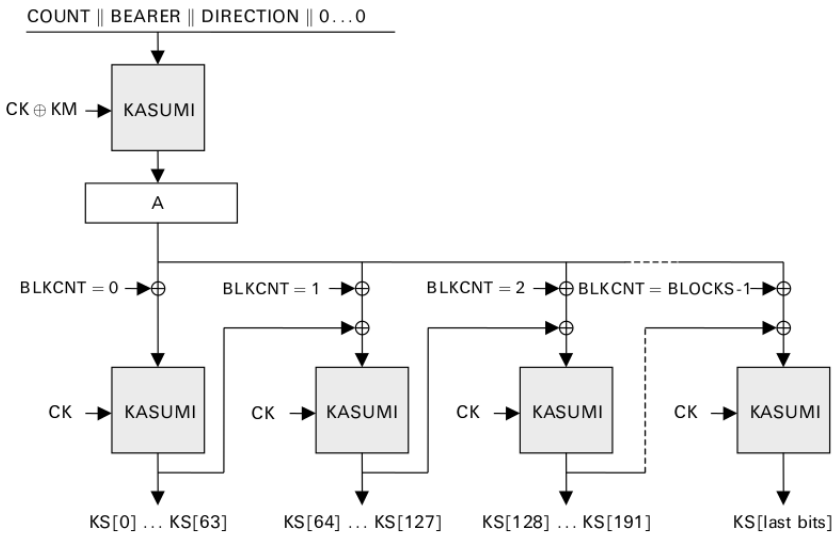

Figura 7: Inizializzazione e generatore di stream della funzione f8

4. definiamo KSB_0 a zero

5. eseguiamo, una sola volta, l'algoritmo KASUMI usando come input il registro A e una versione modificata della chiave CK. Si avrà quindi

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

3.1.2 Generatore di keystream

Conclusa la procedura di inizializzazione possiamo procedere con la procedura che definisce le chiavi usate nella catena di cifratura e decifratura dei dati in input alla funzione $f8$.
Definiamo $BLOCKS = LENGTH/64$ arrotondato per eccesso.
Definiamo il blocco di keystream KSB come

$$KSB_n = KASUMI[A \oplus BLKCNT \oplus KSB_{n-1}]_{CK}$$

dove n e' un numero intero, tale che $1 \leq n \leq BLOCKS$ e $BLKCNT = n - 1$.
Estraiamo, da sinistra verso destra, i bit del keystream KSB_n tale che

$$KS[((n-1) * 64) + i] = KSB_n[i]$$

e dove i e' definito come $0 \leq n \leq 63$ e n và da 1 a $BLOCKS$.

3.1.3 Cifratura/Decifratura

Essendo un cifrario a blocchi la parte di cifratura e decifratura sono identiche ed è attuata usando l'operatore logico XOR tra i dati in input, definiti con IBS, e la chiave proveniente dal generatore di keystream KS. Si ha quindi, che l'output OBS è definito nella seguente maniera

$$OBS[i] = IBS[i] \oplus KS[i]$$

dove i è un intero che và da 0 a $LENGTH - 1$.

3.2 Funzione f9

La funzione $f9$ si prefigge il compito di garantire che il messaggio ricevuto sia sicuramente quello inviato e che quindi non abbia subito trasformazioni durante la trasmissione. Questo permette di proteggere l'intergrità dei dati e di autenticare l'origine dei messaggi di segnalazione. Nello standard UMTS viene implementato a livello RRC[10]. In crittografia la classe di algoritmi che identifica tali requisiti è chiamato MAC (Message authentication code). La funzione $f9$ userà una versione modificata del noto CBC-MAC[11] che altrimenti avrebbe compromesso la sicurezza dell'algoritmo, visto che la lunghezza dei blocchi prodotti da KASUMI è troppo piccola per garantire una bassa probabilità di collisione tra i blocchi aventi diverso input. Ma vediamo in dettaglio come venga implementato e quali informazioni riceva in ingresso la funzione $f9$.

INPUT

- **COUNT-I** (32-bit): evita l'attacco replay perchè viene incrementato di 1 ad ogni nuovo messaggio. Composto da due parti: Hyper Frame Number (HFN) e dal Radio Resource Control (RRC) sequence number (SQN).
- **FRESH** (32-bit): numero random.
- **DIRECTION** (1-bit): direzione della trasmissione.
- **IK** (128-bit): chiave.
- **LENGTH** (intero): numero di bit che dovranno essere elaborati.
- **IBS** (LENGTH): dato da elaborare. Non è imposto nessun vincolo di lunghezza.

[10] Radio Resource Control
[11] Cipher Block Chaining Message Authentication Code

OUTPUT

- **MAC-I** (32): MAC generato a partire da IBS.

L'implementazione è sempre basata su KASUMI e permette di generare un digest del messaggio in output lungo 64 bit, vedi figura 8.

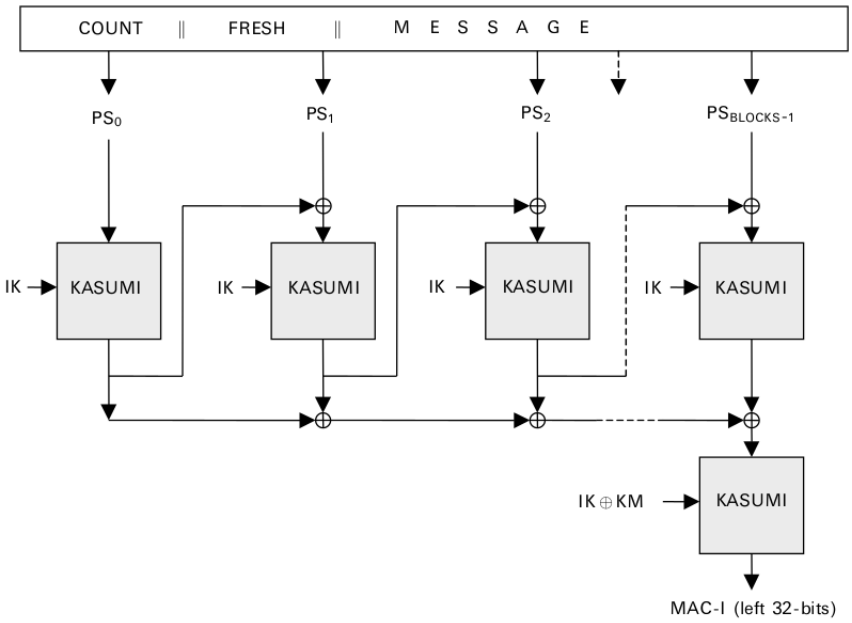

Figura 8: funzione f9

Il problema, come definito nei requisiti di output, è che necessitiamo di un digest lungo 32 bit e non di 64 bit come KASUMI ci fornisce. Per ovviare a questo problema, prendiamo soltanto i 32 bit "sinistri" ($MAC - I[0]..MAC - I[31]$).
Definiti gli input e gli output della funzione vediamo come questa sia stata implementata in due fase: il primo, dedito all'inizializzazione del sistema, e il secondo, dedito al calcolo del digest.

3.2.1 Inizializzazione

Come già avvenuto all'interno della funzione $f8$, definiamo e inizializziamo le variabili e registri che saranno usati nella fase di calcolo del MAC.

1. definiamo a zero le variabili A e B.

2. definiamo la chiave modificatrice $KM = 0xA^{32}$

3. definiamo

$$PS = COUNT \| FRESH \| MESSAGE \| DIRECTION \| 1 \| 0*$$

e aggiungiamo, dopo $DIRECTION$, un bit 1 e tanti zeri quanti servono per avere il blocco PS multiplo di 64.

3.2.2 Calcolo del MAC

Iniziamo col dividere la stringa PS in blocchi lunghi 64 bit, tale che

$$PS = PS0 \| PS1 \| PS2 \| .... \| PS_{BLOCKS-1}$$

Eseguiamo

$$A = KASUMI[A \oplus PS_n]_{IK}$$

$$B = B \oplus A$$

per ogni i, tale che $0 \leq n \leq BLOCKS - 1$.
A questo punto applichiamo una o più volte l'algoritmo KASUMI tale che

$$B = KASUMI[B]_{IK \oplus KM}$$

Il risultato di tale trasformazione sarà il blocco B lungo 64 bit che rappresenta il nostro MAC. Ma come detto in precedenza, questo dovrà essere lungo 32 bit, quindi applichiamo la seguente trasformazione

$$MAC - I = lefthalf[B]$$

che ci permette di avere il risultato richiesto.

3.3 Gestione della chiave in UMTS

Fino a questo punto non abbiamo mai trattato di come venga gestita o generata la chiave che verrà poi usata nelle funzioni $f8$ ed $f9$. La motivazione è molto semplice, non esiste un solo algoritmo od un solo modo. La spiegazione è presto data, bisogna permettere agli operatori di poterla salvare ed usarla al meglio nelle proprie infrastrutture già presenti e non c'è nessuna necessità che tra operatori diversi ci sia interoperabilità. Si tenga presente che l'operatore ha le seguenti necessità: generare la chiave per l'utente e autenticarlo. Questi algoritmi, di norma, vengono implementati all'interno del network dell'operatore, chiamato AuCs[12], e nella USIM[13] dell'utente. Ad ogni modo, alla conclusione della definizione delle due funzioni, il SAGE si preoccupò di definire anche un insieme di algoritmi che permettessero l'autenticazione e la generazione di chiavi. Questo insieme venne chiamato AKA. Nello stesso periodo, anche la GSM Association Security Group definì un nuovo insieme di algoritmi che servisse a rimpiazzare l'oramai insicuro COMP128-1, che chiamò COMP128-2. Visto la lunghezza e la diversità dell'argomento, tali spiegazioni non verranno trattate in questo documento. Una completa trattazione la troverete nel seguente libro [15].

[12] Authentication Centres
[13] Universal Subscriber Identity Modules

4 Conclusione

Per concludere, vediamo brevemente due aspetti che, in un certo qual modo, possono definirsi uno la conseguenza dell'altro. Nel primo vedremo gli studi di crittoanalisi effettuati in questi anni nei confronti dell'algoritmo KASUMI. Nel secondo verrà presentata una soluzione alternativa all'uso della crittografia simmetrica, tutt'ora usata per assicurare la confidenzialità o l'integrità dei messaggi nello standard GSM e UMTS, come si è visto nei precedenti capitoli.

4.1 Crittoanalisi

Il primo documento che tratta di crittoanalisi di KASUMI apparve nel 2001 a firma di Ulrich Kühn il quale riuscì a dimostrare che usando un attacco differenziale impossibile[14] e una versione ridotta a sei round dell'algoritmo era possibile ricavare la chiave con un tempo minore a quello necessario usando una ricerca completa, pari a 2^{128}. Un articolo successivo (2005) pubblicato dal gruppo di ricerca isreliano composto da Eli Biham, Orr Dunkelman e Nathan Keller dimostrò che era possibile, con l'uso di un attacco *related-key rectangle (boomerang)*[15], ridurre il tempo di ricerca della chiave, richiedendo solo $2^{54.6}$ testi in chiaro. La complessità di tale algoritmo è equivalente a $2^{76.1}$ esecuzioni dell'algoritmo KASUMI. Come si può immaginare questo attacco non è realmente praticabile, ma dimostrò che alcune pubblicazioni del gruppo in seno all'algoritmo KASUMI non erano valide. Lo stesso obiettivo fece nascere una nuova pubblicazione (2010), che portò a risultati davvero sorprendenti. Il gruppo composto da Dunkleman, Keller e Shamir[8] dimostrò che usando un attacco related-key era possibile ricavare completamente la chiave senza dover ricorrere alla ricerca completa. La complessità temporale e spaziale dell'attacco permise agli autori di recuperare la chiave in circa due ore con un computer desktop. Vediamo brevemente come il gruppo abbia portato a termine l'attacco. Il documento inizia col presentare un nuovo tipo di attacco, chiamato *sandwich*, il quale permise agli autori di costruire un semplice processo discriminante che portò la probabilità dei primi sette round a 2^{-14}. Usando tale discriminante e analizzando il round rimanente possiamo derivare la chiave lunga 128 bit usando solo 4 chiavi, 2^{26} dati, 2^{30} byte di memoria e 2^{32} secondi. Nella conclusione del documento, gli autori constatano che tale metodo non ha nessun effetto sull'algoritmo originale, MISTY1, il quale necessità ancora di una ricerca completa per ricavare la chiave. Questo, rimarcò il fatto che le scelte introdotte da SAGE indebolirono l'algoritmo originale.

[14] impossible differential attack - http://www.schneier.com/crypto-gram-9809.html#impossible

[15] http://www.cs.berkeley.edu/~daw/papers/boomerang-fse99.ps

4.2 Crittografia Asimmetrica

Un'alternativa alla crittografia simmetrica potrebbe essere l'uso della crittografia asimmetrica, già usata in molteplici campi, si pensi per esempio al protocollo HTTPS[16]. Il documento pubblicato nel 2003 da Constantinos F. Grecas, Sotirios I. Maniatis e Iakovos S. Venieris [9] illustra come tale cambiamento possa essere introdotto nelle attuali infrastrutture dell'operatore e quali siano le diverse parti interessate nel processo. In particolare, presenta come sia possibile introdurre la crittografia asimmetrica tra le tre componenti fondamentali (MS[17], VLR[18] e HLR[19]) presenti nella rete dell'operatore e come quest'ultimo debba gestire le chiavi. Infatti, l'operatore dovrà costituire una Public Key Infrastructure (PKI) atta a gestire le chiavi pubbliche e private che verranno usare tra VLR-HLR e MS-VLR. Senza addentrarci nei particolari che esulano dalla trattazione di questo documento, diciamo solo che le chiavi lato utente saranno salvate internamente alla SIM, in caso di rete GSM, o nella USIM, in caso di rete UMTS, e che l'operatore autenticherà l'utente attraverso tali parametri. Ad ora, non siamo a conoscenza di operatori e/o protocolli che implementino tale cambiamento. Probabilmente ciò è dovuto ai costi relativi al cambiamento e all'aumento di potenza computazionale richiesta a parità di servizio.

[16] http://tools.ietf.org/rfc/rfc2818.txt
[17] Mobile station
[18] Visitor location registe
[19] Home location register

Bibliografia

[1] 3GPP. General report on the design, speification and evaluation of 3gpp standard confidentiality and integrity algorithms. Technical Report 4.0, 2001.

[2] 3GPP. Specification of the 3gpp confidentiality and integrity algorithms (f8 and f9 specification). Technical Report 10, 3GPP, 03 2011.

[3] 3GPP. Specification of the 3gpp confidentiality and integrity algorithms (kasumi specification). Technical Report 10, 3GPP, 03 2011.

[4] Animesh Agarwal, Vaibhav Shrimali, and Manik Lal Das. Gsm security using identity-based cryptography. *Network*, page 10, 2009.

[5] Steve Babbage and Laurent Frisch. On misty1 higher order differential cryptanalysis. In *Proceedings of the Third International Conference on Information Security and Cryptology*, ICISC '00, pages 22–36, London, UK, UK, 2001. Springer-Verlag.

[6] Tomás Balderas-Contreras and René A. Cumplido-Parra. Security architecture in umts third generation cellular networks. Technical Report CCC-04-002, Coordinación de Ciencias Computacionales (INAOE), 02 2004.

[7] Eli Biham, Orr Dunkelman, and Nathan Keller. A related-key rectangle attack on the full kasumi. In *Proceedings of ASIACRYPT 2005, LNCS 3788*, pages 443–461. Springer-Verlag, 2005.

[8] Orr Dunkelman, Nathan Keller, and Adi Shamir. A practical-time attack on the a5/3 cryptosystem used in third generation gsm telephony. Cryptology ePrint Archive, Report 2010/013, 2010.

[9] Constantinos F. Grecas, Sotirios Maniatis, and Iakovos S. Venieris. Introduction of the asymmetric cryptography in gsm, gprs, umts, and its public key infrastructure integration. *MONET*, 8(2):145–150, 2003.

[10] Tetsu Iwata and Tadayoshi Kohno. New security proofs for the 3gpp confidentiality and integrity algorithms. In *Fast Software Encryption, FSE 2004*, pages 306–318. Springer-Verlag, 2003.

[11] Ju-Sung Kang, Okyeon Yi, Dowon Hong, and Hyunsook Cho. Pseudorandomness of misty-type transformations and the block cipher kasumi. In *Proceedings of the 6th Australasian Conference on Information Security and Privacy*, ACISP '01, pages 60–73, London, UK, UK, 2001. Springer-Verlag.

[12] Lars R. Knudsen and Chris J. Mitchell. An analysis of the 3gpp-mac scheme. *Electronic Notes in Discrete Mathematics*, 6:346–355, 2001.

[13] Mitsuri Matsui. New structure of block ciphers with provable security against differential and linear cryptanalysis. In *Proceedings of the Third International Workshop on Fast Software Encryption*, pages 205–218, London, UK, 1996. Springer-Verlag.

[14] NESSIE. *New European Schemes for Signatures, Integrity, and Encryption*. Springer-Verlag, 2004.

[15] Valtteri Niemi and Kaisa Nyberg. *UMTS security*. Wiley, 2003.

[16] Ju sung Kang, Sang uk Shin, Dowon Hong, and Okyeon Yi. Provable security of kasumi and 3gpp encryption mode f8. In *Proceedings of ASIACRYPT 2001, LNCS 2248*, pages 255–271. Springer-Verlag, 2001.