Agenzia per la
Cybersicurezza Nazionale

LINEE GUIDA FUNZIONI CRITTOGRAFICHE

Cifrari a flusso

MAGGIO 2026

Versione	Data di pubblicazione	Note
1.0	**12/05/2026**	**Prima pubblicazione**

Sommario

pag.

Scopo del documento 5
Lista dei simboli matematici utilizzati 6
1. Introduzione 7
2. Cifrari a flusso 8
2.1. Cifrari a flusso sincroni 8
2.2. Cifrari a flusso auto-sincronizzanti 9
2.3. Costruzione di cifrari a flusso 9
2.4. RC4 10
2.4.1. Specifiche tecniche 10
2.4.2. Sicurezza del cifrario 10
3. Cifrari a flusso basati su LFSR 12
3.1. Registri a scorrimento a retroazione lineari 12
3.1.1. Combinazione non lineare di LFSR 14
3.1.2. Generatore a filtro per un LFSR 14
3.1.3. Aggiornamento dipendente da LFSR 14
3.2. Registri a scorrimento a retroazione non lineari 15
3.3. E0 (Bluetooth) 15
3.3.1. Specifiche tecniche 15
3.3.2. Sicurezza del cifrario 15
4. Cifrari a flusso derivanti da cifrari a blocchi 16
4.1. Cipher feedback (CFB) 16
4.2. Output feedback (OFB) 16
4.3. Counter (CTR) 16
5. Attacchi ai cifrari a flusso 17
5.1. Strategie divide et impera 17
5.2. Attacchi dovuti alla correlazione 18
5.3. Attacchi indovina e determina 18
5.4. Crittoanalisi lineare e differenziale 18
5.5. Attacchi algebrici 18
6. Conclusioni 20
Bibliografia 21

Indice delle figure

pag.

Figura 1 - Schema di aggiornamento di un LFSR 13
Figura 2 - Esempio di un LFSR 13

Indice delle tabelle

Tabella 1 - Aggiornamento del LFSR rappresentato a sinistra 13

Scopo del documento

Questo documento costituisce parte della serie "Linee Guida Funzioni Crittografiche" e fornisce le raccomandazioni di ACN in merito ai **cifrari a flusso**, da adottare in tutti i contesti in cui si necessita di garantire la confidenzialità dei dati. Per ulteriori informazioni sulle Linee Guida e sulle utilità delle funzioni crittografiche in base ai contesti specifici, si faccia riferimento al documento introduttivo della serie [1].

Ogni documento tiene in considerazione le minacce presenti al giorno della sua pubblicazione. Data la diversa natura dei sistemi informativi di destinazione, non è possibile garantire che queste raccomandazioni possano essere utilizzate senza adattamenti specifici. In qualsiasi caso, la pertinenza dell'attuazione delle soluzioni proposte deve essere sottoposta, preventivamente, a valutazione e validazione da parte dei responsabili della sicurezza dei sistemi informativi di destinazione.

I contenuti delle Linee Guida sono indirizzati a sviluppatori, produttori di dispositivi e fornitori di servizi digitali, al fine di promuovere l'utilizzo di primitive crittografiche e relativi parametri di configurazione sicuri fin dalla fase di progettazione di prodotti, reti, applicazioni e servizi. Questi documenti sono altresì rivolti a security manager, referenti e responsabili della cybersicurezza di soggetti pubblici e privati affinché verifichino che i sistemi utilizzati dalla propria organizzazione siano conformi alle raccomandazioni fornite.

La legge 28 giugno 2024, n. 90 relativa a "Disposizioni in materia di rafforzamento della cybersicurezza nazionale e di reati informatici", all'articolo 9, stabilisce a tal fine che *«le strutture di cui all'articolo 8 della presente legge nonché quelle che svolgono analoghe funzioni per i soggetti di cui all'articolo 1, comma 2-bis, del decreto-legge 21 settembre 2019, n. 105, convertito, con modificazioni, dalla legge 18 novembre 2019, n. 133, e al decreto legislativo 18 maggio 2018, n. 65, verificano che i programmi e le applicazioni informatiche e di comunicazione elettronica in uso, che utilizzano soluzioni crittografiche, rispettino le linee guida sulla crittografia nonché quelle sulla conservazione delle password adottate dall'Agenzia per la cybersicurezza nazionale e dal Garante per la protezione dei dati personali e non comportino vulnerabilità note, atte a rendere disponibili e intellegibili a terzi i dati cifrati»*.

Il documento è stato curato dal Centro Nazionale di Crittografia istituito presso ACN.

Lista dei simboli matematici utilizzati

$\oplus$	Operazione XOR, cioè somma bit a bit tra stringhe binarie	swap	Funzione che scambia tra loro due valori
$\|\|$	Concatenazione di stringhe	maj	Maggioranza tra 3 bit, fornisce il valore binario più presente
$\mathrm{mod}\, n$	Riduzione modulo n, restituisce il resto della divisione per n	$\{0,1\}$	Campo binario dei valori assumibili da un singolo bit

1 Introduzione

La crittografia moderna consente di risolvere diverse problematiche riguardanti la sicurezza dei dati.
Uno degli obiettivi principali è assicurare la segretezza delle comunicazioni che avvengono tra due o più utenti nei confronti di entità malevole, quella che nella teoria dell'informazione viene definita confidenzialità dei messaggi.
I cifrari a flusso sono sistemi crittografici che permettono di assicurare tale proprietà e appartengono ai metodi di cifratura simmetrica, come la famiglia dei cifrari a blocchi, per i quali si rimanda al documento dedicato [2]. Si tratta quindi di metodi che cifrano e decifrano utilizzando la stessa chiave, che deve perciò essere condivisa tra i partecipanti e tenuta nascosta a qualsiasi entità esterna.
La principale differenza tra cifrari a blocchi e a flusso risiede nel fatto che i primi processano il messaggio dividendolo in blocchi di lunghezza prefissata mentre i secondi cifrano il messaggio agendo su un bit (o byte) alla volta. L'idea alla base dei cifrari a flusso è quella di generare una stringa di bit pseudocasuali, spesso lunga almeno quanto il messaggio, e di utilizzarla per cifrare e decifrare, solitamente calcolandone lo XOR con i dati in chiaro. Di conseguenza, i cifrari a flusso permettono di effettuare cifratura e decifratura anche solo parziali: è possibile, ad esempio, decifrare solo una parte di un documento invece che l'intero file. Allo stesso modo, la cifratura bit a bit consente di inviare i dati cifrati non appena vengono generati, senza dover attendere la cifratura dell'intero messaggio.

Queste caratteristiche rendono i cifrari a flusso particolarmente adatti per comunicazioni real-time, come chiamate vocali, streaming video o gaming online.
I cifrari a flusso solitamente presentano una struttura semplice, per cui sono spesso più facili da implementare e presentano esecuzioni più rapide rispetto a quelli a blocchi. Tuttavia, i cifrari a blocchi standardizzati, come AES, sono altamente ottimizzati a livello hardware e risultano quindi comparabili con i moderni cifrari a flusso anche a livello di prestazioni. Inoltre, esistono specifiche modalità di operazione che permettono di utilizzare un cifrario a blocchi come se fosse un cifrario a flusso.
Il documento presenta la seguente struttura: nel capitolo 2 sono riportate le definizioni principali e le caratteristiche generali relative ai cifrari a flusso, analizzando gli aspetti più importanti da un punto di vista crittografico; il capitolo 3 si incentra sui cifrari a flusso derivanti dalla combinazione di registri a scorrimento a retroazione lineari (LFSR), definendone la struttura e analizzandone le caratteristiche che li rendono sicuri; nel capitolo 4 si trattano invece le particolari modalità di operazione dei cifrari a blocchi che li trasformano, di fatto, in cifrari a flusso; nel capitolo 5 vengono riportate le principali strategie di attacco contro i cifrari a flusso, analizzando anche possibili contromisure e aspetti critici da tenere in considerazione; infine, nel capitolo 6 vengono riportate le conclusioni e le raccomandazioni da seguire quando si implementa un cifrario a flusso.

2 Cifrari a flusso

Un **cifrario a flusso** [3, 4] è un sistema crittografico che consente di assicurare la **confidenzialità** dei dati scambiati tra due utenti, impedendo quindi che un attore terzo possa accedere alle informazioni contenute nella conversazione. Per fare ciò, un cifrario a flusso utilizza una stringa K lunga k bit, detta **chiave**, nota solo ai due interlocutori, e genera una stringa di bit pseudocasuali detta keystream, indicata con Z. Questa stringa consente di cifrare il testo in chiaro M, solitamente calcolando lo XOR per ottenere il testo cifrato C:

$$C = c_1 c_2 \dots c_n = z_1 z_2 \dots z_n \oplus m_1 m_2 \dots m_n = Z \oplus M.$$

In tal caso, il cifrario a flusso si dice **binario additivo**. L'idea alla base di questa tipologia di cifrari deriva dal cifrario di **Vernam** o one-time pad, il quale prevede lo XOR tra il messaggio e una chiave segreta della stessa lunghezza. Per ulteriori dettagli a riguardo, si veda il documento introduttivo [1]. Nonostante one-time pad fornisca sicurezza incondizionata, lo scambio sicuro di una chiave così lunga è altamente inefficiente. I cifrari binari additivi risolvono tale problematica sostituendo la chiave segreta con una stringa pseudocasuale generata a partire da una chiave più corta. Dato che la funzione di cifratura è lo XOR con il testo in chiaro, il keystream generato deve essere lungo almeno quanto il messaggio.
Un cifrario a flusso è caratterizzato da uno **stato** che viene inizializzato a S_0 utilizzando la chiave K e, a ogni passo $i > 0$, viene aggiornato a S_i, da cui si estrae l'i-esimo bit del keystream. Spesso, lo stato iniziale è composto dall'unione di più **registri**, oltre a eventuali vettori di inizializzazione (IV) e contatori.
Il keystream può essere generato solo a partire dalla chiave, rendendo questo processo indipendente sia dal testo in chiaro che da quello cifrato. In questo caso si parla di cifrario a flusso **sincrono**. Un'altra possibilità è che la generazione del keystream dipenda anche da un certo numero di bit del testo in chiaro o del testo cifrato, caso in cui si parla di cifrario a flusso **auto-sincronizzante**. Queste due tipologie di cifrari sono spiegate nel dettaglio nelle prossime sezioni.

2.1 Cifrari a flusso sincroni

In un cifrario a flusso sincrono, la generazione del keystream dipende solo dalla chiave K, per cui la cifratura del bit m_i in posizione $i > 0$ si ottiene componendo le seguenti funzioni

$$\begin{aligned} S_i &= F(S_{i-1}, K), \\ z_i &= G(S_i, K), \\ c_i &= H(z_i, m_i), \end{aligned}$$

per cui, dopo aver inizializzato lo stato S_0, a ogni passo i si applica la funzione di aggiornamento F che restituisce il nuovo stato interno, il quale viene processato dalla funzione di generazione del keystream G per ricavare il bit z_i del keystream Z, che infine viene combinato al bit in chiaro m_i mediante la funzione H per ottenere il bit c_i del cifrato C.

Come già affermato precedentemente, in alcuni casi i dati vengono processati per byte invece che per bit e spesso la funzione H è semplicemente uno XOR.
Le principali caratteristiche dei cifrari a flusso sincroni sono:

- **necessità di sincronizzazione**: il mittente e il destinatario devono obbligatoriamente essere sincronizzati, ossia devono utilizzare la stessa chiave e partire dallo stesso stato iniziale. Se la sincronizzazione si perde, per esempio a causa di inserimento o cancellazione di bit nel cifrato, la decifratura fallisce. In questi casi, occorre adottare particolari tecniche addizionali che consentono di eseguire una ri-sincronizzazione;
- **assenza di propagazione di errore**: la modifica accidentale o intenzionale di un bit del cifrato causa un errore di decifratura del singolo bit, ma non compromette la corretta decifratura dell'intero messaggio.

Dal punto di vista degli **attacchi attivi**, la prima proprietà permette al ricevente di accorgersi del tentativo di aggiunta o rimozione di bit da parte di un avversario, al costo della decifratura errata dell'intero messaggio. L'assenza di propagazione degli errori consente invece a un avversario attivo di apportare cambiamenti al cifrato sapendo esattamente quali saranno le modifiche risultanti sul testo in chiaro. Per queste ragioni, quando si utilizza un cifrario a flusso di questo tipo, è importante adottare meccanismi aggiuntivi per garantire l'**integrità** e l'**autenticazione** dei dati

2.2 Cifrari a flusso auto-sincronizzanti

In un cifrario a flusso auto-sincronizzante, o asincrono, la generazione del keystream è una funzione della chiave e di un certo numero fissato di bit di testo cifrato già ottenuti. In particolare, la cifratura del bit m_i con $i > 0$ viene descritta dalle seguenti funzioni:

$$\begin{aligned} S_i &= c_{i-t}c_{i-t+1}\dots c_{i-1}, \\ z_i &= G(S_i, K), \\ c_i &= H(z_i, m_i), \end{aligned}$$

dove t è un valore prefissato che rappresenta il numero di bit del testo cifrato dai quali dipende l'aggiornamento del keystream, lo stato iniziale $S_0 = c_{-t+1}c_{-t+2}\dots c_0$ è composto da t bit ottenuti elaborando la chiave e gli input di inizializzazione, G è la funzione di generazione dei bit z_i del keystream Z e H è la funzione che combina il messaggio e il keystream per ottenere i bit c_i del testo cifrato C.

Le principali caratteristiche dei cifrari a flusso asincroni sono:

- **sincronizzazione automatica**: nel caso in cui si verifichi una perdita di sincronizzazione in seguito a inserimento o cancellazione di un bit del testo cifrato, il cifrario è in grado di ri-sincronizzarsi automaticamente, così che solamente una quantità limitata di bit del testo in chiaro risultino irrecuperabili;
- **propagazione di errore limitata**: la modifica di un solo bit del testo cifrato causa la decifratura errata dei successivi t bit di testo cifrato, ma poi la decifratura ritorna a essere corretta;
- **diffusione della statistica del linguaggio**: dato che ogni bit del testo in chiaro influenza la cifratura dei bit seguenti, le proprietà statistiche del messaggio si disperdono nel testo cifrato. Per questo motivo, i cifrari auto-sincronizzanti potrebbero essere più resistenti ad attacchi basati sulla ridondanza rispetto a quelli sincroni.

Le conseguenze dovute ad **attacchi attivi** sono diverse rispetto a quelle descritte per i cifrari sincroni: la prima proprietà rende più difficile il rilevamento da parte del destinatario di inserimenti o rimozioni di bit cifrati, in quanto la decifratura riprende automaticamente la sincronizzazione. Tuttavia, grazie alla propagazione di errore limitata, è più facile rilevare la manomissione di un bit nel testo cifrato, in quanto essa compromette anche la decifratura dei seguenti t bit. Di conseguenza, come nel caso precedente, si rende necessario l'utilizzo di meccanismi aggiuntivi che garantiscano **integrità** e **autenticazione** dei dati.

2.3 Costruzione di cifrari a flusso

A prescindere dalle tipologie descritte, esistono diverse strategie per sviluppare un cifrario a flusso. Le due opzioni più utilizzate sono tramite i cosiddetti registri di scorrimento a retroazione lineare (meglio conosciuti come LFSR) oppure sfruttando particolari modalità di funzionamento per cifrari a blocchi. Questi due casi vengono descritti nel dettaglio nei capitoli successivi.
In alcuni casi, sono stati sviluppati cifrari a flusso con strutture diverse, che spesso sfruttano particolari **funzioni non lineari** per l'aggiornamento dello stato interno. Un caso particolare è dato dai cifrari ARX (Addition, Rotation, XOR), che prendono il nome dalle tre operazioni svolte all'interno

della funzione non lineare utilizzata. Questa, seppur semplice da implementare, viene generalmente ripetuta per vari round in modo da rendere l'operazione computazionalmente irreversibile. Il risultato è un blocco di keystream da combinare con il testo in chiaro per ottenere il cifrato. Un cifrario ARX molto utilizzato è ChaCha [5], in particolare nella sua variante ChaCha20 (dove 20 rappresenta il numero di round) descritta nel dettaglio nel documento dedicato alla cifratura autenticata [6].
Nonostante queste soluzioni particolari possano risultare ottimali per le implementazioni in particolari contesti, si sconsiglia l'utilizzo di algoritmi non standard. La sezione seguente descrive uno di questi cifrari che ha avuto una particolare importanza storica ma che ora risulta obsoleto.

2.4 RC4

RC4 (Rivest Cipher 4) [7] è un cifrario a flusso sincrono sviluppato da Ron Rivest per conto della compagnia RSA Security nel 1987 ed era un segreto commerciale fino al 1994. Grazie alla sua semplicità, questo cifrario si è rapidamente diffuso ed è stato utilizzato in protocolli di sicurezza per le comunicazioni internet come SSL e TLS, ma anche nei primi protocolli per le reti wireless come WEP e WPA.

2.4.1 Specifiche tecniche

Per prima cosa, la chiave K di lunghezza k compresa tra 1 e 256 byte viene utilizzata per inizializzare un vettore di stato lungo 256 byte $S = S_0 \,\|\, S_1 \,\|\, \ldots \,\|\, S_{255}$. In particolare, questa fase di **inizializzazione** è composta dai seguenti passi:

1. i byte di S vengono riempiti con i valori da 0 a 255 in ordine crescente, ossia $S_i = i$;
2. si costruisce un vettore temporaneo T di 256 byte contenente la chiave. Se questa ha una lunghezza inferiore a 256 byte, allora viene ripetuta per il numero di volte necessario fino a riempire T, scartando eventualmente i bit in eccesso;
3. partendo da $i = 0$ e arrivando a 255, lo stato S viene aggiornato calcolando

$$\begin{aligned} j &= (j + S_i + T_i) \mod 256, \\ &\mathrm{swap}(S_i, S_j), \end{aligned}$$

dove la funzione swap scambia i due valori in input.

Una volta conclusa l'inizializzazione dello stato S, non si fa più uso della chiave K e si avvia la **generazione** del keystream Z. Se si intende cifrare un messaggio M lungo n byte, partendo da $i, j = 0$, per $l = 1, \ldots, n$, è necessario eseguire le seguenti operazioni

$$\begin{aligned} i &= (i + 1) \mod 256, \\ j &= (j + S_i) \mod 256, \\ &\mathrm{swap}(S_i, S_j), \\ Z_l &= S_{(S_i + S_j) \mod 256}. \end{aligned}$$

Il byte di keystream Z_l così ottenuto viene combinato tramite XOR con il byte M_l di testo in chiaro, così che il cifrato risulta essere

$$C = C_1 \,\|\, \ldots \,\|\, C_n = Z_1 \oplus M_1 \,\|\, \ldots \,\|\, Z_n \oplus M_n.$$

Come si può notare dalle specifiche, RC4 è un cifrario a flusso **sincrono**, in quanto la generazione del keystream è indipendente dal cifrato o dal messaggio in chiaro

2.4.2 Sicurezza del cifrario

A differenza dei cifrari a flusso più moderni, RC4 non prende un **nonce** in input. Questo significa che l'algoritmo di cifratura è **deterministico** e per cifrare in modo sicuro più messaggi con la stessa chiave è necessario specificare una tecnica di derivazione della chiave di sessione a partire da quella iniziale e da un nonce. Una semplice soluzione consiste nel generare la chiave di sessione calcolando l'hash della concatenazione chiave-nonce. Tuttavia, spesso questo processo viene sottovalutato e l'adozione di soluzioni deboli dà origine ad attacchi con **chiavi correlate**, come l'attacco Fluhrer, Mantin e Shamir [8] che è stato utilizzato per rompere il protocollo WEP delle connessioni Wi-Fi.
Essendo un cifrario a flusso, RC4 risulta particolarmente **malleabile** e quindi deve essere affiancato a un codice di autenticazione dei messaggi (MAC) sicuro. Senza questa precauzione, l'algoritmo risulta vulnerabile agli attacchi di **bit-flipping**, cioè quelli che permettono a un avversario di modificare il cifrato in modo che le differenze nel testo in chiaro siano prevedibili.
La vulnerabilità più importante di RC4 è dovuta alla correlazione tra i primi byte del keystream e delle prime permutazioni con la chiave, che consente di effettuare la **ricostruzione della chiave** a partire dallo stato interno [9].
Sempre considerando le correlazioni tra dati, è stato

dimostrato che il keystream generato da RC4 tende a convergere verso delle sequenze specifiche ed è quindi possibile effettuare con successo il **riconoscimento** del cifrato rispetto a testi casuali [10].
Inoltre, nel contesto TLS, qualora un attaccante riesca a intercettare una grande quantità di cifrati, allora potrebbe essere in grado di eseguire degli attacchi che sfruttano l'analisi statistica sulle possibili chiavi RC4 per recuperare i testi in chiaro [11]. Per questo motivo, l'utilizzo di RC4 in TLS è considerato **obsoleto** dal 2015 [12].

RC4 è stato descritto a scopo puramente esemplificativo. Per via delle suddette vulnerabilità, RC4 è ritenuto **insicuro** e il suo utilizzo è **fortemente sconsigliato**.

3 Cifrari a flusso basati su LFSR

Tra le possibili componenti per la costruzione di funzioni generatrici di keystream, le più famose sono sicuramente i **registri di scorrimento a retroazione lineari** (LFSR, dall'inglese Linear Feedback Shift Register). Si tratta di particolari generatori deterministici di numeri casuali, che vengono utilizzati spesso nei cifrari a flusso in quanto non richiedono grandi capacità computazionali. Inoltre, se impostati opportunamente, consentono di generare keystream molto lunghi prima di dover essere inizializzati nuovamente. Tuttavia, essendo funzioni **lineari**, gli output degli LFSR risultano facili da prevedere, e quindi devono essere introdotte delle tecniche per combinarli, come l'utilizzo di componenti non lineari, in modo da poter ottenere un cifrario a flusso sicuro.

3.1 Registri a scorrimento a retroazione lineari

Un **LFSR** è un registro contenente una stringa di bit che vengono aggiornati in modo deterministico tramite una particolare funzione lineare. Possiamo immaginare un LFSR come un nastro finito di bit dal quale, a ogni iterazione, si estrae il bit nella posizione più a destra. Per fare ciò, si calcola una particolare combinazione lineare dei bit contenuti nel registro e si inserisce il risultato nella posizione più a sinistra, dopo aver fatto scorrere tutti gli altri bit verso destra. Il bit che si trovava nella posizione più a destra è il valore estratto, che andrà a comporre il keystream del LFSR. Più formalmente, un LFSR è definito da uno stato $S = s_1 \dots s_l$ composto da l bit e da un aggiornamento lineare P, chiamato anche **clock**. Questo viene solitamente rappresentato tramite un polinomio di grado l

$$p(x) = x^l + p_l x^{l-1} + \dots + p_2 x + p_1,$$

con coefficienti binari $p_i \in \{0, 1\}$.

La rappresentazione degli LFSR descritta in questo documento colloca il bit più significativo dello stato (s_1) a destra del registro e il bit meno significativo (s_l) a sinistra. Esistono tuttavia rappresentazioni alternative, che ad esempio collocano il bit meno significativo a destra. Allo stesso modo, l'estrazione del bit per il keystream qui viene descritta a sinistra, ma può avvenire a destra. Anche per il polinomio associato si può utilizzare una notazione alternativa che utilizza il **reciproco**, cioè il polinomio ottenuto da $p(x)$ invertendo l'ordine delle potenze

$$p^*(x) = 1 + p_l x + \dots + p_2 x^{l-1} + p_1 x^l = x^l p(x^{-1}),$$

il quale ha sempre termine noto non nullo e può avere grado inferiore alla lunghezza dello stato.

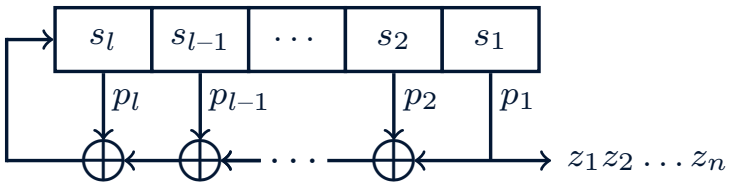


Figura 1 - Schema di aggiornamento di un LFSR

Prima di procedere con l'estrazione del keystream, si inizializza il registro utilizzando la chiave segreta e altri eventuali parametri dipendenti dal sistema preso in considerazione. Quindi, per ogni passo j da 1 alla lunghezza desiderata n del keystream, il processo è:

1. si calcola
$$P(S) = p_l s_l \oplus p_{l-1} s_{l-1} \oplus \ldots \oplus p_2 s_2 \oplus p_1 s_1,$$
cioè lo XOR tra le combinazioni dei coefficienti del polinomio binario $p(x)$ e i bit nel registro. Le posizioni dei bit che influenzano l'aggiornamento del registro vengono chiamati **tap**;
2. si estrae il bit più a destra $z_j = s_1$;
3. si aggiornano i bit per i da 2 a l facendo scorrere i valori nel registro verso destra, cioè $s_i = s_{i+1}$;
4. infine, il bit più a sinistra assume il valore calcolato al passo 1, cioè $s_l = P(S)$.

Il processo è rappresentato in Figura 1. Si può notare come le potenze di x nel polinomio $p(x)$ che hanno coefficiente non nullo indicano le posizioni all'interno del registro relative ai bit il cui XOR genera il nuovo bit del LFSR a ogni passo. Per questo motivo, si deve evitare uno stato iniziale nullo, che comporterebbe un keystream di soli zeri.

Per un esempio pratico, si consideri un LFSR con tre bit, che parte dallo stato 100 e con il clock definito dal polinomio $x^3 + x + 1$, cioè dato da $s_2 \oplus s_1$. In Figura 2 viene rappresentato il registro inizializzato e le operazioni del clock, mentre la Tabella 1 raccoglie i valori passo a passo. Il keystream risultante è quindi $Z = 1001011$ e si può notare che, al passo 7, lo stato interno del registro assume il valore iniziale e quindi il keystream inizia a ripetersi dal passo 8. Il comportamento osservato nell'esempio si verifica anche in generale: un registro di lunghezza l possiede un numero finito di valori (non nulli) possibili all'interno del registro. Conoscere tale valore è importante per evitare di raggiungere stati già utilizzati e quindi, data la natura lineare degli LFSR, cifrare più parti del messaggio con lo stesso keystream.

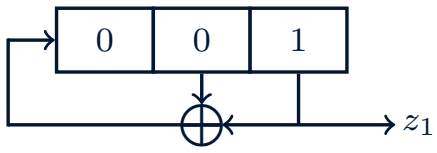


Figura 2 - Esempio di un LFSR

j	s_3	s_2	s_1	z_j
0	0	0	1	–
1	$0 \oplus 1 = 1$	0	0	1
2	$0 \oplus 0 = 0$	1	0	0
3	$1 \oplus 0 = 1$	0	1	0
4	$0 \oplus 1 = 1$	1	0	1
5	$1 \oplus 0 = 1$	1	1	0
6	$1 \oplus 1 = 0$	1	1	1
7	$1 \oplus 1 = 0$	0	1	1
8	$0 \oplus 1 = 1$	0	0	1

Tabella 1 - Aggiornamento del LFSR rappresentato a sinistra

Il numero di clock prima che si verifichi una ripetizione è detto **periodo** e può avere un valore massimo pari a $2^l - 1$. Il periodo di un LFSR, così come altre proprietà, si può determinare tramite dei metodi algebrici a partire dalla configurazione iniziale e dal polinomio che determina l'aggiornamento [13]. Ad esempio, nel caso analizzato in precedenza, il periodo è $7 = 2^3 - 1$, che corrisponde al massimo valore raggiungibile da un LFSR di lunghezza $l = 3$.
Gli LFSR sono ampiamente utilizzati nella costruzione di cifrari a flusso in quanto si basano su operazioni molto semplici che risultano facilmente implementabili su hardware. Inoltre, se impostati correttamente, gli LFSR possono produrre sequenze con periodi molto lunghi e con buone proprietà statistiche. Infine, essendo dipendenti da strutture algebriche, lo studio della loro sicurezza si può eseguire semplicemente tramite l'utilizzo di tecniche algebriche.
Tuttavia, come già osservato precedentemente, il funzionamento degli LFSR è lineare e quindi i keystream generati risultano estremamente prevedibili. Di conseguenza, non è sicuro utilizzarli singolarmente, ma è necessario introdurre delle operazioni **non lineari**. Queste possono essere:

- una funzione non lineare per combinare output di LFSR;
- una funzione non lineare per filtrare il risultato di un LFSR;
- alcuni LFSR per determinare l'aggiornamento in un LFSR;
- registri non lineari (NLFSR) al posto di alcuni LFSR.

Nella progettazione dei cifrari a flusso di questo tipo, si utilizza spesso una combinazione delle tecniche precedenti, in modo da migliorare notevolmente la sicurezza del cifrario.

3.1.1 Combinazione non lineare di LFSR

Una delle strategie più comuni per implementare un cifrario a flusso è considerare diversi LFSR per poi utilizzare una funzione non lineare per combinarli in un unico keystream. In maniera più rigorosa, se si vuole ottenere un output sicuro da N diversi LFSR, si considera una funzione booleana G altamente non lineare, chiamata funzione di **combinazione** (combining function), che prende un bit da ogni LFSR e restituisce il bit del keystream per la cifratura. In formule, indicando con $Z^{(i)}$ i keystream dei singoli LFSR,

$$Z = G\big(Z^{(1)}, \dots, Z^{(N)}\big).$$

Scegliendo opportunamente la funzione di combinazione non lineare si possono evitare diversi attacchi, tra cui quelli di correlazione o di ricostruzione lineare del sistema.
Per maggiori dettagli sugli attacchi si veda il capitolo 5.

3.1.2 Generatore a filtro per un LFSR

Un altro modo per trasformare un LFSR è quello di usare un **generatore a filtro** (filter generator). In questo caso, invece di estrarre direttamente un bit dallo stato del LFSR dopo ogni clock, viene applicata una funzione booleana non lineare che prende in input tutti i bit dello stato interno e restituisce il bit del keystream in output. In formule, scelta una funzione non lineare G, si ha

$$Z = G(S).$$

In questo caso, l'aggiornamento dello stato S continua a seguire le regole standard di clock degli LFSR, e quindi resta lineare.

3.1.3 Aggiornamento dipendente da LFSR

Questa strategia consiste nel far dipendere il clock di un LFSR da informazioni ricavate da altri LFSR. In questo caso si parla di **generatore a clock controllato** (clock-controlled generator) e la nonlinearità del keystream risultante è data dall'irregolarità dell'aggiornamento modificato.
Per esempio, si può considerare il caso del cifrario a flusso **A5/1**, in cui sono presenti tre LFSR nei quali l'aggiornamento avviene secondo una particolare funzione di controllo: in ognuno dei tre registri si sceglie un bit, rispettivamente indicato con $s^{(1)}$, $s^{(2)}$ e $s^{(3)}$, che assume il ruolo di bit di controllo o **clocking bit**. Questi valori vengono presi in input dalla funzione di maggioranza data da

$$\mathrm{maj}\big(s^{(1)}, s^{(2)}, s^{(3)}\big) = \begin{cases} 1, & \text{se almeno due dei bit sono } 1, \\ 0, & \text{altrimenti.} \end{cases}$$

A questo punto, ognuno dei tre registri si aggiorna solo se il valore attuale del suo clocking bit rientra nella maggioranza, cioè se $s^{(i)} = \mathrm{maj}\big(s^{(1)}, s^{(2)}, s^{(3)}\big)$ si esegue il clock nel registro i. In questo modo, ad ogni istante verranno aggiornati o tutti o solo due registri. Infine, i tre bit in output vengono combinati tramite XOR per ottenere il bit del keystream finale. Nonostante questa costruzione possa sembrare abbastanza imprevedibile, il cifrario A5/1 è considerato **insicuro**, in particolare rispetto ad attacchi KPA (Known-Plaintext Attack).

3.2 Registri a scorrimento a retroazione non lineari

Un **registro a scorrimento a retroazione non lineare** (NLFSR) ha la stessa struttura di un LFSR ma presenta un aggiornamento non lineare dello stato interno.
Di conseguenza, un NLFSR si comporta in modo analogo a quanto descritto nella sezione 3.1, con l'unica differenza che lo XOR tra i bit del registro viene sostituito da una funzione più complessa. Per esempio, possono essere utilizzate funzioni che applicano AND e XOR, come nel caso del NLFSR di Grain [14], un famoso cifrario a flusso.
Una situazione molto comune per ottenere un cifrario a flusso sicuro è combinare LFSR, semplici da implementare, con gli NLFSR, che introducono nonlinearità. In questo modo, l'aggiornamento totale del sistema risulta più difficile da prevedere per un attaccante.

3.3 E0 (Bluetooth)

Un importante cifrario a flusso basato su LFSR è **E0** [15], progettato appositamente per essere utilizzato nel protocollo **Bluetooth** [16]. La parte principale dell'algoritmo è costituita dalla generazione del keystream, che si ottiene tramite l'utilizzo di quattro diversi LFSR indipendenti tra loro e combinati utilizzando un particolare NLFSR che agisce su uno stato di 4 bit.

3.3.1 Specifiche tecniche

E0 genera un keystream a partire da una chiave segreta di 128 bit, operando su uno stato di 132 bit tramite quattro LFSR combinati tramite una funzione chiamata **summation combiner**, che esegue lo XOR con il risultato del NLFSR.
In particolare, lo stato interno dell'intero cifrario a flusso viene inizializzato prendendo come input la chiave e altri dati dipendenti dal protocollo Bluetooth (indirizzo e master counter). Il risultato può essere suddiviso in

$$S = s_1 \dots s_{25} \,\|\, t_1 \dots t_{31} \,\|\, u_1 \dots u_{33} \,\|\, v_1 \dots v_{39} \,\|\, w_1 \dots w_4,$$

dove con s_i, t_i, u_i, v_i sono indicati rispettivamente gli stati di partenza dei quattro LFSR e con w_i quelli del NLFSR.
Gli LFSR utilizzati in E0 vengono aggiornati secondo le regole definite dai seguenti polinomi:

$$\begin{aligned} p_1(x) &= x^{25} + x^{17} + x^{13} + x^5 + 1, \\ p_2(x) &= x^{31} + x^{19} + x^{15} + x^7 + 1, \\ p_3(x) &= x^{33} + x^{29} + x^9 + x^5 + 1, \\ p_4(x) &= x^{39} + x^{35} + x^{11} + x^3 + 1. \end{aligned}$$

A differenza degli LFSR classici in cui si preleva il bit nell'ultima posizione, in E0 si estraggono i bit in specifiche posizioni intermedie ottenendo, a ogni passo $j > 0$,

$$z_j^{(1)} = s_2, \quad z_j^{(2)} = t_8, \quad z_j^{(3)} = u_2, \quad z_j^{(4)} = v_8.$$

Questi bit vengono presi in input dal summation combiner assieme ai w_i. L'operatore restituisce il bit del keystream

$$z_j = z_j^{(1)} \oplus z_j^{(2)} \oplus z_j^{(3)} \oplus z_j^{(4)} \oplus w_1,$$

e inoltre aggiorna i w_i tramite il NLFSR. Nello specifico, questo agisce su $w_1w_2w_3w_4$ combinando le coppie di bit w_1w_2 e w_3w_4 ai risultati $z_j^{(i)}$ degli LFSR mediante operazioni non lineari. Il risultato è una coppia di bit che viene salvata in w_1w_2, mentre w_3w_4 assumono i vecchi valori di w_1w_2.

3.3.2 Sicurezza del cifrario

Data la rapida espansione dell'utilizzo del Bluetooth nei primi anni successivi alla sua introduzione, la crittoanalisi dei sistemi che ne determinano la sicurezza è stata sempre molto attiva. Già agli inizi degli anni 2000 si sono ottenuti i primi risultati sulla **riduzione di sicurezza** a 65 bit, a prescindere dalla lunghezza della chiave utilizzata [17].
In seguito, sono stati sviluppati altri attacchi di tipo KPA che consentono di **ricostruire la chiave** segreta a partire dai primi 24 bit di un numero computazionalmente ottenibile di comunicazioni Bluetooth [18].
Nonostante tutto, E0 continua a essere presente nella suite crittografica del protocollo Bluetooth, per consentire la comunicazione con i sistemi più obsoleti.

E0 è stato descritto a scopo puramente esemplificativo. Per via delle suddette vulnerabilità, E0 è ritenuto **insicuro** e il suo utilizzo è **fortemente sconsigliato**.

4 Cifrari a flusso derivanti da cifrari a blocchi

In alternativa alle tecniche di costruzione basate su LFSR, si possono ottenere cifrari a flusso a partire dai cifrari a blocchi adottando particolari **modalità di funzionamento** che permettono la cifratura bit a bit (o byte a byte). Vista la sicurezza e l'efficienza di AES come cifrario a blocchi, questa possibilità è oggigiorno piuttosto diffusa e considerata molto sicura. Tra le più comuni modalità di operazione di questo tipo ci sono CFB, OFB e CTR.
Nelle seguenti sezioni si richiamano solamente i dettagli principali, per le descrizioni approfondite si rimanda al documento dedicato ai cifrari a blocchi [2].

4.1 Cipher feedback (CFB)

La modalità **CFB** [19, 20] permette di utilizzare un cifrario a blocchi come un cifrario a flusso e quindi ottiene il cifrato come XOR tra testo in chiaro e keystream. In particolare, quest'ultimo è la concatenazione dei blocchi $Z_1, Z_2, \ldots, Z_n$ ottenuti cifrando un IV al primo passo e, nei passi successivi, il blocco di testo cifrato al passo precedente.

4.2 Output feedback (OFB)

La modalità **OFB** [19, 20] genera un keystream come concatenazione di blocchi $Z_1, Z_2, \ldots, Z_n$. Il processo inizia come per CFB, in quanto il primo blocco si ottiene cifrando un IV, ma ogni blocco successivo si ottiene cifrando quello precedente. La cifratura del messaggio è infine lo XOR con il keystream così ottenuto.

4.3 Counter (CTR)

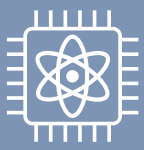
La modalità **CTR** [19, 20] genera i blocchi $Z_1, Z_2, \ldots, Z_n$ che compongono il keystream a partire da un counter (contatore) che viene aggiornato a ogni passo, incrementando il suo valore a ogni iterazione. Come per le modalità precedenti, i blocchi di testo cifrato si ottengono semplicemente calcolando lo XOR tra il keystream così generato e il messaggio in chiaro.
Questa modalità risulta una delle più utilizzate in molti dei contesti in cui è necessario un cifrario a flusso, come nelle connessioni Wi-Fi o 5G.

5 Attacchi ai cifrari a flusso

In questo capitolo si descrivono le principali tecniche di **crittoanalisi** utilizzabili per attaccare i cifrari a flusso. Data la natura informativa di questo documento, non si forniscono dettagli tecnici su attacchi specifici e si rimanda alla letteratura per ulteriori approfondimenti [21].
Gli attacchi ai cifrari a flusso mirano a ricostruire la chiave segreta a partire da uno specifico insieme di informazioni e richiedono particolari requisiti di potenza computazionale. Per ulteriori dettagli sugli scenari e sulla complessità degli attacchi, si rimanda al documento introduttivo [1].
Per quanto riguarda i cifrari a flusso, bisogna tenere in considerazione che la struttura dell'algoritmo è conosciuta e quindi gli attacchi generalmente si dividono in una fase di computazione preliminare, svolta **offline**, e una attiva che si svolge **online**. Questo consente di ottimizzare i requisiti di tempo e memoria eseguendo a priori tutte le operazioni valide sul cifrario in generale, per poi passare all'attacco sull'istanza specifica.
Dato che nei cifrari a flusso keystream e testo in chiaro vengono combinati bit a bit (o byte a byte), è facile per un attaccante apportare modifiche al testo cifrato così che la decifratura del testo in chiaro possa essere modificata a piacere da un malintenzionato. Ad esempio, se il valore cifrato rappresenta l'importo di una transazione, la semplice modifica di uno dei bit più significativi comporta una grande variazione nella cifra del denaro inviato. In questi casi si parla di un cifrario **malleabile**.
La più semplice contromisura alla malleabilità di un cifrario è considerare una soluzione che assicuri l'**autenticazione** dei dati oltre alla loro confidenzialità. Tale funzione viene svolta, ad esempio, dai codici di autenticazione di messaggi (MAC) o dagli algoritmi di cifratura autenticata. Per ulteriori dettagli a riguardo, si rimanda ai rispettivi documenti [6, 22].

5.1 Strategie divide et impera

Suddividere il problema alla base di un cifrario in istanze più piccole consente di ridurre la complessità computazionale degli attacchi. Questa è l'idea alla base delle strategie **divide et impera** (divide and conquer, in inglese) [23]: se l'attaccante riesce a ottenere un'equivalenza tra il problema principale e la combinazione di differenti problemi minori, la complessità dell'attacco si riduce notevolmente.
La realizzabilità di un divide et impera dipende dalla struttura stessa del cifrario. Le componenti ad alto rischio sono ovviamente quelle che utilizzano la chiave segreta, cosa che spesso accade solo nell'algoritmo di inizializzazione dello stato interno (come nel caso degli LFSR). Ad esempio, se ogni bit dello stato iniziale dipende da una selezione dei bit della chiave, si potrebbe suddividere il problema iniziale in tanti problemi minori.
Per contrastare questa tipologia di attacchi è necessario un'analisi approfondita sulla struttura del cifrario. Per questo motivo, si sconsiglia l'utilizzo di soluzioni fai-da-te in quanto poco studiate dalla comunità scientifica.



Minaccia quantistica

Come la maggior parte della crittografia simmetrica, i cifrari a flusso risultano suscettibili agli attacchi perpetrati da un computer quantistico tramite l'**algoritmo di Grover** [24], che tuttavia garantisce al più un aumento quadratico della velocità degli attacchi di forza bruta. Quindi, un raddoppio delle dimensioni della chiave è sufficiente a garantire lo stesso livello di sicurezza degli standard attuali.

5.2 Attacchi dovuti alla correlazione

In statistica, si parla di **correlazione** quando una variabile tende a cambiare in funzione di un'altra. Questo può verificarsi in un cifrario quando l'output risulta altamente influenzato da alcuni bit dello stato interno.
In tal caso, un attaccante può effettuare un'analisi statistica per determinare l'informazione contenuta in questi bit. Così facendo, riesce a ridurre lo spazio delle possibili chiavi segrete e quindi migliorare l'efficienza dell'attacco.
I cifrari a flusso possono essere vulnerabili ad attacchi di correlazione, in particolare quando si hanno cifrari derivanti da LFSR. Questo è dovuto al fatto che, se un bit viene cifrato con uno specifico stato, questo verrà aggiornato nella cifratura successiva che quindi dipende da valori di bit già usati in precedenza [25].
La componente fondamentale che garantisce la resistenza agli attacchi di correlazione è la **funzione di combinazione**. In particolare, se questa funzione booleana ha buone probabilità statistiche, il cifrario che ne deriva risulterà resistente a tali attacchi.
In termini algebrici, si cercano funzioni il più possibili bilanciate, cioè che restituiscano in output lo stesso numero di 0 e di 1, ma anche **immuni alle correlazioni** (o correlation immune), cioè che non vi sia una corrispondenza tra gli input della funzione e l'output.
Esistono anche varianti più evolute di questo attacco che utilizzano tecniche di teoria dei codici per ottimizzare la computazione e renderlo più efficiente. In questo caso si parla di **fast correlation attack** [26].

Nuovamente, risulta fondamentale evitare soluzioni fai-da-te per evitare che si presentino correlazioni indesiderate. Gli algoritmi standard, al contrario, hanno passato diversi studi sulle proprietà statistiche e quindi possono essere ritenuti sicuri da questo punto di vista.

5.3 Attacchi indovina e determina

Nel caso in cui un cifrario a flusso non utilizzi l'intero stato interno per ottenere il keystream, può presentarsi una vulnerabilità agli attacchi di tipo **indovina e determina** (guess and determine, in inglese). In particolare, se solo alcuni bit dello stato interno vengono presi come input della funzione di generazione, allora un attaccante può provare a indovinarne i valori confrontando il bit di keystream risultante con un valore intercettato. Una volta trovata una corrispondenza, è sufficiente aggiornare lo stato e passare al bit successivo, provando di nuovo a indovinare eventuali valori mancanti [27].
Per evitare che un attacco di questo tipo riduca la complessità della rottura di un cifrario, e quindi la sua sicurezza, si consiglia di utilizzare sempre tutti i bit dello stato interno e, in generale, adottare sistemi di cifratura riconosciuti sicuri dalla comunità scientifica e che quindi non presentano queste vulnerabilità.

5.4 Crittoanalisi lineare e differenziale

La crittoanalisi **lineare** e quella **differenziale** sono tra i metodi di crittoanalisi più diffusi applicabili ai cifrari a flusso, ma anche a cifrari a blocchi o a funzioni di hash. La prima sfrutta possibili relazioni lineari tra i bit del testo in chiaro, del testo cifrato e di una chiave di round, al fine di ricavare quest'ultima; la seconda invece si basa sull'idea di studiare la propagazione di differenze in input nelle corrispondenti differenze in output. Per una descrizione approfondita di queste tecniche, si rimanda al documento dedicato ai cifrari a blocchi [2].

5.5 Attacchi algebrici

Alcuni cifrari a flusso utilizzano una o più componenti lineari per poi combinarle non linearmente (come avviene con gli LFSR). In questi casi, è possibile modellare gli operatori lineari come funzioni booleane, ovvero a coefficienti binari, e quindi l'intercettazione di più bit permette di comporre un sistema di equazioni booleane. Questa è la base degli

attacchi algebrici, che mirano a risolvere il sistema per risalire agli input e quindi alla chiave segreta [28].
La resistenza a questa tipologia di attacchi risiede nel corretto utilizzo della funzione di combinazione non lineare: se questa non lascia trapassare informazioni sugli output delle componenti lineari, allora un attaccante non può intercettare i bit necessari per l'attacco.
Un altro aspetto che può introdurre vulnerabilità algebrica è il **crittoperiodo** della chiave, cioè l'intervallo di tempo oltre il quale è necessario sostituirla. Infatti, più si utilizza la stessa chiave per effettuare cifrature distinte, più si forniscono dati correlati allo stesso segreto. Questo è un problema in generale, ma per i cifrari a flusso esistono algoritmi specifici in grado di ricostruire l'operazione lineare semplicemente avendo a disposizione un numero sufficiente di bit del keystream. Un importante esempio è l'algoritmo di Berlekamp-Massey [29], che consente di ottenere il più piccolo LFSR che genera un dato keystream.

6 Conclusioni

I cifrari a flusso offrono una maggiore semplicità di implementazione, specialmente in ambienti hardware a bassa complessità, e risultano generalmente più veloci nell'elaborazione di dati in tempo reale. Tuttavia, i cifrari a blocchi sono più ampiamente studiati, supportati da implementazioni ottimizzate e, se utilizzati con modalità operative adeguate, offrono una sicurezza più robusta contro diverse tipologie di attacchi.
Per questo motivo, l'uso di **cifrari a blocchi** standardizzati è preferibile, eventualmente impiegando modalità di funzionamento che li trasformano in cifrari a flusso.

Inoltre, come osservato precedentemente, i cifrari a flusso assicurano solamente la confidenzialità e risultano quindi malleabili. Pertanto, in caso di trasmissione dei dati cifrati, si **raccomanda** di adottare soluzioni che garantiscano anche l'autenticazione e l'integrità dei dati, come quelle di **cifratura autenticata**.
Infine è importante utilizzare solamente cifrari standardizzati seguendo attentamente le specifiche tecniche, in quanto l'impiego di una qualsiasi forma di cifrario o adattamento fai-da-te potrebbe introdurre vulnerabilità indesiderate.

Si **raccomanda** di adottare sempre una delle soluzioni che permettono la **cifratura autenticata** descritte nel documento dedicato [6].
L'uso di cifrari a flusso per il solo scopo di confidenzialità è ammesso solo in assenza di trasmissione dei dati cifrati. Per questi casi, si raccomandano le modalità di funzionamento per i cifrari a blocchi raccomandate nel documento dedicato [2].

Bibliografia

[1] ACN. *Introduzione alla Crittografia e alle Linee Guida*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

[2] ACN. *Cifrari a Blocchi e Modalità di Funzionamento*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

[3] A. J. Menezes, P. C. V. Oorschot e S. A. Vanstone. *Handbook of applied cryptography (1st edition)*. CRC Press, 1997. DOI: 10.1201/9780429466335.

[4] D. R. Stinson e M. Paterson. *Cryptography: Theory and Practice (4th edition)*. Chapman e Hall/CRC, 2017. DOI: 10.1201/9781315282497.

[5] D. J. Bernstein. *ChaCha, a variant of Salsa20*. The State of the Art of Stream Ciphers. 2008. URL: https://cr.yp.to/chacha/chacha-20080120.pdf.

[6] ACN. *Cifratura Autenticata*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

[7] B. Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C, Second Edition*. John Wiley & Sons Inc, 1996. ISBN: 9780471117094.

[8] S. Fluhrer, I. Mantin e A. Shamir. «Weaknesses in the Key Scheduling Algorithm of RC4». In: *Selected Areas in Cryptography*. Lecture Notes in Computer Science. 2001, pp. 1–24. DOI: 10.1007/3-540-45537-X_1.

[9] E. Biham e Y. Carmeli. «Efficient Reconstruction of RC4 Keys from Internal States». In: *Fast Software Encryption (FSE)*. Lecture Notes in Computer Science. 2008, pp. 270–288. DOI: 10.1007/978-3-540-71039-4_17.

[10] I. Mantin e A. Shamir. «A Practical Attack on Broadcast RC4». In: *Fast Software Encryption (FSE)*. Lecture Notes in Computer Science. 2002, pp. 152–164. DOI: 10.1007/3-540-45473-X_13.

[11] P. Sepehrdad, S. Vaudenay e M. Vuagnoux. «Discovery and Exploitation of New Biases in RC4». In: *Selected Areas in Cryptography*. Lecture Notes in Computer Science. 2010, pp. 74–91. DOI: 10.1007/978-3-642-19574-7_5.

[12] A. Popov. *Prohibiting RC4 Cipher Suites*. RFC 7465. 2015. DOI: 10.17487/RFC7465.

[13] R. Lidl e H. Niederreiter. *Introduction to Finite Fields and their Applications*. Cambridge University Press, 1994. DOI: 10.1017/CBO9781139172769.

[14] M. Hell, T. Johansson e W. Meier. «Grain: a stream cipher for constrained environments». In: *International Journal of Wireless and Mobile Computing* 2.1 (2007), pp. 86–93. DOI: 10.1504/IJWMC.2007.013798.

[15] C. Fontaine. «E0 (Bluetooth)». In: *Encyclopedia of Cryptography and Security*. Springer, 2005, pp. 169–169. DOI: 10.1007/0-387-23483-7_117.

[16] Bluetooth. *Core Specification 6.1*. 2025. URL: https://www.bluetooth.com/specifications/specs/core-specification-6-1/.

[17] S. Fluhrer. *Improved key recovery of level 1 of the Bluetooth Encryption System*. Cryptology ePrint Archive, Paper 2002/068. 2002. URL: https://eprint.iacr.org/2002/068.

[18] Y. Lu, W. Meier e S. Vaudenay. «The Conditional Correlation Attack: A Practical Attack on Bluetooth Encryption». In: *Advances in Cryptology - CRYPTO 2005*. Lecture Notes in Computer Science. 2005, pp. 97–117. DOI: 10.1007/11535218_7.

[19] ISO. *Information technology – Security techniques – Modes of operation for an n-bit block cipher*. ISO/IEC 10116. 2017. URL: https://www.iso.org/standard/64575.html.

[20] M. Dworkin. *Recommendation for Block Cipher Modes of Operation: Methods and Techniques*. SP 800-38A. NIST, 2001. DOI: 10.6028/NIST.SP.800-38A.

[21] R. Verdult. *Introduction to Cryptanalysis: Attacking Stream Ciphers*. 2015. URL: https://www.cs.ru.nl/~rverdult/Introduction_to_Cryptanalysis-Attacking_Stream_Ciphers.pdf.

[22] ACN. *Codici di Autenticazione di Messaggi (MAC)*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

[23] E. Dawson e A. C. and. «Divide and Conquer Attacks on Certain Classes of Stream Ciphers». In: *Cryptologia* 18.1 (1994), pp. 25–40. DOI: 10.1080/0161-119491882757.

[24] L. K. Grover. «A fast quantum mechanical algorithm for database search». In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing (STOC)*. 1996, pp. 212–219. DOI: 10.1145/237814.237866.

[25] T. Siegenthaler. «Decrypting a Class of Stream Ciphers Using Ciphertext Only». In: *IEEE Transactions on Computers* C-34.1 (1985), pp. 81–85. DOI: 10.1109/TC.1985.1676518.

[26] W. Meier e O. Staffelbach. «Fast correlation attacks on certain stream ciphers». In: *Journal of Cryptology* 1 (1989), pp. 159–176. DOI: 10.1007/BF02252874.

[27] P. Hawkes e G. G. Rose. «Guess-and-Determine Attacks on SNOW». In: *Selected Areas in Cryptography*. Lecture Notes in Computer Science. 2002, pp. 37–46. DOI: 10.1007/3-540-36492-7_4.

[28] N. T. Courtois e W. Meier. «Algebraic Attacks on Stream Ciphers with Linear Feedback». In: *Advances in Cryptology - EUROCRYPT 2003*. Lecture Notes in Computer Science. 2003, pp. 345–359. DOI: 10.1007/3-540-39200-9_21.

[29] J. L. Massey. «Shift-register synthesis and BCH decoding». In: *IEEE Transactions on Information Theory* 15.1 (1969), pp. 122–127. DOI: 10.1109/TIT.1969.1054260.