Agenzia per la
Cybersicurezza Nazionale

LINEE GUIDA FUNZIONI CRITTOGRAFICHE

Cifratura Autenticata

MAGGIO 2026

Versione	Data di pubblicazione	Note
1.0	**26/02/2025**	**Prima pubblicazione**
1.1	**12/05/2026**	**Aggiunta sezione “Scopo del documento”, modifiche minori**

Sommario

pag.

Indice delle figure

pag.

Indice delle tabelle

Scopo del documento

Questo documento costituisce parte della serie "Linee Guida Funzioni Crittografiche" e fornisce le raccomandazioni di ACN in merito agli algoritmi per la **cifratura autenticata**, da adottare in tutti i contesti in cui si necessita di garantire la confidenzialità, integrità e autenticazione dei dati inviati. 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

$\{0,1\}$	Campo binario dei valori assumibili da un singolo bit	$\boxplus$	Somma tra interi modulo 2^{32}
$\{0,1\}^n$	Spazio vettoriale delle stringhe binarie di lunghezza n	$\ll k$	Rotazione a sinistra di k posizioni con reinserimento
$\{0,1\}^*$	Insieme di stringhe binarie di lunghezza arbitraria	$\|$	Concatenazione di stringhe
$\oplus$	Operazione XOR, ovvero la somma bit a bit tra stringhe binarie	$\|X\|$	Lunghezza in byte della stringa X
$X\|_t$	Stringa ottenuta troncando a destra la stringa X a t bit, scartando i restanti bit		

1 Introduzione

La prima funzione assunta storicamente dalla crittografia è assicurare la confidenzialità dei messaggi scambiati tra due o più interlocutori. Come descritto nel documento dedicato ai cifrari a blocchi [2] e in quello sui cifrari a flusso [3], questi due meccanismi sono le principali soluzioni che assolvono questo compito. Tuttavia, nella maggior parte delle situazioni, oltre a necessitare della confidenzialità dei dati, si richiede anche la loro autenticazione, che consiste nel fornire la possibilità di verificare l'identità del mittente, e la loro integrità, cioè la protezione dall'alterazione del messaggio trasmesso. Si parla in questo caso di cifratura autenticata. In alcuni casi, inoltre, si aggiunge la possibilità di utilizzare dei dati in chiaro aggiuntivi che vengono coinvolti nell'esecuzione dell'algoritmo, e si parla di cifratura autenticata con dati associati o AEAD (Authenticated Encryption with Associated Data).

Le richieste della cifratura autenticata possono essere soddisfatte tramite diversi metodi. Una prima possibilità è quella di sfruttare i codici di autenticazione di messaggi (MAC), già trattati nel documento dedicato [4], i quali permettono di assicurare integrità e autenticazione. Combinare cifrari e MAC permette dunque di ottenere sia la confidenzialità che l'autenticazione e quindi una cifratura autenticata. Questo processo richiede tuttavia delle accortezze particolari per evitare degli attacchi dovuti al modo in cui queste due componenti interagiscono fra loro. Una valida alternativa consiste nell'utilizzare delle specifiche modalità di funzionamento per i cifrari a blocchi che permettono di generare, in aggiunta al testo cifrato, anche un tag di autenticazione. Infine, esistono degli schemi crittografici di AEAD costruiti appositamente per questo scopo.

Il documento presenta la seguente struttura: nel capitolo 2 si introducono le definizioni principali e i diversi approcci per la cifratura autenticata. In seguito, nel capitolo 3, vengono descritti gli algoritmi dedicati alla cifratura autenticata con dati associati, mentre nel capitolo 4 sono raccolte le modalità di funzionamento dei cifrari a blocchi che risultano garantire anche l'autenticazione e l'integrità. Infine, nel capitolo 5, sono raccolte le raccomandazioni e le conclusioni.

2 Cifratura autenticata

L'espressione **cifratura autenticata** si riferisce a un processo che permette di ottenere sia la confidenzialità che l'autenticazione e l'integrità dei dati [5]. Questo risultato può essere ottenuto combinando un metodo di cifratura come i cifrari a blocchi o a flusso con uno che assicuri autenticazione e integrità come i MAC. Questi metodi possono interagire tra di loro secondo diverse tecniche, presentate in questo capitolo.

Nelle seguenti sezioni, verrà utilizzata la seguente notazione:

- M rappresenta il messaggio in chiaro, solitamente di lunghezza arbitraria;
- K_{Enc} indica la chiave segreta utilizzata per la cifratura;
- $E : \{0,1\}^* \longrightarrow \{0,1\}^*$ è la funzione di cifratura che usa la chiave segreta K_{Enc};
- C rappresenta il testo cifrato, solitamente della stessa lunghezza di M;
- K_{MAC} indica la chiave utilizzata per generare il tag;
- n indica la lunghezza in bit del tag;
- $MAC : \{0,1\}^* \longrightarrow \{0,1\}^n$ è la funzione utilizzata per ottenere il tag utilizzando la chiave K_{MAC};
- T rappresenta il tag da utilizzare per l'autenticazione.

2.1 Encrypt-then-MAC

Questo paradigma prevede di cifrare il messaggio in chiaro e poi di ottenere il tag a partire dal testo cifrato.

Più formalmente, viene prima applicata la funzione di cifratura per ottenere $C = E(M)$. In seguito, a partire dal testo cifrato, si ottiene il tag $T = MAC(C)$. La coppia (C, T) viene inviata al destinatario, il quale verifica prima la correttezza del tag T sul cifrato C e, solo in caso di riscontro positivo, decifra quest'ultimo per ottenere il messaggio di partenza. Questo approccio è il più efficiente dal punto di vista teorico, in quanto il destinatario del messaggio può verificare l'autenticità del messaggio ricevuto prima di effettuare qualsiasi operazione di decifratura per cui, se l'autenticazione non è valida, non è necessario avviare la fase di decifratura.

2.2 Encrypt-and-MAC

In questo metodo la cifratura e l'applicazione della funzione per ottenere il tag avvengono parallelamente. Il messaggio viene cifrato per ottenere $C = E(M)$ e, sempre a partire da M, si genera il tag $T = MAC(M)$. La coppia (C, T) viene inviata al destinatario, il quale decifra C per ottenere M e poi verifica che il tag T sia corretto.

2.3 MAC-then-Encrypt

In questa tecnica, innanzitutto, si ottiene il tag a partire dal messaggio in chiaro e poi si applica la cifratura alla concatenazione del messaggio e del tag ottenuto.

Quindi il mittente ottiene $T = MAC(M)$ e poi calcola il testo cifrato $C = E(M \,\|\, T)$. Quest'ultimo viene trasmesso al destinatario il quale lo può decifrare per ottenere M e T e in seguito può verificare che T sia un tag valido per il messaggio M.

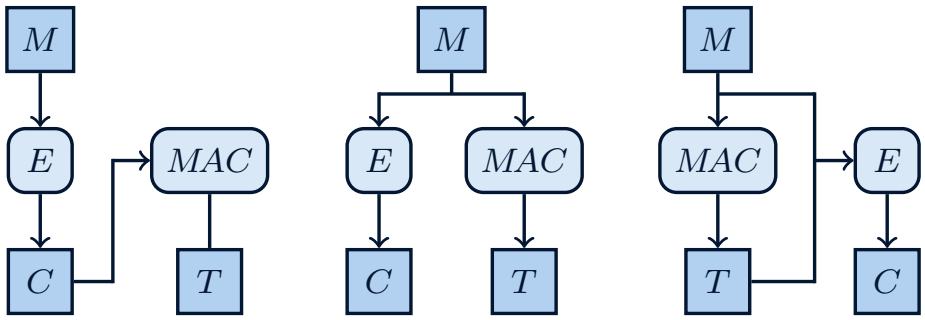

Figura 1 - Da sinistra a destra, Encrypt-then-MAC, Encrypt-and-MAC e MAC-then-Encrypt

Tra le tre metodologie, rappresentate in Figura 1, **Encrypt-then-MAC** è la più diffusa e utilizzata, tanto da essere stata sviluppata un'estensione dedicata [6] nel protocollo TLS, per il quale si rimanda al documento dedicato [7]. Da un punto di vista di efficienza, presenta il vantaggio che il ricevente non deve eseguire la decifratura nel caso in cui la verifica della correttezza del tag di autenticazione fallisca. Inoltre, da un punto di vista di sicurezza, un risultato dovuto a Bellare e Namprempre [8] dimostra che questo metodo è sicuro se i sistemi sottostanti utilizzati lo sono. Al contrario, gli altri due approcci potrebbero risultare insicuri in alcuni contesti. In particolare, Encrypt-and-MAC utilizzato in protocolli come SSH causa vulnerabilità riguardo la **contraffazione** del tag [9], mentre MAC-then-Encrypt rende insicuro il protocollo SSL/TLS contro **padding oracle attacks** [10].

Per le motivazioni descritte, nel caso in cui si voglia effettuare una cifratura autenticata utilizzando un algoritmo di cifratura simmetrico e un MAC, è sempre **raccomandato** l'utilizzo del paradigma **Encrypt-then-MAC**.

3 Cifratura autenticata con dati associati

In molti contesti di cifratura autenticata è necessario poter autenticare anche dei dati aggiuntivi che non necessitano di cifratura e vengono trasmessi **in chiaro**. Questi dati di solito contengono delle informazioni aggiuntive riguardanti il messaggio cifrato, come stringhe di testo iniziali per i pacchetti di rete o indicazioni di padding del messaggio.
La soluzione che si adotta in questi casi è quella della **cifratura autenticata con dati associati** o **AEAD** (Authenticated Encryption with Associated Data), introdotta da Phillip Rogaway nel 2002 [11].
Questo tipo di algoritmo è diventato sempre più utilizzato. Ad esempio, nella versione 1.3 del protocollo TLS, gli algoritmi AEAD sono l'unica possibilità per le suite di cifratura, al fine di garantire ottime proprietà di sicurezza nello scambio dei messaggi.
Le sezioni seguenti sono dedicate alla descrizione dettagliata di due dei sistemi per AEAD più diffusi.

3.1 Ascon

Ascon è una suite di algoritmi di crittografia simmetrica, di cui fanno parte una funzione di hash e un algoritmo di cifratura autenticata con dati associati (AEAD).
L'algoritmo AEAD di Ascon è risultato vincitore della competizione NIST dedicata agli algoritmi **lightweight** [12]. Questa tipologia di algoritmi crittografici utilizza operazioni con bassi requisiti così da poter essere implementabili anche su dispositivi con risorse limitate. Si tratta di un contesto di applicazione in rapida espansione, in particolare per via dei moderni sviluppi nel campo dei dispositivi IoT, nei quali spesso non è possibile utilizzare processori che contengono implementazioni hardware di algoritmi di cifratura standard come AES.
La proposta di Ascon come algoritmo di cifratura autenticata prevedeva inizialmente due versioni chiamate "128" e "128a", ma solamente la seconda è stata standardizzata dal NIST come algoritmo lightweight con il nome di **Ascon-AEAD128** [13].

3.1.1 La permutazione di Ascon

La principale componente della suite di Ascon è una **permutazione** P, cioè una funzione invertibile, con input e output di 320 bit. La struttura di P è quella di un round di un cifrario a blocchi, e viene ripetuta per un numero di volte che dipende dalla versione dell'algoritmo.
Nello specifico, all'interno di P l'input si scompone in 5 registri $B = b_1 \| \ldots \| b_5$ di 64 bit ciascuno, e si eseguono le seguenti operazioni:

- XOR ($\oplus$) tra l'ultimo byte del terzo registro e una costante che dipende dal numero del round;
- una S-box di 5 bit, il cui input prende un bit per ogni registro e si applica, quindi, 64 volte in parallelo;
- un mixing layer composto da cinque trasformazioni $\Sigma_1, \ldots, \Sigma_5$, ognuna delle quali viene applicata sui 64 bit del registro corrispondente.

Minaccia quantistica

Oltre alle versioni "128" e "128a", la suite prevede una modalità "post-quantum" chiamata Ascon-80pq, anch'essa non inclusa nello standard [13]. Trattandosi di un algoritmo di cifratura simmetrico, la minaccia quantistica consiste nell'**algoritmo di Grover** [14], che tuttavia garantisce al più un aumento quadratico della velocità degli attacchi di forza bruta. Per resistere a tale attacco, Ascon-80pq prevede semplicemente di aumentare la lunghezza della chiave a 160 bit per garantire 80 bit di sicurezza.

3.1.2 Parametri e modalità

Gli input della funzione di Ascon-AEAD128 sono:

- K, una chiave segreta di lunghezza $k = 128$ bit;
- N, un nonce di lunghezza 128 bit;
- A, i dati associati di lunghezza arbitraria;
- M, il testo in chiaro di lunghezza arbitraria.

Il risultato dell'algoritmo di cifratura, invece, produce:

- C, il testo cifrato di lunghezza uguale a quella di M;
- T, un tag di lunghezza $t = 128$ bit.

Gli algoritmi di cifratura e decifratura si basano sulla costruzione a **spugna** [15], descritta nel dettaglio nel documento dedicato alle funzioni di hash [16], e sulla costruzione **duplex** [17]. Dopo una fase di inizializzazione dello stato, la fase di **assorbimento** viene applicata sui dati associati mentre il messaggio in chiaro viene preso in input solo nella fase di **spremitura**. Per questo motivo, gli algoritmi di cifratura e decifratura sono piuttosto simili: il ricevente può decifrare procedendo in modo analogo alla cifratura fino alla fase di spremitura.

Durante le varie fasi, lo stato viene aggiornato applicando ripetutamente la permutazione P. In Ascon-AEAD128, questa viene utilizzata per $a = 12$ round in fase di inizializzazione e finalizzazione, dove viene quindi indicata con P^a, e per $b = 8$ round nelle fasi di assorbimento e spremitura, dove si indica con P^b.

Come descritto nel documento dedicato alle funzioni di hash [16], lo stato interno di una costruzione a spugna viene suddiviso in due parti di r e c bit, chiamati rispettivamente **bit-rate** e **capacità**. In Ascon-AEAD128, $r = 128$ e $c = 192$. Nonostante per le funzioni di hash la capacità sia direttamente proporzionale alla resistenza agli attacchi alle collisioni e alla preimmagine, in Ascon questo parametro non condiziona la sicurezza dal punto di vista di confidenzialità e autenticazione dei dati, che risulta essere di 128 bit (pari al minimo tra le lunghezze di chiave e tag).

3.1.3 Inizializzazione

Per prima cosa lo stato del sistema, di 320 bit, viene inizializzato come la concatenazione di:

- un vettore di inizializzazione di 64 bit dato da $IV = 1 \,\|\, 0^8 \,\|\, a \,\|\, b \,\|\, t \,\|\, r/8 \,\|\, 0^{16}$;
- la chiave segreta K;
- il nonce N.

A questo punto, si applica la permutazione P per $a = 12$ volte e viene calcolato lo XOR tra gli ultimi 128 bit dello stato e la chiave K. L'inizializzazione è rappresentata a sinistra in Figura 2, dove l'ultima operazione viene rappresentata utilizzando il blocco $0^{64} \| K$ per indicare una stringa di lunghezza $c = 192$ bit.

3.1.4 Assorbimento dei dati associati

I dati associati A vengono processati in blocchi di $r = 128$ bit, dopo aver applicato un padding che concatena a essi una stringa del tipo $10 \ldots 0$, al fine di rendere la lunghezza di A un multiplo di 128. A questo punto, i dati associati si spezzano in m blocchi $A_1 \,\|\, \ldots \,\|\, A_m$.

Come illustrato in Figura 2, ogni blocco A_i viene assorbito tramite lo XOR con i primi 128 bit dello stato, applicando poi la permutazione P per $b = 8$ volte. Infine, quando tutti i blocchi di A sono stati processati, la fase di assorbimento si conclude calcolando lo XOR tra $0 \ldots 01$ e lo stato.

3.1.5 Cifratura e decifratura

Come per i dati associati, anche il messaggio M viene suddiviso in blocchi di $r = 128$ bit ciascuno, per cui inizialmente si applica lo stesso padding utilizzato per A. In questo modo, la lunghezza di M risulta divisibile per 128 e si ottengono n blocchi $B_1 \,\|\, \ldots \,\|\, B_n$.

I blocchi di testo in chiaro vengono processati in sequenza: viene calcolato lo XOR tra B_i e i primi $r = 128$ bit dello stato, il cui risultato viene estratto come blocco C_i del cifrato.

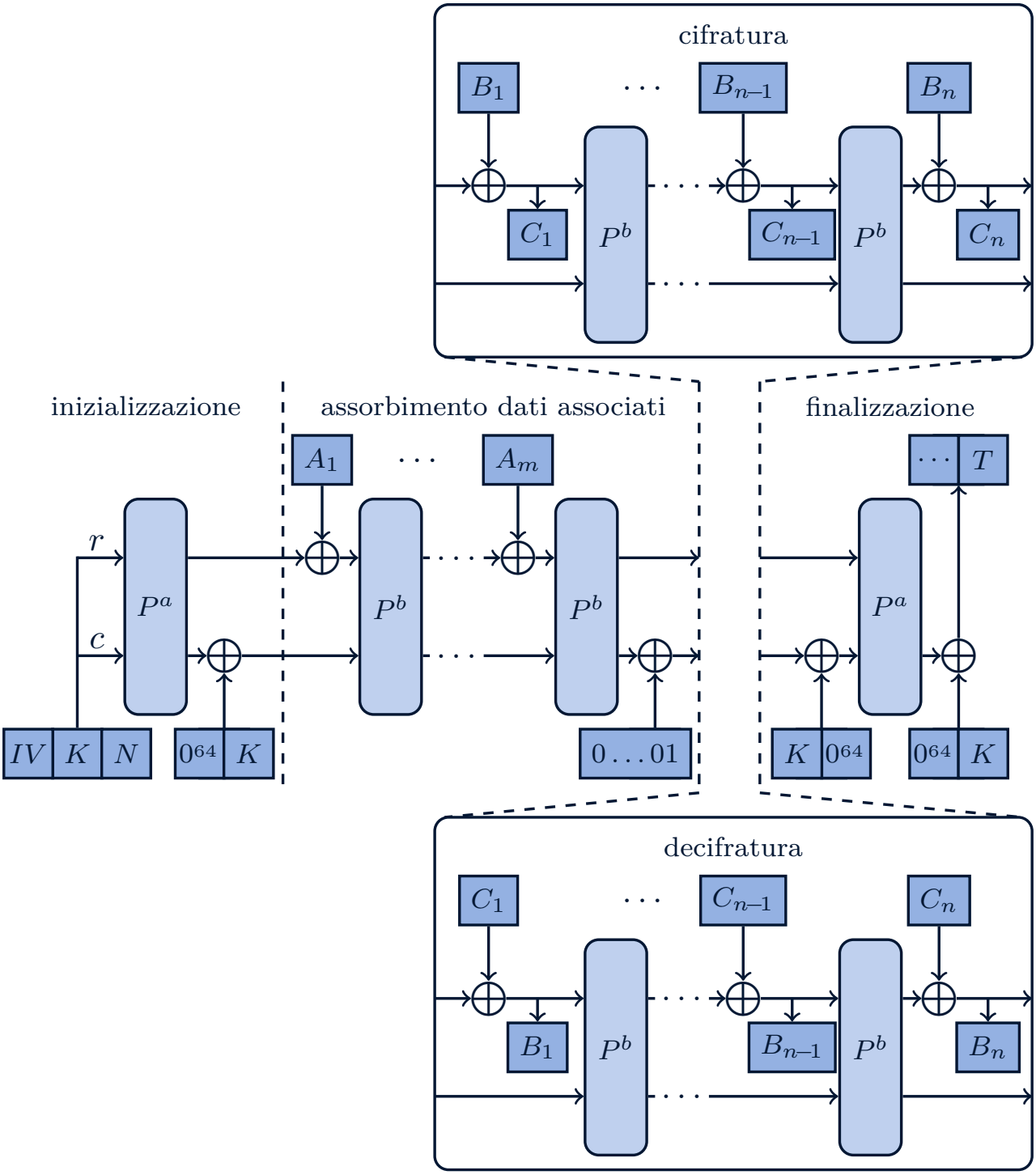

Figura 2 - Algoritmi di Ascon, sopra cifratura e sotto decifratura

Quindi viene applicata la permutazione P per $b = 8$. Una volta processati tutti i blocchi del testo in chiaro e ottenuti tutti i blocchi $C_1, \ldots, C_n$, il testo cifrato C si ottiene troncando la loro concatenazione in modo che sia di lunghezza pari a quella di M.
La fase di decifratura risulta analoga. Dopo aver applicato il padding e aver diviso il cifrato in n blocchi di 128 bit, ogni blocco C_i viene utilizzato per ottenere il relativo blocco B_i di testo in chiaro tramite lo XOR con i primi $r = 128$ bit dello stato. Quindi viene applicata la permutazione P per $b = 8$ volte. Il risultato della decifratura è la concatenazione dei blocchi ottenuti e anche in questo caso si effettua un opportuno troncamento per ottenere un messaggio M della stessa lunghezza di C.
In Figura 2, le due funzioni di cifratura e decifratura vengono rappresentate come alternative da eseguire in seguito all'assorbimento dei dati associati e prima della fase di finalizzazione.

3.1.6 Finalizzazione

La fase di finalizzazione genera il tag ed è dunque necessaria affinché la cifratura possa dirsi autenticata.
Come si può osservare nella parte destra di Figura 2, per prima cosa si calcola lo XOR tra gli ultimi $c = 192$ bit dello stato e la chiave K seguita da un numero adeguato di zeri, nello specifico $c - k = 64$. Poi viene applicata la permutazione P per $a = 12$ volte e, infine, il tag T è la stringa di 128 bit ottenuta come XOR tra la chiave e gli ultimi 128 bit dello stato così aggiornato.
Se la procedura di decifratura si è conclusa correttamente, il tag generato dal destinatario coincide con il tag ricevuto, per cui l'utente può autenticare il messaggio e i dati associati, oltre a verificarne l'integrità.

3.2 ChaCha20-Poly1305

Seppur i cifrari a blocchi abbiano raggiunto maggior fama in termini di prestazioni, la confidenzialità di un dato può essere ottenuta anche tramite cifrari a flusso. Un cifrario di tale tipologia che risulta molto utilizzato è **ChaCha**, ideato da Daniel J. Bernstein [18] nel 2008. In particolare, per poter verificare integrità e autenticazione, ChaCha può essere combinato con un MAC chiamato **Poly1305**, ideato dallo stesso autore, [19]. Sebbene questi due algoritmi siano stati pensati indipendentemente l'uno dall'altro, è possibile combinarli apportando alcune leggere modifiche per ottenere un algoritmo di cifratura autenticata di tipo AEAD con il nome di ChaCha20-Poly1305 [20].
Nonostante i due algoritmi non siano consigliati per essere utilizzati singolarmente, l'algoritmo AEAD risultante dalla loro combinazione presenta un'ottima sicurezza, tanto da essere una delle opzioni più utilizzate nei contesti in cui non è presente un'ottimizzazione hardware di AES.
Nelle seguenti sezioni si introducono le due singole componenti per poi descrivere nel dettaglio l'algoritmo AEAD che le utilizza.

3.2.1 ChaCha20

Come anticipato, ChaCha è un cifrario a flusso, che quindi genera un **keystream** che viene sommato al messaggio in chiaro tramite XOR.
Questo algoritmo utilizza solamente funzioni semplici (XOR, somma e rotazioni), in quanto è stato progettato per essere più efficiente rispetto ad AES, a parità di sicurezza. Si tratta quindi di una valida alternativa da adottare su architetture che non utilizzano chip dedicati all'utilizzo di AES come algoritmo di cifratura simmetrica.
ChaCha20 è una variante di ChaCha che utilizza uno stato interno S di 512 bit e prende in input una chiave K di 256 bit, un vettore di inizializzazione IV di 96 bit e un contatore ct di 32 bit. Il messaggio M viene diviso in blocchi da 512 bit, cioè $M = B_1 \| \ldots \| B_n$, dove l'ultimo blocco può essere anche più corto. La cifratura di B_i si ottiene calcolando lo XOR tra il blocco e l'output generato con input K, IV, $ct = i$, il quale è un keystream lungo 512 bit. In formule,

$$E_{K,IV}(M) = B_1 \oplus \mathrm{ChaCha20}(K, IV, 1) \| \ldots \| B_n \oplus \mathrm{ChaCha20}(K, IV, n).$$

Un'altra scelta comune per il valore iniziale di ct è 0, ma in generale può iniziare da un valore qualsiasi.
La coppia (K, IV) di ChaCha20 consente di cifrare messaggi di al più $2^{32} - 1$ blocchi, aggiornando di volta in volta il contatore ct, e deve essere necessariamente sostituita a ogni invocazione di $E_{K,IV}$. Risulta inoltre importante generare il valore di IV in maniera adeguata, in modo che sia **unico** per ogni chiave.
In ogni applicazione di ChaCha20, lo stato S è organizzato secondo una matrice (analogamente a quanto avviene per AES) le cui entrate sono stringhe di 32 bit.

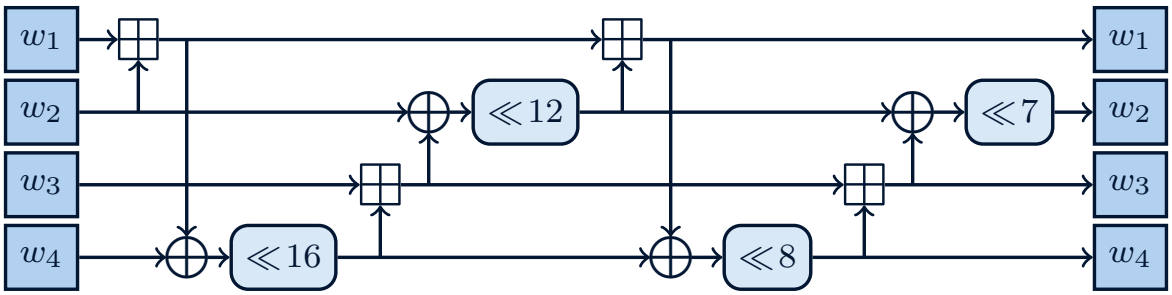

Figura 3 - Funzione QuarterRound di ChaCha20

Se $S = b_1 \| \ldots \| b_{16}$, allora l'ordine in cui vengono memorizzate le parole b_i nella matrice è il seguente:

$$\begin{pmatrix} b_1 & b_2 & b_3 & b_4 \\ b_5 & b_6 & b_7 & b_8 \\ b_9 & b_{10} & b_{11} & b_{12} \\ b_{13} & b_{14} & b_{15} & b_{16} \end{pmatrix}.$$

Lo stato viene inizializzato come $S = const \| K \| ct \| IV$, dove $const$ è una costante di 128 bit che occupa le prime quattro parole.
Lo stato S viene aggiornato in 20 round ripetuti, da cui il nome della variante del cifrario, in ognuno dei quali si applica per quattro volte l'operazione $\mathrm{QuarterRound}$, che prende quattro parole (w_1, w_2, w_3, w_4) in input e compie le seguenti operazioni:

1. $w_1 = w_1 \boxplus w_2, \quad w_4 = w_4 \oplus w_1, \quad w_4 \lll 16;$
2. $w_3 = w_3 \boxplus w_4, \quad w_2 = w_2 \oplus w_3, \quad w_2 \lll 12;$
3. $w_1 = w_1 \boxplus w_2, \quad w_4 = w_4 \oplus w_1, \quad w_4 \lll 8;$
4. $w_3 = w_3 \boxplus w_4, \quad w_2 = w_2 \oplus w_3, \quad w_2 \lll 7;$

dove $\boxplus$ indica la somma tra interi modulo 2^{32} e $\lll k$ la rotazione a sinistra di k posizioni con reinserimento. Uno schema di questa funzione si può trovare in Figura 3.
Le parole da aggiornare dipendono dal numero di round. Nei round dispari lo stato si aggiorna colonna per colonna, cioè:

- $\mathrm{QuarterRound}(b_1, b_5, b_9, b_{13})$;
- $\mathrm{QuarterRound}(b_2, b_6, b_{10}, b_{14})$;
- $\mathrm{QuarterRound}(b_3, b_7, b_{11}, b_{15})$;
- $\mathrm{QuarterRound}(b_4, b_8, b_{12}, b_{16})$;

mentre, nei round pari, si applica la stessa funzione sulle diagonali, cioè:

- $\mathrm{QuarterRound}(b_1, b_6, b_{11}, b_{16})$;
- $\mathrm{QuarterRound}(b_2, b_7, b_{12}, b_{13})$;
- $\mathrm{QuarterRound}(b_3, b_8, b_9, b_{14})$;
- $\mathrm{QuarterRound}(b_4, b_5, b_{10}, b_{15})$.

L'output che si ottiene alla fine del ventesimo round è un keystream di 512 bit che viene sommato tramite XOR con un blocco del messaggio in chiaro. Per cifrare il blocco successivo, si incrementa di 1 il contatore ct e si ripete l'operazione con stessa coppia (K, IV) per ottenere un nuovo keystream. Se l'ultimo blocco B_n ha lunghezza inferiore a 512 bit, vengono scartati gli ultimi bit del keystream ottenuto in modo che la sua lunghezza sia pari a quella di B_n.

3.2.2 Poly1305

Il codice di autenticazione di messaggi Poly1305 genera un tag di autenticazione utilizzando una chiave segreta che deve essere **effimera**, cioè diversa per ogni tag da generare. Per validare il messaggio M, Poly1305 genera un tag T lungo 128 bit a partire da una chiave K lunga 256 bit utilizzabile una sola volta. Il nome dell'algoritmo deriva dall'utilizzo di una costante che assume il valore del numero primo $p = 2^{130} - 5$. In una fase di pre-elaborazione, si inizializza casualmente la chiave, che viene divisa in due parti da 128 bit ciascuna, chiamate r e s, cioè $K = r \| s$.
La parte r viene modificata per fare in modo che il terzo, il settimo, l'undicesimo e il quindicesimo byte abbiano i primi quattro bit uguali a zero e che quindi, visti come decimali, assumano valori minori di 16. Invece, il quarto, l'ottavo e il dodicesimo byte vengono modificati in modo da avere gli ultimi due bit nulli, così che, se visti come decimali, siano divisibili per 4.

Il messaggio viene quindi diviso in blocchi da 16 byte, a ognuno dei quali si concatena un byte con valore "01" (in esadecimale). Se l'ultimo blocco risulta più corto degli altri, si applica un padding che aggiunge la quantità di zeri necessaria affinché si arrivi a una lunghezza di 17 byte.
I blocchi vengono utilizzati uno ad uno per aggiornare un accumulatore a, inizialmente nullo. Ad ogni iterazione, a assume il valore della somma tra il valore precedente e quello del blocco visto come numero decimale, il tutto moltiplicato per r modulo p. Infine, una volta finito di processare i blocchi, il tag T si ottiene considerando i primi 128 bit della somma del valore finale di a con s.

3.2.3 L'algoritmo AEAD

Nel protocollo di cifratura autenticata in cui vengono uniti i due algoritmi descritti in precedenza, viene utilizzata la stessa chiave K da 256 bit sia per la cifratura che per la generazione del MAC. Il vettore di inizializzazione IV, lungo 96 bit, deve essere diverso per ogni esecuzione con la stessa chiave al fine di garantire la sicurezza dell'algoritmo.
L'algoritmo permette di cifrare e validare un messaggio in chiaro di lunghezza arbitraria M, eventualmente facendo uso anche dei dati associati A, anch'essi senza vincolo di lunghezza.
Come illustrato nella Figura 4, ChaCha20 viene utilizzato come funzione di derivazione di chiave per ottenere una chiave temporanea K_{MAC}, che assume il valore del blocco di keystream ottenuto con input K, IV e $ct = 0$. In seguito, si cifra il messaggio M usando ChaCha20 con input K, IV e inizializzando il counter ct con il valore 1, ottenendo così il testo cifrato C.
L'input per Poly1305 è la concatenazione

$$D = A \,\|\, 0 \ldots 0 \,\|\, C \,\|\, 0 \ldots 0 \,\|\, |A| \,\|\, |C|,$$

dove ad A e C si applica un padding che aggiunge la quantità necessaria di 0 affinché la lunghezza in byte sia un multiplo di 16 (eventualmente, non aggiungendone nessuno), mentre $|A|$ e $|C|$ rappresentano le rispettive lunghezze (prima del padding) come stringhe di 8 byte.
Infine, si procede alla generazione del tag T, utilizzando l'algoritmo Poly1305 sulla stringa D e usando la chiave K_{MAC}. Vengono quindi inviati C e T al destinatario del messaggio.
La fase di decifratura è descritta nella Figura 5 ed è analoga alla cifratura. Per prima cosa si ricalcola il tag usando lo stesso procedimento e si confronta con quello ricevuto. Se i due valori coincidono, l'algoritmo di decifratura prosegue andando a generare tramite ChaCha20 lo stesso keystream ottenuto in fase di cifratura. Lo XOR tra questa stringa e C permette quindi di ricostruire il messaggio in chiaro M.

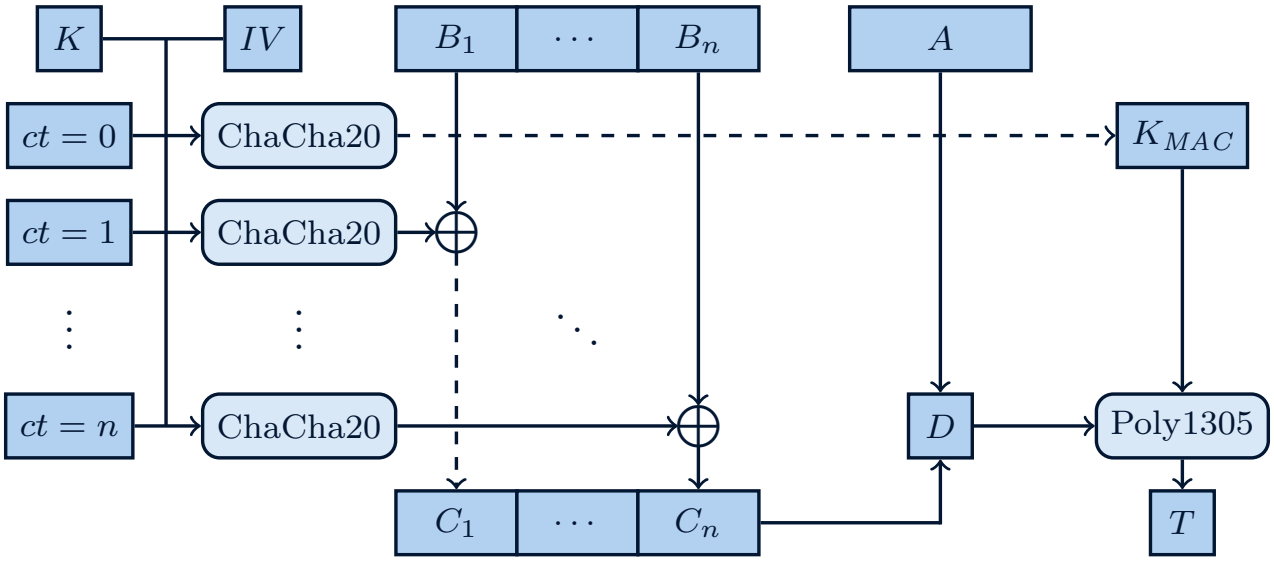

Figura 4 - Cifratura autenticata con ChaCha20-Poly1305

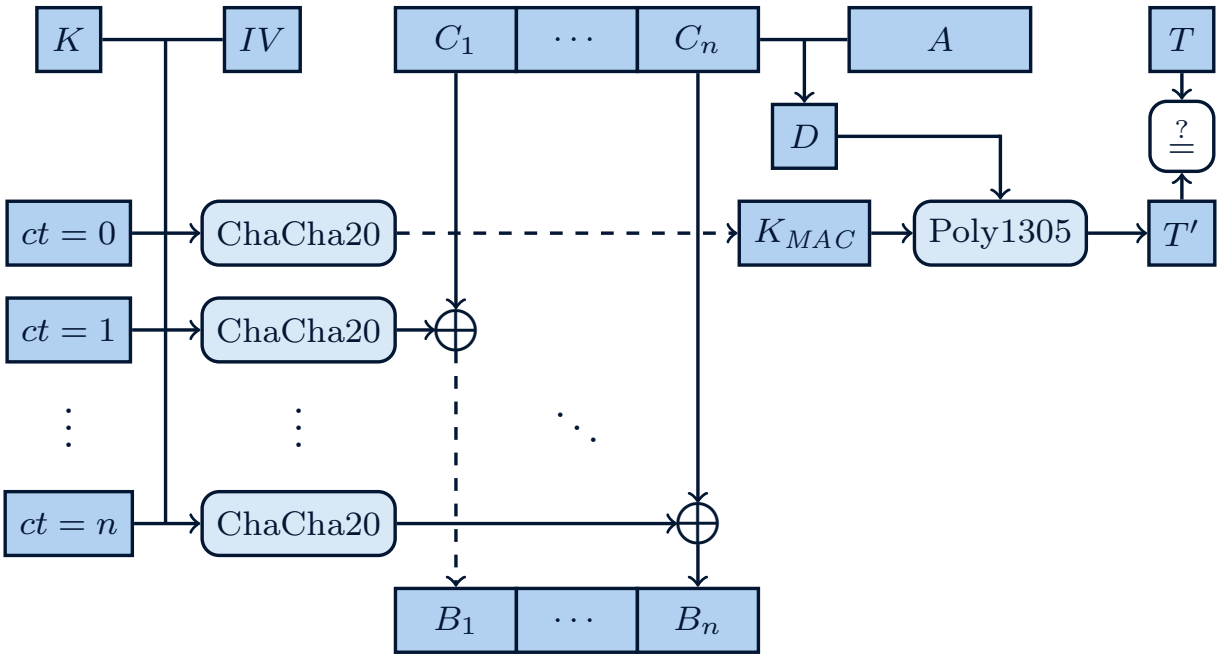

Figura 5 - Decifratura e controllo del tag con ChaCha20-Poly1305

4 Modalità di funzionamento per la cifratura autenticata

Oltre agli algoritmi indicati in precedenza, è possibile ottenere un algoritmo di AEAD applicando direttamente un cifrario a blocchi con particolari modalità di funzionamento costruite appositamente per lo scopo. Queste modalità sono tra le più utilizzate e permettono l'applicazione diretta dell'algoritmo senza dover scegliere uno dei paradigmi per la cifratura autenticata. Attualmente, queste risultano le scelte migliori per l'aggiornamento dei sistemi informatici a metodi crittografici più sicuri, in quanto già largamente implementate dalla maggior parte delle librerie dedicate.

4.1 Counter con CBC-MAC (CCM)

La modalità **CCM** [21, 22] permette in contemporanea la cifratura di blocchi del testo in chiaro M e la generazione di un tag T, sfruttando dei dati associati A, e quindi rientra nella categoria degli algoritmi AEAD. In particolare, l'autenticazione avviene sia sui dati associati sia sui blocchi in chiaro utilizzando un CBC-MAC (Cipher Block Chaining Message Authentication Code), cioè un tag ottenuto mediante la modalità CBC, mentre la cifratura sfrutta la modalità CTR partendo da un counter generato da un IV. Entrambe le procedure prendono in input la chiave segreta K e fanno uso dello stesso cifrario a blocchi E, che utilizza K e agisce su blocchi di 128 bit (tipo AES).
L'algoritmo è caratterizzato da due parametri, nello specifico la lunghezza t del tag T tra 32, 48, 64, 80, 96, 112 o 128 bit e un valore $2 \leq l \leq 8$ che descrive il trade-off tra le lunghezze di IV e M.

Per prima cosa avviene una fase di inizializzazione, durante la quale vengono processati gli input per generare i blocchi da utilizzare per la generazione del tag, in particolare:

- si genera un primo blocco A_0 dipendente dai parametri t e l, dalle lunghezze di A e M e contenente il valore di IV;
- se presenti, si processano i dati associati A generando dei blocchi $A_1, \ldots, A_m$ contenenti la lunghezza di A e i dati in esso contenuti, seguiti da un padding di soli 0 se la lunghezza dell'ultimo blocco risulta inferiore a 128 bit;
- infine, il testo in chiaro viene diviso in blocchi da 128 bit $M = B_1 \| \ldots \| B_n$. Nel caso in cui B_n risulti lungo $r < 128$ bit, si applica un padding di bit uguali a 0.

L'algoritmo di cifratura, rappresentato in Figura 6, si articola nei seguenti passi:

1. si calcola $T = E(A_0)$ come valore iniziale;
2. se sono presenti dei dati associati, si processano $A_1, \ldots, A_m$ aggiornando il tag come
$$T = E(T \oplus A_i);$$
3. partendo dall'ultimo valore generato, si processano $B_1, \ldots, B_n$ ottenendo
$$T = E(T \oplus B_i);$$
4. si cifra il testo in chiaro tramite la modalità CTR, che prende in input un contatore ct_i dipendente da l, IV e $i = 0, \ldots, n$, e genera i blocchi di keystream
$$Z_i = E(ct_i);$$

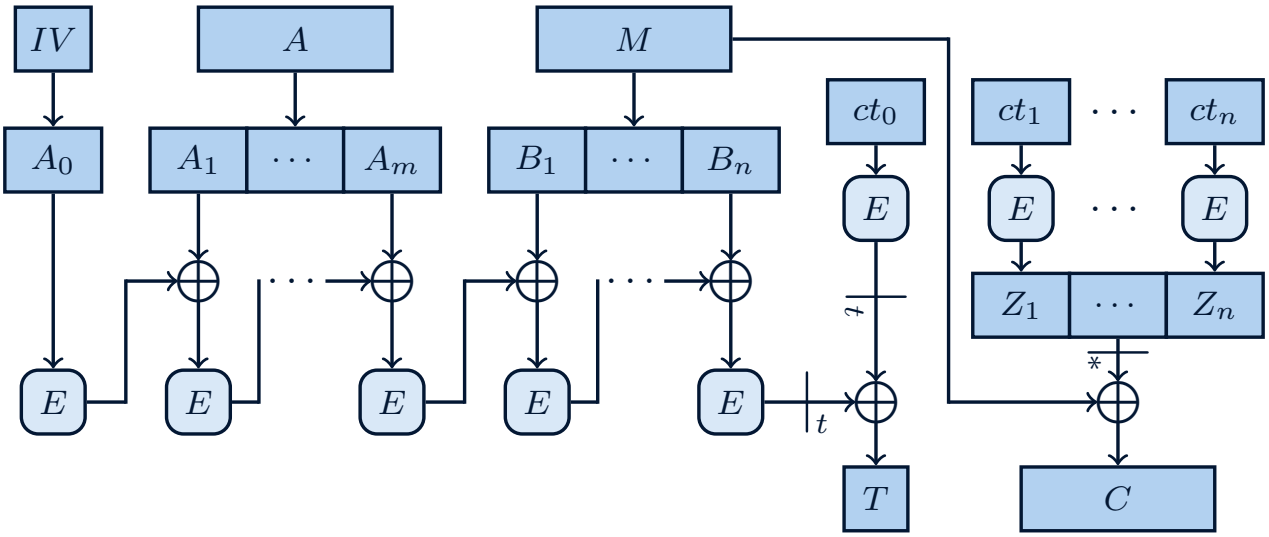

Figura 6 - Cifratura autenticata con la modalità CCM

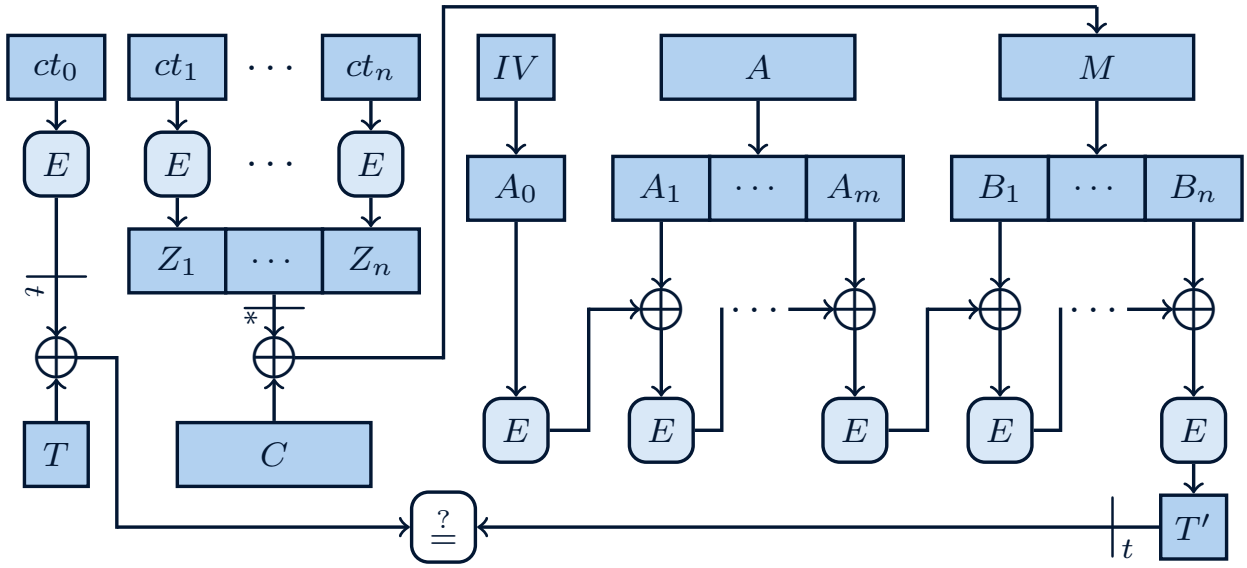

Figura 7 - Decifratura e controllo del tag con la modalità CCM

5. infine, si ottiene il tag come

$$T = T|_t \oplus Z_0|_t \,,$$

e il testo cifrato come lo XOR tra il testo in chiaro e il keystream troncato alla giusta lunghezza, cioè

$$Z = Z_1 \,\|\, \ldots \,\|\, Z_n|_r \,, \qquad C = M \oplus Z.$$

La verifica del tag e la decifratura avvengono a partire dallo stesso IV di partenza, dal tag T di lunghezza t, dai dati associati A e dal testo cifrato C. Per prima cosa, si esegue l'inizializzazione dei blocchi per CBC-MAC, che ripete i passi descritti precedentemente eseguendo però nell'ultimo la divisione del testo cifrato in blocchi da 128 bit $C = C_1 \,\|\, \ldots \,\|\, C_n$, includendo in C_n la stessa quantità di zeri aggiunti al messaggio in chiaro in fase di cifratura.
La decifratura, illustrata in Figura 7, avviene come segue:

1. si generano i blocchi del keystream $Z_i = E(ct_i)$ con ct_i dipendente da l, IV e $i = 0, \ldots, n$;
2. si ricostruisce il testo in chiaro $M = C \oplus Z$ con

$$Z = Z_1 \,\|\, \ldots \,\|\, Z_n|_r \,;$$

3. si aggiorna il tag ricevuto calcolando $T = T \oplus Z_0|_t$;
4. si ricostruisce il tag $T' = E(A_0)$;
5. se sono presenti dati associati, si aggiorna il valore di T' per ogni blocco A_i, cioè $T' = E(T' \oplus A_i)$;
6. si aggiorna il valore di T' con i blocchi del messaggio in chiaro $M = B_1 \,\|\, \ldots \,\|\, B_n$ ricavato al passo 3, cioè per ogni $i = 1, \ldots, n$ si calcola $T' = E(T' \oplus B_i)$;
7. si tronca il valore di T' a t bit e si verifica se corrisponde a T, cioè si verifica se $T'|_t \stackrel{?}{=} T$. Nel caso in cui questa relazione viene verificata il messaggio è autenticato e accettato, altrimenti viene rifiutato.

4.2 Galois Counter Mode (GCM)

La modalità **GCM** [22, 23] utilizza la modalità CTR per la cifratura e il campo finito (o di Galois, da cui il nome della modalità) di dimensione 2^{128} definito dal polinomio $x^{128} + x^7 + x^2 + x + 1$ per l'autenticazione. Anche questa modalità è un AEAD che prende in input la chiave K, un messaggio M e i dati associati A per generare un tag T valido su M e A e per cifrare M, facendo uso di un cifrario a blocchi E che utilizza K e lavora su uno stato di 128 bit (tipo AES). Il tag T possiede una lunghezza t scelta a priori tra 96, 104, 112, 120 e 128 bit.
L'autenticazione del messaggio prende in input A e il messaggio cifrato C per generare il tag usando una funzione G, la quale trasforma un blocco di 128 bit in un elemento del campo di Galois ed effettua delle moltiplicazioni in questo campo per restituire un blocco di 128 bit. In formule,

$$G(H, A, C) = X,$$

dove H e X sono blocchi di 128 bit, mentre A e C possono avere una lunghezza arbitraria, eventualmente anche nulla. All'inizio del procedimento viene scelto un IV di almeno 96 bit di lunghezza. Non è necessario che questo IV sia segreto o impredicibile, l'importante è che venga utilizzato un valore diverso per ogni messaggio che viene cifrato e autenticato con la stessa chiave. Infatti, anche un singolo riutilizzo della stessa coppia (K, IV) fornirebbe abbastanza informazioni a un attaccante per poter effettuare una contraffazione del tag.
Una volta scelto, l'IV viene pre-processato per inizializzare il contatore della modalità CTR di cifratura: se la lunghezza dell'IV è di esattamente 96 bit, allora ad esso si concatena 1 come valore di 32 bit, ottenendo così $ct_0 = IV \,\|\, 0 \ldots 01$; altrimenti, il primo contatore ct_0 assume il valore risultante dall'operazione $G(E(0), \text{“”}, IV)$, dove il primo input è la cifratura di una stringa nulla e il secondo è una stringa vuota. Il testo in chiaro viene invece pre-processato dividendolo in blocchi di 128 bit $M = B_1 \,\|\, \ldots \,\|\, B_n$, dove l'ultimo blocco B_n ha una lunghezza di $r \leq 128$ bit.
Le operazioni di cifratura e generazione del tag, descritte in Figura 8, seguono quindi questo schema:

1. per ogni blocco B_i con $1 \leq i < n$, si ottiene $ct_i = ct_{i-1} \boxplus 1$, dove $\boxplus$ indica la somma tra interi modulo 2^{32}, e il blocco cifrato risultante è

$$C_i = B_i \oplus E(ct_i);$$

2. per la cifratura dell'ultimo blocco si aggiorna il contatore come di consueto calcolando il valore $ct_n = ct_{n-1} \boxplus 1$ e si ottiene l'ultimo cifrato come

$$C_n = B_n \oplus E(ct_n)|_r \,;$$

3. si concatenano i blocchi ottenuti nel testo cifrato

$$C = C_1 \,\|\, \ldots \,\|\, C_n;$$

4. si genera il tag di lunghezza t prefissata come

$$T = \big(G(E(0), A, C) \oplus E(ct_0)\big)\big|_t \,,$$

dove $E(0)$ rappresenta la cifratura della stringa nulla.

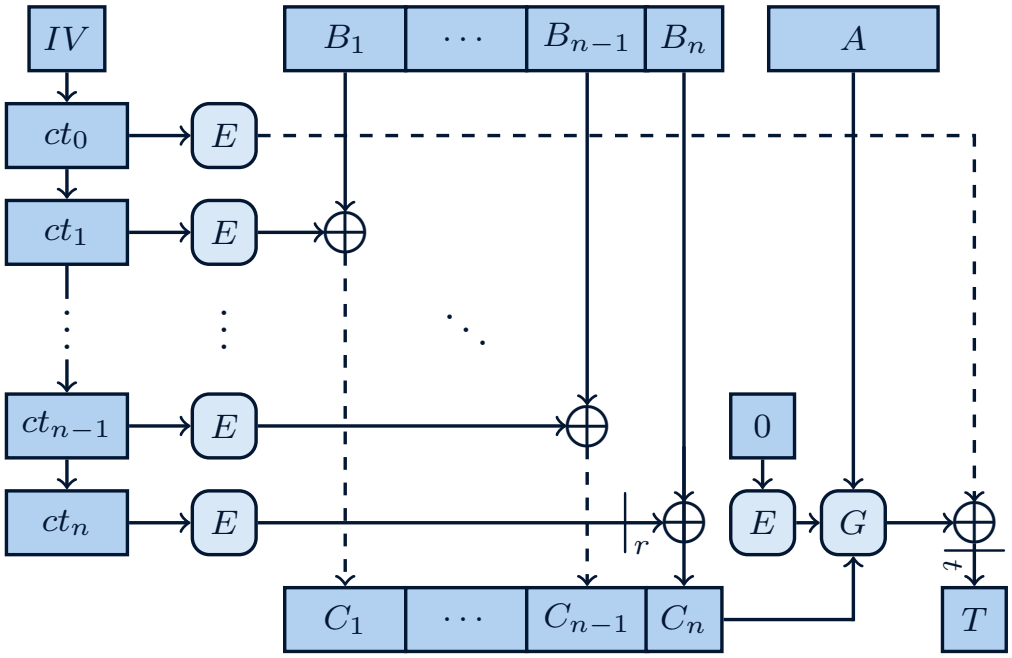

Figura 8 - Cifratura autenticata con la modalità GCM

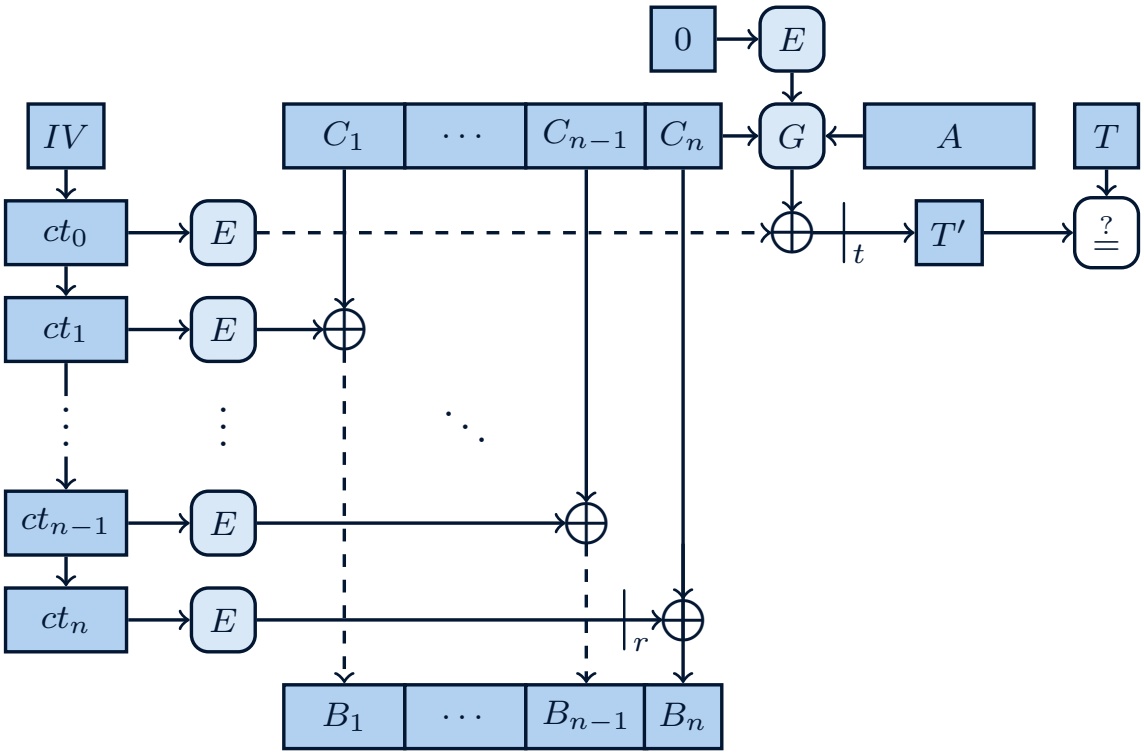

Figura 9 - Decifratura e controllo del tag con la modalità GCM

La decifratura, rappresentata in Figura 9, avviene a partire dal cifrato C, dal tag T, dai dati associati A e dal vettore di inizializzazione IV, utilizzando la chiave K per il cifrario a blocchi E.
Analogamente a prima, si calcola ct_0 a partire da IV e il cifrato viene suddiviso in blocchi da 128 bit $C = C_1 \| \ldots \| C_n$, dove l'ultimo blocco può essere lungo $r \leq 128$ bit. I passi da seguire per la decifratura sono i seguenti:

1. si calcola il tag
$$T' = \left(G(E(0), A, C) \oplus E(ct_0)\right)\big|_t ,$$
e si verifica che sia uguale a T.
Se $T' \stackrel{?}{=} T$ restituisce responso negativo, allora si riporta un errore e si interrompe la procedura;
2. si procede quindi con la decifratura: per ogni blocco C_i con $1 \leq i < n$ si calcolano
$$ct_i = ct_{i-1} \boxplus 1, \qquad B_i = C_i \oplus E(ct_i);$$
3. per l'ultimo blocco C_n, si procede a incrementare il contatore come di consueto ottenendo il valore ct_n e si ottiene il corrispondente testo in chiaro come
$$B_n = C_n \oplus E(ct_n)\big|_r ;$$
4. si ricostruisce il messaggio in chiaro $M = B_1 \| \ldots \| B_n$.

4.3 Encrypt-then-Authenticate-then-Translate (EAX)

La modalità **EAX** [22], come le precedenti, è un algoritmo AEAD che utilizza un cifrario a blocchi in modalità CTR e l'algoritmo CMAC come codice di autenticazione dei messaggi che, a sua volta, prevede l'utilizzo dello stesso cifrario a blocchi. Questa modalità [24] è stata ideata come alternativa **on-line**, cioè che può processare un flusso di dati anche un blocco alla volta, senza aver bisogno di conoscere a priori la lunghezza esatta dell'input, come ad esempio accade in CCM. Questa proprietà risulta molto utile soprattutto nei sistemi dotati di risorse limitate. Inoltre, EAX non prevede dei valori vincolati per la lunghezza dei blocchi, ma può essere adattato a qualsiasi cifrario con i blocchi di lunghezza ℓ.
Una volta scelto ℓ, si suddivide il messaggio in blocchi $M = B_1 \| \ldots \| B_n$ di tale lunghezza, con l'ultimo blocco che risulterà di lunghezza $r \leq \ell$. Dopo aver selezionato una lunghezza $t \leq \ell$ per il tag T, si genera casualmente un vettore di inizializzazione IV lungo ℓ bit, che servirà anche per fornire al destinatario l'informazione sulla lunghezza dei blocchi prevista dal cifrario a blocchi E utilizzato. La chiave segreta K viene utilizzata sia come chiave per E, sia per generare i tag tramite la funzione $CMAC$.
L'algoritmo procede come segue:

1. si calcolano i tag
$$T_1 = CMAC(0^\ell \| IV),$$
$$T_2 = CMAC(0^{\ell-1} \| 1 \| A)$$
dove T_2 autentica i dati associati A;
2. i blocchi $B_1, \ldots, B_{n-1}$ vengono cifrati con E in modalità CTR, utilizzando T_1 come primo contatore. L'aggiornamento di quest'ultimo avviene aumentando di uno il suo valore a ogni iterazione. Si ottengono così i blocchi cifrati $C_1, \ldots, C_{n-1}$;
3. l'ultimo blocco si cifra calcolando lo XOR tra B_n e la cifratura dell'ultimo contatore troncata a r bit, ottenendo l'ultimo blocco cifrato C_n;
4. si calcola il tag
$$T_3 = CMAC(0^{\ell-2} \| 10 \| C_1 \| \ldots \| C_n);$$
5. il tag completo che autentica i dati associati e il messaggio è dato da
$$T = (T_1 \oplus T_2 \oplus T_3)\big|_t ,$$
troncato al fine di ottenere una lunghezza di t bit.

Come si evince anche dalla Figura 10, il valore di T_2 coinvolge solamente i dati associati e quindi può essere precalcolato, velocizzando il processo in caso di dati statici su più messaggi da cifrare.
La decifratura avviene in maniera analoga alla cifratura, ma in ordine inverso: una volta calcolati T_1, T_2 e T_3 come sopra e verificato che il tag T corrisponda allo XOR tra questi, è possibile decifrare i blocchi $C_1, \ldots, C_n$ utilizzando nuovamente T_1 come counter per la modalità CTR. Analogamente alla cifratura, anche in questo caso verrà calcolato lo XOR tra l'ultimo blocco C_n e l'ultimo contatore troncato per ottenere la lunghezza adatta. Il processo è rappresentato in Figura 11.

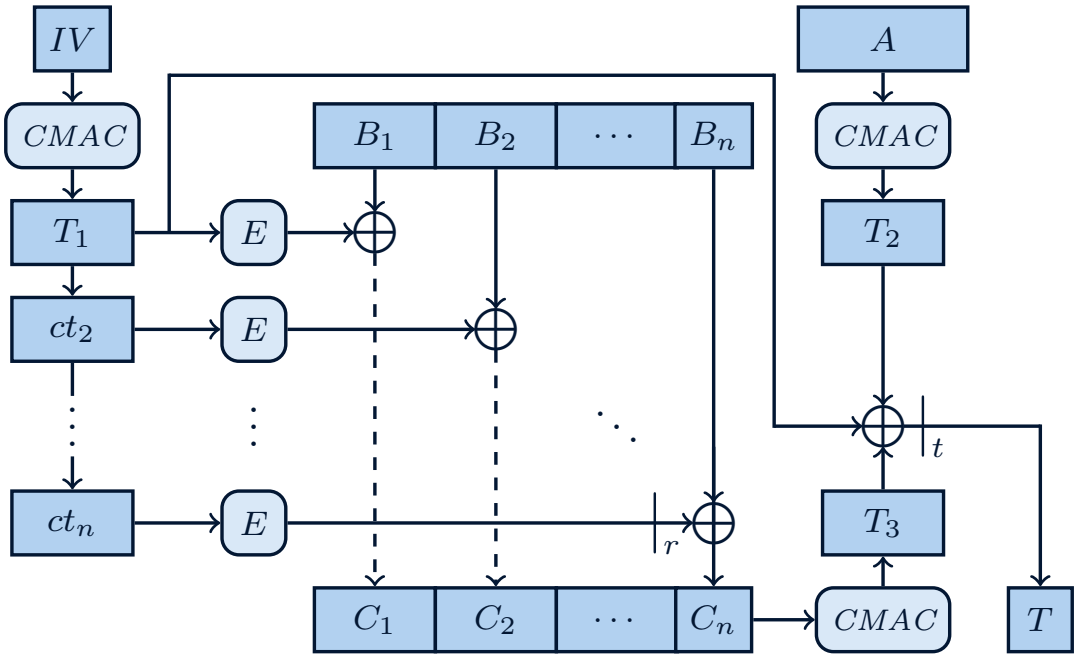

Figura 10 - Cifratura autenticata con la modalità EAX

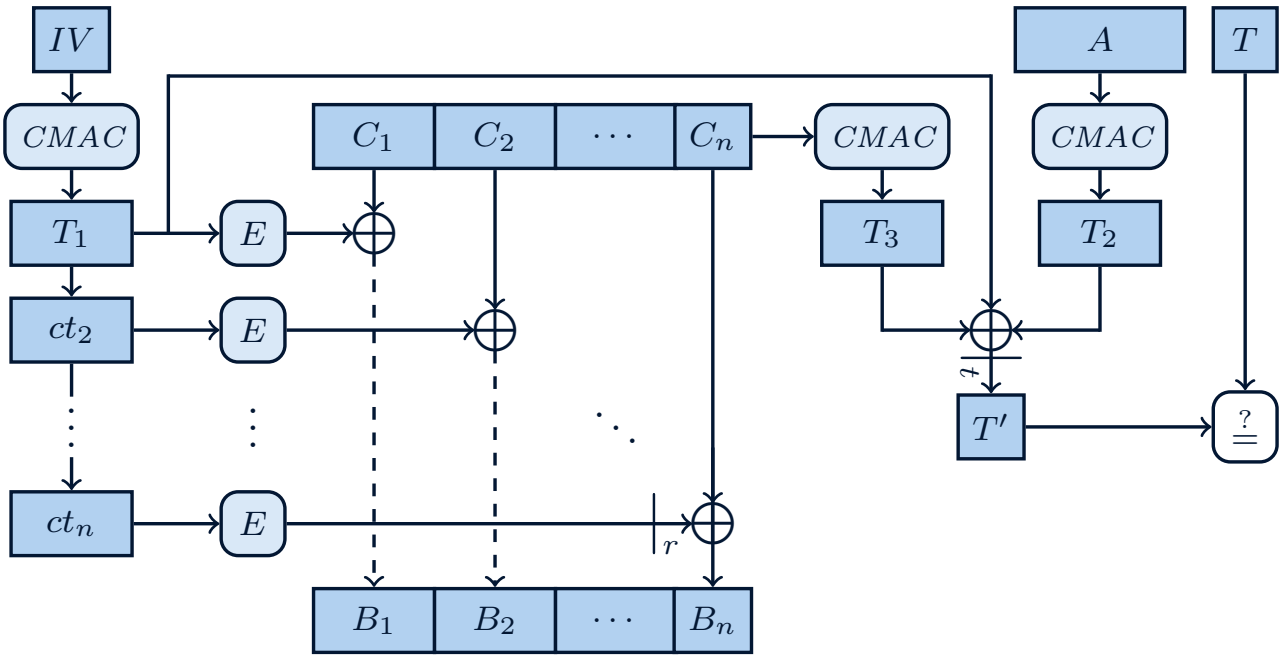

Figura 11 - Decifratura e controllo del tag con la modalità EAX

5 Conclusioni

Quando si desidera preservare la confidenzialità di un dato, si raccomanda sempre di preferire un algoritmo di **cifratura autenticata** alla sola applicazione di un algoritmo di cifratura. In questo modo, è possibile ottenere delle garanzie aggiuntive sull'integrità e l'accessibilità del dato inviato, incrementando di fatto il livello di sicurezza. Questo ragionamento è stato, per esempio, già applicato nella scelta degli algoritmi della suite crittografica di TLS 1.3, preferendo direttamente soluzioni AEAD. Per maggiori dettagli, si faccia riferimento alla linea guida dedicata [7].
In Tabella 1 vengono riportati gli algoritmi di cifratura autenticata raccomandati.
Per quanto riguarda la modalità Encrypt-then-MAC, si raccomanda di utilizzare l'algoritmo AES in una delle modalità di cifratura indicate nel documento dedicato [2], affiancandolo a un MAC tra quelli raccomandati nel documento che fornisce i dettagli a riguardo.
Ascon-AEAD128 è la raccomandazione per i contesti con ridotte capacità computazionali e di memoria che quindi richiedono l'utilizzo di un algoritmo di cifratura lightweight. Se il contesto di applicazione lo consente, si raccomanda l'utilizzo delle altre soluzioni elencate in tabella.
ChaCha20-Poly1305 viene raccomandato in particolare nei contesti in cui non è presente un'ottimizzazione hardware dell'algoritmo di cifratura AES.
Per tutti gli altri casi, si raccomanda l'utilizzo dell'algoritmo di cifratura a blocchi AES con una delle modalità di funzionamento descritte in questo documento e elencate in tabella. In ognuna di queste soluzioni, si raccomanda di generare un tag di almeno 128 bit.

A titolo di esempio per il paradigma Encrypt-then-MAC, alcune scelte per gli algoritmi da adottare possono essere AES-CBC o AES-CTR con HMAC. Se si intende generare il tag con CMAC o GMAC, considerando le specifiche dei rispettivi algoritmi, si raccomanda di utilizzare direttamente AES-CCM o AES-GCM.

Algoritmo	Note
AES-GCM	Generare un tag di almeno 128 bit
AES-CCM	
AES-EAX	
ChaCha20-Poly1305	Utilizzare in mancanza di implementazioni hardware di AES
Ascon-AEAD128	Cifrario lightweight Utilizzare in mancanza di implementazioni hardware di AES Tenere in considerazione il riquadro di warning sottostante
Encrypt-then-MAC	Fare riferimento alle Linee Guida: [2, 3] per gli algoritmi di cifratura raccomandati [4] per i sistemi di autenticazione raccomandati

Tabella 1 - Algoritmi per la cifratura autenticata e parametri raccomandati

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] ACN. *Cifrari a Flusso*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

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

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

[6] P. Gutmann. *Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*. RFC 7366. 2014. DOI: 10.17487/RFC7366.

[7] ACN. *Transport Layer Security (TLS)*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

[8] M. Bellare e C. Namprempre. «Authenticated encryption: Relations among notions and analysis of the generic composition paradigm». In: *Journal of Cryptology* 21 (2008), pp. 469–491. DOI: 10.1007/s00145-008-9026-x.

[9] M. Bellare, T. Kohno e C. Namprempre. «Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm». In: *ACM Transactions on Information and System Security* 7.2 (2004), pp. 206–241. DOI: 10.1145/996943.996945.

[10] X. Wang et al. «Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS...» In: *Advances in Cryptology - EUROCRYPT 2002*. Lecture Notes in Computer Science. 2002, pp. 534–545. DOI: 10.1007/3-540-46035-7_35.

[11] P. Rogaway. «Authenticated-encryption with associated-data». In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. 2002, pp. 98–107. DOI: 10.1145/586110.586125.

[12] NIST. *Lightweight Cryptography*. 2017. URL: https://csrc.nist.gov/projects/lightweight-cryptography.

[13] M. S. Turan et al. *Ascon-Based Lightweight Cryptography Standards for Constrained Devices: Authenticated Encryption, Hash, and Extendable Output Functions*. SP 800-232. NIST, 2025. DOI: 10.6028/NIST.SP.800-232.

[14] 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.

[15] G. Bertoni et al. *Sponge functions*. ECRYPT Hash Functions Workshop. 2007. URL: https://keccak.team/files/SpongeFunctions.pdf.

[16] ACN. *Funzioni di Hash*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

[17] G. Bertoni et al. «Duplexing the sponge: single-pass authenticated encryption and other applications». In: *Selected Areas in Cryptography*. Lecture Notes in Computer Science. 2012, pp. 320–337. DOI: 10.1007/978-3-642-28496-0_19.

[18] 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.

[19] D. J. Bernstein. «The Poly1305-AES Message-Authentication Code». In: *Fast Software Encryption (FSE)*. Lecture Notes in Computer Science. 2005, pp. 32–49. DOI: 10.1007/11502760_3.

[20] Y. Nir e A. Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 8439. 2018. DOI: 10.17487/RFC8439.

[21] M. Dworkin. *Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality*. SP 800-38C. NIST, 2007. DOI: 10.6028/NIST.SP.800-38C.

[22] ISO. *Information security – Authenticated encryption*. ISO/IEC 19772. 2020. URL: https://www.iso.org/standard/81550.html.

[23] M. Dworkin. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. SP 800-38D. NIST, 2007. DOI: 10.6028/NIST.SP.800-38D.

[24] M. Bellare, P. Rogaway e D. Wagner. «The EAX mode of operation». In: *Fast Software Encryption (FSE)*. Lecture Notes in Computer Science. 2004, pp. 389–407. DOI: 10.1007/978-3-540-25937-4_25.