Analisi Approfondita della Cifratura Autenticata in TLS: Vulnerabilità dell'approccio MAC-then-Encrypt e Attacchi Padding Oracle

Sintesi e approfondimento

1 Cifratura e Integrità: Due Obiettivi Distinti

In un protocollo di comunicazione sicura, la **cifratura** e l' **integrità** sono due obiettivi di sicurezza fondamentali ma concettualmente distinti.

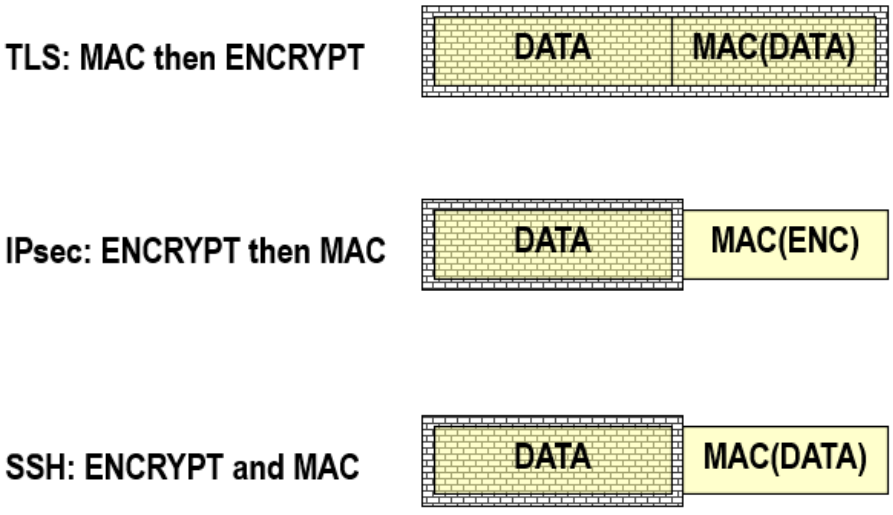
- La **cifratura** garantisce la *confidenzialità*, impedendo a un avversario di leggere il contenuto di un messaggio.
- L' **integrità**, garantita tramite un Message Authentication Code (MAC), previene la *manipolazione* (tampering) o la *falsificazione* (spoofing) dei messaggi da parte di un utente malintenzionato.

È un errore comune, e pericoloso, presumere che un messaggio cifrato sia automaticamente protetto da modifiche. Molti algoritmi di cifratura, se usati in modo scorretto, sono malleabili. Ad esempio, con un cifrario a flusso (come RC4 o un cifrario a blocchi in modalità CTR), dove il testo cifrato C è ottenuto come $C = M \oplus K$ (con K keystream), un attaccante può calcolare un nuovo testo cifrato $C' = C \oplus \Delta$. In fase di decifratura, il destinatario otterrà $M' = C' \oplus K = (C \oplus \Delta) \oplus K = (M \oplus K \oplus \Delta) \oplus K = M \oplus \Delta$. L'attaccante può così modificare il messaggio in chiaro in modo prevedibile, pur non conoscendo né il messaggio originale né la chiave.

Per questo motivo, è essenziale combinare esplicitamente un meccanismo di cifratura con uno di autenticazione. Le uniche eccezioni sono i cosiddetti cifrari **AEAD (Authenticated Encryption with Associated Data)**, come AES-GCM, che sono progettati per fornire simultaneamente sia confidenzialità che integrità in un unico passaggio sicuro. Non a caso, TLS 1.3 ha reso obbligatorio l'uso esclusivo di suite crittografiche AEAD.

2 Come Combinare Cifratura e Autenticazione

Quando si progettano protocolli sicuri, non è sufficiente scegliere un buon algoritmo di cifratura e un buon algoritmo di MAC; è cruciale il *modo* in cui questi due primitivi vengono combinati. Esistono tre approcci principali per combinare un meccanismo di cifratura (ENC) e un MAC, ma solo uno è considerato veramente sicuro.



Di seguito l'analisi dettagliata delle tre costruzioni:

1. **Encrypt-and-MAC (Stile SSH):**
   In questo approccio, si calcola il MAC sul testo in chiaro (M) e lo si accoda al testo cifrato separatamente.
   $$\text{Output} = (ENC(M),\ MAC(M))$$
   Analisi di sicurezza:
   - Questo approccio è classificato come il **più insicuro** (o "Most insecure" nelle slide).
   - **Il problema della privacy:** Un algoritmo di MAC è progettato per garantire l'integrità, non la confidenzialità. Se il MAC viene calcolato sul testo in chiaro e inviato senza cifratura, il tag del MAC stesso potrebbe rivelare informazioni sul messaggio M.
   - La proprietà di "unforgeability" (non falsificabilità) del MAC non garantisce che i bit del tag appaiano casuali o non correlati al messaggio. Pertanto, un attaccante potrebbe dedurre pattern del plaintext semplicemente osservando il MAC.
2. **MAC-then-Encrypt (Stile TLS - versioni storiche):**
   Si calcola prima il MAC sul testo in chiaro, lo si accoda al messaggio, e infine si cifra l'intera sequenza (messaggio + MAC).
   $$\text{Output} = ENC(M \,||\, MAC(M))$$
   Analisi di sicurezza:
   - Sebbene sembri l'approccio più logico (proteggo tutto con la cifratura), è **fortemente sconsigliato** a causa di vulnerabilità pratiche.
   - **Problema dell'ordine delle operazioni:** Chi riceve il messaggio deve prima decifrarlo per poter verificare il MAC. Se il padding della cifratura è errato, il sistema potrebbe restituire un errore diverso rispetto a quando il MAC è errato.
   - **Attacchi specifici:** Questa struttura apre la porta agli attacchi *Chosen Ciphertext* e ai *Padding Oracle Attacks* (come l'attacco "Lucky 13" o "Vaudenay's attack").
   - *Riferimenti storici:* Krawczyk (Crypto 2001) ha dimostrato che questa composizione non è sicura in generale. Degrabriele e Paterson (2010) hanno trovato attacchi pratici su configurazioni IPsec che usavano impropriamente questo ordine.
3. **Encrypt-then-MAC (Stile IPsec):**
   Si cifra prima il messaggio M e successivamente si calcola il MAC esclusivamente sul testo cifrato (C).
   $$C = ENC(M); \quad \text{Output} = (C,\ MAC(C))$$
   Analisi di sicurezza:
   - Questo è l'approccio **provably secure** (provabilmente sicuro).
   - **Vantaggio principale:** Il ricevente verifica l'integrità del crittogramma *prima* di tentare qualsiasi decifratura. Se il MAC è invalido, il pacchetto viene scartato immediatamente. Questo impedisce completamente gli attacchi di tipo Padding Oracle, poiché non avviene alcuna operazione crittografica (decifratura) su dati manipolati.
   - **Requisiti ridotti:** Affinché questo schema sia sicuro, è sufficiente che la cifratura sia semanticamente sicura (CPA - Chosen Plaintext Attack secure). Non è nemmeno richiesto che la cifratura sia robusta contro attacchi attivi da sola, perché il MAC funge da scudo esterno.
   - **Conclusione:** Se si deve scegliere manualmente come combinare le primitive, bisogna **SEMPRE** usare Encrypt-then-MAC.

L'alternativa moderna: AEAD

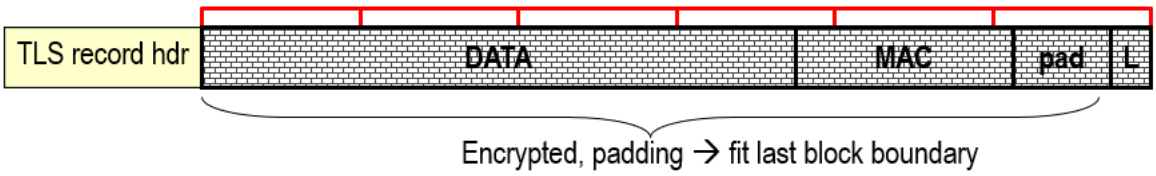
Sebbene *Encrypt-then-MAC* sia la scelta corretta tra le tre opzioni "manuali", la crittografia moderna suggerisce di evitare del tutto il problema utilizzando primitive **AEAD (Authenticated Encryption with Associated Data)**.

Gli algoritmi AEAD (come AES-GCM o ChaCha20-Poly1305) gestiscono internamente la combinazione di cifratura e autenticazione nel modo più sicuro possibile, eliminando il rischio di errori implementativi da parte dello sviluppatore.

3 L'Attacco Padding Oracle su TLS in modalità CBC

L'attacco, scoperto da Serge Vaudenay nel 2002, è un classico esempio di attacco a canale laterale (side-channel attack) che sfrutta la modalità operativa CBC combinata con l'approccio *MAC-then-Encrypt* in TLS 1.0.

L'attacco è di tipo **Chosen Ciphertext Attack (CCA)**: l'attaccante può inviare testi cifrati arbitrari al server e osservare la sua reazione per dedurre informazioni sul testo in chiaro.



3.1 Il Padding in TLS

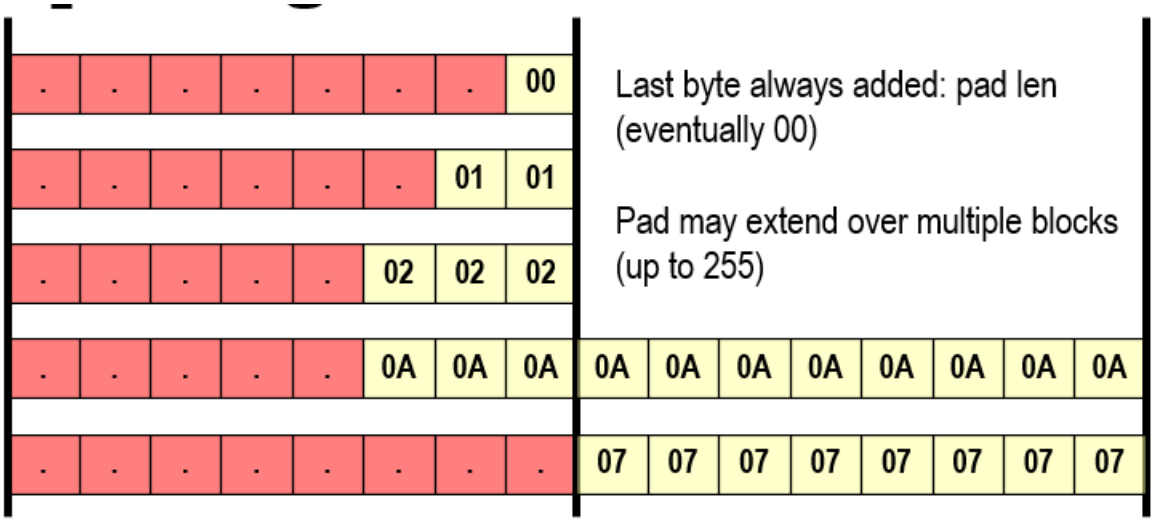
I cifrari a blocchi (come AES o DES) lavorano su blocchi di dimensione fissa (es. 8 o 16 byte). Poiché i messaggi raramente sono multipli esatti della dimensione del blocco, è necessario aggiungere riempimento (padding). Il padding in TLS segue uno schema rigoroso:

- L'ultimo byte del blocco indica la **lunghezza del padding** (L).
- I precedenti L byte devono avere tutti valore uguale a L.

Esempi (supponendo un blocco di 8 byte):

- Padding di 1 byte: `XX XX XX XX XX XX XX 00`
- Padding di 2 byte: `XX XX XX XX XX XX 01 01`
- Padding di 3 byte: `XX XX XX XX XX 02 02 02`

Questa struttura rigida è fondamentale per l'attacco: se decifrando si ottiene un pattern diverso (es. `... 02 01 02`), il padding è considerato *invalido*.



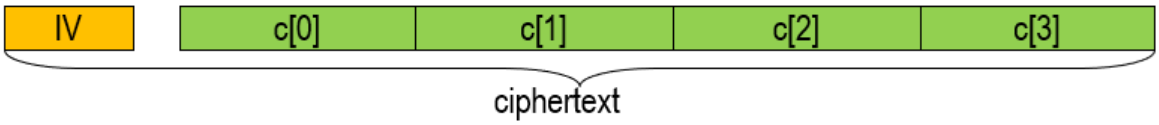
3.2 La Vulnerabilità: L'Oracolo

In TLS 1.0, il processo di decifrazione rivela troppe informazioni a causa dell'ordine delle operazioni (*MAC-then-Encrypt*):

1. Il server decifra il testo cifrato ricevuto.
2. **Check Padding:** Controlla se il padding è corretto. Se fallisce, invia un alert red`decryption_failed`.

3. **Check MAC:** Se il padding è valido, lo rimuove e verifica il MAC. Se il MAC è errato, invia un alert red`bad_record_mac`.

Poiché un attaccante può distinguere questi due errori (tramite il messaggio di errore stesso o misurando il tempo di risposta), il server agisce come un **Padding Oracle**. L'attaccante può chiedere al server: *"Se modifico il testo cifrato in questo modo, la decifrazione produce un padding valido?"*. Risposta `bad_record_mac` = SÌ (Padding OK). Risposta `decryption_failed` = NO (Padding Errato).



3.3 Meccanismo dell'Attacco

L'obiettivo è decifrare un blocco C_1 senza conoscere la chiave K. Ricordiamo la formula di decifratura CBC per ottenere il plaintext M_1:

$$M_1 = D_K(C_1) \oplus C_0$$

Dove C_0 è il blocco precedente (o l'IV). L'attaccante conosce C_0 e C_1. Non conosce $D_K(C_1)$, che chiameremo *Stato Intermedio* (IS). Quindi $M_1 = IS \oplus C_0$.

L'attaccante invia al server una coppia manipolata (C_0', C_1). Il server calcolerà:

$$M_1' = D_K(C_1) \oplus C_0' = IS \oplus C_0'$$

3.3.1 Fase 1: Decifrare l'ultimo byte

Vogliamo scoprire l'ultimo byte di M_1 (chiamiamolo P_{last}).

1. L'attaccante costruisce C_0' copiando C_0 ma modificando l'ultimo byte.
2. Vuole che, dopo la decifratura, l'ultimo byte del plaintext manipolato (M_1') sia `0x00` (che rappresenta un padding valido di lunghezza 1).
3. La formula per l'ultimo byte manipolato è:

$$C_0'[last] = C_0[last] \oplus \underbrace{g}_{guess} \oplus \texttt{0x00}$$

4. Sostituendo nella formula di decifrazione del server:

$$M_1'[last] = IS[last] \oplus C_0'[last]$$

$$M_1'[last] = (M_1[last] \oplus C_0[last]) \oplus (C_0[last] \oplus g \oplus \texttt{0x00})$$

Grazie alle proprietà dello XOR ($A \oplus A = 0$), i termini $C_0[last]$ si annullano:

$$M_1'[last] = M_1[last] \oplus g \oplus \texttt{0x00}$$

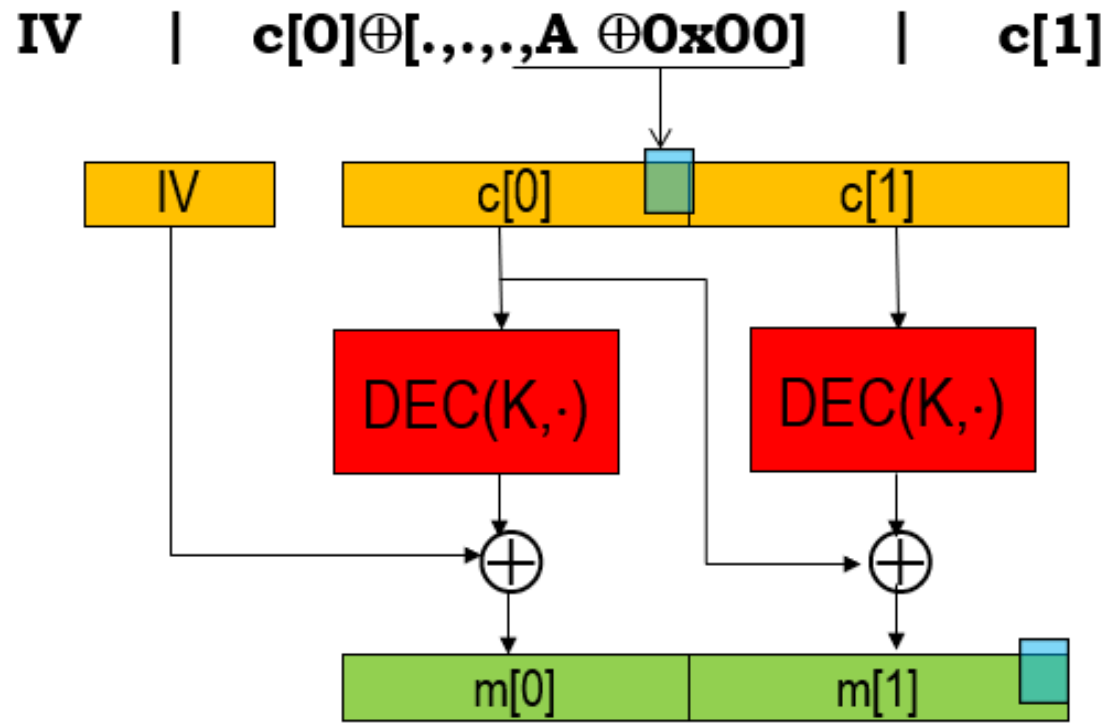
L'attaccante prova tutti i 256 possibili valori di g.

- Se $g \neq M_1[last]$, allora $M_1'[last] \neq$ `0x00`. Il padding è quasi certamente invalido. Errore: `decryption_failed`.
- Se $g == M_1[last]$, allora $M_1'[last] =$ `0x00`. Il server vede un padding valido di 1 byte. Errore: `bad_record_mac`.

Quando l'oracolo risponde "Bad MAC", l'attaccante sa che g è il byte corretto del messaggio originale!

3.3.2 Fase 2: Decifrare i byte precedenti (Iterazione)

Una volta scoperto l'ultimo byte, si passa al penultimo. Ora l'obiettivo è far credere al server che il padding sia `0x01 0x01` (2 byte di padding).



1. Dobbiamo impostare l'ultimo byte di C_0' in modo che decifri a `0x01`. Poiché conosciamo già il valore reale P_{last}, calcoliamo:

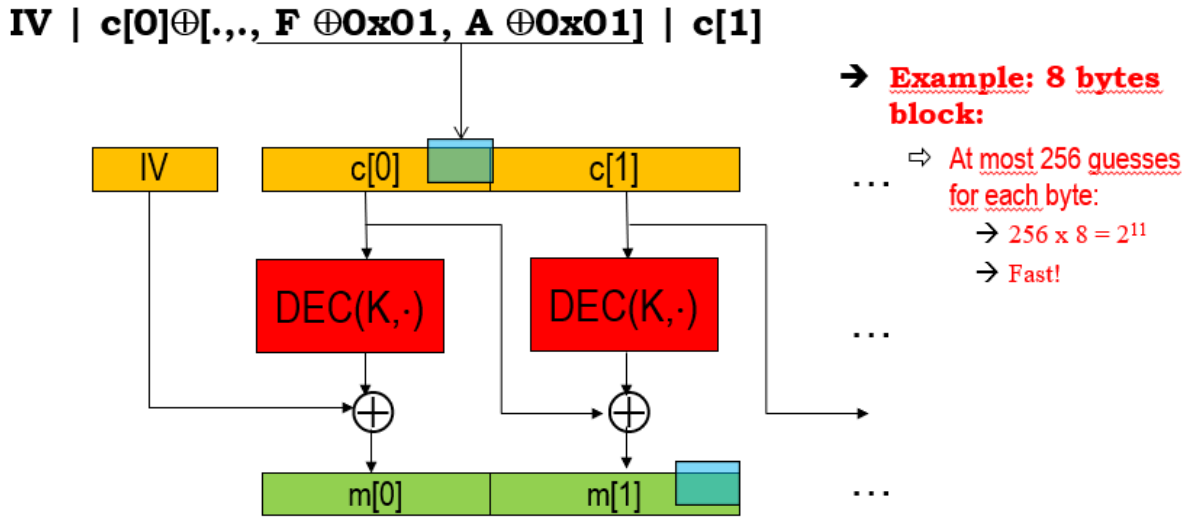
$$C_0'[last] = C_0[last] \oplus P_{last} \oplus \texttt{0x01}$$

2. Ora iteriamo sul **penultimo** byte con un nuovo tentativo g:

$$C_0'[k-1] = C_0[k-1] \oplus g \oplus \texttt{0x01}$$

3. Inviamo (C_0', C_1). Se il server risponde `bad_record_mac`, significa che il plaintext finisce con `01 01`. Quindi il nostro g è corretto.

Questo processo si ripete a ritroso per tutti i byte del blocco. **Complessità:** Invece di dover indovinare l'intero blocco (2^{128} tentativi), indoviniamo byte per byte (128×256 tentativi al massimo). È estremamente efficiente.



4 Praticabilità ed Evoluzione del Padding Oracle

Teoricamente, l'attacco di Vaudenay sembrerebbe difficile da eseguire in uno scenario reale per un motivo fondamentale: i messaggi di errore `bad_record_mac` e `decryption_failed` sono considerati **fatal alerts**.

- Quando si verifica uno di questi errori, la connessione TLS viene immediatamente interrotta.
- La connessione successiva avrà una chiave di sessione differente, rendendo inutili le informazioni parziali acquisite sul blocco precedente.

Tuttavia, nel 2003, **Canvel e Vuagnoux** hanno dimostrato che l'attacco è praticabile su protocolli applicativi come **IMAP** (protetto da TLS):

1. I client di posta inviano periodicamente (ogni pochi minuti) le credenziali di login (stringa fissa `LOGIN "user" "pass"`).
2. Un attaccante *Man-in-the-Middle* intercetta la comunicazione.
3. L'attaccante modifica il testo cifrato e lo invia al server. La connessione cade a causa dell'errore, ma non è un problema: l'attaccante aspetta semplicemente che il client ritenti il login (pochi minuti dopo) con lo stesso plaintext (sebbene cifrato con una nuova chiave, la struttura permette attacchi dizionario ottimizzati).
4. L'attacco permette di recuperare la password in poche ore.

4.1 Contromisure e Canali Laterali Temporali

Per mitigare l'attacco Padding Oracle originale (che si basava sulla differenza dei messaggi di errore), il protocollo TLS è stato aggiornato.

- **La Soluzione in TLS 1.1:** La specifica impone che l'implementazione debba restituire lo stesso alert (`bad_record_mac`) indipendentemente dal fatto che l'errore sia dovuto al padding o al MAC.
- **Il problema (Canvel 2003):** Anche se il messaggio di errore è identico, il **tempo di risposta** può variare.
  - Se il padding è errato, il server si ferma subito (processo più veloce).
  - Se il padding è corretto ma il MAC è errato, il server deve calcolare il MAC su tutto il messaggio prima di fallire (processo più lento).

  Questa differenza temporale reintroduce l'oracolo (*Timing Oracle*).

4.2 Evoluzione delle Vulnerabilità (Fixes and Follow-ups)

La storia delle vulnerabilità legate al MAC-then-Encrypt è proseguita per oltre un decennio:

- **TLS 1.2:** Tenta di risolvere il problema temporale imponendo di validare il MAC in *ogni caso*, anche se il padding è corrotto.
  - *Problema:* Se il padding è corrotto, non sappiamo quanto è lungo il messaggio reale e quanto è lungo il padding. Su quanti dati calcoliamo il MAC fittizio?
  - Se si calcola su tutto il pacchetto, si impiega più tempo rispetto a un pacchetto corretto (che è più corto dopo la rimozione del padding). Questo crea ancora differenze temporali sottili.
- **Lucky Thirteen (2013):** Kenny Paterson ha dimostrato come sfruttare queste micro-differenze temporali per recuperare il plaintext.
- **POODLE (2014):** Unisce il Padding Oracle a un attacco di downgrade a SSL 3.0.
- **Lucky Microseconds (2015) CVE-2016-2107:** Attacchi sempre più raffinati sui canali laterali temporali, colpendo anche le patch che tentavano di rendere il tempo di esecuzione costante.

4.3 Lezioni Apprese

L'analisi di quindici anni di attacchi su TLS in modalità CBC ci insegna due lezioni fondamentali:

1. **Encrypt-then-MAC avrebbe prevenuto tutto:** Se TLS avesse usato la costruzione Encrypt-then-MAC (stile IPsec), il MAC sarebbe stato verificato sul testo cifrato *prima* della decifratura. Il padding non sarebbe mai stato processato in caso di attacco, eliminando alla radice sia l'oracolo del messaggio d'errore sia quello temporale.

2. **L'implementazione è critica:** Anche se il protocollo è teoricamente corretto, implementarlo senza introdurre side-channels (come timing attacks) è estremamente difficile. La regola d'oro è: *"Don't implement crypto yourself"*.