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.

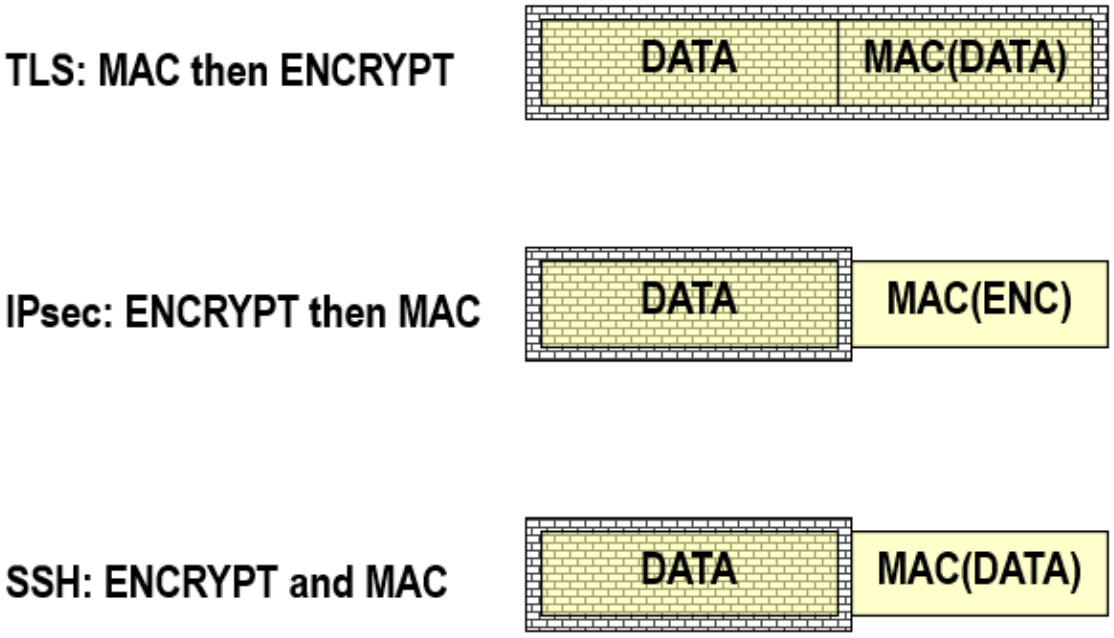
È un errore comune, e pericoloso, presumere che un messaggio cifrato sia automaticamente protetto da modifiche. Molti algoritmi di cifratura, se usati in modo scorretto, sono malleabili. Ad esempio, con un cifrario a flusso (come RC4 o un cifrario a blocchi in modalità CTR), dove il testo cifrato C è 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

Esistono tre approcci principali per combinare un meccanismo di cifratura (ENC) e un MAC:

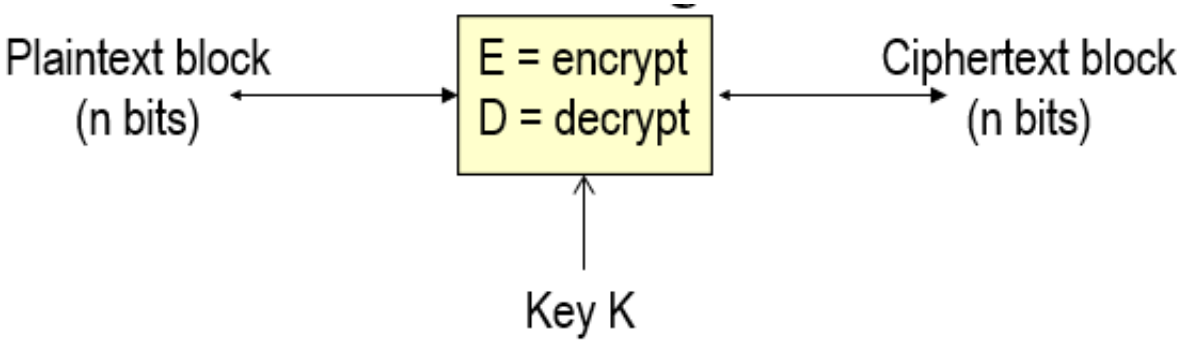
1. **Encrypt-and-MAC (SSH):** Si calcola il MAC sul testo in chiaro e lo si accoda al testo cifrato. L'output è $(ENC(M), MAC(M))$. Questo approccio è considerato il più insicuro, poiché il MAC, essendo calcolato sul plaintext e non cifrato, potrebbe rivelare informazioni sul messaggio originale.
2. **MAC-then-Encrypt (TLS fino a 1.2):** Si calcola il MAC sul testo in chiaro, lo si accoda al messaggio e si cifra il tutto. L'output è $ENC(M||MAC(M))$. Sebbene intuitivo, questo approccio è vulnerabile ad attacchi su testo cifrato scelto, come vedremo in dettaglio.
3. **Encrypt-then-MAC (IPsec):** Prima si cifra il messaggio e poi si calcola il MAC sul testo cifrato. L'output è $(ENC(M), MAC(ENC(M)))$. Questo è l'approccio **provably secure** e raccomandato dalla comunità crittografica, a condizione che la cifratura sia semanticamente sicura e il MAC non falsificabile. Il MAC sul testo cifrato garantisce che quest'ultimo non possa essere manipolato in alcun modo.



La scelta di TLS di utilizzare l'approccio MAC-then-Encrypt ha dato origine a una lunga serie di vulnerabilità pratiche.

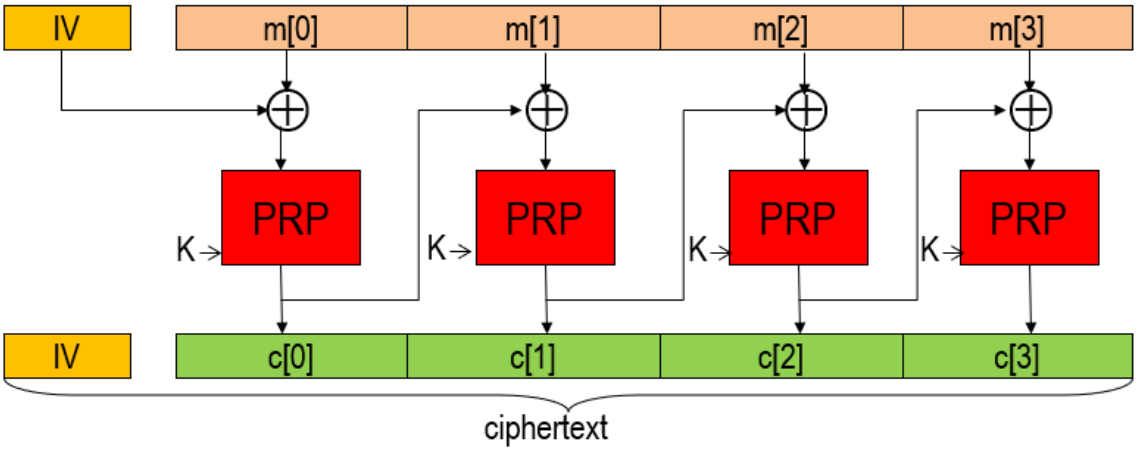
3 Contesto: la Modalità di Cifratura CBC in TLS

Per comprendere gli attacchi, è necessario analizzare la modalità di cifratura a blocchi più comune utilizzata in TLS prima della versione 1.3: il **Cipher Block Chaining (CBC)**. A differenza della modalità **ECB (Electronic Code Book)**, che cifra ogni blocco di testo in chiaro in modo indipendente (e quindi non è sicura, poiché blocchi di plaintext identici producono blocchi di ciphertext identici, rivelando pattern nei dati), la modalità CBC introduce una dipendenza tra i blocchi.

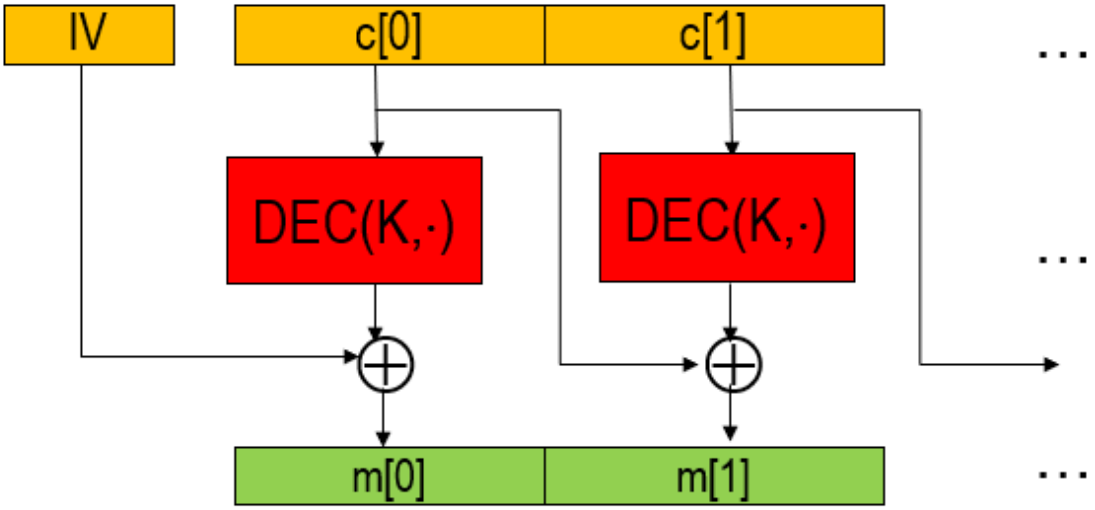


3.1 Funzionamento del CBC

- **Cifratura:** Ogni blocco di testo in chiaro m_i viene prima combinato tramite un'operazione di XOR con il blocco di testo cifrato precedente c_{i-1}, e solo dopo viene passato alla primitiva di cifratura a blocchi (es. AES). Il primo blocco viene combinato con un **Vettore di Inizializzazione (IV)** casuale. La formula è: $c_i = E_K(m_i \oplus c_{i-1})$, con $c_{-1} = IV$.



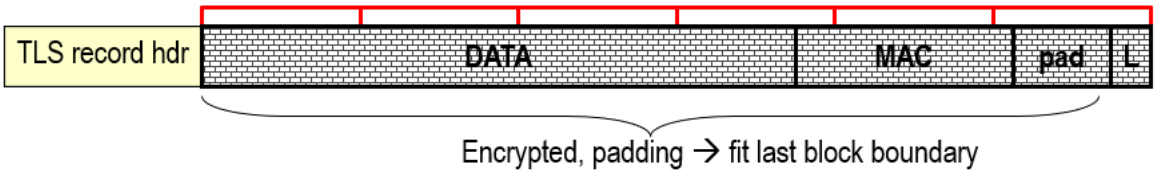
- **Decifratura:** Il processo è invertito. Ogni blocco di testo cifrato c_i viene prima decifrato con la chiave, e il risultato viene combinato in XOR con il blocco di testo cifrato precedente c_{i-1} per ottenere il blocco di testo in chiaro originale. La formula è: $m_i = D_K(c_i) \oplus c_{i-1}$.



Questa "concatenazione" assicura che anche blocchi di plaintext identici producano blocchi di ciphertext diversi, garantendo la sicurezza semantica.

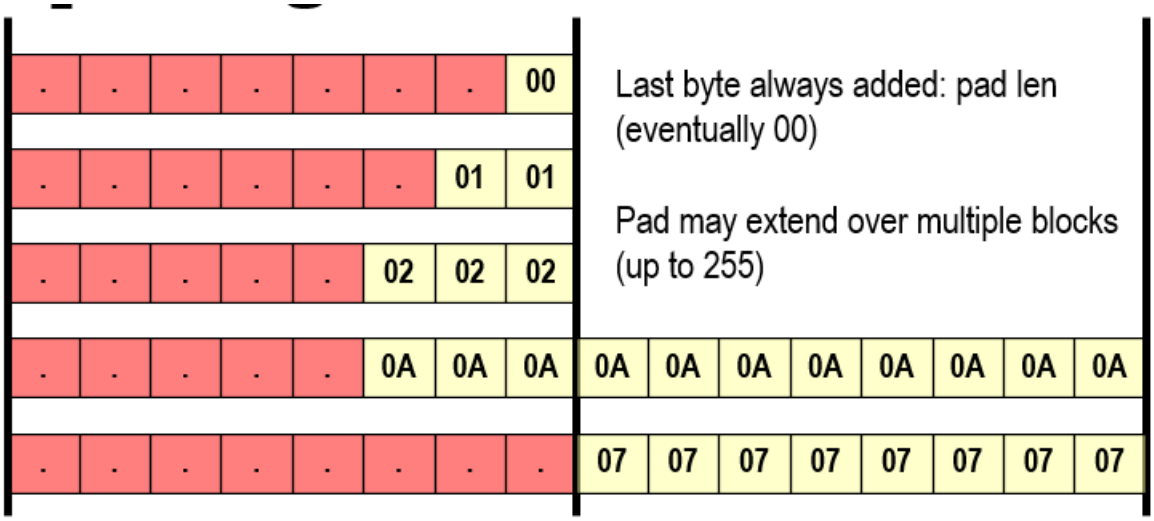
4 L'Attacco Padding Oracle su TLS in modalità CBC

L'attacco, scoperto da Serge Vaudenay nel 2002, sfrutta l'interazione tra l'approccio MAC-then-Encrypt, la modalità CBC e il modo in cui il server gestisce gli errori di decifratura.



4.1 Il Padding in TLS

I cifrari a blocchi operano su blocchi di dimensione fissa (es. 16 byte per AES). Poiché il messaggio (dati + MAC) raramente ha una lunghezza che è un multiplo esatto della dimensione del blocco, è necessario aggiungere del *padding* prima della cifratura. Il padding in TLS è strutturato così: se devono essere aggiunti $L+1$ byte di padding, si aggiungono $L+1$ byte, tutti con il valore L. Il valore dell'ultimo byte indica la lunghezza totale del padding.

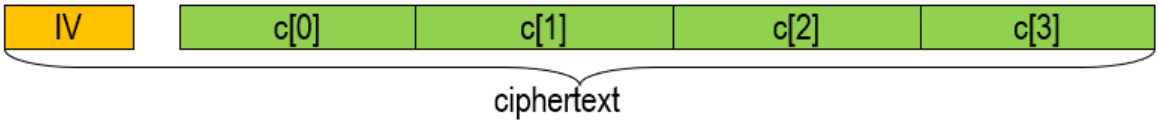


4.2 La Vulnerabilità: l'Oracolo

Il cuore della vulnerabilità risiede nel comportamento del server TLS 1.0 durante la decifratura di un record:

1. **Decifra** il messaggio.
2. **Controlla il padding:** Verifica se l'ultimo byte è un valore valido e se i byte precedenti corrispondono. Se il padding non è corretto, il server risponde con un alert di tipo `decryption_failed`.
3. **Controlla il MAC:** Se il padding è corretto, lo rimuove e verifica il MAC sul messaggio rimanente. Se il MAC è errato, risponde con un alert `bad_record_mac`.

Un attaccante può distinguere tra un errore di padding e un errore di MAC. Questa informazione binaria ("padding corretto" vs "padding errato") agisce come un **oracolo** che l'attaccante può interrogare.



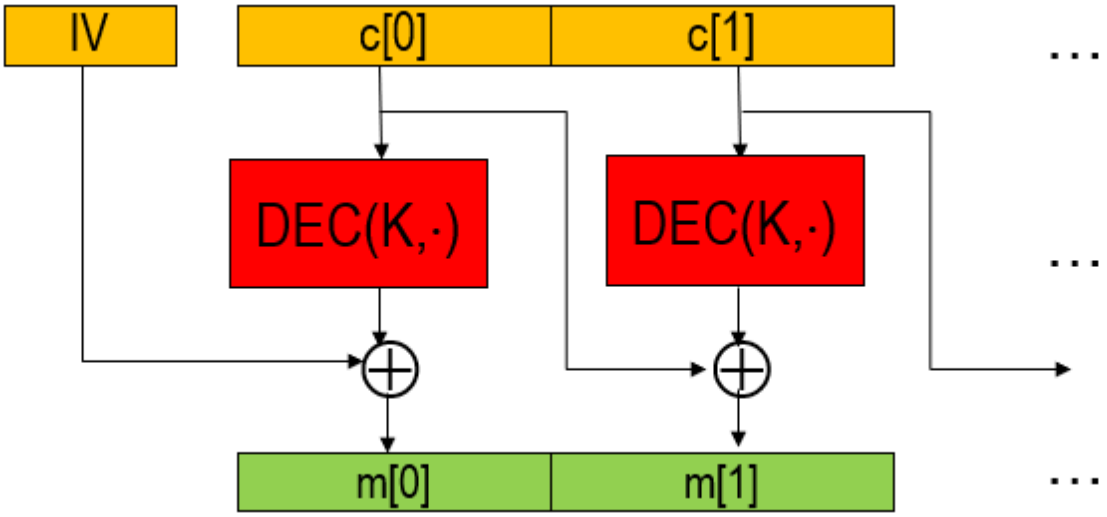
4.3 Meccanismo dell'Attacco

L'obiettivo dell'attaccante è decifrare un blocco di testo cifrato, diciamo c_1, avendo intercettato la coppia (c_0, c_1). Ricordiamo la formula di decifratura: $m_1 = D_K(c_1) \oplus c_0$. L'attaccante non conosce né $D_K(c_1)$ né m_1, ma conosce c_0. L'idea è di manipolare c_0 per "indovinare" i byte di m_1.

1. **Decifrare l'ultimo byte di m_1:**

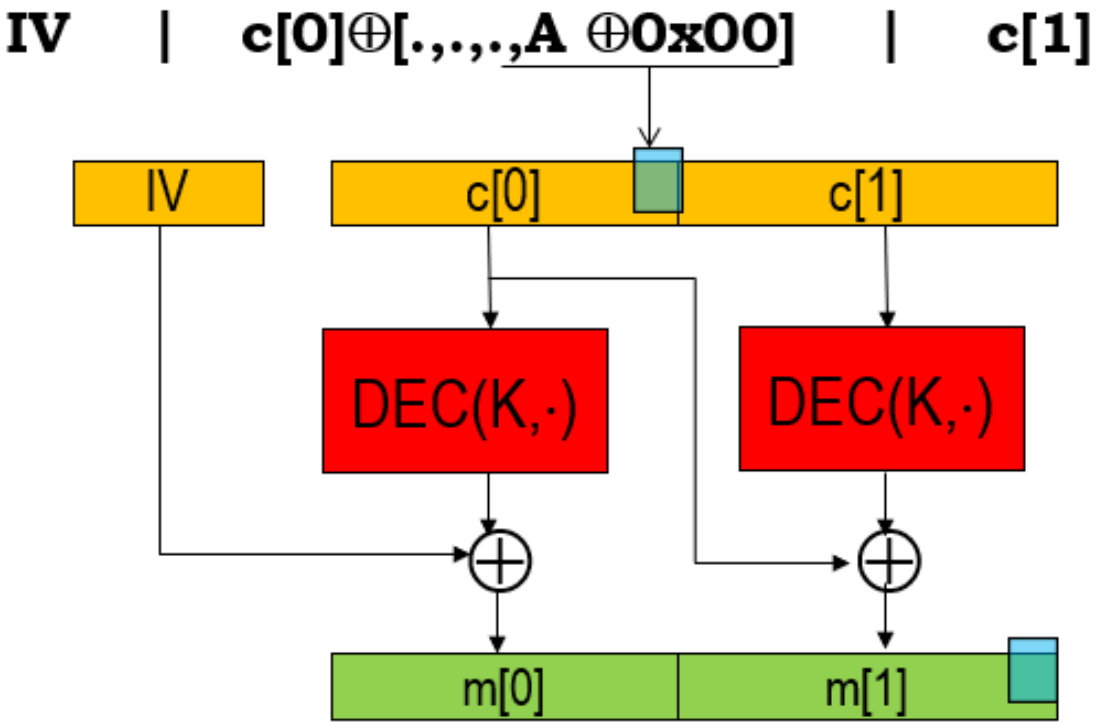
- L'attaccante costruisce un blocco c'_0 e lo invia al server insieme a c_1. Il server calcolerà $m'_1 = D_K(c_1) \oplus c'_0$.
- L'attaccante vuole forzare l'ultimo byte di m'_1 ad essere '0x00', che corrisponde a un padding valido di 1 byte.
- Prova tutti i 256 possibili valori per l'ultimo byte di un blocco 'guess'. Costruisce c'_0 modificando l'ultimo byte di c_0 come segue: $c'_0[\text{last}] = c_0[\text{last}] \oplus \text{guess} \oplus 0\text{x}00$.

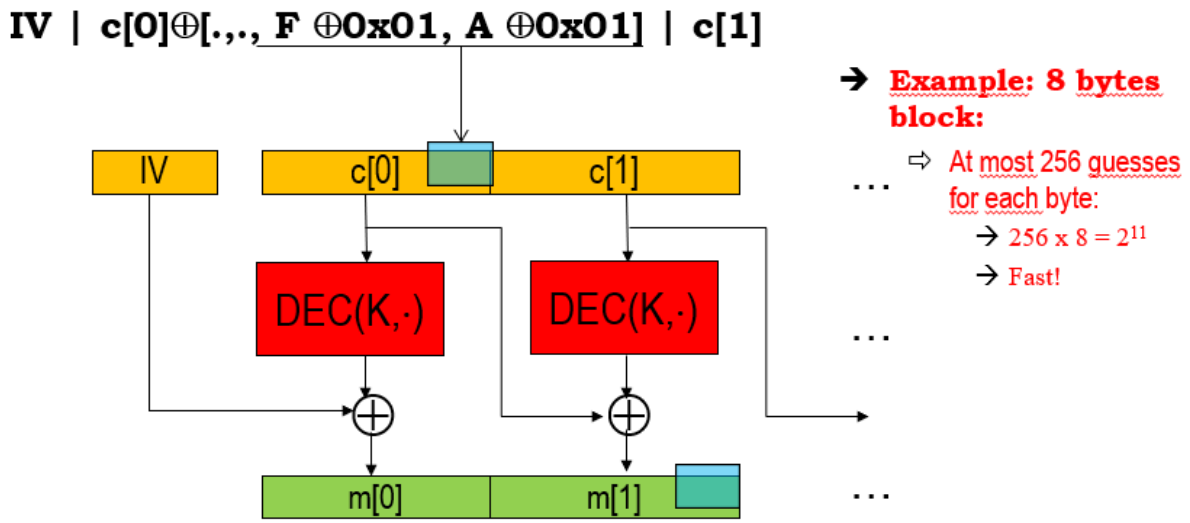
- Quando il server riceve (c'_0, c_1), calcola: $m'_1[\text{last}] = D_K(c_1)[\text{last}] \oplus c'_0[\text{last}] = (m_1[\text{last}] \oplus c_0[\text{last}]) \oplus (c_0[\text{last}] \oplus \text{guess} \oplus 0\text{x}00) = m_1[\text{last}] \oplus \text{guess} \oplus 0\text{x}00$.
- Se l'ipotesi 'guess' è corretta, cioè guess $= m_1[\text{last}]$, allora $m'_1[\text{last}]$ diventa '0x00'. Il padding è valido, e il server risponderà con `bad_record_mac`. L'attaccante ha trovato il valore dell'ultimo byte di m_1 in al più 256 tentativi.



2. **Decifrare i byte precedenti:**

- Ora che l'attaccante conosce l'ultimo byte, può procedere a decifrare il penultimo. L'obiettivo è forzare gli ultimi due byte del messaggio decifrato ad essere '0x01, 0x01' (padding valido di 2 byte).
- Si itera sui 256 possibili valori del penultimo byte e si costruisce un c'_0 modificato in modo da forzare il risultato desiderato.
- L'attacco continua, byte dopo byte, fino a decifrare l'intero blocco.





5 Contromisure e Conseguenze

La soluzione apparentemente semplice era far sì che il server rispondesse sempre con lo stesso messaggio di errore, indipendentemente dalla causa del fallimento. Tuttavia, anche questa contromisura fu aggirata tramite **attacchi side-channel basati sul tempo di risposta** (timing attacks): il calcolo del MAC richiede un tempo leggermente diverso a seconda che il padding sia valido o meno.

Questo ha dato il via a una "guerra" durata quasi 15 anni:

- **Lucky Thirteen (2013):** Un attacco che sfruttava differenze di tempo minime nel processo di validazione del MAC in TLS 1.2.
- **POODLE (2014):** Sfruttava un attacco di downgrade a SSLv3 (che ha una gestione del padding ancora più debole) combinato con un padding oracle.
- **Lucky Microseconds (2015), CVE-2016-2107 (2016):** Ulteriori attacchi basati su canali laterali che sfruttavano anche le patch implementate per correggere le vulnerabilità precedenti.

Questa lunga e dolorosa serie di vulnerabilità è riassunta dal **Cryptographic Doom Principle** di Moxie Marlinspike: "se devi eseguire una qualsiasi operazione crittografica prima di aver verificato il MAC su un messaggio ricevuto, questo porterà inevitabilmente al disastro".

La soluzione definitiva è arrivata solo con **TLS 1.3**, che ha reso obbligatorio l'uso di cifrari AEAD, eliminando alla radice il problema della combinazione manuale di cifratura e autenticazione e chiudendo finalmente questa "scatola di Pandora".