



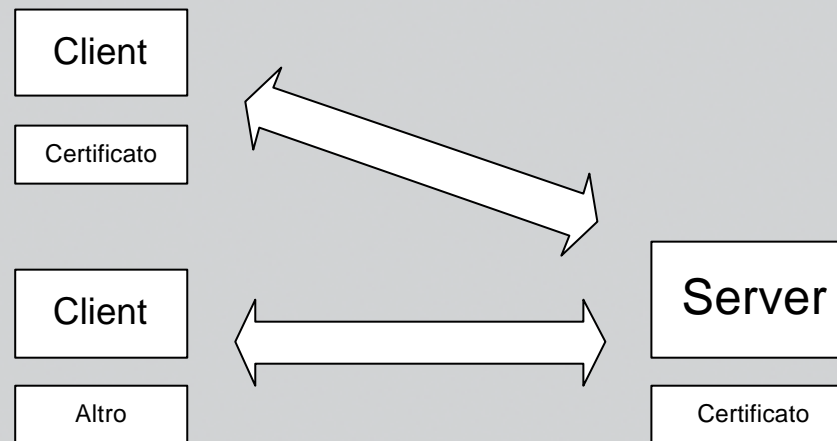
# Side Channel Attack contro il protocollo SSL

Giuliano Bertoletti  
E-Security Manager  
gb@intrinsic.it

Webb.it  
Padova, 10 maggio 2003

**SYMBOLIC**

## Comunicazione sicura client-server su protocollo TCP/IP



Tipicamente si hanno due tipologie d'uso:

- Autenticazione mutua con certificato digitale
- Autenticazione del server ed eventuale autenticazione del client con altri sistemi.

## Caso#1: Autenticazione mutua con certificato digitale

- Entrambe le parti devono essere munite di certificato digitale, firmato da una CA da entrambi riconosciuta (trusted).
- Il protocollo SSL fornisce sempre l'autenticazione del server poiché questa è essenziale per stabilire un canale sicuro.
- L'autenticazione del client viene fornita anch'essa dal protocollo SSL, previo utilizzo di un certificato digitale.
- Una volta stabilito un canale sicuro, le due parti non necessitano di protocolli aggiuntivi di autenticazione e possono iniziare il dialogo.

## **Caso#2 : Autenticazione del server ed eventuale autenticazione del client con altri sistemi**

- Solo il server deve essere munito di certificato digitale.
- Il client deve riconoscere la CA che ha firmato il certificato del server, altrimenti l'identità del server non è garantita.
- Viene stabilita una comunicazione sicura fra un server autenticato e un client ignoto.
- Il client può inoltrare successivamente informazioni al server che permettano a quest'ultimo di identificarlo (es. password), ma l'identità del client non è responsabilità del protocollo SSL.

## Utilizzi atipici: connessione anonima

- E' possibile avere anche una comunicazione anonima fra due entità a livello di protocollo SSL.
- Un possibile impiego è quello di delegare a protocolli di livello superiore la responsabilità di autenticare le parti o comunque effettuare ulteriori controlli.
- Di solito tuttavia questa procedura non è utilizzata.
- La privacy senza autenticazione, espone a problemi di attacchi di tipo *man in the middle*.

## Elementi costitutivi

- **Protocollo di Handshake**: responsabile per l'autenticazione delle parti e la negoziazione dei parametri crittografici atti a rendere sicuro il canale.
- **Record layer**: implementa la bulk encryption per rendere il canale sicuro e permettere il trasferimento dati.

## Attacchi di tipo side channel presentati

- Sul protocollo di Handshake:
  - Attacco PKCS / Versione
  - RSA Timing Attack
- Sul record layer:
  - Attacco pad/cbc

## Protocollo di Handshake

- Lo scopo è quello di permettere lo sharing di un payload di 48 byte chiamato pre-master secret.
- Al completamento dell'handshake, entrambe le parti, e solo esse, saranno in possesso del pre-master secret.
- Esso costituisce il seme di innesco per un generatore di numeri casuali (pensato per scopi crittografici), il quale ha il compito di produrre tutto il materiale crittografico di cui le due parti hanno di bisogno per procedere (es. master secret, chiavi di sessione, ecc.).
- Mediante opportuna sincronizzazione dei rispettivi generatori, entrambe le parti saranno in grado di produrre gli stessi risultati.

## Handshake e algoritmo RSA

- Uno dei possibili algoritmi con cui il pre-master secret viene trasmesso dal client al server è RSA.
- Si tratta quindi di trasmettere questo vettore di 48 byte attraverso un crittogramma RSA...
- Lo standard PKCS#1 definisce il formato con cui questa chiave deve essere trasmessa (EME-PKCS1-v\_1.5)

## Riepilogo algoritmo RSA

RSA basa la sua sicurezza sulla difficoltà di fattorizzare numeri composti dal prodotto di primi molto grandi (tipicamente di 200 cifre decimali e oltre).

- Si definiscono due primi molto grandi  $p$  e  $q$ .
- Si calcola il prodotto  $n=p \cdot q$
- Si seleziona a caso un numero  $e$  in modo che risulti relativamente primo a  $(p-1) \cdot (q-1)$
- Quindi con l'algoritmo di euclide esteso, si calcola un  $d$  tale per cui  $e \cdot d \bmod n = 1$
- $e$  costituisce la chiave pubblica,  $d$  quella privata,  $p$  e  $q$  vengono scartati dopo la produzione delle chiavi.

## Riepilogo algoritmo RSA

L'encryption di un blocco  $m$  per produrre un crittogramma  $c$  avviene attraverso l'operazione:

$$c = m^e \bmod n$$

l'operazione di decryption invece è data da:

$$m = c^d \bmod n$$

pertanto una proprietà delle chiavi RSA è che vale l'identità:

$$m = (m^e)^d \bmod n$$

Il plain text  $m$  e il ciphertext  $c$  sono compresi fra  $0$  e  $n-1$  essendo il modulo l'ultima operazione effettuata su di essi.

## RSA - Caveat Emptor

L'algoritmo per quanto teoricamente sicuro, presenta nella pratica alcuni rischi per gli implementatori.

Uno di questi rischi è il *guessing del ciphertext*.

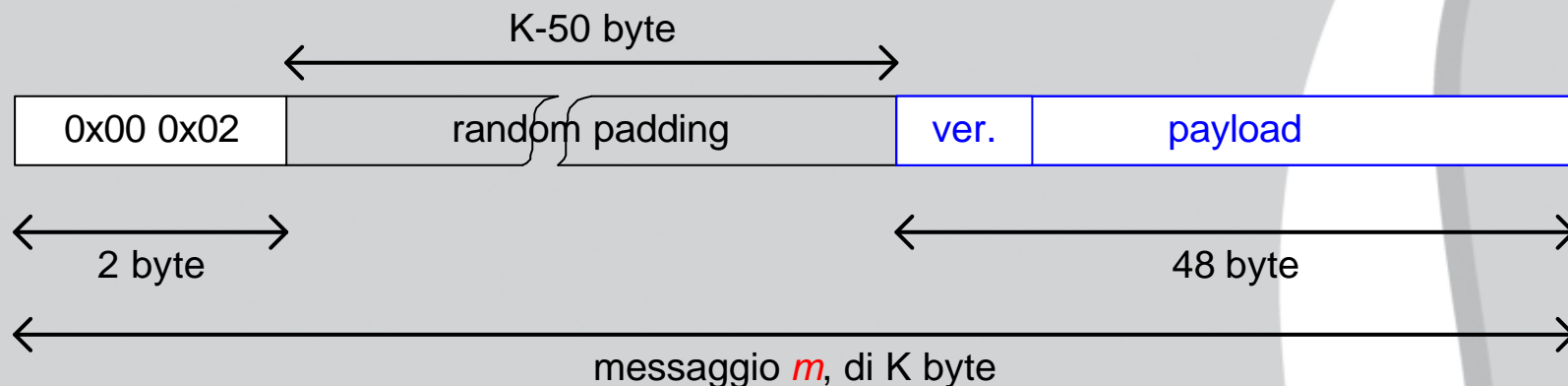
Per quanto senza  $d$  non sia possibile ricavare  $m$  da  $c$ , se non si fa attenzione, può essere possibile fare delle ipotesi su  $m$  e verificare  $c$  con la formula diretta:

$$c = m^e \bmod n$$

Per questa ed altre ragioni, il protocollo PKCS#1 tratta di come effettuare l'encryption dei messaggi  $m$ .

## PKCS#1

Si distingue  $m$ , che è il messaggio da dare in pasto ad RSA da  $p$  che è il payload, ovvero il messaggio in chiaro da cifrare. Tipicamente  $p$  è molto più piccolo di  $m$ . Nel caso di SSL,  $p$  è il pre-master secret. Il numero di byte  $K$  di cui è composto il messaggio  $m$  è tale per cui  $256^K < n < 256^{K+1}$ .



Il numero K dipende quindi dalla lunghezza della chiave.

## EME-PKCS1-v1.5

Il problema sorge in alcune implementazioni, a causa di specifiche lacunose sulla gestione degli errori da parte del protocollo, in caso di messaggi malformati.

Tali errori si aggravano se non vengono trattati come fatali, e non fanno abortire la connessione in modo irrecuperabile.

Su alcune implementazioni essi provocano semplici Alert che pur terminando la connessione, la lasciano in uno stato *resumable* cioè la connessione può essere ripristinata senza dover rinegoziare tutti i valori.

## Schema dell'attacco

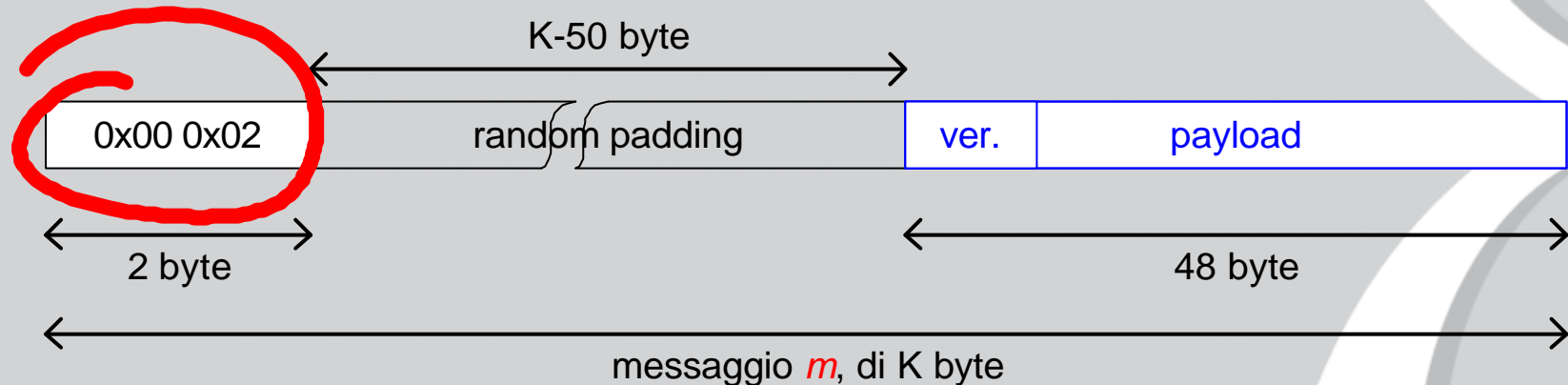
Quando un client si connette al server, l'intruso ascolta la connessione e registra ogni pacchetto trasmesso da ambo le parti.

Successivamente l'intruso negozia una nuova connessione e invia come pre-master secret cifrato un  $c'$  derivato (secondo opportune regole) dal  $c$  originariamente trasmesso dal vero client.

Nel tentativo di decrittare questo  $c'$ , pur fallendo, il server rilascerà informazioni sul vero  $m$  da cui  $c$  era stato prodotto dal vero client.

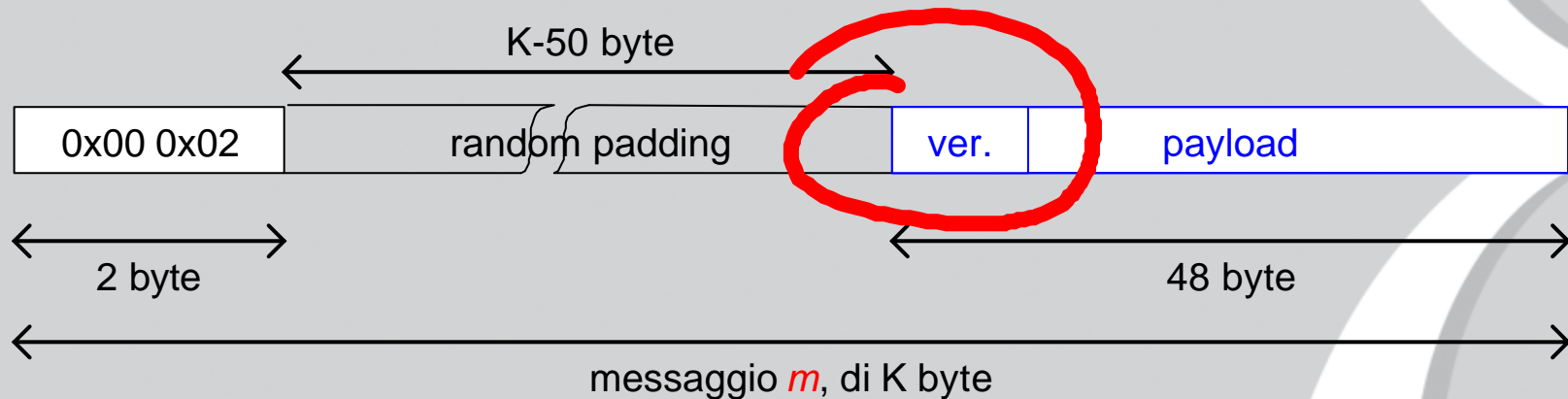
L'intruso effettuerà altre connessioni fino a quando le informazioni fornite dal server saranno sufficienti a determinare  $m$ .

## EME-PKCS1-v1.5



Alcuni server emettono un errore specifico e terminano la connessione se, ricevuto il crittogramma  $c$ , questo non si decifra in un plain-text  $m$  avente l'header in formato PKCS#1, cioè coi primi due byte come stabilito in figura.

## EME-PKCS1-v1.5



Un'altra vulnerabilità è sul numero di versione.

Per evitare un rollback attack, il client usa il premaster secret per il massimo numero di versione supportata.

Tale informazione è presente altrove non criptata.

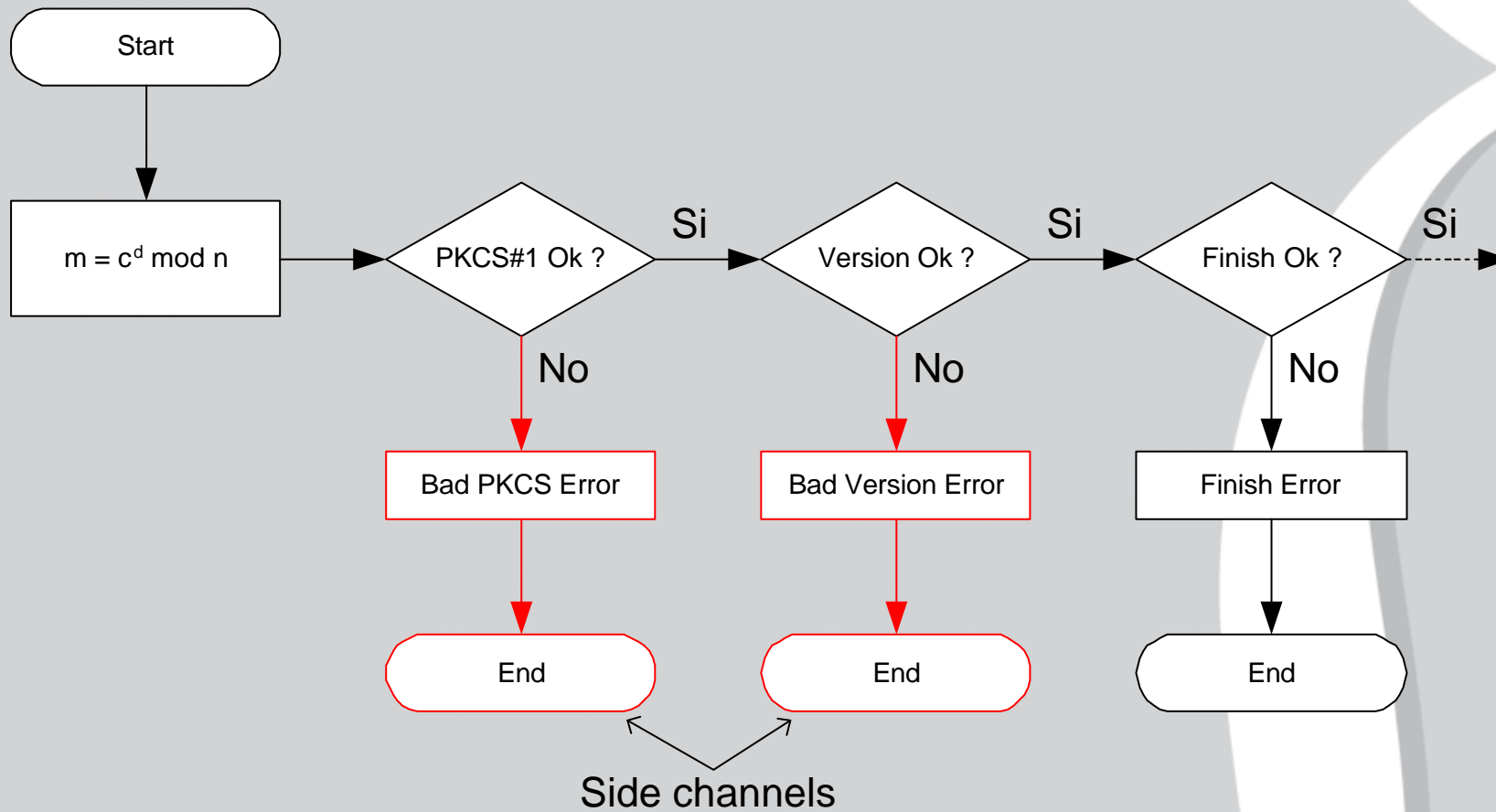
Il server confronta le informazioni, e il protocollo fallisce se i risultati non corrispondono. **Il problema è come fallisce...**

## Attacco di Bleichenbacher e successiva estensione

Mandando al server un certo numero di crittogrammi opportunamente confezionati e ricevendo una risposta *si/no* sul fatto che il messaggio sia PKCS#1 *compliant*, è in teoria possibile ridurre ad uno il numero di possibili  $m$  candidati e quindi arrivare a determinare il pre-master secret.

La stessa informazione può in alcuni casi essere ricavata dalla risposta del server in merito al confronto fra i numeri di versione, dal momento che tale confronto è solitamente fatto soltanto dopo che il messaggio è stato ritenuto PKCS#1 *compliant*.

## Errato comportamento del server



## Timing attack

Il timing attack si basa sulla misurazione del tempo di risposta del server per ottenere informazioni.

Se ad esempio il server non fa distinzioni sul tipo di errore, ma impiega un tempo diverso a rispondere, questo può permettere all'intruso di dedurre le informazioni che il server non restituisce esplicitamente.

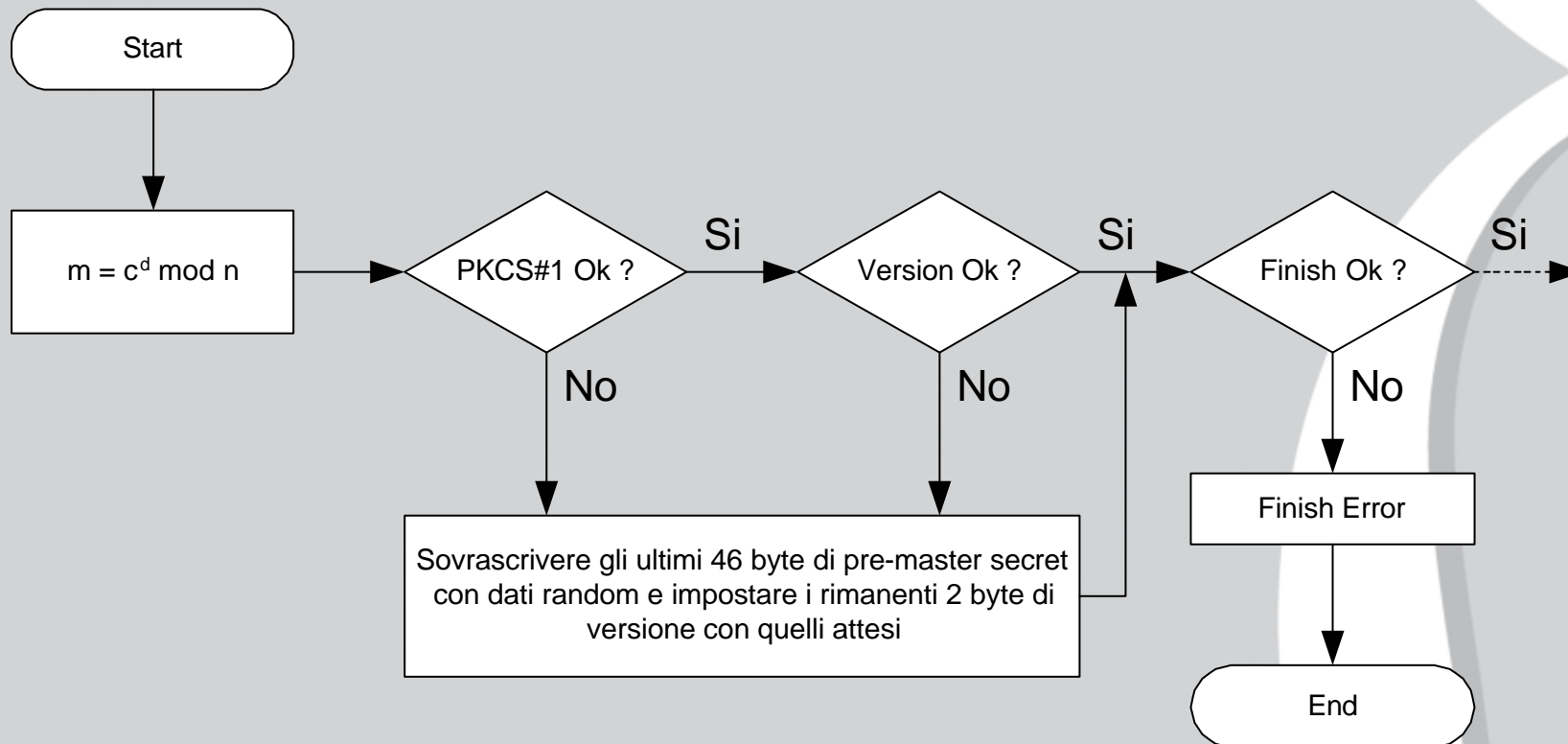
Si rischia di cadere nel caso precedente.

## Attacco di Bleichenbacher e successiva estensione

Il problema generale può essere risolto in questo modo:

- Nel caso *m* non sia PKCS o version compliant viene sostituito il pre-master secret con byte casuali, viene impostato il numero di versione atteso e si procede normalmente.
- In questo modo si ritarda la failure a quando server e client si scambieranno il messaggio di Finished dell'Handshake.
- Poiché infatti i due pre-master secret saranno diversi, i messaggi di Finished saranno diversi da quelli attesi e la connessione verrà abortita.
- L'intruso non potrà tuttavia stabilire per quale ragione tale connessione è stata terminata (errato pre-master secret o *m* PKCS non compliant).

## Corretto comportamento del server



## Dati relativi all'attacco (V.Klima, O.Pokorný, T.Rosa)

Server AMD Athlon 1GHz/256Mb Ram su rete locale 10/100, chiave RSA 1024 bit.

Decryption del pre-master secret:

Tempo impiegato: 14h 22m 45s.

Numero di connessioni effettuate: 2.727.042

Stime su server Internet rendono plausibile un tempo di 54h, distribuibili in un mese, effettuando attacchi di circa 2h al giorno.

## Limitazioni

Le principali limitazioni all'attacco proposto (nel caso di server vulnerabili):

- Possibile collasso del server dovuto alla quantità di log prodotti durante l'attacco.
- Limitazione dovuta ai ritardi e al traffico della rete
- Effettiva capacità del server a gestire un così elevato numero di connessioni.

Inoltre ci si aspetta che per un server correttamente gestito, l'amministratore rilevi per tempo comportamenti anomali e blocchi il client che sta sferrando l'attacco.

## Timing attack

Un altro esempio di timing attack riguarda il calcolo operato dal server quando effettua la decryption del crittogramma RSA.

$$m = c^d \bmod n$$

A seconda di quanto tempo impiega il server ad effettuare la decryption e a rispondere, l'intruso può acquisire informazioni che, previo un opportuno numero di tentativi gli permettono di dedurre  $d$ .

In particolare il tempo di risposta varia a seconda di quanto  $c$  è prossimo ai fattori  $p$  e  $q$  dai quali si è partiti per produrre la chiave.

## Timing attack

La buona riuscita di questo attacco dipende da vari fattori:

- Distanza del server dal client che attacca.
- Implementazione di RSA fatta su quel particolare server.
- Livello di ottimizzazione del codice prodotto dal compilatore.
- Eventuali contromisure adottate per ovviare all'attacco.

## RSA blinding

Una efficiente contromisura al timing attack e il blinding.

Il server sceglie un numero  $r$  casuale compreso fra  $0$  e  $n-1$ .

Per decrittare  $m = c^d \bmod n$ , il server calcola:

$$m' = (r^e \cdot c)^d \bmod n$$

$$m = m' / r \bmod n = (r^{ed} / r) c^d \bmod n$$

Risultata  $m$  perché per le proprietà di RSA vale:  $r^{ed} \bmod n = r$

Poiché  $r$  è casuale anche il prodotto  $(r^e \cdot c) \bmod n$  è casuale e questo nasconde quanto  $c$  sia prossimo a  $p$  o  $q$ .

Si ha una perdita in performance dal 2% al 10%.

## Il record layer

Una volta terminata la fase di handshake e stabilito l'algoritmo con cui proteggere la connessione, il controllo passa al record layer.

Esso ha il compito di cifrare i dati ad un capo e decifrarli all'altro, in modo da proteggere il segnale in transito.

Non si tratta di una semplice encryption, poiché occorrono alcuni accorgimenti (es. un MAC).

Inoltre se viene usato un block cipher, il testo viene cifrato a blocchi di  $n$  bit (tipicamente  $n=64$  o  $n=128$ ), occorre gestire il padding in caso si debbano mandare messaggi non allineati alla dimensione del blocco.

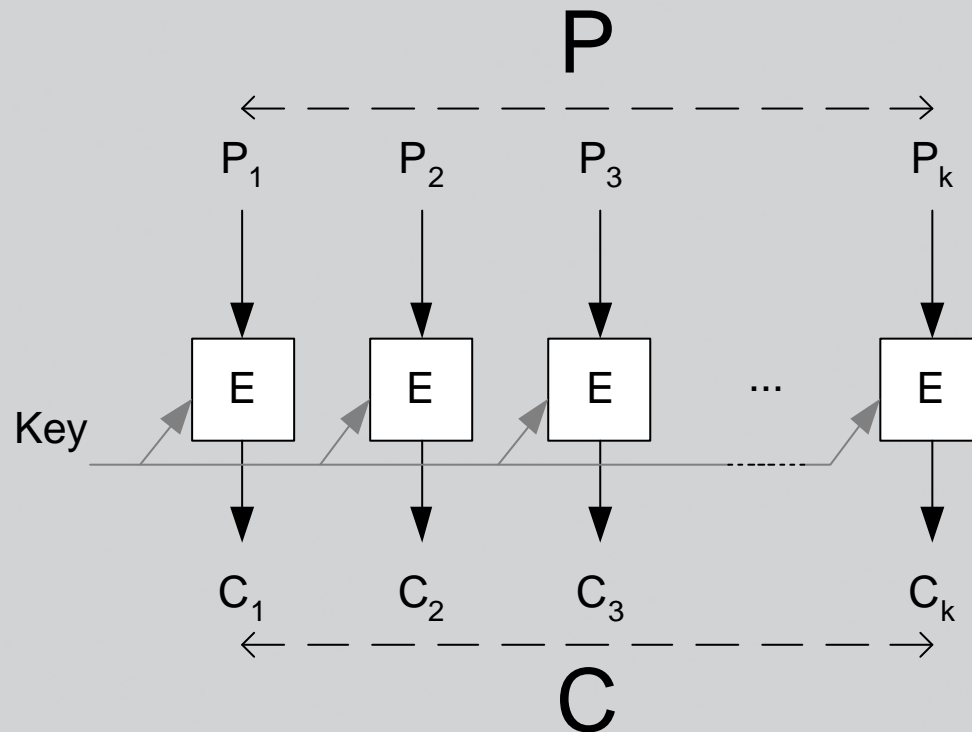
## Record layer e block cipher

Esiste una vulnerabilità del protocollo SSL a livello di record layer quando viene impiegato un block-cipher in modalità CBC in congiunzione ad un sistema di padding.

L'attacco può funzionare anche se il server non consente di fare il resume della connessione passata.

Sebbene le condizioni siano apparentemente meno favorevoli del caso precedente, esistono situazioni reali, anche se non fra le più comuni, in cui questo attacco può essere portato a termine con successo.

## Block cipher in modalità ECB



Encryption

- $C_i = E_{key}(P_i)$

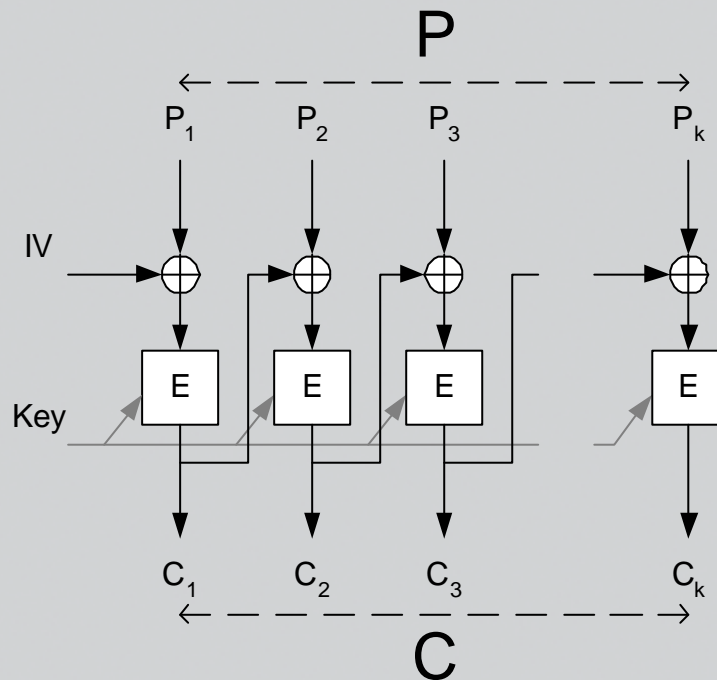
Decryption

- $P_i = D_{key}(C_i)$

Il messaggio  $P$  viene suddiviso in  $k$  blocchi  $P_1, \dots, P_k$  di  $n$  bit.

I blocchi vengono cifrati in modo indipendente producendo  $k$  crittogrammi  $C_1, \dots, C_k$  che vanno a formare il ciphertext  $C$ .

## Block cipher in modalità CBC



### Encryption

- $C_1 = E_{\text{key}}(P_1 + IV)$
- $C_i = E_{\text{key}}(P_i + C_{i-1})$

### Decryption

- $P_1 = D_{\text{key}}(C_1) + IV$
- $P_i = D_{\text{key}}(C_i) + C_{i-1} \quad (i > 1)$

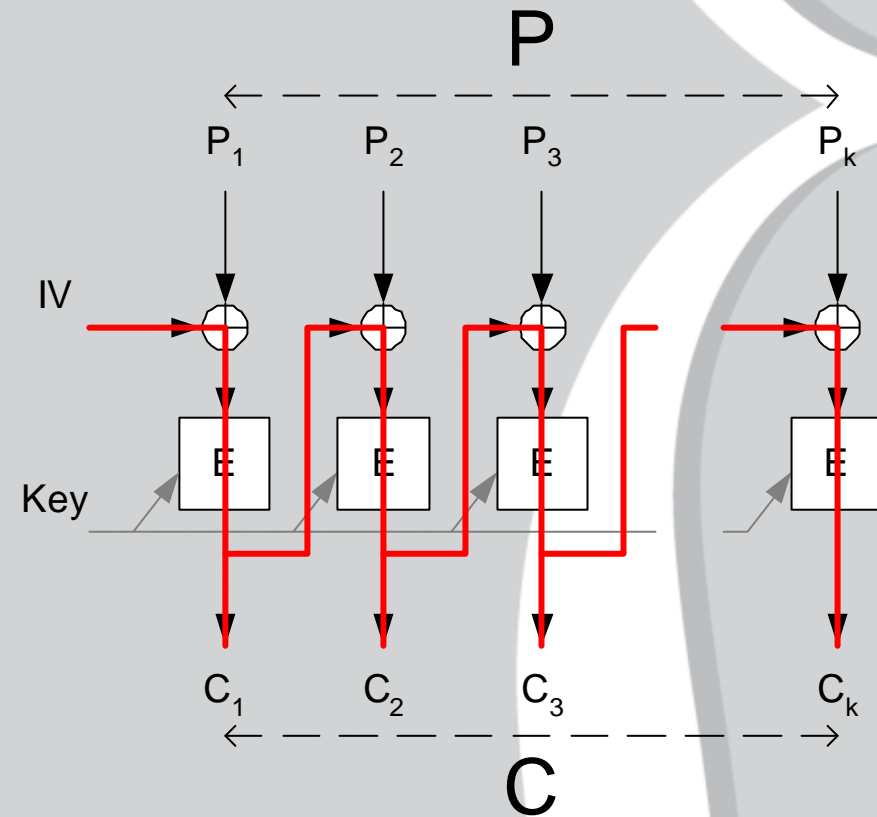
La suddivisione in blocchi del messaggio è la stessa dell'ECB.

Un determinato crittogramma  $C_k$  dipende in modo più complesso da tutti i blocchi  $P_1, \dots, P_k$  che lo precedono. <sup>31</sup>

## Block cipher in modalità CBC

Un vettore di inizializzazione diverso ogni volta garantisce che lo stesso messaggio, anche se cifrato con la stessa chiave, appaia sempre come sequenza differente di crittogrammi.

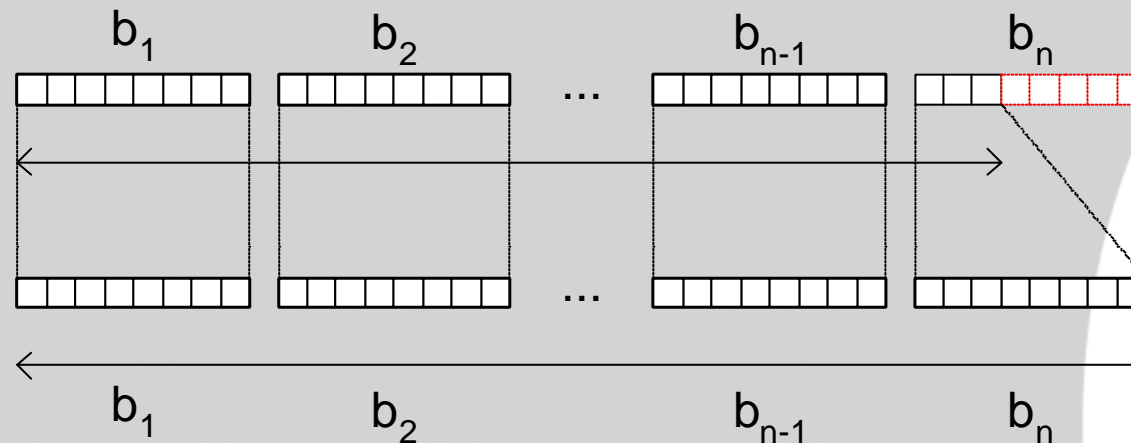
Una variazione su IV si propaga immediatamente a tutti i crittogrammi.



L'IV non deve essere tenuto segreto e viene usualmente allegato in chiaro al messaggio cifrato da trasmettere.

## Padding

Quando il messaggio da crittare non è multiplo della lunghezza del blocco su cui opera il cifrario, occorre completare l'ultimo blocco con una sequenza di bit ben definita chiamata **padding**.



L'unico accorgimento è quello di far sì che tale sequenza possa a destinazione essere individuata in modo univoco ed eliminata in modo da recuperare il messaggio originale.

## Padding

Il pad utilizzato in SSL (RFC2040) consiste nell'aggiungere in coda al messaggio un numero X di byte, aventi valore X, ove X è il numero di posti da riempire per allinearsi al blocco.

Notare che per **evitare ambiguità in fase di decodifica** il pad deve essere sempre almeno di un byte, se il messaggio è allineato al blocco, viene creato un ulteriore blocco con solo il pad.

Esempi con un blocco del cifrario di 8 byte (n=64 bit):

- ... 01 47 81 ff 45 03 03 03
- ... 07 07 07 07 07 07 07 07
- ... f1 6f 9b 19 04 23 4b 01
- ... 39 46 8a fa eb c9 1e 01 08 08 08 08 08 08 08 08 34

## Message Authentication Code (MAC)

Il MAC è una sequenza di byte che permette di stabilire l'integrità di un blocco dati.

Tipicamente si utilizza una funzione di hash che opera sul blocco dati e su una chiave nota solo ai comunicanti.

In questo modo, poiché l'intruso non è in grado di alterare il blocco dati e ricalcolare il MAC corretto (poiché non conosce la chiave), qualunque modifica viene rilevata.

Es.

$$\text{MAC} = \text{MD5}(\text{blocco dati} \mid \text{secret\_key} \mid \text{label})$$

## Tipologia di errori

Le informazioni che viaggiano sul record layer, oltre ad essere cifrate vengono anche autenticate da un MAC.

Se i dati in transito vengono alterati, il protocollo rileva il cambiamento e genera un errore.

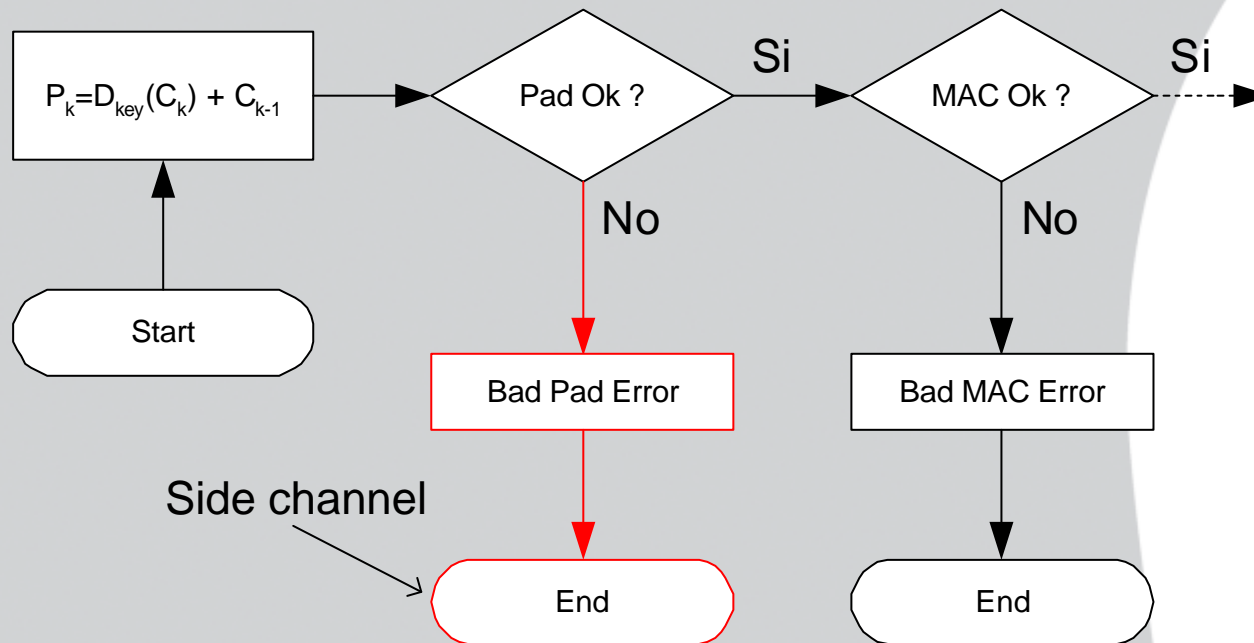
Un errore è generato anche quando il padding non è valido; si consideri ad esempio la stringa seguente:

- ... 07 a9 c2 31 04 03 03 02

Corretti sarebbero ...31 04 03 03 03 , o ...31 04 03 02 02, e anche ...31 04 04 04 04

## Tipologia di errori

Se il server effettua una distinzione fra **errore di pad** ed **errore sul MAC** e comunica al client tale informazione, un intruso può sfruttare questo side channel per acquisire informazioni sul plain-text.



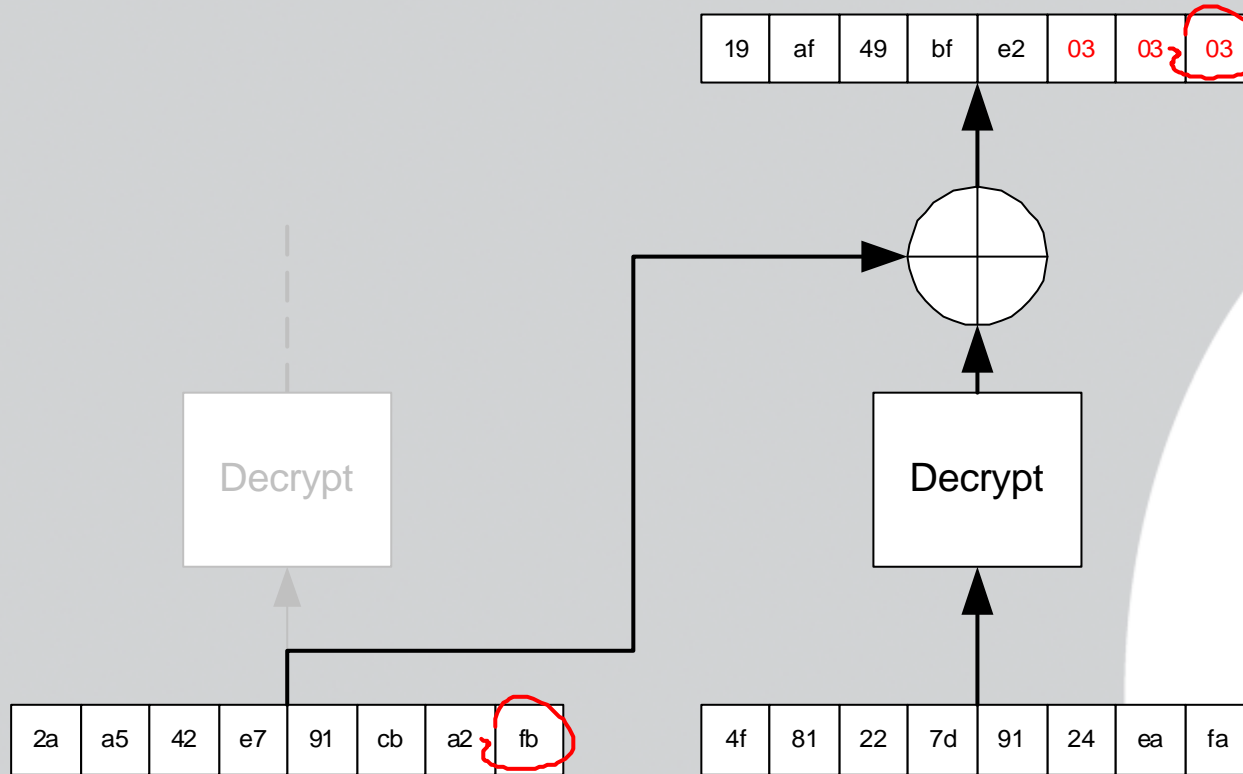
## Dettagli su un generico attacco (S.Vaudenay)

L'attacco funziona se per una qualche ragione il client continua a connettersi al server e a mandare sempre la stessa stringa (sebbene cifrata con chiavi di sessione diverse).

L'intruso effettua un attacco di tipo man-in-the-middle e altera il flusso di dati.

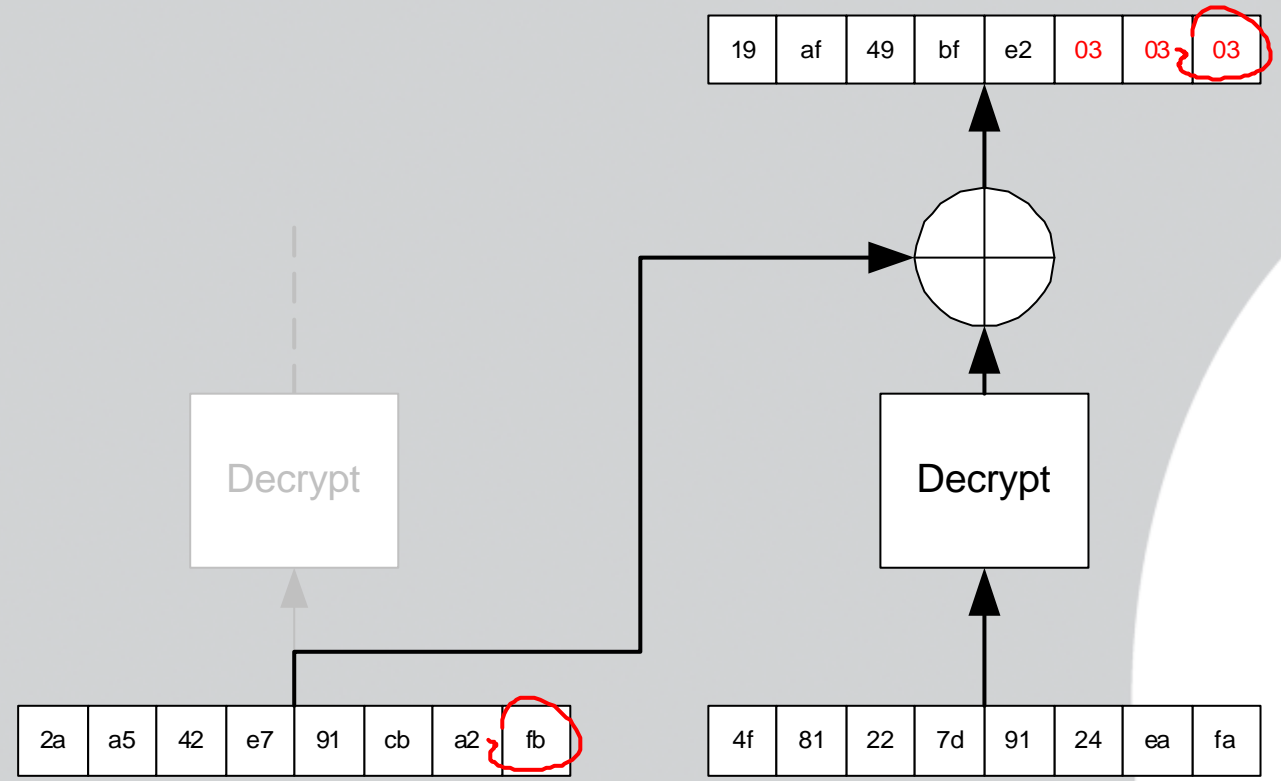
Gli errori restituiti dal server a causa dell'alterazione del flusso possono essere sfruttati dal client per pilotare la sua ricerca e decrittare in parte o in toto il messaggio.

## Schema di attacco



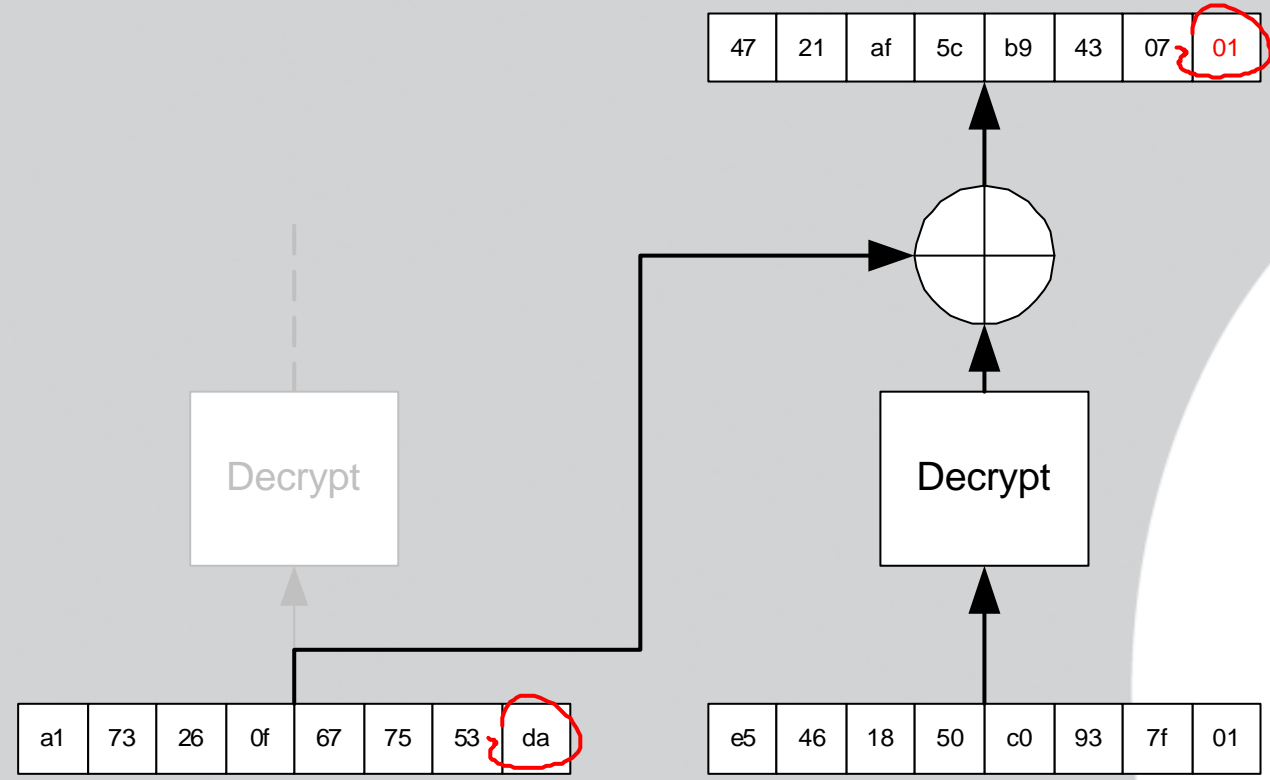
L'intruso altera il padding operando sull'ultimo byte del penultimo blocco. Lo scopo è quello di ottenere uno 0x01 sull'ultimo byte del padding.

## Schema di attacco



Il pad 0x01 viene confermato dal server che restituisce un bad\_record\_mac. A quel punto è facile scoprire il valore dell'ultimo byte trasmesso dal client.

## Schema di attacco



Se il pad è già 0x01, l'intruso se ne accorge poiché tutti i tentativi fatti per trovare un altro pad, a meno di casi anomali, falliscono. I casi anomali si trattano a parte.

## Schema di attacco

Osservazioni:

- Un errore di mac o pad tipicamente è fatale.
- Se tuttavia il client continua a collegarsi e trasmette sempre le stesse informazioni (sebbene in sessioni diverse), l'intruso può comunque recuperare il plain-text.
- Noti gli ultimi **k-byte**, è possibile ricavare il **k+1-esimo**, semplicemente aggiustando il padding dei byte già noti.
- Eventualmente il server può troncare il messaggio per spostarsi ad operare sui blocchi precedenti.

## Costo dell'attacco

In media sono necessari circa  $256/2 = 128$  tentativi per indovinare un byte all'interno dello stream.

La decryption di  $N$  byte costa dunque  $128 \cdot N$  connessioni.

Esistono altri dettagli che complicano leggermente l'attacco.

Per esempio:

- Possono verificarsi casi anomali sul padding che richiedono controlli (e quindi connessioni) extra. Es. ...31 04  
03 02 01
- TLS può utilizzare pad che si estendono oltre il blocco per celare la lunghezza del messaggio. Es. ...31 09 09 09 09 09  
09 09 09 09 09

## Uno scenario concreto

Un client Outlook v6.x si connette ad un server IMAP attraverso SSL.

Per default Outlook effettua il polling del server ogni 5 minuti, mandando login e password.

La stringa mandata ha il seguente formato:

**XXXX LOGIN "username" "password" <13><10><MAC><PAD><LEN>**

Per ogni accesso, se la sessione fallisce, vengono tentate ulteriori connessioni.

In questo modo è possibile ottenere un centinaio di sessioni ogni ora, su cui operare i test.

## Ripercussioni

L'attacco sullo schema di padding ed encryption raramente costituisce un problema, infatti:

- Il principale scopo per cui il protocollo SSL/TLS è utilizzato riguarda la protezione del traffico sul Web (in cui è inverosimile che il client effettui un numero così elevato di connessioni).
- L'algoritmo di bulk encryption utilizzato per default è RC4 che è uno stream cipher e non un block cipher.

## Conclusioni

Restano comunque alcuni aspetti da tenere in considerazione:

- I server SSL/TLS necessitano di essere aggiornati con le ultime patch disponibili.
- Occorre più attenzione nella definizione dei protocolli, soprattutto per quanto riguarda la gestione degli errori.

## Riferimenti bibliografici

- The TLS Protocol (rfc2246)
- SSL Netscape Draft (A.Freier, P.Karlton, P.Kocher)
- PKCS#1 v2.1: RSA Cryptography Standard (RSA Laboratories)
- Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS#1 (D.Bleichenbacher)
- Attacking RSA-based Session in SSL/TLS (V.Klima, O.Pokorný, T.Rosa)
- Remote timing attacks are practical (D.Brumbley, D.Boneh)
- Security Flaws Induced by CBC Padding Applications to SSL, IPSEC, WTLS... (S.Vaudenay)
- Password Interception in a SSL/TLS Channel (B.Canvel)