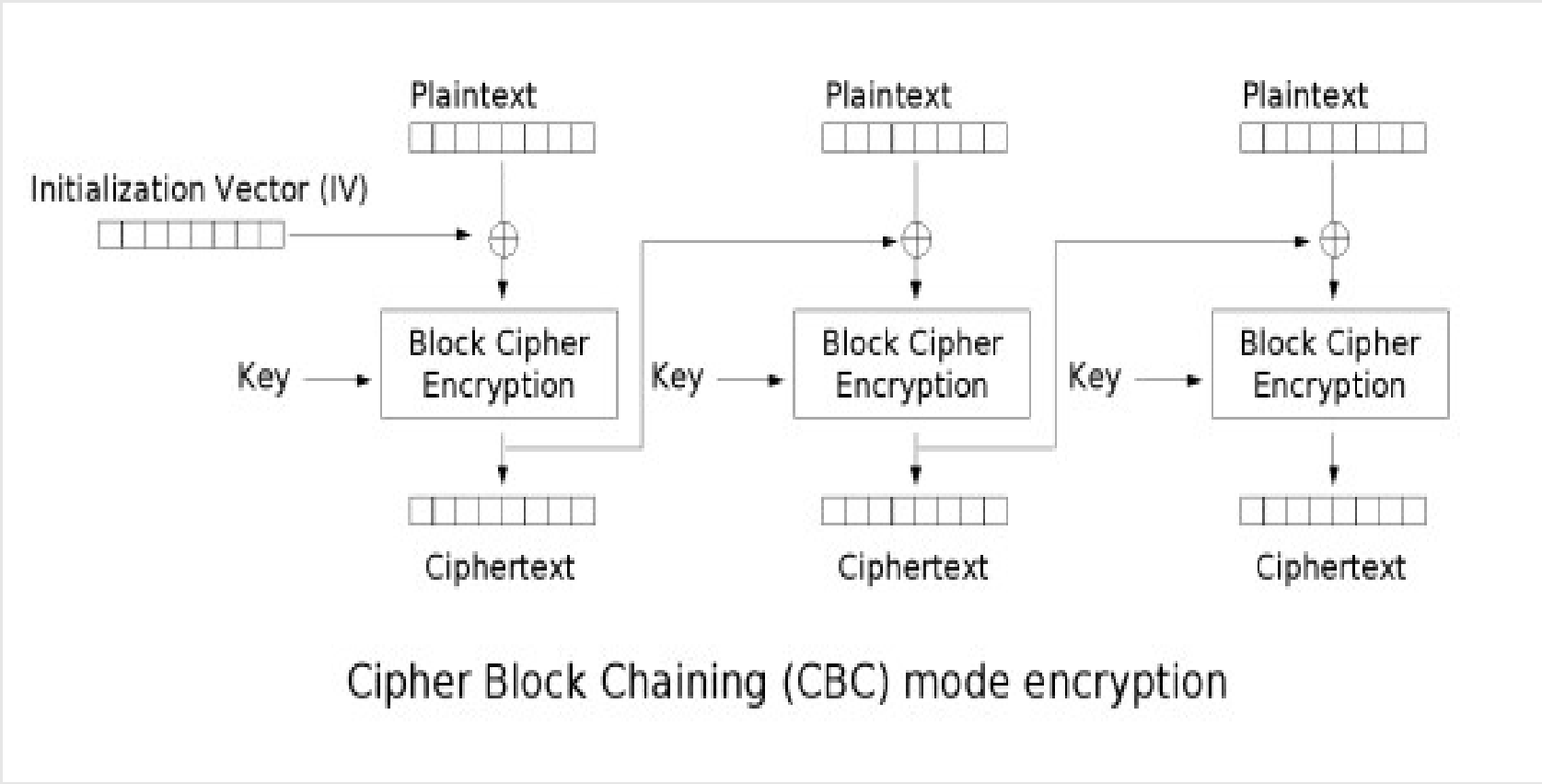
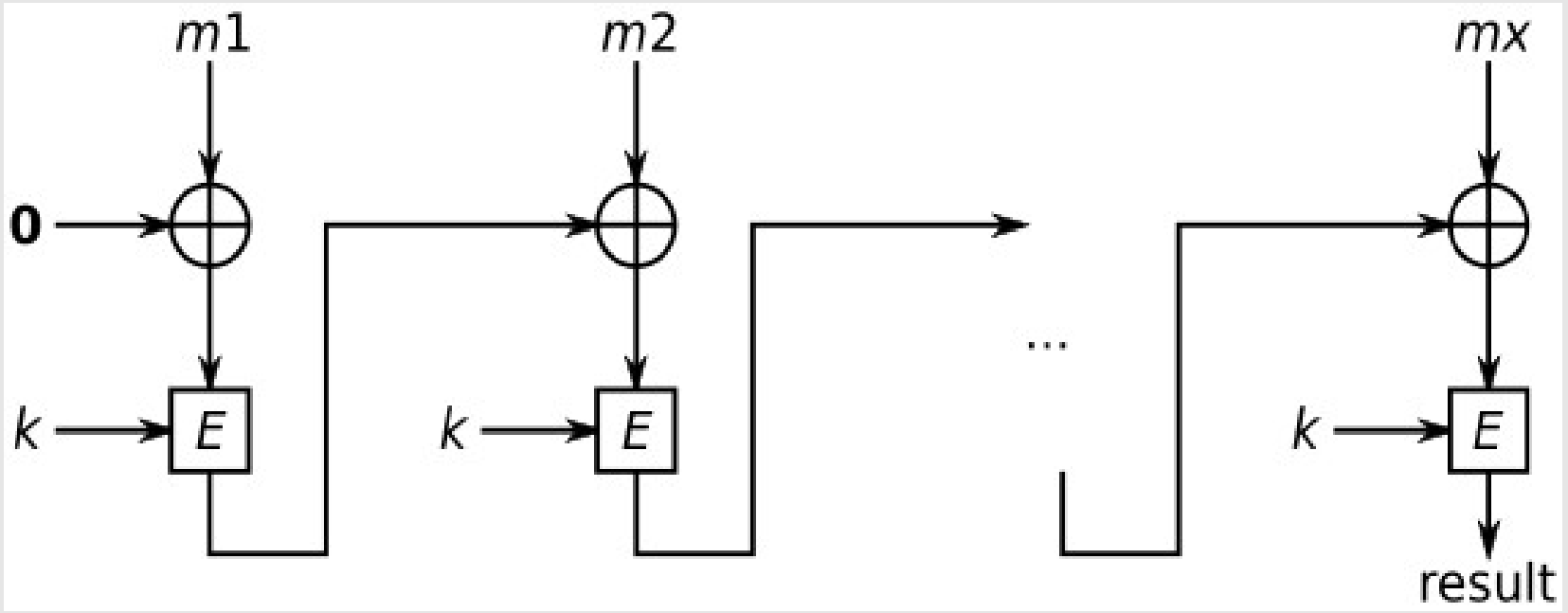
Crittografia avanzata
Lezione del 16 Maggio 2011

CBC



Cipher Block Chaining (CBC) mode encryption

CBC-MAC



Perché CBC-MAC?

- Abbiamo a disposizione un'implementazione certificata ed efficiente di un algoritmo a blocchi (e CBC), perché non usarla?
  - Implementazione hardware
  - Certificazione (Es. FIPS

Riuso della chiave

- È sempre una cattiva idea riutilizzare una chiave
- Nel caso di CBC-MAC, se la chiave di cifratura è riutilizzata per il MAC dello stesso messaggio, abbiamo
- $M = m_0 \,||\, m_1 \,||\, \ldots \,||\, m_n$
- $C = c_0 \,||\, c_1 \,||\, \ldots \,||\, c_n$ con $c_i = E_k(m_i \oplus c_{i-1})$, il primo blocco è l'$IV_c$
- $H = M_k (IV_c \,||\, m_0 \,||\, m_1 \,||\, \ldots \,||\, m_n)$ con IV=0
  - In pratica, il CBC-MAC è l'ultimo blocco del cipherstream, in cui si è usato un diverso IV
    - L'IV non dipende dal testo in chiaro, quindi l'uguaglianza vale anche se si modificano bit del testo in chiaro/cifrato, tranne che nell'ultimo blocco

Modifica ad un blocco

- $M = (m_1 \,||\, m_2 \,||\, \ldots \,||\, m_m \,||\, m_{m+1} \,||\, \ldots \,||\, m_n)$
- Modifichiamo c_m in c_m' es. flippando un bit
- Sia per la cifratura che per il MAC-CBC, fino al blocco m-esimo non cambia niente
- $Cm = E_k(c_{m-1} \oplus D_k(c_m'))$
  - La modifica si può “attribuire” al testo
  - Il testo in chiaro risulta modificato alla decifratura (non solo un bit)
- Lo stesso nel calcolo del CBC-MAC

CBC-MAC e lunghezza variabile

- È possibile appendere una stringa al ciphertext e ottenere lo stesso MAC
  - CBC-MAC è in generale sicuro solo per messaggi di lunghezza fissa
  - Sono state studiate delle varianti
    - CMAC NIST SB 800-38B del 2005
- H(a) = H(b)
- m è un blocco aggiuntivo

$$H(a \,||\, m) = Ek(H(a) \oplus m) = Ek(H(b) \oplus m) = H(b \,||\, m)$$

Tutta colpa dell'XOR?

- Se dobbiamo operare su due bit (es. testo e chiave) e ottenere un risultato che:
  - Dia lo stesso peso ai valori dei bit del testo
  - Dia lo stesso peso ai valori dei bit della chiave
  - Non sia indifferente al testo
  - Non sia indifferente alla chiave
  - Rimangono solo XOR e la sua negazione
    - Oppure operazioni su più bit

Attacchi a SSL/TLS

Version rollback

- Possibile se viene abilitato il supporto a vecchie versioni
  - Può essere necessario solo in contesti “legacy”
  - O se si supportano “vecchi browser”
  - Nota: solo se si passa a SSL 2.0, mancando la protezione
- Può portare all'uso di cifrari obsoleti (es. DES)
  - Quanto rappresentano un rischio in un contesto reale?
- Nota: TLS 1.0 è ancora la versione di TLS più utilizzata, la 1.2 ha poco utilizzo:
  - es. Firefox e Chrome supportano la 1.0
  - Apache solo con un apposito modulo
  - IE9, Opera e IIS sì, ma di default è disabilitato

RSA PKCS#1 1.5

- Presentato da Bleichenbacher nel 1998
- PKCS#1: implementazione di RSA
  - ripubblicata come 2.1 nell'RFC 3447 nel 2003 (dopo l'attacco di Bleichenbach e altri)
- 1.5 attaccata ba Bleichenbacher
- 2.0 attaccata da
- 2.1 attuale

Parametri

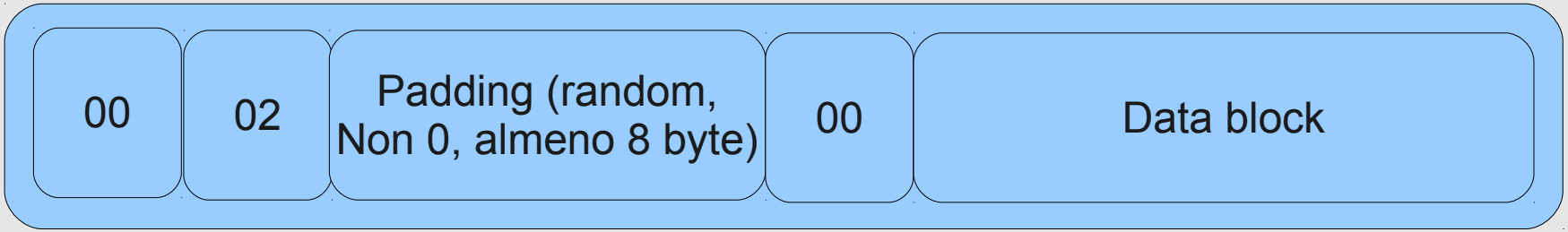
- n,e,d,p,q sono i soliti di RSA
  - n=pq
  - n,e è la chiave pubblica
  - d è quella privata
  - $c=m^e \pmod n$
  - $m=c^d \pmod n$
  - k è la lunghezza in byte di n: $2^{8(k-1)} \leq n < 2^{8k}$
  -

Chosen cyphertext e RSA

- RSA è vulnerabile ad un attacco di chosen cyphertext
  - Per far decifrare un testo c ed avere $m=c^d$ conoscendo la chiave pubblica e,n basta far cifrare, per un intero s:
  - $c'=s^e c \pmod n$
  - Infatti $c'^d=s^{ed}c^d \pmod n = ms \pmod n$ e $m=c'^d s^{-1} \pmod n$

PKCS#1 1.5

- Si attacca il blocco di tipo 2 (per la cifratura)
- I primi due campi sono costanti
- La struttura del blocco che viene convertito in intero e cifrato (elevato a potenza) è:



Conformità

- Un blocco cifrato è conforme se la sua decifratura porta ad una struttura che rispetta il formato definito
  - 00
  - 02
  - Nessuno 0 fino al 10 byte
  - Almeno uno 0 dal byte 11 in poi

Altro risultato utile

- La sicurezza di RSA è uguale alla sicurezza di un qualsiasi bit cifrato
  - Ovvero: se siamo in grado di calcolare un qualsiasi bit del testo in chiaro, dati il testo cifrato e la chiave pubblica, allora siamo in grado di calcolare l'intero testo in chiaro
  - Quindi, per ogni chosen ciphertext, ci basta riuscire a scoprire un bit del testo in chiaro

Oracolo

- Usiamo un server che ci risponda diversamente se il testo decifrato è conforme o meno
  - es. diversi messaggi di errore
  - Vogliamo farlo in un numero “ragionevole” di tentativi
- Usiamo un attacco di chosen ciphertext in cui verifichiamo se i primi due byte del plaintext sono 00 e 02
  - L'oracolo ce lo dice

L'attacco

- Vogliamo scoprire $m=c^d$, dati c,n,e
- Facciamo dei tentativi mandando all'oracolo $cs^e \pmod n$ per diversi s piccoli
- Dato che $2^{8(k-1)} \leq n < 2^{8k}$, anche $2^{8(k-1)} \leq ms \pmod n < 2^{8k}$
- Per semplicità, $2B \leq ms \pmod n < 3B$

L'attacco

- Lo vediamo direttamente sul paper
  - Si itera un algoritmo che definisce di volta in volta intervalli sempre più stretti fra 2B e 3B-1, fino a quando si trova il valore corretto
  - La parte difficile è la costruzione degli intervalli
    - O meglio, la dimostrazione della correttezza del meccanismo
- Punta ad ottenere il segreto condiviso in TLS, che viene inviato al server cifrato con la sua chiave pubblica
  - Prevede circa un milione di query per una chaive RSA di 1024 bit

PKCS#1 2.0

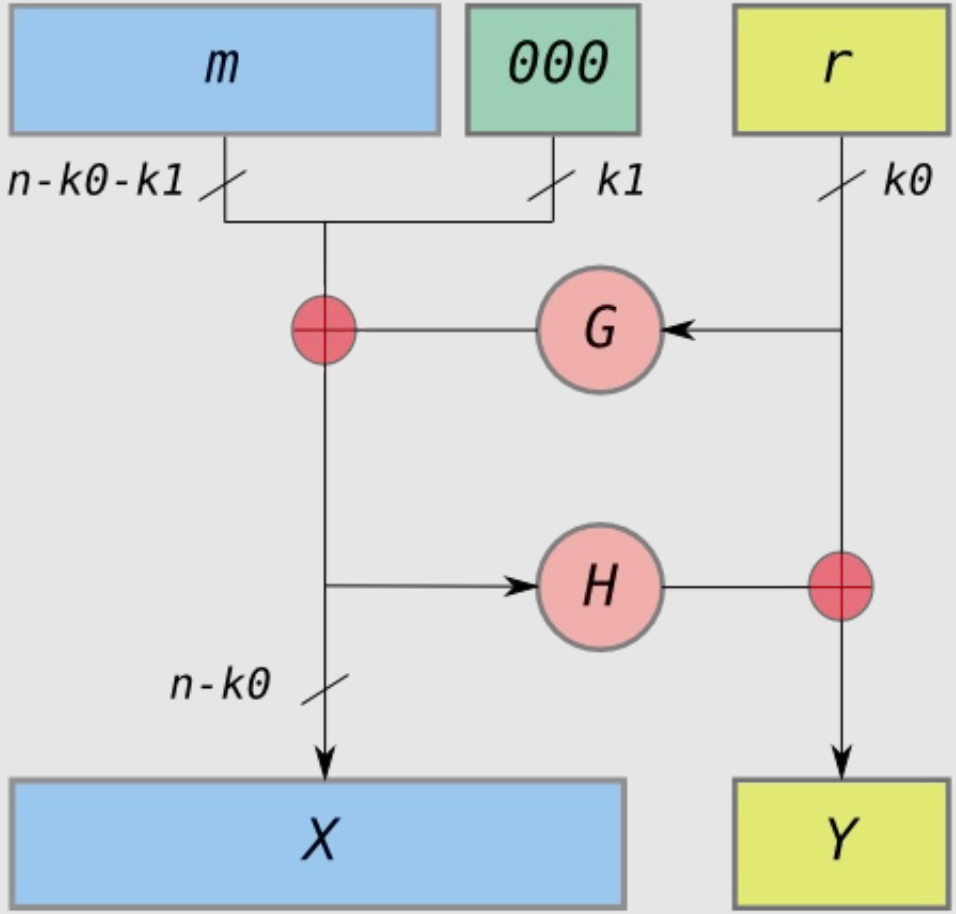
- In seguito all'attacco, nel 1998 viene definito PKCS#1 2.0 definito nell'RFC 2437
  - Rende obsoleto l'RFC 2313 che definiva PKCS#1 1.5
  - Usa “Optimal Asymmetric Encryption Padding (OAEP)” per contrastare l'attacco
- Anche questa versione risulta vulnerabile, sotto certe condizioni
  - Dipendenti dall'implementazione
    - Attenzione: questo vuole dire che non è safe, non che è colpa dell'implementazione: se ci sono vincoli implementativi, devono essere chiariti nell'RFC
  - Con log2 n query, es. circa 1000 per RSA 1024 bit

OAEP

- Optimal asymmetric encryption padding
- Perché il padding?
  - Per evitare inizi/fini prevedibili di messaggi
    - Facilitano la criptoanalisi
  - Per evitare che messaggi abbiano lunghezze prevedibili
    - O per allineare, nel caso semplice e più comune
- Nel caso di RSAES-OAEP, lo scopo è specificamente evitare la manipolazione del plaintext

Schema

* n is the number of bits in the RSA modulus.
* k0 and k1 are integers fixed by the protocol.
* m is the plaintext message, an (n − k0 − k1)-bit string
* G and H are typically some cryptographic hash functions fixed by the protocol.
* r is a random k0-bit string



- To encode
  - messages are padded with k1 zeros to be n − k0 bits in length.
  - G expands the k0 bits of r to n − k0 bits.
  - 4. X = m00..0 ⊕ G(r)
  - 5. H reduces the n − k0 bits of X to k0 bits.
  - 6. Y = r ⊕ H(X)
  - 7. The output is X || Y where X is shown in the diagram as the leftmost block and Y as the rightmost block.

- To decode,
  - recover the random string as r = Y ⊕ H(X)
  - recover the message as m00..0 = X ⊕ G(r)
- La presenza delle funzioni hash garantisce la proprietà “all-or-nothing”

Attacco

- Procedimento simile all'attacco precedente
  - Adaptively chosen ciphertext
  - Non lo studiamo
- Ha portato

SSL timing attack

- Attacco sviluppato nel 2003 contro OpenSSL
  - Come implementazione tipo
- Timing attacks: usati ad es. contro le smart card, dove la misurazione dei tempi è precisa
- Applicabile in rete, con i tempi “coperti” da tanti altri fattori?
- Si riteneva che determinate implementazioni di RSA non fossero vulnerabili
- Verificato in rete locale
- Già allora gli autori suggerivano la criticità di ambienti condivisi, in cui i “disturbi” sono minimi
- Viene suggerito l'uso anche contro il TC/Sealed Storage
  - Ma qui si usano chip analoghi a smart card...?

Blinding

- Un agente fornisce una funzione/servizio senza “conoscere” il vero input e il vero output
  - es. per proteggere dai timing attacks, una funzione di cifratura restituisce il risultato dopo un tempo fisso, maggiore del tempo massimo necessario per cifrare
  - Usate per proteggere dai side-channel attacks
    - Timing
    - Consumo di corrente
    - Emissioni elettromagnetiche
    - ...

L'attacco

- Nella fase di autenticazione vengono inviate diverse chiavi cifrate con la chiave pubblica
- Si valuta la differenza dei tempi nella risposta del server
  - Il CLIENT- KEY- EXCHANGE è volutamente mal formattato, in modo da avere un errore appena la decifratura è finita
  - Determinata in buona parte dall'attività di decifratura, in condizioni ottimali
  - Sono fatti numerosi tentativi (alcune ore) per ottenere dei risultati statisticamente validi

TLS renegotiation attack

- Sviluppato nel 2009
- Permette di iniettare del testo in una sessione
  - Un effettivo MitM

L'attacco

- L'attaccante “cattura” la sessione del client a livello TCP all'handshake
- L'attaccante inizia una propria sessione TLS con il server
- L'attaccante avvia una rinegoziazione della propria sessione con il server
- L'attaccante fa da proxy sulla negoziazione del client, passandola sul proprio canale cifrato come rinegoziazione
- Un'apposita estensione è definita nell'RFC 5746

A cosa serve?

- L'attaccante non vede il traffico del client
  - Ma quel traffico è “appeso” al suo
  - Con http(s), questo vuole dire che può costruire la prima parte della query, e farla completare al client
- Attaccante:
  - GET /send?oggetto=film&indirizzo=casamia HTTP/1.1
  - X-Ignore-This:
    - Dopo X-Ignore-This: non mette il newline
- Client
  - GET /send?oggetto=libro&indirizzo=casasua HTTP/1.1
  - Cookie: victimscookie
- Risultato:
  - GET /send?oggetto=film&indirizzo=casamia HTTP/1.1
  - X-Ignore-This: GET /send?oggetto=libro&indirizzo=casasua HTTP/1.1
  - Cookie: victimscookie

STARTTLS

- Serve a iniziare una sessione TLS su una in chiaro già aperta
  - es. SMTP (Postfix era vulnerabile)
- Il problema: il buffering del testo (in chiaro o cifrato) è implementato “sotto” a TLS
  - Se nel buffer c'è qualcosa, ci rimane dopo il passaggio a TLS
  - Se qualcosa è stato iniettato prima del comando STARLTLS ed è ancora lì, viene preso per autenticato
    - es. STARTTLS\r\nRSET\r\n
- È un problema generale di meccanismi di questo genere (es. SASL), se non forzano un flush del buffer prima di passare alla parte autenticata

Il quizzettino della settimana

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it. [Brian W. Kernighan]

Lo stesso si può dire per crittografia e crittoanalisi

Algoritmo da analizzare

https://github.com/abhishekkr/aqikcipher