Formalismo crittografico

Cataldo Basile
< cataldo.basile @ polito.it >

Politecnico di Torino
Dip. Automatica e Informatica

Operazioni crittografiche (I)

- operazioni di base: encryption e decryption
  - E( ) / Encrypt( )
    - cifratura simmetrica con AES 128 CBC
      - E_AES_128_CBC( )
      - E_AES128CBC()
  - D( ) / Decrypt( )
    - D_AES_128_CBC( ) / D_AES128CBC( )
  - CONSIGLIO: come primo parametro usate la chiave, poi il payload
- indicare i dati data cifrare e decifrare
  - P = plaintext
  - C = ciphertext

Operazioni crittografiche (II)

- digest
  - h / H / Hash
- keyed digest
  - KD / MAC / HMAC
- operazioni Boolean (binarie o bit-a-bit)
  - XOR, AND, OR, NOT
- manipolazione di stringhe
  - concatenazione ||
    - oppure + (come in Python)
  - first_bytes / last_bytes
  - first_block / last_block
- comunicazione
  - send / receive

Digest per verifica integrità

- Alice invia a Bob P (plaintext) ed il suo digest (integrità) e Bob verifica che P non sia stato modificato
  - es. sceglie SHA256
    - più sicuro di SHA-1 (obsoleto), SHA512 è OK, SHA3 anche ma specificate la dimensione del digest
- Alice
  - h = SHA256(P)
- Alice → Bob
  - *send*(P, h)
    - algoritmo di hash implicito (informazione già scambiata con altri metodi)
- Bob
  - *receive*(P, h)
  - h' = SHA256(P)
  - if (h != h') then "Errore"; altrimenti "Messaggio integro"

Digest (algoritmo esplicito)

- Alice invia a Bob P (plaintext) ed il suo digest (integrità) e Bob verifica che P non sia stato modificato
  - sceglie SHA256
- Alice
  - h = SHA256(P || "SHA256")
    - digest calcolato anche sul nome dell'algoritmo
    - soluzione migliore
- Alice → Bob
  - *send*(P, "SHA256", h)
    - NOTA PRATICA (nella realtà): digest di tutto quello che inviate
- Bob
  - *receive*(P, "SHA256", h)
  - h' = SHA256(P || "SHA256")
  - if (h != h') then "Errore"; altrimenti "Messaggio integro"

Cifratura simmetrica con pre-shared secret

- Alice invia a Bob P e vuole garantire confidenzialità
- Alice ←→ Bob: k_{AB} (scambio sicuro)
  - scambio sicuro effettuato in precedenza
    - out of band, DH con certificati o autenticato
  - es., sceglie AES128, ritenuto sicuro (k_{AB} di 128 bit)
- Alice
  - IV = random(128 bit) // *AES block size*
  - C = E_AES128CBC(k_{AB}, P, IV)
- Alice→ Bob
  - send(C,IV, "AES128CBC")

Cifratura simmetrica con pre-shared secret (II)

- Bob
  - receive(C,IV, "AES128CBC")
  - P'= D_AES128CBC(k_{AB}, C, IV)

- problemi nella pratica?
  - come fa BOB a sapere che P' è quello che gli era stato inviato e che nessuno l'ha modificato?
  - come garantire che IV e nome dell'algoritmo siano protetti da manipolazioni?

Esercizio: cifratura simmetrica con pre-shared secret e integrità dei dati

- Alice invia a Bob P e vuole confidentialità and integrità
- fate le assunzioni necessarie e descrivete il protocollo…

Esercizio: Cifratura simmetrica con pre-shared secret e integrità dei dati (soluzione)

- Alice invia a Bob P e vuole confidentialità and integrità
- Alice ←→ Bob: kAB1 (scambio sicuro)
  - scambio sicuro effettuato in precedenza
    - out of band, DH con certificati o autenticato
  - es., sceglie AES128, ritenuto sicuro (kAB di 128 bit)
  - e sceglie HMAC_SHA256 (kAB2 64 byte)
- Alice
  - fissa IV = random(128 bit) // AES block size
  - calcola C = E_AES128CBC(kAB1, P, IV)
  - calcola mac = HMAC_SHA256(kAB2 , C || IV || "AES128CBC" || "HMAC_SHA256")
    - anche IV and algoritmo protetti da manipolazioni

Esercizio: Cifratura simmetrica con pre-shared secret e integrità dei dati (soluzione) (III)

- Alice→ Bob
  - *send*(C, IV, mac, "AES128CBC", "HMAC_SHA256")
- Bob
  - *receive*(C, IV, mac, "AES128CBC", "HMAC_SHA256")
  - mac' = HMAC_SHA256(kAB2 , C || IV || "AES128CBC" || "HMAC-SHA256")
    - if mac != mac' then ERR
      - e salta la decifratura
  - P'= D_AES128CBC(kAB1, C, IV)

Esercizio: cifratura simmetrica e scambio di chiavi con crittografia asimmetrica

- Alice desidera inviare dei dati confidenziali a Bob

- descrivete con un formalismo crittografico appropriato le assunzioni ed i passi necessari perché Alice riesca ad inviare dati confidenziali a Bob scambiando chiavi con crittografia asimmetrica

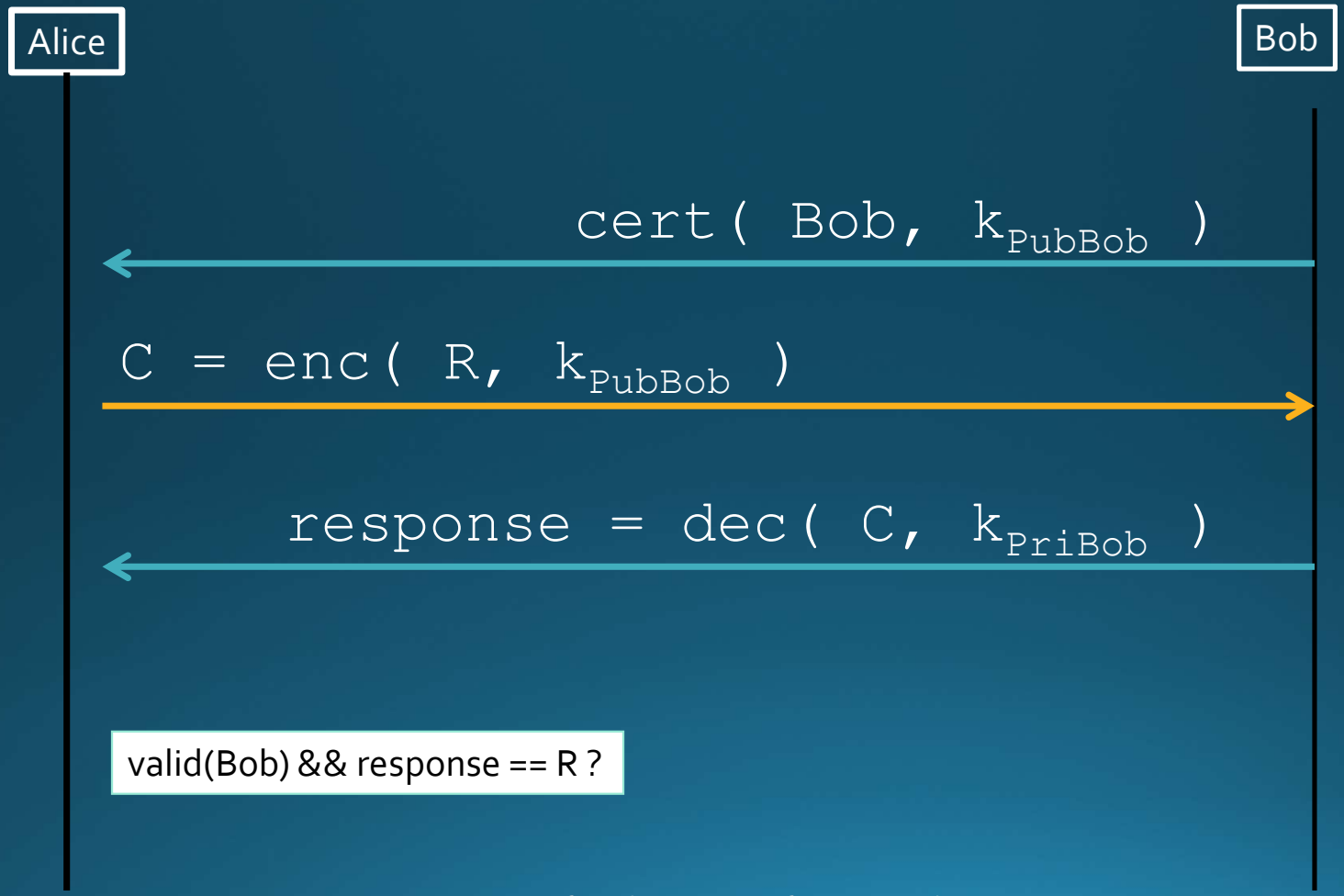
Esercizio: cifratura simmetrica e scambio di chiavi con crittografia asimmetrica (soluzione)

- Alice desidera inviare dei dati confidenziali a Bob
  - domanda: quanti? Tanti? → crittografia simmetrica (AES128)
    - …come scambiare una chiave di messaggio k_{AB}?
      - key encapsulation con RSA (minimo 2048 bit)
- assunzioni
  - Alice ha ottenuto una chiave pubblica di Bob, $k_{B,Pub}$ con metodi fidati
  - Bob possiede la componente privata $k_{B,Priv}$
- Alice
  - genera k_{AB} = random(128 bit) // *AES key size*
  - calcola $C_{rsa} = RSA(k_{BPub}, k_{AB})$
    - C_{rsa} è il payload cifrato
- Alice→ Bob
  - *send*(C_{rsa}) // *algoritmo implicito*

Esercizio: cifratura simmetrica e scambio di chiavi con crittografia asimmetrica (soluzione) (II)

- Bob
  - *receive* (C_{rsa})
  - calcola K_{AB} = RSA($k_{B,Priv}$, C_{rsa})
- Bob → Alice
  - send(ACK) // avverte che la chiave simmetrica è stata impostata
- Alice (come prima)
  - sceglie IV = random(128 bit) // AES block size
  - calcola C = E_AES128CBC(k_{AB}, M, IV)
  - …

Esercizio: autenticazione asimmetrica



Soluzione: autenticazione asimmetrica (I)

- Bob possiede
  - coppia di chiavi RSA (K_{priBob}, K_{pubBob})
  - certBob = cert(ID_{Bob}, K_{pubBob})
- Bob → Alice
  - send($cert_{Bob}$)
- Alice
  - ottiene $cert_{Bob}$ → receive($Cert_{Bob}$)
    [implicito: si fida della CA root nel CA path di Bob]
  - KpubBob = *extract*($cert_{Bob}$)
  - R = random(60B)
    - 60 byte = 480 bit, nessun hash è lungo 480 bit
      - altre lunghezze del random altrettanto buone
  - challenge=RSA(K_{pubBob}, R)

Soluzione: autenticazione asimmetrica (II)

- Alice → Bob
  - send(challenge)
- Bob
  - *receive(challenge)*
  - response=RSA(K_{priBob}, challenge)
- Bob→Alice
  - *send*(response)
- Alice
  - *receive(response)*
  - if (response == R) then OK; else FAIL;

Esercizio: autenticazione con HMAC (I)

- pag. 402 del testo di Menezes & Co. (http://cacr.uwaterloo.ca/hac/about/chap10.pdf)
- A, B sono le identità (Alice, Bob)
- r_A e r_B sono numeri random generati da A e B

$$
\begin{aligned}
A \leftarrow B: &\quad r_B && (1)\\
A \rightarrow B: &\quad r_A,\ h_K(r_A, r_B, B) && (2)\\
A \leftarrow B: &\quad h_K(r_B, r_A, A) && (3)
\end{aligned}
$$

Esercizio: autenticazione con HMAC (II)

- Alice ←→ Bob: $k_{A,B}$ (in modo sicuro)
  - scambio di chiavi simmetriche sicuro
    - precedente scambio off-line, DH con certificati
- Bob
  - r_B = random(60B) // *qualunque lunghezza != digest*
  - send(r_B)
- Alice
  - receive(r_B)
  - genera r_A = random(60B)
  - calcola mac_A=HMAC_SHA256(k_{AB}, r_A || r_B || "Bob")
  - send(r_A,mac_A)

Esercizio: autenticazione con HMAC (III)

- Bob
  - receive(r_A,mac_A)
  - calcola mac_A' = HMAC_SHA256(k_{AB}, r_A || r_B || "Bob")
    - if (mac_A' == mac_A) then "Alice autenticata";
      altrimenti "Autenticazione fallita"
  - calcola mac_B = HMAC_SHA256(k_{AB}, r_B || r_A || "Alice")
  - send(mac_B)

- Alice
  - receive(mac_B)
  - calcola mac_B' = HMAC-SHA256($k_{A,B}$, r_B || r_A || "Alice")
    - if (mac_B' == mac_B) then "Bob autenticato";
      else "Autenticazione fallita"

Esercizio

- Alice vuole inviare via Internet a Bob un file multimediale F rispettando i seguenti requisiti:
  - il file deve essere visto solo da Bob;
  - Alice non si fida nemmeno dei cloud provider;
  - il file non deve essere modificato da nessuno;
  - Alice può usare solo crittografia simmetrica (no hash)

- scrivere le formule crittografiche per lo scambio sicuro tra Alice e Bob
- calcolare inoltre quanti byte al massimo verranno trasmessi oltre al file F

Esercizio: analisi

- Alice vuole inviare via Internet a Bob un file multimediale F
  - il file deve essere visto solo da Bob;
    - segretezza = confidenzialità
  - Alice non si fida nemmeno dei cloud provider;
    - sicurezza di canale non è sufficiente → sicurezza di messaggio
  - il file non deve essere modificato da nessuno;
    - integrità → integrità e autenticazione dei dati
  - Alice può usare solo crittografia simmetrica
    - no PKI, no hash
  - formule crittografiche → bisogna usare formalismo
  - byte di overhead → usare bene formalismo per ottenere valori precisi

Esercizio: Soluzione (I)

- Alice ←→ Bob: $k_{a,b}$ (in modo sicuro)
  - scambio di chiavi simmetriche sicuro
- Alice:
  - C = E($k_{a,b}$, F)
  - mac = CBC-MAC($k_{a,b}$, F)
- Alice→ Bob
  - send(C,mac) oppure send(C|| mac)
- Bob:
  - F'=D($k_{a,b}$, C)
  - mac'=CBC-MAC($k_{a,b}$, F')
  - mac == mac'?
    true → OK/ false → NO
- trasferiti 128 bit del mac (+ 128 bit padding)

Esercizio: soluzione (II) con più dettagli

- Alice ←→ Bob: $k_{a,b}$ (in modo sicuro)
  - scambio di chiavi simmetriche sicuro
    - precedente scambio (es. OOB, DH)
- Alice:
  - scelgo AES128, 128 bit dà sicurezza sufficiente
    - AES192 e AES256 altrettanto buoni, 3DES meno (solo 112 bit e più lento)
  - $k_{a,b}$ deve essere lunga almeno 128 bit
  - IV = random(128 bit) *// lunghezza del blocco AES*
  - C = E_AES128CBC($k_{a,b}$, F, IV)
  - TAG = CBC-MAC_AES128($k_{a,b}$, F)
    - TAG = last_block(E_AES128CBC($k_{a,b}$, F, IV=0))

Esercizio 5: soluzione (II) con più dettagli

- Alice→ Bob
  - send(C,TAG,IV,"AES-128-CBC","AES-128-CBC-MAC")
- Bob:
  - F'= D-AES-128-CBC($k_{a,b}$, C, IV)
  - TAG'= E-AES-128-CBC-MAC($k_{a,b}$, F')
    - TAG←last_block(E-AES-128-CBC($k_{a,b}$, F', 0))
  - TAG==TAG'?

    true → OK/ false → NO
- trasferiti
  - 16 byte(TAG)+16 byte(IV)+
    +strlen("AES-128-CBC")+
    +strlen(" AES-128-CBC-MAC")+
    +128 bit padding?

Esercizio: soluzione (II) con più dettagli

- Alice→ Bob
  - send(C,TAG,IV,"AES-128-CBC","AES-128-CBC-MAC")
- Bob:
  - F'= D-AES-128-CBC($k_{a,b}$, C
  - TAG'= E-AES-128-CBC
    - TAG←last_block(E
  - TAG==TAG'?

    true → OK/ false →
- trasferiti
  - 16 byte(TAG)+16 byte(IV)+
    +strlen("AES-128-CBC")+
    +strlen(" AES-128-CBC-MAC")+
    +128 bit padding?

INSICURA

si calcola CBC-MAC e si cifrano gli stessi dati con la stessa chiave →
esiste un attacco che permette modifiche arbitrarie ed indistinguibili →
usare CMAC

https://cseweb.ucsd.edu/~mihir/papers/cbc.pdf

Esercizio: soluzione (III)

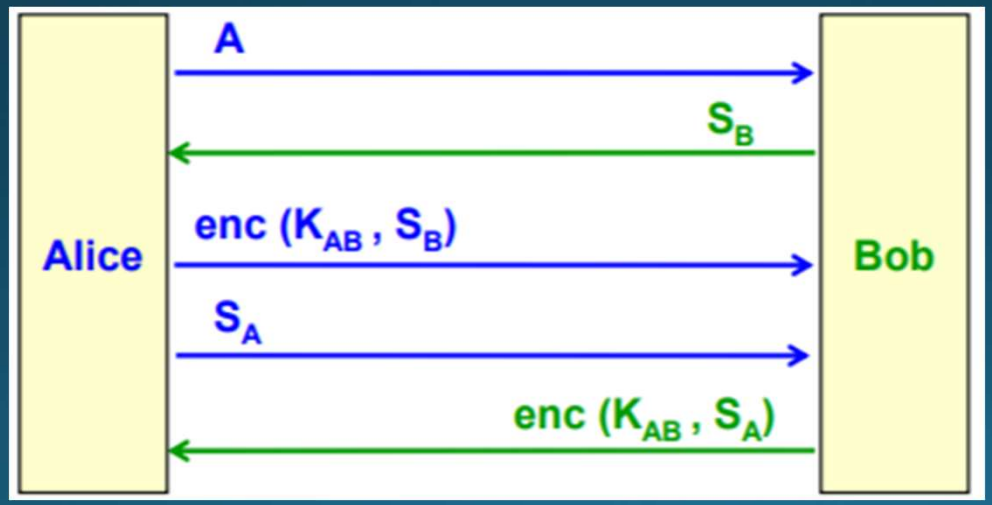
- Alice ←→ Bob: $k_{a,b}$ (in modo sicuro)
  - scambio di chiavi simmetriche sicuro (es. OOB, DH)
- Alice:
  - sceglie AES-128, 128 bit dà sicurezza sufficiente
  - k_1=random(128 bit), k_2=random(128 bit), k_3=random(128 bit)
  - IV=random(128 bit), IV_2=random(128 bit)
  - KEYS= E-AES-128-CBC($k_{a,b}$, $k_1 || k_2 || k_3$, IV)
  - C = E-AES-128-CBC(k_1, F, IV_2)
  - TAG = CMAC((k_2,k_3), F)
    - CMAC cifra l'ultimo blocco = CBC-MAC con chiave diversa
    - TAG = E-AES-128-CBC(k_3, last_block(E-AES-128-CBC(k_2, F,0)), 0)

Soluzione (III)

- Alice→ Bob
  - send(KEYS,C,TAG,IV,"AES-128-CBC","AES-128-CBC-MAC")
- Bob:
  - estrae le chiavi, ripete le stesse operazioni di prima con chiavi diverse e ECBC-MAC

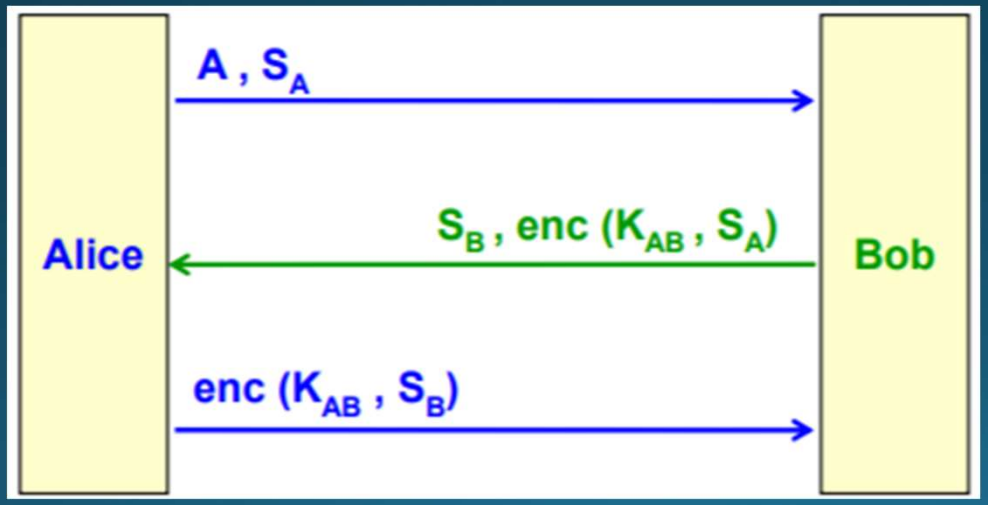
Altri esercizi (I)

Scrivere con formalismo crittografico il protocollo di mutua autenticazione simmetrica (v1).



Altri esercizi (II)

Scrivere con formalismo crittografico il protocollo di mutua autenticazione simmetrica (v2).



Altri esercizi (III)

Scrivere con formalismo crittografico i vari messaggi scambiati nel sistema Kerberos v4.