1 Protocolli di Autenticazione e Scambio Chiavi

1.1 Protocolli di Autenticazione

1.1.1 Autenticazione di Sistema vs. di Messaggi

È importante distinguere due concetti diversi di autenticazione:

- **Autenticazione di Messaggi:** Si occupa di verificare l'origine e l'integrità di un singolo messaggio. Gli strumenti usati sono i **MAC** (autenticazione + integrità) e le **firme digitali** (autenticazione + integrità + non ripudio). La verifica non deve necessariamente avvenire in tempo reale.
- **Autenticazione di Sistema (Entity Authentication):** È il processo di identificazione di un'entità (utente, sistema) che partecipa a un protocollo di rete, supportato da prove. Questa verifica deve avvenire **in real-time**.

I protocolli di autenticazione di sistema devono soddisfare diversi requisiti:

- **Autenticazione:** A deve poter verificare l'identità di B. Idealmente, si ha una **mutua autenticazione** (anche B autentica A).
- **Non trasferibilità:** A non deve poter riutilizzare i dati scambiati per impersonare B verso una terza entità C.
- **Robustezza:** Un'entità C non deve poter impersonare B, anche se C può osservare molti scambi tra A e B (resistenza al *replay attack*).

1.1.2 Basi per l'Autenticazione

Per autenticare B ad A, B deve fornire una prova basata su uno o più dei seguenti fattori:

1. **;**Qualcosa che B conosce: PIN, password, chiave simmetrica, chiave privata.
2. **Qualcosa che B possiede:** Smart-card, token (es. "calcolatrice" one-time-password).
3. **Qualcosa che B è:** Una caratteristica fisica (biometria).

1.1.3 Protocolli Challenge-Response (Autenticazione Forte)

L'autenticazione **forte** (challenge-response) si basa sulla conoscenza di un dato segreto s (es. una chiave crittografica, non una password). Il meccanismo base prevede che B (verifier) invii ad A (claimant) una **challenge** (sfida). A deve calcolare una **response** (risposta) applicando una funzione segreta f_s alla challenge. B verifica la risposta. Per prevenire attacchi di tipo *replay*, la challenge deve essere **tempo-variante**, utilizzando:

- **Nonce:** Un numero usato una sola volta (es. c_1, c_2).
- **Timestamp (t):** Un marcatore temporale (richiede orologi sincronizzati).
- **Sequence Number:** Un contatore (richiede gestione dello stato).

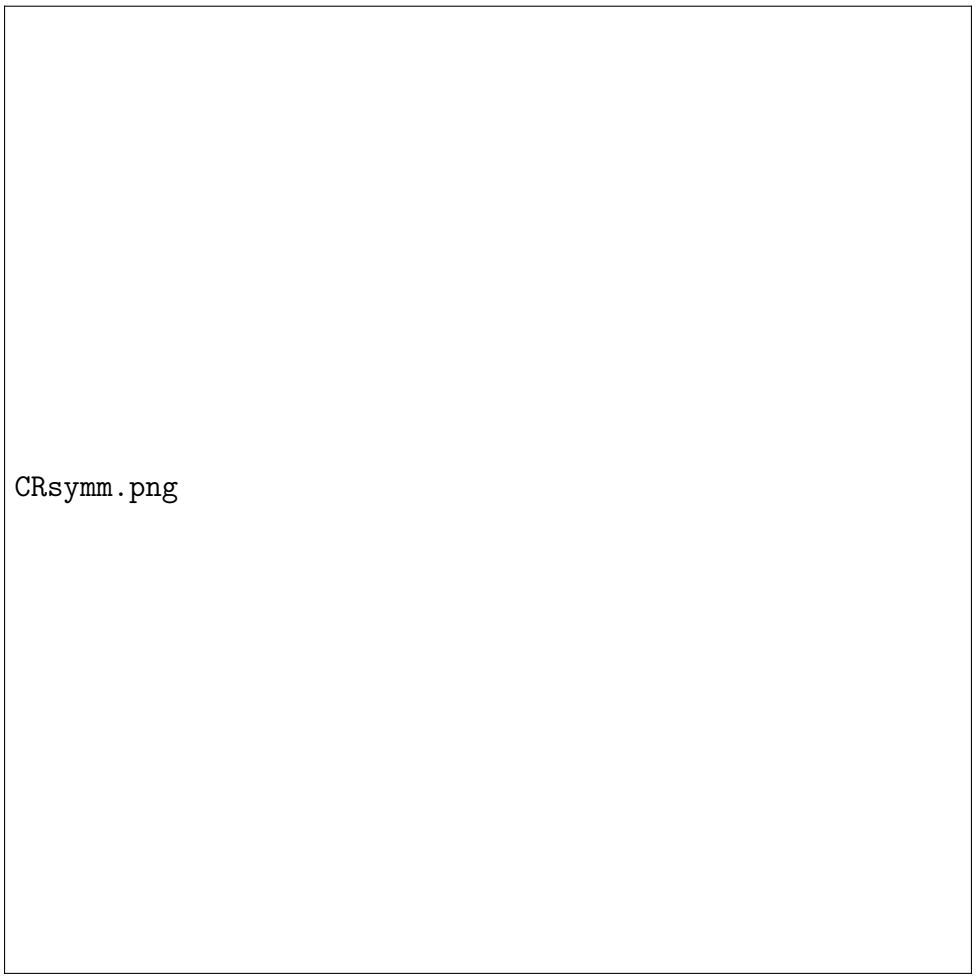

Figura 1: Schema protocollo C/R simmetrico (Pag 18)

Descrizione Immagine: Schema protocollo C/R simmetrico (Pag 18)

Questa immagine (Pag 18) mostra tre protocolli di autenticazione challenge-response tra "Alice" e "Bob" che condividono una chiave simmetrica segreta s.

- **Schema 1 (Timestamp, One-Way):**
  1. Alice invia a Bob: $A, R = E_s(t, B)$.
  2. E_s è la cifratura con la chiave segreta s. t è un timestamp, B è l'identificativo di Bob (incluso per prevenire che Bob riutilizzi R per impersonare Alice verso qualcun altro).

3. Bob riceve R, lo decifra con s ottenendo t' e B'. Controlla che $B' = B$ e che il timestamp t' sia "recente" (es. $t - x < t' < t + x$). Se i controlli passano, Alice è autenticata.

- **Schema 2 (Random Challenge, Mutua Autenticazione - Cifratura):**
  1. Bob invia ad Alice una challenge (nonce): c_1, B.
  2. Alice invia a Bob: $A, R_A = E_s(c_1, c_2, B)$. c_2 è la challenge di Alice per Bob.
  3. Bob decifra R_A con s, controlla che c_1 sia il nonce che ha inviato. Ora Bob ha autenticato Alice.
  4. Bob invia ad Alice: $R_B = E_s(c_2, c_1, A)$. (Notare l'ordine invertito c_2, c_1).
  5. Alice decifra R_B con s, controlla che c_2 sia il nonce che ha inviato. Ora Alice ha autenticato Bob.
- **Schema 3 (Random Challenge, Mutua Autenticazione - MAC):**
  - Identico allo Schema 2, ma invece di usare la cifratura E_s, usa un Message Authentication Code (MAC) calcolato con la chiave s.
  - $R_A = MAC_s(c_1, c_2, B)$
  - $R_B = MAC_s(c_2, c_1, A)$
  - Questo è più efficiente perché non richiede cifratura, ma solo il calcolo di un hash con chiave.

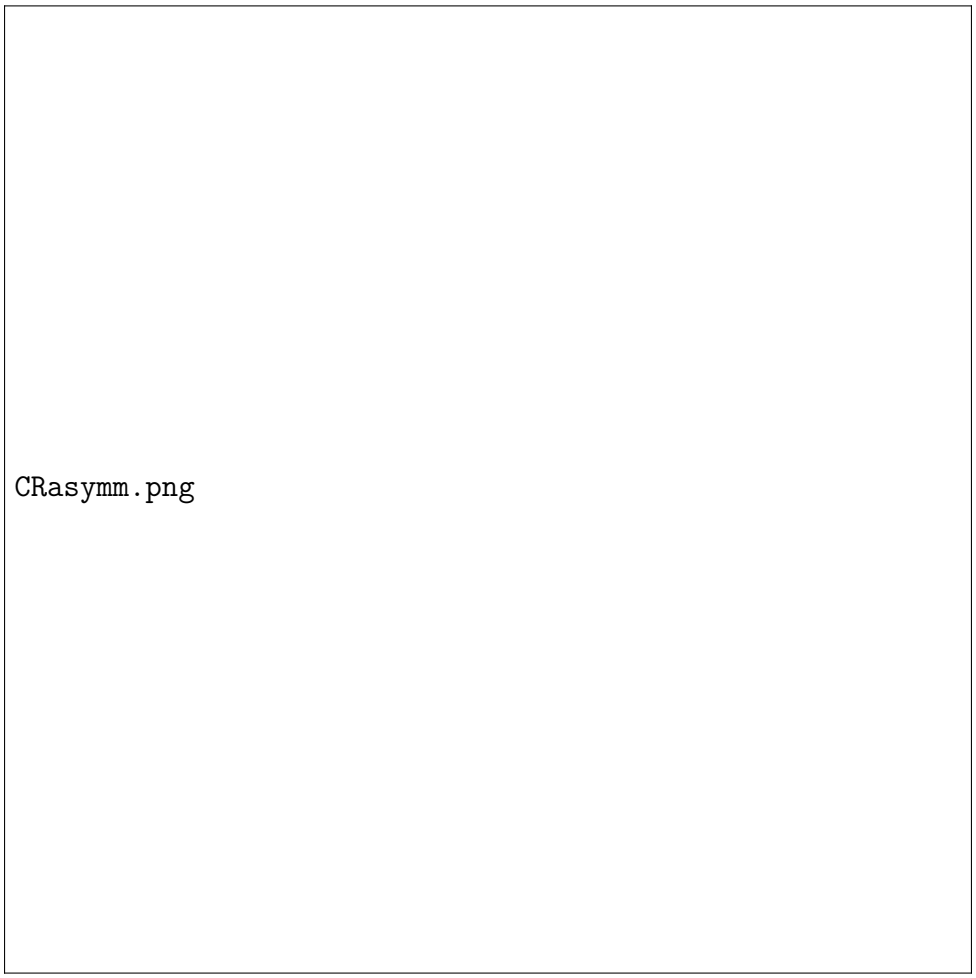

Figura 2: Schema protocollo C/R asimmetrico (Pag 19)

Descrizione Immagine: Schema protocollo C/R asimmetrico (Pag 19)

Questa immagine (Pag 19) mostra due protocolli analoghi ai precedenti, ma basati su crittografia asimmetrica (firme digitali) invece che su una chiave condivisa.

- **Schema 1 (Timestamp, One-Way):**
  1. Alice invia a Bob: $A, t, B, R = S_A(t, B)$.
  2. S_A è la **firma digitale** di Alice, creata usando la sua **chiave privata** sul timestamp t e sull'ID di Bob B.
  3. Bob riceve il messaggio, usa la **chiave pubblica** di Alice (che deve già possedere) per verificare la firma R.
  4. Se la firma è valida, Bob controlla la freschezza di t e che B sia il suo ID. Se sì, Alice è autenticata.
- **Schema 2 (Random Challenge, Mutua Autenticazione):**
  1. Bob invia ad Alice una challenge (nonce): c_1, B.

2. Alice invia a Bob: $A, B, c_2, R_A = S_A(c_1, c_2, B)$. R_A è la sua firma (privata) sulle due challenge e sull'ID di Bob.
3. Bob verifica R_A (con K_{pubA}), controlla c_1 (autenticando Alice).
4. Bob invia ad Alice: $A, R_B = S_B(c_2, c_1, A)$. R_B è la sua firma (privata) sulle challenge (ordine invertito) e sull'ID di Alice.
5. Alice verifica R_B (con K_{pubB}), controlla c_2 (autenticando Bob).

1.2 Instaurazione di Chiavi Effimere (di Sessione)

1.2.1 Perché usare Chiavi Effimere?

- Si limita la quantità di dati cifrati con la stessa chiave (utile contro attacchi *ciphertext-only*).
- **Limitazione dei danni:** Se una chiave effimera K_t viene compromessa, solo la sessione cifrata con essa è compromessa. La master key (a lungo termine) rimane sicura.
- Permette di gestire chiavi diverse per sessioni diverse.

L'architettura di riferimento prevede che ogni entità possegga una *master key* (a lungo termine) usata per generare o trasportare le chiavi effimere K_t.

1.2.2 Perfect Forward Secrecy (PFS)

La ;Perfect Forward Secrecy (PFS) è una proprietà ideale di un protocollo di scambio chiavi. ;Definizione: La compromissione della *master key* (a lungo termine) in un istante t **non** deve compromettere la segretezza delle chiavi effimere generate in passato (fino a $t-1$). Se un attaccante (Trudy) registra una sessione S_1 (cifrata con K_{t1}) e *successivamente* ruba la K_{master}, la PFS garantisce che Trudy non possa comunque usare K_{master} per risalire a K_{t1} e decifrare S_1.

1.2.3 Classi di Protocolli di Scambio Chiavi

K_t **Trasportata** Una entità (es. Alice) genera K_t e la invia all'altra (Bob), crittografandola con la master key.

- **Vantaggi:** Semplice ed efficiente.
- **Svantaggi: Non offre PFS**. Se Trudy ruba la master key, può decifrare il trasporto di K_t e recuperare tutte le chiavi di sessione passate.

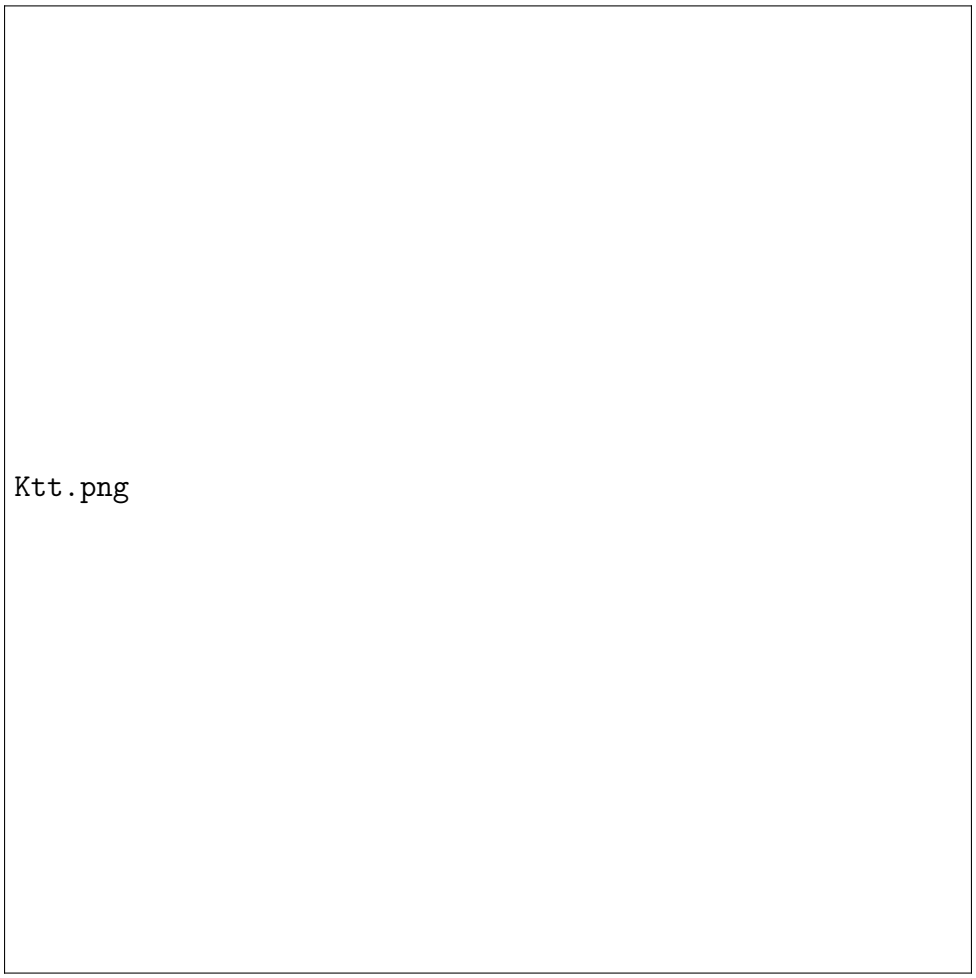

Figura 3: Schema protocollo trasporto K_t (Pag 22)

Descrizione Immagine: Schema protocollo trasporto K_t (Pag 22)

- **Contesto:** L'immagine (Pag 22) mostra tre modi in cui Alice può generare una chiave K_t e "trasportarla" (inviarla) a Bob.
- **Schema 1 (Simmetrico Semplice):**
  - Alice genera $K_t = random()$.
  - Alice invia a Bob: $E_K(K_t, B)$.
  - E_K è la cifratura con una *master key simmetrica* K condivisa tra Alice e Bob.
  - **Criticità:** Non offre PFS ed è vulnerabile a replay attack (Trudy può re-inviare questo messaggio a Bob).
- **Schema 2 (Simmetrico con Nonce):**
  - Bob invia un nonce (numero casuale) n ad Alice.
  - Alice genera $K_t = random()$.
  - Alice invia a Bob: $E_K(K_t, B, n)$.

  - Bob decifra e controlla che n sia quello che ha inviato, sconfiggendo il replay attack.
  - **Criticità:** Non offre ancora PFS.

- **Schema 3 (Asimmetrico/Ibrido):**
  - Alice genera $K_t = random()$.
  - Alice invia a Bob: $E_{kbob}(t, K_t, S_{kalice}(t, K_t, B))$.
  - Questo messaggio è composto:
    * $E_{kbob}(...)$: L'intero messaggio è cifrato con la *chiave pubblica di Bob*. Solo Bob può leggerlo.
    * t: Un timestamp per prevenire replay attack.
    * K_t: La chiave effimera.
    * $S_{kalice}(...)$: Una *firma digitale* (fatta con la *chiave privata di Alice*) su t, K_t e B, per autenticare Alice come mittente.
  - **Criticità:** Non offre PFS. Se la *chiave privata di Bob* (k_{priv_bob}) venisse rubata in futuro, Trudy potrebbe decifrare tutte le sessioni passate registrate, poiché K_t è trasportata (cifrata).

K_t **Derivata** Le entità si scambiano dati (es. numeri casuali) e *derivano* K_t crittograficamente, usando la master key nel processo.

- **Vantaggi: Può offrire PFS** (es. protocollo EKE, che è un DH autenticato).
- **Svantaggi:** Più complesso e meno efficiente in termini di messaggi scambiati.

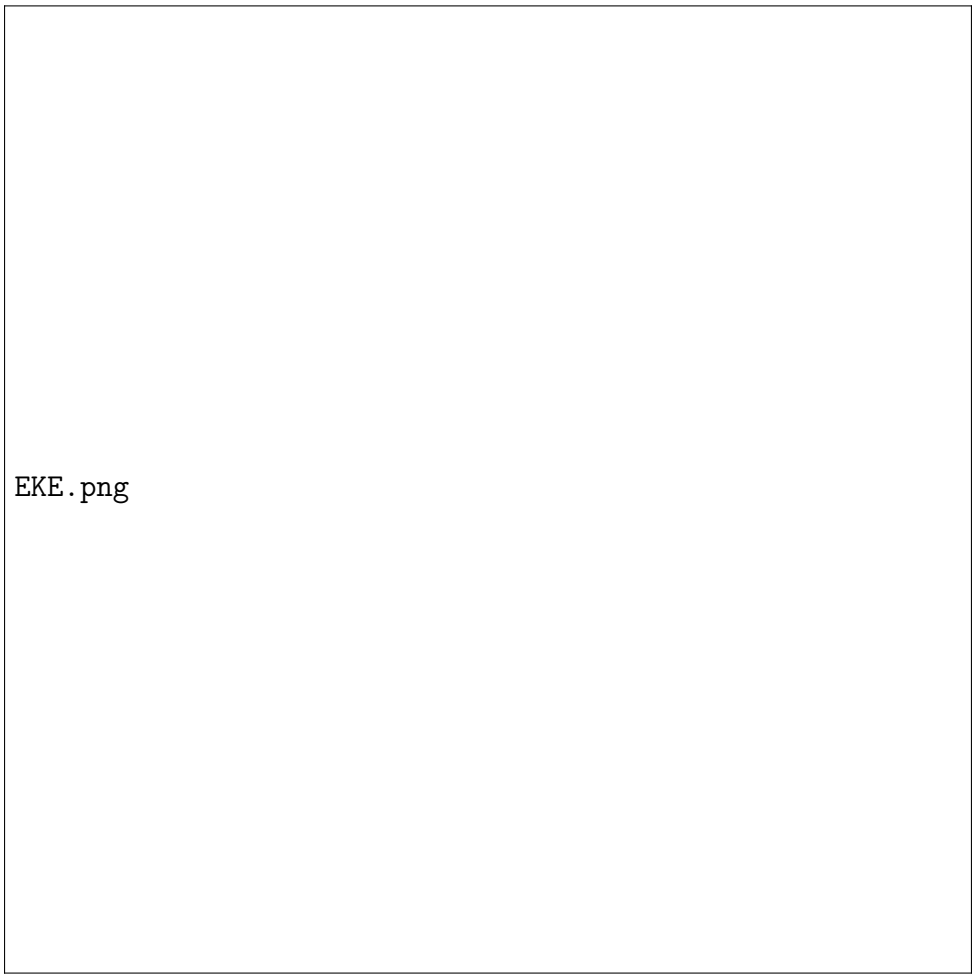

Figura 4: Autenticazione forte con password Encrypted Key Exchange (Pag 17)

Descrizione Immagine: Encrypted Key Exchange (EKE) (Pag 17)

- **Contesto:** L'immagine (Pag 17) mostra il protocollo EKE, che *deriva* una chiave di sessione K usando Diffie-Hellman (DH) e la autentica usando una password (pwd) condivisa. **Questo protocollo offre PFS**.
- **Flusso del Protocollo:**
  1. Alice e Bob calcolano entrambi indipendentemente una chiave temporanea $w = f(pwd)$ (es. hash della password).
  2. Alice sceglie un segreto DH sa. Calcola la sua parte pubblica DH ($g^{sa} \mod p$) e la *cifra* con w.
  3. Alice $\rightarrow$ Bob: $Alice, A = E_w(g^{sa} \mod p)$.
  4. Bob (che conosce w) decifra A per ottenere g^{sa}. Sceglie il suo segreto DH sb. Calcola la sua parte pubblica $g^{sb} \mod p$.
  5. Bob calcola la chiave di sessione finale $K = (g^{sa})^{sb} \mod p$.
  6. Bob $\rightarrow$ Alice: $Bob, B = E_w(g^{sb} \mod p, c_1)$ (invia la sua parte pubblica cifrata, insieme a una challenge c_1).

7. Alice decifra B, ottiene g^{sb} e c_1. Calcola anche lei la chiave finale $K = (g^{sb})^{sa} \mod p$.
8. Alice e Bob usano la chiave K (appena derivata) per scambiarsi le challenge (c_1, c_2) e completare la mutua autenticazione (passaggi $E_k(c1, c2)$ e $E_k(c2)$).

- **Proprietà (PFS):** La chiave finale K dipende dai segreti *sa* e *sb*. Questi segreti non vengono mai trasmessi, nemmeno cifrati. La password w protegge solo lo scambio delle *parti pubbliche* di DH (g^{sa}, g^{sb}). Se un attaccante ruba la password *pwd* in futuro, non può ricavare *sa* o *sb* (che sono esistiti solo in quella sessione) e quindi non può ricavare la chiave K passata.