Azzolini Riccardo 2021-04-26

Applicazioni della cifratura asimmetrica

1 Applicazioni

La cifratura asimmetrica ha due applicazioni standard:

- la **digital envelope** ("busta digitale"), che fornisce un servizio di segretezza;
- la **firma digitale**, **digital signature**, che fornisce un servizio di autenticazione (e integrità).

2 Digital Envelope

In teoria, per realizzare la segretezza sarebbe sufficiente applicare un algoritmo asimmetrico (come RSA) cifrando con la chiave pubblica del destinatario e decifrando con la sua chiave privata. Tuttavia, come già detto, le operazioni di cifratura e decifratura con algoritmi asimmetrici hanno un costo computazionale relativamente elevato, e su un messaggio grande sarebbe necessario eseguirle per ciascuno dei potenzialmente tanti blocchi in cui il messaggio dovrebbe essere suddiviso, dunque ciò non è pratico. Allora, la digital envelope utilizza la cifratura simmetrica (più efficiente) per cifrare i messaggi e la cifratura asimmetrica per risolvere il problema della gestione della chiave segreta:

1. Il mittente genera casualmente una chiave segreta k, detta **chiave di sessione** (**session key**).
2. Il mittente cifra il messaggio M che vuole inviare con un algoritmo simmetrico, usando la chiave k:
$$C = E_k(M)$$
3. Il mittente cifra la chiave segreta con la chiave pubblica KP del destinatario, ottenendo la cifratura $E_{KP}(k)$, che prende il nome di **digital envelope**.
4. Il mittente invia al destinatario il messaggio cifrato C e la digital envelope $E_{KP}(k)$, cioè complessivamente invia $(C, E_{KP}(k))$.
5. Il destinatario decifra la digital envelope con la propria chiave privata KR, ottenendo così la chiave segreta:
$$D_{KR}(E_{KP}(k)) = k$$

6. Il destinatario decifra il ciphertext C con la chiave segreta k, ottenendo così il messaggio in chiaro M:

$$D_k(C) = D_k(E_k(M)) = M$$

Uno svantaggio di questo schema è che, se ci sono errori di trasmissione o se il mittente ha usato la chiave pubblica errata, il destinatario se ne accorge solo dopo aver effettuato entrambe le operazioni di decifratura (i passi 5 e 6), quando il messaggio decifrato M non risulta corretto, leggibile; non se ne può accorgere dopo aver decifrato solo la digital envelope (al passo 5) perché la chiave k è un numero del tutto casuale, non ha proprietà che permettono di determinare se sia corretta senza provare a usarla per decifrare C.

3 Firma digitale

Cifrare con la chiave privata del mettente e decifrare con la sua chiave pubblica fornisce, come detto in precedenza, una prova di autenticità del documento: il destinatario può verificare la provenienza e l'integrità del documento, cioè che esso sia stato generato/approvato dal mittente e che non sia stato successivamente modificato (se fosse cambiato anche solo 1 bit, dalla decifratura non si otterrebbe un messaggio leggibile). Tuttavia, come nel caso della segretezza, cifrare e decifrare l'intero messaggio con un algoritmo asimmetrico sarebbe troppo costoso. La soluzione è generare un "riassunto", un'**impronta** del messaggio originale, chiamata **message digest**, e cifrare soltanto quella, ottenendo così una firma indipendente dal messaggio, che può essere accodata a esso o trasmessa separatamente. Con questo schema il messaggio rimane completamente in chiaro, leggibile da tutti, ma questo non è un problema perché l'obiettivo della firma digitale è fornire autenticazione, non segretezza, e anche l'approccio di cifrare l'intero messaggio non fornirebbe segretezza, dato che la chiave necessaria per decifrare è pubblica.

3.1 Funzioni hash

Un message digest è un messaggio molto più piccolo dell'originale, ottenuto dal messaggio originale tramite una funzione one-way. Il tipo di funzioni one-way più usato a tale scopo sono le funzioni hash.

Una **funzione hash** è una funzione H che mappa una stringa di bit di lunghezza arbitraria a una di lunghezza fissa (ad esempio 256 o 512 bit). Per l'uso in ambito crittografico, una funzione hash deve essere:

- **one-way**: conoscendo y è computazionalmente[1] impossibile trovare un x tale che $y = H(x)$;
- **resistente alle collisioni**: è computazionalmente[1] impossibile trovare una collisione, cioè una coppia di x e x' tali che $H(x) = H(x')$ (siccome una funzione hash accetta input di dimensione arbitraria e produce output di dimensione fissa esistono inevitabilmente delle collisioni, poiché il dominio è infinito mentre il codominio è finito, dunque la funzione non può essere iniettiva, ma non deve esistere un modo facile di trovare tali collisioni).

Soddisfare queste proprietà non è facile, ma nel corso degli anni sono state standardizzate varie funzioni hash: alcune delle principali sono MD5, SHA-1, SHA-2 e SHA-3 (ma MD5 e SHA-1 sono considerate non più sicure).

3.2 Generazione e verifica della firma

Per generare la firma per un messaggio M, il mittente calcola l'impronta hash $H(M)$ del messaggio e la cifra con la propria chiave privata KR: la firma digitale è dunque $FD = E_{KR}(H(M))$. Il messaggio e la firma vengono poi inviati al destinatario; tipicamente, la firma viene inviata aggiungendola in coda al messaggio originale, ma potrebbe anche essere trasmessa separatamente.

Quando il destinatario riceve il messaggio firmato, per verificare la firma calcola anch'esso l'impronta hash $H(M)$ del messaggio, decifra la firma con la chiave pubblica KP del mittente per ottenere l'impronta generata da quest'ultimo, e controlla che le due impronte siano uguali:

$$H(M) = D_{KP}(FD)$$

Se lo sono, il destinatario ha prova

- dell'*autenticità* del messaggio: solo il mittente possiede la chiave privata usata per generare la firma, e se invece fosse stata usata un'altra chiave privata la decifratura $D_{KP}(FD)$ non sarebbe andata a buon fine, producendo un risultato diverso dall'impronta $H(M)$;
- dell'*integrità* del messaggio: se il messaggio fosse stato modificato dopo la generazione della firma, intenzionalmente o a causa di errori di trasmissione, l'impronta hash calcolata dal destinatario sarebbe cambiata rispetto a quella generata dal mittente prima della modifica (se la funzione hash è resistente alle collisioni, modificando anche solo 1 bit del messaggio si ottiene un digest completamente diverso). Inoltre, un attaccante che modifica un messaggio non può modificare anche la firma per "correggerla":

[1] Matematicamente ciò è possibile, basta provare a forza bruta tutti i possibili input della funzione, ma per una funzione hash "ben fatta" un attacco a forza bruta richiederebbe troppo tempo (come minimo, più del tempo di validità dei dati protetti usando tale funzione hash).

– non può sostituirla con una nuova firma corretta perché non ha la chiave privata del mittente (se usasse una chiave diversa per la cifratura, il destinatario si accorgerebbe che il messaggio non è autentico);

– non può modificare direttamente la firma cifrata, perché un algoritmo di cifratura (in questo caso asimmetrico) sicuro non consente di modificare in modo prevedibile un testo in chiaro (qui l'impronta hash) tramite modifiche effettuate sul testo cifrato.

Dopo aver verificata la firma, il destinatario potrebbe scegliere di salvarla per poter verificare nuovamente l'autenticità e integrità del messaggio al momento dell'uso dei suoi contenuti. In questo caso, può essere utile salvare la forma separatamente dai contenuti del messaggio, in modo da non doverla separare nuovamente ogni volta che usa il messaggio.

4 Codifica di dati cifrati/firmati

Esistono degli standard che determinano come codificare la digital envelope e la firma digitale. Gli standard più rilevanti sono i **PKCS**, **Public Key Cryptography Standards** (definiti da RSA Labs). In particolare, lo standard **PKCS#7**, "Cryptographic Message Syntax" (standardizzato anche da IETF come RFC 2315) definisce il formato di:

- signed data (dati firmati);
- enveloped data (dati cifrati con digital envelope);
- signed-and-enveloped data (dati firmati e poi cifrati con digital envelope);
- digested data (dati con solo un message digest per l'integrità, senza una vera e propria firma per l'autenticazione);
- encrypted data (dati cifrati con una qualche chiave gestita separatamente).

I file contenenti dati in formato PKCS#7 hanno tipicamente l'estensione `.p7m`.

Altri standard sono quelli proposti dal W3C, che usano XML per la codifica delle informazioni di sicurezza:

- **XML Encryption** per la cifratura (segretezza);
- **XML Signature** per la firma digitale.

4.1 XML Signature

Lo standard XML Signature è interessante in quanto fornisce una maggiore flessibilità rispetto alla firma digitale "tradizionale", rendendola più adatta a gestire i flussi di informazione molto dinamici che caratterizzano il web (in particolare le architetture service-oriented, SOA, nelle quali si possono comporre più servizi, possono esserci più entità coinvolte, ecc.). A tale scopo, esso definisce:

- il *processo* di generazione e validazione della firma digitale;
- il *formato* XML per codificare la firma digitale.

La firma può essere applicata a vari tipi di risorse:

- un intero documento XML;
- una porzione di un documento XML (cioè alcuni elementi e attributi), che può anche essere specificata come il risultato di una query/trasformazione XPath o XSLT;
- qualsiasi risorsa identificata da un URI (una pagina web, un'immagine, ecc.).

Inoltre, con un'unica firma si possono firmare contemporaneamente più risorse (ad esempio più porzioni di documenti XML anche diversi).

Il processo di generazione è il seguente:

1. si calcolano separatamente i digest di ciascuna risorsa da firmare;
2. i digest ottenuti sono inseriti in un unico elemento XML `<Signature>`, insieme a tutte le altre informazioni necessarie per verificare la firma (qual è la chiave pubblica da usare per verificare la firma, quale algoritmo è stato usato per generare la firma, ecc.);
3. si genera la firma di tutto l'elemento `<Signature>` (calcolandone il digest e cifrandolo con una chiave privata), e questa viene inserita in un sotto-elemento di `<Signature>`.

Lo standard XML Signature supporta tre formati, in particolare tre modi di usare l'elemento `<Signature>`:

- **enveloped**: l'elemento `<Signature>` è inserito all'interno del documento XML su cui è stata generata la firma;
- **enveloping**: l'elemento `<Signature>` ha un sotto-elemento `<Object>` che contiene il documento XML su cui è stata generata la firma;
- **detached**: l'elemento `<Signature>` è un documento XML indipendente, al cui interno è inserito il riferimento (URI) alla risorsa firmata.

L'elemento `<Signature>` ha la struttura[2]

```
<Signature ID?>
  <SignedInfo>
    <CanonicalizationMethod>
    <SignatureMethod>
    (<Reference URI?>
      (<Transforms>)?
      <DigestMethod>
      <DigestValue>
    </Reference>)+
  </SignedInfo>
  <SignatureValue>
  (<KeyInfo>)?
  (<Object ID?>)*
</Signature>
```

e le principali informazioni che contiene sono:

- `<CanonicalizationMethod>`: al fine di evitare variazioni del digest dovute a modifiche della codifica XML che non influiscono sulla semantica del documento, come ad esempio l'aggiunta/eliminazione di spazi e il cambiamento dell'ordine degli attributi, prima di computare la firma si applica all'elemento `<Signature>`[3] un algoritmo di canonicalizzazione (specificato appunto dall'elemento `<CanonicalizationMethod>`) che riporta tutti i documenti semanticamente equivalenti a una stessa codifica (ad esempio, potrebbe eliminare tutti gli spazi non significativi e mettere in ordine alfabetico gli attributi).
- `<SignatureMethod>`: l'algoritmo usato per generare la firma dell'elemento `<Signature>` (ad esempio RSA-SHA1, che indica che il digest viene computato con la funzione hash SHA-1 e cifrato con l'algoritmo RSA).
- `<Reference>`: un riferimento (URI) a una delle risorse che si stanno firmando, per la quale vengono anche indicati:
  - `<Transforms>` (opzionale): delle trasformazioni (tipicamente trasformazioni di un documento XML, specificate tramite XPath o XSLT) che la risorsa deve subire prima che la firma venga generata;
  - `<DigestMethod>`: il metodo (la funzione hash) usato per calcolare il digest della risorsa;

[2] Il simbolo "?" denota zero o una occorrenze, "+" indica una o più occorrenze e "*" indica zero o più occorrenze.

[3] Per la precisione, la canonicalizzazione e il calcolo della firma avvengono sul sotto-elemento `<SignedInfo>`, non sull'intero elemento `<Signature>`.

  - `<DigestValue>`: il valore del digest della risorsa.

  Possono esserci più elementi `<Reference>` che puntano a risorse diverse, le quali possono anche essere diverse porzioni dello stesso documento XML.

- `<SignatureValue>`: il valore della firma dell'elemento `<Signature>`.
- `<KeyInfo>` (opzionale): le informazioni sulla chiave pubblica.
- `<Object>` (opzionale, e può essercene più di uno): una risorsa su cui è applicata la firma, nel caso del formato enveloping.