Firme digitali

Alessandro Pioggia

30 luglio 2022

Indice

Capitolo 1

Introduzione

1.1 Che cos'è la firma digitale?

1.1.1 Dal punto di vista generale

La firma digitale rappresenta una tecnica, fondata su precisi principi matematici, che ha lo scopo di dimostrare l'autenticità di un documento digitale, garantendo:

- l'integrità dei dati in esso contenuti;
- l'autenticità delle informazioni relative al sottoscrittore;
- la non alterabilità del documento;
- le non ripudiabilità della firma.

Le caratteristiche sovraelencate implicano che, il sottoscrittore una volta apposta la firma:

- non potrà disconoscere il documento, il quale non potrà essere assolutamente modificato (una eventuale modifica ne annulla la validità);
- diventa l'unico titolare del certificato, dal momento che è in possesso delle credenziali per accedervi.

1.1.2 Dal punto di vista tecnico

La firma può essere realizzata attraverso protocolli crittografici simmetrici (a chiave privata) o asimmetrici (a chiave pubblica). La maggior parte delle applicazioni utilizzano procolli asimmetrici, perché permettono di creare soluzioni più semplici e computazionalmente efficienti rispetto agli altri.

1.2 Firma digitale vs analogica

Facendo una attenta ricerca, è possibile stabilire che, le proprietà godute dalle firme digitali sono presenti anche in quelle analogiche, eppure è risaputo che le prime sono decisamente più sicure ed affidabili rispetto alle altre, questo perché:

- la firma è falsificabile solo attraverso la conoscenza della chiave privata del firmatario, inoltre, modificando anche solo un bit del documento, esso perde di validità (per il motivo analogo, non è nemmeno riutilizzabile), di conseguenza è una tutela molto più forte rispetto alla firma analogica;
- l'autore non può negare la paternità della dichiarazione presente nel documento siccome, al momento della firma era l'unico in possesso della chiave privata.
- la firma dipende fortemente dal documento sul quale viene posta.

Capitolo 2

Certificati digitali

I protocolli a chiave pubblica portano tanti vantaggi, non è un caso il loro utilizzo pervasivo, in ambito comunicativo, a livello globale... detto ciò, c'è purtroppo un side-effect, derivato dalla natura del protocollo, ovvero il fatto che sia semplice ed immediato il recupero della chiave pubblica. L'esposizione della public key rende il sistema potenzialmente vulnerabile agli attacchi informatici del tipo key-only (il più noto è l'attacco man in the middle), il problema viene parzialmente risolto attraverso il rilascio di certificati digitali.

2.0.1 Attacco man in the middle

Data una comunicazione con protocollo a chiave asimmetrica fra due critto-analisti, l'attacco informatico consiste nell'intrusione di un terzo soggetto, che si interpone fra mittente e destinatario, a loro insaputa.

Scenario base

Supponiamo che Marco voglia comunicare con Riccardo, inconsapevoli della presenza di un intruso:

- Marco richiede la chiave pubblica di Riccardo attraverso l'invio di una mail;
- l'intruso, in attesa del momento giusto, intercetta la richiesta di Marco e risponde inviando la propria chiave pubblica, ingannandolo;
- l'attacco è quasi ultimato, a questo punto l'intruso rimane in ascolto e ciascun crittogramma inviato da Marco, viene cifrato con la chiave dell'intruso e ritrasmesso poi a Riccardo.

2.0.2 Metodo risolutivo

Come già citato, l'attacco può essere contenuto attraverso l'utilizzo di certificati digitali, ovvero dei documenti elettronici che attestano la relazione 1:1 tra la chiave pubblica e l'identità di un soggetto. Il certificato in questione contiene una vasta gamma di informazioni, le più significative sono:

- firma digitale dell'emittente;
- identità del proprietario;

- periodo di validità.

Ciò che rende i certificati digitali validi ed affidabili è il rilascio da parte di un ente terzo e fidato come autorità di certificazione, in sigla CA. Dal momento che si tratta di un trusted third party, si scongiura l'eventualità che il documento venga falsificato.

2.0.3 Certificate authority

Da wikipedia:

In crittografia, una Certificate Authority, o Certification Authority (CA; in italiano: "Autorità Certificativa"[1]), è un soggetto terzo di fiducia (trusted third part), pubblico o privato, abilitato ad emettere un certificato digitale tramite una procedura di certificazione che segue standard internazionali e in conformità alla normativa europea e nazionale in materia. Il sistema in oggetto utilizza la crittografia a doppia chiave, o asimmetrica, in cui una delle due chiavi viene resa pubblica all'interno del certificato (chiave pubblica), mentre la seconda, univocamente correlata con la prima, rimane segreta e associata al titolare (chiave privata). Una coppia di chiavi può essere attribuita ad un solo titolare. L'autorità dispone di un certificato con il quale sono firmati tutti i certificati emessi agli utenti, e ovviamente deve essere installato su una macchina sicura.

2.0.4 CRL

I certificati digitali hanno una scadenza, però possono andare incontro ad una revoca anticipata, l'insieme delle revoche sono catalogate in un certificate revocation list (CRL), che viene pubblicato e di conseguenza aggiornato periodicamente dalla autorità di certificazione. Lo stato di revoca si concretizza nel caso in cui si verifichi almeno uno dei seguenti casi:

- il CA rilascia in modo improprio un certificato;
- si sospetta che una chiave privata sia stata compromessa;
- inadempimento.

Il CRL dà dunque la possibilità di verificare la validità di un certificato, non è però l'unico, ad esempio è possibile interrogare direttamente la CA attraverso il protocollo OCSP.

Capitolo 3

Funzionamento

3.1 Fasi

3.1.1 Generazione impronta digitale

La prima fase consiste nella generazione dell'impronta digitale (o message digest), attraverso una funzione di hashing, che permette di ottenere una stringa di grandezza costante, è unica e non invertibile.

3.1.2 Generazione della firma digitale

Il risultato ottenuto nel passaggio precedente viene cifrato, attraverso la chiave privata, il risultato di questo procedimento è la firma. Nel passaggio successivo la firma viene allegata, insieme alla chiave pubblica al documento. Importante considerare che chiunque può verificare la firma, il caso più comune è quello in cui un giudice si impegna per risolvere un dissidio fra 2 soggetti.

3.1.3 Invio al destinatario

Il mittente invia il documento firmato attraverso il metodo indicato e il certificato (CA) al destinatario.

3.1.4 Verifica del certificato

Il destinatario verifica l'autenticità attraverso la propria copia della chiave pubblica di CA. Se la verifica va a buon fine, si procede con la decifrazione e consecutiva verifica della firma. In particolare, il ricevitore decifra il crittogramma ricevuto attraverso la sua chiave privata. Nel caso in cui il messaggio hashato, sia uguale alla tupla (funzione hash, chiave pubblica del mittente), la verifica va a buon fine.

Hash function

Il ruolo giocato dalla funzione hash è fondamentale in questo processo, le sue caratteristiche fanno in modo che, la probabilità che da documenti diversi, si possa ottenere la stessa impronta, è infinitesimale.

Capitolo 4

Implementazione della firma digitale con RSA

In questo capitolo verrà mostrata l'implementazione di firma digitale, nella pratica, attraverso la generazione di chiavi RSA. Si tratta di uno script in python, che sfrutta la tecnologia pycryptodome per la generazione della chiave. L'esempio comprende creazione, istanziamento e verifica della firma. A livello teorico, il sistema di crittografia a chiave pubblica RSA, fornisce uno schema di firma digitale, composto da firma con conseguente verifica, basato sulle operazioni con modulo e sui logaritmi discreti e sul problema legato alla fattorizzazione degli interi.

4.1 Lo standards PKCS1

Il codice scritto all'interno del capitolo rispetterà lo standard PKCS 1, il primo di una famiglia di regole, chiamate Public-key Cryptography standards, che tutelano la corretta implementazione degli algoritmi RSA. Cita:

- le proprietà che le chiavi devono possedere;
- le operazioni consigliate per quanto riguarda la cifratura e decifratura;
- gli schemi crittografici sicuri.

4.2 Generazione delle chiavi

L'algoritmo RSA è in grado di generare una coppia di chiavi delle dimensioni di 1024, 2048... fino a 16384 (supporta in realtà anche chiavi di dimensione più grande, però la performance diventa troppo scarsa per un eventuale uso pratico). La coppia di chiavi è formata da una chiave:

- pubblica $\{n, e\}$;
- privata $\{n, d\}$

Considerando che, i valori n e d sono generalmente interi di grandi dimensioni, mentre e è di dimensioni più piccole.

Per definizione la coppia di chiavi RSA rispetta la seguente proprietà:

$$(m^e)^d \equiv ((m)^d)^e \equiv m(mod\, n)\, \forall\, m \in [0..n)$$

Attraverso il metodo generate, fornito dalla libreria pycryptodome, si è in grado di generare agilmente la coppia di chiavi, pubblica e privata, sotto-forma di stringhe. Il metodo prende come argomento un intero, che rappresenta il numero di bit della chiave, per questo esempio è stata generata una coppia di chiavi a 128 byte, quindi 1024 bit.

```
from Crypto.PublicKey import RSA
from Crypto.Signature.pkcs1_15 import PKCS115_SigScheme
from Crypto.Hash import SHA256
import binascii

keyPair = RSA.generate(bits=1024)
privateKey, publicKey = keys.privateKey(), keys.publicKey()
```

4.3 Firma del messaggio (mittente)

In seguito alla generazione delle chiavi RSA, si procede con la firma di un messaggio arbitrario sfruttando la chiave privata. La porzione di codice sottostante esegue i seguenti passaggi (considerando la chiave privata di esponente d):

- $h(m) \rightarrow$ hashing del messaggio, sfruttando l'algoritmo SHA-512;
- $f = h^d(modn) \rightarrow$ generazione della firma, attraverso la chiave privata ottenuta precedentemente e creazione del crittogramma, che servirà poi al destinatario per effettuare la verifica.

Importante notare che, dovendo rientrare nel range imposto dalla proprietà vista precedentemente, la firma ottenuta (f) è un intero la cui dimensione è rappresentata dall'intervallo $[0, ..n)$

```
#funzione che esegue lo hash del messaggio
def hash(message):
   return SHA256.new(msg)

#funzione che effettua la firma
def sign(hash, keys):
   signer = PKCS115_SigScheme(keyPair)
   return signer.sign(hash)

message = input("Insert the message that you'd like to sign")
signature = sign(hash(message))
```

4.4 Verifica della firma

In questa fase si effettua la verifica della firma, dunque viene effettuata la decrittazione attraverso la chiave pubblica e comparando lo hash della firma con quello del messaggio originale. Per verificare dunque, una firma f, per un messaggio m, con la chiave pubblica di esponente e è necessario:

- $h(m) \rightarrow$ calcolare lo hash del messaggio;
- $h' = f^e \ (mod\, n) \rightarrow$ eseguire la decrittazione della firma;
- $h' = ?h \rightarrow$ comparare lo hash della firma con quello del messaggio digitale.

Nel caso in cui la firma sia corretta, la seguente proprietà verrà rispettata

$$h' = f^e \ (mod\, n) = (h^d)^e \ (mod\, n) = h$$

```
#Se ritorna true -> firma valida, false -> firma non valida

def verify(message):
   hash = SHA256.new(msg)
   verifier = PKCS115_SigScheme(publicKey)
   return verifier.verify(hash, signature)

message = input("Insert the message that you'd like to verify")
verify(message)
```

4.5 Tentativo di manomissione della firma

In questo ultimo passaggio, sarà nostra premura verificare che lo script funzioni correttamente, in particolare, seguiamo i seguenti passaggi:

- definiamo un messaggio;
- definiamo un elemento di disturbo, chiamato ruiner, che verrà poi concatenato al messaggio originale;
- applichiamo il metodo di verifica sul messaggio originale;
- solo dopo aver richiamato il metodo di verifica sul messaggio originale, lo riapplichiamo al messaggio disturbato, se viene ritornato il valore false, significa che lo script è funzionante.

```
#Se ritorna true -> firma valida, false -> firma non valida

message = "L'esclusione di Damian Lillard dall'all star game e' una vera e propria
    ingiustizia!"
ruiner = "!.d-"

verify(message) #returns true
message = message.concat(ruiner)
verify(message) #returns false
```

Capitolo 5

L'algoritmo DSA

DSA, meglio noto come digital signature alghoritm, è un vero e proprio standard, crittograficamente sicuro, per quanto riguarda la realizzazione (con tanto di meccanismo di verifica) delle firme digitali. Si tratta di una alternativa dell'algoritmo RSA, per via delle limitazioni di brevetto dell'appena citato metodo. La sicurezza di DSA si basa sul'inesistenza di un algoritmo efficiente per il calcolo del logaritmo discreto.

5.1 Funzionamento

5.1.1 Firma

DSA esegue lo hashing di un messaggio, successivamente genera un intero random k e esegue la computazione della firma, che risulterà come una coppia di interi $\{r, s\}$, con r. Il valore r viene ricavato dal numero randomico k, generato al passaggio precedente, mentre s si ottiene sfruttando:

- $h(m) \rightarrow$ il messaggio hashato;
- la chiave privata;
- il valore randomico k.

5.1.2 Verifica

Il meccanismo di verifica sfrutta:

- $h(m) \rightarrow$ il messaggio hashato;
- la chiave pubblica;
- la firma r, s.

5.1.3 Implementazione in pseudocodice dell'algoritmo, con relativa dimostrazione

Generazione di chiavi

Il processo di generazione di chiavi è fondamentale nell'implementazione dell'algoritmo, in quanto i dati estratti da questa operazione formano a tutti gli effetti l'input per le fasi successive.

```
#Viene scelto un numero primo q, di lunghezza d bit
q = getPrimeNumber(size = d)
#Viene scelto un numero primo lungo L bit, tale che p = q*i + 1, con i numero intero
    e L compreso nell'intervallo fra 512 e 1024 e divisibile per 64
i = getInteger()
p = getPrimeNumber(size = L
        , predicate = (p == q * getRandomInteger()
        && L.between(512, 1024) && L mod 64==0))
#Viene scelto h, in modo che sia compreso fra 1 e p - 1
h = getPrimeNumber(predicate = Integer.between(1, p - 1))
#Viene scelto g, che deve risultare maggiore di 1
g = pow(h, z) mod p
#Calcolo la chiave pubblica, si tratta di un valore random compreso fra 0 e q
privateKey = random(predicate = x.between(0, q))
#calcolo la chiave privata, ottenuta da g elevato alla x modulo p
publicKey = pow(g, x) mod p
```

Firma

Sfruttando i dati calcolati nel passo precedente, si procede con il calcolo della firma. In questo passaggio è opportuno fare molta attenzione durante la generazione del valore randomico k, in quanto apre alla possibilità di una potenziale vulnerabilità. In particolare, se due messaggi differenti, sono firmati usando lo stesso valore k e la stessa chiave privata, un hacker può effettuare la computazione della chiave privata del firmatario in maniera diretta.

```
#viene generato un numero casuale k, compreso fra 0 e q
k = random(predicate = k.between(0, q))
#si calcola r
r = (pow(g, k) mod p) mod q
#si calcola il messaggio hashato
m = getMessage()
hash = h(m)
#si calcola s
s = lambda x : (pow(k, -1) * (hash + x * r))
#la firma e' una tupla composta da r ed s
signature = (r, s)
```

Verifica

Arrivati a questo punto, la firma è stata creata, non resta che verificarla.

```
#Il primo controllo verifica che la firma abbia valori r ed s compresi nei range che
    consideriamo accettabili
predicate = r.isBetween(0, q) && s.isBetween(0, q)
if (!predicate) throw signatureNotValidException
#Dopodiche' verranno calcolati i valori w, u1 e u2
w = pow(s, -1)
u1, u2 = (hash() * w) mod q, (r * w) mod q
#Infine viene calcolato v, ovvero la funzione di verifica, che deve risultare uguale
    ad r, il primo valore della firma (r, s)
v = (pow(g, u1) * pow(g, u2) mod p) mod q
if (v == r) return Ok("valid signature") else throw signatureNotValidException
```

5.2 Implementazione in python dell'algoritmo

Come per rsa, ritengo opportuno includere nel capitolo una implementazione dell'algoritmo, in modo che sia possibile avere una idea (molto generale), dell'utilizzo pratico. Il linguaggio utilizzato è, anche in questo caso, python mentre la tecnologia sfruttata è pycryptodome. A differenza del capitolo precedente, non presenterò una trattazione teorica che contestualizzi il codice, in quanto nelle sezioni precedenti sono già stati spiegati i passaggi da effettuare.

```
from Crypto.PublicKey import DSA
from Crypto.Hash import SHA256
from Crypto.Signature import DSS

#Generazione della chiave

msg = "Ciao, come stai?".encode()
key = DSA.generate(size = 1024)
publickey = key.publicKey()

#Firma

hash = SHA256.new(msg)
signer = DSS.new(key, 'fips-186-3')
signature = signer.sign(hash)

#Verifica

verifier = DSS.new(publickey, 'fips-186-3')
return verifier.verify(hash, signature)
```

Capitolo 6

L'algoritmo ECDSA

Nonostante l'algoritmo DSA sia una soluzione solida nel campo delle firme digitale, con l'evoluzione tecnologica è stato possibile creare algoritmi, più semplici e performanti, sfruttando ad esempio la crittografia basata curve ellittiche. Al giorno d'oggi è preferibile una soluzione che comprenda l'utilizzo delle curve ellittiche, in quanto comportano la creazione di chiavi e conseguentemente firme più corte, oltre a livelli di sicurezza più alti.

6.1 Funzionamento

ECDSA è un adattamento del classico DSA, con la differenza che, dal lato matematico, si fa affidamento ai gruppi ciclici di curve ellittiche su campi finiti e sulla complessità computazionale del problema ECLDP(elliptic-curve discrete logarithm problem). Queste curve sono descritte dai loro parametri, definiti da precisi standard crittografici (ad esempio RFC 5693) e sono:

- il punto generatore G, utilizzato per la moltiplicazione scalare sulla curva, si tratta di moltiplicare un intero per un punto della curva ellittica;
- ordine n dei sottogruppi dei punti della curva, generati da G, che definisce la lunghezza delle chiavi private (ad esempio secp256k1 ha 256 bit di chiave).

6.1.1 Generazione di chiavi

La coppia di chiavi che viene generata dall'algoritmo in questione è composta da una chiave privata (valore intero random che rientra nell'intervallo da 0 a $n-1$) ed una pubblica (punto della curva ellittiva), che si ottiene moltiplicando la chiave privata per il punto generatore G. In python l'implementazione, che sfrutta una tecnologia chiamata pycoin, risulta la seguente:

```
from pycoin.ecdsa import generator_secp256k1, sign, verify
import hashlib, secrets

privateKey = secrets.randbelow(generator_secp256k1.order())
publicKey = (generator_secp256k1 * privateKey).pair()
```

6.1.2 Firma

La firma viene generata prendendo in input un messaggio m ed una chiave privata $privateKey$, restituisce in output la firma f.

$$f(m, privateKey) \rightarrow \{r, f\}$$

Il funzionamento può essere semplificato attraverso i seguenti passaggi (si consideri che una implementazione reale è molto più modulare, però difficile da comprendere):

- $h = hash(m) \rightarrow$ come di consueto, il primo passaggio consiste nell'effettuare lo hashing del messaggio da firmare;
- $k = random() \rightarrow$ si prosegue con la generazione di un valore casuale k, a meno che si consideri di sfruttare implementazioni alternative, è importante che non sia deterministico, in quanto potrebbe portare a delle vulnerabilità;
- $R = k * G; r = R.x \rightarrow$ successivamente si calcola r, ovvero il primo elemento della coppia che formerà la firma;
- $f = k^{-1} \cdot (h + (r \cdot privateKey))(modn) \rightarrow$ infine, si calcola la signature proof, ovvero la prova di firma.

Terminati questi passaggi si otterrà la firma, che verrà ritornata come coppia $\{r, f\}$.

Dimostrazione teorica

Il processo di "firma della firma", codifica la coordinata x di un punto random R attraverso le trasformazioni di curve ellittiche. Dunque la chiave privata e il messaggio hashato permettono di ottenere f, che è stata già introdotta come la "prova di firma" (in inglese signature proof), che è la prova che il firmatario conosce la chiave privata. La firma $\{r, s\}$ non può rivelare la chiave privata per via della difficoltà del problema ECDLP.

Implementazione in python

Il metodo sottostante permette di effettuare la firma dato un messaggio (in chiaro), della quale effettuerà lo hashing attraverso SHA3-256. Il secondo argomento richiesto è la chiave privata secp256k1, che servirà a calcolare la firma $\{r, s\}$ come una coppia di interi di 256 bit.

```
def signECDSAsecp256k1(msg, privKey):
   def sha3_256Hash():
      hashBytes = hashlib.sha3_256(msg.encode("utf8")).digest()
      return int.from_bytes(hashBytes, byteorder="big")
   msgHash = sha3_256Hash(msg)
   signature = sign(generator_secp256k1, privKey, msgHash)
   return signature

signature = signECDSAsecp256k1(msg, privKey)
```

6.2 Verifica

Come di consueto, è necessario che la firma venga verificata, l'algoritmo richiede in input il messaggio firmato, la firma $\{r, s\}$ e la chiave pubblica (del firmatario). Una volta ottenuti, si effettua la verifica, che può dare esito positivo (firma verificata) o negativo (firma non verificata). Il funzionamento può essere semplificato attraverso i seguenti passaggi.

- $hash = h(m) \rightarrow$ il primo passaggio richiede lo hashing del messaggio, la funzione hash è la stessa utilizzata nella fase di firma;
- $f_{inv} = f^{-1} \, mod \, n \rightarrow$ successivamente viene calcolato l'inverso della firma, con modulo n;
- $R' = (h \cdot f_{inv}) \cdot G + (r \cdot f_{inv}) \cdot publicKey \rightarrow$ la formula non fa altro che recuperare il punto random utilizzato durante la firma, in particolare ne è stata sfruttata la coordinata x;
- $R.x \; =? \, r \rightarrow$ infine si calcola il risultando confrontando $R.x$ e r.

Dimostrazione teorica

Nella fase di verifica della firma, viene decodificato f, ottenendo il punto R (trovato precedentemente) sfruttando la chiave pubblica e il messaggio hashato $h(m)$, dopodiché viene effettuato il confronto fra: $R.x$ e r.

Implementazione in python

Il metodo sottostante prende in input un messaggio, una firma $\{r, s\}$ e infine una chiave pubblica a 256, in funzione di questi elementi verifica che la firma sia valida o meno. Importante notare che l'algoritmo di hashing è lo stesso utilizzato durante la fase di firma, condizione necessaria per il corretto funzionamento dell'algoritmo.

```
def verifyECDSAsecp256k1(msg, signature, pubKey):
   def sha3_256Hash(msg):
      hashBytes = hashlib.sha3_256(msg.encode("utf8")).digest()
      return int.from_bytes(hashBytes, byteorder="big")
   msgHash = sha3_256Hash(msg)
   valid = verify(generator_secp256k1, pubKey, msgHash, signature)
   return valid

valid = verifyECDSAsecp256k1(msg, signature, pubKey) #ritorna true

msg = 'messaggio modificato'

valid = verifyECDSAsecp256k1(msg, signature, pubKey) #ritorna false
```

Capitolo 7

L'algoritmo EdDSA

L'algoritmo DSA, analizzato nel capitolo precedente, non è la migliore soluzione algoritmica conosciuta, in particolare, sfruttando la tecnologia nota come "curve ellittiche", è possibile costruire un algoritmo particolarmente performante, chiamato EdDSA. Inventato da Edwards, rappresenta una variante deterministica dell'algoritmo della firma di Schnorr. Dal lato pratico, EdDSA è più semplice, più sicuro e più veloce di ECDSA.

Capitolo 8

Falsificazione della firma

La firma digitale garantisce un ottimo grado di sicurezza, detto ciò, nonostante l'utilizzo di certificati rilasciati da enti di terze parti, in determinati casi è stato possibile falsificare la firma. Importante ricordare però che gli attacchi, nella maggior parte dei casi, sono efficaci perché nella creazione della firma digitale sono stati commessi degli errori. Le vulnerabilità più comuni:

- l'utente rende nota, probabilmente per inesperienza, la chiave privata;
- la manipolazione della procedura di firma;
- introduzione di un attore malintenzionato che si assicura che il firmatario veda qualcosa di diverso da ciò che intende firmare.

8.1 Universal signature forgery (USF)

L'USF è un attacco informatico, che sfrutta la proprietà di non falsificazione della firma digitale. In particolare, l'attacco consiste nel rendere appositamente invalida la firma durante la fase di verifica, aggiungendo al documento ulteriori dati. Si dice che l'USF "confonda" la logica di validazione, ovvero l'insieme di operazioni crittografiche che portano alla certificazione della firma. Se l'hacker riesce con il suo attacco USF, la logica di convalida online o l'applicazione di visualizzazione mostrerà che la firma elettronica è valida e appartiene ad un individuo o entità specifica sul suo pannello di visualizzazione.

8.2 Incremental saving attack (ISA)

Nel caso di un Incremental Saving Attack (ISA), l'obiettivo è quello di effettuare un salvataggio incrementale su un documento ridefinendone la struttura. Pertanto, l'obiettivo di questo attacco è il salvataggio incrementale o la funzione di aggiornamento incrementale di un documento PDF, che se utilizzata legittimamente consente a un utente di aggiungere annotazioni al proprio PDF. Queste annotazioni vengono salvate in modo incrementale come un nuovo corpo PDF dopo il contenuto originale del PDF. La funzione di salvataggio incrementale viene utilizzata anche per la firma del PDF e consente di aggiungere l'oggetto firma al contenuto del file originale. Normalmente, qualsiasi modifica dopo la firma di un documento attiverebbe un avviso che il documento è stato manomesso. Tuttavia, quando esegue un attacco ISA, l'autore dell'attacco potrebbe aggiungere contenuto aggiuntivo, come nuove pagine o annotazioni a un PDF già

firmato. Tecnicamente, questa violazione non è un attacco. Invece, è un exploit della funzione di salvataggio incrementale del PDF. Tuttavia, la vulnerabilità si verifica quando la logica di convalida della firma non rileva che il contenuto del file PDF è stato manomesso. Il contenuto non firmato che è stato aggiunto dopo la firma del documento è semplicemente visto come un aggiornamento dalla persona fisica o giuridica che ha originariamente creato la firma elettronica del documento. Un attacco ISA riuscito comporterà la visualizzazione di nuovi aggiornamenti di contenuto/corpo, mentre i processi di verifica della firma rimarranno ignari che sono state apportate modifiche o aggiornamenti al documento PDF.

8.3 Signature Wrapping (SWA)

Un attacco Signature Wrapping (SWA) utilizza un approccio unico per aggirare la protezione della firma di un PDF senza accedere alla sua funzione di salvataggio incrementale. Lo fa spostando la seconda parte del /ByteRange firmato alla fine del documento violato. Nel frattempo, l'attaccante riutilizza quindi il puntatore xrif all'interno del trailer firmato del documento per fare riferimento al suo xrif manipolato. In alcuni casi, l'attaccante può anche avvolgere la seconda parte riposizionata con un oggetto stream o un dizionario per impedirne l'elaborazione da parte del PDF o della funzione di protezione della firma online. In un attacco SWA riuscito, un utente malintenzionato può aggiungere oggetti dannosi non firmati nel documento. Se ha scelto di avvolgere la seconda parte spostata, questi oggetti possono essere posizionati prima o dopo l'xrif manipolato. Se non viene aggiunto alcun wrapping, gli oggetti dannosi verranno posizionati dopo l'xrif manipolato. A seconda del visualizzatore PDF, l'autore dell'attacco può copiare l'ultimo trailer del file e inserirlo dopo il suo xrif manipolato per consentire l'apertura del file PDF e per ignorare la verifica della firma senza che le manipolazioni vengano rilevate.

8.4 Il paradosso della chiave privata

L'utilizzo di una chiave privata, nei protocolli asimmetrici, ne garantisce il funzionamento e dunque è una componente fondamentale nel garantire una comunicazione sicura. Detto ciò però, è necessario memorizzarla in un dispositivo fisico, ad esempio il computer dell'utente a cui è stata rilasciata e questo comporta una vulnerabilità, ovvero il fatto che la sicurezza della chiave privata dipenda fortemente dal dispositivo in cui è memorizzata. A questo proposito, negli anni si è cercato di ovviare al problema, proponendo diverse soluzioni, le più note:

- la smart card;
- l'otp (la one time password);
- token usb;
- token wireless.

8.4.1 Smart card

Una smart card è una carta, generalmente dalle dimensioni di un comune bancomat, che possiede al suo interno, un insieme di dispositivi hardware che permettono di gestire ed elaborare dati, seguendo precisi standard di sicurezza. Ho deciso di citarla perché è possibile memorizzare al suo interno la chiave privata. In particolare la carta, interfacciandosi con un lettore esterno, riceve l'hash calcolato, lo firma con la chiave privata e lo restituisce. Un ulteriore layer (o strato) di sicurezza è dato dal fatto che prima di ogni utilizzo, è necessario inserire un pin, in modo da

pregiudicarne un utilizzo malevolo da parte di un malintenzionato (è inoltre possibile revocare la validità del certificato della carta in caso di furto).

Problema principale

Il noto Ronald Rivest, inventore del crittosistema RSA, ha rimarcato una contraddizione presente nel meccanismo di firma digitale attraverso questi dispositivi. Lo scienziato definisce il meccanismo "intrinsecamente insicuro", in quanto, nonostante la chiave privata sia memorizzata in un dispositivo sicuro, durante il procedimento di firma diventa vulnerabile, in quanto deve dipendere dalla sicurezza del dispositivo con cui si interfaccia. Per ovviare al problema, si potrebbe pensare ad un'applicazione perfettamente affidabile per la firma digitale che sia eseguibile solo ed esclusivamente su una tipologia di dispositivi stand-alone portatili, che non permettano l'esecuzione di altri programmi.

8.4.2 Dispositivi OTP (one-time-password)

Mobile OTP non è altro che una password usa-e-getta, con breve scadenza (pochi secondi dopo la creazione) utile per l'autenticazione online e l'accesso sicuro ai servizi di firma digitale online, anche detta firma digitale "remota". Questo servizio aggira la necessità di utilizzo di token o smartcard fisici per l'accesso a servizi di firma digitale, rendendolo particolarmente efficace per i dispositivi mobili come gli smartphone o i tablet. La OTP Mobile è un sistema basato sull'autenticazione a due fattori (Two-factor Authentication), ossia il sistema utilizzerà due evidenze per verificare, con un ampio grado di sicurezza, l'identità dell'utente. Una volta verificata l'autenticità dell'identità attraverso il sistema OTP Mobile, l'utente potrà accedere ai servizi di firma digitale remota messi a disposizione dai certificatori accreditati DigitPA.

Principali vantaggi

- semplicità, è sufficiente possedere uno smartphone;
- rapidità, l'attivazione è molto rapida rispetto alla firma digitale classica;
- sicurezza, per ovvi motivi, una password che scade dopo pochi secondi elimina il rischio della staticità e violabilità di una password tradizionale;
- accessibilità, il servizio è utilizzabile sempre, comunque ed ovunque.

Considerazioni

I sistemi protetti da OTP, se implementati in maniera impropria, possono risultare vulnerabili ad un attacco informatico, chiamato snooping. Lo snooping rappresenta una tecnica di hacking in cui, in una comunicazione di rete, fra stazioni remote, l'hacker è in grado di intercettare il traffico trasmesso. Il rimedio principale consiste nell'utilizzo dello standard https (secure http), il quale oscura i dati trasmessi, crittografandoli. Al giorno d'oggi è pressoché impossibile trovare siti di aziende rinomate, pubbliche o private, che non siano protette da questo standard, di conseguenza non si tratta di una minaccia.

8.4.3 Token USB

L'usb token è un dispositivo hardware, che permette di effettuare la firma, funziona in modo autonomo ,è sufficiente una uscita usb e comprende il software per la firma e per la gestione del dispositivo oltre ad una memoria di piccole ma sufficienti dimensioni. La caratteristica principale del token USB, è che viene consegnato direttamente dall'ente certificatore, che svincola il possessore del dispositivo dalla necessità di introdurre password temporanee o codici pin di accesso, in quanto è gestito tutto autonomamente dal dispositivo.

Capitolo 9

Bibliografia

- Wikipedia
- https://www.geeksforgeeks.org/types-of-digital-signature-attacks/
- https://www.pd.camcom.it/camera-commercio/notizie/avvisi-e-comunicazioni/Attacco-Malware-Spam-possessori-Firma-Digitale-Infocert
- https://shakeylead.com/en/digital-signature-vulnerabilities-and-protection-methods/