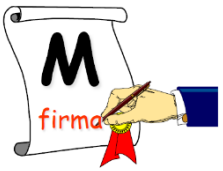
## 10.0 FIRME DIGITALI



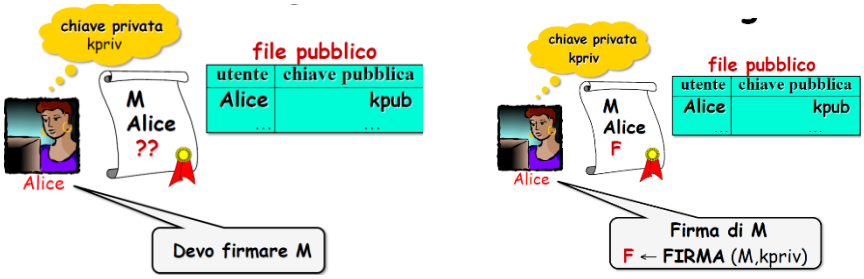

Mediante l'utilizzo della firma digitale, si vuole realizzare un sistema che sia equivalente alla firma convenzionale, ossia quella riconosciuta su carta.

Una soluzione naive potrebbe essere quella di incollare la firma digitalizzata su un documento, ma è un'idea poco utile perché l'azione compiuta, può essere tranquillamente fatta da un'attaccante.

La firma digitale deve soddisfare 3 requisiti, che sono:

- La firma digitale deve poter essere firmata facilmente e prodotta dal legittimo firmatario.
- Nessun utente può riprodurre la firma di altri.
- Chiunque può facilmente verificare una firma.

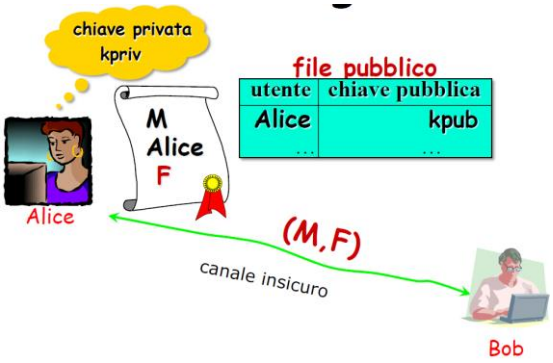
Chiaramente la firma digitale può essere applicata, e ne vediamo ora il suo funzionamento in uno scenario a chiave pubblica:




Alice, allo stesso tempo, costruisce una chiave privata e una chiave pubblica, naturalmente la chiave pubblica è localizzata in un file accessibile a tutti, mentre la privata la tiene per se e non la darà mai a nessuno, e la utilizzerà per emettere la firma digitale.

Quando deve firmare un documento, Alice avendo la sua chiave privata, fa girare un algoritmo di firma: *FIRMA (M,kpriv)* → ***F*** che ha in input il messaggio che deve essere firmato e la chiave privata, e produce la firma digitale F, che rappresenta una stringa di bit che viene aggiunta al documento. Questo è il processo di creazione di una firma digitale.

Successivamente, per verificare una firma digitale, ad esempio assumiamo questo scenario:




Normalmente la firma non viene cifrata, poiché il suo scopo è di autenticare il messaggio per garantire che un messaggio M sia stato effettivamente mandato da una certa persona. Se Bob deve verificare che F è una firma di Alice per M, prende un algoritmo che conoscono tutti: ***VERIFICA (F,M,kpub)= SI***, falsa altrimenti, che prende in input la firma di Alice, il messaggio, e la chiave pubblica di Alice. Se la risposta è SI, allora la firma è corretta, non corretta altrimenti.

Un attaccante potrebbe tentare di rompere quest'algoritmo cercando di generare tutte le possibili firme in maniera tale che prima o poi troverà quella corretta. Chiaramente è possibile farlo in linea teorica, poiché si deve considerare il tempo d'impiego di questa operazione perché se il tempo è molto alto, allora sarà sicuramente impossibile da realizzare.

Formalmente, parlare di sicurezza per uno scherma di firme digitale, deriva dalla definizione di:

- ***Tipo di attacco***, che può portare l'attaccante
- ***Scopo dell'attacco***

I ***tipi di attacco*** sono di 3 tipi diversi:

1. ***Key-only-Attack***: l'attaccante conosce solo kpub di Alice.
2. ***Known Message Attack***: l'attaccante conosce una lista di messaggi e le relative firme di Alice.
3. ***Chosen Message Attack***: l'attaccante sceglie dei messaggi e chiede al Alice di firmali.

La tipologia di risorse che l'attaccante ha a disposizione nell'attacco, è stato inserito in maniera crescente.

Gli ***scopi dell'attacco*** sono:

1. ***Total break***: determinare kpriv di Alice per poter firmare qualsiasi messaggio.
2. ***Selective forgery***: dato un messaggio M selettivo, determinare la firma F tale che ***VERIFICA (F,M,kpub)= SI***.
3. ***Existensial forgery***: determinare una coppia (M,F) tale che ***VERIFICA (F,M,kpub)= SI***.

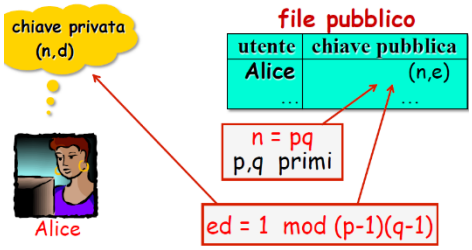
Le firme digitali più utilizzate al giorno d'oggi sono realizzate mediante: RSA, ElGamal, Digital Signature Standard (DSS).

## 10.1 UTILIZZO RSA PER LE FIRME DIGITALI

La sicurezza dell'RSA è basata sulla difficoltà di fattorizzare.

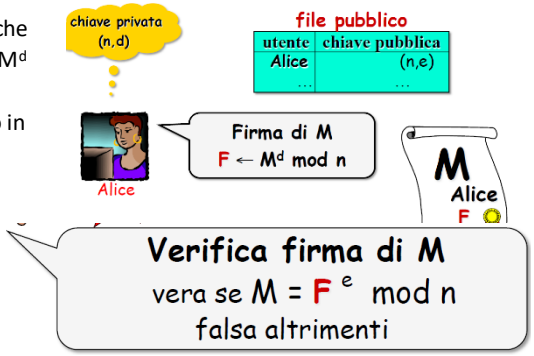
Un piccolo cenno sull'utilizzo di RSA:

Si costruisce contemporaneamente la chiave pubblica (n,e) e la chiave privata (n,d). n è uguale al prodotto di due numeri primi, e soddisfare la proprietà secondo cui il loro prodotto deve essere 1 in modulo (p-1)(q-1)

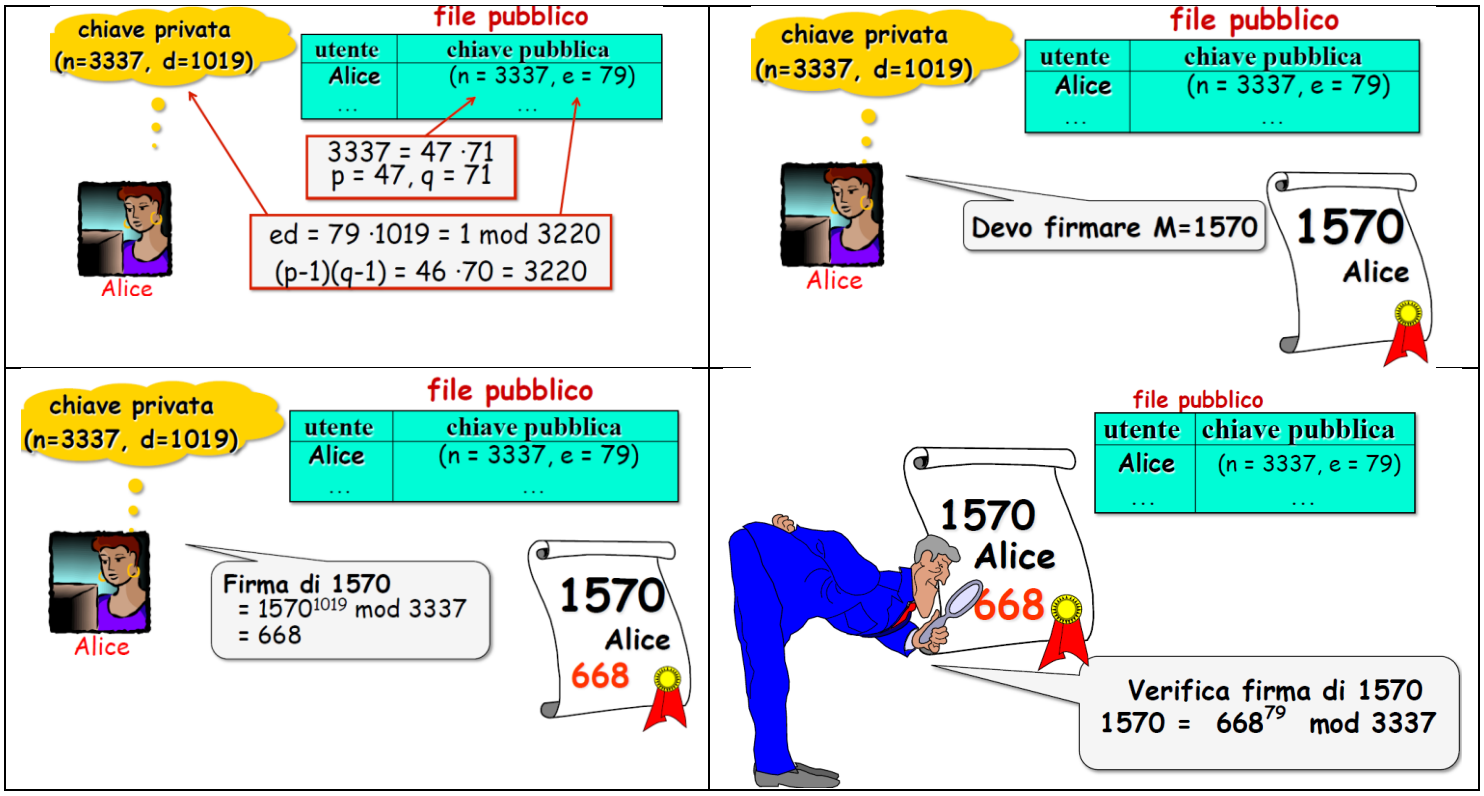



Per quanto riguarda la firma, si osserva che la chiave pubblica e la chiave privata sono le stesse che vengono utilizzate in RSA. Se Alice deve firmare un messaggio, esegue la seguente operazione: ($M^d$ mod n) → F, in cui, preso il messaggio, lo eleva alla d(esponente privato che conosce solo lei) modulo n. Dopo questa operazione, la firma F viene aggiunta al messaggio. Da questo momento in poi, la firma è visibile a tutti quanti ed è la stringa F.

Per verificare la correttezza della firma sul messaggio, conoscendo la chiave pubblica di Alice, si esegue la seguente operazione: ($F^e$ mod n) → M, viene quindi presa la firma, elevata all'esponente e che è presente nel file pubblico, in modulo n. Se il risultato di quest'operazione produce il messaggio originale allora la firma è corretta, altrimenti la firma non è corretta.




Viene mostrato un esempio:




La correttezza della firma mediante RSA è identica alla correttezza della cifratura di RSA:

$$F^e \bmod n = (M^d)^e \bmod n$$
$$= M^{ed} \bmod n \quad (ed = 1 \bmod (p-1)(q-1))$$
$$= M^{1+k(p-1)(q-1)} \bmod n$$
$$= M \cdot (M^{(p-1)(q-1)})^k$$
$$= M \bmod n \quad (\text{Teorema di Eulero: } M \in Z_n^* \Rightarrow M^{(p-1)(q-1)} = 1 \bmod n)$$
$$= M \quad (\text{Per } M \in Z_n / Z_n^* \text{ usa il teorema cinese del resto})$$

(poichè $0 \le M < n$)

Analizziamo la sicurezza di questo sistema di firme, combinando ***tipi di attacco*** e ***scopo dell'attacco***:

***SELECTIVE FORGERY-KEY ONLY ATTACK:*** L'attaccante conosce quindi la chiave pubblica (n,e) e conosce un messaggio selettivo M che gli viene dato in input. Con questi dati in suo possesso, vuole trovare la firma.

L'attaccante ha la necessità di effettuare questo calcolo: $M^d \bmod n$. Questa operazione è equivalente a "rompere" il crittosistema RSA. Naturalmente quindi, è un'operazione che non riesce a fare.

Poiché l'attaccante mediante questo tipo di attacco e questo scopo dell'attacco non riesce ad ottenere l'informazione richiesta, possiamo quindi dire che il sistema di sicurezza di firma dell'RSA è sicuro contro questo tipo di attacco e scopo di attacco.

***EXISTENTIAL FORGERY-KEY ONLY ATTACK***: Dal caso precedente, abbiamo cambiato solo lo scopo dell'attacco, cioè, in questo caso l'attaccante non deve calcolarsi la firma di un messaggio che gli viene dato, ma calcola la firma di un messaggio a suo piacere, e come prima conosce la chiave pubblica (n,e).

In questo caso l'attaccante riesce ad ottenere la coppia messaggio, firma in questo modo: sceglie una firma F a caso, ed esegue la seguente operazione: $(F^e \bmod n) \rightarrow M$. In questo caso quindi la sicurezza non è sicura.

***EXISTENSIAL FORGERY- KNOWN MESSAGE ATTACK:*** In questo caso, rispetto al caso precedente, cambia il tipo di attacco, poiché adesso l'attaccante conosce dei messaggi e le relative firme.

Assumendo dunque che l'attaccante conosca le coppie $(M_1,F_1)$ e $(M_2,F_2)$, e sfruttando la proprietà dell'omomorfismo dell'RSA che ci dice che:

$F1= M_1^d \bmod n \qquad F2= M_2^d \bmod n;$

$(F_1F_2)^e \bmod n= F_1^eF_2^e \bmod n= M_1M_2 \bmod n$

Di conseguenza, il prodotto delle due firme $F_1F_2 \bmod n$ è una firma valida per il prodotto dei due messaggi $M_1M_2 \bmod n$. Quindi l'attaccante potrà falsificare nuovi messaggi che non ha mai visto in precedenza. Anche in questo caso, RSA non è sicura.

***SELECTIVE FORGERY-CHOSEN MESSAGE ATTACK***: Questa volta, l'attaccante possiede un messaggio selettivo M in input e vuole falsificare la firma di quel messaggio, avendo anche la possibilità di scegliere un messaggio da far firmare ad Alice (naturalmente questo diverso dal messaggio selettivo M).

Se l'attaccante riuscisse ad ottenere la firma dal messaggio che manda ad Alice, può ricavare la firma del messaggio M.

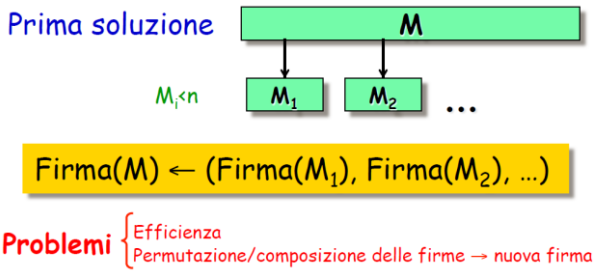
Anche in questo caso si sfrutta la proprietà dell'omomorfismo:

L'attaccante sceglie $M_1$ e $M_2$ tali che $M=M_1M_2 \bmod n$. Chiede ad Alice di firmare $M_1$ e $M_2$ ottenendo $F_1$ e $F_2$. Il prodotto $F_1F_2 \bmod n$, è una firma valida per M. Anche in questo caso, viene rotto il sistema.

Poiché, l'RSA risulta essere vulnerabile per vari attacchi, chiaramente non può essere usato in questo modo. C'è bisogno di qualche modifica al suo funzionamento per garantirne un funzionamento sicuro.

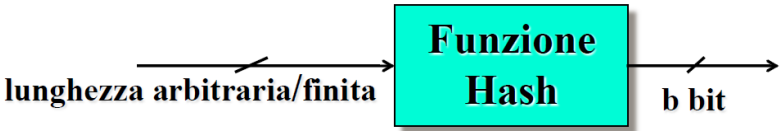
Insieme a questo problema, ne sorge un altro, poiché il messaggio M deve per forza essere più piccolo di n.

Ma se M>n, come si firma? Una possibile soluzione è dividere il messaggio M in blocchi che risultino essere più piccoli di n, ognuno di questi blocchi li firmo. La firma totale è quindi la sequenza di firme di ogni blocco. Da un punto di vista computazionale è oneroso, e risulta essere anche problematico da un punto di vista della sicurezza perché se prendo un insieme di firme e scambio 2 blocchi, ottengo una firma di un messaggio nuovo, in cui ho scambiato i 2 blocchi scelti del messaggio M. Ad esempio: se scambio Firma($M_1$) e Firma ($M_2$), ottengo un la firma di un messaggio in cui ho scambiato $M_1$ e $M_2$. Chiaramente questo crea un problema perché si creano nuove firme.




Come risolvo questi problemi?

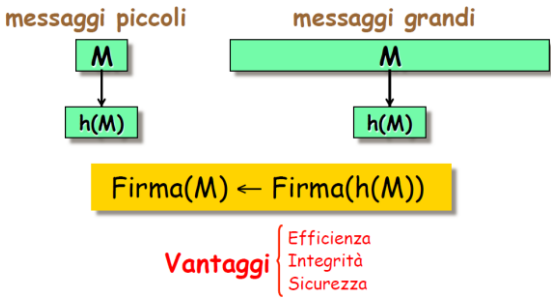
L'idea è di utilizzare la funzione Hash, che è una funzione deterministica che ha la caratteristica che qualunque sia la stringa che do in input di lunghezza arbitraria finita, ottengo un output di lunghezza fissata (256 bit ad esempio):




Il valore Hash h(M) è una rappresentazione non ambigua e non falsificabile del messaggio M. Gode di varie proprietà (le più importanti sono le ultime 2):

- Comprime.
- Facile da computare.
- ***Sicurezza forte***: computazionalmente difficile (tempo necessario impossibile) trovare 2 diversi messaggi con lo stesso valore Hash.
- ***One-way***: dato y è computazionalmente difficile trovare M tale che y= h(M)

Una volta che si ha a disposizione la funzione Hash, risolvo il problema della firma:

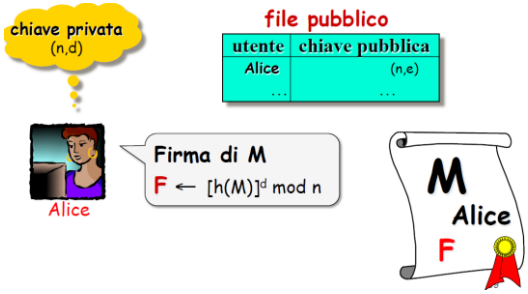



Non si va mai a firmare il messaggio in input, ma si firma sempre l'Hash del messaggio, qualunque esso sia. Quindi si vede che anche messaggi grandi non vengono divisi in blocchi, ma ne viene generato l'Hash.

In questo caso, quindi, il concetto di firma del messaggio è sempre mediato dal valore Hash della funzione.

I vantaggi sono: efficienza perché la funzione Hash e velocissima e non divido il messaggio in blocchi, integrità poiché non si ricevono attacchi come quello di scambiare blocchi di messaggi, sicurezza perché gli attacchi precedenti non valgono più perché mediante la funzione Hash, la proprietà di omomorfismo dell'RSA viene annullata.

Di conseguenza lo schema di firma dell'RSA viene cambiato in questo modo:




Se Alice vuole firmare un messaggio, calcola l'Hash del messaggio e lo eleva alla d modulo n. Così ottiene la firma.

Se qualcuno deve verificare se F è una firma di Alice per M, si effettua questo calcolo: se h(M)= $F^e$ allora la firma è quella giusta, falso altrimenti. Quindi si effettua $F^e$ e si controlla se questo valore è uguale all'Hash di M.

Vediamo solo un attacco che abbiamo descritto in precedenza, se ora funziona con questo nuovo sistema di sicurezza.

***EXISTENTIAL FORGERY-KEY ONLY ATTACK***: In questo caso l'attaccante non deve calcolarsi la firma di un messaggio che gli viene dato, ma calcola la firma di un messaggio a suo piacere, e come prima conosce la chiave pubblica (n,e).

Adesso, se l'attaccante sceglie una firma F a caso, calcola il valore $F^e$ mod n e lo salva in una variabile z. Per calcolare il messaggio deve trovare un messaggio di cui il valore z è l'Hash, sostanzialmente vuol dire cercare di invertire la funzione Hash. Ma la funzione Hash è difficile da invertire, quindi questa operazione di inversione non la riesce a fare in un tempo efficiente e di conseguenza non riesce a calcolare il messaggio M.

Anche gli altri attacchi visti in precedenza, ora che è stata introdotta la funzione Hash, falliscono.

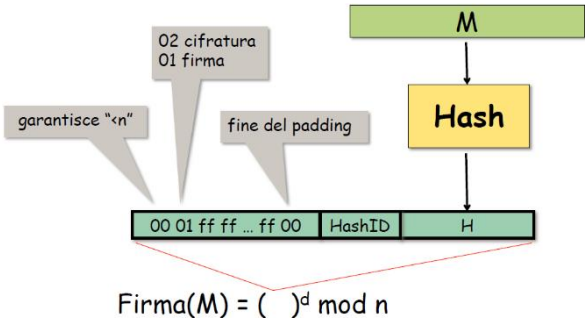
RSA viene sempre utilizzata (mai direttamente), ma con la funzione Hash.

Basta fare utilizzare semplicemente l'RSA con le funzioni Hash per firmare? In realtà no, perché con questo genere di operazioni si fa riferimento a degli standard esistenti, così come succedeva con la crittografia a chiave pubblica con l'RSA. In questo caso, si utilizza lo standard PKCS #1 che riguarda proprio l'RSA, riguardo crittografia a chiave pubblica e firme digitale. Così come per la crittografia a chiave pubblica, anche per la firma digitale, ci sono due schemi per la firma, di cui uno è più semplice (***RSASSA-PKCS1-v1_5***) e uno più elaborato (***RSASSA-PSS***) che riesco a dimostrare essere sicuro con un modello opportuno.

Nella crittografia a chiave pubblica, ***RSASSA-PKCS1-v1_5*** (dove SSA sta per Signature Scheme with Appendix) aveva problemi di sicurezza, mentre per la firma digitale, non esiste alcun attacco su questo schema di firme che dimostra che è poco sicuro, ma non esiste nemmeno una dimostrazione di sicurezza, a differenza del ***RSASSA-PSS.***

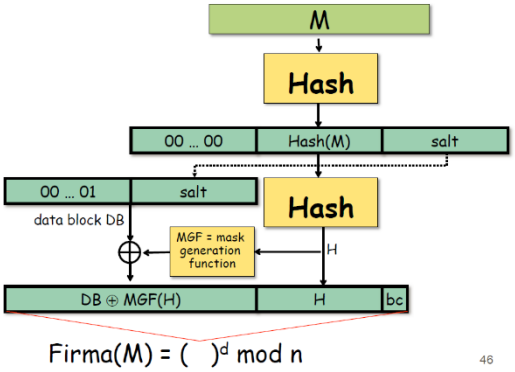
Il primo ***RSASSA-PKCS1-v1_5***:

I primi due byte, 00 mi garantiscono che il risultato dell'elaborazione sia minore di n e poi 01 perché questa è una operazione di firma. Fissati questi byte, la parte rimanente si basa su: gli ultimi bit (meno significativi) sono il risultato Hash del messaggio M. Prima dell'Hash del messaggio, c'è un identificativo della funzione Hash, poiché chiaramente di funzioni Hash ne esistono diverse e inserendo l'identificativo, specifico quale funzione Hash utilizzare. Prima dell'HashID c'è un padding in cui vengono messi tutti ff, che hanno la caratteristica che i bit che sono all'interno sono tutti 1, finchè c'è un byte tutti 0, che mi indica la fine del padding.



Il secondo ***RSASSA-PSS***:

Essenzialmente, del messaggio ne calcolo prima l'Hash e lo metto al centro del messaggio. All'inizio del messaggio inserisco tutti 0 e dopo l'Hash del messaggio viene messo un valore casuale (salt). Tutto questo messaggio appena prodotto verrà dato in input ad una nuova funziona Hash e l'elaborato H viene inserito all'interno del messaggio finale, prima di bc (che sono due cifre esadecimali, byte costante).




Il valore di salt viene passato ad un nuovo messaggio che è preceduto da tutti bit a 0 e l'ultimo a 1 e faccio uno XOR tra questa stringa e il valore di Hash del primo messaggio prodotto a cui viene applicata una funzione MGF (Mask generation function) che serve a mascherare il valore Hash H.

Viene fatto lo XOR e il risultato sarà la prima parte del messaggio finale

A differenza di ***RSASSA-PKCS1-v1_5*** in cui all'inizio inserivo una fissata sequenza, in questo caso la parte iniziale del messaggio è totalmente casuale che dipende dal valore salt e dalla funzione che maschera tutto.

MGF corrisponde a concatenato a 1 e

Hash(H || 0) ||
Hash(H || 1) ||
. . .

ossia è una concatenazione di vari valori (L'Hash dell'input concatenato a 0, concatenato all'Hash dell'input cosi via… fino alla fine della lunghezza)

## 10.2 UTILIZZO ElGamal PER LE FIRME DIGITALI

La sicurezza è basata sull'intrattabilità del problema del logaritmo discreto.

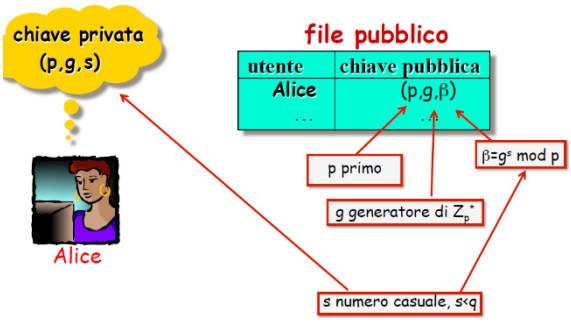
Utilizza il concetto di generatore di $Z_p^*$: p primo, g è un generatore di $Z_p^*$ se $\{g^i \mid 1 \le i \le p-1\} = Z_p^*$, in cui g si dice generatore se, preso g ed elevato a tutti i numeri compresi in p, mi permette di ottenere tutti i numeri compresi in $Z_p^*$.

| a | $a^2$ | $a^3$ | $a^4$ | $a^5$ | $a^6$ | $a^7$ | $a^8$ | $a^9$ | $a^{10}$ | $a^{11}$ | $a^{12}$ | $a^{13}$ | $a^{14}$ | $a^{15}$ | $a^{16}$ | $a^{17}$ | $a^{18}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 4 | 8 | 16 | 13 | 7 | 14 | 9 | 18 | 17 | 15 | 11 | 3 | 6 | 12 | 5 | 10 | 1 |
| 3 | 9 | 8 | 5 | 15 | 7 | 2 | 6 | 18 | 16 | 10 | 11 | 14 | 4 | 12 | 17 | 13 | 1 |
| 4 | 16 | 7 | 9 | 17 | 11 | 6 | 5 | 1 | 4 | 16 | 7 | 9 | 17 | 11 | 6 | 5 | 1 |
| 5 | 6 | 11 | 17 | 9 | 7 | 16 | 4 | 1 | 5 | 6 | 11 | 17 | 9 | 7 | 16 | 4 | 1 |
| 6 | 17 | 7 | 4 | 5 | 11 | 9 | 16 | 1 | 6 | 17 | 7 | 4 | 5 | 11 | 9 | 16 | 1 |
| 7 | 11 | 1 | 7 | 11 | 1 | 7 | 11 | 1 | 7 | 11 | 1 | 7 | 11 | 1 | 7 | 11 | 1 |
| 8 | 7 | 18 | 11 | 12 | 1 | 8 | 7 | 18 | 11 | 12 | 1 | 8 | 7 | 18 | 11 | 12 | 1 |
| 9 | 5 | 7 | 6 | 16 | 11 | 4 | 17 | 1 | 9 | 5 | 7 | 6 | 16 | 11 | 4 | 17 | 1 |
| 10 | 5 | 12 | 6 | 3 | 11 | 15 | 17 | 18 | 9 | 14 | 7 | 13 | 16 | 8 | 4 | 2 | 1 |
| 11 | 7 | 1 | 11 | 7 | 1 | 11 | 7 | 1 | 11 | 7 | 1 | 11 | 7 | 1 | 11 | 7 | 1 |
| 12 | 11 | 18 | 7 | 8 | 1 | 12 | 11 | 18 | 7 | 8 | 1 | 12 | 11 | 18 | 7 | 8 | 1 |
| 13 | 17 | 12 | 4 | 14 | 11 | 10 | 16 | 18 | 6 | 2 | 7 | 15 | 5 | 8 | 9 | 3 | 1 |
| 14 | 6 | 8 | 17 | 10 | 7 | 3 | 4 | 18 | 5 | 13 | 11 | 2 | 9 | 12 | 16 | 15 | 1 |
| 15 | 16 | 12 | 9 | 2 | 11 | 13 | 5 | 18 | 4 | 3 | 7 | 10 | 17 | 8 | 6 | 14 | 1 |
| 16 | 9 | 11 | 5 | 4 | 7 | 17 | 6 | 1 | 16 | 9 | 11 | 5 | 4 | 7 | 17 | 6 | 1 |
| 17 | 4 | 11 | 16 | 6 | 7 | 5 | 9 | 1 | 17 | 4 | 11 | 16 | 6 | 7 | 5 | 9 | 1 |
| 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 |

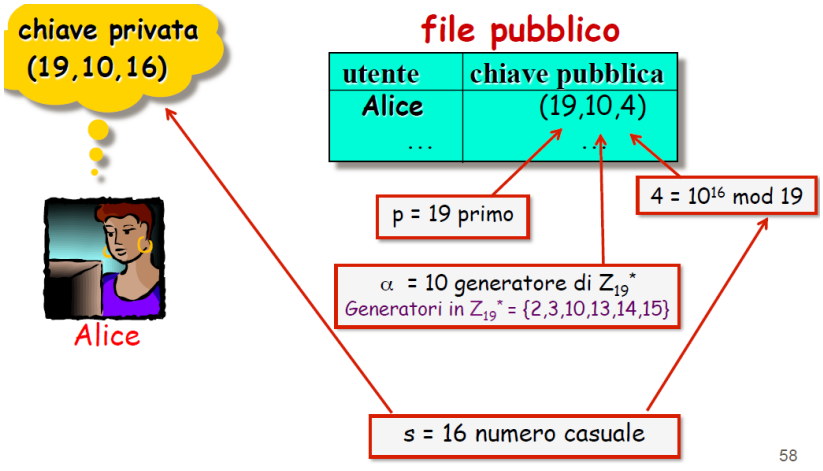
In $Z_{19}^*$ ci sono ben 6 generatori, e posso scegliere 1 di questi 6 dopo aver fissato il numero primo.

Vediamo con ElGamal come cambia il funzionamento per le firme digitali:

Con ElGamal, i valori che si mettono nella chiave pubblica e privata sono: p primo che è presente sia nella chiave pubblica che in quella privata, discorso analogo per il generatore g di $Z_p^*$. Nella chiave pubblica compare un valore $\beta$ che è il risultato di $g^s$ mod p, dove s (valore casuale) è il terzo valore che possiede Alice nella sua chiave privata.

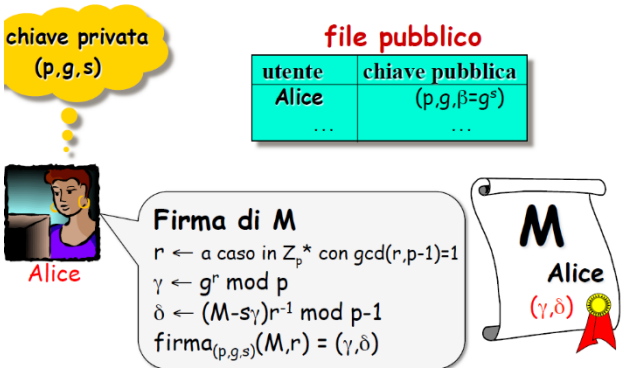



Ad esempio:



Se Alice deve firmare un messaggio M, esegue quest'algoritmo:

- Si sceglie un valore a caso r in $Z_p^*$ che è relativamente primo rispetto a p-1, cioè gcd(r,p-1)=1.
- Calcola un valore $\gamma$(GAMMA) che è uguale al generatore g elevato alla r modulo p, cioè $g^r$ mod p.
- Calcola un valore $\delta$(DELTA) che è soddisfa la seguente equazione: $(M-s\gamma)r^{-1}$ mod p-1.
- La firma finale sarà la coppia ($\gamma$, $\delta$)




Si osserva che, rispetto all'RSA in cui la firma è deterministica rispetto al messaggio, ElGamal avrà sempre una firma differente anche per 2 messaggi uguali. Questo implica che ad ogni messaggio verrà generata una firma diversa, poiché il valore r è scelto sempre a caso.

Inoltre, in RSA, la lunghezza della firma dipende dalla chiave pubblica, mentre in ElGamal, la lunghezza della firma è 2 log p.

Per verificare la firma del messaggio, esiste un algoritmo di verifica (non si entra nei dettagli):

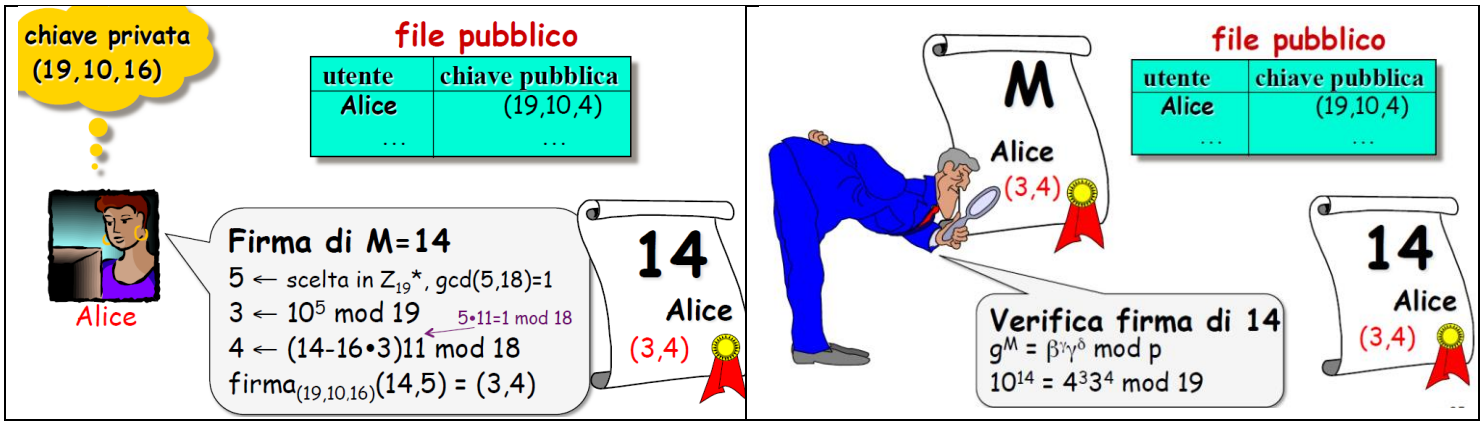
**vera se $g^M = \beta^\gamma\gamma^\delta$ mod p**
**falsa altrimenti**

Correttezza verifica firma ElGamal:

$$\beta^\gamma\gamma^\delta \bmod p = g^{s\gamma}\gamma^\delta \bmod p \quad (\beta = g^s \bmod p)$$
$$= g^{s\gamma}g^{r\delta} \bmod p \quad (\gamma = g^r \bmod p)$$
$$= g^{s\gamma}g^{r(M-s\gamma)r^{-1}} \bmod p \quad (\delta = (M+s\gamma)r^{-1} \bmod p-1)$$
$$= g^{s\gamma}g^{M-s\gamma} \bmod p \quad (r\,r^{-1} = 1 \bmod p-1)$$
$$= g^M \bmod p$$

Teorema di Eulero: $x \in Z_n^* \Rightarrow x^{\phi(n)}=1 \bmod n$ quindi $g^{p-1}=1 \bmod p$

Un esempio:




Come detta in precedenza, la sicurezza di ElGamal si basa sull'intrattabilità del logaritmo discreto in $Z_p^*$, e la lunghezza della firma è un po' grande, se volessi garantire la stessa sicurezza di RSA con ElGamal, dovrei utilizzare il doppio della lunghezza e non va bene.

L'algoritmo DSS modifica qualche caratteristica, infatti la lunghezza della firma non sarà 2 log p, ma lo cambia in maniera tale che la lunghezza della firma diventi 2 log q, dove q ha una caratteristica in particolare, e cioè che è molto più piccolo di p.

A differenza di ElGamal in cui è presente un generatore, in DSS si costruisce un elemento che ha un certo ordine, cioè che l'ordine deve sempre dividere p-1, per cui le computazioni dell'esponente diventano più piccole.

## 10.3 UTILIZZO DSS PER LE FIRME DIGITALI

DSS e DSA sono equivalenti, è stato sottoposto a varie revisioni nel tempo.

È una modifica ingegnosa dello schema di firme ElGamal, la sicurezza basata sull'intrattabilità del logaritmo discreto. Le firme DSS sono piccole (buone per smart card).

Usa numeri primi p e q di lunghezza L ed N, le computazioni avvengono nel seguente modo:

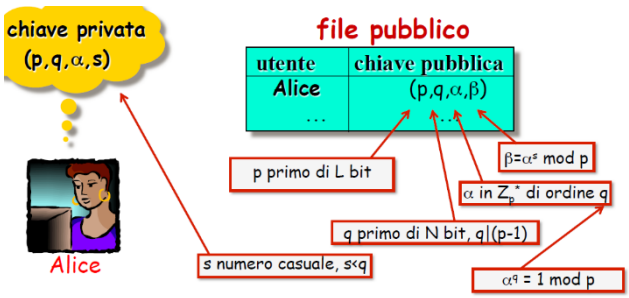
Si osserva che la firma può essere di 320,448,512,512 bit, a seconda dei valori L e N scelti. La lunghezza della firma è migliore rispetto a ElGamal.

| L=\|p\| | N=\|q\| | \|firma\| | |
|---|---|---|---|
| 1.024 | 160 | 320 | Unica scelta in FIPS 186-1 e 186-2 |
| 2.048 | 224 | 448 | Aggiunti nel FIPS 186-3 |
| 2.048 | 256 | 512 | |
| 3.072 | 256 | 512 | |

Lo schema di utilizzo è il seguente:

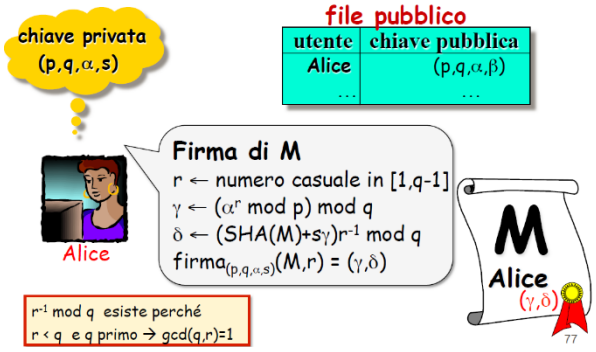
La chiave pubblica si compone si (p,q,$\alpha$,$\beta$), dove $\alpha$ è un elemento di ordine q, $\beta$ è un elemento dato da $\alpha^s$ mod p, dove s è la chiave privata. L'idea è simile al ElGamal, solo che al posto del generatore, c'è un elemento $\alpha$ di un certo ordine, e q che è l'ordine di quell'elemento.

L'ordine di un elemento, è il più piccolo valore tale che, quel valore elevato all'ordine, da 1. Ad esempio, in 4, il numero che devo elevare per ottenere il primo numero è 9 (nella tabella delle potenze di $Z_{19}^*$), a differenza dell'ordine di un generatore che è p-1.




La firma si basa sulla stessa struttura di ElGamal:

Le operazioni vengono però fatte tutte in modulo q, quindi più piccole infatti $\gamma$ e $\delta$ che rappresentano la coppia della firma sono entrambi ridotti in modulo q. All'interno dello standard DSA è inserito anche SHA(M), che rappresenta una funzione Hash rispetto al messaggio M.




La verifica della firma è simile ad ElGamal, solo che le operazioni sono modulo q.

**Verifica firma di M**

$e' \leftarrow SHA(M)\delta^{-1} \bmod q$

$e'' \leftarrow \gamma\delta^{-1} \bmod q$

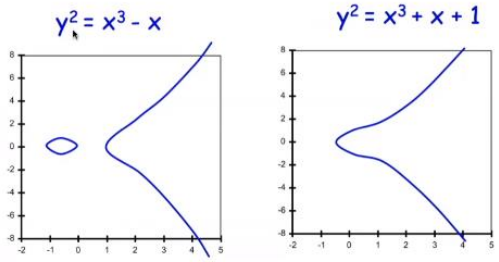
vera se $\gamma = (\alpha^{e'}\beta^{e''} \bmod p) \bmod q$

falsa altrimenti

Nelle nuove versioni dell'DSS, in particolare ECDSA, le computazioni vengono fatte su curve ellittiche, cioè viene usato un punto su curva ed il suo ordine. Questo significa che le operazioni non vengono fatte in $Z_p^*$ ma su queste particolari equazioni cubiche

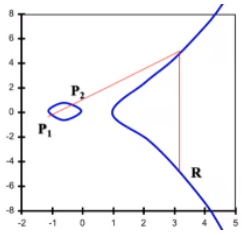
Una curva ellittica, in matematica, è definita da un'equazione cubica di questa forma: $y^2 = x^3 + ax + b$ dove a e b sono 2 costanti. Oltre a x e y, vengono inclusi altri 2 punti: punto all'infinito o punto zero.

Ad esempio:

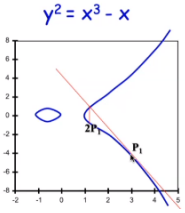



Questi due disegni sono sempre simmetrici all'asse dello 0 perché c'è il valore $y^2$, variando il coefficiente si ottengono curve simili. Su queste curve posso definire operazioni di addizione e sottrazione. L'addizione si definisce nel seguente modo:

Unisco con una retta i punti $P_1$ e $P_2$ e proseguo fino a che non tocco la curva generando un terzo punto. Su questo terzo punto prendo il valore simmetrico rispetto all'asse orizzontale. Il valore R è la somma di $P_1$ e $P_2$. Chiaramente esiste una formula matematica per calcolare R.




La moltiplicazione funziona nel seguente modo: se prendo un punto $P_1$, per calcolarne il doppio, prendo la tangente che passa per quel punto, interseca la curva e prendo l'opposto




$2P_1 = P_1 + P_1$

Le operazioni che vengono eseguite per i nostri algoritmo avvengo in modulo p. L'approccio geometrico visto prima non è più valido.

Viene riepilogata una tabella delle lunghezze:

Bisogna tener conto dell' expected security life

Esempi:

- Fatto nel 2005
- Expected security life = 5 anni →

- Fatto nel 2005
- Expected security life = 6 anni →

**Table 4: Recommended algorithms and minimum key sizes**

| Algorithm security lifetimes | Symmetric key algorithms (Encryption & MAC) | FFC (e.g., DSA, D-H) | IFC (e.g., RSA) | ECC (e.g., ECDSA) |
|---|---|---|---|---|
| Through 2010 (min. of 80 bits of strength) | 2TDEA[23]<br>3TDEA<br>AES-128<br>AES-192<br>AES-256 | Min.:<br>$L = 1024$;<br>$N = 160$ | Min.:<br>$k$=1024 | Min.:<br>$f$=160 |
| Through 2030 (min. of 112 bits of strength) | 3TDEA<br>AES-128<br>AES-192<br>AES-256 | Min.:<br>$L = 2048$<br>$N = 224$ | Min.:<br>$k$=2048 | Min.:<br>$f$=224 |
| Beyond 2030 (min. of 128 bits of strength) | AES-128<br>AES-192<br>AES-256 | Min.:<br>$L = 3072$<br>$N = 256$ | Min.:<br>$k$=3072 | Min.:<br>$f$=256 |

NIST SP 800-57,
"Recommendation for Key Management, Part 1: General (Revised)", Marzo 2007

La firma digitale crea alcune problematiche per l'utilizzo che sono:

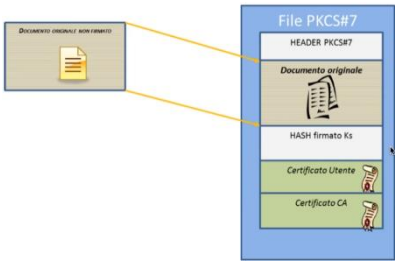
- Certezza legame chiave pubblica ed utente.
- Legislazione e valore legale (in Italia).
- Firme multiple e formati.
- Conservazione ed utilizzo chiave privata.
- Ottenere firma digitale.

Esistono sistemi di firme come la firma grafometrica in cui si firma su un tablet e che ha valore legale.

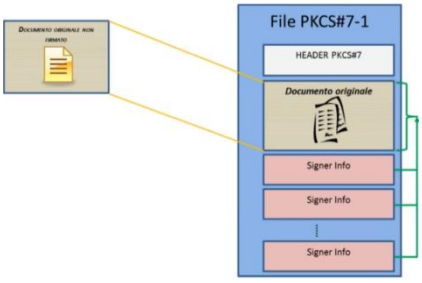
I due firmati di firme che vengono spesso utilizzati sono:

- Firma **CAdES**: Sono file con estensione .p7m, si possono firmare file Word, Excel, OpenOffice, jpeg, gif, png, pdf. Occorre un software specifico per poter fare una firma con questo formato, e una volta firmato, non si può visualizzare il file. La firma è rappresentata da una stringa binaria indipendente dal testo. Naturalmente la firma può essere controllata, ma non visualizzata. Poiché si firma qualsiasi tipo di documento, è detto anche universale.
- Firma **PAdES**: Sono file con estensione .pdf, consentono di firmare solo file pdf, e possono essere visualizzati attraverso un opportuno reader pdf.

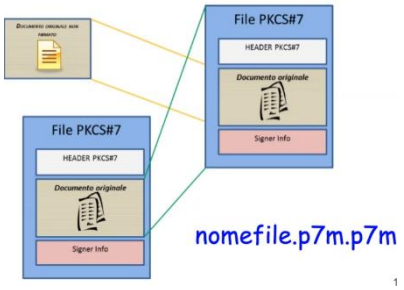
La firma CAdES viene definita nello standard PKCS#7 che definisce:




All'interno di un file PKCS#, c'è naturalmente il documento originale, preceduto da una sintassi dell'HEADER dello standard. Successivamente al documento c'è la stringa HASH della firma dell'utente, seguita da un certificato della chiave pubblica con cui viene firmato il documento, e poi c'è un certificato CA. Avendo uno standard che fornisce le seguenti regole, si può garantire l'autenticità delle firme mediante una serie di controlli.

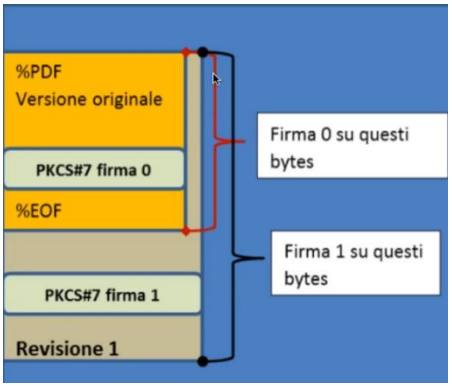



Per quanto riguarda le firme multiple, il concetto diventa più complesso e ci sono due opzioni: nella prima il formato si compone dell'HEADER, del documento, e più slot al cui interno saranno presenti le firme di più utenti. Per poter effettuare una verifica delle firme, si deve effettuare un controllo dalla prima all'ultima firma in ordine sequenziale.




La seconda possibilità di una firma multipla, viene definito firma a matriosca CAdES, in cui, se più utenti devono firmare in maniera congiunta, vengono generati più file PKCS#7 per la firma di ogni utente. Se ad esempio firmano due persone, il nome del file avrà una doppia estensione .p7m per indicare che la firma è stata congiunta.

Per quanto riguarda le firme multiple mediante lo standard PAdES, l'idea è la seguente:

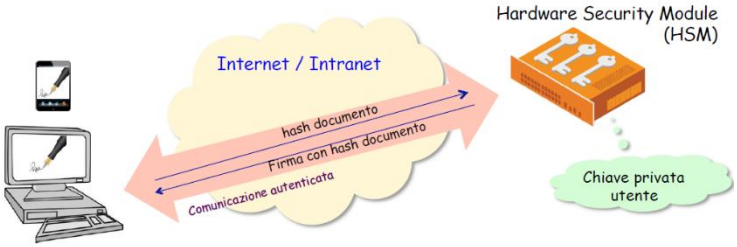



La prima persona che firma, avrà la firma all'interno del file originale. Per garantire un nuova firma per un nuovo utente, si procede con un'estensione del file, ed in generale, si genera un solo file pdf contenente tutte le firme.

Quando si effettua una firma all'interno del proprio pc, la sicurezza potrebbe essere compromessa dalla presenza di malware. Un problema potrebbe essere il seguente: si supponga si stia firmando un file word (ma di qualsiasi estensione anche), e nel file word ci sono delle macroistruzioni (codice eseguibile), se firmassi il formato word, potrebbe succedere che, a causa di codice eseguibile, venga alterata la firma dell'utente e di conseguenza, in fase di controlli, giudicata non valida. Come si risolve un problema di questo tipo? La prima possibilità è che l'utente deve verificare se c'è codice eseguibile, ma naturalmente non tutti gli utenti sono in grado di poter effettuare un controllo del genere. Quindi si usa la seconda possibilità in cui si decide di restringere i formati permessi (ASCII, PDF/A, immagini), limito le firme quindi a pochi formati. Inoltre sono stati stabiliti degli articoli secondo cui una firma digitale non produce gli effetti giuridici desiderati se il documento "contiene macroistruzioni o codici eseguibili, tali da attivare funzionalità che possano modificare gli atti, i fatti o i dati nello stesso rappresentati". Si utilizzano quindi dei formati che non consentono la presenza di macroistruzioni, ad esempio PDF/A, dove A sta per Archive, in cui l'archiviazione è assicurata per un lungo periodo, contiene tutte le informazioni senza link e sono vietati: JavaScript, invocazioni di codice eseguibile, cifratura, link a contenuti esterni.

Ci sono delle problematiche riguardanti firme digitali con smart card, quali: installazione e aggiornamenti del software della firma, installazione e configurazione dei driver del lettore di smart card, ripetere la procedura per ognuno dei computer utilizzato dallo stesso titolare, ricordarsi di gestire il rinnovo dei certificati digitali entro la data di scadenza (generalmente ogni 3 anni). Quello che di conseguenza sta prendendo piede, è di remotizzare la firma digitale, cioè che la firma digitale non viene fatta utilizzando smart card, viene fatta invece da qualche server particolare che possiede le chiavi privata. Uno schema di quanto appena spiegato è il seguente:

Dato un utente che si trova nella sua postazione, e deve firmare un documento, invia l'hash del suo documento ad un server, che possiede la chiave privata dell'utente, il quale server firmerà l'hash mandato dall'utente e rispedirà all'utente la firma e l'utente la potrà utilizzare per firmare il documento. Naturalmente devono esserci le dovute accortezze per poter effettuare un'operazione del genere, infatti la comunicazione appena descritta deve essere autenticata, per garantire sicurezza sull'operazione fatta. È necessario anche che il Server che gestisce le chiavi private sia ben sicuro, e infatti si trova in una macchina definita HSM (Hardware Security Module), che garantisce che nessuno, nemmeno l'amministratore, può leggere la chiave dell'utente. In questo documento abbiamo detto che generalmente è l'utente che possiede la sua chiave privata, mentre in questa situazione, la si da ad un'autorità fidata che la gestisce. In questo modo l'utente può effettuare le firme in un qualsiasi momento e in maniera molto semplice, senza dover usare smart card, perché la firma si trova sul server. La macchina HSM gestisce le generazioni delle chiavi, senza collegamenti all'esterno, e ad ogni utente fornisce la chiave pubblica che l'utente deve utilizzare se desidera avere un collegamento al server. L'HSM ha diverse caratteristiche, molte di protezione (ad esempio, in caso di manomissione sono previsti sistemi che garantiscono la formattazione immediata dei dischi in modo da perdere tutte le informazioni), ed è molto veloce (riesce a fare 7000 firme RSA 1024 bit al secondo).




Questa soluzione viene anche utilizzata per la firma degli esami:

Il funzionamento è il seguente: naturalmente c'è una fase di login e password per i docenti e gli studenti. Poi c'è la compilazione verbale e la firma. Una volta che tutti gli studenti accettano la loro votazione per un esame, quando il docente deve verbalizzare tutti gli esami fatti, genera un verbale e manda l'hash all'HSM e in questo server avviene la firma digitale che poi viene visualizzata al docente e di conseguenza notificato ad ogni studente. Un'idea del genere, per com'è stata spiegata, potrebbe avere delle debolezze, in quanto un attaccante potrebbe conoscere le informazioni di login di un docente e potrebbe verbalizzare esami a studenti che effettivamente non hanno sostenuto alcuna prova. Per rimediare a ciò, per essere sicuri che sia effettivamente il Docente ad effettuare il processo di verbalizzazione, il sistema invia un SMS con un codice (una serie di numeri) al docente che deve inserire nella procedura di firma. Questa firma è naturalmente casuale e cambierà sempre. In questo modo, se un attaccante volesse verbalizzare esami fittizi a studenti, dovrebbe possedere anche il cellulare del docente per poter effettuare la firma digitale.