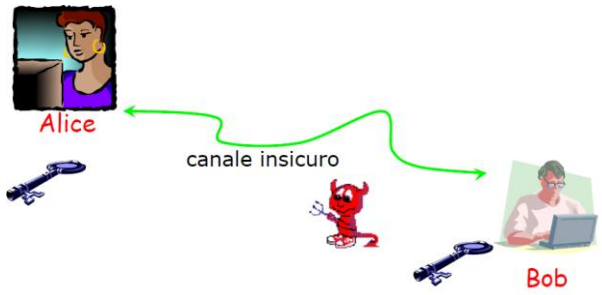
8.0 CIFRARI SIMMETRICI



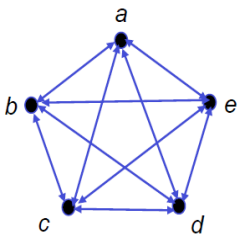
Nei cifrari simmetrici, chi invia un messaggio, e chi riceve il messaggio, condivide la stessa chiave. Il messaggio viene cifrato simmetricamente.

Il problema che sorge, è come fanno le 2 persone a condividere la chiave comune? Si potrebbe pensare di usare un canale privato (corriere fidato), ma come si può soddisfare questo criterio su rete internet?

Un secondo problema si ha se si vuole cifrare in un gruppo di n persone. Ad esempio, date 5 persone che devono scambiare tra loro messaggi cifrati, ciò implica che tutte le persone, a due a due, devono condividere una chiave segreta diversa da tutte le altre, per garantire sicurezza.

Ogni utente deve memorizzare dunque n-1 chiavi. Il numero totale delle chiavi segrete è n(n-1) / 2. Nel caso di 5 persone, ci vogliono dunque 10 chiavi.

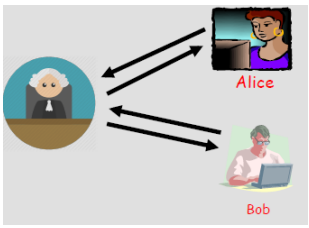
È una soluzione poco scalabile questa, poiché se entra una nuova persona, la nuova persona deve avere una chiave per ognuno degli utenti già presenti, e agli utenti già presenti è necessario dare la nuova chiave per avere contatto con la sesta persona.



10 chiavi segrete: tra *(a,b), (a,c), (a,d), (a,e), (b,c), (b,d), (b,e), (c,d), (c,e), (d,e)*

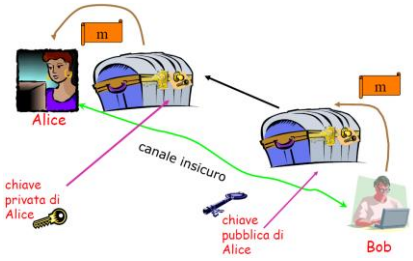
Una soluzione a questi problemi è l'uso di una terza parte fidata che stabilisce la chiave di sessione e la invia ad entrambi in modo sicuro. Nella figura si vede come Alice e Bob possiedano una chiave privata collegata alla terza parte (la quale quindi la conosce). Se Alice e Bob vogliono comunicare, lo dicono alla parte fidata, quest'ultima genera una chiave di sessione temporanea e ne manda la cifratura mediante la chiave privata ad Alice e Bob.

Viene risolto il problema della scalabilità, ma non sono risolti problemi quali carico, ma specialmente la sicurezza, poiché la terza parte deve essere realmente fidata. La terza parte dovrebbe essere anche sempre online per generare la chiave, tutto ciò può creare ulteriori problemi.



Per cifrare in un ambiente distribuito come la rete, si usano cifrari asimmetrici.

8.1 CIFRARI ASIMMETRICI

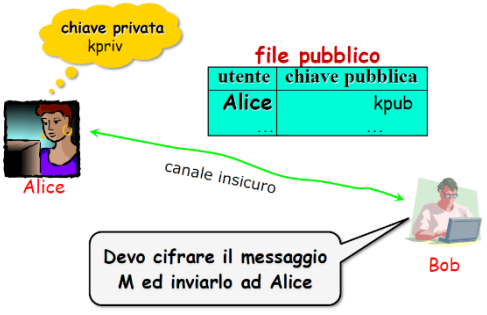


L'idea è di usare due chiavi: una chiave pubblica che serve per cifrare il messaggio, e la chiave privata che serve per decifrare il messaggio per poterne leggere il contenuto. Naturalmente chiave pubblica e privata sono diverse. La chiave pubblica come fa intendere il nome può essere conosciuta da tutti, mentre la privata dal singolo utente.

Dal punto di vista digitale il cifrario asimmetrico viene costruito nel seguente modo: c'è un file pubblico in cui ogni utente possiede la sua chiave pubblica. Contemporaneamente alla costruzione della chiave pubblica, viene costruita anche la chiave privata che tiene per se.

Se qualche utente vuole mandare un messaggio ad Alice, va nel file pubblico a vedere la chiave pubblica di Alice, e quello che fa, è usare un algoritmo di cifratura ***CIFRA (kpub, M) -> C***, in cui kpub è la chiave pubblica di Alice, M è il messaggio in chiaro. L'algoritmo CIFRA genera una messaggio cifrato (sequenza di bit) C che viene mandato ad Alice.

Una volta mandato il testo cifrato ad Alice, questo può essere visualizzato da Alice e solo Alice può decifrare il messaggio, mediante la sua chiave privata. Viene quindi utilizzato un algoritmo di decifratura ***DECIFRA (kpriv, C) -> M***, in cui kpriv è la chiave privata di Alice, C è il messaggio cifrato che ha ricevuto, M è il messaggio decifrato che ha ricevuto e che può leggere.

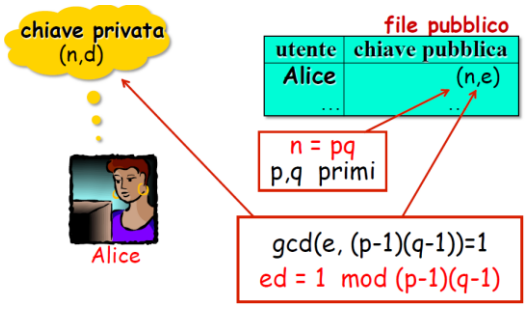


In questo caso si vede come non ci sono chiavi condivise tra utenti.

Per la realizzazione dei cifrari asimmetrici, l'idea è di utilizzare **Funzioni one-way**, che sono funzioni "facili" da calcolare e "difficili" da invertire. Naturalmente per facilità e difficoltà, s'intende il tempo per effettuare il calcolo. In informatica, nessuno ha mai dimostrato che esistono funzioni che soddisfano il requisito di one-way. Ci si basa quindi su assunzioni di difficoltà, ossia quanto è facile/difficile eseguire una operazione.

Non basta che una funzione sia one-way, ma che nessuno deve conoscere una "trapdoor", poiché la sua conoscenza rende facile la decifratura.

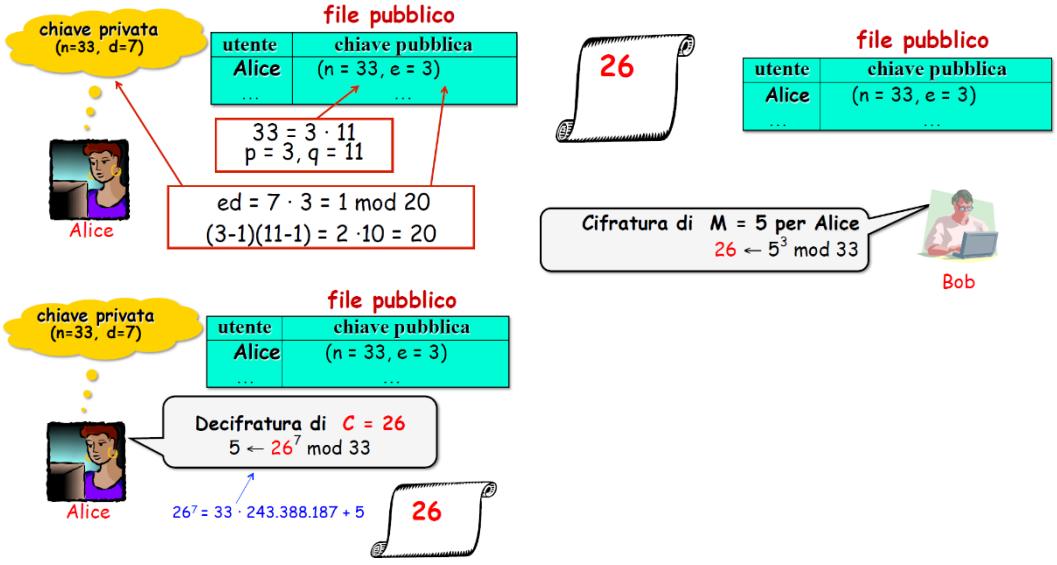
8.1.0 RSA



L'idea è descritta in foto. Come detto in precedenza, Alice costruisce contemporaneamente la sua chiave pubblica e privata. Per la costruzione, costruisce due numeri primi casuali ***p*** e ***q*** che devono essere abbastanza grandi, ne calcola il prodotto che salva in ***n***. Questo ***n*** lo mette come primo elemento della chiave pubblica e primo elemento della privata. Successivamente calcola due valori: ***e***, ***d*** che hanno una particolare caratteristica e cioè che il loro prodotto è congruo a 1 mod (p-1)(q-1). Inoltre per calcolare questi due valori si procede nel seguente modo, il quale rappresenta un prerequisito: viene scelto un valore ***e***, il quale deve essere "validato" nel seguente modo, ossia che il Massimo Comune Divisore (GCD) di ***e*** e (p-1)(q-1) deve essere uguale a 1. Una volta soddisfatta questa equazione e questa validazione di ***e***, il valore ***d*** opportunamente calcolato, diventa secondo membro della chiave privata, mentre ***e*** diventa secondo membro della chiave pubblica.

Supponendo come visto prima che un utente deve cifrare un messaggio M ed inviarlo ad Alice, osserva la chiave pubblica di Alice (***n***,***e***) ed esegue la seguente operazione: $M^{e} \bmod n \rightarrow C$. Il messaggio cifrato **C** viene mandato ad Alice, la quale per decifrarlo esegue la seguente operazione: $C^{d} \bmod n \rightarrow M$. Questa operazione offre ad Alice il messaggio in chiaro, mandato da un qualsiasi utente.

Viene mostrato un esempio:



OSSERVAZIONI:

Se Alice volesse scegliere come chiave pubblica (n=33, e=2):

- È una buona scelta? n è il prodotto di due primi (11 e 3), viene poi scelto e=2, ma una scelta del genere non è adatta perché è stato scelto un numero tale che il gcd(2, 20) è diverso da uno. Quindi non è stato soddisfatto il prerequisito, in quanto sappiamo che deve essere 1.
- Analizziamo la seguente immagine:

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$M^2 \bmod 33$	1	4	9	16	25	3	16	31	15	1	22	12	4	31	27	25

17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
25	27	31	4	12	22	1	15	31	16	3	25	16	9	4	1

  Vengono mostrati tutti i possibili messaggi da 1 a 32 (poiché l'operazione è modulo 33) e ognuno dei messaggi è stato cifrato (naturalmente tenendo conto che e=2). Viene quindi sostituito ogni numero a M per ottenere il corrispettivo cifrato. Cosa si nota? Assumiamo debba essere cifrato 13, il suo testo cifrato è 4. Quanti testi in chiaro ci sono per 4? Ne osserviamo ben 4 (2-13-20-31), questo implica che quando vado a decifrare non so a quale dei 4 messaggi devo associare la mia decifratura, infatti ci sono 4 valori possibili.

Se Alice ha scelto come chiave pubblica (n=33, e=3):

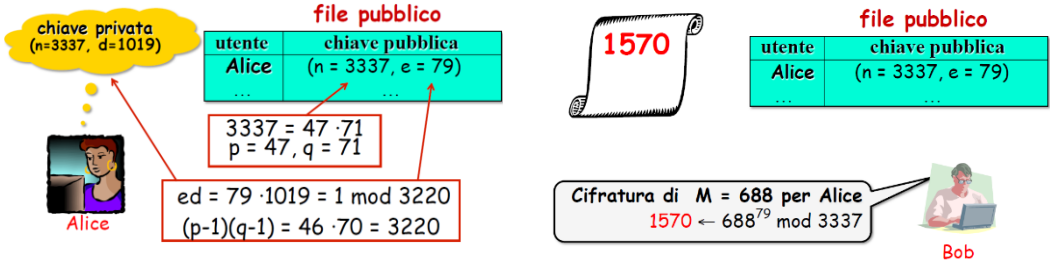
Naturalmente la e è stata scelta correttamente in quanto il gdc(3,20)=1. Questo implica che:

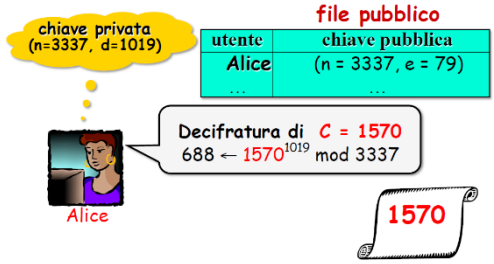
M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$M^3 \bmod 33$	1	8	27	31	18	18	13	17	3	10	11	12	19	5	9	4

17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
29	24	28	14	21	22	23	30	16	20	15	7	2	6	25	32

Per ogni valore cifrato, la decifratura corrisponde ad un singolo valore, ciò dipende esclusivamente dal gdc(3,20)=1

Un altro esempio:





Ulteriore esempio:

Primi p e q di 512 bit

p
961303453135835045741915812806154279093098455949962158225831508796
479404550564706384912571601803475031209866660649242019180878066742
1096063354219926661209

q
120601919572314469182767942044508960015559250546370339360617983217
314821484837646592153894532091752252732268301071206956046025138871
455249690003596600456l7

n=pq
115935041739676149688925098646158875237714573754541447754855261376
147885408326350817276878815968325168468849300625485764111250162414
552339182927162507656772727460097082714127730434960500556347274566
628060099924037102991424472292215772785317270338393813346926841373
276220009666766718318310883734208234443709553

(p-1)(q-1)
115935041739676149688925098646158875237714573754541447754855261376
147885408326350817276878815968325168468849300625485764111250162414
552339182927162507656751054233608492916752034482627988117554787657
013923444405716989581728196098226361075467211864612171359107358640
614008885170265377277264467341066243857664128

e
35535

d
580083028600377639360936612896779175946690620896509621804228661113
805938528223587317062869100300217108590443384021707298690876006115
306202524959884448047568240966247081485817130463240644077704833134
010850947385295645071936774061197326557424237217617674620776371642
076003370853332885321447088595513667029483l

Messaggio THIS IS A TEST (codifica da 00 a 25, con 26="spazio")

M
THIS IS A TEST
1907081826081826002619041819

C=M^e
475309123646226827206365550610545180942371796070491716523239243054
452960613199328566617843418359114151197411252005682979794571736036
101278218847892741566090480023507190715277185914975188465888632101
148354103361657898467968386763733765777465625079280521148141844048
141844308127730590046928742485591664621086556

C^d
1907081826081826002619041819

Dove n=pq è naturalmente a 1024 bit che oggi giorno non è più usato, infatti al giorno d'oggi p e q sono di 1024 bit.

Il problema difficile su cui si basa l'RSA è la fattorizzazione dei numeri primi. Calcolare la fattorizzazione di 1024 bit è un problema difficile. Non è stata infatti mai fatta.

<u>PER CONTINUARE QUESTO CAPITOLO, LEGGERE IL WORD PRECEDENTE 7.Elementi di Teoria dei Numeri</u>

Dimostriamo la correttezza della decifratura dell'RSA:

$$C^d \bmod n = (M^e)^d \bmod n$$
$$= M^{ed} \bmod n \quad (ed = 1 \bmod (p-1)(q-1))$$
$$= M^{1+k(p-1)(q-1)} \bmod n$$
$$= M \cdot (M^{(p-1)(q-1)})^k$$
$$= M \bmod n \quad (\text{Teorema di Eulero: } M \in Z_n^* \Rightarrow M^{(p-1)(q-1)} = 1 \bmod n)$$
$$= M \quad (\text{Per } M \in Z_n/Z_n^* \text{ usa il teorema cinese del resto; poichè } 0 \le M < n)$$

C, corrisponde al messaggio M elevato alla e. Un'elevazione e una seconda elevazione corrisponde a M^{ed}, ma ed abbiamo detto che equivale a 1+k(p-1)(q-1). M elevato a questo valore, poiché ce un +, lo posso vedere come prodotto. $M^{(p-1)(q-1)}$ per il teorema di Eulero che ci dice che quando prendo un elemento e lo elevo alla Φ(n) ottengo 1. Quindi resta solo M che è il messaggio in chiaro. Quindi la decifrazione è sempre corretta.

In realtà questa dimostrazione non vale per tutti i valori, vale infatti per i valori per cui vale il teorema di Eulero, cioè tutti i messaggi relativamente primi rispetto a N. Cosa succede se un valore non è relativamente primo rispetto a N? L'RSA funziona ugualmente, solo che la dimostrazione dovrebbe continuare con il teorema cinese del resto (ma non viene illustrata).

Facciamo ora un discorso relativo all'efficienza delle computazioni dell'RSA:

La generazione dei numeri primi p e q avviene efficientemente.

La generazione di e,d avviene nel seguente modo: Genero prima e e poi d, come inverso moltiplicativo di e. Tutto ciò in maniera efficiente

L'elevazione a potenza modulare per cifratura e decifratura avviene efficientemente.

Dal punto di vista reale, se voglio usare l'RSA si utilizza la seguente impostazione:

1. Input L (lunghezza modulo)
2. Genera 2 primi di lunghezza L/2
3. $n \leftarrow p \cdot q$
4. Scegli a caso e
5. **If** gcd (e, (p-1)(q-1)) = 1
   then $d \leftarrow e^{-1} \bmod (p-1)(q-1)$
   else goto 4.

L'input L deve essere abbastanza grande. Quando genero L, la prima cosa che faccio è esplicitare la lunghezza del modulo (ossia, voler utilizzare l'RSA-1024 bit, o RSA-2048 bit e così via). Fissata la lunghezza, genero le chiavi con quella lunghezza. Supponiamo di volere RSA-2048 bit, la lunghezza del modulo è 2048. Per generare i valori di RSA, genero 2 primi di lunghezza L/2 ognuno. Una volta generati calcolo n. Scelgo a caso e. Se il gcd è 1 allora scelgo d che è l'inverso moltiplicativo di e, mediante teorema di eulero esteso. Altrimenti ritorno al punto 4.

Quando viene scelto un valore, si ha un problema, e cioè che l'esponente e è un esponente pubblico utilizzato per la cifratura. La cifratura avviene in modo efficiente, però se scegliessi l'esponente pubblico ancora più piccolino, per esempio e=3 oppure e= $2^{16}+1$ le operazioni di cifratura diverrebbero ancora più facili. Se l'esponente pubblico fosse quindi $2^{16}+1$ (che in binario si scrive 10000000000000001), quando vado a cifrare, il messaggio viene quindi elevato a $2^{16}+1$ che ha solo 2 uni. Esiste un algoritmo che cifra che diventa molto efficiente con questo valore: il right-to-left. Poiché quindi se si utilizzasse $2^{16}+1$ si risparmia tanto in tempo computazionale, l'algoritmo varia leggermente per garantire l'inizializzazione di e a $2^{16}+1$ (questo lo faccio perché in termini di sicurezza, avere un esponente o un altro non altera):

1. Input L (lunghezza modulo)
2. $e \leftarrow 3$ oppure $e \leftarrow 2^{16}+1$
3. Genera 2 primi di lunghezza L/2
4. $n \leftarrow p \cdot q$
5. **If** gcd (e, (p-1)(q-1)) = 1
   then $d \leftarrow e^{-1} \bmod (p-1)(q-1)$
   else goto 3.

Scelgo come input la lunghezza del modulo, successivamente fisso l'esponente, genero 2 primi di lunghezza L/2 e così via come prima… con la differenza che in questo caso, nell'else ritorno al punto 3, perché devo generare 2 primi che devono essere relativamente primi rispetto a 2^{16+1}.

Analizziamo la **Sicurezza dell'RSA:**

Ci sono 2 tipi di attacchi: ***attacco sulla generazione della chiave*** e ***attacco rispetto alla cifratura***.

Riguardo un attacco sulla ***generazione della chiave***, supponendo di conoscere la chiave pubblica (n,e), l'attaccante vuole calcolare la chiave privata d. Per questo calcolo, può optare per diversi attacchi:

Attacco 1: fattorizzare n, infatti se n viene fattorizzato, il calcolo di d può essere fatto mediante algoritmo di Euclide esteso, secondo questi step:

1. Fattorizza n
2. Computa $\varphi(n)=(p-1)(q-1)$
3. Computa $d \leftarrow e^{-1} \bmod (p-1)(q-1)$

Attacco 2: computare $\Phi(n)$. Se l'utente può calcolare $\Phi(n)$ riuscirebbe a fattorizzare n nel seguente modo:

$$\begin{cases} n = pq \\ \varphi(n) = (p-1)(q-1) \end{cases}$$

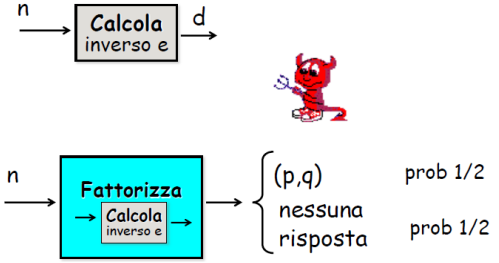
sostituendo $p = n/q$
$p^2 - (n-\varphi(n)+1)p + n = 0$

$$\frac{(n-\phi+1) \pm \sqrt{(n-\varphi(n)+1)^2 - 4 \cdot n}}{2}$$

In questo sistema ci sono due incognite: p e q. Da queste due equazioni, in particolare dalla prima, ricavo p come n/q, e lo vado a sostituire alla seconda equazione del sistema. Ottengo un'equazione di secondo grado in p, i cui 2 valori che calcolo sono proprio p e q. Il calcolo è il classico dell'equazione di secondo grado.

Attacco 3: computare d, infatti se si riesce a computare d, si può fattorizzare n. Esiste infatti un algoritmo (non illustrato) che trasforma d con input (n,e) in un algoritmo che fattorizza n con probabilità $\geq$ ½. Tali algoritmi che hanno questa caratteristica vengono chiamati ***Las Vegas***.

L'algoritmo prende in input (n,e) e calcola l'inverso di e, cioè d. Questo valore può essere utilizzato come oracolo all'interno di un algoritmo più complesso che dà in output la coppia (p,q) con probabilità ½ e nessuna risposta con probabilità ½. La probabilità non dipende dall'input ma da scelte interne all'algoritmo.



Per questi 3 attacchi spiegati si nota come ricorre sempre il concetto della fattorizzazione. Quanto è complesso il problema della fattorizzazione?

Un primo algoritmo, più semplice, è il seguente:

Per tutti i primi p in [2, $\sqrt{n}$], se p|n allora p è fattore di n.

Esempio:

n=77 (7*11), numeri primi in [2, $\sqrt{77}$]= 2,3,5,7

n=143 (11*13), numeri primi in [2, $\sqrt{143}$]= 2,3,5,7,11

Perché nell'algoritmo considero tutti i primi fino a $\sqrt{n}$? Perché se continuassi oltre a $\sqrt{n}$, poi si ricomincerebbero a ripetere. L'algoritmo cerca infatti il fattore o i fattori più piccoli. Il fattore più piccolo di n non può essere maggiore di $\sqrt{n}$, altrimenti il loro prodotto sarebbe più grande di n. Questo significa che almeno un fattore deve essere più piccolo di $\sqrt{n}$.

La complessità di questo algoritmo, se lo volessi implementare, dipende dalla lunghezza del valore dato in input. Se è piccolo è un'operazione semplice, se fosse un valore di 2048 bit e quindi voglio fare la fattorizzazione di 2 valori entrambi di 1024 bit, fattorizzare questi numeri implica come massimo $\sqrt{2^{2048}}$ ossia 2^{1024} e quindi è abbastanza pesante come algoritmo.

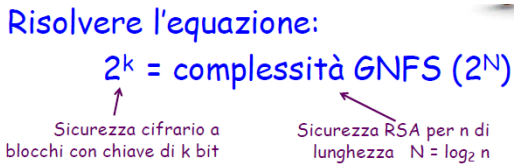
Esistono ulteriori algoritmi più complessi per la fattorizzazione, che hanno complessità: numero di nepero e, elevato ad altri valori che coinvolgono q, ossia il numero più piccolo. Una complessità di tipo esponenziale quindi.

La lunghezze dell'RSA che sono equivalenti agli algoritmi AES sono le seguenti:

Security Strength	Symmetric key algorithms	FFC (e.g., DSA, D-H)	IFC (e.g., RSA)	ECC (e.g., ECDSA)
≤ 80	2TDEA	$L = 1024$ $N = 160$	$k = 1024$	$f = 160\text{-}223$
112	3TDEA	$L = 2048$ $N = 224$	$k = 2048$	$f = 224\text{-}255$
128	AES-128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256\text{-}383$
192	AES-192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384\text{-}511$
256	AES-256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$

Non approvati per la protezione di informazioni del Governo Federale USA
Inclusi negli standard NIST
Non ancora inclusi negli standard NIST per motivi di interoperabilità e di efficienza

Come si stabilisce l'equivalenza (per la sicurezza) tra le lunghezze delle chiavi pubbliche e simmetriche? Bisogna di risolvere la seguente equazione:



Analizziamo ora un **attacco rispetto alla cifratura**, ossia, conoscendo la chiave pubblica (n,e) e il messaggio cifrato C che è uguale a M^e mod n, l'attaccante vuole calcolare il messaggio M.

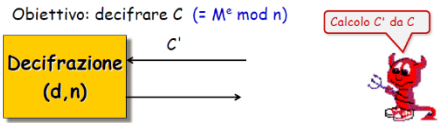
Se l'attaccante potesse fattorizzare n potrebbe anche computare M nel seguente modo:

1. Fattorizza n
2. Computa $\varphi(n)=(p-1)(q-1)$
3. Computa $d \leftarrow e^{-1} \bmod (p-1)(q-1)$
4. Ricava M decifrando C

La cosa importante è che se venisse trovato un algoritmo che dato un testo cifrato, genera il corrispettivo in chiaro, questo è equivalente a fattorizzare? Nessuno ha mai dimostrato che esiste un algoritmo che fattorizza, utilizzando come oracolo un algoritmo che decifra un testo cifrato. In linea teorica quindi si potrebbe decifrare un messaggio M senza fattorizzare, ma è una idea teorica poiché al giorno d'oggi nessuno ha mai fatto ciò.

Ci sono alcuni attacchi che si possono attuare all'RSA che non si basano sulla fattorizzazione:

il primo potrebbe essere un attacco di tipo **chosen ciphertext attack** in cui l'attaccante possiede un testo cifrato C e vuole ottenere il corrispettivo in chiaro. Supponiamo abbia accesso ad un algoritmo di decifrazione, ossia una device che genera i messaggi in chiaro dei cifrati dati in input. Se può utilizzare questa routine di decifrazione, riesce ad ottenere il testo originale.

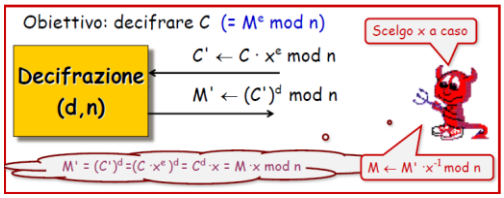


Questo attacco è possibile farlo perché RSA gode di una proprietà di omomorfismo che dice che se si hanno i due testi cifrati C_1 e C_2 che sono 2 testi cifrati di due messaggi M_1 e M_2, se si vuole sapere qual è il testo cifrato che corrisponde al prodotto dei due messaggi in chiaro, la cifratura del prodotto dei due messaggi in chiaro è uguale al prodotto dei due messaggi cifrati:

$$\left.\begin{aligned} C_1 &= M_1^e \bmod n \\ C_2 &= M_2^e \bmod n \end{aligned}\right\} \quad (M_1 \cdot M_2)^e = M_1^e \cdot M_2^e = C_1 \cdot C_2 \bmod n$$

Proprietà di omomorfismo

Questa proprietà fa sì che l'attaccante possa rompere un testo cifrato:



L'idea è che se l'attaccante ha il testo cifrato C, che vuole rompere, dal testo cifrato C si calcola un nuovo testo cifrato C', utilizza questo testo cifrato C' nella device e ottiene il messaggio corrispondente a C' e da questo messaggio si calcola il messaggio originario che rompe il testo cifrato C. Come fa tutte queste operazioni? Utilizzando la proprietà di omomorfismo, e la fa con i calcoli mostrati in foto. C'è uguale al testo cifrato C moltiplicato per un x elevato alla e, dove x è scelto a caso. Quando la device decifra e calcola il messaggio, lui riesce a calcolarsi il messaggio originario che ha la caratteristica che moltiplicato per x, dà il messaggio che riceve. Quindi per ottenere il messaggio originario basta prendere M' e dividerlo per x.

Esistono altri tipi di attacchi con certe caratteristiche, ad esempio sul tempo. Se qualcuno ha accesso ad una device fisica che riesce a decifrare, poiché una decifrazione è un processo di elevazione a potenza, se si riescono a calcolare i tempi in cui vengono calcolate le decifrature e so qual è l'algoritmo della decifratura, riesco ad ottenere delle informazioni sull'esponente d, per esempio quanti bit a 1 ha e così via.

La stessa cosa si può fare analizzando una caratteristica come la potenza. Si è infatti analizzato che se si va a vedere la potenza che viene consumata da alcune device per la decifratura, sapendo che ci sono alcune operazioni che costano di più mentre altre di meno, si può ricavare l'informazione sul valore d.

Questi tipi di attacchi non dipendono da fattorizzazione o proprietà aritmetiche.

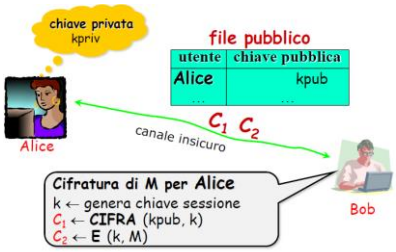
In genere per difendersi da attacchi di questo tipo, s'introducono dei ritardi casuali, cosi che chi legge queste informazioni, non ha corrispondenza con valori reali. Il ritardo può essere costante o casuale.

Facciamo un confronto tra cifrari simmetrici e asimmetrici:

I vantaggi della crittografia a chiave pubblica sono che le chiavi private non sono mai trasmesse ed è possibile la firma digitale.

I vantaggi della crittografia a chiave privata sono che è molto più veloce ed è sufficiente in diverse situazioni (ad esempio, applicazioni a singolo utente).

Nella crittografia a chiave pubblica possono esserci comunque dei problemi, e allora quello che si fa è utilizzare dei **Cifrari ibridi:**



L'RSA in realtà non cifra mai il vero messaggio. RSA in realtà cifra solo la chiave di sessione che verrà utilizzata per cifrare il messaggio vero e proprio, dal punto di vista pratico quindi non si cifrerà mai il messaggio con RSA. L'approccio utilizzato è il seguente: se Bob deve cifrare un messaggio M, genera una chiave di sessione k a caso ad esempio con AES, cifra con RSA la chiave di sessione utilizzando la chiave pubblica di Alice. Poi cifrerà con quella chiave di sessione il messaggio che invierà. Quindi ad Alice verranno mandati 2 testi cifrati: una cifratura della chiave di sessione, scelta a caso per necessità, e poi una cifratura del messaggio fatto con un algoritmo a chiave simmetrica che sarà la parte C_2. Diversi attacchi che si basavano sulla proprietà di omomorfismo descritta in precedenza, con questa modalità, non hanno più senso.

Se si ha un RSA di 2048 bit, posso cifrare un valore con RSA di al massimo 2048 bit. Se devo cifrare un file di 1 giga con RSA come faccio, tenendo conto che posso cifrare al massimo un valore di 2048 bit? Cifro con l'RSA una chiave di sessione di 2048 bit, e con questa chiave cifrata posso cifrare un file di 1 giga e di qualsiasi dimensione essa sia, utilizzando le tecniche a chiave simmetrica, quindi l'algoritmo AES con una modalità operativa opportuna, che garantisce maggiore efficienza. RSA lo uso quindi per cifrare solo la chiave di sessione.

Per decifrare, Alice, per prima cosa deve recuperare la chiave di sessione, e quindi deve decifrare il primo testo cifrato. Una volta ottenuta la chiave di sessione k, la utilizza per il secondo testo cifrato per ottenere il messaggio M in chiaro.

Questo approccio ibrido, fa sì che la crittografia a chiave pubblica venga usata solo per stabilire e trasmettere una chiave, quindi la complessità computazionale dell'invio di un file di grande dimensioni, dipende dall'algoritmo simmetrico e non più dall'algoritmo a chiave pubblica.

Vediamo un uso dell'RSA, che viene chiamato **Textbook RSA**, ossia un utilizzo così come descritto in precedenza. Usando questo approccio ci sono dei problemi di sicurezza e quindi bisogna fare qualcosa in più che semplicemente cifrare in maniera diretta una chiavi di sessione.

Assumiamo un semplice attacco su textbook RSA:

Supponiamo di voler cifrare una chiave di 64 bit e la cifro con RSA, quindi $C= K^e \bmod n$. Il problema dell'attaccante sarà quello di decifrare questo testo cifrato. Se vuole rompere questo testo cifrato e ottenere la chiave, potrebbe fare in diversi modi, ad esempio generare tutte le possibili chiavi di 64 bit, cifrarle e confrontarla al testo cifrato di suo interesse. Appena la trova, ottiene la chiave di sessione. Ma un attacco del genere avviene in tempo nell'ordine di 2^{64} bit. Adesso analizziamo un attacco che ci mette meno tempo di 2^{64}:

Se viene scelto K a caso, allora esiste una certa probabilità che dice che posso scrivere K come prodotto di due numeri K_1 e K_2 che possono essere a loro volta composti e in cui entrambi i numeri sono minori di 2^{34}. Questo evento accade con una certa probabilità che grossomodo la si può stimare intorno al 20%. Se faccio questo, ho che $K= K_1*K_2$, allora: $C/K_1^e = K_2^e \bmod n$. Se questo è vero posso montare un attacco del genere:

Cioè costruisco la tabella in figura, a sinistra ci sono tutte le cifrature degli elementi da 1^e a 2^{34e}, quindi sto cifrando con la chiave pubblica dell'RSA e in mod n. Nella seconda colonna metto il testo cifrato, diviso il valore della prima colonna. Successivamente alla creazione della tabella, per tutti i possibili valori da 0 a 2^{34}, vado a calcolare un ipotetico valore K_2^e e vado a vedere se questo valore si trova nella seconda colonna della tabella. Se è presente, allora ho trovato la chiave. Ho quindi rotto il messaggio cifrato. Se vado ad analizzare la complessità di questo attacco, per costruire la tabella ci metto 2^{34}, la complessità del for è 2^{34}*34 perché corrisponde al logaritmo in base 2 di 2^{34} che corrisponde al tempo di ricerca. Il tempo totale è di conseguenza 2^{40} << 2^{64} precedente.

Decifra ($C = K^e$ mod n)

Costruiamo tabella:

1^e	$C/1^e$
2^e	$C/2^e$
3^e	$C/3^e$
...	...
2^{34e}	$C/ 2^{34e}$

For $K_2 = 0,\dots, 2^{34}$

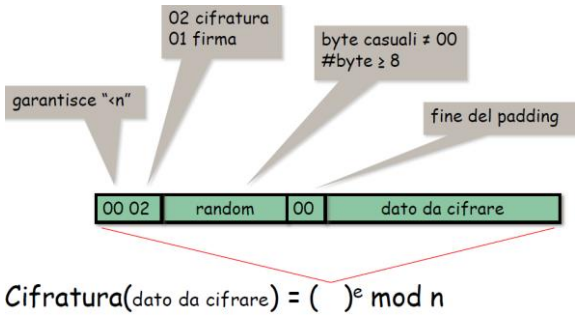
if K_2^e è nella tabella, "chiave trovata"

Di conseguenza non userei mai il textbook RSA come descritto ora perché dall'analisi dell'attacco, introdurrei una criticità al sistema.

Quello che si fa realmente per cifrare la chiave di sessione, è che prima di cifrare la chiave, faccio un preprocessing, cioè lo modifico in un modo opportuno per evitare l'attacco descritto precedentemente. Questo è il vero modo in cui lo uso

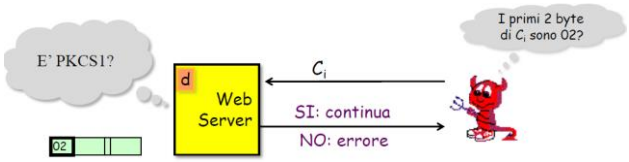
Per realizzare il preprocessing utilizzo 1 di 2 standard che esistono per l'RSA. Questi standard si chiamano **PKCS** (**P**ublic-**K**ey **C**ryptography **S**tandards), ce ne sono diversi e il #1 parla di RSA. Ci sono due schemi per la cifratura: il primo è **RSAES-PKCS1-v1_5** che ha avuto dei problemi di sicurezza e per questo è stata creata una seconda variante chiamata **RSAES-OAEP** (**O**ptimal **A**symmetric **E**ncryption **P**adding) che ha la caratteristica che sul modello di calcolo opportuno, si dimostra che la cifratura che realizzo con questo standard è sicura.

Analizziamo il primo, cioè RSAES-PKCS1-v1_5:



Il messaggio viene costruito in un modo particolare in cui il valore che devo cifrare si trova alla fine di questo array e rappresenta i bit meno significativi, poi nella parte precedente si mettono dei valori casuali diversi dagli 8 bit uguali a 0, seguito da un doppio 0. Faccio questo perché questo è un padding che inserisco, e il byte 00 che inserisco serve come virgola per separare i valori casuali che metto come padding, rispetto ai valori che devo realmente cifrare. Nel formato i primi 2 byte sono diversi: il secondo è 02 perché nello standard 01 significa che è una firma digitale, mentre 02 è una cifratura. Il primo byte invece sono tutti 0 perché la successiva parte verde, che ha la stessa lunghezza di n, dopo la devo cifrare: cioè devo prendere questa stringa e elevarla alla ^{e} mod n. Quando vado a fare questa operazione a me interessa che il valore che vado a cifrare sia più piccolo di n. Quindi per assicurarmi che la mia stringa rappresentata in figura in verde sia più piccola di n, metto i primi 8 bit a 0. Con questa struttura evito l'attacco precedente.

Questo primo schema di cifratura ha avuto degli attacchi in SSL:



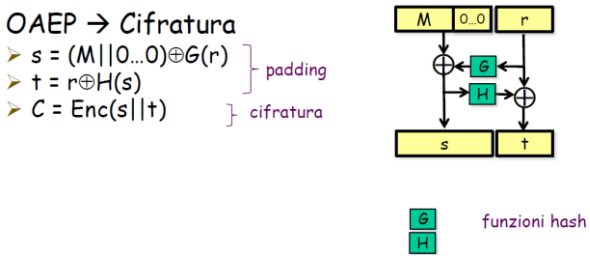
Il server deve autenticarsi al Client. Il server possiede un certificato in cui c'è una chiave pubblica e il client quando deve fare l'autenticazione del Server, cifra degli elementi (una qualsiasi chiave di sessione) e inizia la conversazione. Il Client può quindi inviare dei testi cifrati al Server e il Server può accettarli o non accettarli. Se il server li accetta, significa il testo cifrato inviato è conforme al RSAES-PKCS1-v1_5. Se non è conforme il Web Server non lo accetta, e chiude la conversazione. L'utilizzo di questo standard, consente al Client di eseguire delle query, mandare dei testi cifrati senza considerare se i testi siano conformi al RSAES-PKCS1-v1_5. Se il Web Server accetta significa che, quando viene decifrato, i primi byte sono tutti 0 e poi 02, altrimenti questo non accade. Essenzialmente questo è un attacco di tipo Chosen Ciphertext Attack, poiché manda dei testi cifrati e ottiene delle informazioni su questo testo cifrato. L'attacco si formalizza nel modo seguente:

L'attaccante deve rompere un testo cifrato che è nella forma $C=(PKCS1(M))^e$ mod n. Per rompere questo testo cifrato, modifica il testo cifrato, quindi sceglie un valore a caso $r_i \in Z_n$, lo eleva alla ^{e} cosicché il valore che ottiene è come se fosse: $C_i = (r_i * PKCS1(M))^e$ mod n.

A questo punto invia C_i al server web ed annota la risposta: se è accettato allora i primi byte di r_i * PKCS1(M) mod n sono = 00 02, altrimenti sono diversi da 00 02.

Se scelgo dei valori in maniera opportuna e faccio delle query, riesco a sapere quanto vale il messaggio. Data questa vulnerabilità, questo standard non viene usato.

Vediamo quindi il funzionamento dell'RSAES-OAEP:

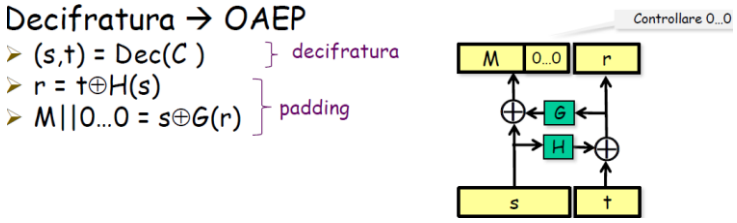


L'idea generale è che, preso il messaggio M, faccio un padding con tanti 0. Poi scelgo un valore casuale r. Il valore r, da una parte lo processo con una funzione G (supponiamo sia una funzione hash) e ne faccio lo XOR bit a bit con il messaggio, con questo valore ottengo il mio ***s*** finale, e contemporaneamente questo output lo processo con una funzione H (supponiamo sia una funzione hash) e ne faccio uno XOR bit a bit con il testo r, in questo modo ottengo la parte finale ***t***.

L'idea è quindi che da un messaggio M, ho scelto un valore casuale r e ho calcolato ***s*** e ***t*** in cui non ci sono bit espliciti cosi come nel caso precedente. Successivamente ***s*** e ***t*** li elevo alla ^{e} mod n, così da ottenere il messaggio cifrato.

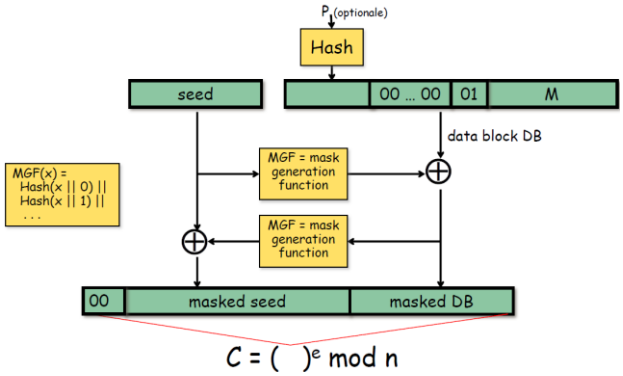
Questo processo evita tutti gli attacchi precedenti.

È stato dimostrato che in un'eventuale ipotesi in cui riesco a costruire le funzioni hash G e H con un modello particolare, allora il sistema risulta essere sicuro rispetto a questi Chosen Ciphertext Attack.



Nella decifratura, per calcolare r mi basta fare lo XOR tra t e la funzione hash applicata ad s, mentre per calcolare M faccio lo XOR tra s e la funzione hash applicata allo XOR tra t e la funzione hash su G.

Questa è l'idea. Quando si deve fare uno standard si ha un problema, cioè che G e H essenzialmente sono delle funzioni hash però se RSA è molto grande, queste dovrebbero essere delle funzioni molto grandi che danno dei valori molto grandi, che in realtà non ci sono, perché le funzioni hash che si usano sono di 224, 384, 512 bit massimi. Quindi, quando costruisco lo standard, devo pensare a qualcosa di diverso per queste funzioni G e H. Nasce da ciò il vero standard seguente:



L'utilizzo è come descritto in precedenza, soltanto che c'è qualcosina in più: le due funzioni hash sono state sostituite da funzioni chiamate MGF, c'è uno 00 iniziale per garantire che il valore sia minore a n, c'è il seed che è un valore casuale, e all'inizio è presente un sistema in più: il valore hash specifico di una stringa vuota ed è quindi un valore fissato. Poi ci sono degli 0 consecutivi e 01 cosi che si sappia quando inizia il messaggio.

La funzione MGF su input x avviene nel seguente modo: si prende la funzione Hash, si fa l'Hash del messaggio concatenato a 0, il tutto viene concatenato a Hash(x || 1) dove || sta per concatenato, e così via. Questo è lo standard della cifratura RSA, e la struttura evita attacchi causati dall'omomorfismo dell'RSA.