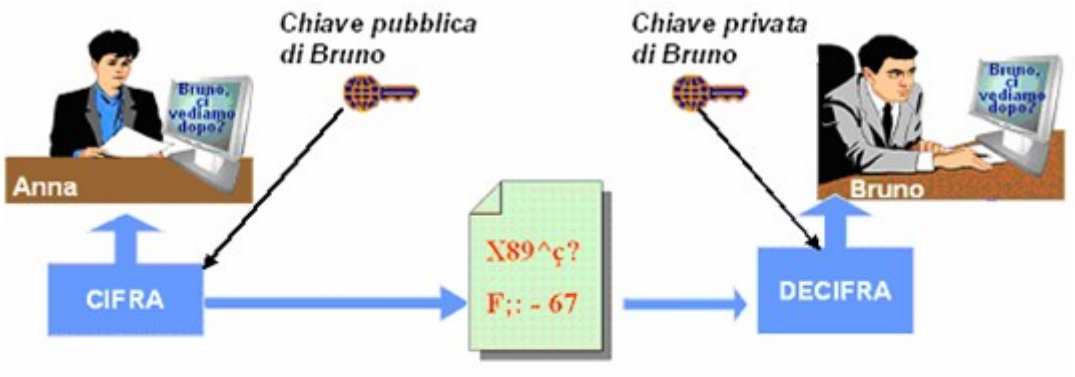
RSA

In crittografia la sigla RSA indica un algoritmo di crittografia asimmetrica, inventato nel 1977 da Ronald Rivest, Adi Shamir e Leonard Adleman utilizzabile per cifrare o firmare informazioni.

Il sistema di crittografia asimmetrico si basa sull'esistenza di due chiavi distinte, che vengono usate per cifrare e decifrare. Se la prima chiave viene usata per la cifratura, la seconda deve necessariamente essere utilizzata per la decodifica e viceversa. La questione fondamentale è che nonostante le due chiavi siano fra loro dipendenti, non sia possibile risalire dall'una all'altra, in modo che se anche si è a conoscenza di una delle due chiavi, non si possa risalire all'altra, garantendo in questo modo l'integrità della crittografia. Facendo un esempio pratico, se Anna vuole spedire un messaggio a Bruno e non vuole che altri all'infuori di Bruno possano leggerlo, Anna cercherà sull'elenco la chiave pubblica di Bruno e con quella potrà cifrare il messaggio. Essendo Bruno l'unico a possedere la chiave inversa, sarà anche l'unico a poter decifrare il messaggio, che rimarrà così segreto per tutti gli altri, compresa Anna, che non disponendo della chiave inversa non sarà in grado di decifrare il messaggio da lei stessa creato.



- **Anna** deve inviare dei dati riservati a Bruno
- **Anna** cifra i dati utilizzando la chiave pubblica di Bruno

- **Bruno** decodifica il messaggio cifrato da Anna, utilizzando la sua chiave privata (chiave segreta, unica chiave in grado di decifrare il messaggio)

Per poter realizzare con il cifrario asimmetrico un sistema crittografico pubblico è importante che un utente si crei autonomamente entrambe le chiavi, denominate "diretta" ed "inversa", e ne renda pubblica una soltanto. Così facendo si viene a creare una sorta di "elenco telefonico" a disposizione di tutti gli utenti, che raggruppa tutte le chiavi dirette, mentre quelle inverse saranno tenute segrete dagli utenti che le hanno create e da questi utilizzate solo quando ricevono un messaggio cifrato con la rispettiva chiave pubblica dell'elenco da parte di un certo mittente, ottenendo in questo modo i presupposti necessari alla sicurezza del sistema.

Per ottenere una discreta sicurezza è necessario utilizzare chiavi binarie di almeno 2048 bit. Quelle a 512 bit sono ricavabili in poche ore. Le chiavi a 1024 bit, ancora oggi largamente utilizzate, non sono più consigliabili. La fattorizzazione di interi grandi, infatti, è progredita rapidamente mediante l'utilizzo di hardware dedicati, al punto che potrebbe essere possibile fattorizzare un intero di 1024 bit in un solo anno di tempo, al costo di un milione di dollari (un costo sostenibile per qualunque grande organizzazione, agenzia o intelligence.

La matematica del sistema RSA

RSA fa uso dell'**aritmetica modulare**, che si basa sul concetto di congruenza modulo n. Dati tre numeri interi a, b, n, con n≠0, si dice che a e b sono congruenti modulo n se la loro differenza (a-b) è un multiplo di n. In questo caso si scrive:

$$a \equiv b \pmod{n}$$

Il concetto di congruenza modulo n equivale anche a dire:

$$a = b + k \cdot n$$

Per esempio si può scrivere:

$$38 \equiv 14 \pmod{12}$$

poiché 38-14=24, che è un multiplo di 12, oppure $38 = 14 + 2 \cdot 12$.

Per l'aritmetica modulare valgono le seguenti proprietà:

- **Riflessiva**: ogni numero è congruo a sé stesso modulo *n*, per ogni *n* diverso da 0 fissato;
  $a \equiv a \pmod{n} \quad \forall a \in \mathbb{N}, \forall n \in \mathbb{N}_0$
- **Simmetrica**: se a è congruo a b modulo n allora b è congruo ad a modulo n;
  $a \equiv b \pmod{n} \Rightarrow b \equiv a \pmod{n} \quad \forall a, b \in \mathbb{N}, \forall n \in \mathbb{N}_0$
- **Transitiva**: se a è congruo a b modulo n e b è congruo a c modulo n allora anche a è congruo a c modulo n.
  $a \equiv b \pmod{n} \wedge b \equiv c \pmod{n} \Rightarrow a \equiv c \pmod{n} \quad \forall a, b, c \in \mathbb{N}, \forall n \in \mathbb{N}_0$

Le proprietà riflessiva, simmetrica e transitiva descritte sopra indicano che la relazione di congruenza modulo n è una relazione di equivalenza e definisce quindi un insieme quoziente. La divisione euclidea di un intero a per n, per cui $a = k \cdot n + r$, ovvero $a - r = k \cdot n$, consente di suddividere l'insieme $\mathbb{N}$ dei naturali in n classi (sottoinsiemi) secondo la seguente relazione di equivalenza: si dice che un intero a è equivalente a $r \pmod{n}$ se e solo se la differenza a-r è un multiplo relativo di n. Si definisce così $\mathbb{Z}_n$ l'insieme quoziente di $\mathbb{Z}$ rispetto a tale relazione di equivalenza e formato dalle n classi [0],[1], ... , [n-2] e [n-1], chiamate classi di resto modulo n.

RSA fa anche uso della **funzione di Eulero** $\varphi(n)$, che indica il numero di naturali, non superiori a n, che sono primi con n (cioè che non hanno divisori comuni con n) contando tra i primi anche l'1

(cioè $\varphi(1)=1$).

Ad esempio:

- $\varphi(10)=4$ infatti 1,3,7,9 sono i 4 numeri minori di n primi con n;
- $\varphi(15)=8$ infatti 1,2,4,7,8,11,13,14 sono gli 8 numeri minori di n primi con n.

Tra le proprietà più importanti della funzione $\varphi(n)$ si ricordano le seguenti:

1. $\varphi(p)=p-1$, se p è un numero primo. Ad esempio: $\varphi(5)=4$
2. $\varphi(n\cdot m)=\varphi(n)\cdot\varphi(m)$ se n e m sono numeri primi tra di loro. Ad esempio: $\varphi(21)=\varphi(7)\cdot\varphi(3)=6\cdot 2=12$
3. $\varphi(p^k)=p^k-p^{k-1}=p^{k-1}\cdot(p-1)$ se p è un numero primo. Ad esempio: $\varphi(5^6)=5^6-5^5=5^5(5-1)=12500$
4. $a^{\varphi(n)}\equiv 1(mod\, n)$ con a ed n primi tra loro. Anche $a^{\varphi(n)\cdot k}\equiv 1(mod\, n)$. Ad esempio: $5^6\equiv 1(mod\, 7)$
5. $a^{\varphi(n)+1}\equiv a(mod\, n)$ con a ed n primi tra loro (moltiplicando per a entrambi i membri dell'equivalenza precedente). Anche $a^{\varphi(n)\cdot k+1}\equiv a(mod\, n)$. Ad esempio: $5^7\equiv 5(mod\, 7)$

Inoltre, viene fatto uso delle **equazioni diofantee** (cioè espresse nella forma $a\cdot x+b\cdot y=c$ di cui si vogliono trovare x e y soluzioni intere). Si supponga di avere l'equazione $e\cdot d\equiv 1(mod\, 120)$, ovvero $e\cdot d=1+k\cdot 120$. Conoscendo il numero e=17, si voglia determinare il numero intero d tale che $17\cdot d=1+k\cdot 120$. Si passi a leggere la precedente uguaglianza nelle classi modulo 17: $0\cdot d\equiv 1+1\cdot k(mod\, 17)$ cioè $-1\cdot k\equiv 1(mod\, 17)$, considerato che $120\, mod\, 17=1$. Poiché in aritmetica modulare sottrarre 1 è come sommare il numero opposto 16, cioè quel numero che sommato a 1 dà come risultato 0 $1+16\equiv 0(mod\, 17)$, allora:
$16\cdot k\equiv 1(mod\, 17)$ da cui si ricava facilmente $k=16$, provando con tutti i valori di k a partire da 1.
Sostituendo k nell'equazione:
$17\cdot d=1+16\cdot 120$ e quindi $d=113$

Infine si usa un algoritmo efficace, noto con il nome di **algoritmo di Legendre**, per il calcolo di $a^r\equiv b(mod\, n)$. I passi sono i seguenti:

- Si converte l'esponente r in base 2: $r=\sum_{i=0}^{d} r_i\cdot 2^i$ dove $r_i\in\{0,1\}$;
- Si riscrive $a^r\equiv b(mod\, n)$ come produttoria dei termini per cui $r_i\neq 0$: $a^r=\prod_{i=0}^{d} a^{r_i\cdot 2^i}$;
- Si ricompone $a^r\equiv b(mod\, n)$ calcolando di volta in volta a^{2^i} riferendosi al termine precedente, cioè $a^{2^{2\cdot i}}\equiv a^{2^i}\cdot a^{2^i}(mod\, n)$ partendo da $a^{2^0}=a$.

Ad esempio: per calcolare 7^{18} in $\mathbb{Z}_{15}$:

- Si converte 18 in base 2: $18=2^4+2^1=16+2$
- Si scrive $7^{18}=7^{16}\cdot 7^2$
- Si calcolano le potenze di 7 in modulo 15:
  $7\equiv 7(mod\, 15)$
  $7^2=7\cdot 7\equiv 7\cdot 7(mod\, 15)\equiv 4(mod\, 15)$
  $7^4=7^2\cdot 7^2\equiv 4\cdot 4(mod\, 15)\equiv 1(mod\, 15)$
  $7^8=7^4\cdot 7^4\equiv 1\cdot 1(mod\, 15)\equiv 1(mod\, 15)$
  $7^{16}=7^8\cdot 7^8\equiv 1\cdot 1(mod\, 15)\equiv 1(mod\, 15)$
- Infine si calcola $7^{18}=7^{16}\cdot 7^2\equiv 1\cdot 4(mod\, 15)\equiv 4(mod\, 15)$

Funzionamento

RSA è basato sull'elevata complessità computazionale della fattorizzazione in numeri primi. Il suo funzionamento base è il seguente:

- si scelgono a caso due numeri primi, p e q abbastanza grandi da garantire la sicurezza dell'algoritmo (ad esempio, il più grande numero RSA, RSA-2048, utilizza due numeri primi lunghi più di 300 cifre decimali)
- si calcola il loro prodotto $n = p \cdot q$, chiamato modulo (dato che tutta l'aritmetica seguente è modulo n), e il prodotto $\varphi(n) = (p-1) \cdot (q-1)$
- si considera che la fattorizzazione di n è segreta e solo chi sceglie i due numeri primi, p e q, la conosce
- si sceglie poi un numero e (chiamato esponente pubblico), coprimo (primi tra di loro) con $\varphi(n)$ e più piccolo di $\varphi(n)$.
- si calcola il numero d (chiamato esponente privato) tale che il suo prodotto con e sia congruo ad 1 modulo $\varphi(n)$ ovvero che $e \cdot d \equiv 1 (mod\, \varphi(n))$.

La chiave pubblica è (n, e), mentre la chiave privata è (n, d) . I due numeri primi p e q possono essere distrutti, anche se spesso vengono mantenuti insieme alla chiave privata.

La forza dell'algoritmo sta nel fatto che per calcolare d da e o viceversa, non basta la conoscenza di n, ma serve il numero $\varphi(n) = (p-1)(q-1)$ e che il suo calcolo richiede tempi molto elevati; infatti fattorizzare in numeri primi (cioè scomporre un numero nei suoi divisori primi) è un'operazione molto lenta.

Un messaggio m viene cifrato attraverso l'operazione $c \equiv m^e (mod\, n)$ trasformandolo nel messaggio cifrato c. Una volta trasmesso, c viene decifrato con $c^d \equiv m (mod\, n)$. Il procedimento funziona solo se la chiave e utilizzata per cifrare e la chiave d utilizzata per decifrare sono legate tra loro dalla relazione $d \cdot e \equiv 1 (mod\, \varphi(n))$, e quindi quando un messaggio viene cifrato con una delle due chiavi può essere decifrato solo utilizzando l'altra.

Esempio pratico

Ecco un esempio di cifratura e decifratura RSA. I numeri scelti sono volutamente primi piccoli, ma nella realtà sono usati numeri dell'ordine di 10^{100} .

Generazione delle chiavi

Bruno vuole ricevere un messaggio da Anna senza che gli altri possano leggerlo. Bruno deve per prima cosa generare la chiave pubblica e quella privata:

- Sceglie due numeri primi p e q, ad esempio p=11 e q=13;
- Calcola $n = p \cdot q = 143$ e $\varphi(n) = \varphi(11) \cdot \varphi(13) = (11-1) \cdot (13-1) = 120$
- Sceglie un numero e coprimo con $\varphi(n)$ (cioè $MCD(e, (p-1) \cdot (q-1)) = 1$) e minore di $\varphi(n)$, ad esempio e=17.
- Infine calcola l'inverso di e in $\mathbb{Z}_{\varphi(n)}$, cioè quel numero tale che $e \cdot d \equiv 1 (mod\, n)$.

Grazie al metodo di risoluzione delle equazioni diofantee visto in precedenza, Bruno trova il valore d=113 e quindi ottiene la chiave privata (n=143,d=113) e la chiave pubblica (n=143,e=17).

Riassumendo:

p	q	$n=p\cdot q$	e tale che $MCD(e,\varphi(n))=1$	d tale che $e\cdot d\equiv 1(mod\, n)$
3	11	33	7	3

Cifratura e decifratura

Anna codifica il messaggio m = 75 da inviare a Bruno, che a sua volta gli ha comunicato la chiave pubblica ma non quella privata. Anna quindi calcola $c\equiv m^e(mod\, n)$ cioè $c=75^{17}(mod\, 143)$.
Per farlo, converte 17 in binario:

$17=2^4+2^0=16+1$

Calcola quindi le potenze di 75 in modulo :

$75\equiv 75(mod\, 143)$
$75^2=75\cdot 75\equiv 75\cdot 75(mod\, 143)\equiv 48(mod\, 143)$
$75^4=75^2\cdot 75^2\equiv 48\cdot 48(mod\, 143)\equiv 16(mod\, 143)$
$75^8=75^4\cdot 75^4\equiv 16\cdot 16(mod\, 143)\equiv 113(mod\, 143)$
$75^{16}=75^8\cdot 75^8\equiv 113\cdot 113(mod\, 143)\equiv 42(mod\, 143)$

Anna può dire infine che:

$75^{17}=75^{16}\cdot 75^1\equiv 42\cdot 75(mod\, 143)\equiv 4(mod\, 143)$

Una volta giunto a destinazione il messaggio, Bruno deve usare la chiave privata che solo lui conosce per tornare al messaggio originale. Bruno quindi calcola: $m\equiv c^d(mod\, n)$ cioè $4^{113}\equiv 75(mod\, 143)$.

Hacker

Che cosa deve fare l'hacker Enrico per decodificare il messaggio?

1. Trovare i fattori primi di n: p e q;
2. Trovare d come soluzione dell'equazione $e\cdot d\equiv 1(mod\,(p-1)\cdot(q-1))$
3. Calcolare $m\equiv c^d(mod\, n)$ per decodificare il messaggio intercettato.

Implementazione in Java

Iniziamo a generare il certificato seguendo i seguenti passi:

- Si prendono in input *p* e *q* primi e maggiori di 1:

```
while(!primo(p) || p<=1) { ...
```

- Calcoliamo *m* e *n*:

```
m=(p-1)*(q-1);
n=p*q;
```

- Si prende in input *e* coprimo con $\varphi(n)$, compreso tra 1 e $\varphi(n)$:
  `while(MCD(e,phi)!=1 || e<=1 || e>=phi){ ...`
- L'equazione $e \cdot d \equiv 1 (mod\, \varphi(n))$ può essere riscritta come $e \cdot d = 1 + k \cdot \varphi(n)$. Per trovare una soluzione intera, risolviamola come equazione diofantea nelle classi di modulo *e:* $0 \cdot d \equiv 1 + k \cdot \varphi(n) (mod\, e)$ *cioè* $k \cdot -\varphi(n) \equiv 1 (mod\, e)$. A tal fine, calcoliamo il resto della divisione $-\varphi(n)/e$ considerando che il resto di una divisione in informatica non è il resto di una divisione in matematica. Una divisione per interi in matematica, infatti, si calcola come: $q = floor(a/b)$ e $r = a - q \cdot b$, dove con floor si intende il cosiddetto arrotondamento per difetto o troncamento. In Java sarà quindi:
  ```
  phi_div_e=(int)Math.floor(-(double)phi/(double)e);
  phi_mod_e=-phi-div_phi*e;
  ```
- Partendo da k=1, si trova il valore di k che soddisfa l'equazione $k \cdot -\varphi(n) \equiv 1 (mod\, e)$. In java:
  ```
  k=1;
  while ((phi_mod_e*k)%e!=1)
        k++;
  ```
- Si calcola infine d:
  `d=(k*phi+1)/e;`

Per codificare e decodificare:

- Prendo in input la chiave pubblica (n,e) o privata (n,d) e l'elemento x da decodificare;
- Codifico o decodifico il valore x calcolando $base^{esponente} \equiv risultato (mod\, n)$:
  ```
  base=x%n;
  risultato=1;
  while(e>0){
        if(e%2==1)
              risultato=(risultato*base)%n;
        e=e/2;
        base=(base*base)%n;
  }
  ```

Per hackerare un certificato:

- Si prendono in input *n* ed *e;*
- Si calcolano *p* e *q* cercando i fattori primi di n:
  ```
  p=2;
  while(n%p!=0)
        p++;
  q=n/p;
  ```
- Si verifica che *p* e *q* siano primi e n>1, altrimenti il valore di *n* inserito non è valido:
  `while(!primo(p)||!primo(q)||n<=1){`
- Si procede come se si dovesse generare il certificato.

Per codificare o decodificare un'intera parola, si può convertire una stringa in un vettore di interi e viceversa facendo uso dei seguenti sottoprogrammi:

```
	private static String trasformaVettoreInStringa(int[] msg) {
		int i,length=msg.length;
		StringBuilder string = new StringBuilder();
		string.setLength(length);
		for(i=0;i<length;i++)
			string.setCharAt(i, (char)msg[i]);
		return string.toString();
	}

	private static int[] trasformaStringaInVettore(String msg) {
		int i,length=msg.length();
		int[] vettore=new int[length];
		for(i=0;i<length;i++)
			vettore[i]=msg.codePointAt(i);
		return vettore;
	}
```

Prova a:

- Prendere in input una parola e il certificato pubblico;
- Trasformare la parola presa in input in vettore e stamparlo;
- Codificare ogni elemento del vettore, memorizzare il risultato in un nuovo vettore e stamparlo;
- Trasformare il vettore in stringa e stamparla.

Al contrario:

- Prendere in input un vettore codificato e il certificato privato;
- Trasformare il vettore in stringa e stamparla;
- Decodificare ogni elemento del vettore, memorizzare il risultato in un nuovo vettore e stamparlo;
- Trasformare il nuovo vettore in stringa e stamparla.