Scambio di chiavi di Diffie-Hellman

Prof. Francesco Bergadano

Dipartimento di Informatica
Università di Torino

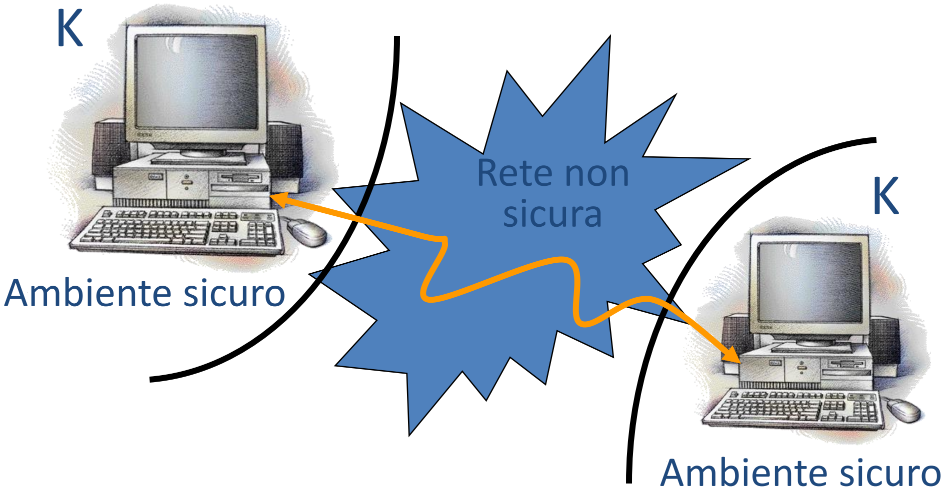
Copyright Notice

Prof. Francesco Bergadano

Dipartimento di Informatica
Università di Torino

Scambio di chiavi di Diffie-Hellman

- Primo esempio di tecnica crittografica 'a chiave pubblica' (1977)
- Non un vero e proprio cifrario
- Usato per scambio di chiavi simmetriche



Problema della distribuzione della chiave condivisa (cifrario simmetrico)

Scambio di chiavi
di Diffie-Hellman

Prof. Francesco Bergadano

Dipartimento di Informatica
Università di Torino

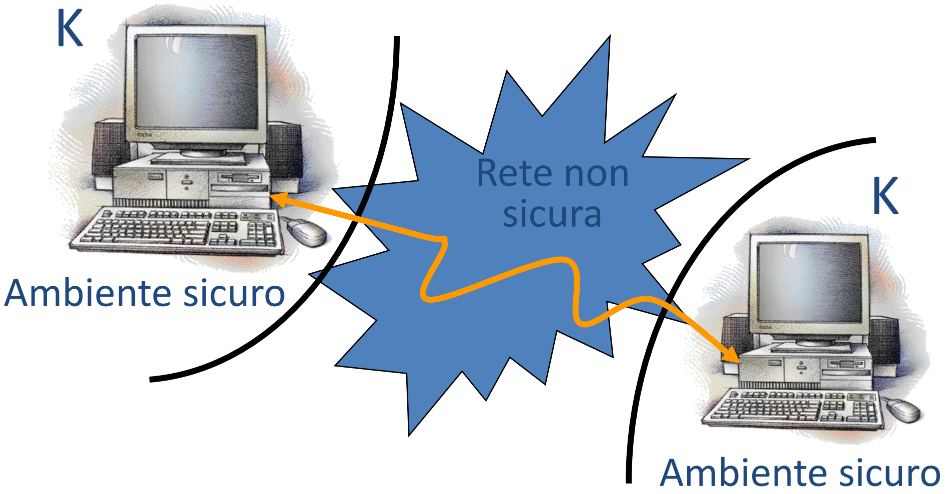
Copyright Notice

Prof. Francesco Bergadano

Dipartimento di Informatica
Università di Torino

Scambio di chiavi di Diffie-Hellman

- Primo esempio di tecnica crittografica 'a chiave pubblica' (1977)
- Non un vero e proprio cifrario
- Usato per scambio di chiavi simmetriche



Problema della distribuzione della chiave condivisa (cifrario simmetrico)

Scambio di chiavi di Diffie-Hellman

Prof. Francesco Bergadano

Dipartimento di Informatica
Università di Torino

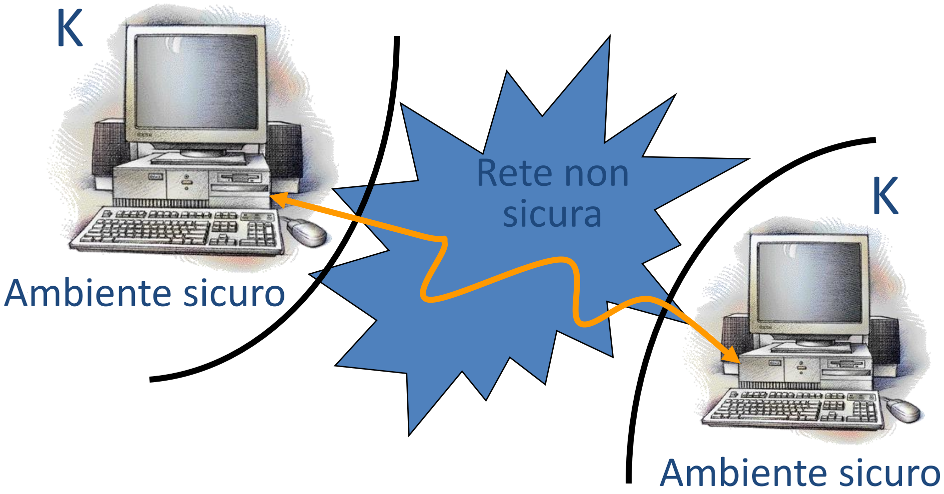
Copyright Notice

Prof. Francesco Bergadano

Dipartimento di Informatica
Università di Torino

Scambio di chiavi di Diffie-Hellman

- Primo esempio di tecnica crittografica 'a chiave pubblica' (1977)
- Non un vero e proprio cifrario
- Usato per scambio di chiavi simmetriche



Problema della distribuzione della chiave condivisa (cifrario simmetrico)

Scambio di chiavi di Diffie-Hellman

Prof. Francesco Bergadano

Dipartimento di Informatica
Università di Torino

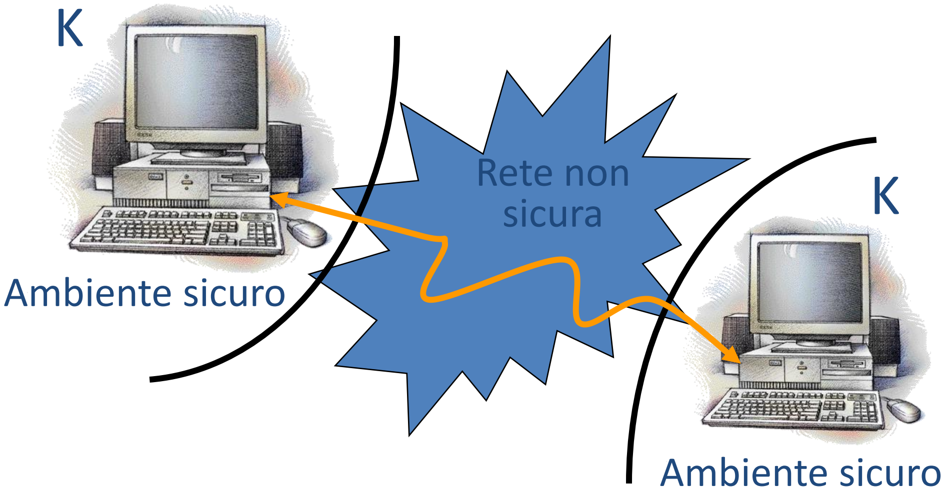
Copyright Notice

Prof. Francesco Bergadano

Dipartimento di Informatica
Università di Torino

Scambio di chiavi di Diffie-Hellman

- Primo esempio di tecnica crittografica ‘a chiave pubblica’ (1977)
- Non un vero e proprio cifrario
- Usato per scambio di chiavi simmetriche



Problema della distribuzione della chiave condivisa (cifrario simmetrico)

Scambio di chiavi di Diffie-Hellman

Come altre tecniche crittografiche a chiave pubblica, è basato su operazioni in aritmetica modulare

Definizione: dato M, a mod M (oppure a%M) denota il resto della divisione di a per M (quindi per a<M, a mod M = a).

Conviene "portare all'interno di un prodotto" l'operazione modulo

Proprietà: (ab) mod M =
= (a mod M)(b mod M) mod M

Dimostrazione:

Siano a mod M = x e b mod M = y, con x e y < M,

allora (∃k,j) Mk+x=a & Mj+y=b.

Pertanto (ab) mod M = [(Mk+x)(Mj+y)] mod M =
= (MkMj+Mky+Mjx+xy) mod M = xy mod M =
= [(a mod M)(b mod M)] mod M

Stessa cosa per le potenze

Proprietà: (a^b) mod M =
= (a mod M)^b mod M

Dimostrazione:

Esercizio

Notazione (I)

X | Y
→
"X divide Y"
ovvero (∃k) Y = kX
ovvero Y mod X = 0

Per ogni X, X|0 (infatti 0 = 0*X)
Per ogni X, X|X (infatti X = 1*X)
Per ogni Y, 1|Y (infatti Y=Y*1)

Scambio di chiavi di Diffie-Hellman

Come altre tecniche crittografiche a chiave pubblica, è basato su operazioni in aritmetica modulare

Definizione: dato M, a mod M (oppure a%M) denota il resto della divisione di a per M (quindi per a<M, a mod M = a).

Conviene "portare all'interno di un prodotto" l'operazione modulo

Proprietà: (ab) mod M =
= (a mod M)(b mod M) mod M

Dimostrazione:

Siano a mod M = x e b mod M = y, con x e y < M,

allora (∃k,j) Mk+x=a & Mj+y=b.

Pertanto (ab) mod M = [(Mk+x)(Mj+y)] mod M =
= (MkMj+Mky+Mjx+xy) mod M = xy mod M =
= [(a mod M)(b mod M)] mod M

Stessa cosa per le potenze

Proprietà: (a^b) mod M =
= $(a \bmod M)^b$ mod M

Dimostrazione:

Esercizio

Notazione (I)

X | Y
→
"X divide Y"
ovvero (∃k) Y = kX
ovvero Y mod X = 0

Per ogni X, X|0 (infatti 0 = 0*X)
Per ogni X, X|X (infatti X = 1*X)
Per ogni Y, 1|Y (infatti Y=Y*1)

Scambio di chiavi di Diffie-Hellman

Come altre tecniche crittografiche a chiave pubblica, è basato su operazioni in aritmetica modulare

Definizione: dato M, a mod M (oppure a%M) denota il resto della divisione di a per M (quindi per a<M, a mod M = a).

Conviene "portare all'interno di un prodotto" l'operazione modulo

Proprietà: (ab) mod M =
= (a mod M)(b mod M) mod M

Dimostrazione:

Siano a mod M = x e b mod M = y, con x e y < M,

allora (∃k,j) Mk+x=a & Mj+y=b.

Pertanto (ab) mod M = [(Mk+x)(Mj+y)] mod M =
= (MkMj+Mky+Mjx+xy) mod M = xy mod M =
= [(a mod M)(b mod M)] mod M

Stessa cosa per le potenze

Proprietà: (a^b) mod M =
= $(a \bmod M)^b$ mod M

Dimostrazione:

Esercizio

Notazione (I)

X | Y
→
"X divide Y"
ovvero (∃k) Y = kX
ovvero Y mod X = 0

Per ogni X, X|0 (infatti 0 = 0*X)
Per ogni X, X|X (infatti X = 1*X)
Per ogni Y, 1|Y (infatti Y=Y*1)

Scambio di chiavi di Diffie-Hellman

Come altre tecniche crittografiche a chiave pubblica, è basato su operazioni in aritmetica modulare

Definizione: dato M, a mod M (oppure a%M) denota il resto della divisione di a per M (quindi per a<M, a mod M = a).

Conviene "portare all'interno di un prodotto" l'operazione modulo

Proprietà: (ab) mod M =
= (a mod M)(b mod M) mod M

Dimostrazione:

Siano a mod M = x e b mod M = y, con x e y < M,

allora (∃k,j) Mk+x=a & Mj+y=b.

Pertanto (ab) mod M = [(Mk+x)(Mj+y)] mod M =
= (MkMj+Mky+Mjx+xy) mod M = xy mod M =
= [(a mod M)(b mod M)] mod M

Stessa cosa per le potenze

Proprietà: (a^b) mod M =
= $(a \bmod M)^b$ mod M

Dimostrazione:

Esercizio

Notazione (I)

X | Y
→
"X divide Y"
ovvero (∃k) Y = kX
ovvero Y mod X = 0

Per ogni X, X|0 (infatti 0 = 0*X)
Per ogni X, X|X (infatti X = 1*X)
Per ogni Y, 1|Y (infatti Y=Y*1)

Proprietà (I)

se X|Y e Y|Z, allora X|Z, infatti

sia Y=kX e Z=jY, allora Z=jkX, ovvero X|Z

se X|Y e X|Z, allora X|(iY+jZ) per ogni intero i,j

infatti, siano Y=rX e Z=sX, allora

iY+jZ = irX+jsX = (ir+js)X, ovvero X|(iY+jZ)

Proprietà (II)

se n|XY e MCD(X,n)=1, allora n|Y

Dimostrazione -> esercizio

Notazione (II)

X mod M = Y mod M

→

"X è congruente a Y modulo M"

$X \equiv Y \pmod{M}$

Es. $17 \equiv 12 \pmod{5}$

$X \equiv 1 \pmod{M}$

Significa che X diviso M dà resto 1

(quindi X = KM + 1 per qualche K)

Es. $17 \equiv 1 \pmod{8}$, e infatti 17 = K*8+1 con K=2

Scambio di chiavi di Diffie-Hellman

Informazioni note: q (numero primo)

α (radice primitiva di q)

Chiave privata di A: Sa < q

Chiave pubblica di A: $Pa = \alpha^{Sa} \bmod q$

Chiave privata di B: Sb < q

Chiave pubblica di B: $Pb = \alpha^{Sb} \bmod q$

Chiave condivisa: $K = Pb^{Sa} \bmod q =$

$(\alpha^{Sb})^{Sa} \bmod q = (\alpha^{Sa})^{Sb} \bmod q = Pa^{Sb} \bmod q$

Radici primitive - esempi

q=7
$\alpha=2$

$2^0=1$
$2^1=2$
$2^2=4$
$2^3=1$

$2^4 \bmod 7 =$
$(2^3 * 2) \bmod 7 =$
$[(2^3 \bmod 7) * (2 \bmod 7)] \bmod 7$
$= (1 * 2)$

q=7
$\alpha=3$

$3^0=1$
$3^1=3$
$3^2=2$
$3^3=6$
$3^4=4$
$3^5=5$
$3^6=1$

$3^4 \bmod 7 =$
$(3^3 * 3) \bmod 7 =$
$[(3^3 \bmod 7) * (3 \bmod 7)] \bmod 7$
$= (6 * 3) \bmod 7 = 18 \bmod 7 = 4$

Proprietà (I)

se $X|Y$ e $Y|Z$, allora $X|Z$, infatti

sia $Y=kX$ e $Z=jY$, allora $Z=jkX$, ovvero $X|Z$

se $X|Y$ e $X|Z$, allora $X|(iY+jZ)$ per ogni intero i,j

infatti, siano $Y=rX$ e $Z=sX$, allora

$iY+jZ = irX+jsX = (ir+js)X$, ovvero $X|(iY+jZ)$

Proprietà (II)

se $n|XY$ e $MCD(X,n)=1$, allora $n|Y$

Dimostrazione -> esercizio

Notazione (II)

$X \bmod M = Y \bmod M$

→

"X è congruente a Y modulo M"

$X \equiv Y \pmod M$

Es. $17 \equiv 12 \pmod 5$

$X \equiv 1 \pmod M$

Significa che X diviso M dà resto 1

(quindi $X = KM + 1$ per qualche K)

Es. $17 \equiv 1 \pmod 8$, e infatti $17 = K*8+1$ con $K=2$

Scambio di chiavi di Diffie-Hellman

Informazioni note: q (numero primo)

α (radice primitiva di q)

Chiave privata di A: $Sa < q$

Chiave pubblica di A: $Pa = \alpha^{Sa} \bmod q$

Chiave privata di B: $Sb < q$

Chiave pubblica di B: $Pb = \alpha^{Sb} \bmod q$

Chiave condivisa: $K = Pb^{Sa} \bmod q =$

$(\alpha^{Sb})^{Sa} \bmod q = (\alpha^{Sa})^{Sb} \bmod q = Pa^{Sb} \bmod q$

Radici primitive - esempi

$q=7$
$\alpha=2$

$2^0=1$
$2^1=2$
$2^2=4$
$2^3=1$

$2^4 \bmod 7 =$
$(2^3 *2) \bmod 7 =$
$[(2^3 \bmod 7) * (2 \bmod 7)] \bmod 7$
$= (1 * 2)$

$q=7$
$\alpha=3$

$3^0=1$
$3^1=3$
$3^2=2$
$3^3=6$
$3^4=4$
$3^5=5$
$3^6=1$

$3^4 \bmod 7 =$
$(3^3 *3) \bmod 7 =$
$[(3^3 \bmod 7) * (3 \bmod 7)] \bmod 7$
$= (6 * 3) \bmod 7 = 18 \bmod 7 = 4$

Proprietà (I)

se X|Y e Y|Z, allora X|Z, infatti

sia Y=kX e Z=jY, allora Z=jkX, ovvero X|Z

se X|Y e X|Z, allora X|(iY+jZ) per ogni intero i,j

infatti, siano Y=rX e Z=sX, allora

iY+jZ = irX+jsX = (ir+js)X, ovvero X|(iY+jZ)

Proprietà (II)

se n|XY e MCD(X,n)=1, allora n|Y

Dimostrazione -> esercizio

Notazione (II)

X mod M = Y mod M

→

“X è congruente a Y modulo M”

X ≡ Y (mod M)

Es. 17 ≡ 12 (mod 5)

X ≡ 1 (mod M)

Significa che X diviso M dà resto 1

(quindi X = KM + 1 per qualche K)

Es. 17 ≡ 1 (mod 8), e infatti 17 = K*8+1 con K=2

Scambio di chiavi di Diffie-Hellman

Informazioni note: q (numero primo)

α (radice primitiva di q)

Chiave privata di A: Sa < q

Chiave pubblica di A: Pa = α^{Sa} mod q

Chiave privata di B: Sb < q

Chiave pubblica di B: Pb = α^{Sb} mod q

Chiave condivisa: K = Pb^{Sa} mod q =

$(\alpha^{Sb})^{Sa}$ mod q = $(\alpha^{Sa})^{Sb}$ mod q = Pa^{Sb} mod q

---

Radici primitive - esempi

q=7
α =2

$2^0=1$
$2^1=2$
$2^2=4$
$2^3=1$

2^4 mod 7 =
$(2^3 *2)$ mod 7 =
$[(2^3 \text{ mod } 7) * (2 \text{ mod } 7)]$ mod 7
= (1 * 2)

q=7
α =3

$3^0=1$
$3^1=3$
$3^2=2$
$3^3=6$
$3^4=4$
$3^5=5$
$3^6=1$

3^4 mod 7 =
$(3^3 *3)$ mod 7 =
$[(3^3 \text{ mod } 7) * (3 \text{ mod } 7)]$ mod 7
= (6 * 3) mod 7 = 18 mod 7 = 4

Proprietà (I)

se X|Y e Y|Z, allora X|Z, infatti

sia Y=kX e Z=jY, allora Z=jkX, ovvero X|Z

se X|Y e X|Z, allora X|(iY+jZ) per ogni intero i,j

infatti, siano Y=rX e Z=sX, allora

iY+jZ = irX+jsX = (ir+js)X, ovvero X|(iY+jZ)

Proprietà (II)

se n|XY e MCD(X,n)=1, allora n|Y

Dimostrazione -> esercizio

Notazione (II)

X mod M = Y mod M

→

“X è congruente a Y modulo M”

$X \equiv Y \pmod{M}$

Es. $17 \equiv 12 \pmod{5}$

$X \equiv 1 \pmod{M}$

Significa che X diviso M dà resto 1

(quindi X = KM + 1 per qualche K)

Es. $17 \equiv 1 \pmod{8}$, e infatti 17 = K*8+1 con K=2

Scambio di chiavi di Diffie-Hellman

Informazioni note: q (numero primo)

α (radice primitiva di q)

Chiave privata di A: Sa < q

Chiave pubblica di A: Pa = α^{Sa} mod q

Chiave privata di B: Sb < q

Chiave pubblica di B: Pb = α^{Sb} mod q

Chiave condivisa: K = Pb^{Sa} mod q =

$(\alpha^{Sb})^{Sa}$ mod q = $(\alpha^{Sa})^{Sb}$ mod q = Pa^{Sb} mod q

Radici primitive - esempi

q=7

α =2

$2^0=1$

$2^1=2$

$2^2=4$

$2^3=1$

2^4 mod 7 =

(2^3 *2) mod 7 =

[(2^3 mod 7) * (2 mod 7)] mod 7

= (1 * 2)

q=7

α =3

$3^0=1$

$3^1=3$

$3^2=2$

$3^3=6$

$3^4=4$

$3^5=5$

$3^6=1$

3^4 mod 7 =

(3^3 *3) mod 7 =

[(3^3 mod 7) * (3 mod 7)] mod 7

= (6 * 3) mod 7 = 18 mod 7 = 4

Esponente modulare e logaritmo discreto

$Pa = \alpha^{Sa} \bmod q$

Esponente modulare:

Trovare Pa a partire da α, q, Sa.

Logaritmo discreto:

Trovare Sa a partire da α, q, Pa.

Per realizzare DH, occorre

- Algoritmo efficiente per calcolare a^bmod q
- Algoritmo efficiente per generare q primo
- Algoritmo efficiente per generare una radice primitiva di q

I primi due algoritmi sono anche necessari per cifratura e firma con RSA

Per realizzare DH, occorre

- Algoritmo efficiente per calcolare a^bmod q
- Algoritmo efficiente per generare q primo
- Algoritmo efficiente per generare una radice primitiva di q

---

Calcolo di a^b mod q (ricorsivo)

b = 0	$a^b \bmod q = 1$
b pari	$a^b \bmod q = [(a^{b/2} \bmod q)]^2 \bmod q$
b dispari	$a^b \bmod q = [a*(a^{b-1} \bmod q)] \bmod q$

Esponente modulare e logaritmo discreto

$Pa = \alpha^{Sa} \bmod q$

Esponente modulare:

Trovare Pa a partire da α, q, Sa.

Logaritmo discreto:

Trovare Sa a partire da α, q, Pa.

Per realizzare DH, occorre

- Algoritmo efficiente per calcolare $a^b \bmod q$
- Algoritmo efficiente per generare q primo
- Algoritmo efficiente per generare una radice primitiva di q

I primi due algoritmi sono anche necessari per cifratura e firma con RSA

Per realizzare DH, occorre

- Algoritmo efficiente per calcolare $a^b \bmod q$
- Algoritmo efficiente per generare q primo
- Algoritmo efficiente per generare una radice primitiva di q

---

Calcolo di $a^b \bmod q$ (ricorsivo)

b = 0	$a^b \bmod q = 1$
b pari	$a^b \bmod q = [(a^{b/2} \bmod q)]^2 \bmod q$
b dispari	$a^b \bmod q = [a*(a^{b-1} \bmod q)] \bmod q$

Esponente modulare e logaritmo discreto

Pa = α^{Sa} mod q

Esponente modulare:

Trovare Pa a partire da α, q, Sa.

Logaritmo discreto:

Trovare Sa a partire da α, q, Pa.

Per realizzare DH, occorre

- Algoritmo efficiente per calcolare a^bmod q
- Algoritmo efficiente per generare q primo
- Algoritmo efficiente per generare una radice primitiva di q

I primi due algoritmi sono anche necessari per cifratura e firma con RSA

Per realizzare DH, occorre

- Algoritmo efficiente per calcolare a^bmod q
- Algoritmo efficiente per generare q primo
- Algoritmo efficiente per generare una radice primitiva di q

Calcolo di a^b mod q (ricorsivo)

b = 0	a^b mod q = 1
b pari	a^b mod q = $[(a^{b/2}$ mod q$)]^2$ mod q
b dispari	a^b mod q = [a*(a^{b-1} mod q)] mod q

Esponente modulare e logaritmo discreto

Pa = α^{Sa} mod q

Esponente modulare:

Trovare Pa a partire da α, q, Sa.

Logaritmo discreto:

Trovare Sa a partire da α, q, Pa.

Per realizzare DH, occorre

- Algoritmo efficiente per calcolare a^bmod q
- Algoritmo efficiente per generare q primo
- Algoritmo efficiente per generare una radice primitiva di q

I primi due algoritmi sono anche necessari per cifratura e firma con RSA

Per realizzare DH, occorre

- Algoritmo efficiente per calcolare a^bmod q
- Algoritmo efficiente per generare q primo
- Algoritmo efficiente per generare una radice primitiva di q

---

Calcolo di a^b mod q (ricorsivo)

b = 0	a^b mod q = 1
b pari	a^b mod q = $[(a^{b/2} \bmod q)]^2$ mod q
b dispari	a^b mod q = $[a*(a^{b-1} \bmod q)]$ mod q

Calcolo di a^b mod q (ricorsivo)

```
int expmod (int a, int b, int q) {
// restituisce un valore minore di q
if (b = = 0) return 1; // $a^0$ mod q = 0
if (b%2 = = 0)
   return sq(expmod(a,b/2,q))%q; // b pari
else
   return (a*expmod(a,b-1,q))%q; // b dispari
}   // (ab) mod q = (a mod q) (b mod q) mod q
```

Calcolo di a^b mod q (iterativo)

```
// b = $b_k \dots b_2 b_1 b_0$
// c non serve al calcolo, solo per la
   dimostrazione di correttezza
c = 0; d = 1;
for (i=k; i>=0; i--) {
   c = c*2; d = (d*d)%q;
   if ($b_i$ = = 1) {
        c = c+1; d = (d*a)%q; }
   }
return d;
```

Calcolo di a^b mod q (iterativo)

```
// b = $b_k \dots b_2 b_1 b_0$
// numero b in binario, di k+1 bit, ovvero $\Sigma_{bj=1,0 \leq j \leq k} 2^j = b$
c = 0; d = 1;
for (i=k; i>=0; i--)
   c = c*2; d = (d*d)%q;
   // $c = \Sigma_{bj=1 \,\&\, j>i} 2^{j-i}$, $d = a^c$ mod q
   if ($b_i$ = = 1) {
        c = c+1; d = (d*a)%q; }
   // $c = \Sigma_{bj=1 \,\&\, j>i} 2^{j-i} + \Sigma_{bj=1 \,\&\, j=i} 1$, $d = a^c$ mod q
   }// $c = \Sigma_{bj=1,0 \leq j \leq k} 2^j = b$, $d = a^c$ mod q = $a^b$ mod q
return d;
```

Calcolo di a^b mod q (iterativo)

```
// b = $b_k \dots b_2 b_1 b_0$
c = 0; d = 1;
for (i=k; i>=0; i--)
   c = c*2; d = (d*d)%q;
   // $c = \Sigma_{bj=1 \,\&\, j>i} 2^{j-i}$, $d = a^c$ mod q
   ….. questo è vero la prima volta con c=0, d=1
```

Calcolo di a^b mod q (ricorsivo)

```
int expmod (int a, int b, int q) {
// restituisce un valore minore di q
if (b = = 0) return 1; // $a^0$ mod q = 0
if (b%2 = = 0)
   return sq(expmod(a,b/2,q))%q; // b pari
else
   return (a*expmod(a,b-1,q))%q; // b dispari
}   // (ab) mod q = (a mod q) (b mod q) mod q
```

Calcolo di a^b mod q (iterativo)

```
// b = $b_k…b_2b_1b_0$
// c non serve al calcolo, solo per la
   dimostrazione di correttezza
c = 0; d = 1;
for (i=k; i>=0; i--) {
   c = c*2; d = (d*d)%q;
   if ($b_i$ = = 1) {
        c = c+1; d = (d*a)%q; }
   }
return d;
```

Calcolo di a^b mod q (iterativo)

```
// b = $b_k…b_2b_1b_0$
// numero b in binario, di k+1 bit, ovvero $\Sigma_{bj=1,0\le j\le k}2^j$ = b
c = 0; d = 1;
for (i=k; i>=0; i--)
   c = c*2; d = (d*d)%q;
   // c = $\Sigma_{bj=1 \& j>i}2^{j-i}$, d = $a^c$ mod q
   if ($b_i$ = = 1) {
        c = c+1; d = (d*a)%q; }
   // c = $\Sigma_{bj=1 \& j>i}2^{j-i}+\Sigma_{bj=1 \& j=i}1$, d = $a^c$ mod q
   }// c = $\Sigma_{bj=1,0\le j\le k}2^j$ = b, d = $a^c$ mod q = $a^b$ mod q
return d;
```

Calcolo di a^b mod q (iterativo)

```
// b = $b_k…b_2b_1b_0$
c = 0; d = 1;
for (i=k; i>=0; i--)
   c = c*2; d = (d*d)%q;
   // c = $\Sigma_{bj=1 \& j>i}2^{j-i}$, d = $a^c$ mod q
   ….. questo è vero la prima volta con c=0, d=1
```

Calcolo di a^b mod q (ricorsivo)

```
int expmod (int a, int b, int q) {
// restituisce un valore minore di q
if (b = = 0) return 1; // $a^0$ mod q = 0
if (b%2 = = 0)
   return sq(expmod(a,b/2,q))%q; // b pari
else
   return (a*expmod(a,b-1,q))%q; // b dispari
}   // (ab) mod q = (a mod q) (b mod q) mod q
```

Calcolo di a^b mod q (iterativo)

```
// b = $b_k…b_2b_1b_0$
// c non serve al calcolo, solo per la
   dimostrazione di correttezza
c = 0; d = 1;
for (i=k; i>=0; i--) {
   c = c*2; d = (d*d)%q;
   if ($b_i$ = = 1) {
        c = c+1; d = (d*a)%q; }
   }
return d;
```

Calcolo di a^b mod q (iterativo)

```
// b = $b_k…b_2b_1b_0$
// numero b in binario, di k+1 bit, ovvero $\Sigma_{bj=1,0≤j≤k}2^j$ = b
c = 0; d = 1;
for (i=k; i>=0; i--)
   c = c*2; d = (d*d)%q;
   // c = $\Sigma_{bj=1 \& j>i}2^{j-i}$, d = $a^c$ mod q
   if ($b_i$ = = 1) {
        c = c+1; d = (d*a)%q; }
   // c = $\Sigma_{bj=1 \& j>i}2^{j-i}+\Sigma_{bj=1 \& j=i}1$, d = $a^c$ mod q
   }// c = $\Sigma_{bj=1,0≤j≤k}2^j$ = b, d = $a^c$ mod q = $a^b$ mod q
return d;
```

Calcolo di a^b mod q (iterativo)

```
// b = $b_k…b_2b_1b_0$
c = 0; d = 1;
for (i=k; i>=0; i--)
   c = c*2; d = (d*d)%q;
   // c = $\Sigma_{bj=1 \& j>i}2^{j-i}$, d = $a^c$ mod q
   ….. questo è vero la prima volta con c=0, d=1
```

Calcolo di a^b mod q (ricorsivo)

```
int expmod (int a, int b, int q) {
// restituisce un valore minore di q
if (b = = 0) return 1; // $a^0$ mod q = 0
if (b%2 = = 0)
   return sq(expmod(a,b/2,q))%q; // b pari
else
   return (a*expmod(a,b-1,q))%q; // b dispari
}   // (ab) mod q = (a mod q) (b mod q) mod q
```

Calcolo di a^b mod q (iterativo)

```
// b = $b_k...b_2b_1b_0$
// c non serve al calcolo, solo per la
   dimostrazione di correttezza
c = 0; d = 1;
for (i=k; i>=0; i--) {
   c = c*2; d = (d*d)%q;
   if ($b_i$ = = 1) {
        c = c+1; d = (d*a)%q; }
   }
return d;
```

Calcolo di a^b mod q (iterativo)

```
// b = $b_k...b_2b_1b_0$
// numero b in binario, di k+1 bit, ovvero $\Sigma_{bj=1,0\leq j\leq k}2^j$ = b
c = 0; d = 1;
for (i=k; i>=0; i--)
   c = c*2; d = (d*d)%q;
   // c = $\Sigma_{bj=1 \& j>i}2^{j-i}$, d = $a^c$ mod q
   if ($b_i$ = = 1) {
        c = c+1; d = (d*a)%q; }
   // c = $\Sigma_{bj=1 \& j>i}2^{j-i}+\Sigma_{bj=1 \& j=i}1$, d = $a^c$ mod q
   }// c = $\Sigma_{bj=1,0\leq j\leq k}2^j$ = b, d = $a^c$ mod q = $a^b$ mod q
return d;
```

Calcolo di a^b mod q (iterativo)

```
// b = $b_k...b_2b_1b_0$
c = 0; d = 1;
for (i=k; i>=0; i--)
   c = c*2; d = (d*d)%q;
   // c = $\Sigma_{bj=1 \& j>i}2^{j-i}$, d = $a^c$ mod q
   ..... questo è vero la prima volta con c=0, d=1
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k \ldots b_2 b_1 b_0$
… chiamo c', d' i valori prima dell'istruzione if
    // $c' = \Sigma_{bj=1 \& j>i} 2^{j-i}$, $d' = a^{c'} \bmod q$
    if ($b_i$ = = 1) {
        c = c'+1; d = (d'*a)%q; }
    // se $b_i = 1$
    // $c = c'+1 = \Sigma_{bj=1 \& j>i} 2^{j-i}+1$
              $= \Sigma_{bj=1 \& j>i} 2^{j-i} + \Sigma_{bj=1 \& j=i} 1$
    // $d = (d'*a) \bmod q = (a^{c'} \bmod q)*a \bmod q =$
    //    $= a^{c'+1} \bmod q = a^c \bmod q$
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k \ldots b_2 b_1 b_0$
… chiamo c', d' i valori prima dell'istruzione if
    // $c' = \Sigma_{bj=1 \& j>i} 2^{j-i}$, $d' = a^{c'} \bmod q$
    if ($b_i$ = = 1) {
        c = c'+1; d = (d'*a)%q; }
    // se $b_i = 0$, l'istruzione non viene eseguita, e
    // $c = c' = \Sigma_{bj=1 \& j>i} 2^{j-i}+0$
              $= \Sigma_{bj=1 \& j>i} 2^{j-i} + \Sigma_{bj=1 \& j=i} 1$

    // $d = d' = a^{c'} \bmod q = a^c \bmod q$
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k \ldots b_2 b_1 b_0$
// numero b in binario, di k+1 bit, ovvero $\Sigma_{bj=1, 0 \leq j \leq k} 2^j = b$
c = 0; d = 1;
for (i=k; i>=0; i--)
    c = c*2; d = (d*d)%q;
    // $c = \Sigma_{bj=1 \& j>i} 2^{j-i}$, $d = a^c \bmod q$
    if ($b_i$ = = 1) {
        c = c+1; d = (d*a)%q; }
    // $c = \Sigma_{bj=1 \& j>i} 2^{j-i} + \Sigma_{bj=1 \& j=i} 1$, $d = a^c \bmod q$
    } // $c = \Sigma_{bj=1, 0 \leq j \leq k} 2^j = b$, $d = a^c \bmod q = a^b \bmod q$
return d;
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k \ldots b_2 b_1 b_0$
// $c' = \Sigma_{bj=1 \& j>i'} 2^{j-i'} + \Sigma_{bj=1 \& j=i'} 1$, $d' = a^{c'} \bmod q$
    i=i'-1;
    c = c'*2; d = (d'*d')%q;
// $c = c'*2 = (\Sigma_{bj=1 \& j>i'} 2^{j-i'} + \Sigma_{bj=1 \& j=i'} 1)*2 =$

      $= \Sigma_{bj=1 \& j>i'} 2^{j-(i'-1)} + \Sigma_{bj=1 \& j=i'} 2^{j-(i'-1)} =$

      $= \Sigma_{bj=1 \& j>i'-1} 2^{j-(i'-1)} = \Sigma_{bj=1 \& j>i} 2^{j-i}$

// $d = (d'*d') \bmod q = (a^{c'} \bmod q)*(a^{c'} \bmod q) \bmod q =$
      $= a^{2*c'} \bmod q = a^c \bmod q$
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k \ldots b_2 b_1 b_0$
… chiamo c’, d’ i valori prima dell’istruzione if
    // $c' = \Sigma_{bj=1\ \&\ j>i} 2^{j-i}$, $d' = a^{c'}$ mod q
    if ($b_i$ = = 1) {
        c = c’+1; d = (d’*a)%q; }
    // se $b_i$ = 1
    // $c = c'+1 = \Sigma_{bj=1\ \&\ j>i} 2^{j-i}+1$
    //          $= \Sigma_{bj=1\ \&\ j>i} 2^{j-i} + \Sigma_{bj=1\ \&\ j=i} 1$
    // d = (d’*a) mod q = ($a^{c'}$ mod q)*a mod q =
    //    = $a^{c'+1}$ mod q = $a^c$ mod q
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k \ldots b_2 b_1 b_0$
… chiamo c’, d’ i valori prima dell’istruzione if
    // $c' = \Sigma_{bj=1\ \&\ j>i} 2^{j-i}$, $d' = a^{c'}$ mod q
    if ($b_i$ = = 1) {
        c = c’+1; d = (d’*a)%q; }
    // se $b_i$ = 0, l’istruzione non viene eseguita, e
    // $c = c' = \Sigma_{bj=1\ \&\ j>i} 2^{j-i}+0$
    //          $= \Sigma_{bj=1\ \&\ j>i} 2^{j-i} + \Sigma_{bj=1\ \&\ j=i} 1$

    // d = d’ = $a^{c'}$ mod q = $a^c$ mod q
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k \ldots b_2 b_1 b_0$
// numero b in binario, di k+1 bit, ovvero $\Sigma_{bj=1, 0 \le j \le k} 2^j = b$
c = 0; d = 1;
for (i=k; i>=0; i--)
    c = c*2; d = (d*d)%q;
    // $c = \Sigma_{bj=1\ \&\ j>i} 2^{j-i}$, d = $a^c$ mod q
    if ($b_i$ = = 1) {
        c = c+1; d = (d*a)%q; }
    // $c = \Sigma_{bj=1\ \&\ j>i} 2^{j-i} + \Sigma_{bj=1\ \&\ j=i} 1$, d = $a^c$ mod q
    } // $c = \Sigma_{bj=1, 0 \le j \le k} 2^j$ = b, d = $a^c$ mod q = $a^b$ mod q
return d;
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k \ldots b_2 b_1 b_0$
// $c' = \Sigma_{bj=1\ \&\ j>i'} 2^{j-i'} + \Sigma_{bj=1\ \&\ j=i'} 1$, $d' = a^{c'}$ mod q
    i=i’-1;
    c = c’*2; d = (d’*d’)%q;
// $c = c'*2 = (\Sigma_{bj=1\ \&\ j>i'} 2^{j-i'} + \Sigma_{bj=1\ \&\ j=i'} 1)*2 =$

      $= \Sigma_{bj=1\ \&\ j>i'} 2^{j-(i'-1)} + \Sigma_{bj=1\ \&\ j=i'} 2^{j-(i'-1)} =$

      $= \Sigma_{bj=1\ \&\ j>i'-1} 2^{j-(i'-1)} = \Sigma_{bj=1\ \&\ j>i} 2^{j-i}$

// d = (d’*d’) mod q = ($a^{c'}$ mod q)*($a^{c'}$ mod q) mod q =
      = $a^{2*c'}$ mod q = $a^c$ mod q
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k…b_2b_1b_0$
… chiamo c', d' i valori prima dell'istruzione if
   // $c' = \Sigma_{bj=1 \& j>i}2^{j-i}$, $d' = a^{c'}$ mod q
   if ($b_i$ = = 1) {
         c = c'+1; d = (d'*a)%q; }
   // se $b_i = 1$
   // $c = c'+1 = \Sigma_{bj=1 \& j>i}2^{j-i}+1$
                 $=\Sigma_{bj=1 \& j>i}2^{j-i}+\Sigma_{bj=1 \& j=i}1$
   // d = (d'*a) mod q = ($a^{c'}$ mod q)*a mod q =
   //    = $a^{c'+1}$ mod q = $a^c$ mod q
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k…b_2b_1b_0$
… chiamo c', d' i valori prima dell'istruzione if
   // $c' = \Sigma_{bj=1 \& j>i}2^{j-i}$, $d' = a^{c'}$ mod q
   if ($b_i$ = = 1) {
         c = c'+1; d = (d'*a)%q; }
   // se $b_i = 0$, l'istruzione non viene eseguita, e
   // $c = c' = \Sigma_{bj=1 \& j>i}2^{j-i}+0$
                $= \Sigma_{bj=1 \& j>i}2^{j-i}+\Sigma_{bj=1 \& j=i}1$

   // d = d' = $a^{c'}$ mod q = $a^c$ mod q
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k…b_2b_1b_0$
// numero b in binario, di k+1 bit, ovvero $\Sigma_{bj=1,0\leq j\leq k}2^j = b$
c = 0; d = 1;
for (i=k; i>=0; i--)
   c = c*2; d = (d*d)%q;
   // $c = \Sigma_{bj=1 \& j>i}2^{j-i}$, $d = a^c$ mod q
   if ($b_i$ = = 1) {
        c = c+1; d = (d*a)%q; }
   // $c = \Sigma_{bj=1 \& j>i}2^{j-i}+\Sigma_{bj=1 \& j=i}1$, $d = a^c$ mod q
   }// $c = \Sigma_{bj=1,0\leq j\leq k}2^j = b$, $d = a^c$ mod q = $a^b$ mod q
return d;
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k…b_2b_1b_0$
// $c' = \Sigma_{bj=1 \& j>i'}2^{j-i'}+\Sigma_{bj=1 \& j=i'}1$, $d' = a^{c'}$ mod q
   i=i'-1;
   c = c'*2; d = (d'*d')%q;
// $c = c'*2 = (\Sigma_{bj=1 \& j>i'}2^{j-i'}+\Sigma_{bj=1 \& j=i'}1)*2 =$

      $= \Sigma_{bj=1 \& j>i'}2^{j-(i'-1)}+\Sigma_{bj=1 \& j=i'}2^{j-(i'-1)} =$

      $= \Sigma_{bj=1 \& j>i'-1}2^{j-(i'-1)} = \Sigma_{bj=1 \& j>i}2^{j-i}$

// d = (d'*d') mod q = ($a^{c'}$ mod q)*($a^{c'}$ mod q) mod q =
      = $a^{2*c'}$mod q = $a^c$ mod q
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k \ldots b_2 b_1 b_0$
… chiamo c', d' i valori prima dell'istruzione if
   // $c' = \Sigma_{bj=1 \,\&\, j>i} 2^{j-i}$, $d' = a^{c'} \bmod q$
   if ($b_i$ = = 1) {
         c = c'+1; d = (d'*a)%q; }
   // se $b_i = 1$
   // $c = c'+1 = \Sigma_{bj=1 \,\&\, j>i} 2^{j-i}+1$
                 $= \Sigma_{bj=1 \,\&\, j>i} 2^{j-i} + \Sigma_{bj=1 \,\&\, j=i} 1$
   // $d = (d'*a) \bmod q = (a^{c'} \bmod q)*a \bmod q =$
   //    $= a^{c'+1} \bmod q = a^c \bmod q$
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k \ldots b_2 b_1 b_0$
… chiamo c', d' i valori prima dell'istruzione if
   // $c' = \Sigma_{bj=1 \,\&\, j>i} 2^{j-i}$, $d' = a^{c'} \bmod q$
   if ($b_i$ = = 1) {
         c = c'+1; d = (d'*a)%q; }
   // se $b_i = 0$, l'istruzione non viene eseguita, e
   // $c = c' = \Sigma_{bj=1 \,\&\, j>i} 2^{j-i}+0$
                 $= \Sigma_{bj=1 \,\&\, j>i} 2^{j-i} + \Sigma_{bj=1 \,\&\, j=i} 1$

   // $d = d' = a^{c'} \bmod q = a^c \bmod q$
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k \ldots b_2 b_1 b_0$
// numero b in binario, di k+1 bit, ovvero $\Sigma_{bj=1, 0 \le j \le k} 2^j = b$
c = 0; d = 1;
for (i=k; i>=0; i--)
   c = c*2; d = (d*d)%q;
   // $c = \Sigma_{bj=1 \,\&\, j>i} 2^{j-i}$, $d = a^c \bmod q$
   if ($b_i$ = = 1) {
        c = c+1; d = (d*a)%q; }
   // $c = \Sigma_{bj=1 \,\&\, j>i} 2^{j-i} + \Sigma_{bj=1 \,\&\, j=i} 1$, $d = a^c \bmod q$
   } // $c = \Sigma_{bj=1, 0 \le j \le k} 2^j = b$, $d = a^c \bmod q = a^b \bmod q$
return d;
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k \ldots b_2 b_1 b_0$
// $c' = \Sigma_{bj=1 \,\&\, j>i'} 2^{j-i'} + \Sigma_{bj=1 \,\&\, j=i'} 1$, $d' = a^{c'} \bmod q$
   i=i'-1;
   c = c'*2; d = (d'*d')%q;
// $c = c'*2 = (\Sigma_{bj=1 \,\&\, j>i'} 2^{j-i'} + \Sigma_{bj=1 \,\&\, j=i'} 1)*2 =$

      $= \Sigma_{bj=1 \,\&\, j>i'} 2^{j-(i'-1)} + \Sigma_{bj=1 \,\&\, j=i'} 2^{j-(i'-1)} =$

      $= \Sigma_{bj=1 \,\&\, j>i'-1} 2^{j-(i'-1)} = \Sigma_{bj=1 \,\&\, j>i} 2^{j-i}$

// $d = (d'*d') \bmod q = (a^{c'} \bmod q)*(a^{c'} \bmod q) \bmod q =$
      $= a^{2*c'} \bmod q = a^c \bmod q$
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k \ldots b_2 b_1 b_0$
// numero b in binario, di k+1 bit, ovvero $\Sigma_{b_j=1, 0 \le j \le k} 2^j = b$
c = 0; d = 1;
for (i=k; i>=0; i--)
    c = c*2; d = (d*d)%q;
    // $c = \Sigma_{b_j=1 \,\&\, j>i} 2^{j-i}$, $d = a^c \bmod q$
    if ($b_i$ = = 1) {
          c = c+1; d = (d*a)%q; }
    // $c = \Sigma_{b_j=1 \,\&\, j>i} 2^{j-i} + \Sigma_{b_j=1 \,\&\, j=i} 1$, $d = a^c \bmod q$ ↴
    } // $c = \Sigma_{b_j=1, 0 \le j \le k} 2^j = b$, $d = a^c \bmod q = a^b \bmod q$
return d;
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k \ldots b_2 b_1 b_0$
// numero b in binario, di k+1 bit, ovvero $\Sigma_{b_j=1, 0 \le j \le k} 2^j = b$
// $c = \Sigma_{b_j=1 \,\&\, j>i} 2^{j-i} + \Sigma_{b_j=1 \,\&\, j=i} 1$, $d = a^c \bmod q$
} // ovvero il ciclo termina e i=0
// $c = \Sigma_{b_j=1 \,\&\, j>0} 2^j + \Sigma_{b_j=1 \,\&\, j=0} 1 = \Sigma_{b_j=1 \,\&\, j>0} 2^j + \Sigma_{b_j=1 \,\&\, j=0} 2^j =$
      $= \Sigma_{b_j=1 \,\&\, 0 \le j} 2^j$
// $c = \Sigma_{b_j=1, 0 \le j \le k} 2^j = b$, $d = a^c \bmod q = a^b \bmod q$
return d;
```

Calcolo di a^b mod q (iterativo) – esempio

// $b = b_k \ldots b_2 b_1 b_0$ = 10110 (decimale 22)
// a = 3, d = 3^22 mod q, q = 31 (primo)
// simulare i 5 step con i=4,3,2,1,0

// risultato da calcolatrice
// 3^22 div 31 = 1012292245 = M1
// M1*31 = 31381059595 = M2
// 3^22 – M2 = 3^22 mod 31 = 14

Per realizzare DH, occorre

- Algoritmo efficiente per calcolare a^bmod q
- Algoritmo efficiente per generare q primo
- Algoritmo efficiente per generare una radice primitiva di q

Calcolo di a^b mod q (iterativo)

```
// $b = b_k \ldots b_2 b_1 b_0$
// numero b in binario, di k+1 bit, ovvero $\Sigma_{b_j=1, 0 \le j \le k} 2^j = b$
c = 0; d = 1;
for (i=k; i>=0; i--)
    c = c*2; d = (d*d)%q;
    // $c = \Sigma_{b_j=1 \,\&\, j>i} 2^{j-i}$, $d = a^c \bmod q$
    if ($b_i$ = = 1) {
          c = c+1; d = (d*a)%q; }
    // $c = \Sigma_{b_j=1 \,\&\, j>i} 2^{j-i} + \Sigma_{b_j=1 \,\&\, j=i} 1$, $d = a^c \bmod q$ ↴
    } // $c = \Sigma_{b_j=1, 0 \le j \le k} 2^j = b$, $d = a^c \bmod q = a^b \bmod q$
return d;
```

Calcolo di a^b mod q (iterativo)

```
// $b = b_k \ldots b_2 b_1 b_0$
// numero b in binario, di k+1 bit, ovvero $\Sigma_{b_j=1, 0 \le j \le k} 2^j = b$
// $c = \Sigma_{b_j=1 \,\&\, j>i} 2^{j-i} + \Sigma_{b_j=1 \,\&\, j=i} 1$, $d = a^c \bmod q$
 } // ovvero il ciclo termina e i=0
// $c = \Sigma_{b_j=1 \,\&\, j>0} 2^j + \Sigma_{b_j=1 \,\&\, j=0} 1 = \Sigma_{b_j=1 \,\&\, j>0} 2^j + \Sigma_{b_j=1 \,\&\, j=0} 2^j =$
      $= \Sigma_{b_j=1 \,\&\, 0 \le j} 2^j$
// $c = \Sigma_{b_j=1, 0 \le j \le k} 2^j = b$, $d = a^c \bmod q = a^b \bmod q$
return d;
```

Calcolo di a^b mod q (iterativo) – esempio

// $b = b_k \ldots b_2 b_1 b_0 = 10110$ (decimale 22)

// a = 3, d = 3^22 mod q, q = 31 (primo)

// simulare i 5 step con i=4,3,2,1,0

// risultato da calcolatrice

// 3^22 div 31 = 1012292245 = M1

// M1*31 = 31381059595 = M2

// 3^22 – M2 = 3^22 mod 31 = 14

Per realizzare DH, occorre

- Algoritmo efficiente per calcolare a^bmod q
- Algoritmo efficiente per generare q primo
- Algoritmo efficiente per generare una radice primitiva di q

Calcolo di a^b mod q (iterativo)

```
// b = $b_k … b_2 b_1 b_0$
// numero b in binario, di k+1 bit, ovvero $\Sigma_{b_j=1, 0 \le j \le k} 2^j = b$
c = 0; d = 1;
for (i=k; i>=0; i--)
    c = c*2; d = (d*d)%q;
    // $c = \Sigma_{b_j=1 \,\&\, j>i} 2^{j-i}$, d = $a^c$ mod q
    if ($b_i$ = = 1) {
        c = c+1; d = (d*a)%q; }
    // $c = \Sigma_{b_j=1 \,\&\, j>i} 2^{j-i} + \Sigma_{b_j=1 \,\&\, j=i} 1$, d = $a^c$ mod q ↴
    } // $c = \Sigma_{b_j=1, 0 \le j \le k} 2^j$ = b, d = $a^c$ mod q = $a^b$ mod q
return d;
```

Calcolo di a^b mod q (iterativo)

```
// b = $b_k … b_2 b_1 b_0$
// numero b in binario, di k+1 bit, ovvero $\Sigma_{b_j=1, 0 \le j \le k} 2^j = b$
// $c = \Sigma_{b_j=1 \,\&\, j>i} 2^{j-i} + \Sigma_{b_j=1 \,\&\, j=i} 1$, d = $a^c$ mod q
} // ovvero il ciclo termina e i=0
// $c = \Sigma_{b_j=1 \,\&\, j>0} 2^j + \Sigma_{b_j=1 \,\&\, j=0} 1 = \Sigma_{b_j=1 \,\&\, j>0} 2^j + \Sigma_{b_j=1 \,\&\, j=0} 2^j =$
      $= \Sigma_{b_j=1 \,\&\, 0 \le j} 2^j$
// $c = \Sigma_{b_j=1, 0 \le j \le k} 2^j$ = b, d = $a^c$ mod q = $a^b$ mod q
return d;
```

Calcolo di a^b mod q (iterativo) – esempio

```
// b = $b_k … b_2 b_1 b_0$ = 10110 (decimale 22)
// a  = 3, d = 3^22 mod q, q = 31 (primo)
// simulare i 5 step con i=4,3,2,1,0

// risultato da calcolatrice
// 3^22 div 31 = 1012292245 = M1
// M1*31 = 31381059595 = M2
// 3^22 – M2 = 3^22 mod 31 = 14
```

Per realizzare DH, occorre

- Algoritmo efficiente per calcolare a^bmod q
- Algoritmo efficiente per generare q primo
- Algoritmo efficiente per generare una radice primitiva di q

Calcolo di $a^b \bmod q$ (iterativo)

```
// $b = b_k \dots b_2 b_1 b_0$
// numero b in binario, di k+1 bit, ovvero $\Sigma_{b_j=1,0\le j\le k} 2^j = b$
c = 0; d = 1;
for (i=k; i>=0; i--)
    c = c*2; d = (d*d)%q;
    // $c = \Sigma_{b_j=1 \& j>i} 2^{j-i}$, $d = a^c \bmod q$
    if ($b_i$ = = 1) {
        c = c+1; d = (d*a)%q; }
    // $c = \Sigma_{b_j=1 \& j>i} 2^{j-i} + \Sigma_{b_j=1 \& j=i} 1$, $d = a^c \bmod q$ ↴
    } // $c = \Sigma_{b_j=1,0\le j\le k} 2^j = b$, $d = a^c \bmod q = a^b \bmod q$
return d;
```

Calcolo di $a^b \bmod q$ (iterativo)

```
// $b = b_k \dots b_2 b_1 b_0$
// numero b in binario, di k+1 bit, ovvero $\Sigma_{b_j=1,0\le j\le k} 2^j = b$
// $c = \Sigma_{b_j=1 \& j>i} 2^{j-i} + \Sigma_{b_j=1 \& j=i} 1$, $d = a^c \bmod q$
} // ovvero il ciclo termina e i=0
// $c = \Sigma_{b_j=1 \& j>0} 2^j + \Sigma_{b_j=1 \& j=0} 1 = \Sigma_{b_j=1 \& j>0} 2^j + \Sigma_{b_j=1 \& j=0} 2^j =$
      $= \Sigma_{b_j=1 \& 0\le j} 2^j$
// $c = \Sigma_{b_j=1,0\le j\le k} 2^j = b$, $d = a^c \bmod q = a^b \bmod q$
return d;
```

Calcolo di $a^b \bmod q$ (iterativo) – esempio

// $b = b_k \dots b_2 b_1 b_0 = 10110$ (decimale 22)
// a = 3, d = 3^22 mod q, q = 31 (primo)
// simulare i 5 step con i=4,3,2,1,0

// risultato da calcolatrice
// 3^22 div 31 = 1012292245 = M1
// M1*31 = 31381059595 = M2
// 3^22 – M2 = 3^22 mod 31 = 14

Per realizzare DH, occorre

- Algoritmo efficiente per calcolare $a^b \bmod q$
- Algoritmo efficiente per generare q primo
- Algoritmo efficiente per generare una radice primitiva di q

Generazione di (grandi) numeri primi

Metodo diretto:

```
1. genera M di k bit a caso
2. for (i=2; i*i<=M; i++)
       if (M%i = = 0) goto 1 // M non primo
3. return M // M primo
```

▶ e' facile che un numero casuale M sia primo?

E' facile che un numero casuale M sia primo?

Esistono infiniti numeri primi (Euclide)

Dimostrazione :

Supponiamo che p sia l'ultimo numero primo.

Sia q = 2*3*5*…*p il prodotto dei numeri primi fino a p.

Se q+1 è primo, p non era l'ultimo.

Se q+1 non è primo, è divisibile per r>p con r primo (in quanto non è divisibile per alcun primo fino a p). Di nuovo r>p, quindi p non è l'ultimo numero primo.

E' facile che un numero casuale M sia primo?

Teorema dei numeri primi:

Sia $\pi(x)$ il numero di primi minori dell'intero x.

Allora $\pi(x) \sim x/\ln(x)$ ovvero $\lim_{x\to\infty} \pi(x)/[x/\ln(x)] = 1$.

Es.

$\pi(10) = 4$, $10/\ln(10)=4{,}34$

$\pi(30) = 10$, $30/\ln(30)=8{,}82$

E' facile che un numero casuale M di 100 cifre sia primo?

In crittografia (ad esempio per DH ed RSA), useremo numeri primi grandi, di almeno 100 cifre decimali.

I numeri primi di cento cifre sono circa

$\pi(10^{100})- \pi(10^{99}) \approx$

$\approx [10^{100}/\ln(10^{100})] - [10^{99}/\ln(10^{99})] \approx 3.9 * 10^{97}$.

I numeri di 100 cifre sono 10^{100}.

Quindi avremo in media un numero primo ogni $10^{100}/3.9*10^{97}$, ovvero ogni 1000/3.9 = 256 tentativi

(potremo evitare i numeri pari, e i multipli di 3, 5, 7 e 11).

Generazione di (grandi) numeri primi

Metodo diretto:

```
1. genera M di k bit a caso
2. for (i=2; i*i<=M; i++)
       if (M%i = = 0) goto 1 // M non primo
3. return M // M primo
```

▶ e’ facile che un numero casuale M sia primo?

E’ facile che un numero casuale M sia primo?

Esistono infiniti numeri primi (Euclide)

Dimostrazione :

Supponiamo che p sia l’ultimo numero primo.

Sia q = 2*3*5*…*p il prodotto dei numeri primi fino a p.

Se q+1 è primo, p non era l’ultimo.

Se q+1 non è primo, è divisibile per r>p con r primo (in quanto non è divisibile per alcun primo fino a p). Di nuovo r>p, quindi p non è l’ultimo numero primo.

E’ facile che un numero casuale M sia primo?

Teorema dei numeri primi:

Sia $\pi(x)$ il numero di primi minori dell’intero x.

Allora $\pi(x) \sim x/\ln(x)$ ovvero $\lim_{x\to\infty} \pi(x)/[x/\ln(x)] = 1$.

Es.

$\pi(10) = 4$, $10/\ln(10)=4,34$

$\pi(30) = 10$, $30/\ln(30)=8,82$

E’ facile che un numero casuale M di 100 cifre sia primo?

In crittografia (ad esempio per DH ed RSA), useremo numeri primi grandi, di almeno 100 cifre decimali.

I numeri primi di cento cifre sono circa

$\pi(10^{100})- \pi(10^{99}) \approx$

$\approx [10^{100}/\ln(10^{100})] - [10^{99}/\ln(10^{99})] \approx 3.9 * 10^{97}$.

I numeri di 100 cifre sono 10^{100}.

Quindi avremo in media un numero primo ogni $10^{100}/3.9*10^{97}$, ovvero ogni 1000/3.9 = 256 tentativi

(potremo evitare i numeri pari, e i multipli di 3, 5, 7 e 11).

Generazione di (grandi) numeri primi

Metodo diretto:

```
1. genera M di k bit a caso
2. for (i=2; i*i<=M; i++)
       if (M%i = = 0) goto 1 // M non primo
3. return M // M primo
```

▶ e’ facile che un numero casuale M sia primo?

E’ facile che un numero casuale M sia primo?

Esistono infiniti numeri primi (Euclide)

Dimostrazione :

Supponiamo che p sia l’ultimo numero primo.

Sia q = 2*3*5*…*p il prodotto dei numeri primi fino a p.

Se q+1 è primo, p non era l’ultimo.

Se q+1 non è primo, è divisibile per r>p con r primo (in quanto non è divisibile per alcun primo fino a p). Di nuovo r>p, quindi p non è l’ultimo numero primo.

E’ facile che un numero casuale M sia primo?

Teorema dei numeri primi:

Sia $\pi(x)$ il numero di primi minori dell’intero x.

Allora $\pi(x) \sim x/\ln(x)$ ovvero $\lim_{x\to\infty} \pi(x)/[x/\ln(x)] = 1$.

Es.

$\pi(10) = 4$, $10/\ln(10)=4,34$

$\pi(30) = 10$, $30/\ln(30)=8,82$

E’ facile che un numero casuale M di 100 cifre sia primo?

In crittografia (ad esempio per DH ed RSA), useremo numeri primi grandi, di almeno 100 cifre decimali.

I numeri primi di cento cifre sono circa

$\pi(10^{100})$- $\pi(10^{99}) \approx$

$\approx [10^{100}/\ln(10^{100})] - [10^{99}/\ln(10^{99})] \approx 3.9 * 10^{97}$.

I numeri di 100 cifre sono 10^{100}.

Quindi avremo in media un numero primo ogni $10^{100}/3.9*10^{97}$, ovvero ogni 1000/3.9 = 256 tentativi

(potremo evitare i numeri pari, e i multipli di 3, 5, 7 e 11).

Generazione di (grandi) numeri primi

Metodo diretto:

```
1. genera M di k bit a caso
2. for (i=2; i*i<=M; i++)
       if (M%i = = 0) goto 1 // M non primo
3. return M // M primo
```

▶ e’ facile che un numero casuale M sia primo?

E’ facile che un numero casuale M sia primo?

Esistono infiniti numeri primi (Euclide)

Dimostrazione :

Supponiamo che p sia l’ultimo numero primo.

Sia q = 2*3*5*…*p il prodotto dei numeri primi fino a p.

Se q+1 è primo, p non era l’ultimo.

Se q+1 non è primo, è divisibile per r>p con r primo (in quanto non è divisibile per alcun primo fino a p). Di nuovo r>p, quindi p non è l’ultimo numero primo.

E’ facile che un numero casuale M sia primo?

Teorema dei numeri primi:

Sia $\pi(x)$ il numero di primi minori dell’intero x.

Allora $\pi(x) \sim x/\ln(x)$ ovvero $\lim_{x\to\infty} \pi(x)/[x/\ln(x)] = 1$.

Es.

$\pi(10) = 4$, $10/\ln(10)=4,34$

$\pi(30) = 10$, $30/\ln(30)=8,82$

E’ facile che un numero casuale M di 100 cifre sia primo?

In crittografia (ad esempio per DH ed RSA), useremo numeri primi grandi, di almeno 100 cifre decimali.

I numeri primi di cento cifre sono circa

$\pi(10^{100})- \pi(10^{99}) \approx$

$\approx [10^{100}/\ln(10^{100})] - [10^{99}/\ln(10^{99})] \approx 3.9 * 10^{97}$.

I numeri di 100 cifre sono 10^{100}.

Quindi avremo in media un numero primo ogni $10^{100}/3.9*10^{97}$, ovvero ogni 1000/3.9 = 256 tentativi

(potremo evitare i numeri pari, e i multipli di 3, 5, 7 e 11).

Esercizio

Abbiamo in media un numero primo ogni $10^{100}/3.9*10^{97}$, ovvero ogni 1000/3.9 = 256 tentativi:

Come si riduce questo numero evitando i numeri pari?

Idea:

M (dispari)

Saltare M+1, M+3, … perche’ sono pari

Esercizio

Abbiamo in media un numero primo ogni $10^{100}/3.9*10^{97}$, ovvero ogni 1000/3.9 = 256 tentativi:

Come si riduce questo numero evitando i multipli di 3 e 5?

Idea:

M3 = M oppure M-1 oppure M-2, multiplo di 3

M5 = M oppure M-1 oppure … M-4, multiplo di 5

Saltare M3, M3+3, M3+6 … (senza contare quelli pari)

Saltare M5, M5+5, M5+10, … (senza i multipli di 2 e di 3)

Generazione di (grandi) numeri primi

Metodo diretto:

```
1. genera M di k bit a caso
2. for (i=2; i*i<=M; i++)
       if (M%i = = 0) goto 1 // M non primo
3. return M // M primo
```

▶ Per numeri di 100 cifre potremo fare un centinaio di tentativi e trovare M primo

---

Generazione di (grandi) numeri primi

Metodo diretto:

```
1. genera M di k bit a caso
2. for (i=2; i*i<=M; i++)
       if (M%i = = 0) goto 1 // M non primo
3. return M // M primo
```

▶ ma, troppo lento! $M^{(1/2)} = 2^{(k/2)}$ divisioni

Esercizio

Abbiamo in media un numero primo ogni $10^{100}/3.9*10^{97}$, ovvero ogni 1000/3.9 = 256 tentativi:

Come si riduce questo numero evitando i numeri pari?

Idea:

M (dispari)

Saltare M+1, M+3, … perche' sono pari

Esercizio

Abbiamo in media un numero primo ogni $10^{100}/3.9*10^{97}$, ovvero ogni 1000/3.9 = 256 tentativi:

Come si riduce questo numero evitando i multipli di 3 e 5?

Idea:

M3 = M oppure M-1 oppure M-2, multiplo di 3

M5 = M oppure M-1 oppure … M-4, multiplo di 5

Saltare M3, M3+3, M3+6 … (senza contare quelli pari)

Saltare M5, M5+5, M5+10, … (senza i multipli di 2 e di 3)

Generazione di (grandi) numeri primi

Metodo diretto:

```
1. genera M di k bit a caso
2. for (i=2; i*i<=M; i++)
       if (M%i = = 0) goto 1 // M non primo
3. return M // M primo
```

▶ Per numeri di 100 cifre potremo fare un centinaio di tentativi e trovare M primo

---

Generazione di (grandi) numeri primi

Metodo diretto:

```
1. genera M di k bit a caso
2. for (i=2; i*i<=M; i++)
       if (M%i = = 0) goto 1 // M non primo
3. return M // M primo
```

▶ ma, troppo lento! $M^{(1/2)} = 2^{(k/2)}$ divisioni

Esercizio

Abbiamo in media un numero primo ogni $10^{100}/3.9*10^{97}$, ovvero ogni 1000/3.9 = 256 tentativi:

Come si riduce questo numero evitando i numeri pari?

Idea:

M (dispari)

Saltare M+1, M+3, … perche’ sono pari

Esercizio

Abbiamo in media un numero primo ogni $10^{100}/3.9*10^{97}$, ovvero ogni 1000/3.9 = 256 tentativi:

Come si riduce questo numero evitando i multipli di 3 e 5?

Idea:

M3 = M oppure M-1 oppure M-2, multiplo di 3

M5 = M oppure M-1 oppure … M-4, multiplo di 5

Saltare M3, M3+3, M3+6 … (senza contare quelli pari)

Saltare M5, M5+5, M5+10, … (senza i multipli di 2 e di 3)

Generazione di (grandi) numeri primi

Metodo diretto:

```
1. genera M di k bit a caso
2. for (i=2; i*i<=M; i++)
       if (M%i = = 0) goto 1 // M non primo
3. return M // M primo
```

▶ Per numeri di 100 cifre potremo fare un centinaio di tentativi e trovare M primo

Generazione di (grandi) numeri primi

Metodo diretto:

```
1. genera M di k bit a caso
2. for (i=2; i*i<=M; i++)
       if (M%i = = 0) goto 1 // M non primo
3. return M // M primo
```

▶ ma, troppo lento! $M^{(1/2)} = 2^{(k/2)}$ divisioni

Esercizio

Abbiamo in media un numero primo ogni $10^{100}/3.9*10^{97}$, ovvero ogni 1000/3.9 = 256 tentativi:

Come si riduce questo numero evitando i numeri pari?

Idea:

M (dispari)

Saltare M+1, M+3, … perche’ sono pari

Esercizio

Abbiamo in media un numero primo ogni $10^{100}/3.9*10^{97}$, ovvero ogni 1000/3.9 = 256 tentativi:

Come si riduce questo numero evitando i multipli di 3 e 5?

Idea:

M3 = M oppure M-1 oppure M-2, multiplo di 3

M5 = M oppure M-1 oppure … M-4, multiplo di 5

Saltare M3, M3+3, M3+6 … (senza contare quelli pari)

Saltare M5, M5+5, M5+10, … (senza i multipli di 2 e di 3)

Generazione di (grandi) numeri primi

Metodo diretto:

```
1. genera M di k bit a caso
2. for (i=2; i*i<=M; i++)
       if (M%i = = 0) goto 1 // M non primo
3. return M // M primo
```

▶ Per numeri di 100 cifre potremo fare un centinaio di tentativi e trovare M primo

Generazione di (grandi) numeri primi

Metodo diretto:

```
1. genera M di k bit a caso
2. for (i=2; i*i<=M; i++)
       if (M%i = = 0) goto 1 // M non primo
3. return M // M primo
```

▶ ma, troppo lento! $M^{(1/2)} = 2^{(k/2)}$ divisioni

Generazione di (grandi) numeri primi

Metodo diretto per generare M di k bit

circa $M^{(1/2)} = 2^{(k/2)}$ divisioni

per k = 512, $2^{(k/2)}$ divisioni = 1,2e+77divisioni

Generazione di (grandi) numeri primi

Metodo probabilistico:

```
1. genera M di k bit a caso
2. for (i=0; i<T; i++) // ripeti T volte
        genera a<M a caso
        if (test(a,M)) goto 1 // M non primo
        // M primo con probabilità > ½
3. return M // Pr(M non primo) < $2^{-T}$
             // Pr(M primo) > $1 - 2^{-T}$
```

▶ Pr(M primo) molto alta!

Esempio

Pr(M primo) > $1 - 2^{-T}$

es. per T = 100,

Pr(M primo) >
0,999999999999999999999999999999211

Test di Miller-Rabin

Pr(M non primo & $\text{Test}_{\text{Miller-Rabin}}$(M)=false) < 1/4

Si usano due teoremi:

1. (Piccolo teorema di Fermat, sarà dimostrato in seguito, come corollario del teorema di Eulero)
   se M è primo e a<M, allora a^{M-1} mod M = 1
2. (conseguenza del «principio fondamentale», vedi slide successiva)
   se M è primo e x^2 mod M = 1,
   allora x = 1 o x = M-1

Generazione di (grandi) numeri primi

Metodo diretto per generare M di k bit

circa $M^{(1/2)} = 2^{(k/2)}$ divisioni

per k = 512, $2^{(k/2)}$ divisioni = 1,2e+77divisioni

Generazione di (grandi) numeri primi

Metodo probabilistico:

```
1. genera M di k bit a caso
2. for (i=0; i<T; i++) // ripeti T volte
          genera a<M a caso
          if (test(a,M)) goto 1 // M non primo
          // M primo con probabilità > ½
3. return M // Pr(M non primo) < $2^{-T}$
               // Pr(M primo) > 1 - $2^{-T}$
```

▶ Pr(M primo) molto alta!

Esempio

Pr(M primo) > $1 - 2^{-T}$

es. per T = 100,

Pr(M primo) >
0,999999999999999999999999999999211

Test di Miller-Rabin

Pr(M non primo & $\text{Test}_{\text{Miller-Rabin}}(M)$=false) < 1/4

Si usano due teoremi:

1. (Piccolo teorema di Fermat, sarà dimostrato in seguito, come corollario del teorema di Eulero)
   se M è primo e a<M, allora a^{M-1} mod M = 1
2. (conseguenza del «principio fondamentale», vedi slide successiva)
   se M è primo e x^2 mod M = 1,
   allora x = 1 o x = M-1

Generazione di (grandi) numeri primi

Metodo diretto per generare M di k bit

circa $M^{(1/2)} = 2^{(k/2)}$ divisioni

per k = 512, $2^{(k/2)}$ divisioni = 1,2e+77divisioni

Generazione di (grandi) numeri primi

Metodo probabilistico:

```
1. genera M di k bit a caso
2. for (i=0; i<T; i++) // ripeti T volte
       genera a<M a caso
       if (test(a,M)) goto 1 // M non primo
       // M primo con probabilità > ½
3. return M // Pr(M non primo) < $2^{-T}$
            // Pr(M primo) > 1 - $2^{-T}$
```

▶ Pr(M primo) molto alta!

Esempio

Pr(M primo) > $1 - 2^{-T}$

es. per T = 100,

Pr(M primo) >
0,99999999999999999999999999999211

Test di Miller-Rabin

Pr(M non primo & $\text{Test}_{\text{Miller-Rabin}}$(M)=false) < 1/4

Si usano due teoremi:

1. (Piccolo teorema di Fermat, sarà dimostrato in seguito, come corollario del teorema di Eulero)
   se M è primo e a<M, allora a^{M-1} mod M = 1
2. (conseguenza del «principio fondamentale», vedi slide successiva)
   se M è primo e x^2 mod M = 1,
   allora x = 1 o x = M-1

Generazione di (grandi) numeri primi

Metodo diretto per generare M di k bit

circa $M^{(1/2)} = 2^{(k/2)}$ divisioni

per k = 512, $2^{(k/2)}$ divisioni = 1,2e+77divisioni

Generazione di (grandi) numeri primi

Metodo probabilistico:

```
1. genera M di k bit a caso
2. for (i=0; i<T; i++) // ripeti T volte
        genera a<M a caso
        if (test(a,M)) goto 1 // M non primo
        // M primo con probabilità > ½
3. return M // Pr(M non primo) < $2^{-T}$
               // Pr(M primo) > 1 - $2^{-T}$
```

▶ Pr(M primo) molto alta!

Esempio

Pr(M primo) > 1 - 2^{-T}

es. per T = 100,

Pr(M primo) >
0,999999999999999999999999999999211

Test di Miller-Rabin

Pr(M non primo & $\text{Test}_{\text{Miller-Rabin}}$(M)=false) < 1/4

Si usano due teoremi:

1. (Piccolo teorema di Fermat, sarà dimostrato in seguito, come corollario del teorema di Eulero)
   se M è primo e a<M, allora a^{M-1} mod M = 1
2. (conseguenza del «principio fondamentale», vedi slide successiva)
   se M è primo e x^2 mod M = 1,
   allora x = 1 o x = M-1

Principio fondamentale

Se esistono x e y, t.c. $x^2 \equiv y^2 \pmod{n}$ e $\neg(x \equiv \pm y \pmod{n})$, allora n non è primo.

Dimostrazione:

Sia d = MCD(x-y,n). Ci sono due casi:

1. Se d=n, allora $x \equiv y \pmod{n}$ – quindi $d \neq n$.
2. Sia d=1. Sappiamo che $n | x^2-y^2$, quindi $n | (x-y)(x+y)$.
   Per quanto sopra, n non può dividere x-y. Quindi deve dividere x+y.
   Ciò contraddice $\neg(x \equiv - y \pmod{n})$ - quindi $d \neq 1$.

Siccome MCD(x-y,n) è diverso da 1 e da n,

allora vi è un divisore di n diverso da 1 ed n, ovvero n non è primo.

Principio fondamentale

Se esistono x e y, t.c. $x^2 \equiv y^2 \pmod{n}$ e $\neg(x \equiv \pm y \pmod{n})$, allora n non è primo.

Dimostrazione:

Sia d = MCD(x-y,n). Ci sono due casi:

1. Se d=n, allora $x \equiv y \pmod{n}$ – quindi $d \neq n$.
2. Sia d=1. Sappiamo che $n | x^2-y^2$, quindi $n | (x-y)(x+y)$.
   Per quanto sopra, n non può dividere x-y. Quindi deve dividere x+y.
   Ciò contraddice $\neg(x \equiv - y \pmod{n})$ - quindi $d \neq 1$.

Siccome MCD(x-y,n) è diverso da 1 e da n,

allora vi è un divisore di n diverso da 1 ed n, ovvero n non è primo.

Se n|XY e MCD(n,X)=1, allora n|Y (proprietà II dimostrata in precedenza)

Principio fondamentale

Se esistono x e y, tali che
$x^2 \equiv y^2 \pmod{n}$ e $\neg(x \equiv \pm y \pmod{n})$, allora n non è primo.

Pertanto,

se M è primo e $x^2 \bmod M = 1$,

allora x = 1 o x = M-1

Generazione di (grandi) numeri primi

Metodo probabilistico:

```
1. genera M di k bit a caso
2. for (i=0; i<T; i++) // ripeti T volte
        genera a<M a caso
        if (test(a,M)) goto 1 // M non primo
3. return M // Pr(M primo) > 1 - $4^{-T}$
```

▶ test(a,M) = test di Miller-Rabin

Principio fondamentale

Se esistono x e y, t.c. $x^2 \equiv y^2 \pmod n$ e $\neg(x \equiv \pm y \pmod n)$, allora n non è primo.

Dimostrazione:

Sia d = MCD(x-y,n). Ci sono due casi:

1. Se d=n, allora $x \equiv y \pmod n$ – quindi $d \neq n$.
2. Sia d=1. Sappiamo che $n | x^2-y^2$, quindi $n | (x-y)(x+y)$.
   Per quanto sopra, n non può dividere x-y. Quindi deve dividere x+y.
   Ciò contraddice $\neg(x \equiv - y \pmod n)$ - quindi $d \neq 1$.

Siccome MCD(x-y,n) è diverso da 1 e da n,

allora vi è un divisore di n diverso da 1 ed n, ovvero n non è primo.

Principio fondamentale

Se esistono x e y, t.c. $x^2 \equiv y^2 \pmod n$ e $\neg(x \equiv \pm y \pmod n)$, allora n non è primo.

Dimostrazione:

Sia d = MCD(x-y,n). Ci sono due casi:

1. Se d=n, allora $x \equiv y \pmod n$ – quindi $d \neq n$.
2. Sia d=1. Sappiamo che $n | x^2-y^2$, quindi $n | (x-y)(x+y)$.
   Per quanto sopra, n non può dividere x-y. Quindi deve dividere x+y.
   Ciò contraddice $\neg(x \equiv - y \pmod n)$ - quindi $d \neq 1$.

Siccome MCD(x-y,n) è diverso da 1 e da n,

allora vi è un divisore di n diverso da 1 ed n, ovvero n non è primo.

Se n|XY e MCD(n,X)=1, allora n|Y (proprietà II dimostrata in precedenza)

Principio fondamentale

Se esistono x e y, tali che
$x^2 \equiv y^2 \pmod n$ e $\neg(x \equiv \pm y \pmod n)$, allora n non è primo.

Pertanto,

se M è primo e $x^2 \bmod M = 1$,

allora x = 1 o x = M-1

Generazione di (grandi) numeri primi

Metodo probabilistico:

```
1. genera M di k bit a caso
2. for (i=0; i<T; i++) // ripeti T volte
       genera a<M a caso
       if (test(a,M)) goto 1 // M non primo
3. return M // Pr(M primo) > 1 - $4^{-T}$
```

▶ test(a,M) = test di Miller-Rabin

Principio fondamentale

Se esistono x e y, t.c. $x^2 \equiv y^2$ (mod n) e $\neg(x \equiv \pm y$ (mod n)), allora n non è primo.

Dimostrazione:

Sia d = MCD(x-y,n). Ci sono due casi:

1. Se d=n, allora $x \equiv y$ (mod n) – quindi $d \neq n$.
2. Sia d=1. Sappiamo che $n|x^2-y^2$, quindi $n|(x-y)(x+y)$.
   Per quanto sopra, n non può dividere x-y. Quindi deve dividere x+y.
   Ciò contraddice $\neg(x \equiv - y$ (mod n)) - quindi $d \neq 1$.

Siccome MCD(x-y,n) è diverso da 1 e da n,

allora vi è un divisore di n diverso da 1 ed n, ovvero n non è primo.

Principio fondamentale

Se esistono x e y, t.c. $x^2 \equiv y^2$ (mod n) e $\neg(x \equiv \pm y$ (mod n)), allora n non è primo.

Dimostrazione:

Sia d = MCD(x-y,n). Ci sono due casi:

1. Se d=n, allora $x \equiv y$ (mod n) – quindi $d \neq n$.
2. Sia d=1. Sappiamo che $n|x^2-y^2$, quindi $n|(x-y)(x+y)$.
   Per quanto sopra, n non può dividere x-y. Quindi deve dividere x+y.
   Ciò contraddice $\neg(x \equiv - y$ (mod n)) - quindi $d \neq 1$.

Siccome MCD(x-y,n) è diverso da 1 e da n,

allora vi è un divisore di n diverso da 1 ed n, ovvero n non è primo.

Se n|XY e MCD(n,X)=1, allora n|Y (proprietà II dimostrata in precedenza)

Principio fondamentale

Se esistono x e y, tali che
$x^2 \equiv y^2$ (mod n) e $\neg(x \equiv \pm y$ (mod n)), allora n non è primo.

Pertanto,

se M è primo e x^2 mod M = 1,

allora x = 1 o x = M-1

Generazione di (grandi) numeri primi

Metodo probabilistico:

```
1. genera M di k bit a caso
2. for (i=0; i<T; i++) // ripeti T volte
        genera a<M a caso
        if (test(a,M)) goto 1 // M non primo
3. return M // Pr(M primo) > 1 - 4^-T
```

▶ test(a,M) = test di Miller-Rabin

Principio fondamentale

Se esistono x e y, t.c. $x^2 \equiv y^2 \pmod{n}$ e $\neg(x \equiv \pm y \pmod{n})$, allora n non è primo.

Dimostrazione:

Sia d = MCD(x-y,n). Ci sono due casi:

1. Se d=n, allora $x \equiv y \pmod{n}$ – quindi $d \neq n$.
2. Sia d=1. Sappiamo che $n | x^2-y^2$, quindi $n | (x-y)(x+y)$.
   Per quanto sopra, n non può dividere x-y. Quindi deve dividere x+y.
   Ciò contraddice $\neg(x \equiv -y \pmod{n})$ - quindi $d \neq 1$.

Siccome MCD(x-y,n) è diverso da 1 e da n,

allora vi è un divisore di n diverso da 1 ed n, ovvero n non è primo.

Principio fondamentale

Se esistono x e y, t.c. $x^2 \equiv y^2 \pmod{n}$ e $\neg(x \equiv \pm y \pmod{n})$, allora n non è primo.

Dimostrazione:

Sia d = MCD(x-y,n). Ci sono due casi:

1. Se d=n, allora $x \equiv y \pmod{n}$ – quindi $d \neq n$.
2. Sia d=1. Sappiamo che $n | x^2-y^2$, quindi $n | (x-y)(x+y)$.
   Per quanto sopra, n non può dividere x-y. Quindi deve dividere x+y.
   Ciò contraddice $\neg(x \equiv -y \pmod{n})$ - quindi $d \neq 1$.

Siccome MCD(x-y,n) è diverso da 1 e da n,

allora vi è un divisore di n diverso da 1 ed n, ovvero n non è primo.

Se n|XY e MCD(n,X)=1, allora n|Y (proprietà II dimostrata in precedenza)

Principio fondamentale

Se esistono x e y, tali che
$x^2 \equiv y^2 \pmod{n}$ e $\neg(x \equiv \pm y \pmod{n})$, allora n non è primo.

Pertanto,

se M è primo e $x^2 \bmod M = 1$,

allora x = 1 o x = M-1

Generazione di (grandi) numeri primi

Metodo probabilistico:

```
1. genera M di k bit a caso
2. for (i=0; i<T; i++) // ripeti T volte
       genera a<M a caso
       if (test(a,M)) goto 1 // M non primo
3. return M // Pr(M primo) > 1 - 4^-T
```

test(a,M) = test di Miller-Rabin

Esempio (con Test di Miller-Rabin)

Pr(M primo) > $1 - 4^{-T}$

es. per T = 10,

Pr(M primo) > $1-4^{-10}$ =
= 0,99999904632568359375

Test di Miller-Rabin

```
// M-1 = $b^k…b^2b^1b^0$
d = 1;
for (i=k; i>=0; i--) {
   x = d; d = (d*d)%M; // d ≡ $x^2$ (mod M)
   if  (d = = 1 && x != 1 && x != M-1)
        return(TRUE); // 1 ≡ $x^2$ (mod M)
   if ($b^i$ = = 1) d = (d*a)%M; }
// d = $a^{M-1}$ mod M = 1
if (d != 1) return TRUE; else return (FALSE);
```

Test di Miller-Rabin

```
// M-1 = $b^k…b^2b^1b^0$
d = 1;
for (i=k; i>=0; i--) {
   x = d; d = (d*d)%M; // d ≡ $x^2$ (mod
   if  (d = = 1 && x != 1 && x != M-1)
        return(TRUE); // 1 ≡ $x^2$ (mod M)
   if ($b^i$ = = 1) d = (d*a)%M; }
// d = $a^{M-1}$ mod M = 1
if (d != 1) return TRUE; else return (FALSE);
```

Teorema: se M
è primo e
x^2 mod M = 1,
x = 1 o x = M-1

Test di Miller-Rabin

```
// M-1 = $b^k…b^2b^1b^0$
d = 1;
for (i=k; i>=0; i--) {
   x = d; d = (d*d)%M;
   if  (d = = 1 && x != 1
        return(TRUE); //
   if ($b^i$ = = 1) d = (d*a)%M; }
// d = $a^{M-1}$ mod M = 1
if (d != 1) return TRUE; else return (FALSE);
```

Teorema: se M
primo e a < M,
a^{M-1} mod M = 1
(piccolo teorema
di Fermat)

Esempio (con Test di Miller-Rabin)

Pr(M primo) > $1 - 4^{-T}$

es. per T = 10,

Pr(M primo) > $1-4^{-10} =$
= 0,9999990463256835937 5

Test di Miller-Rabin

```
// M-1 = $b^k \ldots b^2 b^1 b^0$
d = 1;
for (i=k; i>=0; i--) {
   x = d; d = (d*d)%M; // d ≡ $x^2$ (mod M)
   if  (d = = 1 && x != 1 && x != M-1)
        return(TRUE); // 1 ≡ $x^2$ (mod M)
   if ($b^i$ = = 1) d = (d*a)%M; }
// d = $a^{M-1}$ mod M = 1
if (d != 1) return TRUE; else return (FALSE);
```

Test di Miller-Rabin

```
// M-1 = $b^k \ldots b^2 b^1 b^0$
d = 1;
for (i=k; i>=0; i--) {
   x = d; d = (d*d)%M; // d ≡ $x^2$ (mod
   if  (d = = 1 && x != 1 && x != M-1)
        return(TRUE); // 1 ≡ $x^2$ (mod M)
   if ($b^i$ = = 1) d = (d*a)%M; }
// d = $a^{M-1}$ mod M = 1
if (d != 1) return TRUE; else return (FALSE);
```

Teorema: se M
è primo e
x^2 mod M = 1,
x = 1 o x = M-1

Test di Miller-Rabin

```
// M-1 = $b^k \ldots b^2 b^1 b^0$
d = 1;
for (i=k; i>=0; i--) {
   x = d; d = (d*d)%M;
   if  (d = = 1 && x != 1
        return(TRUE); //
   if ($b^i$ = = 1) d = (d*a)%M; }
// d = $a^{M-1}$ mod M = 1
if (d != 1) return TRUE; else return (FALSE);
```

Teorema: se M
primo e a < M,
a^{M-1} mod M = 1
(piccolo teorema
di Fermat)

Esempio (con Test di Miller-Rabin)

Pr(M primo) > $1 - 4^{-T}$

es. per T = 10,

Pr(M primo) > $1-4^{-10}$ =
= 0,9999990463256835937 5

Test di Miller-Rabin

```
// $M-1 = b^k…b^2b^1b^0$
d = 1;
for (i=k; i>=0; i--) {
    x = d; d = (d*d)%M; // $d \equiv x^2$ (mod M)
    if  (d = = 1 && x != 1 && x != M-1)
          return(TRUE); // $1 \equiv x^2$ (mod M)
    if ($b^i$ = = 1) d = (d*a)%M; }
// $d = a^{M-1}$ mod M = 1
if (d != 1) return TRUE; else return (FALSE);
```

Test di Miller-Rabin

```
// $M-1 = b^k…b^2b^1b^0$
d = 1;
for (i=k; i>=0; i--) {
    x = d; d = (d*d)%M; // $d \equiv x^2$ (mod
    if  (d = = 1 && x != 1 && x != M-1)
          return(TRUE); // $1 \equiv x^2$ (mod M)
    if ($b^i$ = = 1) d = (d*a)%M; }
// $d = a^{M-1}$ mod M = 1
if (d != 1) return TRUE; else return (FALSE);
```

Teorema: se M
è primo e
x^2 mod M = 1,
x = 1 o x = M-1

Test di Miller-Rabin

```
// $M-1 = b^k…b^2b^1b^0$
d = 1;
for (i=k; i>=0; i--) {
    x = d; d = (d*d)%M;
    if  (d = = 1 && x != 1
          return(TRUE); //
    if ($b^i$ = = 1) d = (d*a)%M; }
// $d = a^{M-1}$ mod M = 1
if (d != 1) return TRUE; else return (FALSE);
```

Teorema: se M
primo e a < M,
a^{M-1} mod M = 1
(piccolo teorema
di Fermat)

Esempio (con Test di Miller-Rabin)

$Pr(M\ primo) > 1 - 4^{-T}$

es. per T = 10,

$Pr(M\ primo) > 1-4^{-10} =$
$= 0{,}99999904632568359375$

Test di Miller-Rabin

```
// $M-1 = b^k \ldots b^2 b^1 b^0$
d = 1;
for (i=k; i>=0; i--) {
   x = d; d = (d*d)%M; // $d \equiv x^2$ (mod M)
   if  (d = = 1 && x != 1 && x != M-1)
        return(TRUE); // $1 \equiv x^2$ (mod M)
   if ($b^i$ = = 1) d = (d*a)%M; }
// $d = a^{M-1}$ mod M = 1
if (d != 1) return TRUE; else return (FALSE);
```

Test di Miller-Rabin

```
// $M-1 = b^k \ldots b^2 b^1 b^0$
d = 1;
for (i=k; i>=0; i--) {
   x = d; d = (d*d)%M; // $d \equiv x^2$ (mod
   if  (d = = 1 && x != 1 && x != M-1)
        return(TRUE); // $1 \equiv x^2$ (mod M)
   if ($b^i$ = = 1) d = (d*a)%M; }
// $d = a^{M-1}$ mod M = 1
if (d != 1) return TRUE; else return (FALSE);
```

Teorema: se M
è primo e
x^2 mod M = 1,
x = 1 o x = M-1

Test di Miller-Rabin

```
// $M-1 = b^k \ldots b^2 b^1 b^0$
d = 1;
for (i=k; i>=0; i--) {
   x = d; d = (d*d)%M;
   if  (d = = 1 && x != 1
        return(TRUE); // 1 mod M
   if ($b^i$ = = 1) d = (d*a)%M; }
// $d = a^{M-1}$ mod M = 1
if (d != 1) return TRUE; else return (FALSE);
```

Teorema: se M
primo e a < M,
a^{M-1} mod M = 1
(piccolo teorema
di Fermat)

Risultati utili per DH (e per RSA)

Definizione: $\Phi(n)$ è il numero di interi minori di n che sono relativamente primi con n

Teorema (Eulero):
se M,a relativamente primi $a^{\Phi(M)}$ mod M = 1

Corollario (Piccolo teorema di Fermat):
se M primo e a < M, a^{M-1} mod M = 1

Risultati utili per DH (e per RSA)

Proprietà III:
se M,a relativamente primi, allora

$ax \equiv ay \pmod{M} \rightarrow x \equiv y \pmod{M}$

In altre parole, posso dividere per a e semplificare.

Dimostrazione:
Verrà fornita alla fine della trattazione del cifrario RSA.

Risultati utili per DH (e per RSA)

Teorema (Eulero):
se M,a relativamente primi $a^{\Phi(M)}$ mod M = 1

Dimostrazione:
sia $R=\{x_1, x_2,..., x_{\Phi(M)}\}$ l'insieme dei numeri minori di M relativamente primi con M, e sia $S=\{ax_1 \bmod M, ax_2 \bmod M, ..., ax_{\Phi(M)} \bmod M\}$.
Dimostriamo ora che S=R.

Dimostrazione (segue):

Dimostriamo ora che S=R.

1) S incluso in R:

ax_i mod M è relativamente primo con M, poiché sia a sia x_i sono relativamente primi con M

2) R incluso in S:

non ci sono ripetizioni in S, infatti, per la proprietà III, poiché a è relativamente primo con M: se ax_i mod M = ax_j mod M allora $x_i = x_j$.

Risultati utili per DH (e per RSA)

Definizione: $\Phi(n)$ è il numero di interi minori di n che sono relativamente primi con n

Teorema (Eulero):
se M,a relativamente primi $a^{\Phi(M)} \bmod M = 1$

Corollario (Piccolo teorema di Fermat):
se M primo e $a < M$, $a^{M-1} \bmod M = 1$

Risultati utili per DH (e per RSA)

Proprietà III:
se M,a relativamente primi, allora

$ax \equiv ay \pmod{M} \rightarrow x \equiv y \pmod{M}$

In altre parole, posso dividere per a e semplificare.

Dimostrazione:
Verrà fornita alla fine della trattazione del cifrario RSA.

Risultati utili per DH (e per RSA)

Teorema (Eulero):
se M,a relativamente primi $a^{\Phi(M)} \bmod M = 1$

Dimostrazione:
sia $R=\{x_1, x_2, \dots, x_{\Phi(M)}\}$ l'insieme dei numeri minori di M relativamente primi con M, e sia $S=\{ax_1 \bmod M, ax_2 \bmod M, \dots, ax_{\Phi(M)} \bmod M\}$.
Dimostriamo ora che S=R.

Dimostrazione (segue):

Dimostriamo ora che S=R.

1) S incluso in R:

$ax_i \bmod M$ è relativamente primo con M, poiché sia a sia x_i sono relativamente primi con M

2) R incluso in S:

non ci sono ripetizioni in S, infatti, per la proprietà III, poiché a è relativamente primo con M: se $ax_i \bmod M = ax_j \bmod M$ allora $x_i = x_j$.

Risultati utili per DH (e per RSA)

Definizione: $\Phi(n)$ è il numero di interi minori di n che sono relativamente primi con n

Teorema (Eulero):
se M,a relativamente primi $a^{\Phi(M)} \bmod M = 1$

Corollario (Piccolo teorema di Fermat):
se M primo e a < M, $a^{M-1} \bmod M = 1$

Risultati utili per DH (e per RSA)

Proprietà III:
se M,a relativamente primi, allora

$ax \equiv ay \pmod M \rightarrow x \equiv y \pmod M$

In altre parole, posso dividere per a e semplificare.

Dimostrazione:
Verrà fornita alla fine della trattazione del cifrario RSA.

Risultati utili per DH (e per RSA)

Teorema (Eulero):
se M,a relativamente primi $a^{\Phi(M)} \bmod M = 1$

Dimostrazione:

sia $R=\{x_1, x_2,\ldots, x_{\Phi(M)}\}$ l'insieme dei numeri minori di M relativamente primi con M, e sia

$S=\{ax_1 \bmod M, ax_2 \bmod M, \ldots, ax_{\Phi(M)} \bmod M\}$.

Dimostriamo ora che S=R.

Dimostrazione (segue):

Dimostriamo ora che S=R.

1) S incluso in R:

$ax_i \bmod M$ è relativamente primo con M, poiché sia a sia x_i sono relativamente primi con M

2) R incluso in S:

non ci sono ripetizioni in S, infatti, per la proprietà III, poiché a è relativamente primo con M: se $ax_i \bmod M = ax_j \bmod M$ allora $x_i = x_j$.

Risultati utili per DH (e per RSA)

Definizione: $\Phi(n)$ è il numero di interi minori di n che sono relativamente primi con n

Teorema (Eulero):
se M,a relativamente primi $a^{\Phi(M)}$ mod M = 1

Corollario (Piccolo teorema di Fermat):
se M primo e a < M, a^{M-1} mod M = 1

Risultati utili per DH (e per RSA)

Proprietà III:
se M,a relativamente primi, allora

$ax \equiv ay \pmod{M} \rightarrow x \equiv y \pmod{M}$

In altre parole, posso dividere per a e semplificare.

Dimostrazione:
Verrà fornita alla fine della trattazione del cifrario RSA.

Risultati utili per DH (e per RSA)

Teorema (Eulero):
se M,a relativamente primi $a^{\Phi(M)}$ mod M = 1

Dimostrazione:

sia $R=\{x_1, x_2,\ldots, x_{\Phi(M)}\}$ l'insieme dei numeri minori di M relativamente primi con M, e sia $S=\{ax_1 \bmod M, ax_2 \bmod M, \ldots, ax_{\Phi(M)} \bmod M\}$.

Dimostriamo ora che S=R.

Dimostrazione (segue):

Dimostriamo ora che S=R.

1) S incluso in R:

ax_i mod M è relativamente primo con M, poiché sia a sia x_i sono relativamente primi con M

2) R incluso in S:

non ci sono ripetizioni in S, infatti, per la proprietà III, poiché a è relativamente primo con M: se ax_i mod M = ax_j mod M allora $x_i = x_j$.

Dimostrazione (segue):

$R=\{x_1, x_2,\ldots, x_{\Phi(M)}\}= S=\{ax_1 \bmod M, ax_2 \bmod M, \ldots, ax_{\Phi(M)} \bmod M\}$

$x_1x_2\ldots x_{\Phi(M)} \bmod M = (ax_1 \bmod M)(ax_2 \bmod M)\ldots(ax_{\Phi(M)} \bmod M) \bmod M$

$x_1x_2\ldots x_{\Phi(M)} \bmod M = (ax_1)(ax_2)\ldots(ax_{\Phi(M)}) \bmod M =$

$a^{\Phi(M)} (x_1x_2\ldots x_{\Phi(M)}) \bmod M$

$x_1x_2\ldots x_{\Phi(M)} \bmod M = \quad a^{\Phi(M)} (x_1x_2\ldots x_{\Phi(M)}) \bmod M$

Ora, per la proprietà III, siccome $x_1x_2\ldots x_{\Phi(M)}$ è relativamente primo con M, in quanto prodotto di fattori primi con M, posso semplificare, ottenendo.

$1 \bmod M = a^{\Phi(M)} \bmod M$

Per realizzare DH, occorre

- Algoritmo efficiente per calcolare $a^b \bmod q$
- Algoritmo efficiente per generare q primo
- Algoritmo efficiente per generare una radice primitiva α di q

---

Radice primitiva α di q

Definizione:

α è una radice primitiva di q

se

per ogni b<q esiste i tale che $b = \alpha^i \bmod q$

quindi $\{ \alpha^0 \bmod q, \ldots, \alpha^{q-1} \bmod q \} =$
$= \{ 1, \ldots, q-1 \}$

Dato q, come ottenere una radice primitiva α di q?

se a relativamente primo con q non è una radice primitiva di q, allora $a^i \bmod q = 1$ per $i<q-1$.

tuttavia, $a^{q-1} \bmod q = 1$ (Fermat),

quindi $q-1 = k*i$ (i divisore di q-1).

esempio $q=7$, $\alpha = 3$ *(è una radice primitiva)*:

$\alpha^0 = 1, \alpha^1 = 3, \alpha^2 = 2, \alpha^3 = 6, \alpha^4 = 4, \alpha^5 = 5, \alpha^6 = 1$

$\alpha = 2$: $\alpha^0 = 1, \alpha^1 = 2, \alpha^2 = 4, \alpha^3 = 1, \alpha^4 = 2, \alpha^5 = 4, \alpha^6 = 1$

$q-1 = 6 = 2*3 = k*i$, *con* $\alpha^i = 1$.

Dimostrazione (segue):

$R=\{x_1, x_2,\ldots, x_{\Phi(M)}\}= S=\{ax_1 \bmod M, ax_2 \bmod M, \ldots, ax_{\Phi(M)} \bmod M\}$

$x_1x_2\ldots x_{\Phi(M)} \bmod M = (ax_1 \bmod M)(ax_2 \bmod M)\ldots(ax_{\Phi(M)} \bmod M) \bmod M$

$x_1x_2\ldots x_{\Phi(M)} \bmod M = (ax_1)(ax_2)\ldots(ax_{\Phi(M)}) \bmod M =$

$a^{\Phi(M)} (x_1x_2\ldots x_{\Phi(M)}) \bmod M$

$x_1x_2\ldots x_{\Phi(M)} \bmod M = \quad a^{\Phi(M)} (x_1x_2\ldots x_{\Phi(M)}) \bmod M$

Ora, per la proprietà III, siccome $x_1x_2\ldots x_{\Phi(M)}$ è relativamente primo con M, in quanto prodotto di fattori primi con M, posso semplificare, ottenendo.

$1 \bmod M = a^{\Phi(M)} \bmod M$

Per realizzare DH, occorre

- Algoritmo efficiente per calcolare a^bmod q
- Algoritmo efficiente per generare q primo
- Algoritmo efficiente per generare una radice primitiva α di q

---

Radice primitiva α di q

Definizione:

α è una radice primitiva di q

se

per ogni b<q esiste i tale che b = α^i mod q

quindi $\{ \alpha^0 \bmod q, \ldots, \alpha^{q-1} \bmod q \} =$
$= \{ 1, \ldots, q-1 \}$

Dato q, come ottenere una radice primitiva α di q?

se a relativamente primo con q non è una radice primitiva di q, allora a^i mod q = 1 per i<q-1.

tuttavia, a^{q-1} mod q = 1 (Fermat),

quindi q-1 = k*i (i divisore di q-1).

esempio q=7, $\alpha = 3$ (è una radice primitiva):

$\alpha^0 = 1,\ \alpha^1 = 3,\ \alpha^2 = 2,\ \alpha^3 = 6,\ \alpha^4 = 4,\ \alpha^5 = 5,\ \alpha^6 = 1$

$\alpha = 2\text{: } \alpha^0 = 1,\ \alpha^1 = 2,\ \alpha^2 = 4,\ \alpha^3 = 1,\ \alpha^4 = 2,\ \alpha^5 = 4,\ \alpha^6 = 1$

*q-1 = 6 = 2*3 = k*i, con* $\alpha^i = 1$.

Dimostrazione (segue):

$R=\{x_1, x_2,\dots, x_{\Phi(M)}\}= S=\{ax_1 \bmod M, ax_2 \bmod M, \dots, ax_{\Phi(M)} \bmod M\}$

$x_1x_2\dots x_{\Phi(M)} \bmod M = (ax_1 \bmod M)(ax_2 \bmod M)\dots(ax_{\Phi(M)} \bmod M) \bmod M$

$x_1x_2\dots x_{\Phi(M)} \bmod M = (ax_1)(ax_2)\dots(ax_{\Phi(M)}) \bmod M =$

$a^{\Phi(M)} (x_1x_2\dots x_{\Phi(M)}) \bmod M$

$x_1x_2\dots x_{\Phi(M)} \bmod M = \quad a^{\Phi(M)} (x_1x_2\dots x_{\Phi(M)}) \bmod M$

Ora, per la proprietà III, siccome $x_1x_2\dots x_{\Phi(M)}$ è relativamente primo con M, in quanto prodotto di fattori primi con M, posso semplificare, ottenendo.

$1 \bmod M = a^{\Phi(M)} \bmod M$

Per realizzare DH, occorre

- Algoritmo efficiente per calcolare $a^b \bmod q$
- Algoritmo efficiente per generare q primo
- Algoritmo efficiente per generare una radice primitiva α di q

---

Radice primitiva α di q

Definizione:

α è una radice primitiva di q

se

per ogni b<q esiste i tale che $b = \alpha^i \bmod q$

quindi $\{ \alpha^0 \bmod q, \dots, \alpha^{q-1} \bmod q \} =$
$= \{ 1, \dots, q-1 \}$

Dato q, come ottenere una radice primitiva α di q?

se a relativamente primo con q non è una radice primitiva di q, allora $a^i \bmod q = 1$ per i<q-1.
tuttavia, $a^{q-1} \bmod q = 1$ (Fermat),
quindi q-1 = k*i (i divisore di q-1).

esempio q=7, $\alpha = 3$ (è una radice primitiva):
$\alpha^0 = 1, \alpha^1 = 3, \alpha^2 = 2, \alpha^3 = 6, \alpha^4 = 4, \alpha^5 = 5, \alpha^6 = 1$

$\alpha = 2$: $\alpha^0 = 1, \alpha^1 = 2, \alpha^2 = 4, \alpha^3 = 1, \alpha^4 = 2, \alpha^5 = 4, \alpha^6 = 1$
*q-1 = 6 = 2*3 = k*i, con* $\alpha^i = 1$.

Dimostrazione (segue):

$R=\{x_1, x_2,\ldots, x_{\Phi(M)}\}= S=\{ax_1 \bmod M, ax_2 \bmod M, \ldots, ax_{\Phi(M)} \bmod M\}$

$x_1x_2\ldots x_{\Phi(M)} \bmod M = (ax_1 \bmod M)(ax_2 \bmod M)\ldots(ax_{\Phi(M)} \bmod M) \bmod M$

$x_1x_2\ldots x_{\Phi(M)} \bmod M = (ax_1)(ax_2)\ldots(ax_{\Phi(M)}) \bmod M =$

$a^{\Phi(M)} (x_1x_2\ldots x_{\Phi(M)}) \bmod M$

$x_1x_2\ldots x_{\Phi(M)} \bmod M = \quad a^{\Phi(M)} (x_1x_2\ldots x_{\Phi(M)}) \bmod M$

Ora, per la proprietà III, siccome $x_1x_2\ldots x_{\Phi(M)}$ è relativamente primo con M, in quanto prodotto di fattori primi con M, posso semplificare, ottenendo.

$1 \bmod M = a^{\Phi(M)} \bmod M$

Per realizzare DH, occorre

- Algoritmo efficiente per calcolare a^bmod q
- Algoritmo efficiente per generare q primo
- Algoritmo efficiente per generare una radice primitiva α di q

---

Radice primitiva α di q

Definizione:

α è una radice primitiva di q

se

per ogni b<q esiste i tale che $b = \alpha^i \bmod q$

quindi $\{\alpha^0 \bmod q, \ldots, \alpha^{q-1} \bmod q\} =$
$= \{1, \ldots, q-1\}$

Dato q, come ottenere una radice primitiva α di q?

se a relativamente primo con q non è una radice primitiva di q, allora $a^i \bmod q = 1$ per i<q-1.

tuttavia, $a^{q-1} \bmod q = 1$ (Fermat),

quindi q-1 = k*i (i divisore di q-1).

esempio q=7, $\alpha = 3$ (è una radice primitiva):

$\alpha^0 = 1,\ \alpha^1 = 3,\ \alpha^2 = 2,\ \alpha^3 = 6,\ \alpha^4 = 4,\ \alpha^5 = 5,\ \alpha^6 = 1$

$\alpha = 2$: $\alpha^0 = 1,\ \alpha^1 = 2,\ \alpha^2 = 4,\ \alpha^3 = 1,\ \alpha^4 = 2,\ \alpha^5 = 4,\ \alpha^6 = 1$

*q-1 = 6 = 2*3 = k*i, con* $\alpha^i = 1$.

Ripetizione dei numeri generati

se a relativamente primo con q non è una radice primitiva di q, allora $a^i \bmod q = 1$ per $i<q-1$.

quindi il ciclo si ripete dopo i:

$\alpha^0 = 1,\ \alpha^1 = \alpha,\ \ldots,\ \alpha^\iota = 1,\ \alpha^{\iota+1} = \alpha,\ \ldots,\ \alpha^{2\iota} = 1,\ \alpha^{2\iota+1} = \alpha,\ \ldots,$

da $\alpha^1 = \alpha$ fino a $\alpha^{\iota-1}$ non ci sono valori pari a 1. Stessa cosa da $\alpha^{\iota+1} = \alpha$ fino a $\alpha^{2\iota-1}$, perché il ciclo si ripete. Quindi i soli valori pari a 1 sono per $\alpha^{\kappa*\iota}$.

Dato che $a^{q-1} \bmod q = 1$, dovrà essere $q-1= k*i$, per qualche k.

Dato q, come ottenere una radice primitiva α di q?

- *Algoritmo (scegliere q t.c. sia facile fattorizzare q-1)*:

- generare $\alpha<q$ a caso, tale che α sia relativamente primo con q
- fattorizzare q-1 ($f_1 f_2 \ldots f_j = q-1$)
- se esiste f tale che $\alpha^{((q-1)/f)} \bmod q = 1$
  - allora torna al passo 1,
  - altrimenti restituisci α

Perché?

se a non è una radice primitiva,

allora $(\exists i<q-1)\ a^i \bmod q = 1$, e $(\exists k)\ i*k=q-1$

$a^{i*k} \bmod q = 1$, e esplicitando i fattori primi

$a^{q-1} \bmod q = (a^{i_1*i_2*\ldots*i_s})^{k_1*k_2*..*k_t} \bmod q = 1,$

$a^i \bmod q = a^{i_1*i_2*\ldots*i_s} \bmod q = 1,$

quindi $(a^{i_1*i_2*\ldots*i_s})^x \bmod 1 = 1$, per ogni x,

e, in particolare, per $x=k_1*k_2*\ldots*k_{t-1}$,

$a^{i_1*i_2*\ldots*i_s\ *\ k_1*k_2*\ldots*k_{t-1}} \bmod q = a^{(q-1)/k_t} \bmod q = 1$

Problemi di DH

- Protegge rispetto a lettura non autorizzata dei dati trasmessi su rete, non protegge rispetto ad attacchi di tipo attivo

- Permette solo di scambiare chiavi, non permette di cifrare direttamente messaggi arbitrari, quindi non può essere usato direttamente per autenticare/firmare

Ripetizione dei numeri generati

se a relativamente primo con q non è una radice primitiva di q, allora a^i mod q = 1 per i<q-1.
quindi il ciclo si ripete dopo i:

$\alpha^0 = 1, \alpha^1 = \alpha, \ldots, \alpha^\iota = 1, \alpha^{\iota+1} = \alpha, \ldots, \alpha^{2\iota} = 1, \alpha^{2\iota+1} = \alpha, \ldots,$

da $\alpha^1 = \alpha$ fino a $\alpha^{\iota-1}$ non ci sono valori pari a 1. Stessa cosa da $\alpha^{\iota+1} = \alpha$ fino a $\alpha^{2\iota-1}$, perché il ciclo si ripete. Quindi i soli valori pari a 1 sono per $\alpha^{\kappa*\iota}$.

Dato che a^{q-1} mod q = 1, dovrà essere q-1= k*i, per qualche k.

Dato q, come ottenere una radice primitiva α di q?

- *Algoritmo (scegliere q t.c. sia facile fattorizzare q-1)*:

- generare α<q a caso, tale che α sia relativamente primo con q
- fattorizzare q-1 ($f_1 f_2 \ldots f_j$ = q-1)
- se esiste f tale che $\alpha^{((q-1)/f)}$mod q = 1
  allora torna al passo 1,
  altrimenti restituisci α

Perché?

se a non è una radice primitiva,
allora (∃i<q-1) a^i mod q = 1, e (∃k) i*k=q-1
a^{i*k} mod q = 1, e esplicitando i fattori primi
a^{q-1} mod q = $(a^{i_1*i_2*\ldots*i_s})^{k_1*k_2*..*k_t}$ mod q = 1,
a^i mod q = $a^{i_1*i_2*\ldots*i_s}$ mod q = 1,
quindi $(a^{i_1*i_2*\ldots*i_s})^x$ mod 1 = 1, per ogni x,
e, in particolare, per $x=k_1*k_2*\ldots*k_{t-1}$,
$a^{i_1*i_2*\ldots*i_s * k_1*k_2*\ldots*k_{t-1}}$ mod q = $a^{(q-1)/k_t}$ mod q = 1

Problemi di DH

- Protegge rispetto a lettura non autorizzata dei dati trasmessi su rete, non protegge rispetto ad attacchi di tipo attivo

- Permette solo di scambiare chiavi, non permette di cifrare direttamente messaggi arbitrari, quindi non può essere usato direttamente per autenticare/firmare

Ripetizione dei numeri generati

se a relativamente primo con q non è una radice primitiva di q, allora $a^i \bmod q = 1$ per $i<q-1$.

quindi il ciclo si ripete dopo i:

$\alpha^0 = 1, \alpha^1 = \alpha, \ldots, \alpha^\iota = 1, \alpha^{\iota+1} = \alpha, \ldots, \alpha^{2\iota} = 1, \alpha^{2\iota+1} = \alpha, \ldots,$

da $\alpha^1 = \alpha$ fino a $\alpha^{\iota-1}$ non ci sono valori pari a 1. Stessa cosa da $\alpha^{\iota+1} = \alpha$ fino a $\alpha^{2\iota-1}$, perché il ciclo si ripete. Quindi i soli valori pari a 1 sono per $\alpha^{\kappa*\iota}$.

Dato che $a^{q-1} \bmod q = 1$, dovrà essere $q-1= k*i$, per qualche k.

Dato q, come ottenere una radice primitiva α di q?

- *Algoritmo (scegliere q t.c. sia facile fattorizzare q-1)*:

- generare $\alpha<q$ a caso, tale che α sia relativamente primo con q
- fattorizzare q-1 ($f_1 f_2 \ldots f_j = q-1$)
- se esiste f tale che $\alpha^{((q-1)/f)} \bmod q = 1$
  allora torna al passo 1,
  altrimenti restituisci α

Perché?

se a non è una radice primitiva,

allora ($\exists i<q-1$) $a^i \bmod q = 1$, e ($\exists k$) $i*k=q-1$

$a^{i*k} \bmod q = 1$, e esplicitando i fattori primi

$a^{q-1} \bmod q = (a^{i_1*i_2*\ldots*i_s})^{k_1*k_2*..*k_t} \bmod q = 1,$

$a^i \bmod q = a^{i_1*i_2*\ldots*i_s} \bmod q = 1,$

quindi $(a^{i_1*i_2*\ldots*i_s})^x \bmod 1 = 1$, per ogni x,

e, in particolare, per $x=k_1*k_2*\ldots*k_{t-1}$,

$a^{i_1*i_2*\ldots*i_s * k_1*k_2*\ldots*k_{t-1}} \bmod q = a^{(q-1)/k_t} \bmod q = 1$

Problemi di DH

- Protegge rispetto a lettura non autorizzata dei dati trasmessi su rete, non protegge rispetto ad attacchi di tipo attivo

- Permette solo di scambiare chiavi, non permette di cifrare direttamente messaggi arbitrari, quindi non può essere usato direttamente per autenticare/firmare

Ripetizione dei numeri generati

se a relativamente primo con q non è una radice primitiva di q, allora a^i mod q = 1 per i<q-1.

quindi il ciclo si ripete dopo i:

$\alpha^0 = 1, \alpha^1 = \alpha, \dots, \alpha^\iota = 1, \alpha^{\iota+1} = \alpha, \dots, \alpha^{2\iota} = 1, \alpha^{2\iota+1} = \alpha, \dots,$

da $\alpha^1 = \alpha$ fino a $\alpha^{\iota-1}$ non ci sono valori pari a 1. Stessa cosa da $\alpha^{\iota+1} = \alpha$ fino a $\alpha^{2\iota-1}$, perché il ciclo si ripete. Quindi i soli valori pari a 1 sono per $\alpha^{\kappa*\iota}$.

Dato che a^{q-1} mod q = 1, dovrà essere q-1= k*i, per qualche k.

Dato q, come ottenere una radice primitiva α di q?

- *Algoritmo (scegliere q t.c. sia facile fattorizzare q-1)*:

- generare α<q a caso, tale che α sia relativamente primo con q
- fattorizzare q-1 ($f_1 f_2 \dots f_j$ = q-1)
- se esiste f tale che $\alpha^{((q-1)/f)}$mod q = 1

  allora torna al passo 1,

  altrimenti restituisci α

Perché?

se a non è una radice primitiva,

allora ($\exists$i<q-1) a^i mod q = 1, e ($\exists$k) i*k=q-1

a^{i*k} mod q = 1, e esplicitando i fattori primi

a^{q-1} mod q = $(a^{i_1*i_2*\dots*i_s})^{k_1*k_2*..*k_t}$ mod q = 1,

a^i mod q = $a^{i_1*i_2*\dots*i_s}$ mod q = 1,

quindi $(a^{i_1*i_2*\dots*i_s})^x$ mod 1 = 1, per ogni x,

e, in particolare, per $x=k_1*k_2*\dots*k_{t-1}$,

$a^{i_1*i_2*\dots*i_s * k_1*k_2*\dots*k_{t-1}}$ mod q = $a^{(q-1)/k_t}$ mod q = 1

Problemi di DH

- Protegge rispetto a lettura non autorizzata dei dati trasmessi su rete, non protegge rispetto ad attacchi di tipo attivo

- Permette solo di scambiare chiavi, non permette di cifrare direttamente messaggi arbitrari, quindi non può essere usato direttamente per autenticare/firmare

Scambio di chiavi di Diffie-Hellman

A

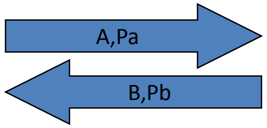
genera Sa < q

Pa = α^{Sa} % q

B

genera Sb < q

Pb = α^{Sb} % q



ottiene K = Pb^{Sa}

ottiene K = Pa^{Sb}

Attacco ‘man in the middle’ allo scambio di chiavi di Diffie-Hellman

A

Sa<q

Pa=α^{Sa}%q

C

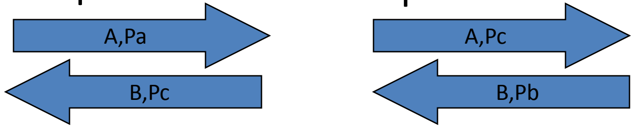
Sc<q

Pc=α^{Sc}%q

B

Sb<q

Pb=α^{Sb}%q



K1=Pc^{Sa}

K1=Pa^{Sc}

K2=Pb^{Sc}

K2=Pc^{Sb}

Soluzione

Utilizzare canale autenticato per lo scambio di Pa, Pb

Problemi di DH

- Protegge rispetto a lettura non autorizzata dei dati trasmessi su rete, non protegge rispetto ad attacchi di tipo attivo
- Permette solo di scambiare chiavi, non permette di cifrare direttamente messaggi arbitrari, quindi non può essere usato direttamente per autenticare/firmare

Scambio di chiavi di Diffie-Hellman

A

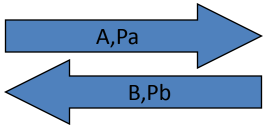
genera Sa < q

Pa = $\alpha^{Sa} \% q$

B

genera Sb < q

Pb = $\alpha^{Sb} \% q$



ottiene K = Pb^{Sa}

ottiene K = Pa^{Sb}

Attacco ‘<u>man in the middle</u>’ allo scambio di chiavi di Diffie-Hellman

A

Sa<q

$Pa=\alpha^{Sa}\%q$

C

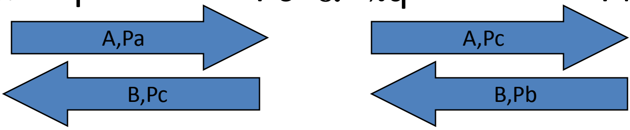
Sc<q

$Pc=\alpha^{Sc}\%q$

B

Sb<q

$Pb=\alpha^{Sb}\%q$



$K1=Pc^{Sa}$

$K1=Pa^{Sc}$

$K2=Pb^{Sc}$

$K2=Pc^{Sb}$

Soluzione

Utilizzare canale autenticato per lo scambio di Pa, Pb

Problemi di DH

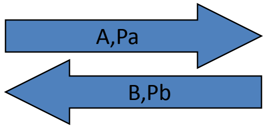
- Protegge rispetto a lettura non autorizzata dei dati trasmessi su rete, non protegge rispetto ad attacchi di tipo attivo
- Permette solo di scambiare chiavi, non permette di cifrare direttamente messaggi arbitrari, quindi non può essere usato direttamente per autenticare/firmare

Scambio di chiavi di Diffie-Hellman

A

genera Sa < q

Pa = α^{Sa} % q



B

genera Sb < q

Pb = α^{Sb} % q

ottiene K = Pb^{Sa}

ottiene K = Pa^{Sb}

Attacco 'man in the middle' allo scambio di chiavi di Diffie-Hellman

A

Sa<q

Pa=α^{Sa}%q

C

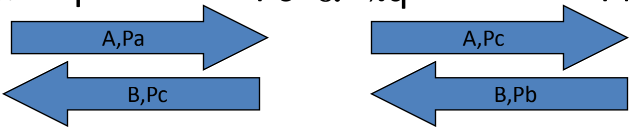
Sc<q

Pc=α^{Sc}%q

B

Sb<q

Pb=α^{Sb}%q



K1=Pc^{Sa}

K1=Pa^{Sc}

K2=Pb^{Sc}

K2=Pc^{Sb}

Soluzione

Utilizzare canale autenticato per lo scambio di Pa, Pb

Problemi di DH

- Protegge rispetto a lettura non autorizzata dei dati trasmessi su rete, non protegge rispetto ad attacchi di tipo attivo
- Permette solo di scambiare chiavi, non permette di cifrare direttamente messaggi arbitrari, quindi non può essere usato direttamente per autenticare/firmare

Scambio di chiavi di Diffie-Hellman

A

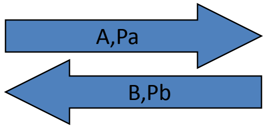
genera Sa < q

Pa = α^{Sa} % q

B

genera Sb < q

Pb = α^{Sb} % q



ottiene K = Pb^{Sa}

ottiene K = Pa^{Sb}

Attacco ‘<u>man in the middle</u>’ allo scambio di chiavi di Diffie-Hellman

A

Sa<q

Pa=α^{Sa}%q

C

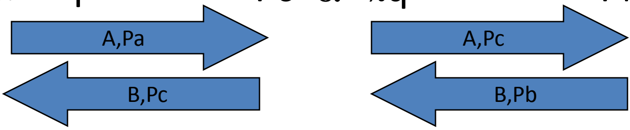
Sc<q

Pc=α^{Sc}%q

B

Sb<q

Pb=α^{Sb}%q



K1=Pc^{Sa}

K1=Pa^{Sc}

K2=Pb^{Sc}

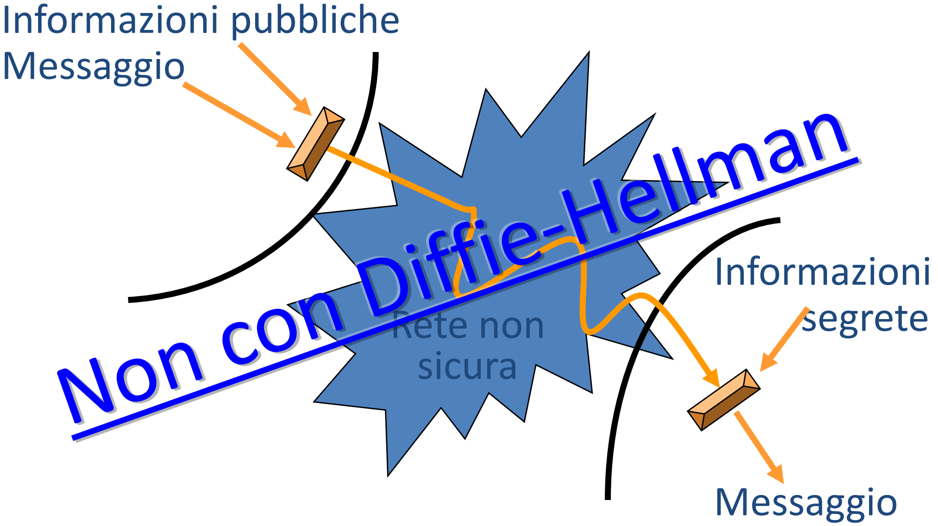
K2=Pc^{Sb}

Soluzione

Utilizzare canale autenticato per lo scambio di Pa, Pb

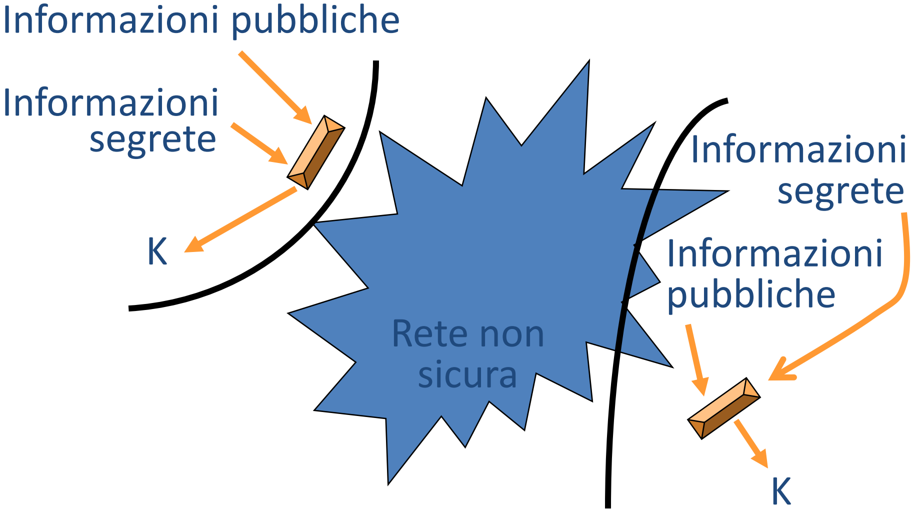
Problemi di DH

- Protegge rispetto a lettura non autorizzata dei dati trasmessi su rete, non protegge rispetto ad attacchi di tipo attivo
- Permette solo di scambiare chiavi, non permette di cifrare direttamente messaggi arbitrari, quindi non può essere usato direttamente per autenticare/firmare

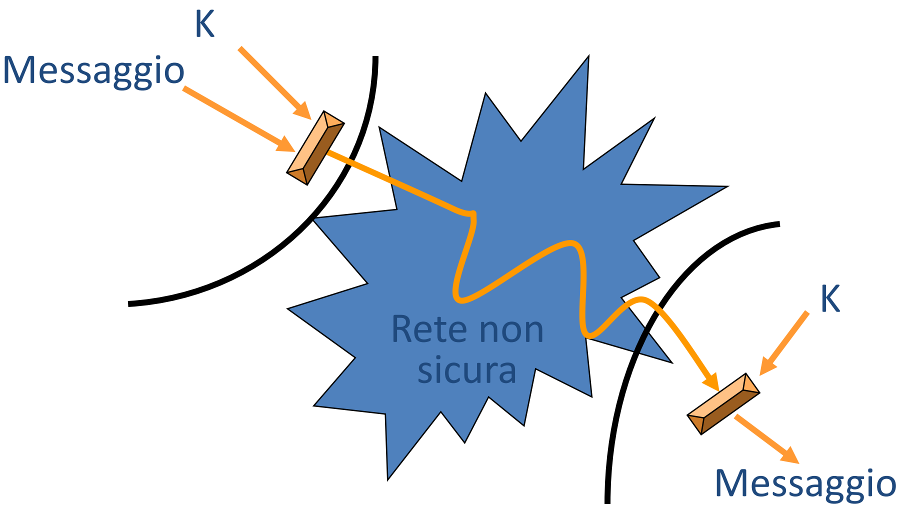


Non con Diffie-Hellman

Cifratura asimmetrica



Scambio chiavi DH ...

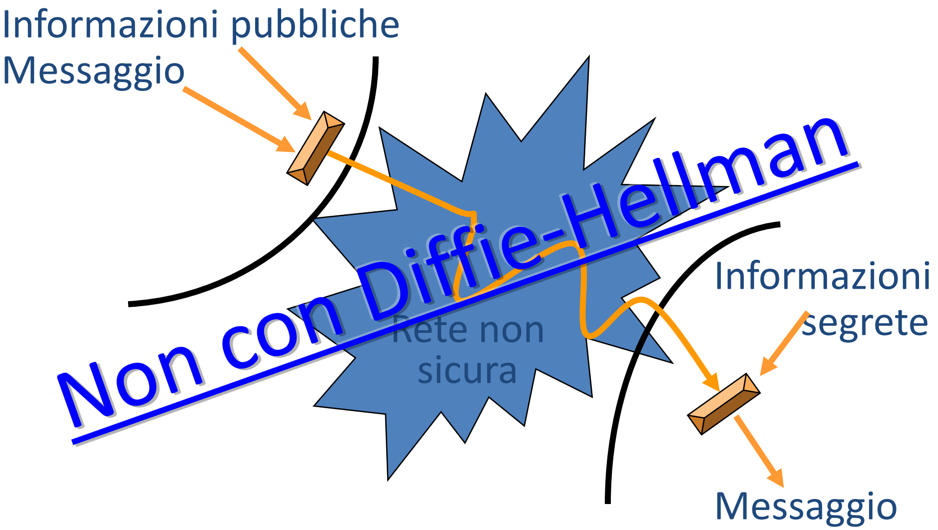


… poi cifratura simmetrica

Non con Diffie-Hellman

Si con RSA

Cifratura asimmetrica



Informazioni pubbliche

Messaggio

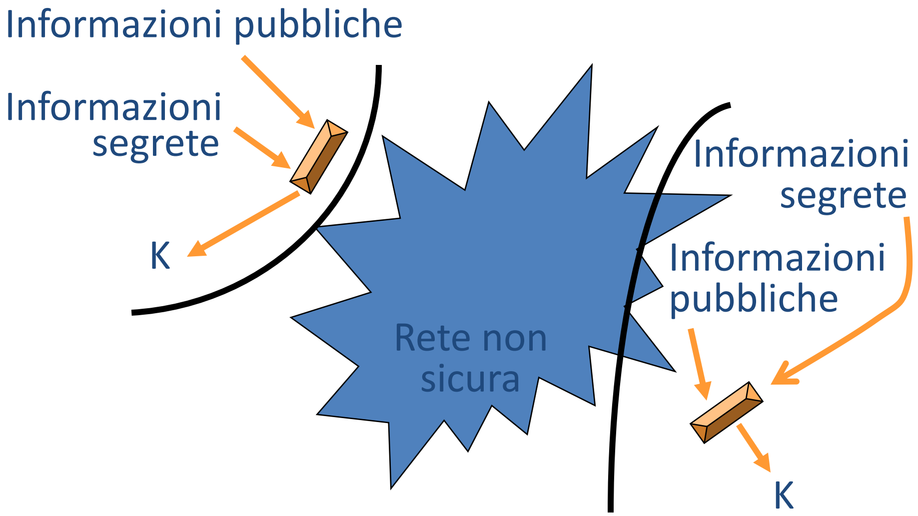
Rete non sicura

Informazioni segrete

Messaggio

Non con Diffie-Hellman

Cifratura asimmetrica



Informazioni pubbliche

Informazioni segrete

K

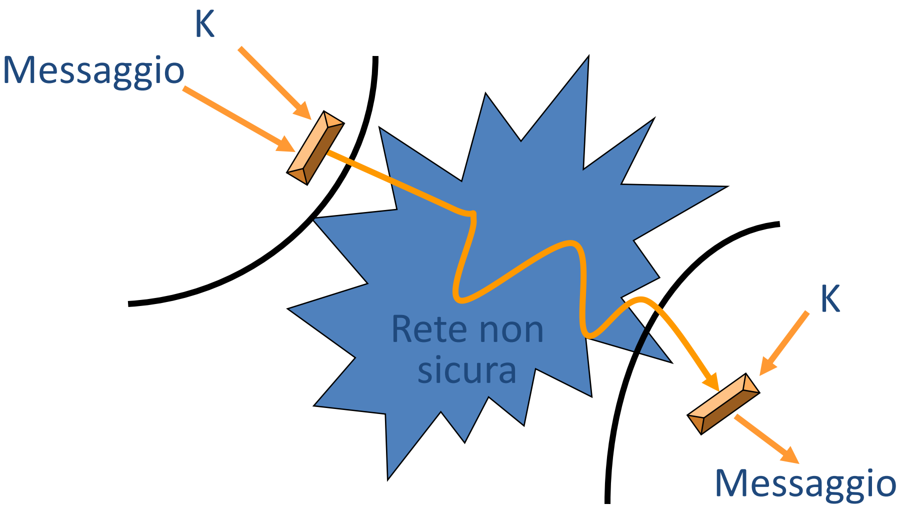
Rete non sicura

Informazioni segrete

Informazioni pubbliche

K

Scambio chiavi DH ...



K

Messaggio

Rete non sicura

K

Messaggio

... poi cifratura simmetrica

$K(B)^+$

Messaggio

A

B

Rete non sicura

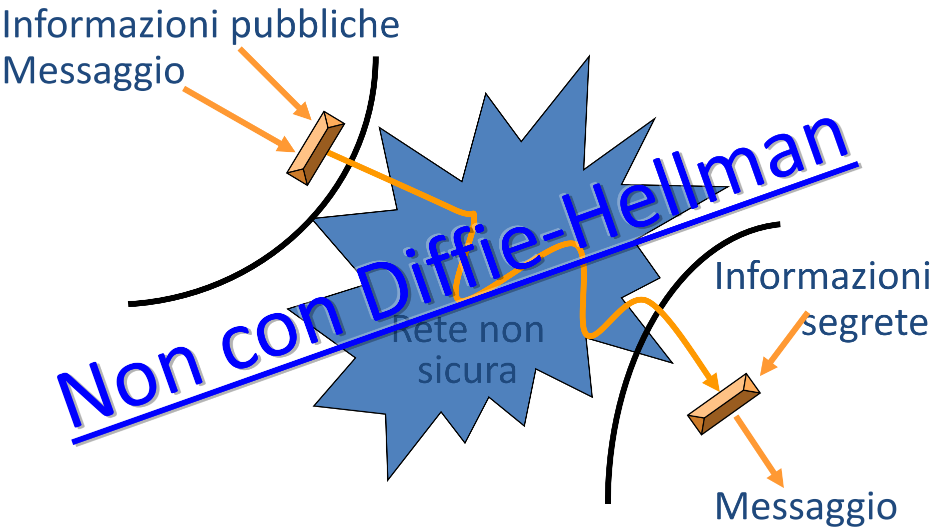
$K(B)^-$

Messaggio

Non con Diffie-Hellman

Si con RSA

Cifratura asimmetrica



Informazioni pubbliche
Messaggio

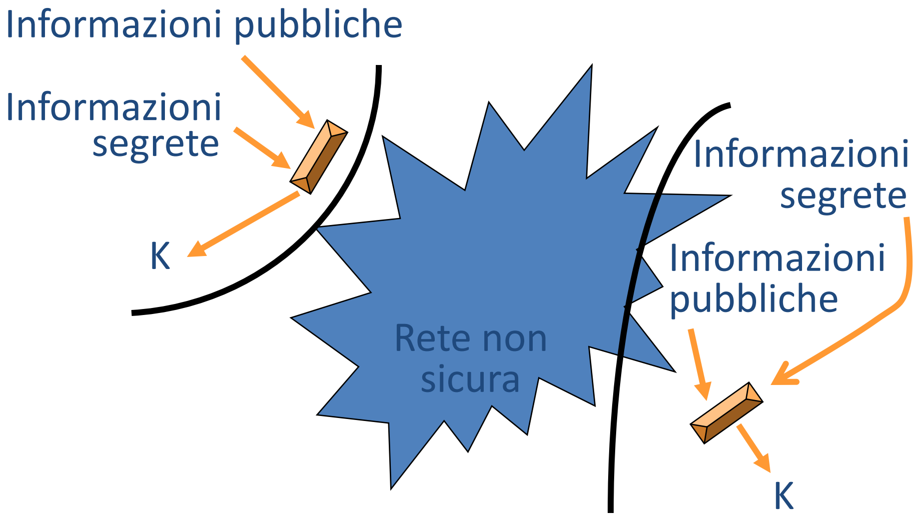
Rete non sicura

Informazioni segrete

Messaggio

Non con Diffie-Hellman

Cifratura asimmetrica



Informazioni pubbliche

Informazioni segrete

K

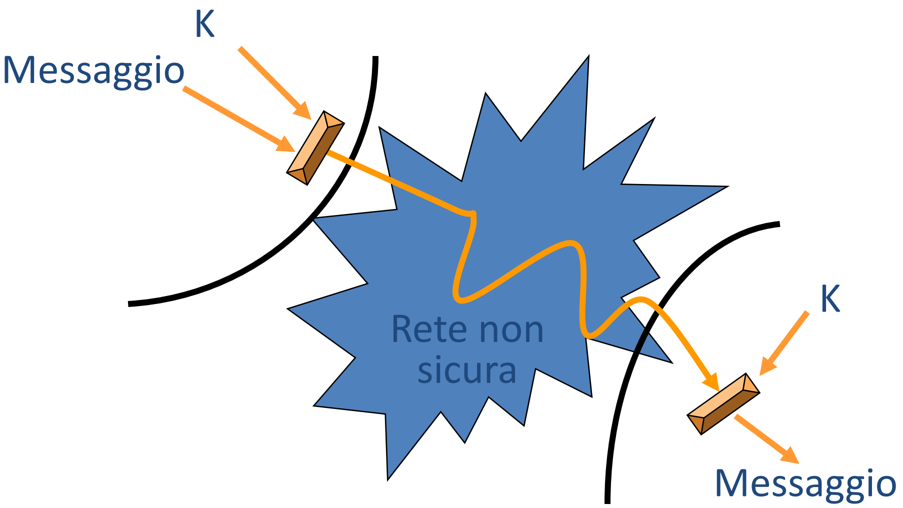
Rete non sicura

Informazioni segrete

Informazioni pubbliche

K

Scambio chiavi DH ...



K
Messaggio

Rete non sicura

K

Messaggio

… poi cifratura simmetrica

$K(B)^+$
Messaggio

A

B

Rete non sicura

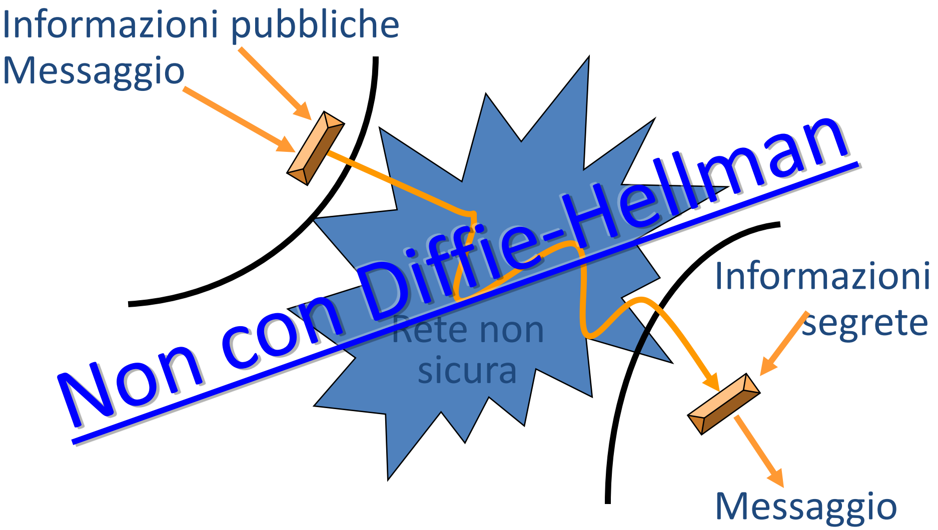
$K(B)^-$

Messaggio

Non con Diffie-Hellman

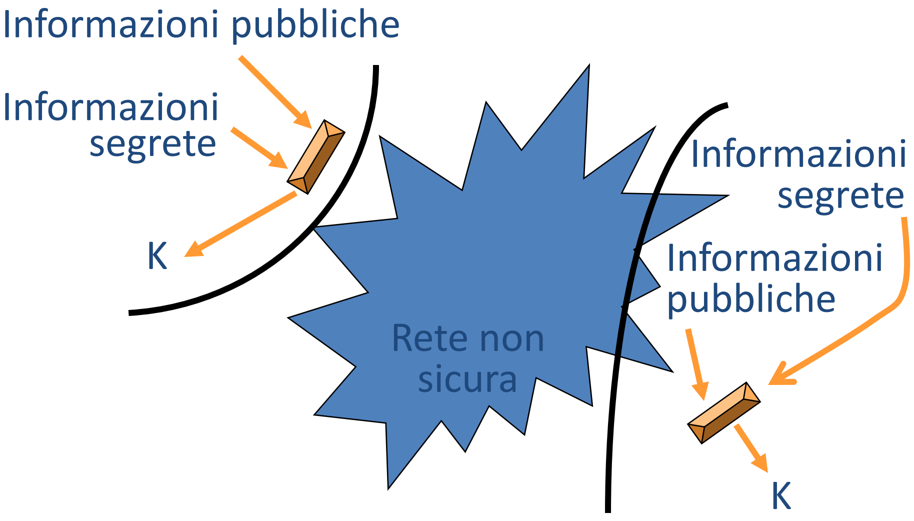
Si con RSA

Cifratura asimmetrica

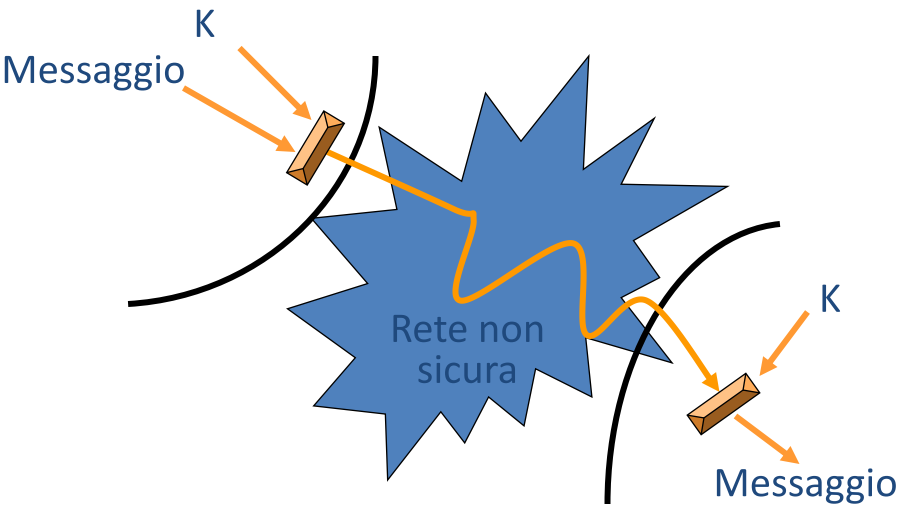


Non con Diffie-Hellman

Cifratura asimmetrica



Scambio chiavi DH ...



… poi cifratura simmetrica

Non con Diffie-Hellman

Si con RSA

Cifratura asimmetrica

Riferimenti

W. Trappe & L.C. Washington
Crittografia con elementi di teoria dei codici
Seconda edizione, 2009
(Pearson/Prentice Hall)