

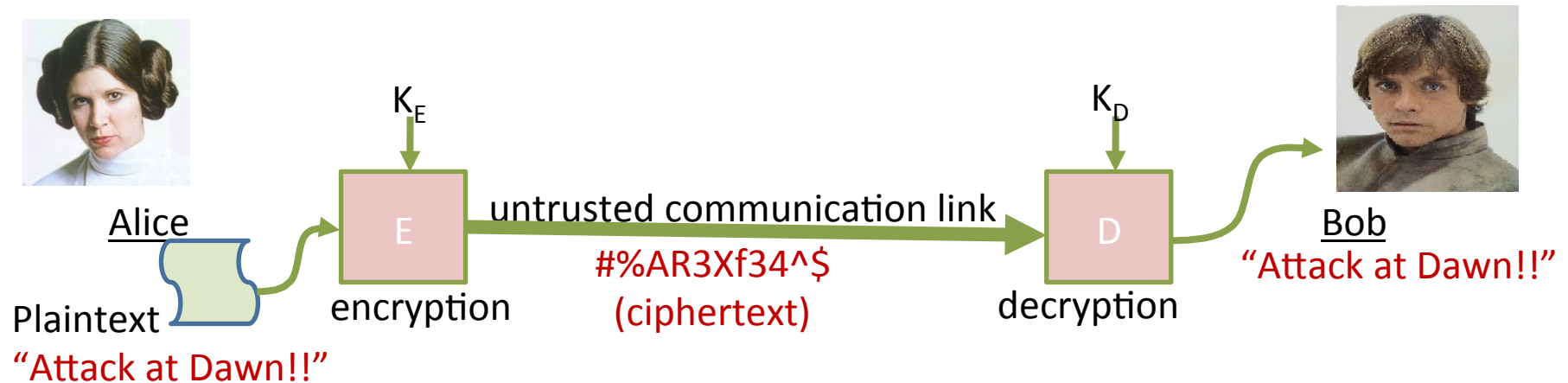
# RSA and Public Key Cryptography

Chester Rebeiro  
IIT Madras

# Ciphers

- Symmetric Algorithms
  - Encryption and Decryption use the same key
  - i.e.  $K_E = K_D$
  - Examples:
    - Block Ciphers : DES, AES, PRESENT, etc.
    - Stream Ciphers : A5, Grain, etc.
- Asymmetric Algorithms
  - Encryption and Decryption keys are different
  - $K_E \neq K_D$
  - Examples:
    - RSA
    - ECC

# Asymmetric Key Algorithms



The Key  $K$  is a secret

Encryption Key  $K_E$  not same as decryption key  $K_D$

$K_E$  known as Bob's public key;  
 $K_D$  is Bob's private key

Advantage : No need of secure  
key exchange between Alice and  
Bob

Asymmetric key algorithms based on trapdoor one-way functions

# One Way Functions

- Easy to compute in one direction
- Once done, it is difficult to inverse



**Press to lock  
(can be easily done)**



**Once locked it is  
difficult to unlock  
without a key**

# Trapdoor One Way Function

- One way function with a trapdoor
- Trapdoor is a special function that if possessed can be used to easily invert the one way



**Locked**  
**(difficult to unlock)**

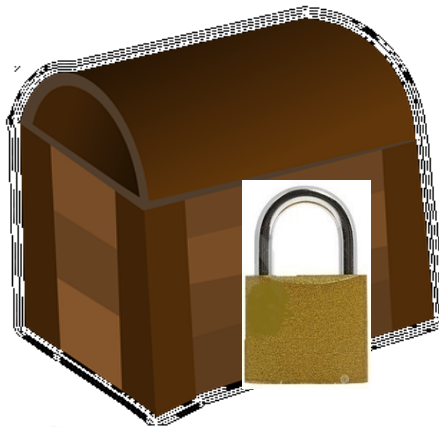


**trapdoor**

**Easily Unlocked**

# Public Key Cryptography (An Analogy)

- Alice puts message into box and locks it
- Only Bob, who has the key to the lock can open it and read the message



# Mathematical Trapdoor One way functions

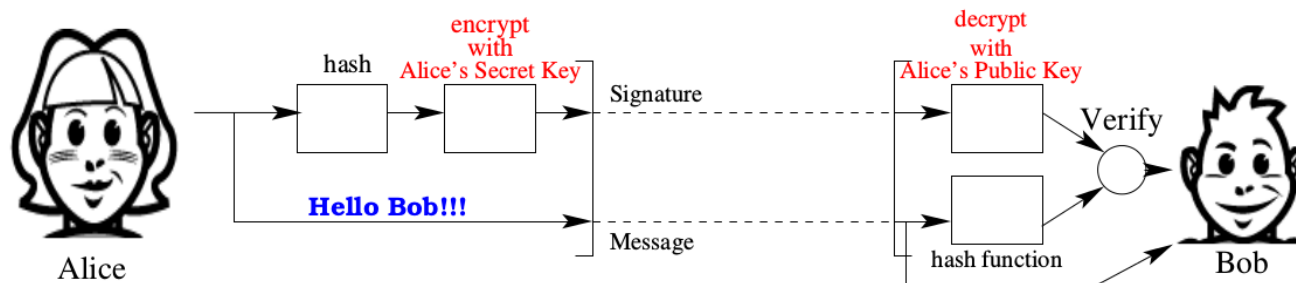
- Examples
  - Integer Factorization (in NP, maybe NP-complete)
    - Given  $P, Q$  are two primes
    - and  $N = P * Q$ 
      - It is easy to compute  $N$
      - However given  $N$  it is difficult to factorize into  $P$  and  $Q$
    - Used in cryptosystems like RSA
  - Discrete Log Problem (in NP)
    - Consider  $b$  and  $g$  are elements in a finite group and  $b^k = g$ , for some  $k$
    - Given  $b$  and  $k$  it is easy to compute  $g$
    - Given  $b$  and  $g$  it is difficult to determine  $k$
    - Used in cryptosystems like Diffie-Hellman
    - A variant used in ECC based crypto-systems

# Applications of Public key Cryptography

- Encryption
- Digital Signature :

“Is this message really from Alice?”

- Alice signs by ‘encrypting’ with private key
- Anyone can verify signature by ‘decrypting’ with Alice’s public key
- Why it works?
  - Only Alice, who owns the private key could have signed





# Applications of Public key Cryptography

Diffie-Hellman  
Key Exchange

- **Key Establishment :**

“Alice and Bob want to use a block cipher for encryption. How do they agree upon the secret key”



Alice and Bob agree upon a prime  $p$  and a generator  $g$ .  
This is public information



choose a secret  $a$   
compute  $A = g^a \bmod p$

choose a secret  $b$   
compute  $B = g^b \bmod p$

Compute  $K = B^a \bmod p$  ←  $B$        $A$  → Compute  $K = A^b \bmod p$

$$A^b \bmod p = (g^a)^b \bmod p = (g^b)^a \bmod p = B^a \bmod p$$

# RSA



Shamir, Rivest, Adleman (1977)

# More Number Theory

Mathematical Background

# RSA : Key Generation

Bob first creates a pair of keys (one public the other private)



1. Generate two large primes  $p, q$  ( $p \neq q$ )
2. Compute  $n = p \times q$  and  $\phi(n) = (p - 1)(q - 1)$
3. Choose a random  $b$  ( $1 < b < \phi(n)$ ) and  $\gcd(b, \phi(n)) = 1$
4. Compute  $a = b^{-1} \bmod(\phi(n))$

*Bob's public key is  $(n, b)$*

*Bob's private key is  $(p, q, a)$*

Given the private key it is easy to compute the public key

Given the public key it is difficult to derive the private key

# RSA Encryption & Decryption



Encryption

$$e_K(x) = y = x^b \bmod n$$

where  $x \in \mathbb{Z}_n$



Decryption

$$d_K(x) = y^a \bmod n$$

# RSA Example

1. Take two primes  $p = 653$  and  $q = 877$
2.  $n = 653 \times 877 = 572681$ ;  $\phi(n) = 652 \times 876 = 571152$
3. Choose public key  $b = 13$ ; note that  $\gcd(13, 571152) = 1$
4. Private key  $a = 395413 = 13^{-1} \bmod 571152$

*Message*  $x = 12345$

*encryption*:  $y = 12345^{13} \bmod 572681 \equiv 536754$

*decryption*:  $x = 536754^{395413} \bmod 572681 \equiv 12345$

# Correctness



*when  $x \in Z_n$  and  $\gcd(x, n) = 1$*



Encryption

$$e_K(x) = y = x^b \bmod n$$

where  $x \in Z_n$

Decryption

$$d_K(x) = y^a \bmod n$$

$$\begin{aligned} y^a &\equiv (x^b)^a \bmod n \\ &\equiv (x^{ab}) \bmod n \\ &\equiv (x^{t\phi(n)+1}) \bmod n \\ &\equiv (x^{t\phi(n)} x) \bmod n \\ &\equiv x \end{aligned}$$

$$\begin{aligned} ab &\equiv 1 \bmod \phi(n) \\ ab - 1 &= t\phi(n) \\ ab &= t\phi(n) + 1 \end{aligned}$$

From Fermat's theorem

# Correctness

when  $x \in Z_n$  and  $\gcd(x, n) \neq 1$

Since  $n = pq$ ,  $\gcd(x, n) = p$  or  $\gcd(x, n) = q$

*If*

$$x \equiv x^{ab} \pmod{p}$$

$$x \equiv x^{ab} \pmod{q}$$

$$\Rightarrow x \equiv x^{ab} \pmod{n}$$

(by CRT)

Assume  $\gcd(n, x) = p$

$$\Rightarrow p \mid x \Rightarrow pk = x$$

$$LHS : x \pmod{p} \equiv pk \pmod{p} \equiv 0$$

$$RHS : x^{ab} \pmod{p} \equiv 0$$

$\because \gcd(p, x) = p$  it implies  $\gcd(q, x) = 1$

$$x^{ab} \pmod{q} \equiv x^{t\phi(n)+1} \pmod{q}$$

$$\equiv x^{t\phi(p)\phi(q)+1} \pmod{q}$$

$$\equiv (x^{\phi(q)})^{t\phi(p)} \cdot x \pmod{q}$$

$$\equiv (1)^{t\phi(p)} \cdot x \pmod{q} \equiv x$$



# RSA Implementation

$$y = x^c \bmod n$$

**Algorithm** : SQUARE-AND-MULTIPLY( $x, c, n$ )

```

 $z \leftarrow 1$ 
for  $i \leftarrow \ell - 1$  downto 0
  do  $\begin{cases} z \leftarrow z^2 \bmod n \\ \text{if } c_i = 1 \\ \text{then } z \leftarrow (z \times x) \bmod n \end{cases}$ 
return ( $z$ )
    
```

$$c = 23 = (10111)_2$$

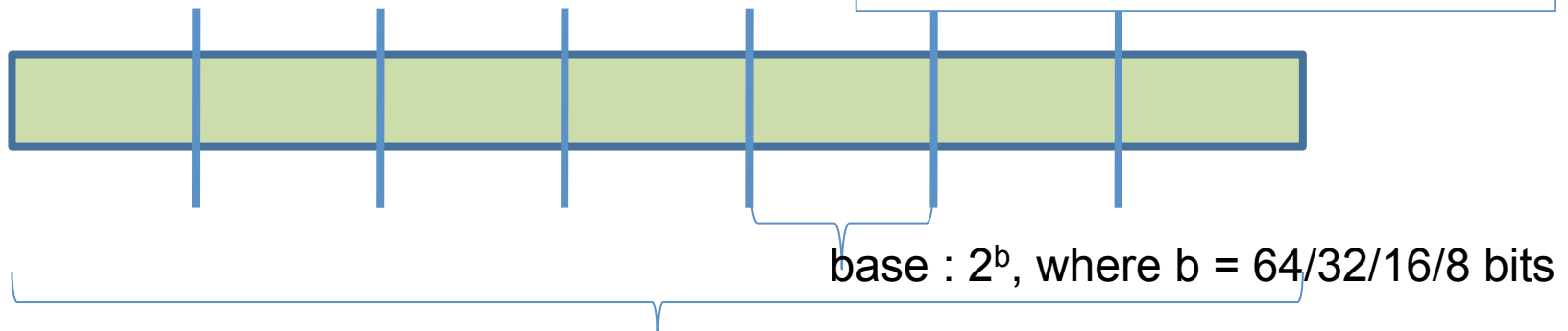
i	$e_i$	z
4	1	$1^{2*} x = x$
3	0	$x^2$
2	1	$x^4 * x = x^5$
1	1	$x^{10} * x = x^{11}$
0	1	$x^{22} * x = x^{23}$

# RSA Implementation in Software (Multi-precision Arithmetic)

- RSA requires arithmetic in 1024 or 2048 bit numbers
- Modern processors have ALUs that are 8, 16, 32, 64 bit
  - Typically can perform arithmetic on 8/16/32/64 bit numbers
- solution: multi-precision arithmetic (gmp library)

```
#define NBITS      xxx
#define WORDSIZE   xxx
#define MAXDIGITS (NBITS/WORDSIZE)

typedef unsigned long word;
typedef struct{
    word digits[MAXDIGITS];
    int  sign;
}bignum_t;
```



# Multi-precision Addition

- ADD :  $a = 9876543210 = (2, 76, 176, 22, 234)_{256}$   
 $b = 1357902468 = (80, 239, 242, 132)_{256}$   
 base = 8 bit (256)

i	$a_i$	$b_i$	$c_{in}$	$a_i + b_i + c_{in} \pmod{256}$	Carry?	$c_{out}$
0	234	132	0	110	$(110 < 234)?$	1
1	22	242	1	9	$(9 < 22)?$	1
2	176	239	1	160	$(160 \leq 176)?$	1
3	76	80	1	157	$(157 \leq 76)?$	0
4	2	0	0	2	$(2 \leq 2)?$	0

$$a + b = (2, 157, 160, 9, 110)_{256}$$

$$= 11234445678$$

# Multi-Precision Addition Algorithm

---

**Algorithm 2: Add :** *Multi-Precision Addition.* The function performs  $r = a + b$ . Each input is of size  $n$  words.

---

```
Input: word *a, word *b, int n
Output: word *r
1 begin
2   carry  $\leftarrow$  0
3   for  $i \in (0, 1, 2, \dots, n - 1)$  do
4      $t \leftarrow a[i]$ 
5      $t \leftarrow t + \text{carry}$ 
6      $\text{carry} \leftarrow (t < \text{carry})$ 
7      $l \leftarrow t + b[i]$ 
8      $\text{carry} \leftarrow \text{carry} + (l < t)$ 
9      $r[i] \leftarrow l$ 
10  end
11  return r
12 end
```

---

- The asymptotic complexity of multi-precision addition is  $\mathcal{O}(\text{MAXDIGITS})$ .
- The algorithm requires MAXDIGITS single precision additions to be performed, where each addition is of WORDSIZE.
- This also requires  $2 \times \text{MAXDIGITS}$  comparisons as carry is compared with both the operands in each iteration of the loop.

# Multi-precision Subtraction

- SUB :  $a = 9876543210 = (2, 76, 176, 22, 234)_{256}$   
 $b = 1357902468 = (80, 239, 242, 132)_{256}$

base = 256 (8 bit)

i	$a_i$	$b_i$	$C_{in}$	Borrow?	$C_{out}$	$a_i - b_i - c_{in} \pmod{256}$
0	234	132	0	$(234 < 132)?$	0	102
1	22	242	0	$(22 < 242)?$	1	$-220 = 36$
2	176	239	1	$(176 < 239)?$	1	$-64 = 192$
3	76	80	1	$(76 < 80)?$	1	$-5 = 251$
4	2	0	1	$(2 < 0)?$	0	1

$$a - b = (1, 251, 192, 36, 102)_{256}$$

$$= 8658640742$$

# Multi-Precision Subtraction Algorithm

---

**Algorithm 3: Sub** : *Multi-Precision Subtraction*. The function performs  $r = a - b$ . Each input is of size  $n$  words.

---

```
Input: word *a, word *b, int n
Output: word *r
1 begin
2   borrow ← 0
3   for  $i \in (0, 1, 2, \dots, n - 1)$  do
4      $r[i] \leftarrow (a[i] - b[i] - \text{borrow})$ 
5     if  $(a[i] \neq b[i])$  then
6       |  $\text{borrow} = (a[i] < b[i])$ 
7     end
8   end
9   return r
10 end
```

---

## Analysis of Multi-Precision Subtraction

- The asymptotic complexity of multi-precision subtraction is  $\mathcal{O}(\text{MAXDIGITS})$ .
- The algorithm requires MAXDIGITS subtractions to be performed. Each subtraction is of WORDSIZE.
- This also requires MAXDIGITS comparisons as operands are compared to know the borrow in each iteration of the loop.

# Multi-Precision Multiplication

$$C = A \times B \bmod N$$

(without Modular operation)

- Classical (School book) algorithm
- Karatusba algorithm
- Toom-3 algorithm
- FFT

# Multi-precision Multiplication (Classical Multiplication)

- MUL :  $a = 1234567 = (18, 214, 135)_{256}$   
 $b = 76543210 = (4, 143, 244, 234)_{256}$

base = 8 bit (256)

$a * b =$

$(0\ 85\ 241\ 247\ 25\ 195\ 102)_{256}$

$= 99447721140070$

$i$	$a_i$	$j$	$b_j$	$a_i b_j = (h, l)_B$	Operation	$c$
					Initialization	$(0, 0, 0, 0, 0, 0, 0)_B$
0	135	0	234	$(123, 102)_B$	Add 102 at pos 0	$(0, 0, 0, 0, 0, 0, 102)_B$
					Add 123 at pos 1	$(0, 0, 0, 0, 0, 123, 102)_B$
					Add 172 at pos 1	$(0, 0, 0, 0, 1, 39, 102)_B$
					Add 128 at pos 2	$(0, 0, 0, 0, 129, 39, 102)_B$
					Add 105 at pos 2	$(0, 0, 0, 0, 234, 39, 102)_B$
1	214	0	234	$(195, 156)_B$	Add 75 at pos 3	$(0, 0, 0, 75, 234, 39, 102)_B$
					Add 28 at pos 3	$(0, 0, 0, 103, 234, 39, 102)_B$
					Add 2 at pos 4	$(0, 0, 2, 103, 234, 39, 102)_B$
					Add 156 at pos 1	$(0, 0, 2, 103, 234, 195, 102)_B$
					Add 195 at pos 2	$(0, 0, 2, 104, 173, 195, 102)_B$
1	214	1	244	$(203, 248)_B$	Add 248 at pos 2	$(0, 0, 2, 105, 165, 195, 102)_B$
					Add 203 at pos 3	$(0, 0, 3, 52, 165, 195, 102)_B$
					Add 138 at pos 3	$(0, 0, 3, 190, 165, 195, 102)_B$
					Add 119 at pos 4	$(0, 0, 122, 190, 165, 195, 102)_B$
					Add 88 at pos 4	$(0, 0, 210, 190, 165, 195, 102)_B$
2	18	0	234	$(16, 116)_B$	Add 3 at pos 5	$(0, 3, 210, 190, 165, 195, 102)_B$
					Add 116 at pos 2	$(0, 3, 210, 191, 25, 195, 102)_B$
					Add 16 at pos 3	$(0, 3, 210, 207, 25, 195, 102)_B$
					Add 40 at pos 3	$(0, 3, 210, 247, 25, 195, 102)_B$
					Add 17 at pos 4	$(0, 3, 227, 247, 25, 195, 102)_B$
2	18	1	244	$(17, 40)_B$	Add 14 at pos 4	$(0, 3, 241, 247, 25, 195, 102)_B$
					Add 10 at pos 5	$(0, 13, 241, 247, 25, 195, 102)_B$
					Add 72 at pos 5	$(0, 85, 241, 247, 25, 195, 102)_B$
					Add 0 at pos 6	$(0, 85, 241, 247, 25, 195, 102)_B$



# Multi-precision Multiplication (Karatsuba Multiplication)

Let  $a, b$  be two multiprecision integers with  $n$  B – ary words.

Let  $m = n / 2$

$$a = a_h B^m + a_l$$

$$b = b_h B^m + b_l$$

$$\begin{aligned} a \times b &= (a_h b_h) B^{2m} + (a_h b_l + a_l b_h) B^m + a_l b_l \\ &= (a_h b_h) B^{2m} + (a_h b_h + a_l b_l - (a_h - a_l)(b_h - b_l)) B^m + a_l b_l \end{aligned}$$

$$\text{using } (a_h - a_l)(b_h - b_l) = a_h b_h - a_h b_l - a_l b_h + a_l b_l$$

Karatsuba multiplication converts  $n$  bit multiplications into 3 multiplications of  $n/2$  bits  
The penalty is an increased number of additions

# Multi-precision Multiplication (Karatsuba Multiplication)

$B = 256$ ;  
 $a = 123456789 = (7, 91, 205, 21)_{256}$   
 $b = 987654321 = (58, 222, 104, 177)_{256}$

$n=4; m=2$   
 $a_h = (7, 91); a_l = (205, 21)$   
 $a = (7, 91)256^2 + (205, 21)$   
  
 $b_h = (58, 222); b_l = (104, 177)$   
 $b = (58, 222)256^2 + (104, 177)$

$a_h b_h = (1, 176, 254, 234)_{256}$   
 $a_l b_l = (83, 222, 83, 133)_{256}$   
 $a_h - b_h = -(197, 186)_{256}$   
 $a_l - b_l = -(45, 211)_{256}$   
 $(a_h - b_h)(a_l - b_l) = (35, 100, 170, 78)_{256}$   
 $a_h b_l + a_l b_h$   
 $= a_h b_h + a_l b_l - (a_h - b_h)(a_l - b_l)$   
 $= (50, 42, 168, 33)_{256}$

1	176	254	234				
		50	42	168	33		
				83	222	83	133
1	177	49	20	251	255	83	133

# Performing Modular Reduction

- Divide and get remainder  
(repeated subtraction)

Alternatively, we could use Montgomery multiplication that will not require modular reduction.

# Montgomery Multiplication

$$c = a \times b \bmod m$$

Select  $R = 2^x$ ,  $\gcd(R, m) = 1$ ,  $R$  slightly greater than  $m$

Use Extended Euclidean Algorithm to find  $R^{-1}$  and  $m'$   
 $s.t \quad R \cdot R^{-1} - m \cdot m' = 1$

Convert multiplicands to Montgomery domain

$$\bar{a} = aR \bmod m$$

$$\bar{b} = bR \bmod m$$

$$\text{Note that } c = \bar{a} \cdot \bar{b} \cdot R^{-2} \bmod m$$

*The Montgomery multiplier computes*

$$\bar{c} = \bar{a} \cdot \bar{b} \cdot R^{-1} \bmod m$$

← No specific benefits this way

# Montgomery's Trick

*Montgomery's trick*

1)  $t = \bar{a} \cdot \bar{b}$

2)  $u = (t + ((t \bmod R) \cdot m' \bmod R) \cdot m) / R$

3) if  $(u \geq m)$  return  $u - m$ ; else return  $u$ .

# Montgomery's Trick (why it works)

*Montgomery's trick*

1)  $t = \bar{a} \cdot \bar{b}$

2)  $u = (t + ((t \bmod R) \cdot m' \bmod R) \cdot m) / R$

3) if  $(u \geq m)$  return  $u - m$ ; else return  $u$ .

- First note that  $R \mid t$
- Then  $R \mid (t \cdot m' \cdot m \bmod R)$   
*....this follows because  $RR^{-1} - m'm = 1$ ; then take mod  $R$*
- Therefore  $R \mid (t + t \cdot m' \cdot m \bmod R)$   
*....the division in step 2 is valid*
- $u \cdot R = t + t \cdot m' \cdot m \bmod R$   
$$= t + t \cdot m' \cdot m$$
$$= t + k \cdot m$$
$$= t \bmod m$$

# Montgomery Multiplier in the Montgomery Ladder

**Input:**  $c, y$

**Output:**  $y^c \bmod N$

**exp(c, y) {**

**R0** = 1 \*  $R \bmod N$

**R1** =  $y * R \bmod N$

for  $i=n-1$  to 0 do

if  $c_i = 0$  then

**R1** = **R0** \* **R1**

**R0** = **R0** \* **R0**

else

**R0** = **R0** \* **R1**

**R1** = **R1** \* **R1**

return (**R0** \*  $R^{-1}$ )

}

Convert to Montgomery domain.

Multiplications in Montgomery domain.  
Note. Each result is also in Montgomery domain.

Return to Original domain

# Speeding RSA decryption with CRT

- Decryption is done as follows :

$$x = y^a \bmod n$$

- Bob can also decrypt by using CRT

$$x = y^a \bmod p$$

$$x = y^a \bmod q$$

(since he knows the factors of  $n$ , i.e.  $p, q$ )

- CRT turns out to be much faster since the size (in bits) of  $p$  and  $q$  is about  $\frac{1}{2}$  that of  $n$



# Multi-precision libraries

- GMP : GNU Multi-precision library
- Make use of Intel's SSE/AVX instructions
  - These are SIMD instructions that have large registers (128, 256, 512 bit)
- Crypto libraries
  - OpenSSL, PolarSSL, NaCL, etc.

# RSA Speeds

Table 1: Evaluation of RSA on Intel 64-bit System.

<b>Input Size</b>	<b>Without CRT (Seconds)</b>	<b>With CRT (Seconds)</b>
128	0.000074	0.000022
256	0.000523	0.000299
512	0.001707	0.001155
1024	0.012381	0.010940
2048	0.091174	0.077656

Table 2: Evaluation of RSA on Intel 32-bit System.

<b>Input Size</b>	<b>Without CRT (Seconds)</b>	<b>With CRT (Seconds)</b>
128	0.000730	0.000229
256	0.004576	0.002664
512	0.034216	0.026493
1024	0.278812	0.213975
2048	2.280441	1.908730

# RSA Speeds

32 Bit ARM Cortex

Table 3: Evaluation of RSA on LPCXpresso 1347.

Input Size	Without CRT (Seconds)	With CRT (Seconds)
128	5.799000	2.344000
256	37.806000	24.069000
512	326.877000	231.231000

16 Bit TI Micro-controller

Table 4: Execution Time on Varying Input Size on MSP-430

Input Size	Without CRT (Seconds)	With CRT (Seconds)
128	5.06	5.029
256	36.025	33.044
512	260.007	254.13
1024	2011	2028

# Finding Primes

# Test for Primes

- How to generate large primes?
  - Select a random large number
  - Test whether or not the number is prime
- What is the probability that the chosen number is a prime?
  - Let  $\pi(N)$  be the number of primes  $< N$
  - From number theory,  $\pi(N) \approx N/\ln N$
  - Therefore probability of a random number ( $< N$ ) being a prime is  $1/\ln N$ 
    - As  $N$  increases, it becomes increasingly difficult to find large primes

# GIMPS

- There are infinite prime numbers (proved by Euclid)
- Finding them becomes increasingly difficult as N increases
- GIMPS : Great Internet Mersenne Prime Search
  - Mersenne Prime has the form  $2^n - 1$
  - Largest known prime (found in Dec 2017) has 23 million digits  $2^{77,232,917} - 1$ 
    - \$3000 to beat this 😊

# Primality Tests with Trial Division

- School book methods (trial division)
  - Find if  $N$  divides any number from 2 to  $N-1$
  - find if  $N$  divides any number from 2 to  $N^{1/2}$
  - Find if  $N$  divides any prime number from 2 to  $N^{1/2}$
  - Too slow!!!
    - Need to divide by  $N-1$  numbers
    - Need to divide by  $N^{1/2}$  numbers
    - Need to divide by  $(N/\ln N)^{1/2}$  primes
      - For example, if  $n$  is approx  $2^{1024}$ , then need to check around  $2^{507}$  numbers
    - Need something better for large primes
      - Randomized algorithms

# Randomized Algorithms for Primality Testing

- Monte-carlo Randomized Algorithms
  - Always runs in polynomial time
  - May produce incorrect results with bounded probability
  - Yes-based Monte-carlo method
    - Answer YES is always correct, but answer NO may be wrong
  - No-based Monte-carlo method
    - Answer NO is always correct, but answer YES may be wrong



# Finding Large Primes (using Fermat's Theorem)

```
is_composite(n){  
  pick  $a \leftarrow Z_n$   
  if ( $a^{n-1} \equiv 1 \pmod n$ )  
    return FALSE  
  else  
    return TRUE  
}
```

If  $n$  is prime, then  $a^{n-1} \equiv 1 \pmod n$  is **true** for any 'a'. Therefore the algorithm would always return FALSE.

If  $n$  is composite  $a^{n-1} \equiv 1 \pmod n$  is **false** but **may be true** for some choices of  $a$ . In this case, the algorithm may return TRUE sometime and FALSE other times.

For example:  $n = 221$  ( $13 \cdot 17$ )  
and  $a = 38$  then  
 $38^{220} \pmod{221} \equiv 1$ .  
(FALSE returned)

We need to increase our confidence with more values of  $a$

# Fermat's Primality Test

- Increasing confidence with multiple bases

```
primality_test(n){  
    c = 0  
    for(i = 0; i < 1000; ++i){  
        if(is_composite(n) == TRUE)  
            return COMPOSITE  
    }  
    return probably PRIME  
}
```

# Carmichael Number

Some composites act as primes.

Irrespective of the 'a' chosen, the test  $a^{n-1} \equiv 1 \pmod{n}$  passes.

for example Carmichael numbers are composite numbers which satisfy Fermat's little theorem irrespective of the value of a.

Eg.  $561 = 3 \times 11 \times 17$

# Strong probable-primality test

- If  $n$  is prime, the square root of  $a^{n-1}$  is either  $+1$  or  $-1$

$$\text{let } a^{\frac{n-1}{2}} = b$$

$$b^2 \equiv 1 \pmod{n}$$

$$b^2 - 1 \equiv 0 \pmod{n}$$

$$(b+1)(b-1) \equiv 0 \pmod{n}$$

$$\text{either } (b+1) \equiv 0 \pmod{n} \text{ or } (b-1) \equiv 0 \pmod{n}$$

# Miller-Rabin Primality Test

- Yes-base primality test for composites
- Does not suffer due to Carmichael numbers
- Write  $n-1 = 2^s d$ 
  - where  $d$  is odd and  $s$  is non-negative
  - $n$  is a composite if

$$a^d \not\equiv 1 \pmod{n} \text{ and } (a^d)^{2^r} \not\equiv -1 \pmod{n}$$

*for all numbers  $r$  less than  $s$*

# Proof of Miller-Rabin test

- Write  $n-1 = 2^s d$

$$a^d \not\equiv 1 \pmod{n} \quad \text{and} \quad (a^d)^{2^r} \not\equiv -1 \pmod{n}$$

*for all number  $r$  less than  $s$*

- Proof: We prove the contra-positive. We will assume  $n$  to be prime. Thus,

$$a^d \equiv 1 \pmod{n} \quad \text{or} \quad (a^d)^{2^r} \equiv -1 \pmod{n}$$

*for some number  $r$  less than  $s$*

# Proof of Miller-Rabin test

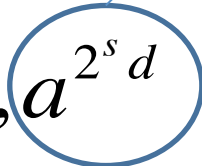
Proof: We prove the contra-positive. We will assume  $n$  to be prime. Thus we prove,

$$a^d \equiv 1 \pmod{n} \quad \text{or} \quad (a^d)^{2^r} \equiv -1 \pmod{n}$$

*for some number  $r$  less than  $s$*

<sup>1</sup> (Fermat's and we assume  $n$  is prime)

- Consider the sequence :

$$a^d, a^{2^1 d}, a^{2^2 d}, a^{2^3 d}, \dots, a^{2^{s-1} d}, a^{2^s d}$$


- The roots of  $x^2 = 1 \pmod{n}$  is either  $+1$  or  $-1$
- In the sequence, if  $a^d$  is  $1$ , then all elements in the sequence will be  $1$
- If  $a^d$  is not  $1$ , then there should be some element in the sequence which is  $-1$ , in order to have the final element as  $1$

# Miller-Rabin Algorithm (test for composites)

Input  $n$

*T1. Find an odd integer  $d$  such that  $n - 1 = 2^s d$*

*T2. Select at random a nonzero  $a \in \mathbb{Z}_n$*

*T3. Compute  $b = a^d \bmod n$*

*If  $b = \pm 1$ , return ' $n$  is prime'*

*T4. For  $i = 1, \dots, r - 1$ , calculate  $c \equiv b^{2^i} \bmod n$*

*If  $c = -1$ , return ' $n$  is prime'*

*T5. Otherwise return ' $n$  is composite'*

- $\Pr(\text{input=composite} \mid \text{ans=composite}) = 1$
- $\Pr(\text{ans=prime} \mid \text{input=composite}) < 1/2$
- $\Pr(\text{input=composite} \mid \text{ans=prime}) \leq 1/4$



# Quadratic Residues

**Definition.** Let  $a, m \in \mathbb{N}$ . Then  $a$  is a **quadratic residue of  $m$**  iff  $(a, m) = 1$  and there is an  $x \in \mathbb{Z}$  so that  $x^2 \equiv a \pmod{m}$ .

- Example :  $m=13$ , square elements in  $\mathbb{Z}_{13}$ .

a cannot be 0

1, 4, 9, 3, 12, 10, 10, 12, 3, 9, 4, 1

The quadratic residues  $\mathbb{Z}_{13}$  are therefore

$\{1, 4, 3, 9, 10, 12\}$

If an element is not a quadratic residue, then it is a **quadratic non-residue**

**quadratic non-residues in  $\mathbb{Z}_{13}$  are  $\{2, 5, 6, 7, 8, 11\}$**

# Legendre Symbol

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } p \mid a \\ 1 & \text{if } a \text{ is a } QR \bmod p \\ -1 & \text{if } a \text{ is a } QNR \bmod p \end{cases}$$

Given  $p$  is an odd prime

# Euler's Criteria

A result from Euler

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod{p}$$

when  $p \mid a$

$$a^{\frac{p-1}{2}} \equiv 0 \pmod{p}$$

when  $a$  is a QR,  $\exists x \in \mathbb{Z}_p$  s.t.  $a \equiv x^2 \pmod{p}$

$$\begin{aligned} \Rightarrow a^{\frac{p-1}{2}} &\equiv x^{2 \cdot \frac{(p-1)}{2}} \pmod{p} \\ &\equiv x^{p-1} \pmod{p} \\ &\equiv 1 \end{aligned}$$

# when Quadratic Non Residue

when  $a$  is a QNR, no such  $x \in \mathbb{Z}_p$  exists s.t.  $a \equiv x^2 \pmod{p}$

consider :  $a^{\frac{p-1}{2}} \pmod{p}$  (note  $p-1$  is even, if  $p$  is an odd prime)

squaring :  $a^{p-1} \pmod{p} \equiv 1$

$$\text{so, } \left( a^{\frac{p-1}{2}} \right)^2 \equiv 1 \pmod{p}$$

Thus,  $a^{\frac{p-1}{2}} \equiv \pm 1 \pmod{p}$

$a^{\frac{p-1}{2}} \not\equiv 1 \pmod{p}$ , since  $a$  is not a QR

Thus  $a^{\frac{p-1}{2}} \equiv -1 \pmod{p}$

# Examples

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod{p}$$

4 is a *QR* mod 13

$$4^{\frac{13-1}{2}} \pmod{13} \equiv 4^6 \pmod{13} \equiv 1$$

5 is a *QNR* mod 13

$$5^6 \pmod{13} \equiv 12 \pmod{13} \equiv -1$$

Congruence always holds when  
n is an odd prime

Euler's Witness

$$7^{\frac{15-1}{2}} \pmod{15} \equiv 7^7 \pmod{15} \equiv -2$$

Euler's Liar

$$14^{\frac{15-1}{2}} \pmod{15} \equiv 14^7 \pmod{15} \equiv -1$$

Congruence **may**  
**or may not** hold  
when  
n is not prime

# Solovay Strassen Primality Test

```
SOLOVAYSTRASSEN(n) {  
    choose a random integer a such that  $1 \leq a \leq n-1$   
    compute  $x = \left(\frac{a}{n}\right)$   
    if ( $x = 0$ ) return COMPOSITE  
    compute  $y = a^{\frac{n-1}{2}} \bmod n$   
    if ( $x \equiv y \bmod n$ ) return possibly PRIME  
    else return COMPOSITE  
}
```

How to compute


Legendre's symbol

error probability is at most  $\frac{1}{2}$

# Jacobi Symbol

- Jacobi Symbol is a generalization of the Legendre symbol
- Let  $n$  be any positive odd integer and  $a \geq 0$  any integer. The Jacobi symbol is defined as:

Suppose  $n$  is an odd positive integer with prime factorization

  $n = p_1^{e_1} \times p_2^{e_2} \times p_3^{e_3} \times p_4^{e_4} \dots$

Then,

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{e_1} \times \left(\frac{a}{p_2}\right)^{e_2} \times \left(\frac{a}{p_3}\right)^{e_3} \times \left(\frac{a}{p_4}\right)^{e_4} \times \dots$$

# Jacobi Properties

P1. If  $a \equiv b \pmod{n}$  then  $\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$

P2.  $\left(\frac{2}{n}\right) = \begin{cases} 1 & \text{if } n \equiv \pm 1 \pmod{8} \\ -1 & \text{if } n \equiv \pm 3 \pmod{8} \end{cases}$

P3.  $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$

P4. if  $a$  is even,  $a = 2^k t$ ,  $\left(\frac{a}{n}\right) = \left(\frac{2}{n}\right)^k \left(\frac{t}{n}\right)$

P5. if  $a$  is odd,

$$\left(\frac{a}{n}\right) = \begin{cases} -\left(\frac{n}{a}\right) & \text{if } n \equiv a \equiv 3 \pmod{4} \\ \left(\frac{n}{a}\right) & \text{otherwise} \end{cases}$$



# Computing Jacobi

$$\left(\frac{1001}{9907}\right) = \left(\frac{7}{9907}\right) \left(\frac{11}{9907}\right) \left(\frac{13}{9907}\right).$$

$$\left(\frac{7}{9907}\right) = -\left(\frac{9907}{7}\right) = -\left(\frac{2}{7}\right) = -1.$$

$$\left(\frac{11}{9907}\right) = -\left(\frac{9907}{11}\right) = -\left(\frac{7}{11}\right) = \left(\frac{11}{7}\right) = \left(\frac{4}{7}\right) = 1.$$

$$\left(\frac{13}{9907}\right) = \left(\frac{9907}{13}\right) = \left(\frac{1}{13}\right) = 1.$$

$$\left(\frac{1001}{9907}\right) = -1.$$

From the theorem

P5, P1, then P2

P5, P1, P5, P1, P3, P2

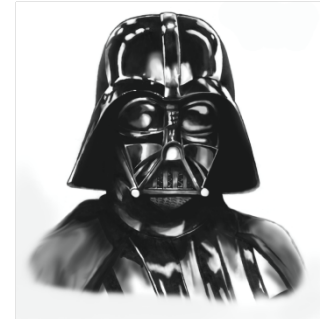
P5, P1

and 1 is a QR mod 13

# Factoring Algorithms

# Factorization to get the private key

- Public information  $(n, b)$
- If Mallory can factorize  $n$  into  $p$  and  $q$  then,
  - She can compute  $\phi(n) = (p-1)(q-1)$
  - She can then compute the private key by finding  $a \equiv b^{-1} \pmod{\phi(n)}$



How to factorize  $n$ ?

# Trial Division

Fundamental theorem of arithmetic

Any integer number (greater than 1) is either prime or a product of prime powers

$$n = p_1^{e_1} p_2^{e_2} p_3^{e_3} \cdots p_k^{e_k}$$

```
def trial_division(n):  
    """Return a list of the prime factors for a natural number."""  
    if n < 2:  
        return []  
    prime_factors = []  
    for p in prime_sieve(int(n**0.5) + 1):  
        if p*p > n: break  
        while n % p == 0:  
            prime_factors.append(p)  
            n //= p  
    if n > 1:  
        prime_factors.append(n)  
    return prime_factors
```

prime generation algorithm

Prime factors of  $n$  cannot be greater than  $\sqrt{n}$

$n = n / p$  : remove this factor from  $n$

Running Time of algorithm order of  $\pi(n^{1/2})$

# Pollard p-1 Factorization

$$n = p \times q$$

- 1 choose a random integer  $a$  ( $1 < a < n$ ).  
If  $\gcd(a, n) \neq 1$ , then  $a$  is a prime factor.  
However, this is most likely not the case.

- 2 Suppose we select some  $L$  and compute  $d = \gcd(a^L - 1, n)$   
if  $1 < d < n$  then we have factored  $n$   
 $d \mid n$  and  $d \mid (a^L - 1)$   
 $d$  has to be the prime  $p$  or the prime  $q$

why  $a^L - 1$ ?

since  $d$  is prime and  $d \mid (a^L - 1)$

$$a^L \equiv 1 \pmod{d}$$

$$\varphi(d) \mid L \Rightarrow (d-1)k = L$$

Thus we need to find  $L$  which is some factor of  $(d-1)$ .

- 3 If  $\gcd(a^L - 1, n) = n$   
This is possible only when  $p \mid n$  and  $p \mid a^L - 1$  (or  $q \mid n$  and  $q \mid a^L - 1$ )  
and  $a^L - 1 > n$

How to choose  $L$ ?

No easy way, trial and error!!

Factorials have a lot of divisors. So that is a nice way.

So, take  $L$  as a factorial of some number  $r$ .

# Pollard p-1 Factorization

Pollard p-1 factorization for  $n$ .

*S1.  $a \leftarrow 2$*

*S2. if  $\gcd(a, n) > 1$ , then this gcd is a prime factor of  $n$ , we are done.*

*S3. compute  $d \leftarrow \gcd(a^{r!} - 1, n)$*

*if  $d = n$ , start again from S1 with next value of  $a$*

*else if  $d = 1$ , increment  $r$  and repeat S3*

*else  $d$  is the prime factor of  $n$ ; we are done!*

$r = 2, 3, 4, \dots$

1. Will the algorithm terminate?
2. When will we choose the next value of  $a$ ? (will we get an infinite loop?)

When  $r = d-1$  then  $L = r! = (d-1)! = d-1(d-2)! = (d-1)k$

$(d-1) \mid L \rightarrow$  we will get the  $\gcd(a^{k(d-1)}, n) = n$  or its prime factor.

# Pollard Rho Algorithm

- Form a sequence S1 by selecting randomly (all different) from the set  $Z_n$

$$S1 = x_0, x_1, x_2, x_3, x_4, \dots$$

- Also assume we **magically** find a new sequence S2 comprising of

$$S2 = \bar{x}_0, \bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4, \dots \text{ where}$$

$$\begin{aligned} \bar{x}_0 &\equiv x_0 \pmod{p} \\ \bar{x}_1 &\equiv x_1 \pmod{p} \\ \bar{x}_2 &\equiv x_2 \pmod{p} \\ \bar{x}_3 &\equiv x_3 \pmod{p} \\ \bar{x}_4 &\equiv x_4 \pmod{p} \end{aligned}$$

- If we keep adding elements to S1, we will eventually find an  $x_i$  and  $x_j$  ( $i \neq j$ ) such that  $\bar{x}_i = \bar{x}_j$   
When this happens,

$$p \mid (x_i - x_j)$$

$\therefore p \mid n$  also,  $\gcd((x_i - x_j), n)$  is  $p$ . We found a factor of  $n$ !!

# Doing without magic

- Form a sequence  $S1$  by selecting randomly (with replacement) from the set  $Z_n$

$$S1 = x_0, x_1, x_2, x_3, x_4, \dots$$

- For every pair  $i, j$  in the sequence compute

$$d \leftarrow \gcd((x_i - x_j), n)$$

- If  $d > 1$  then it is a factor of  $n$



# Selecting elements of S1

To choose the next element of S1, Pollard suggests using a function  $f : Z_n \rightarrow Z_n$  with requirement that the output looks random.

$$\text{Example : } f(x) = x^2 + 1 \bmod n$$

$$S1 = \left( \begin{array}{l} \left\{ \begin{array}{l} x_0 \\ x_i \end{array} \right. \quad \begin{array}{l} \text{where } x_0 \text{ is chosen randomly from } Z_n \\ i > 0 \text{ and } x_i = f(x_{i-1}) \end{array} \end{array} \right)$$

# Example

- $N = 82123$ ,  $x_0 = 631$ ,  $f(x) = x^2 + 1$

This column is just for understanding.  
In reality we will not know this

$i$	$x_i \bmod N$	$\overline{x_i} = x_i \bmod p$	$i$	$x_i \bmod N$	$\overline{x_i} = x_i \bmod p$
0	631	16	10	6685	2
1	69670	11	11	14314	5
2	28986	40	12	75835	26
3	69907	2	13	37782	21
4	13166	5	14	17539	32
5	64027	26	15	65887	0
6	40816	21	16	74990	1
7	80802	32	17	45553	2
8	20459	0	18	73969	5
9	71874	1	19	50210	26

Drawback...  
Large number of GCD  
Computations. 55 gcd  
computations in this case

Can we reduce the number  
of gcd computations?

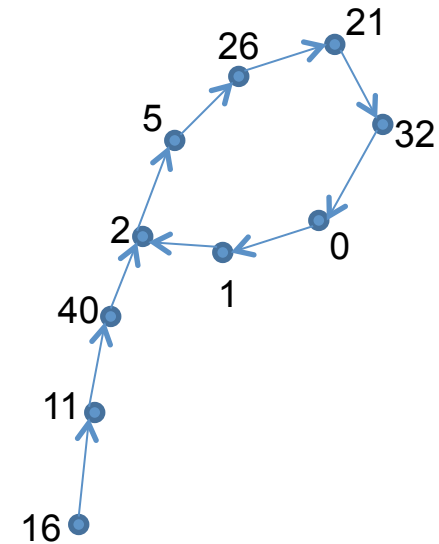
Given  $x_i \bmod N$ , we compute gcds of every pair until we find a gcd greater than 1

$$\gcd(x_3 - x_{10}, N) = \gcd(63222, 82123) = 41 \leftarrow \text{A factor of } N$$

# The Rho in Pollard-Rho

- $N = 82123$ ,  $x_0 = 631$ ,  $f(x) = x^2 + 1$

$i$	$x_i \bmod N$	$\overline{x_i} = x_i \bmod p$	$i$	$x_i \bmod N$	$\overline{x_i} = x_i \bmod p$
0	631	16	10	6685	2
1	69670	11	11	14314	5
2	28986	40	12	75835	26
3	69907	2	13	37782	21
4	13166	5	14	17539	32
5	64027	26	15	65887	0
6	40816	21	16	74990	1
7	80802	32	17	45553	2
8	20459	0	18	73969	5
9	71874	1	19	50210	26



$$\overline{x_t} = \overline{x_{t+l}} \bmod p$$

- The smallest value of  $t$  and  $l$ , for which the above congruence holds is  $t=3, l=7$
  - For  $l=7$ , all values of  $t > 3$  satisfy the congruence
  - This leads to a cycle as shown in the figure  
(and a shape like the Greek letter rho)
- $$\overline{x_j} = \overline{x_{j+l}} \pmod{p} \quad t \geq 3$$

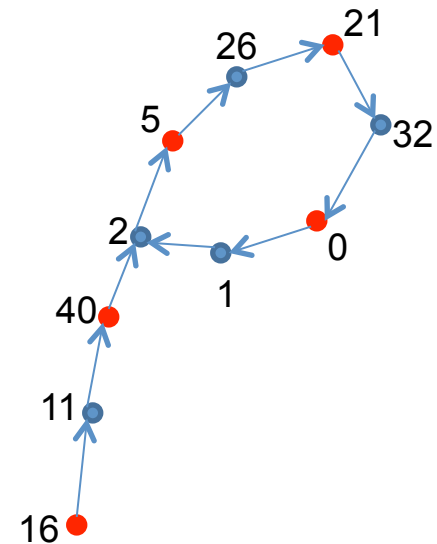
$$\overline{x_j} = \overline{x_{j+l}} \bmod p \quad t \geq 3$$

# Reducing gcd computations

- GCD computations can be expensive.
- Use Floyd's cycle detection algorithm to reduce the number of GCD computations.

*choose a random  $x_0 = y_0 \in Z_n$*

*loop* {  
     $x_i = f(x_{i-1})$   
     $y_i = x_{2i} = f(f(y_{i-1}))$   
    *If  $d = \gcd(x_i - y_i, N) > 0$ , return  $d$*



claim : The first time  $x_i = y_i \bmod p$  occurs when  $i \leq t + l$

This means that we get a collision before  $x$  completing an entire circle

# The first time $x_i = y_i \bmod p$ occurs is when $i \leq t + l$

- $l$  is the number of points in the cycle
- $t$  is the smallest value of  $i$  such that  $x_i \equiv y_i \bmod p$

$x_i$  and  $y_i$  meet at the same point in the cycle

Therefore,  $y_i$  must have traversed (some) cycles more

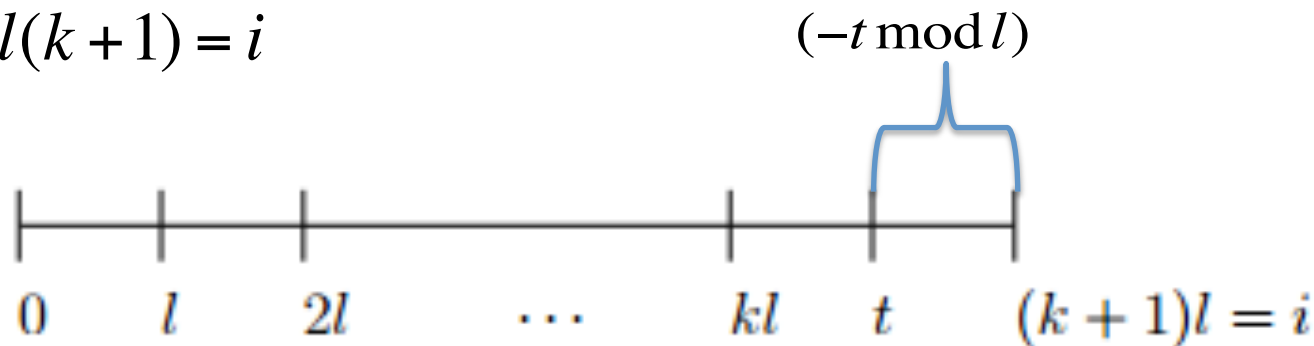
$$x_i \equiv y_i \bmod N$$

$$x_i \equiv x_{2i} \bmod N$$

$$l \mid (2i - i)$$

$$l \mid i \Rightarrow l(k + 1) = i$$

$$\text{consider } i = (k + 1)l = t + (-t \bmod l) \leq t + l$$



# Expected number of operations before a collision

- Can be obtained from Birthday paradox  
to be  $\sqrt{p}$

# Congruences of Squares

- Given  $N=p \times q$ , we need to find  $p$  and  $q$
- Suppose we find an  $x$  and  $y$  such that  $x^2 \equiv y^2 \pmod{N}$
- Then,

$$N \mid (x^2 - y^2) \Rightarrow N \mid (x - y)(x + y)$$

- This implies,

$$\gcd(N, (x - y)) \text{ and } \gcd(N, (x + y)) \text{ factors } N$$

# Example

- Consider  $N = 91$

$$10^2 \equiv 3^2 \pmod{91}$$

$$91 \mid (10 - 3)(10 + 3)$$

$$91 \mid (7 \times 13)$$

$$\gcd(91, 13) = 13$$

$$\gcd(91, 7) = 7$$

$$34^2 \equiv 8^2 \pmod{91}$$

$$91 \mid (34 + 8)(34 - 8)$$

$$91 \mid 42 \times 26$$

$$\gcd(91, 26) = 13$$

$$\gcd(91, 42) = 7$$

So... we can use  $x$  and  $y$  to factorize  $N$ .

$$x^2 \equiv y^2 \pmod{N}$$

But how do we find such pairs?



# Another Example

- $N = 1649$

$$41^2 \equiv 32 \pmod{1649}$$

$$43^2 \equiv 200 \pmod{1649}$$

32 and 200 are not perfect squares.  
However  $(32 \times 200 = 6400) = 80^2$   
is a perfect square

$$\begin{aligned}(41 \times 43)^2 &\equiv (32 \times 200) \pmod{1649} \\ &\equiv 80^2 \pmod{1649}\end{aligned}$$

Thus, it is possible to combine non-squares to form a perfect square

# Forming Perfect Squares

Recall, Fundamental theorem of arithmetic

Any integer number (greater than 1) is either prime or a product of prime powers

$$n = p_1^{e_1} p_2^{e_2} p_3^{e_3} \cdots p_k^{e_k}$$

Thus, a number is a perfect square if its prime factors have even powers.

$e_1, e_2, e_3, \dots$  is even

Thus,

$32 = 2^5 5^0$  not a perfect square

$200 = 2^3 5^2$  not a perfect square

$(32 \times 200) = 2^5 5^0 \times 2^3 5^2 = 2^8 5^2 = (2^4 5^1)^2$  is a perfect square

# Dixon's Random Squares Algorithm

1. Choose a set B comprising of 'b' smallest primes. Add -1 to this set.  
(A number is said to be b-smooth, if its factors are in this set)
2. Select an r at random
  - Compute  $y = r^2 \bmod N$
  - Test if y factors completely in the set B.
  - If NO, then discard. ELSE save (y, r) (these are called B-smooth numbers)
3. Repeat step 2, until we have b+1 such (y,r) pairs
4. Solve the system of linear congruencies

# Example

- $N = 1829$
- $b = 6$        $B = \{-1, 2, 3, 5, 7, 11, 13\}$
- Choose random values of  $r$ , square and factorize

✓  $42^2 = 1764 = -65 = -1 \cdot 5 \cdot 13 \pmod{1829}$

✓  $43^2 = 20 = 2^2 \cdot 5 \pmod{1829}$

✗  $60^2 = 1771 = -58 = -1 \cdot 2 \cdot 29 \pmod{1829}$

✓  $61^2 = 63 = 3^2 \cdot 7 \pmod{1829}$

✓  $74^2 = 1818 = -11 = -1 \cdot 11 \pmod{1829}$

✗  $75^2 = 138 = 2 \cdot 3 \cdot 23 \pmod{1829}$

✓  $85^2 = 1738 = -91 = -1 \cdot 7 \cdot 13 \pmod{1829}$

✓  $86^2 = 80 = 2^4 \cdot 5 \pmod{1829}$

All numbers are 6-smooth  
except 60 and 75.  
Leave these and  
consider all others

# Check Exponents

	-1	2	3	5	7	11	13	
-65	1	0	0	1	0	0	1	$42^2 = 1764 = -65 = -1 \cdot 5 \cdot 13 \pmod{1829}$
20	0	2	0	1	0	0	0	$43^2 = 20 = 2^2 \cdot 5 \pmod{1829}$
63	0	0	2	0	1	0	0	$60^2 = 1771 = -58 = -1 \cdot 2 \cdot 29 \pmod{1829}$
-11	1	0	0	0	0	1	0	$61^2 = 63 = 3^2 \cdot 7 \pmod{1829}$
-91	1	0	0	0	1	0	1	$74^2 = 1818 = -11 = -1 \cdot 11 \pmod{1829}$
80	0	4	0	1	0	0	0	$75^2 = 138 = 2 \cdot 3 \cdot 23 \pmod{1829}$
								$85^2 = 1738 = -91 = -1 \cdot 7 \cdot 13 \pmod{1829}$
								$86^2 = 80 = 2^4 \cdot 5 \pmod{1829}$

# Check Exponents

	-1	2	3	5	7	11	13	
-65	1	0	0	1	0	0	1	$42^2 = 1764 = -65 = -1 \cdot 5 \cdot 13 \pmod{1829}$
20	0	2	0	1	0	0	0	$43^2 = 20 = 2^2 \cdot 5 \pmod{1829}$
63	0	0	2	0	1	0	0	$60^2 = 1771 = -58 = -1 \cdot 2 \cdot 29 \pmod{1829}$
-11	1	0	0	0	0	1	0	$61^2 = 63 = 3^2 \cdot 7 \pmod{1829}$
-91	1	0	0	0	1	0	1	$74^2 = 1818 = -11 = -1 \cdot 11 \pmod{1829}$
80	0	4	0	1	0	0	0	$75^2 = 138 = 2 \cdot 3 \cdot 23 \pmod{1829}$
								$85^2 = 1738 = -91 = -1 \cdot 7 \cdot 13 \pmod{1829}$
								$86^2 = 80 = 2^4 \cdot 5 \pmod{1829}$

Find rows where exponents sum is even  
-65, 20, 63, -91

sum	2	2	2	2	2	0	2
-----	---	---	---	---	---	---	---

$$(42 \times 43 \times 61 \times 85)^2 \equiv (-1 \times 2 \times 3 \times 5 \times 7 \times 13)^2 \pmod{1829}$$

$$1459^2 \equiv 901^2 \pmod{1829}$$

# Final Steps

$$(42 \times 43 \times 61 \times 85)^2 \equiv (-1 \times 2 \times 3 \times 5 \times 7 \times 13)^2 \pmod{1829}$$

$$1459^2 \equiv 901^2 \pmod{1829}$$

$$1829 \mid (1459 + 901)(1459 - 901)$$

$$\Rightarrow 1829 \mid 2360 \quad \gcd(1829, 2360) = 59$$

$$\Rightarrow 1829 \mid 558 \quad \gcd(1829, 558) = 31$$

$$\text{Thus } 1829 = 59 \times 31$$

# State of the Art

## Factorization Techniques

- Quadratic Sieve
  - Fastest for less than 100 digits
- General Number field Sieve
  - Fastest technique known so far for greater than 100 digits
  - Open source code (google GGNFS)
- RSA factoring challenge
  - Best so far is 768 bit factorization
  - Current challenges 896 bits (reward \$75,000), 1024 bit (\$100,000)



# **RSA Attacks**

**attacks that don't require  
factorization algorithms**

# $\Phi(n)$ leaks

- If an attacker gets  $\Phi(n)$  then  $n$  can be factored

$$n = pq \qquad q = n / p$$

$$\begin{aligned}\phi(n) &= (p-1)(q-1) \\ &= pq - (p+q) + 1\end{aligned}$$

$$\phi(n) = n - \left(p + \frac{n}{p}\right) + 1$$

$$p^2 - (n - \phi(n) + 1)p + n = 0$$

Solve to get  $p$  (a factor of  $n$ )

# square roots of 1 mod n

There are two trivial and two non-trivial solutions for  $y^2 \equiv 1 \pmod{n}$

The trivial solutions are +1 and -1

By CRT, these congruences  
are equivalent

$$y^2 \equiv 1 \pmod{n} \iff \begin{cases} y^2 \equiv 1 \pmod{p} \\ y^2 \equiv 1 \pmod{q} \end{cases}$$

$$\begin{cases} y \equiv 1 \pmod{p} \\ y \equiv -1 \pmod{p} \end{cases}$$

$$\begin{cases} y \equiv 1 \pmod{q} \\ y \equiv -1 \pmod{q} \end{cases}$$

To get the non-trivial solutions solve using CRT

$$\begin{aligned} y &\equiv +1 \pmod{p} \\ y &\equiv -1 \pmod{q} \end{aligned}$$

$$\begin{aligned} y &\equiv -1 \pmod{p} \\ y &\equiv +1 \pmod{q} \end{aligned}$$

# Example

- $n=403 = 13 \times 31$
- To get the non-trivial solutions of  $y^2 \equiv 1 \pmod{n}$  solve using CRT

$$\begin{aligned} y &\equiv +1 \pmod{p} \\ y &\equiv -1 \pmod{q} \end{aligned}$$

$$\begin{aligned} y &\equiv -1 \pmod{p} \\ y &\equiv +1 \pmod{q} \end{aligned}$$

$$(31 \cdot 31^{-1} \pmod{13} - 13 \cdot 13^{-1} \pmod{31}) \pmod{403}$$

$$(31 \cdot 8 - 13 \cdot 12) \pmod{403} \equiv 92$$

$$403 - 92 = 311$$

The non-trivial solutions are 92 and 311

*Note:*  $92^2 \equiv 311^2 \equiv 1 \pmod{403}$

What happens when we solve  $y \equiv +1 \pmod{p}$   
 $y \equiv +1 \pmod{q}$

# Decryption exponent leaks

- If the decryption exponent 'a' leaks, then n can be factored
- The attacker can then compute  $ab$

$$ab \equiv 1 \pmod{\phi(n)} \qquad k\phi(n) = (ab - 1)$$

- Now, for any message  $x \neq 0$

$$x^{ab-1} \equiv 1 \pmod{n}$$

- **Attack Plan**, take square root :  $y \equiv x^{\frac{ab-1}{2}} \pmod{n}$

$$\text{i.e., } y^2 \equiv 1 \pmod{n} \Rightarrow n \mid (y^2 - 1)$$

$$\Rightarrow n \mid (y - 1)(y + 1)$$

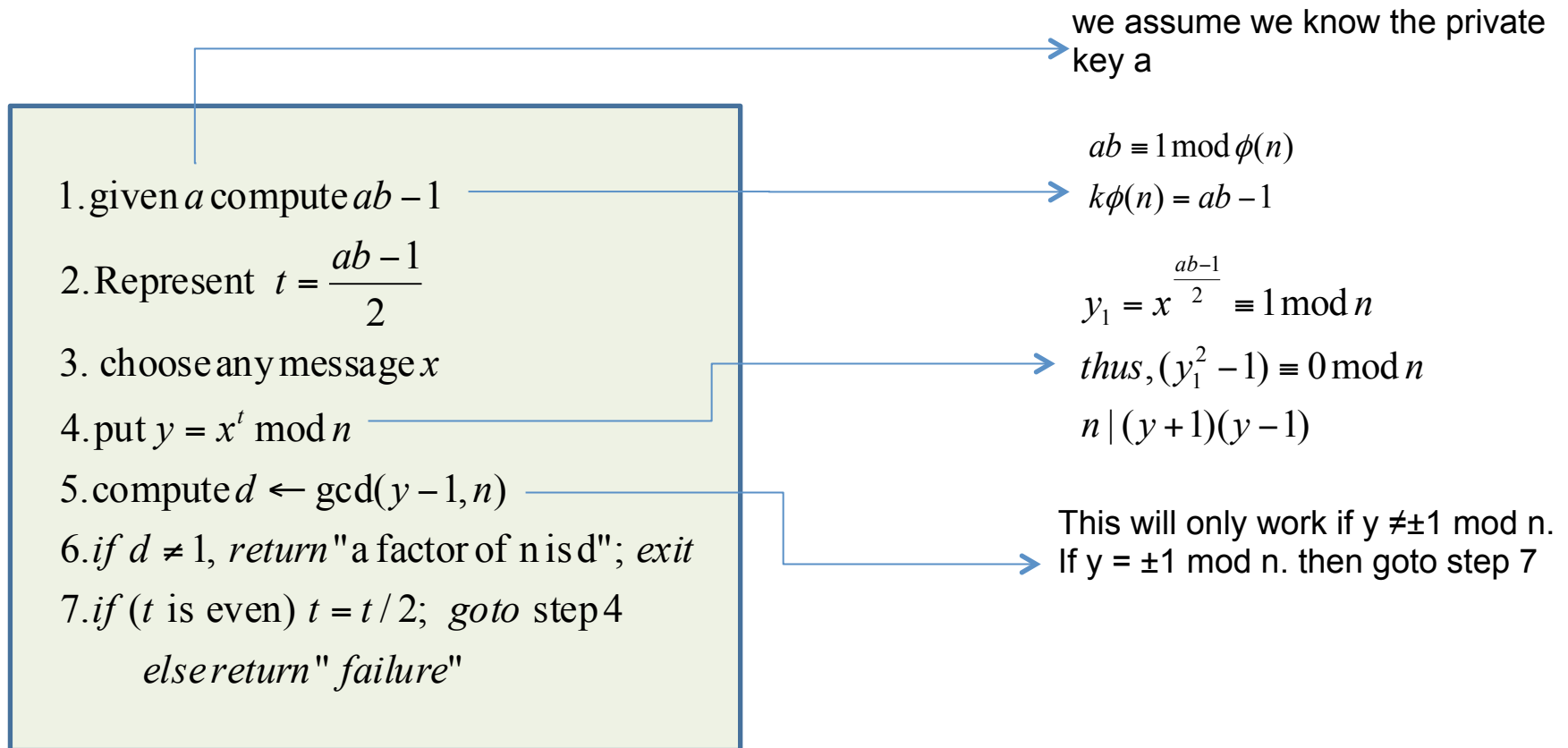
$\gcd(n, y - 1)$  is a factor of  $n$

However we  
need

$$y \neq \pm 1$$

to have a non-  
trivial result

# The Attack (basic idea)



Probability of success of the attack is at-least  $1/2$

# Example

- $N=403$ ,  $b=23$ ,  $a=47$

$$\begin{aligned} t &= ab - 1 = 1080 & x &= 2 \\ \text{loop1: } t &= \frac{1080}{2} = 540 & y &\equiv x^t \bmod 403 = 2^{540} \bmod 403 \equiv 1 \\ \text{loop2: } t &= \frac{540}{2} = 270 & y &\equiv x^t \bmod 403 = 2^{270} \bmod 403 \equiv 311 \\ & & \text{gcd}(310, 403) &= 31 \text{ (a factor of } n) \end{aligned}$$

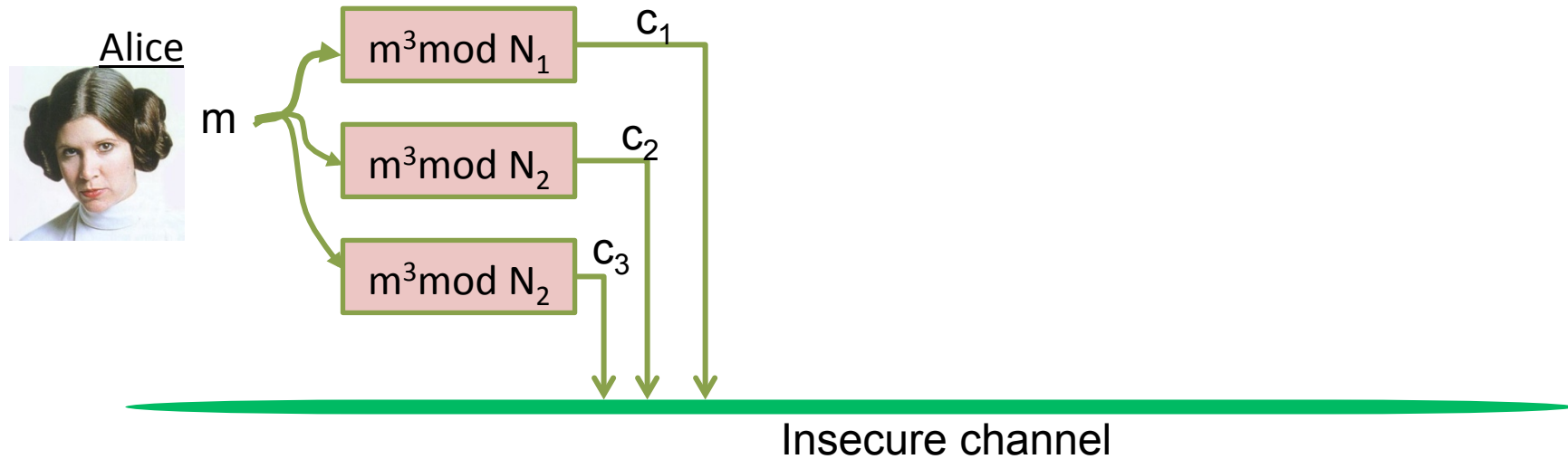
$$\begin{aligned} t &= ab - 1 = 1080 & x &= 9 \\ \text{loop1: } t &= \frac{1080}{2} = 540 & y &\equiv x^t \bmod 403 = 9^{540} \bmod 403 \equiv 1 \\ \text{loop2: } t &= \frac{540}{2} = 270 & y &\equiv x^t \bmod 403 = 9^{270} \bmod 403 \equiv 1 \\ \text{loop3: } t &= \frac{270}{2} = 135 & y &\equiv x^t \bmod 403 = 9^{135} \bmod 403 \equiv 1 \\ & & \text{can't divide 135 further. failure} \end{aligned}$$

# Small Encryption Exponent

- In order to improve efficiency of encryption, a small encryption exponent is preferred
- However, this can lead to a vulnerability

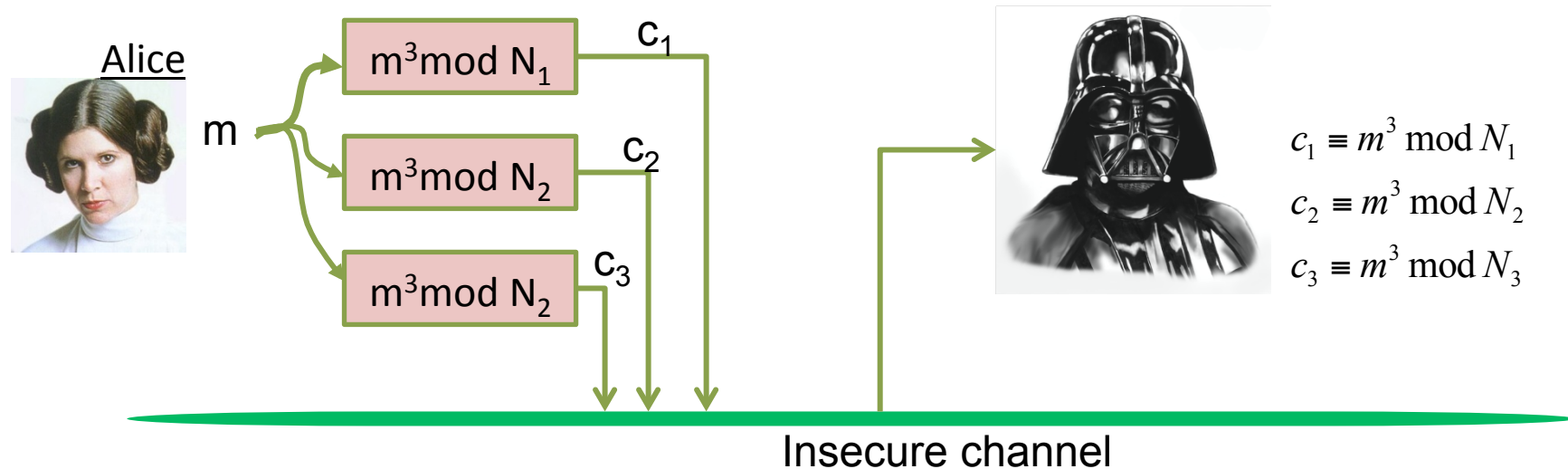


# Small Encryption Exponent



- Consider, Alice sending the same message  $x$  to 3 different people.
  - Each having a different  $N$  (say  $N_1, N_2, N_3$ )
  - But same public key  $b$  (say 3)

# Small Encryption Exponent



- Consider, Alice sending the same message  $x$  to 3 different people.
  - Each having a different  $N$  (say  $N_1, N_2, N_3$ )
  - But same public key  $b$  (say 3)
- This allows Mallory to snoop in and get 3 ciphertexts

# Small Encryption Exponent

By CRT

$$\begin{cases} c_1 \equiv m^3 \pmod{N_1} \\ c_2 \equiv m^3 \pmod{N_2} \\ c_3 \equiv m^3 \pmod{N_3} \end{cases} \quad \langle \Rightarrow \rangle X \equiv m^3 \pmod{N_1 \cdot N_2 \cdot N_3}$$

- Thus, Mallory can compute  $X$
- Since  $m < N_1, m < N_2, m < N_3 \Rightarrow m < (N_1 \times N_2 \times N_3)$
- Thus,  $X^{1/3} = m$ 
  - i.e. The message can be decrypted

It is tempting to have small private and public keys, so that encryption or decryption may be carried out efficiently. However you would do this at the cost of security!!

# Low Decryption Exponent

- The attack applies when the private key  $a$  is small,  $a < \frac{\sqrt[4]{n}}{3}$
- In such a case 'a' can be computed efficiently

# Partial Information of Plaintexts

## Computing Jacobi of the plaintext

$y \equiv x^b \pmod n$        $y$  is the ciphertext;  $x$  the message

$b$  is the public key and  $\gcd(b, \phi(n)) = 1$

Thus,  $\gcd(b, (p-1)(q-1)) = 1$

$(p-1)(q-1)$  is even, therefore  $b$  must be odd

*consider Jacobi*

$$\left(\frac{y}{n}\right) = \pm 1$$

$$\left(\frac{y}{n}\right) = \left(\frac{x}{n}\right)^b = \left(\frac{x}{n}\right)$$

since  $b$  is odd

thus, RSA encryption leaks the value of the Jacobi symbol  $\left(\frac{x}{n}\right)$

# Partial Information of Plaintexts

## first half or second half?

- given  $y = x^b \bmod n$ ,
  - is it possible to determine if
$$(0 \leq x < n/2) \quad \text{or} \quad (n/2 \leq x < n-1)$$

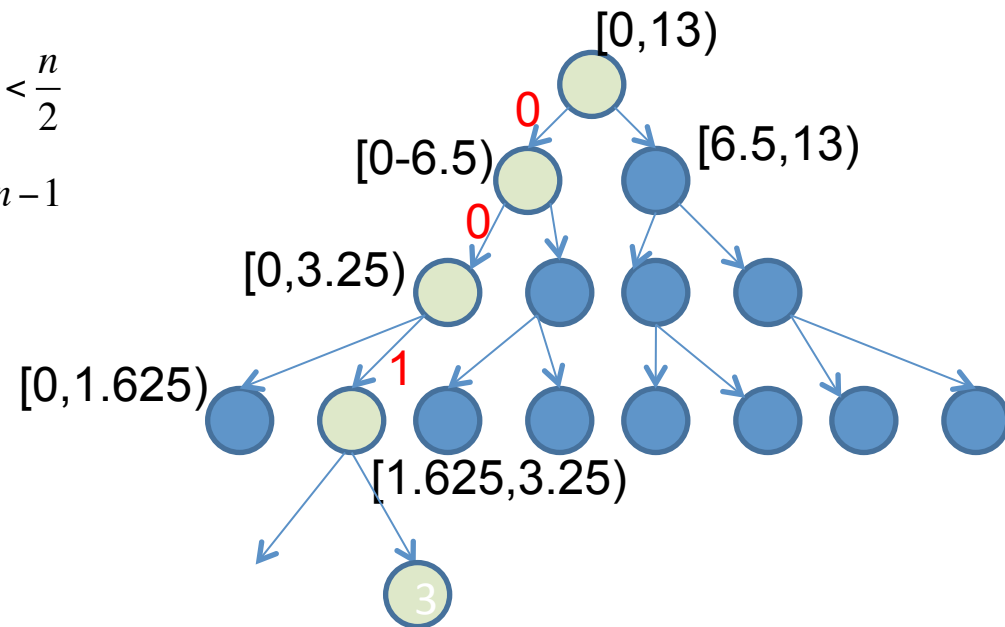
*first half**second half*
- We prove that RSA does not leak this information
  - If there exists an efficient algorithm that can determine if  $x$  is in the first or second half then, the entire plaintext can be obtained

# Find x

Consider this function

$$HALF(m) = \begin{cases} 0 & \text{if } 0 \leq mx \bmod n < \frac{n}{2} \\ 1 & \text{if } \frac{n}{2} \leq mx \bmod n < n-1 \end{cases}$$

example



# Partial Information of Plaintexts (first or second half proof)

- Assume a hypothetical oracle called HALF as follows

$$HALF(n, b, y) = \begin{cases} 0 & \text{if } 0 \leq x < \frac{n}{2} \\ 1 & \text{if } \frac{n}{2} \leq x < n-1 \end{cases}$$

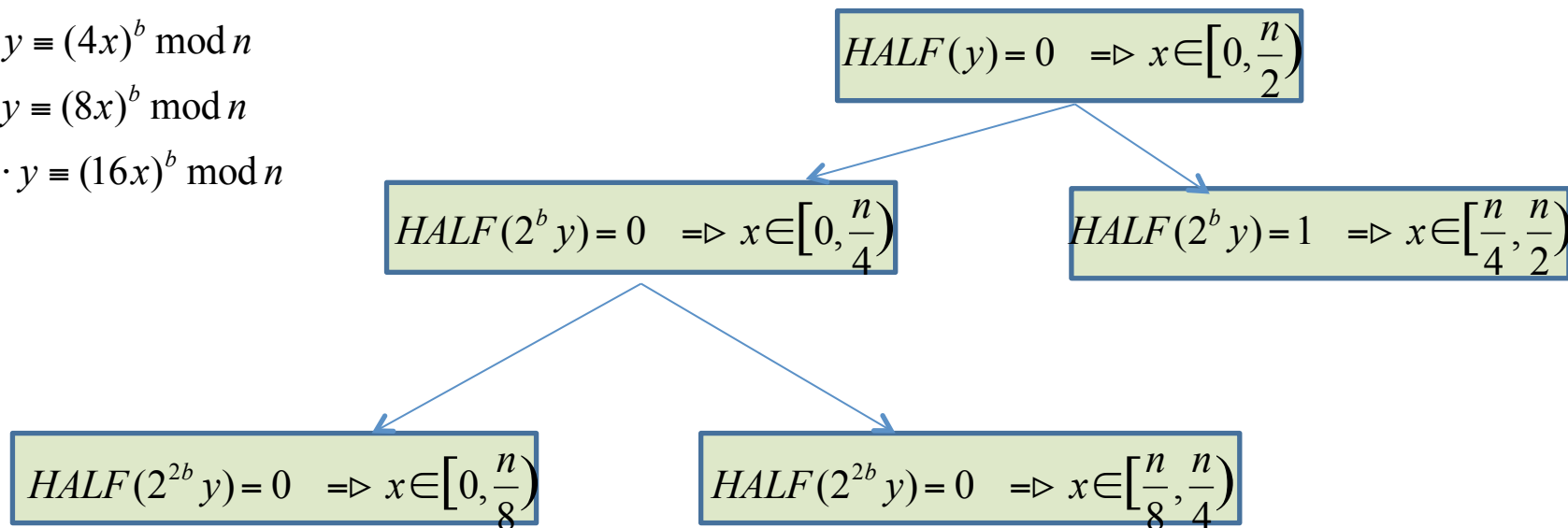
$$y \equiv x^b \pmod{n}$$

$$2^b \cdot y \equiv (2x)^b \pmod{n}$$

$$4^b \cdot y \equiv (4x)^b \pmod{n}$$

$$8^b \cdot y \equiv (8x)^b \pmod{n}$$

$$16^b \cdot y \equiv (16x)^b \pmod{n}$$





# Example

$n=1457, b=779, y=722$

**Algorithm** : ORACLE RSA DECRYPTION( $n, b, y$ )

**external** HALF

$k \leftarrow \lfloor \log_2 n \rfloor$

**for**  $i \leftarrow 0$  **to**  $k$

**do**  $\begin{cases} h_i \leftarrow \text{HALF}(n, b, y) \\ y \leftarrow (y \times 2^b) \bmod n \end{cases}$

$lo \leftarrow 0$

$hi \leftarrow n$

**for**  $i \leftarrow 0$  **to**  $k$

**do**  $\begin{cases} mid \leftarrow (hi + lo)/2 \\ \text{if } h_i = 1 \\ \quad \text{then } lo \leftarrow mid \\ \quad \text{else } hi \leftarrow mid \end{cases}$

**return** ( $\lfloor hi \rfloor$ )

$h_i$	$i$	$lo$	$mid$	$hi$
1	0	0.00	728.50	1457.00
0	1	728.50	1092.75	1457.00
1	2	728.50	910.62	1092.75
0	3	910.62	1001.69	1092.75
1	4	910.62	956.16	1001.69
1	5	956.16	978.92	1001.69
1	6	978.92	990.30	1001.69
1	7	990.30	996.00	1001.69
1	8	996.00	998.84	1001.69
0	9	998.84	1000.26	1001.69
0	10	998.84	999.55	1000.26
		998.84	999.55	999.55

Thus, if we have an efficient function HALF, we can recover the plaintext message.

# Man in the Middle Attack

- The process of encryption with a public key cipher



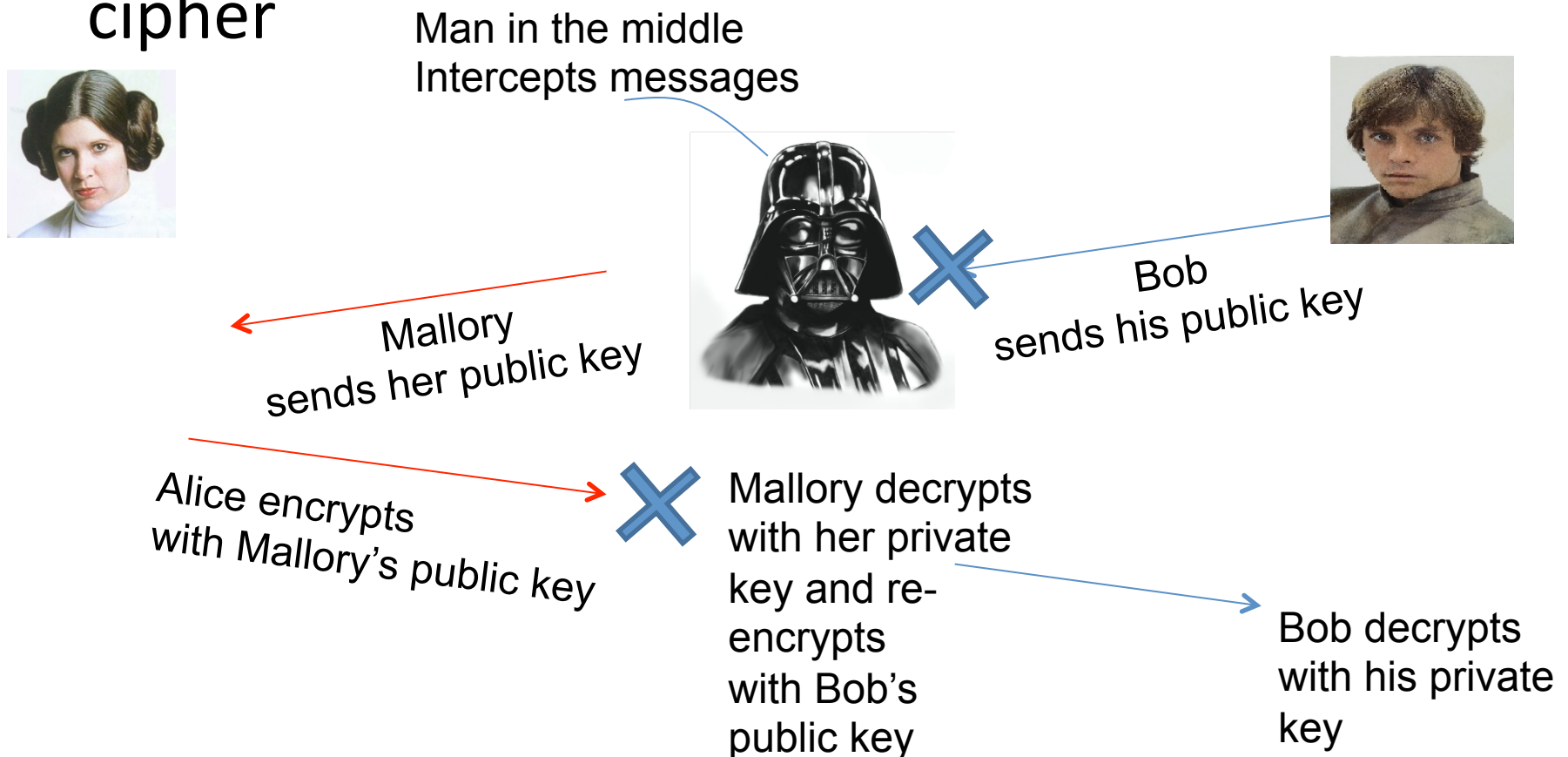
Bob sends his public key

Alice encrypts with Bob's public key

Bob decrypts with his private key

# Man in the Middle Attack

- The process of encryption with a public key cipher



# Searching the Message Space

- Suppose message space is small,
  - Mallory can try all possible messages, encrypt them (since she knows Bob's public key) and check if it matches Alice's ciphertext



Bob sends his public key

Alice encrypts with Bob's public key

Bob decrypts  
with his private  
key

# Bad Prime Generation Algorithms

- Suppose the prime generation was faulty
  - So that, primes generated were always from a small subset
  - Then, RSA can be broken
- Pairwise GCD of over a million RSA moduli collected from the Internet showed that
  - 2 in 1000 have a common prime factor

# Discrete Log Problem, ElGamal, and Diffie Hellman

# Primitive Elements of a Group

Let  $(G, \cdot)$  be a group of order  $n$ .

Let  $\alpha \in G$ ,

The order of  $\alpha$  is the smallest integer  $m$  such that  $\alpha^m = 1$

$\alpha$  is termed as a *primitive element* if it has order  $n$ .

If  $\alpha$  is a primitive element then

$$\langle \alpha \rangle = \{\alpha^i : 0 \leq i \leq n-1\} \text{ generates all elements in } G$$

Consider  $Z_{13}^* = \{1, 2, 3, \dots, 12\}$

$(Z_{13}^*, \cdot)$  forms a group of order 12

Let  $7 \in Z_{13}^*$ ,

$$\langle 7 \rangle = \{7, 10, 5, 9, 11, 12, 6, 3, 8, 4, 2, 1\}$$

$\langle 7 \rangle$  has order 12  
and generates all elements in  $Z$ .  
Thus, 7 is a primitive element

# Discrete Log Problem

Let  $(G, \cdot)$  be a group

Let  $\alpha \in G$  be a primitive element in the group with order  $n$

Define the set

$$\langle \alpha \rangle = \{\alpha^i : 0 \leq i \leq n-1\}$$

For any unique integer  $a$  ( $0 \leq a \leq n-1$ ),

$$\text{let } \alpha^a = \beta$$

Denote  $a = \log_{\alpha} \beta$  as the discrete logarithm of  $\beta$

Given  $\alpha$  and  $a$ , it is easy to compute  $\beta$

Given  $\alpha$  and  $\beta$  it is computationally difficult to determine what  $a$  was



# ElGamal Public Key Cryptosystem

- Fix a prime  $p$  (and group  $Z_p$ )
- Let  $\alpha \in Z_p$  be a primitive element
- Choose a secret 'a' and compute  $\beta \equiv \alpha^a \pmod{p}$

Public keys :  $\alpha, \beta, p$

Private key :  $a$



Encryption

choose a random (secret)  $k \leftarrow Z_p$

$$e_k(x) = (y_1, y_2)$$

where  $y_1 = \alpha^k \pmod{p}$ ,

$$y_2 = x \cdot \beta^k \pmod{p}$$



Decryption

$$\begin{aligned} d_k(x) &= y_2 (y_1^a)^{-1} \pmod{p} \\ &= x \cdot \beta^k (\alpha^{ka})^{-1} \pmod{p} \\ &= x \cdot \alpha^{ka} (\alpha^{ka})^{-1} \pmod{p} \\ &\equiv x \end{aligned}$$

# ElGamal Example

- $p = 2579$ ,  $\alpha = 2$  ( $\alpha$  is a primitive element mod  $p$ )
- Choose a random  $a = 765$
- Compute  $\beta \equiv 2^{765} \bmod 2579$

## Encryption of message $x = 1299$

choose a random key  $k = 853$

$$y_1 = 2^{853} \bmod 2579 = 435$$

$$y_2 = 1299 \times 949^{853} = 2396$$

## Decryption of cipher (435, 2396)

$$\begin{aligned} & 2396 \times (435^{765})^{-1} \bmod p \\ &= 1299 \end{aligned}$$

# Finding the Log

$$\beta \equiv \alpha^a \pmod{p}$$

Given  $\alpha$  and  $\beta$  it is computationally difficult to determine what  $a$  was

- Brute force (compute intensive)

compute  $\alpha, \alpha^2, \alpha^3, \alpha^4, \dots$  (until you reach  $\beta$ )

this would definitely work, but not practical if  $p$  is large

complexity  $O(p)$ , space complexity  $O(1)$

- Memory Intensive

precompute  $\alpha, \alpha^2, \alpha^3, \alpha^4, \dots$  (all values). Sort and store.

For any given  $\beta$  look up the table of stored values.

complexity  $O(1)$  but space complexity  $O(n)$

# Shank's Algorithm

(also known as Baby-step Giant-step)

$$\beta \equiv \alpha^a \pmod{p}$$

Rewrite  $a$  as  $a = mq + r$

where  $m = \lceil \sqrt{p} \rceil$

$$\beta \equiv \alpha^{mq} \alpha^r \pmod{p}$$

$$\beta (\alpha^{-m})^q \equiv \alpha^r \pmod{p}$$

We neither know  $q$  nor  $r$ , so we need to try out several values for  $q$  and  $r$  until we find a collision

# Shank's Algorithm (example)

- $p=31$  and  $\alpha=3$ . Suppose  $\beta=6$ .
- What is  $a$ ?

$$m = \lceil \sqrt{31} \rceil = 6$$

$$\alpha \equiv 3 \quad \leftarrow \text{collision}$$

List 1

$$\alpha^2 \equiv 9$$

$$\alpha^3 \equiv 27$$

$$\alpha^4 = 81 \equiv 19 \pmod{31}$$

$$\alpha^5 = 19 \cdot 3 \equiv 26 \pmod{31}$$

List 2

$$(3^{-1})^6 \pmod{31} = 2$$

$$\beta(\alpha^{-6})^0 = 6 \cdot 2^0 = 6$$

$$\beta(\alpha^{-6})^1 = 6 \cdot 2^1 = 12$$

$$\beta(\alpha^{-6})^2 = 6 \cdot 2^2 = 24$$

$$\beta(\alpha^{-6})^3 = 6 \cdot 2^3 \equiv 17 \pmod{31}$$

$$\beta(\alpha^{-6})^4 = 6 \cdot 2^4 \equiv 3 \pmod{31}$$

Thus,  $m=6, q=4, r=1, \quad a = mq+r = 25$

# Shank's Algorithm

**Algorithm 6.1:** SHANKS( $G, n, \alpha, \beta$ )

1.  $m \leftarrow \lceil \sqrt{n} \rceil$
2. **for**  $j \leftarrow 0$  **to**  $m - 1$   
    **do** compute  $\alpha^{mj}$
3. Sort the  $m$  ordered pairs  $(j, \alpha^{mj})$  with respect to their second coordinates, obtaining a list  $L_1$
4. **for**  $i \leftarrow 0$  **to**  $m - 1$   
    **do** compute  $\beta\alpha^{-i}$
5. Sort the  $m$  ordered pairs  $(i, \beta\alpha^{-i})$  with respect to their second coordinates, obtaining a list  $L_2$
6. Find a pair  $(j, y) \in L_1$  and a pair  $(i, y) \in L_2$  (i.e., find two pairs having identical second coordinates)
7.  $\log_{\alpha} \beta \leftarrow (mj + i) \bmod n$

**Create List 1**

**Create List 2**

**Find collision**

# Complexity of Shank's Algorithm

## Algorithm 6.1: SHANKS( $G, n, \alpha, \beta$ )

1.  $m \leftarrow \lceil \sqrt{n} \rceil$
2. **for**  $j \leftarrow 0$  **to**  $m - 1$   
    **do** compute  $\alpha^{mj}$  →  $O(m)$
3. Sort the  $m$  ordered pairs  $(j, \alpha^{mj})$  with respect to their second coordinates, obtaining a list  $L_1$  →  $O(m \log m)$
4. **for**  $i \leftarrow 0$  **to**  $m - 1$   
    **do** compute  $\beta \alpha^{-i}$  →  $O(m)$
5. Sort the  $m$  ordered pairs  $(i, \beta \alpha^{-i})$  with respect to their second coordinates, obtaining a list  $L_2$  →  $O(m \log m)$
6. Find a pair  $(j, y) \in L_1$  and a pair  $(i, y) \in L_2$  (i.e., find two pairs having identical second coordinates) →  $O(\log m)$
7.  $\log_{\alpha} \beta \leftarrow (mj + i) \bmod n$

$$O(m \log m) \sim O(m) = O(p^{1/2})$$

# Other Discrete Log Algorithms

$$\beta \equiv \alpha^a \pmod{n}$$

- Pollard-Hellman Algorithm  
used when  $n$  is a composite
- Pollard-Rho Algorithm  
about the same runtime as the Shank's algorithm, but has much less memory requirements



# Diffie Hellman Problem

*Let  $(G, \cdot)$  be a group*

*Let  $\alpha \in G$  be a primitive element in the group with order  $n$*

*Define the set*

$$\langle \alpha \rangle = \{\alpha^i : 0 \leq i \leq n-1\}$$

*given  $\alpha^a$  and  $\alpha^b$ , find  $\alpha^{ab}$*

Computational DH (CDH)

*given  $\alpha^a, \alpha^b$  and  $\alpha^c$ , determine if  $c \equiv ab \pmod{n}$*

Decision DH (DDH)

# Recall...

## Diffie Hellman Key Exchange



Alice and Bob agree upon a prime  $p$  and a generator  $g$ .  
This is public information



choose a secret  $a$   
compute  $A = g^a \bmod p$

choose a secret  $b$   
compute  $B = g^b \bmod p$

Compute  $K = B^a \bmod p$  ←  $B$        $A$  → Compute  $K = A^b \bmod p$

$$A^b \bmod p = (g^a)^b \bmod p = (g^b)^a \bmod p = B^a \bmod p$$