Introduction to Basic Cryptography

Attacks on RSA, DLP

Kalyan Chakraborty
Harish-Chandra Research Institute

CIMPA School of Number Theory in Cryptography and Its Applications
School of Science, Kathmandu University,
Dhulikhel, Nepal
July 19 - July 31, 2010

Lecture 2: July 21, 2010

http://www.hri.res.in/~jaymehta/cryptographynotesCIMPA2010.pdf

Outline

Factoring Algorithms

Factoring Algorithms

The obvious way to attack RSA is to attempt to factor the public modulus.

If n is composite, then the basic method of dividing an integer n by all primes $p \leq \sqrt{n}$ is much too slow for most purposes.

The Pollard $p-1$ Algorithm

- If one of the prime factors of n has a special property, it is sometime easier to factor n. For example, if $p|n$ and $p-1$ has "small" prime factors, the following method is effective.

The Pollard $p-1$ Algorithm

- If one of the prime factors of n has a special property, it is sometime easier to factor n. For example, if $p|n$ and $p-1$ has "small" prime factors, the following method is effective.
- This algorithm was invented by Pollard in 1974.

The Pollard $p-1$ Algorithm

- If one of the prime factors of n has a special property, it is sometime easier to factor n. For example, if $p|n$ and $p-1$ has "small" prime factors, the following method is effective.
- This algorithm was invented by Pollard in 1974.
- Fix an integer B. Given an integer n, this algorithm finds a prime p such that $p|n$ and $p-1$ has prime factors $\leq B$.

The Pollard $p-1$ Algorithm

- If one of the prime factors of n has a special property, it is sometime easier to factor n. For example, if $p|n$ and $p-1$ has "small" prime factors, the following method is effective.
- This algorithm was invented by Pollard in 1974.
- Fix an integer B. Given an integer n, this algorithm finds a prime p such that $p|n$ and $p-1$ has prime factors $\leq B$.

Pollard $p-1$ factoring Algorithm (n, B)

$a = 2$

for $j = 2$ to B

do $a = a^j \bmod n$

$d = gcd(a-1, n)$

if $1 < d < n$

then return(d)

else return ("failure")

Method

Suppose $p|n$ and $q \leq B$ $\forall$ prime power $q|p-1$.
Then it must be the case that $p-1|B!$.
At the end of for loop one has

$$a \equiv 2^{B!} \pmod{n}.$$

As $p|n$ one has

$$a \equiv 2^{B!} \pmod{p}.$$

By Fermat's little theorem;

$$2^{p-1} \equiv 1 \pmod{p}$$

As, $p-1|B!$, one has

$$a \equiv 1 \pmod{p}$$

$\Rightarrow p|a-1$. Also, $p|n \Rightarrow p|d$; where $d = gcd(a-1, n)$.
d will be a non-trivial divisor of n (unless $a = 1$)

- How to choose B ?
  For a small B the algorithm will run quickly but will have little chance of success, while large B will make it very slow.
  The actual choice of B will depend on the situation at hand.

- <u>How to choose B</u> ?
  For a small B the algorithm will run quickly but will have little chance of success, while large B will make it very slow.
  The actual choice of B will depend on the situation at hand.

- <u>Avoding $p-1$ attack</u> :
  We use $n = pq$, which is hard to factor. So, we should ensure that $p-1$ has at least one large prime factor.

  For p to have around 100 digits, choose large prime p_0 (around 10^{40}). Look at integers of the form $kp_0 + 1$ (k could be around 10^{60}) and test it for primality by Miller-Rabin Test. This produces desired value of p in less than 100 steps.
  Apply same procedure for q.

  Then the RSA modulus n will be resistant to $p-1$ attack.

Complexity:

There are $B-1$ modular exponentiation, each requiring at most $2\log_2 B$ modular multiplications. The gcd can be computed in time $\mathrm{O}\big((\log n)^3\big)$.

Hence the complexity of the algorithm is

$$O\Big(B\log B(\log n)^3 + (\log n)^3\Big).$$

If B is $\mathrm{O}\big((\log n)^i\big)$ for some fixed i, then algorithm is indeed polynomial time. One cannot again choose a very small B.

The more powerful elliptic curve factoring algorithm developed by Lenstra, is infact a generalization of the $p-1$ algorithm.

Example

- Suppose $n = 15770708441$. If we apply the algorithm with $B = 180$ then one gets $a = 11620221425$ and $d = 135979$
- In fact $15770708441 = 135979 \times 115979$
- This factorization succeeds because 135978 has “small” prime factors:
  $135978 = 2 \times 3 \times 131 \times 173.$
- Hence by taking $B \geq 173$, it will be the case that $135978|B!$ as required.

Example

- Suppose $n = 15770708441$. If we apply the algorithm with $B = 180$ then one gets $a = 11620221425$ and $d = 135979$
- In fact $15770708441 = 135979 \times 115979$
- This factorization succeeds because 135978 has "small" prime factors:
  $135978 = 2 \times 3 \times 131 \times 173$.
- Hence by taking $B \geq 173$, it will be the case that $135978|B!$ as required.

Exercise: Factor $n = 376875575426394855599989992897873239$ by the $p-1$ Method. One can choose bound $B = 100$.

Pollard Rho Algorithm

- Let p be prime divisor of n. Suppose $\exists\ x, x' \in \mathbb{Z}_n$ such that $x \neq x'$ and $x \equiv x' \mod p$. Then $p \leq (x - x', n) < n$. One obtains a non-trivial factor of n by computing gcd.

Pollard Rho Algorithm

- Let p be prime divisor of n. Suppose $\exists\ x, x' \in \mathbb{Z}_n$ such that $x \neq x'$ and $x \equiv x' \mod p$. Then $p \leq (x - x', n) < n$. One obtains a non-trivial factor of n by computing gcd.
- One chooses a random subset $X \subset \mathbb{Z}_n$ and then compute $(x - x', n)$ for all distinct values in X. This method will be successful if and only if the mapping $x \to x \mod p$ gives atleast one collision for $x \in X$.

Pollard Rho Algorithm

- Let p be prime divisor of n. Suppose $\exists\ x, x' \in \mathbb{Z}_n$ such that $x \neq x'$ and $x \equiv x' \mod p$. Then $p \leq (x - x', n) < n$. One obtains a non-trivial factor of n by computing gcd.
- One chooses a random subset $X \subset \mathbb{Z}_n$ and then compute $(x - x', n)$ for all distinct values in X. This method will be successful if and only if the mapping $x \to x \mod p$ gives atleast one collision for $x \in X$.
- The Pollard Rho Algorithm is a variation of this technique which needs fewer gcd computation and less memory and quickly finds relatively small prime factor p of composite numbers in perhaps $\sqrt{p}$ steps.

Description

Description: Given an integer n, define a function f by

$$f(x) = x^2 + a \bmod n \text{ (usually } a = 1 \text{ is used)}$$

and initialize by setting $x = 2, x' = f(x)$.

Description

Description: Given an integer n, define a function f by

$$f(x) = x^2 + a \bmod n \text{ (usually } a = 1 \text{ is used)}$$

and initialize by setting $x = 2, x' = f(x)$.

Compute $d = gcd(x - x', n)$
if $1 < d < n$, **stop**, d is proper divisor of n.
if $d = 1$, replace x by $f(x)$ and x' by $f(f(x'))$;
and repeat.

Description

Description: Given an integer n, define a function f by

$$f(x) = x^2 + a \bmod n \text{ (usually } a = 1 \text{ is used)}$$

and initialize by setting $x = 2, x' = f(x)$.

Compute $d = gcd(x - x', n)$
if $1 < d < n$, **stop**, d is proper divisor of n.
if $d = 1$, replace x by $f(x)$ and x' by $f(f(x'))$;
and repeat.

(If $d = n$; we have a failure and the algorithm has to be re-initialized).

Pollard Rho Factoring Algorithm (n, x_1)

external f

$x \leftarrow x_1$

$x' \leftarrow f(x) \bmod n$

$p \leftarrow \gcd(x - x', n)$

while $p = 1$

$$\textbf{do} \begin{cases} \text{comment: in the ith iteration, } x = x_i \text{ and } x' = x_{2i} \\ x \leftarrow f(x) \pmod{n} \\ x' \leftarrow f(x') \pmod{n} \\ x' \leftarrow f(x') \pmod{n} \\ p \leftarrow \gcd(x - x', n) \end{cases}$$

if $p = n$

then return ("n is prime")

else return (p)

- Let $x_1 \in \mathbb{Z}_n$. Consider the sequence $x_1, x_2, \ldots$ where

$$x_j = f(x_{j-1}) \text{mod } n \; \forall \; j \geq 2.$$

Set $X = \{x_1, x_2, \ldots x_m\}$ for some m.

- Let $x_1 \in \mathbb{Z}_n$. Consider the sequence $x_1, x_2, \ldots$ where

$$x_j = f(x_{j-1}) \text{mod } n \; \forall \; j \geq 2.$$

  Set $X = \{x_1, x_2, \ldots x_m\}$ for some m.

- One is looking for two distinct values $x_i, x_j \in X$ such that $\gcd(x_j - x_i, n) > 1$.
  It turns out that one can reduce the gcd computation.

- Let $x_1 \in \mathbb{Z}_n$. Consider the sequence $x_1, x_2, \ldots$ where

$$x_j = f(x_{j-1}) \text{mod } n \; \forall \; j \geq 2.$$

Set $X = \{x_1, x_2, \ldots x_m\}$ for some m.

- One is looking for two distinct values $x_i, x_j \in X$ such that $\gcd(x_j - x_i, n) > 1$.
  It turns out that one can reduce the gcd computation.

- Now we show that $x_i \equiv x_j \pmod{p} \Rightarrow x_{i+1} \equiv x_{j+1} \pmod{p}$:
  Suppose $x_i \equiv x_j \pmod{p} \Rightarrow f(x_i) \equiv f(x_j) \pmod{p}$;

$$x_{i+1} = f(x_i) \bmod n$$
$$x_{j+1} = f(x_j) \bmod n$$

gives

- Let $x_1 \in \mathbb{Z}_n$. Consider the sequence $x_1, x_2, \ldots$ where

$$x_j = f(x_{j-1}) \text{mod } n \;\forall\; j \geq 2.$$

Set $X = \{x_1, x_2, \ldots x_m\}$ for some m.

- One is looking for two distinct values $x_i, x_j \in X$ such that $\gcd(x_j - x_i, n) > 1$.
  It turns out that one can reduce the gcd computation.

- Now we show that $x_i \equiv x_j \pmod{p} \Rightarrow x_{i+1} \equiv x_{j+1} \pmod{p}$:
  Suppose $x_i \equiv x_j \pmod{p} \Rightarrow f(x_i) \equiv f(x_j) \pmod{p}$;

$$x_{i+1} = f(x_i) \bmod n$$
$$x_{j+1} = f(x_j) \bmod n$$

gives

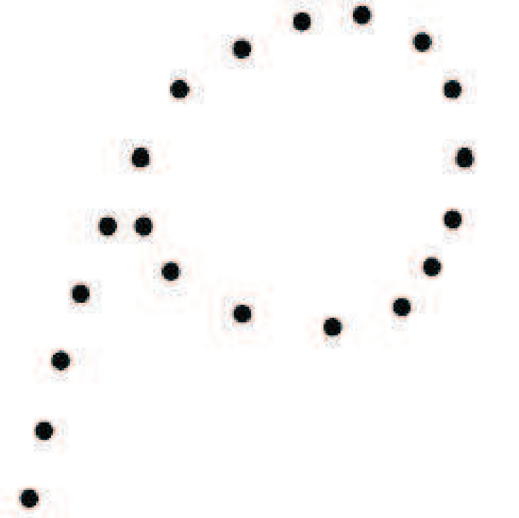
$$x_{i+1} \bmod p = \big(f(x_i) \bmod n\big) \bmod p = f(x_i) \bmod p \text{ as } p|n.$$
$$x_{j+1} \bmod p = \big(f(x_j) \bmod n\big) \bmod p = f(x_j) \bmod p \text{ as } p|n.$$

Therefore, $x_{i+1} \equiv x_{j+1} \pmod{p}$.

- So, if $x_i \equiv x_j \pmod{p}$ then $x_{i+\delta} \equiv x_{j+\delta} \pmod{p}$; $\delta \geq 0$.
  Denoting $l = j - i$
  $\Rightarrow x_{i'} \equiv x_{j'} \pmod{p}$ if $j' > i' \geq i$ and $j' - i' = 0 \pmod{l}$.

- So, if $x_i \equiv x_j \pmod{p}$ then $x_{i+\delta} \equiv x_{j+\delta} \pmod{p}$; $\delta \geq 0$.
  Denoting $l = j - i$
  $\Rightarrow x_{i'} \equiv x_{j'} \pmod{p}$ if $j' > i' \geq i$ and $j' - i' = 0 \pmod{l}$.
- The goal is to discover 2 terms $x_i \equiv x_j \pmod{p}$ with $i < j$, by computing a gcd. In order to simplify and improve the algorithm, we restrict our search for collision by taking $j = 2i$.

- So, if $x_i \equiv x_j \pmod{p}$ then $x_{i+\delta} \equiv x_{j+\delta} \pmod{p}$; $\delta \geq 0$.
  Denoting $l = j - i$
  $\Rightarrow x_{i'} \equiv x_{j'} \pmod{p}$ if $j' > i' \geq i$ and $j' - i' = 0 \pmod{l}$.
- The goal is to discover 2 terms $x_i \equiv x_j \pmod{p}$ with $i < j$, by computing a gcd. In order to simplify and improve the algorithm, we restrict our search for collision by taking $j = 2i$.
- If $x_i \equiv x_j \pmod{p}$ then it is the case that $x_{i'} \equiv x_{2i'} \pmod{p}$ for all i' such that $i' \equiv 0 \pmod{l}$ and $i' \geq i$.
- Among the l consecutive integers $i, \ldots, j-1$, there must be one that is divisible by l. Therefore, the smallest value i' that satisfies the condition is atmost $j-1$. Hence the # of iterations required to find a factor p is atmost j.



Example

$n = 7171 = 71 \times 101, f(x) = x^2 + 1, x_1 = 1$, then the sequence x_i are

1	2	5	26	677	6557	4101
6347	4903	2218	219	4936	·	·
4872	375	4377	4389	2016	·	·

reduced mod 71

1	2	5	26	38	25	58
28	4	17	6	37	21	16
44	20	46	58	28	4	17

$\Rightarrow x_7 \bmod 71 = x_{18} \bmod 71 = 58$.

Here, $i = 7$, $j = 18$ and $l = j - i = 11$ which is the length of the cycle. The smallest integer $i' \geq 7$ which is divisible by 11 is $i' = 11$. Therefore, this algorithm will give the factor 71 of n when it computes $\gcd(x_{11} - x_{22}, n) = 71$.

Exercises

Exercise:
Factor the following numbers using Pollard Rho Algorithm if the function f is defined to be $f(x) = x^2 + 1$:

1. 262063
2. 9420457
3. 181937053

How many iterations are needed in each cases?

Weiner's Low Decryption Exponent attack

- This attack succeeds in computing the secret decryption exponent d, whenever

$$3d < n^{1/4} \quad \text{and} \quad q < p < 2q. \tag{1}$$

Weiner's Low Decryption Exponent attack

- This attack succeeds in computing the secret decryption exponent d, whenever

$$3d < n^{1/4} \quad \text{and} \quad q < p < 2q. \tag{1}$$

- This attack will work when d has fewer than $l/4^{-1}$ bits in binary representation (assuming n has l-bits) and p and q are not far apart.
- This result shows that method of reducing time should be avoided.

- As, $de \equiv 1 \pmod{\phi(n)}$ $\exists\, t \in \mathbb{Z}$ such that

$$de - t\phi(n) = 1$$

Now, $n = pq > q^2$, we have $q < \sqrt{n}$. Therefore,

$$0 < n - \phi(n) = p + q - 1 < 2q + q - 1 < 3q < 3\sqrt{n}.$$

Thus,

$$\left|\frac{e}{n} - \frac{t}{d}\right| = \left|\frac{ed - tn}{dn}\right| = \left|\frac{1 + t(\phi(n) - n)}{dn}\right| < \frac{3t\sqrt{n}}{dn} = \frac{3t}{d\sqrt{n}}.$$

- As, $de \equiv 1 \pmod{\phi(n)}$ $\exists\, t \in \mathbb{Z}$ such that

$$de - t\phi(n) = 1$$

  Now, $n = pq > q^2$, we have $q < \sqrt{n}$. Therefore,

$$0 < n - \phi(n) = p + q - 1 < 2q + q - 1 < 3q < 3\sqrt{n}.$$

  Thus,

$$\left|\frac{e}{n} - \frac{t}{d}\right| = \left|\frac{ed - tn}{dn}\right| = \left|\frac{1 + t(\phi(n) - n)}{dn}\right| < \frac{3t\sqrt{n}}{dn} = \frac{3t}{d\sqrt{n}}.$$

- As $t < d$; one has $3t < 3d < n^{1/4}$.
  Therefore

$$\left|\frac{e}{n} - \frac{t}{d}\right| = \frac{1}{dn^{1/4}} < \frac{1}{3d^2}$$

  $\frac{t}{d}$ is very close approximation to $\frac{e}{n}$.

- From the theory of continued fractions, it is known that any approximation of $\frac{e}{n}$ that is this close must be one of the convergents of the continued fraction expansion of $\frac{e}{n}$.

- From the theory of continued fractions, it is known that any approximation of $\frac{e}{n}$ that is this close must be one of the convergents of the continued fraction expansion of $\frac{e}{n}$.
- **Theorem:** Suppose $(a, b) = (c, d) = 1$ and

$$\left|\frac{a}{b} - \frac{c}{d}\right| < \frac{1}{2d^2}.$$

Then $\frac{c}{d}$ is one of the convergents of the continued fractions expansion of $\frac{a}{b}$.

Example

Example: Compute the continued fraction expansion of $\frac{34}{99}$.
The Euclidean Algorithm gives:

$$\begin{aligned} 34 &\equiv 0 \times 99 + 34 \\ 99 &\equiv 2 \times 34 + 31 \\ 34 &\equiv 1 \times 31 + 3 \\ 31 &\equiv 10 \times 3 + 1 \\ 3 &\equiv 3 \times 1 \end{aligned}$$

Hence, the continued fraction expansion of $\frac{34}{99} = [0, 2, 1, 10, 3]$.

$$\frac{34}{99} = 0 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{10 + \frac{1}{3}}}}$$

The convergents are:

$$[0] = 0,\ [0,2] = \frac{1}{2},\ [0,2,1] = \frac{1}{3},\ [0,2,1,10] = \frac{11}{32},\ [0,2,1,10,3] = \frac{34}{99}.$$

- Thus if (1) holds then the unknown fraction $\frac{t}{d}$ is a close approximation to $\frac{e}{n}$.

  If $\frac{t}{d}$ is a convergent of $\frac{e}{n}$; then one can compute the value of

  $$\phi(n) = (de - 1)/t.$$

- Once n and $\phi(n)$ are known we can factor n.
- One tries convergent of $\frac{e}{n}$ until the factorization of n is found.

Weiner's Algorithm (n, e)

$(q_1, \ldots, q_m; r_m) \leftarrow$ Euclidean Algorithm (e, n).
$c_0 = 1$, $c_1 = q_1$, $d_0 = 0$, $d_1 = 1$
for $j = 1$ to m
do
$c_j = q_j c_{j-1} + c_{j-2}$
$d_j = q_j d_{j-1} + d_{j-2}$
$n' = (d_j e - 1)/c_j$
comment: $n' = \phi(n)$ if c_j/d_j is the correct convergent.
if n' is an integer

then {
- let p, q be roots of $x^2 - (n - n' + 1)x + n = 0$
- **if** p, q are primitive integers $< n$
- **then** return (p, q)

return("failure")

End of RSA Notes

Discrete Logarithm Problem

- Let G be a finite multiplicative group of order n.
  Let $\alpha \in G$ be of order n.

Discrete Logarithm Problem

- Let G be a finite multiplicative group of order n.
  Let $\alpha \in G$ be of order n.
- Consider,
  $$< \alpha > = \{\alpha^i \ : \ 0 \leq i \leq n-1\}.$$
  Set $\beta \ \in < \alpha >$.

Discrete Logarithm Problem

- Let G be a finite multiplicative group of order n.

  Let $\alpha \in G$ be of order n.

- Consider,

  $$< \alpha > = \{\alpha^i \ : \ 0 \leq i \leq n-1\}.$$

  Set $\beta \ \in < \alpha >$.

- For example, take G to be multiplicative group of a finite field $\mathbb{Z}_p$ and α to be a primitive element modulo p.

- **Problem :** Find the unique integer 'a', $0 \leq a \leq n-1$ such that

$$\alpha^a = \beta.$$

Denote 'a' by $\log_\alpha \beta$; discrete logarithm of β.

- **Problem :** Find the unique integer 'a', $0 \leq a \leq n-1$ such that

$$\alpha^a = \beta.$$

  Denote 'a' by $\log_\alpha \beta$; discrete logarithm of β.

- The utility of 'dlp' in a cryptographic setting is that finding 'dl' is difficult.

- **Problem :** Find the unique integer 'a', $0 \leq a \leq n-1$ such that

$$\alpha^a = \beta.$$

  Denote 'a' by $\log_\alpha \beta$; discrete logarithm of β.

- The utility of 'dlp' in a cryptographic setting is that finding 'dl' is difficult.

- Stated Another Way: Exponentiation is a one-way formula in suitable groups.

DLP Example

$$G = \mathbb{Z}_{19}^{*} = \{1, 2, \ldots, 18\}.$$

$$n = 18;\ \text{generator } g = 2$$

i	1	2	3	4	5	6	7	8	9
g^i	2	4	8	16	13	7	14	9	18

i	10	11	12	13	14	15	16	17	18
g^i	17	15	11	3	6	12	5	10	1

Then,

$$\log_2 14 = 7$$

$$\log_2 6 = 14$$

Exercises

1. Let $p = 13$. Compute $\log_2 3$.

Computer Exercises:

1. Let $p = 53047$. Verify that $\log_3 8576 = 1234$.
2. Let $p = 3989$. Show that $\log_2 3925 = 2000$.

The ElGamal Cryptosystem This cryptosystem is based on DLP.
ElGamal proposed a PKC based on dlp in $(\mathbb{Z}_p^*, \cdot)$.

The ElGamal Cryptosystem This cryptosystem is based on DLP.
ElGamal proposed a PKC based on dlp in $(\mathbb{Z}_p^*, \cdot)$.

Encryption : Let p be a prime and g be a generator of $\mathbb{Z}_p^*$. The private key x is an integer between 1 and $p-2$. Set

$$y = g^x \bmod p.$$

Public key $= (p, g, y)$. Releasing $y = g^x \bmod p$ doesn't reveal x.

The ElGamal Cryptosystem This cryptosystem is based on DLP.
ElGamal proposed a PKC based on dlp in $(\mathbb{Z}_p^*, \cdot)$.

Encryption : Let p be a prime and g be a generator of $\mathbb{Z}_p^*$. The private key x is an integer between 1 and $p-2$. Set

$$y = g^x \bmod p.$$

Public key $= (p, g, y)$. Releasing $y = g^x \bmod p$ doesn't reveal x. To encrypt a plaintext M, a random integer k coprime to $p-1$ is selected and compute

$$a = g^k \bmod p$$
$$b = My^k \bmod p$$

Ciphertext $C = (a, b)$.

The ElGamal Cryptosystem This cryptosystem is based on DLP.

ElGamal proposed a PKC based on dlp in $(\mathbb{Z}_p^*, \cdot)$.

Encryption : Let p be a prime and g be a generator of $\mathbb{Z}_p^*$. The private key x is an integer between 1 and $p-2$. Set

$$y = g^x \bmod p.$$

Public key $= (p, g, y)$. Releasing $y = g^x \bmod p$ doesn't reveal x. To encrypt a plaintext M, a random integer k coprime to $p-1$ is selected and compute

$$a = g^k \bmod p$$
$$b = My^k \bmod p$$

Ciphertext $C = (a, b)$.

Decryption : $M = \frac{b}{a^x} \bmod p$. Indeed we have;

$$\begin{aligned} \frac{b}{a^x} \bmod p &= My^k(a^x)^{-1} \bmod p \\ &= Mg^{xk}(g^{kx})^{-1} \bmod p \\ &= M. \end{aligned}$$

Example

- Suppose $p = 2579, \alpha = 2$. Let $a = 765$. So,

$$\beta = 2^{765} \mod 2579 = 949.$$

Example

- Suppose $p = 2579, \alpha = 2$. Let $a = 765$. So,

$$\beta = 2^{765} \mod 2579 = 949.$$

- Alice wishes to send a message $x = 1299$ to Bob. $k = 853$. Then she computes,

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

and

$$y_2 = 1299 \times 949^{853} \mod 2579 = 2396.$$

Example

- Suppose $p = 2579, \alpha = 2$. Let $a = 765$. So,

$$\beta = 2^{765} \mod 2579 = 949.$$

- Alice wishes to send a message $x = 1299$ to Bob. $k = 853$. Then she computes,

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

  and

$$y_2 = 1299 \times 949^{853} \mod 2579 = 2396.$$

- Bob receives the ciphertext $y = (435, 2396)$ and he computes

$$x = 2396 \times (435^{765})^{-1} \mod 2579 = 1299.$$

Massey - Omura Encryption

- **Parameters**
  - p : prime number.
  - e : a random private integer such that

$$0 < e < p \quad \text{and} \quad \gcd(e, p-1) = 1$$

  - d : an inverse of e

$$d = e^{-1} \bmod p-1, \text{ i.e., } de \equiv 1 \bmod p-1$$

  - M : a message to be encrypted and decrypted

Massey - Omura Encryption

- **Parameters**
  - p : prime number.
  - e : a random private integer such that

$$0 < e < p \quad \text{and} \quad \gcd(e, p-1) = 1$$

  - d : an inverse of e

$$d = e^{-1} \bmod p-1, \text{ i.e., } de \equiv 1 \bmod p-1$$

  - M : a message to be encrypted and decrypted
- **Alice wants to send a message M to Bob**
  - For Alice : e_A and d_A are both private
  - For Bob : e_B and d_B are both private

Massey-Omura for message transmission

Alice

1. Encryption (1)

$$C_1 = M^{e_A} \bmod p$$

Massey-Omura for message transmission

Alice

1. Encryption (1)

$$C_1 = M^{e_A} \bmod p$$

Bob

2. Encryption(2)

$$\begin{aligned} C_2 &= {C_1}^{e_B} \\ &= M^{e_A e_B} \bmod p \end{aligned}$$

Massey-Omura for message transmission

Alice

Bob

1. Encryption (1)

$$C_1 = M^{e_A} \bmod p$$

2. Encryption(2)

$$\begin{aligned} C_2 &= {C_1}^{e_B} \\ &= M^{e_A e_B} \bmod p \end{aligned}$$

3. Encryption (3)

$$\begin{aligned} C_3 &= {C_2}^{d_A} \\ &= (M^{e_A e_B})^{d_A} \\ &= M^{e_B} \bmod p \end{aligned}$$

Massey-Omura for message transmission

Alice

Bob

1. Encryption (1)

$$C_1 = M^{e_A} \bmod p$$

2. Encryption(2)

$$\begin{aligned} C_2 &= {C_1}^{e_B} \\ &= M^{e_A e_B} \bmod p \end{aligned}$$

3. Encryption (3)

$$\begin{aligned} C_3 &= {C_2}^{d_A} \\ &= (M^{e_A e_B})^{d_A} \\ &= M^{e_B} \bmod p \end{aligned}$$

4. Decryption

$$\begin{aligned} M &= {C_3}^{d_B} \\ &= M^{e_B d_B} \bmod p \end{aligned}$$