Asymmetric cryptography

Cryptographic Engineering

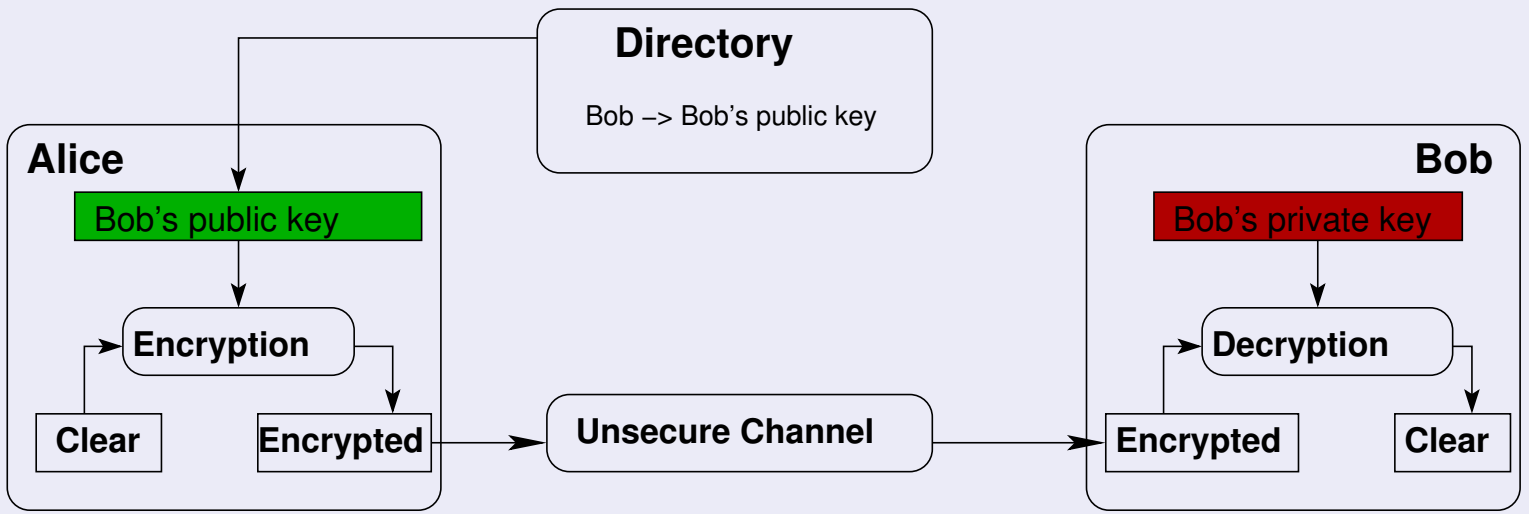
Clément Pernet

M2 Cyber Security,
UFR-IM^{2}AG, Univ. Grenoble-Alpes
ENSIMAG, Grenoble INP

Introduction

Asymmetric role of the encryption/decryption keys

Public key cryptography



Analogy of the **mailbox:**

- Anyone can write to Bob
- Only Bob can read the mail

Outline

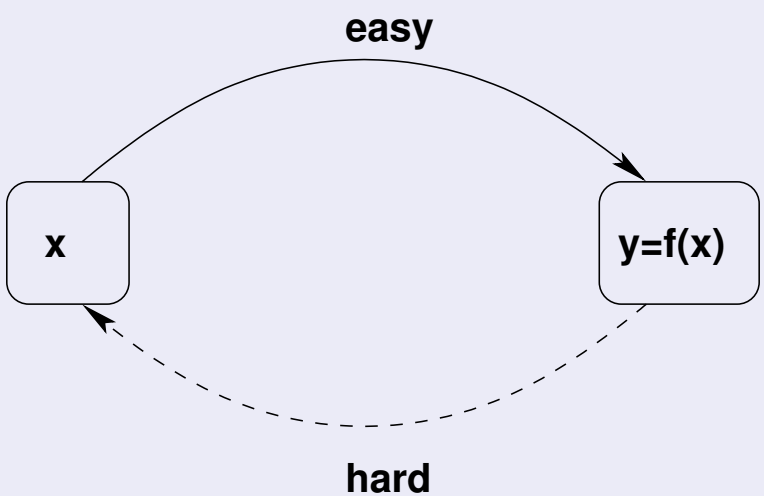
One way functions

- The private key K_d is fully determined by the public key K_e
  $\Rightarrow$no randomness involved, no statistical analysis
- But requires a notion of *difficulty of computation*

One way functions

- The private key K_d is fully determined by the public key K_e
  $\Rightarrow$no randomness involved, no statistical analysis
- But requires a notion of *difficulty of computation*

One way function

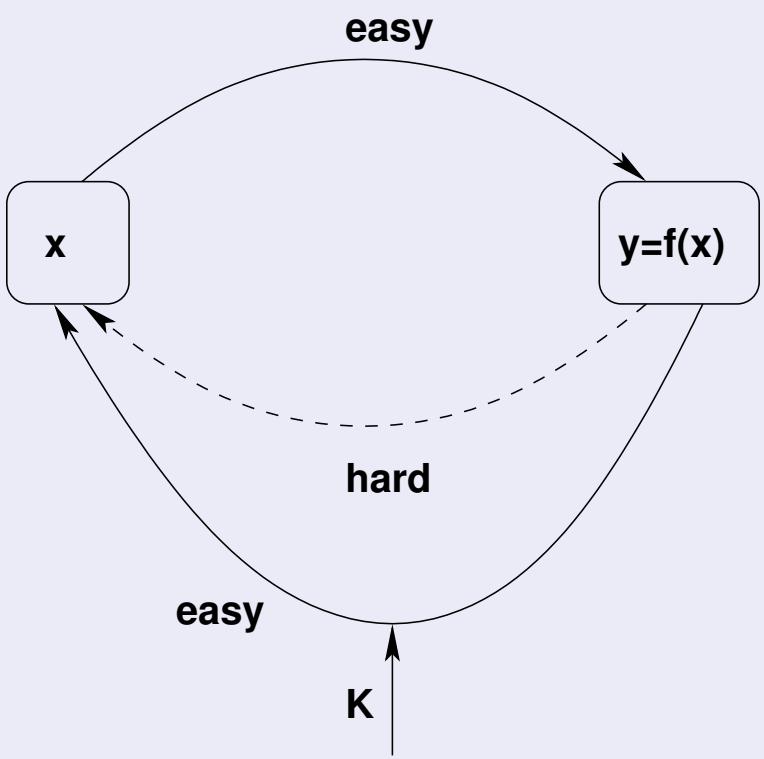


- easy to compute in one way
- hard to invert (other way) ...

One way functions

- The private key K_d is fully determined by the public key K_e
  $\Rightarrow$no randomness involved, no statistical analysis
- But requires a notion of *difficulty of computation*

One way function



- easy to compute in one way
- hard to invert (other way) ...
- ... unless some trapdoor is known

Outline

Outline

A one way function: the exponentiation

Problem

Data: $x \in G$ *a group* , $k \in \{1..|G|\}$
Result: $y = x^k$

begin
  $y = 1$;
  for $i = 1 \ldots k$ **do**
    $y = y \times x$;

$\Rightarrow O(k) = O(|G|)$ operations over G

A one way function: the exponentiation

Problem

Data: $x \in G$ *a group* , $k \in \{1..|G|\}$
Result: $y = x^k$

begin
  $y = 1$;
  for $i = 1 \ldots k$ **do**
    $y = y \times x$;

$\Rightarrow O(k) = O(|G|)$ operations over G

begin
  if *k=0* **then**
    return 1
  Compute recursively
  $z = x^{\lfloor k/2 \rfloor}$;
  if k *is even* **then**
    return z^2
  else
    return $z^2 \times x$

$\Rightarrow O(\log_2 k) = O(\log_2 |G|)$ operations over G

A one way function: the exponentiation

Problem

Data: $x \in G$ *a group* , $k \in \{1..|G|\}$
Result: $y = x^k$

begin
  if *k=0* **then**
    return 1
  Compute recursively $z = x^{\lfloor k/2 \rfloor}$;
  if k *is even* **then**
    return z^2
  else
    return $z^2 \times x$

begin
  $y = 1$;
  for $i = 1 \ldots k$ **do**
    $y = y \times x$;

$\Rightarrow O(k) = O(|G|)$ operations over G

begin
  Let $k = [k_0, \ldots, k_{\log_2 k}]$ be the binary representation of k;
  $h = x, y = 1$;
  for $i = 1 \ldots \log_2 k$ **do**
    if $k_i = 1$ **then**
      $y = y \times h$
    $h = h^2$;
  return y

$\Rightarrow O(\log_2 k) = O(\log_2 |G|)$ operations over G

Reciprocal: the discrete logarithm

Problem

From $y, x \in G$*, compute* k *such that* $y = x^k$

- Enumerate every x^i, for $i = 1 \ldots n \quad \Rightarrow O(|G|)$
- Improvement: BabyStep/GiantStep, Pollard $\quad \Rightarrow O\left(\sqrt{|G|}\right)$
- No known algorithm with polynomial time in $\log |G|$

One way function

- Exp_x: easy (polynomial time complexity)
- $\log_x$: hard (no known polynomial time algorithm)

Instances of groups

- Multiplicative groups of finite rings: $(\mathbb{Z}/n\mathbb{Z})^*$, $(\mathbb{F}_q\ )^*$

1

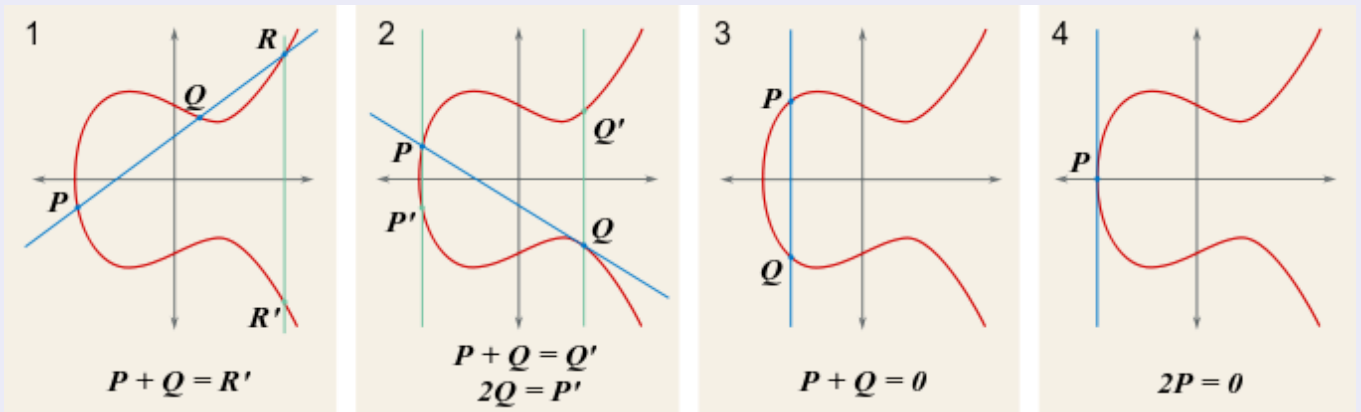
Instances of groups

- Multiplicative groups of finite rings: $(\mathbb{Z}/n\mathbb{Z})^*$, $(\mathbb{F}_q\ )^*$
- Group of points on an elliptic curve.

Elliptic curve arithmetic

$(E) : y^2 = ax^3 + b \mod p$

- Additive group structure for the points on (E).



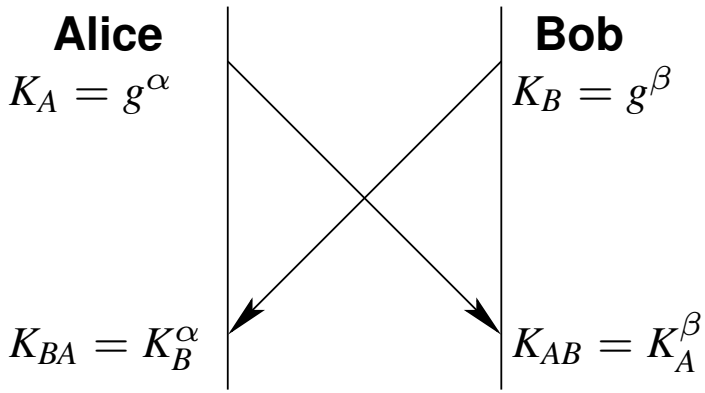
[1]

- $(p, k) \longrightarrow kP = \underbrace{P + \cdots + P}_{k \text{ times}}$ is easy
- $(P, Y) \longrightarrow k$ s.t. $Y = kP$ is hard.

[1] CC-BY-SA-3.0, SuperManu at English Wikipedia

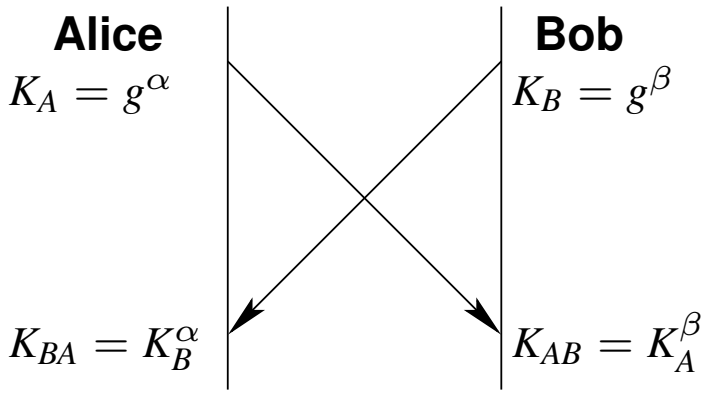
Diffie Hellman key exchange protocol

1. Choose publicly a group G and a generator g
2. Alice:
   - chooses a secret α
   - computes $K_A = g^{\alpha}$
   - sends K_A to Bob
3. Bob:
   - chooses a secret β
   - computes $K_B = g^{\beta}$
   - sends K_B to Alice



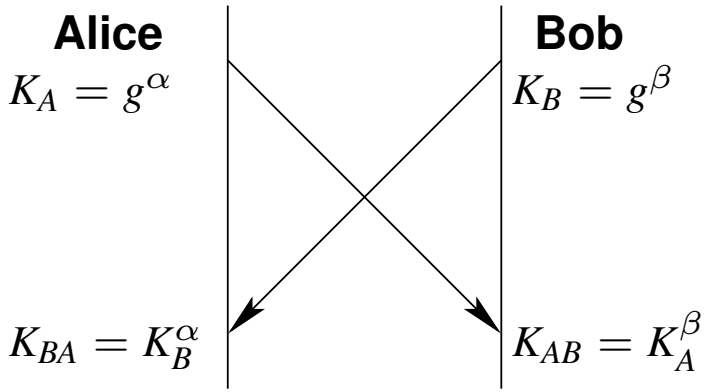
Diffie Hellman key exchange protocol

1. Choose publicly a group G and a generator g
2. Alice:
   - chooses a secret α
   - computes $K_A = g^{\alpha}$
   - sends K_A to Bob
   - computes $K_{BA} = K_B^{\alpha}$
3. Bob:
   - chooses a secret β
   - computes $K_B = g^{\beta}$
   - sends K_B to Alice
   - computes $K_{AB} = K_A^{\beta}$



Diffie Hellman key exchange protocol

1. Choose publicly a group G and a generator g
2. Alice:
   - chooses a secret α
   - computes $K_A = g^{\alpha}$
   - sends K_A to Bob
   - computes $K_{BA} = K_B^{\alpha}$
3. Bob:
   - chooses a secret β
   - computes $K_B = g^{\beta}$
   - sends K_B to Alice
   - computes $K_{AB} = K_A^{\beta}$



Property

$K_{AB} = K_{BA} = g^{\alpha\beta}$

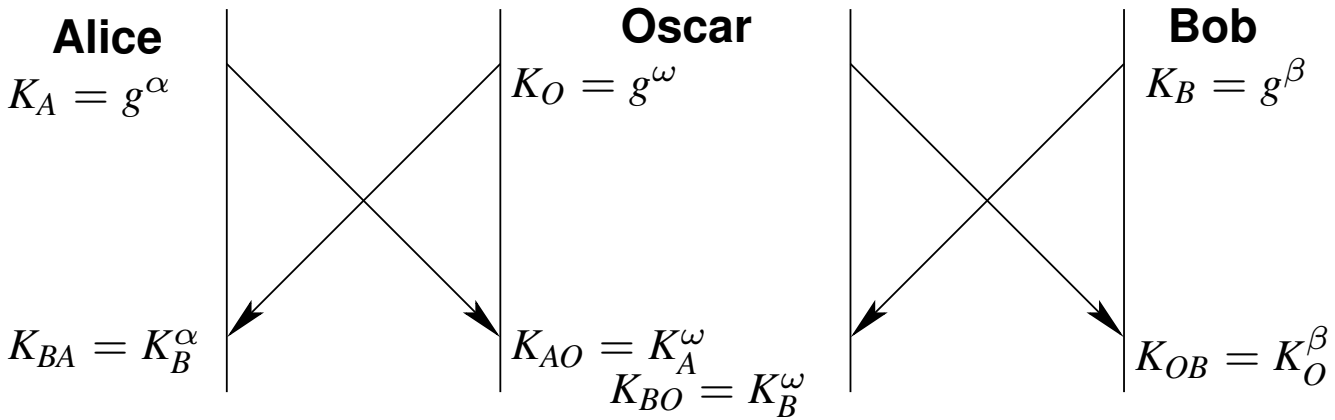
$\Rightarrow$sharing a key without ever sending α, β.

Security of the Diffie-Hellmann protocol

- From g, K_A, K_B, Oscar needs to compute a discrete logarithm in order to know α, β.

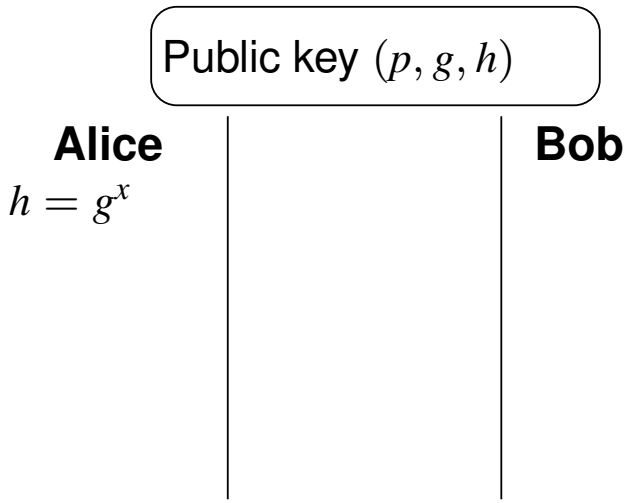
Security of the Diffie-Hellmann protocol

- From g, K_A, K_B, Oscar needs to compute a discrete logarithm in order to know α, β.
- Weakness : *Man in the middle attack*



Public key cipher: El Gamal

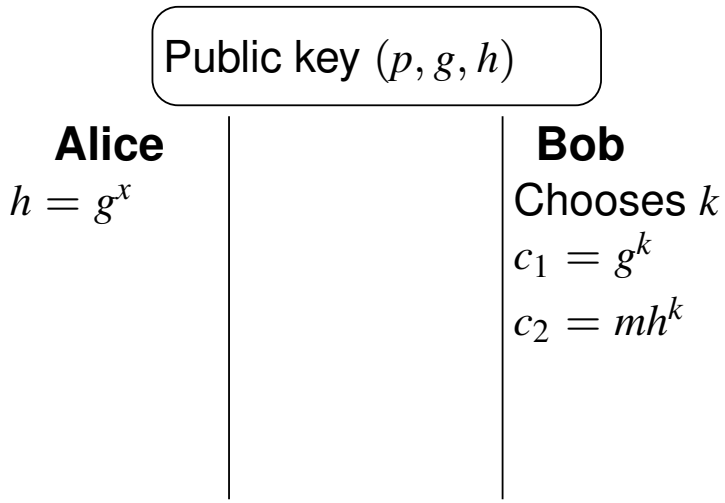
1. Alice:
   - chooses G and a generator g
   - chooses a secret x
   - computes $h = g^x$
   - Public key: (g, h)



Public key cipher: El Gamal

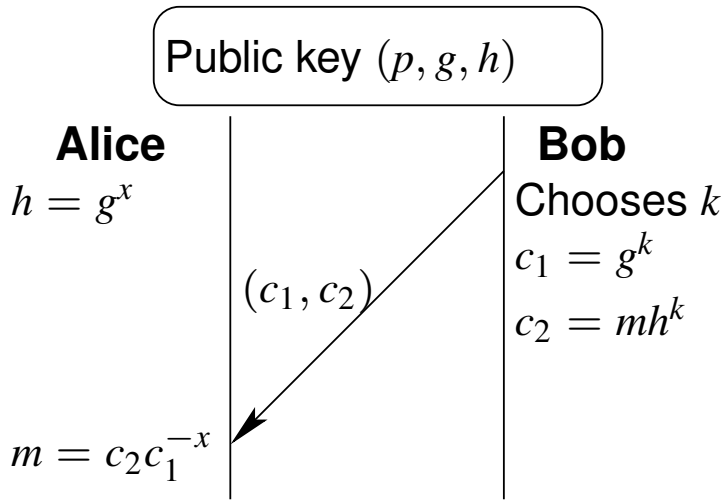
1. Alice:
   - chooses G and a generator g
   - chooses a secret x
   - computes $h = g^x$
   - Public key: (g, h)

2. Bob: wants to send the message $m \in G$
   - chooses k randomly
   - computes $c_1 = g^k$
   - computes $c_2 = mh^k$
   - sends (c_1, c_2) to Alice



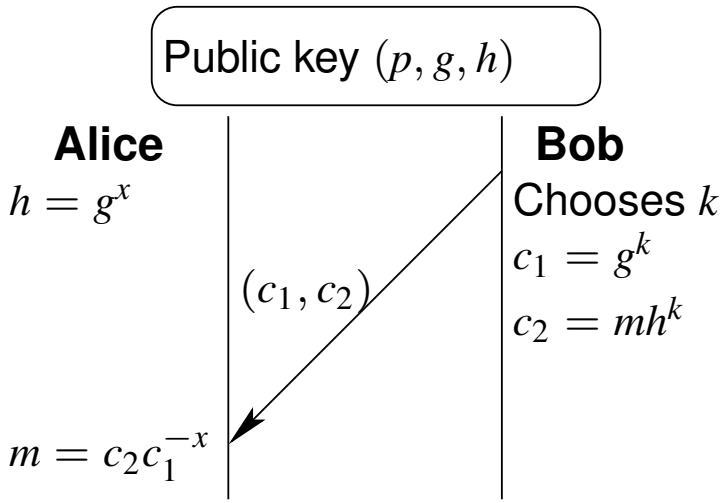
Public key cipher: El Gamal

1. Alice:
   - chooses G and a generator g
   - chooses a secret x
   - computes $h = g^x$
   - Public key: (g, h)
   - computes $c_2 c_1^{-x}$
2. Bob: wants to send the message $m \in G$
   - chooses k randomly
   - computes $c_1 = g^k$
   - computes $c_2 = mh^k$
   - sends (c_1, c_2) to Alice



Public key cipher: El Gamal

1. Alice:
   - chooses G and a generator g
   - chooses a secret x
   - computes $h = g^x$
   - Public key: (g, h)
   - computes $c_2 c_1^{-x}$
2. Bob: wants to send the message $m \in G$
   - chooses k randomly
   - computes $c_1 = g^k$
   - computes $c_2 = mh^k$
   - sends (c_1, c_2) to Alice



$$c_2 c_1^{-x} = mh^k g^{-kx} = mg^{kx} g^{-kx} = m$$

Outline

The RSA code: *Rivest Shamir Adelman*

Theorem (Rivest Shamir Adelman 78)

Let p, q be primes and $n = pq$. Then

$$\forall a \in \mathbb{Z}/n\mathbb{Z}, \forall k \in \mathbb{Z} \quad a^{1+k(p-1)(q-1)} = a \mod n$$

Proof:

- If $a = 0 \mod p$, then $a^{k\varphi(n)+1} = 0 = a \mod p$
- Else, a is invertible modulo p and $a^{p-1} = 1 \mod p$. Thus

$$a^{k(p-1)(q-1)+1} = a \mod p$$

- Similarly $\mod q$: $a^{k(p-1)(q-1)+1} = a \mod q$
- Chinese Remainder Theorem: $a^{k(p-1)(q-1)+1} = a \mod pq$

The RSA code

- Let $e < \varphi(n)$ coprime with $\varphi(n)$: $\Rightarrow \exists d$ s.t. $ed = 1 \mod \varphi(n)$

$$ed = 1 + k\varphi(n)$$

The RSA code:

- Public key: (e, n), encoding: $c = m^e \mod n$
- Private key: (d, n), decoding: $c^d \mod n$

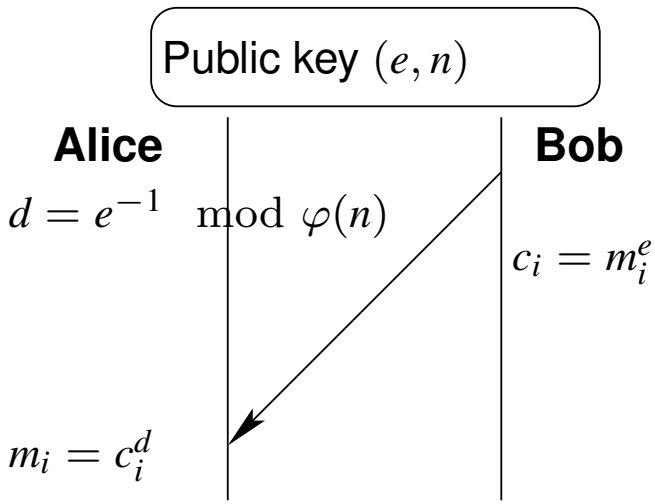
RSA Theorem: $c^d = m^{ed} = m^{1+k\varphi(n)} = m \mod n$

Warning:

- $\varphi(n)$ must remain private (since inverting $e \mod \varphi(n)$ is easy)
- p, q must remain private

The RSA code

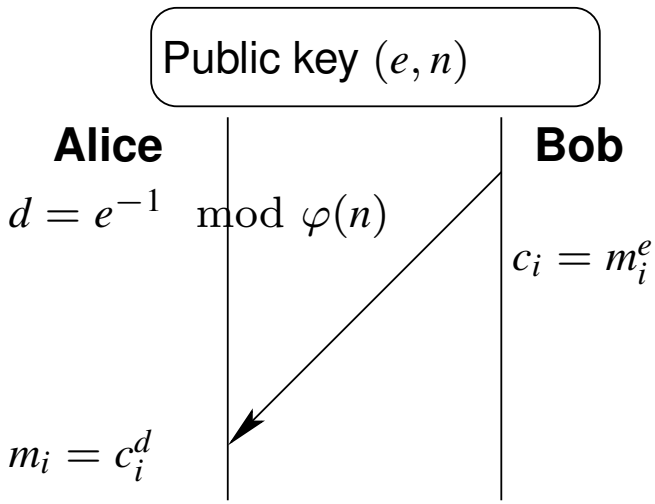
1. Alice:
   - Chooses p, q and computes $n = pq$
   - Chooses e prime with $\varphi(n)$
   - Computes $d = e^{-1} \mod \varphi(n)$
   - Public key: (e, n)



The RSA code

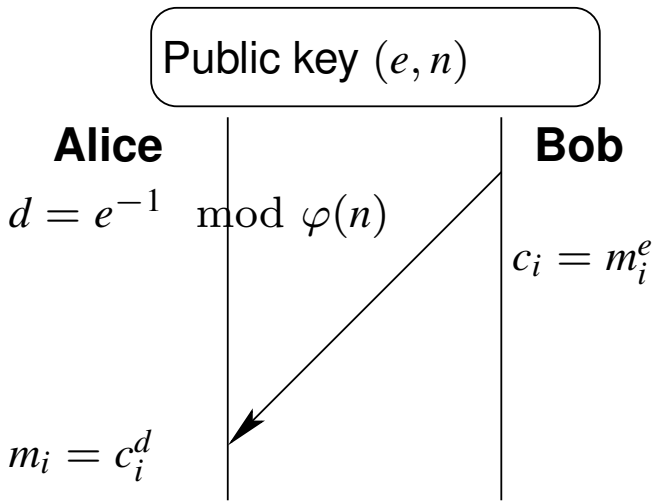
1. Alice:
   - Chooses p, q and computes $n = pq$
   - Chooses e prime with $\varphi(n)$
   - Computes $d = e^{-1} \mod \varphi(n)$
   - Public key: (e, n)

2. Bob: wants to send the message m
   - splits $m = (m_1, \ldots, m_k)$ st. $m_i < n$
   - For each m_i
   - sends $c_i = m_i^e \mod n$ to Alice



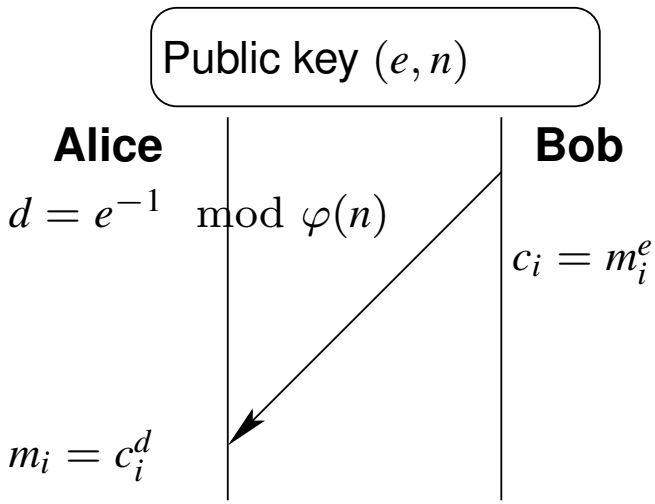
The RSA code

1. Alice:
   - Chooses p, q and computes $n = pq$
   - Chooses e prime with $\varphi(n)$
   - Computes $d = e^{-1} \mod \varphi(n)$
   - Public key: (e, n)
   - Computes $m_i = c_i^d \mod n$
2. Bob: wants to send the message m
   - splits $m = (m_1, \ldots, m_k)$ st. $m_i < n$
   - For each m_i
   - sends $c_i = m_i^e \mod n$ to Alice



The RSA code

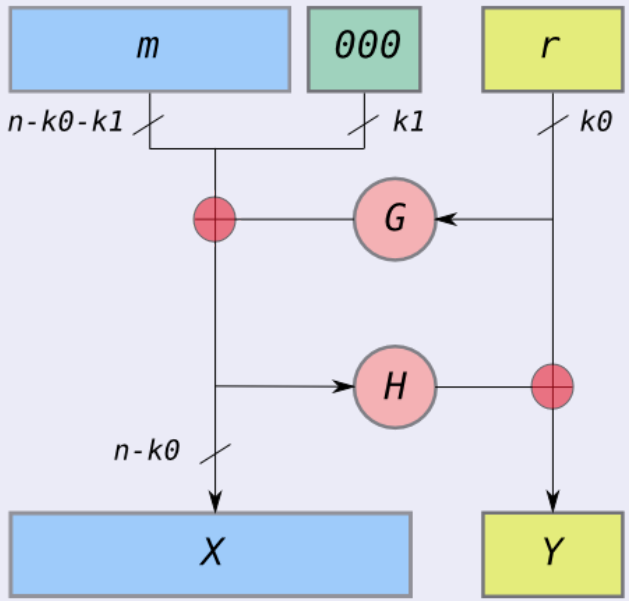
1. Alice:
   - Chooses p, q and computes $n = pq$
   - Chooses e prime with $\varphi(n)$
   - Computes $d = e^{-1} \mod \varphi(n)$
   - Public key: (e, n)
   - Computes $m_i = c_i^d \mod n$
2. Bob: wants to send the message m
   - splits $m = (m_1, \ldots, m_k)$ st. $m_i < n$
   - For each m_i
   - sends $c_i = m_i^e \mod n$ to Alice



Not semantically secure: use randomized padding on each block

Randomized Padding

The OAEP padding[2]



- Works as a Feistel function: inverted without inverting G and H.

[2] CC-BY-SA-3.0, Ozga at English Wikipedia

RSA algorithmic

- Computing p, q primes $\Rightarrow$prime number generator
- Computing $n, \varphi(n) = (p-1)(q-1)$ $\Rightarrow$Int. multiplication $\tilde{O}(\log n)$
- Computing $d = e^{-1} \mod \varphi(n)$ $\Rightarrow$Ext. Euclidean alg. $\tilde{O}(\log n)$
- Encoding: $c = m^e \mod n$ $\Rightarrow$Modular exponentiation $\tilde{O}\left(\log^2 n\right)$
- Decoding: $m = c^d \mod n$ $\Rightarrow$Modular exponentiation $\tilde{O}\left(\log^2 n\right)$

Yet recovering d from e, n is closely related to integer factorisation, a difficult problem for which no polynomial time algorithm is known. (see lecture on reductions).

Chinese Remainder based RSA

Exercise

1. Write a modular exponentiation algorithm computing $m^d \mod n$ based on the Chinese remainder theorem applied to $n = pq$.
2. Can it be applied to both the encryption and decryption of RSA?
3. Compute its time complexity (assuming classic integer mutliplication). How does it compare to the standard modular exponentiation?

CRT based RSA decryption

$$m, \qquad d \qquad \longrightarrow \quad m^d \mod n$$

CRT based RSA decryption

$$
\begin{array}{lll}
m, & d & \longrightarrow \quad m^d \mod n \\
\text{Reduction} \downarrow & \text{Reduction} \downarrow & \\
m_p = m \mod p, & d_p = d \mod p-1 & \\
m_q = m \mod q, & d_q = d \mod q-1 &
\end{array}
$$

CRT based RSA decryption

$$
\begin{array}{llll}
m, & d & \longrightarrow & m^d \mod n \\
\text{Reduction} \downarrow & \text{Reduction} \downarrow & & \\
m_p = m \mod p, & d_p = d \mod p-1 & \xrightarrow{Exp} & m_p^{d_p} \mod p \\
m_q = m \mod q, & d_q = d \mod q-1 & \xrightarrow{Exp} & m_q^{d_q} \mod q
\end{array}
$$

CRT based RSA decryption

$$
\begin{array}{lll}
m, & d & \longrightarrow \quad m^d \mod n \\
\text{Reduction} \downarrow & \text{Reduction} \downarrow & \qquad\quad CRT \uparrow \\
m_p = m \mod p, & d_p = d \mod p-1 & \xrightarrow{Exp} \quad m_p^{d_p} \mod p \\
m_q = m \mod q, & d_q = d \mod q-1 & \xrightarrow{Exp} \quad m_q^{d_q} \mod q
\end{array}
$$

CRT based RSA decryption

$$
\begin{array}{lll}
m, & d & \longrightarrow \quad m^d \mod n \\
\text{Reduction} \downarrow & \text{Reduction} \downarrow & \qquad\quad CRT \uparrow \\
m_p = m \mod p, & d_p = d \mod p-1 & \xrightarrow{Exp} \quad m_p^{d_p} \mod p \\
m_q = m \mod q, & d_q = d \mod q-1 & \xrightarrow{Exp} \quad m_q^{d_q} \mod q
\end{array}
$$

Algorithm(s)

$$
\begin{aligned}
c \quad = \quad & c_p q (q^{-1} \mod p) + \\
& c_q p (p^{-1} \mod q) \mod n
\end{aligned}
$$

CRT based RSA decryption

$$
\begin{array}{lll}
m, & d & \longrightarrow \quad m^d \mod n \\
\text{Reduction} \downarrow & \text{Reduction} \downarrow & \qquad\quad CRT \uparrow \\
m_p = m \mod p, & d_p = d \mod p-1 & \xrightarrow{Exp} \quad m_p^{d_p} \mod p \\
m_q = m \mod q, & d_q = d \mod q-1 & \xrightarrow{Exp} \quad m_q^{d_q} \mod q
\end{array}
$$

Algorithm(s)

$$
\begin{aligned}
c \quad = \quad & c_p q(q^{-1} \mod p) + \\
& c_q p(p^{-1} \mod q) \mod n
\end{aligned}
$$

1. Precompute:
   - 1.1 $a \leftarrow q(q^{-1} \mod p) \mod n$
   - 1.2 $b \leftarrow p(p^{-1} \mod q) \mod n$
2. $m_p \leftarrow m \mod p$
3. $m_q \leftarrow m \mod q$
4. $c_p \leftarrow \texttt{ExpMod}(m_p, d_p, p)$
5. $c_q \leftarrow \texttt{ExpMod}(m_q, d_q, q)$
6. $c \leftarrow ac_p + bc_q$

CRT based RSA decryption

$$
\begin{array}{lll}
m, & d & \longrightarrow \quad m^d \mod n \\
\text{Reduction} \downarrow & \text{Reduction} \downarrow & \quad CRT \uparrow \\
m_p = m \mod p, & d_p = d \mod p-1 & \xrightarrow{Exp} \quad m_p^{d_p} \mod p \\
m_q = m \mod q, & d_q = d \mod q-1 & \xrightarrow{Exp} \quad m_q^{d_q} \mod q
\end{array}
$$

Algorithm(s)

$$
\begin{aligned}
c &= c_p q(q^{-1} \mod p) + \\
&\quad c_q p(p^{-1} \mod q) \mod n
\end{aligned}
\qquad
\begin{aligned}
c &= p(p^{-1} \mod q)(c_q - c_p) + \\
&\quad c_p \mod n
\end{aligned}
$$

1. Precompute:
   - 1.1 $a \leftarrow q(q^{-1} \mod p) \mod n$
   - 1.2 $b \leftarrow p(p^{-1} \mod q) \mod n$
2. $m_p \leftarrow m \mod p$
3. $m_q \leftarrow m \mod q$
4. $c_p \leftarrow \texttt{ExpMod}(m_p, d_p, p)$
5. $c_q \leftarrow \texttt{ExpMod}(m_q, d_q, q)$
6. $c \leftarrow ac_p + bc_q$

CRT based RSA decryption

$$
\begin{array}{lll}
m, & d & \longrightarrow \quad m^d \mod n \\
\text{Reduction} \downarrow & \text{Reduction} \downarrow & \qquad\quad CRT \uparrow \\
m_p = m \mod p, & d_p = d \mod p-1 & \xrightarrow{Exp} \quad m_p^{d_p} \mod p \\
m_q = m \mod q, & d_q = d \mod q-1 & \xrightarrow{Exp} \quad m_q^{d_q} \mod q
\end{array}
$$

Algorithm(s)

$$c = c_p q(q^{-1} \mod p) + c_q p(p^{-1} \mod q) \mod n$$

1. Precompute:
   - 1.1 $a \leftarrow q(q^{-1} \mod p) \mod n$
   - 1.2 $b \leftarrow p(p^{-1} \mod q) \mod n$
2. $m_p \leftarrow m \mod p$
3. $m_q \leftarrow m \mod q$
4. $c_p \leftarrow \texttt{ExpMod}(m_p, d_p, p)$
5. $c_q \leftarrow \texttt{ExpMod}(m_q, d_q, q)$
6. $c \leftarrow ac_p + bc_q$

$$c = p(p^{-1} \mod q)(c_q - c_p) + c_p \mod n$$

1. Precompute:
   - 1.1 $a \leftarrow p(p^{-1} \mod q) \mod n$
2. $m_p \leftarrow m \mod p$
3. $m_q \leftarrow m \mod q$
4. $c_p \leftarrow \texttt{ExpMod}(m_p, d_p, p)$
5. $c_q \leftarrow \texttt{ExpMod}(m_q, d_q, q)$
6. $c \leftarrow m_p + a(c_q - c_p)$

CRT based RSA decryption

- Require the knowledge of p and q $\Rightarrow$only for decryption

CRT based RSA decryption

- Require the knowledge of p and q $\Rightarrow$only for decryption

Reminder: Modular exponentiation costs: $\approx 2 \log e$ Mult $= 2 \log^3 n$

- Cost:

$$\begin{aligned} T(n) &= 2 \log^3 p + 2 \log^3 q + O(\log^2 n) \\ &= 4 \log^3 \sqrt{n} + O(\log^2 n) \\ &= \frac{1}{2} \log^3 (n) + O(\log^2 n) \end{aligned}$$

CRT based RSA decryption

- Require the knowledge of p and q $\Rightarrow$only for decryption

Reminder: Modular exponentiation costs: $\approx 2 \log e$ Mult $= 2 \log^3 n$

- Cost:

$$\begin{aligned} T(n) &= 2 \log^3 p + 2 \log^3 q + O(\log^2 n) \\ &= 4 \log^3 \sqrt{n} + O(\log^2 n) \\ &= \frac{1}{2} \log^3(n) + O(\log^2 n) \end{aligned}$$

vs $2 \log^3 n$ for the standard modular exponentiation
$\Rightarrow$near $4\times$ speed-up

Attacks by fault injection on CRT-RSA

- A chip performs authentication by applying a the RSA private key to a challenge.
- Using a laser, one can flip a bit in a register of the chip at a given time, thus generating a fault.

Exercise

Suppose one has succesfully managed to generate a fault during one of the two modular exponentiations in the RSA-CRT encryption algorithm.

1. Explain how one can recover the private factors p and q from this faulty signature and a correct one.
2. Explore which part of the algorithm is sensitive to the same attack

Fault injection on CRT-RSA

$$c = ac_p + bc_q \mod n, \text{ with } a = q(q^{-1} \mod p), b = p(p^{-1} \mod q)$$

1. If only one of c_p or c_q is incorrect (say c_p) then
   - the faulty c' still verifies $c' = c_q \mod q$.
   - hence $c' - c = 0 \mod q$ and q divides $c' - c$
   - moreover $c' - c = a(c'_p - c_p) + \lambda n$.
   - but q divides a ($a = q(q^{-1} \mod p)$)
   - Therefore q divides $c' - c$ but p does not.
   - Thus $\gcd(c' - c, n) = q$.

Fault injection on CRT-RSA

$$c = ac_p + bc_q \mod n, \text{ with } a = q(q^{-1} \mod p), b = p(p^{-1} \mod q)$$

1. If only one of c_p or c_q is incorrect (say c_p) then
   - the faulty c' still verifies $c' = c_q \mod q$.
   - hence $c' - c = 0 \mod q$ and q divides $c' - c$
   - moreover $c' - c = a(c'_p - c_p) + \lambda n$.
   - but q divides a ($a = q(q^{-1} \mod p)$)
   - Therefore q divides $c' - c$ but p does not.
   - Thus $\gcd(c' - c, n) = q$.
2. Faults on $m_p \leftarrow m \mod p$ or $m_p \leftarrow m \mod p$ or loading m to register for these reductions also lead to the same problem

Fault injection on CRT-RSA

$$c = ac_p + bc_q \mod n, \text{ with } a = q(q^{-1} \mod p), b = p(p^{-1} \mod q)$$

1. If only one of c_p or c_q is incorrect (say c_p) then
   - the faulty c' still verifies $c' = c_q \mod q$.
   - hence $c' - c = 0 \mod q$ and q divides $c' - c$
   - moreover $c' - c = a(c'_p - c_p) + \lambda n$.
   - but q divides a ($a = q(q^{-1} \mod p)$)
   - Therefore q divides $c' - c$ but p does not.
   - Thus $\gcd(c' - c, n) = q$.
2. Faults on $m_p \leftarrow m \mod p$ or $m_p \leftarrow m \mod p$ or loading m to register for these reductions also lead to the same problem

Conter-measures

- physical protections
- perform the signature twice, and compare if they differ
- verification that $c^e \mod n = m$ before answering the challenge

Other attacks on RSA

Common modulus attack

Alice and Bob have RSA public keys (e_A, n_A) and (e_B, n_B) with the same modulus $n_A = n_B$. Suppose that e_A and e_B are co-prime.

- Explain how Eve can easily recover the plain text if she intercepts the two corresponding ciphertexts encrypted for Alice and for Bob.
- Conclusion?

Other attacks on RSA

Common modulus attack

Alice and Bob have RSA public keys (e_A, n_A) and (e_B, n_B) with the same modulus $n_A = n_B$. Suppose that e_A and e_B are co-prime.

- Explain how Eve can easily recover the plain text if she intercepts the two corresponding ciphertexts encrypted for Alice and for Bob.
- Conclusion?

Diffusion and common small exponent attack

Suppose Jack, William and Avrel have the RSA keys $(3, n_J)$,$(3, n_W)$,$(3, n_A)$ Joe sends the message m to them using their respective encryption keys.

- Explain how Lucky-luke can recover the plaintext m from these 3 ciphertexts.

  hint: computing $\lfloor x^{1/k} \rfloor$ can be achieved in $\tilde{O}\left(\log^2 x\right)$ bit operations using Newton's iteration.

Other attacks on RSA

Factorial attack

Let $B \in \mathbb{Z}$ such that $(p-1)$ divides $B!$.

1. Show that for any prime factor p_1 of $p-1$, $p_1 \leq B$.
2. For $a \in \mathbb{Z}$ show that $a^{B!} = 1 \mod p$.
3. Let $A = a^{B!} \mod n$. Show that p divides $A - 1$.
4. How much does the computation of A costs (as a function of n and B)
5. How can one try to factor n using the above results? Under which condition does it work?
6. Contermeasure?

Other attacks on RSA

Decryption variant

Let's consider the following variant on RSA: for $n = pq$ with p and q prime numbers

- let $\mu(n) = \frac{(p-1)(q-1)}{\delta}$ where $\delta = \gcd(p-1, q-1)$.
- The public key is still a pair (e, n) where e is co-prime with $(p-1)(q-1)$
- let $d' = e^{-1} \mod \mu(n)$ so that the private key now becomes (d', n)
- encryption is $E(x) = x^e \mod n$
- decryption is $D(x) = x^{d'} \mod n$

1. Explain why the decryption still works
2. compute the private keys d and d' for $p = 19$ and $q = 31$.
3. What do you think of this variant in terms of efficiency and security?

Outline

Attacking the hard problems

- Integer factorization
- Discrete logarithm
  - over a finite field
  - over elliptic curves

Motivation

Find the best algorithms/implementation attacking those problems to

- estimate the difficulty of practical attacks
- relate the key size to the security parameters
- anticipate the evolution of key sizes

Outline

Applied to integer factorization

Problem

1. *Decompose* n *into* $n = \prod_{i=1}^{k} p_i^{\alpha_i}$, *where* p_i *are prime*
2. *Weaker formulation: find* $p > 1$ *such that* $p|n$
   $\Rightarrow$*recursive computation of the factors of* n/p.

Applied to integer factorization

Problem

1. *Decompose n into $n = \prod_{i=1}^{k} p_i^{\alpha_i}$, where p_i are prime*
2. *Weaker formulation: find $p > 1$ such that $p|n$*
   $\Rightarrow$*recursive computation of the factors of n/p.*

Principle: exploiting a collision

Suppose we found X and Y such that

$$\begin{cases} X & = & Y & \mod p \\ X & \neq & Y & \mod n \end{cases}$$

p divides $X - Y$ but n does not.

$$\Rightarrow \text{GCD}(X - Y, n) = p$$

How to find X and Y ?

Applied to integer factorization

Problem

1. *Decompose* n *into* $n = \prod_{i=1}^{k} p_i^{\alpha_i}$, *where* p_i *are prime*
2. *Weaker formulation: find* $p > 1$ *such that* $p|n$
   $\Rightarrow$*recursive computation of the factors of* n/p.

Principle: exploiting a collision

Suppose we found X and Y such that

$$\begin{cases} X & = & Y & \mod p \\ X & \neq & Y & \mod n \end{cases}$$

p divides $X - Y$ but n does not.

$$\Rightarrow \text{GCD}(X - Y, n) = p$$

How to find X and Y ? $\Rightarrow$recurring sequences

Collision in a recurring sequence

Principle:

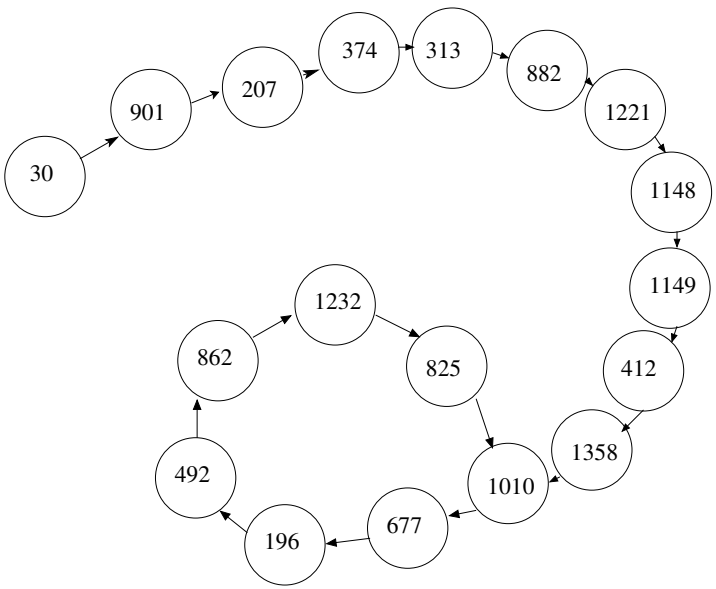
- Recurring sequence mod n: $\begin{cases} X_0 & = & \texttt{rand()}, \\ X_{i+1} & = & f(X_i) \mod n \end{cases}$
- Finite number of states: ultimately periodic

Collision in a recurring sequence

Principle:

- Recurring sequence mod n: $\begin{cases} X_0 & = & \texttt{rand()}, \\ X_{i+1} & = & f(X_i) \mod n \end{cases}$
- Finite number of states: ultimately periodic

  mod n: period λ_n s.t. $X_{t+\lambda_n} = X_t \mod n \; \forall t \geq \mu_n$



modulo $1517 = 41 \times 37$

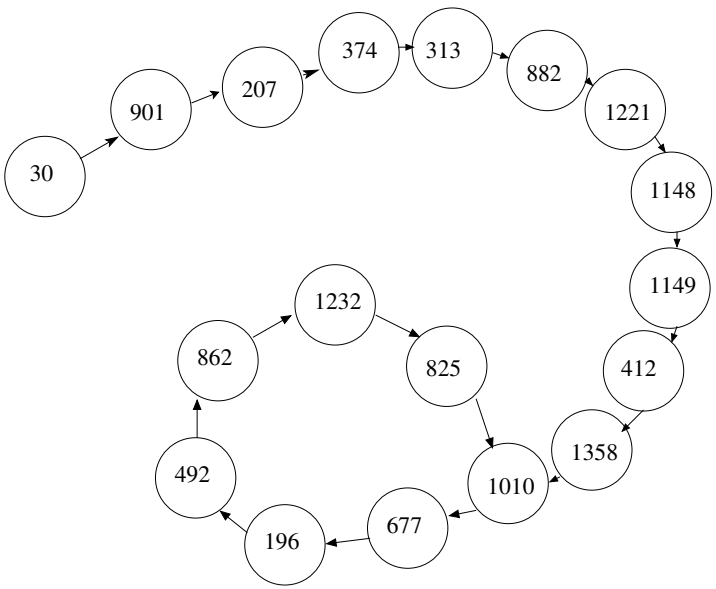
Collision in a recurring sequence

Principle:

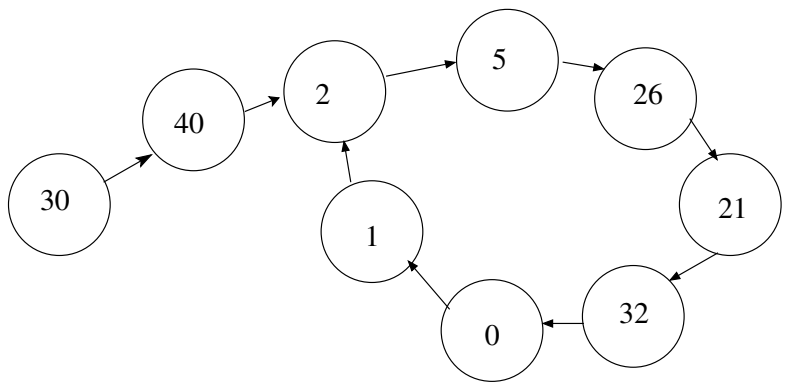
- Recurring sequence mod n: $\begin{cases} X_0 & = & \texttt{rand()}, \\ X_{i+1} & = & f(X_i) \mod n \end{cases}$
- Finite number of states: ultimately periodic

  mod n: period λ_n s.t. $X_{t+\lambda_n} = X_t \mod n \; \forall t \geq \mu_n$

  mod p: period $\lambda_p \leq \lambda_n$ s.t. $X_{t+\lambda_p} = X_t \mod p \; \forall t \geq \mu_p$



modulo $1517 = 41 \times 37$



modulo 41

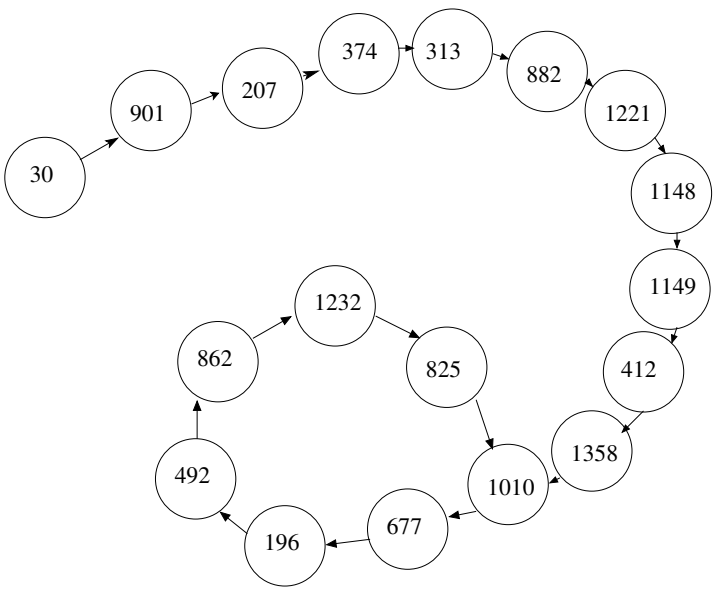
Collision in a recurring sequence

Principle:

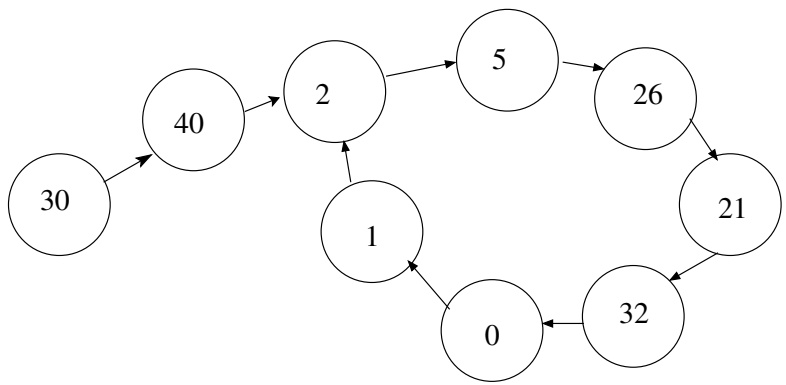
- Recurring sequence mod n: $\begin{cases} X_0 & = & \texttt{rand()}, \\ X_{i+1} & = & f(X_i) \mod n \end{cases}$
- Finite number of states: ultimately periodic

  mod n: period λ_n s.t. $X_{t+\lambda_n} = X_t \mod n \; \forall t \geq \mu_n$

  mod p: period $\lambda_p \leq \lambda_n$ s.t. $X_{t+\lambda_p} = X_t \mod p \; \forall t \geq \mu_p$
- GCD$((X_{t+k} - X_t), n) = p$ or pq $\Rightarrow$If $X_{t+k} - X_t \neq 0 \mod n$ $\Rightarrow$get p

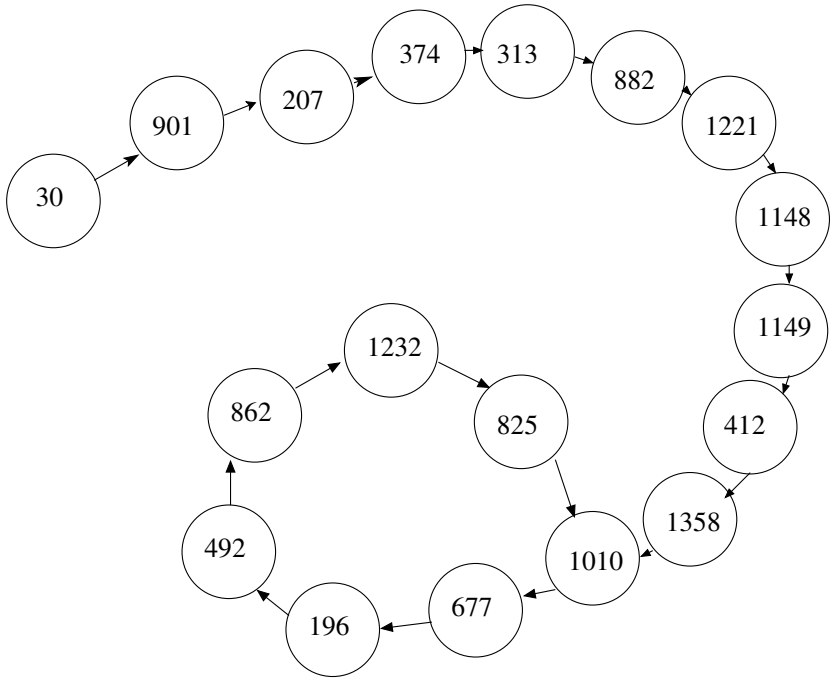


modulo $1517 = 41 \times 37$

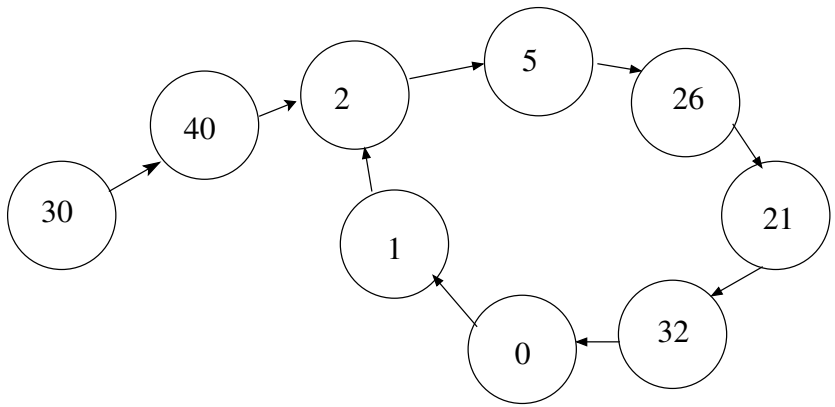


modulo 41

How to detect a collision?



modulo $1517 = 41 \times 37$



modulo 41

Exercise

1. Propose (an) algorithm(s) finding indices i and j of a collision: such that $X_i = X_j \mod p$.
2. What is its time and space complexity?

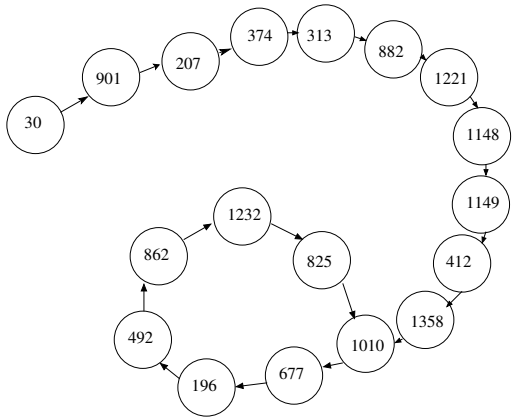
Floyd's cycle detection

Let $u_{i+1} = f(u_i)$ be the recurring sequence.

Tail length: μ

Period: λ

$$u_{j+\mu+\lambda} = u_{j+\mu} \ \forall j \geq 0.$$



Iterate through u_i with 2 paces:

- $X_i = u_i$ (by steps of 1)
- $Y_i = u_{2i}$ (by steps of 2)

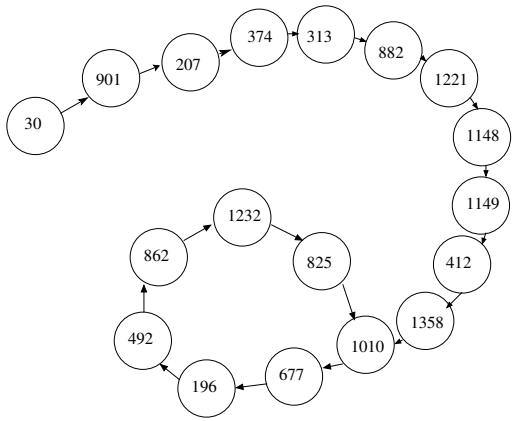
Floyd's cycle detection

Let $u_{i+1} = f(u_i)$ be the recurring sequence.

Tail length: μ

Period: λ

$$u_{j+\mu+\lambda} = u_{j+\mu} \ \forall j \geq 0.$$



Iterate through u_i with 2 paces:

- $X_i = u_i$ (by steps of 1)
- $Y_i = u_{2i}$ (by steps of 2)

Lemma

$$u_i = u_{2i} \Leftrightarrow i = k\lambda \geq \mu$$

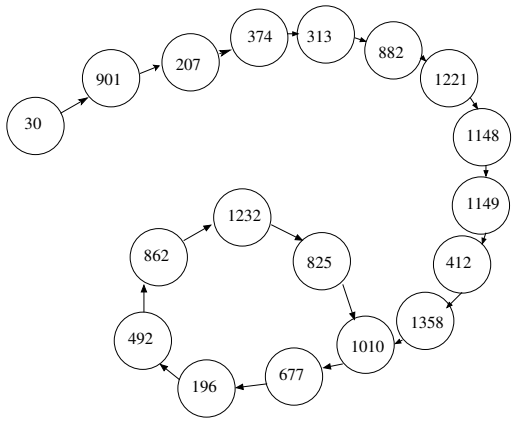
Floyd's cycle detection

Let $u_{i+1} = f(u_i)$ be the recurring sequence.

Tail length: μ

Period: λ

$$u_{j+\mu+\lambda} = u_{j+\mu} \ \forall j \geq 0.$$



Iterate through u_i with 2 paces:

- $X_i = u_i$ (by steps of 1)
- $Y_i = u_{2i}$ (by steps of 2)

Lemma

$$u_i = u_{2i} \Leftrightarrow i = k\lambda \geq \mu$$

In particular, the first such i is for $k = \lceil \frac{\mu}{\lambda} \rceil$ and satifies

$$\mu \leq i \leq \lambda + \mu$$

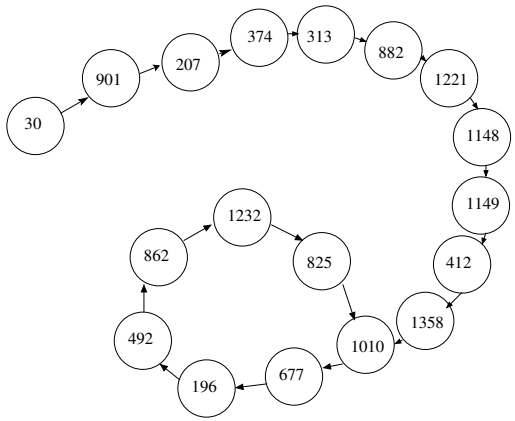
Floyd's cycle detection

Let $u_{i+1} = f(u_i)$ be the recurring sequence.

Tail length: μ

Period: λ

$$u_{j+\mu+\lambda} = u_{j+\mu} \ \forall j \geq 0.$$



Iterate through u_i with 2 paces:

- $X_i = u_i$ (by steps of 1)
- $Y_i = u_{2i}$ (by steps of 2)

Lemma

$$u_i = u_{2i} \Leftrightarrow i = k\lambda \geq \mu$$

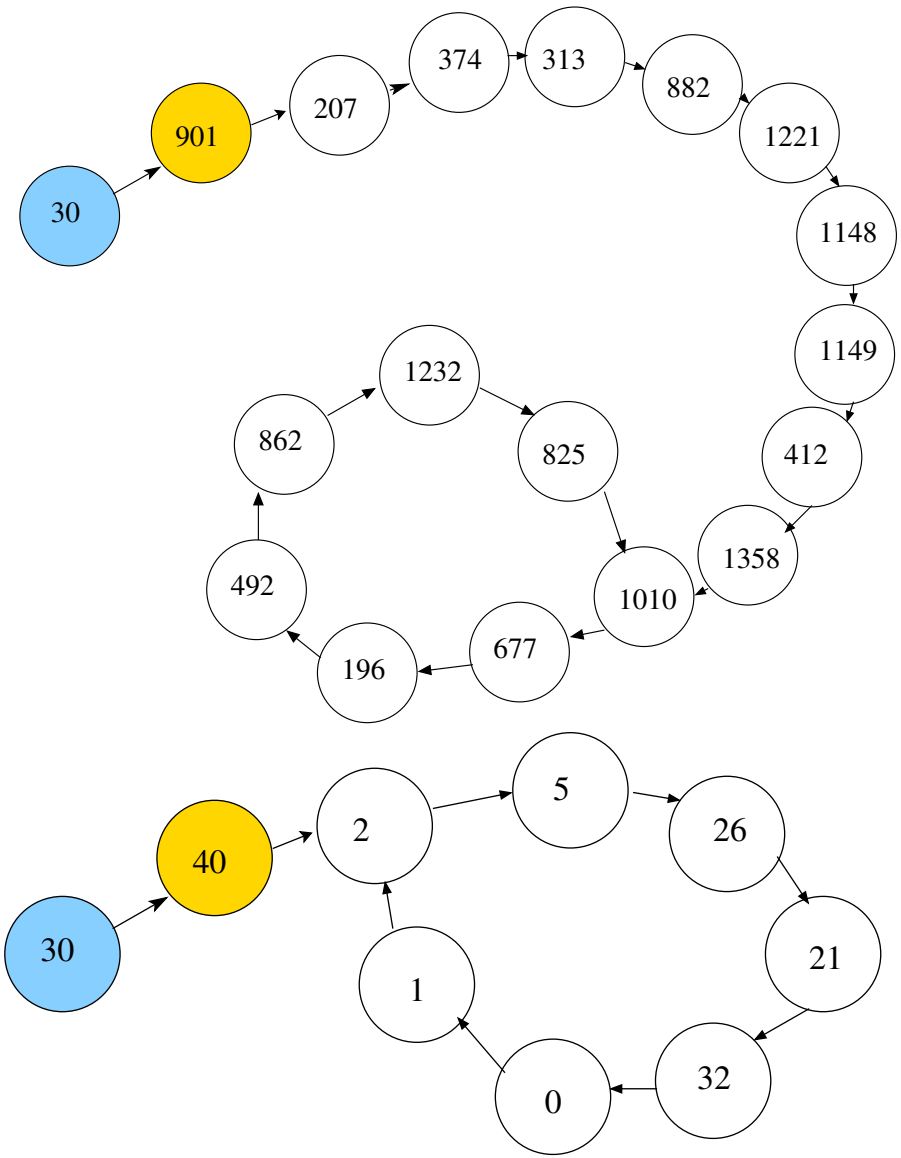
In particular, the first such i is for $k = \lceil \frac{\mu}{\lambda} \rceil$ and satifies

$$\mu \leq i \leq \lambda + \mu$$

Cost

$\lambda + \mu \leq p$ hence $\Theta(\sqrt{n})$ iterations.

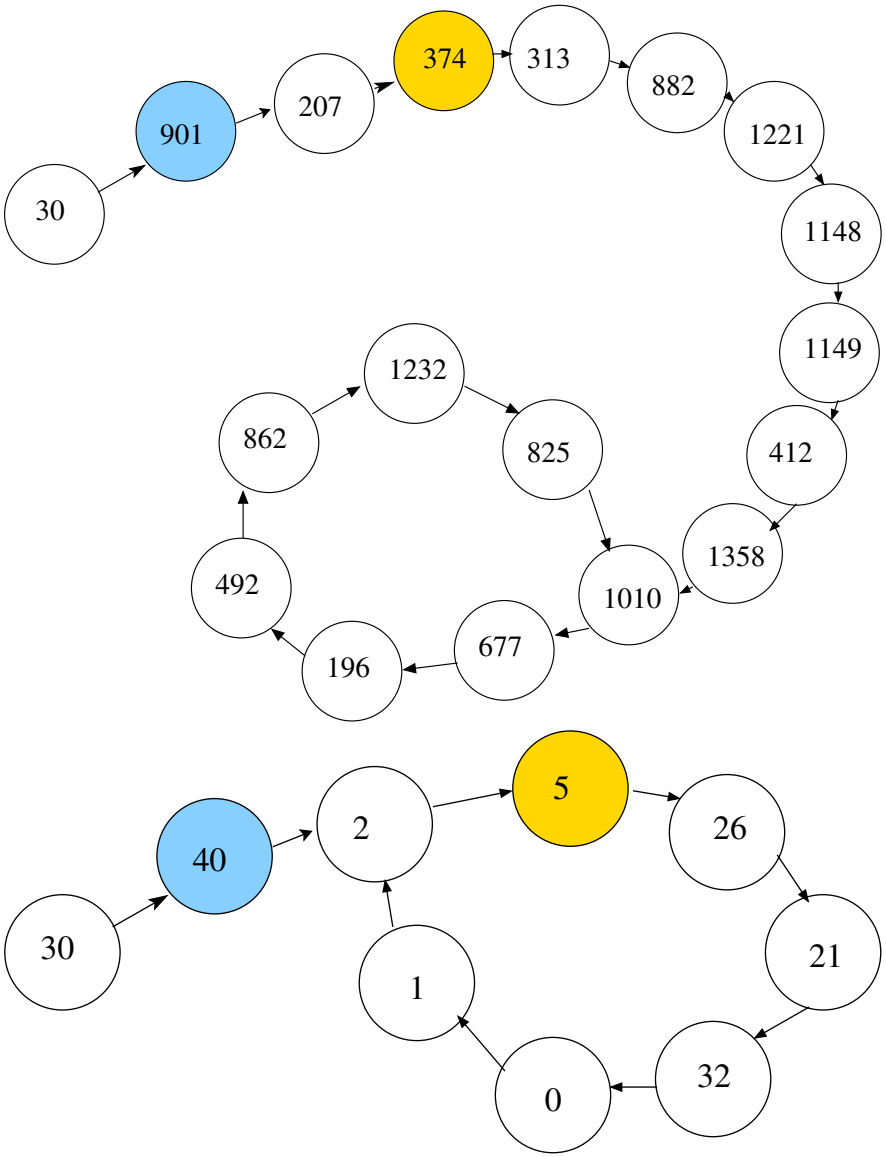
Rho Pollard: illustration



$X_0 \leftarrow \mathsf{rand}(), g \leftarrow 1, i \leftarrow 1;$
$Y_0 \leftarrow f(X_0);$
while $g = 1$ **do**
  $X_i \leftarrow f(X_{i-1});$
  $Y_i \leftarrow f(f(Y_{i-1}));$
  $g \leftarrow \mathsf{GCD}(Y_i - X_i, n);$
  $i \leftarrow i + 1;$
return g

$$\mathsf{GCD}(901 - 30, 1517) = 1$$

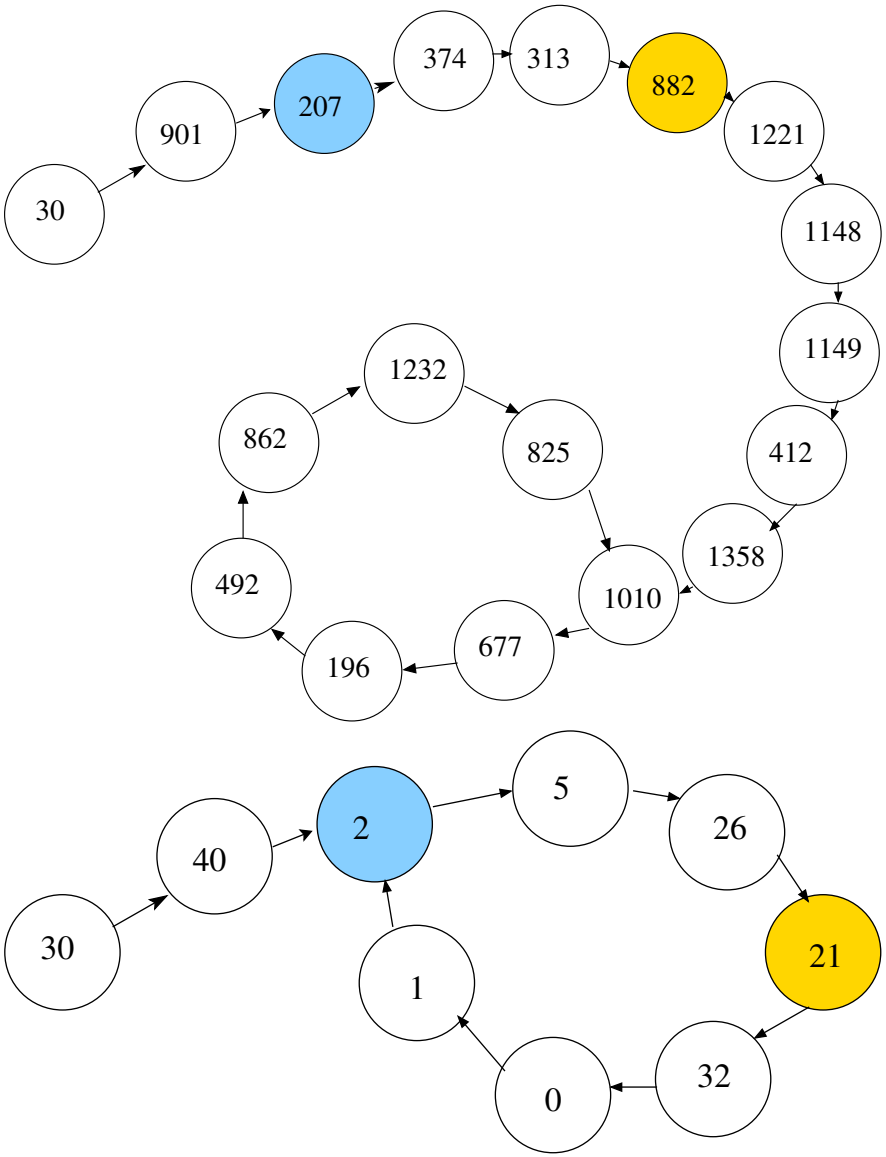
Rho Pollard: illustration



$X_0 \leftarrow \mathsf{rand}(), g \leftarrow 1, i \leftarrow 1;$
$Y_0 \leftarrow f(X_0);$
while $g = 1$ **do**
  $X_i \leftarrow f(X_{i-1});$
  $Y_i \leftarrow f(f(Y_{i-1}));$
  $g \leftarrow \mathsf{GCD}(Y_i - X_i, n);$
  $i \leftarrow i + 1;$
return g

$$\mathsf{GCD}(374 - 901, 1517) = 1$$

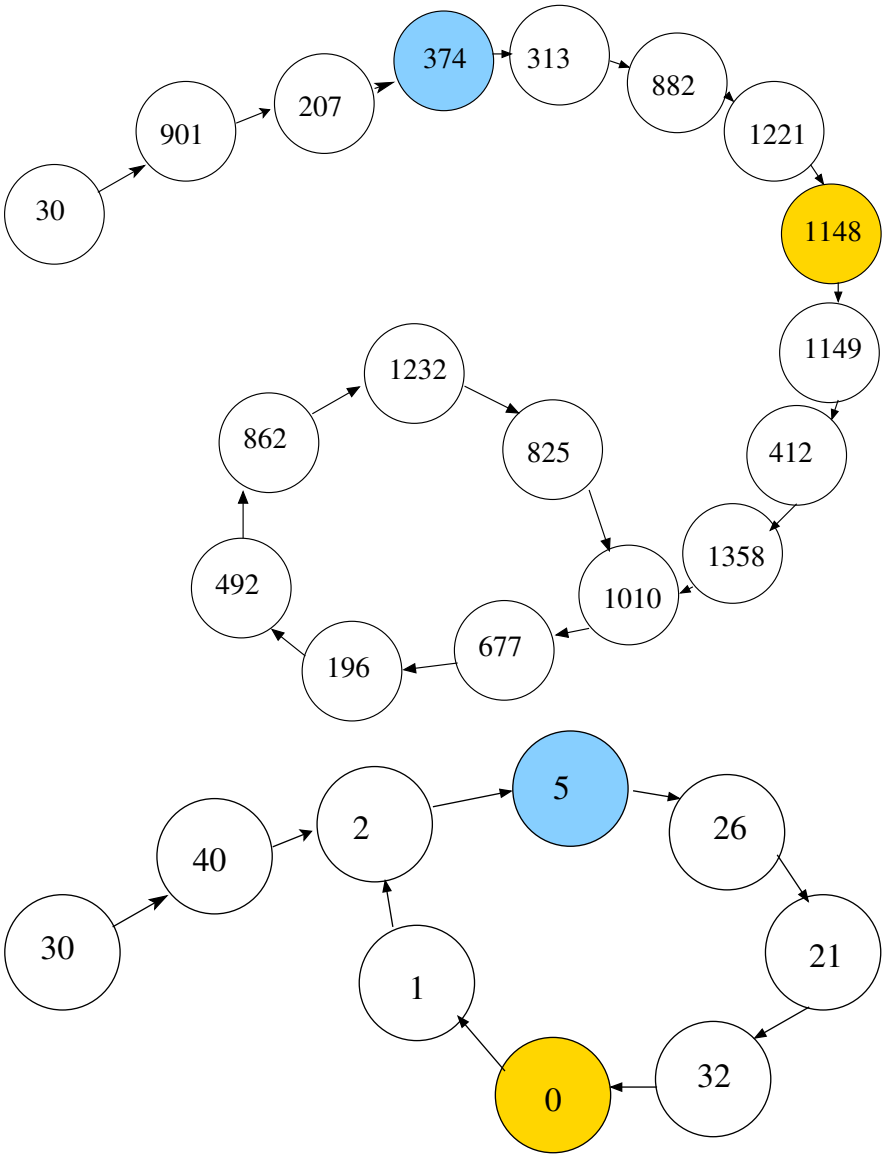
Rho Pollard: illustration



$X_0 \leftarrow \mathsf{rand}(), g \leftarrow 1, i \leftarrow 1;$
$Y_0 \leftarrow f(X_0);$
while $g = 1$ **do**
  $X_i \leftarrow f(X_{i-1});$
  $Y_i \leftarrow f(f(Y_{i-1}));$
  $g \leftarrow \mathsf{GCD}(Y_i - X_i, n);$
  $i \leftarrow i + 1;$
return g

$$\mathsf{GCD}(882 - 207, 1517) = 1$$

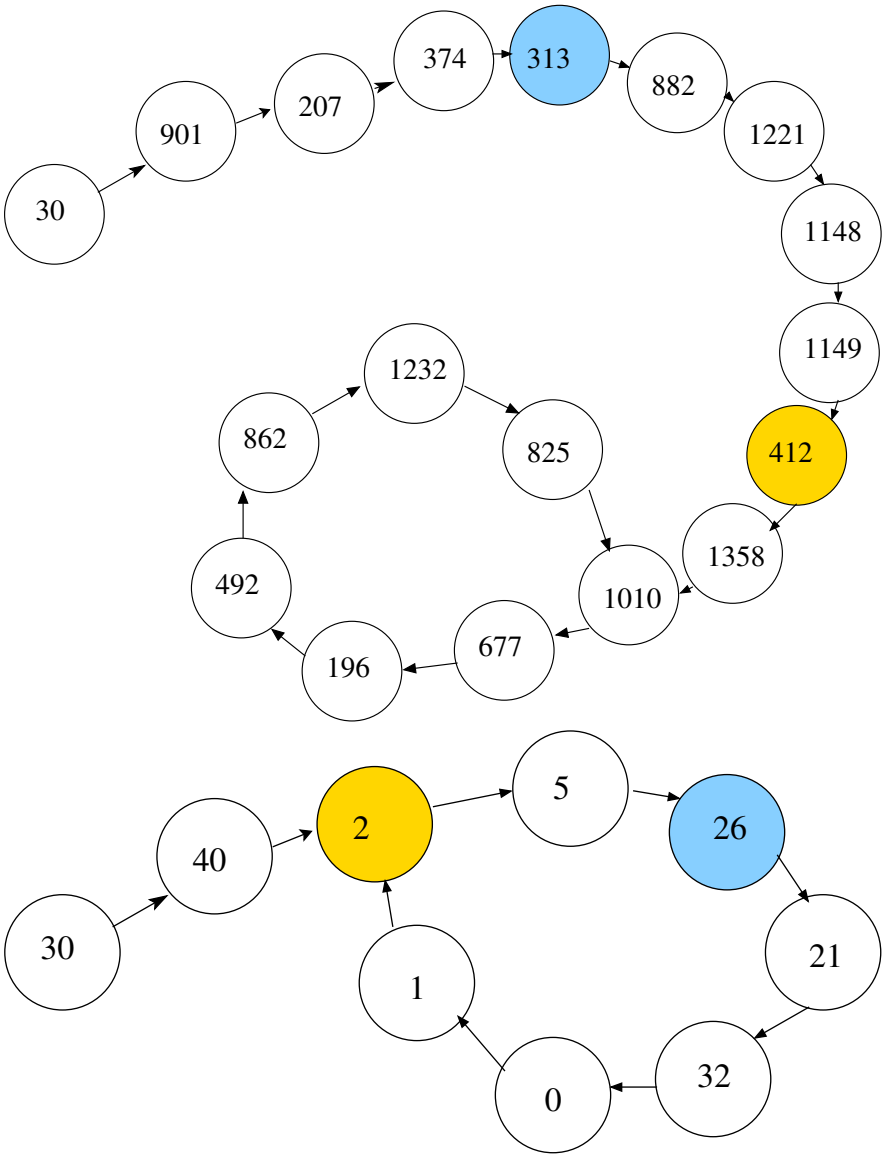
Rho Pollard: illustration



$X_0 \leftarrow \mathsf{rand}(), g \leftarrow 1, i \leftarrow 1;$
$Y_0 \leftarrow f(X_0);$
while $g = 1$ **do**
  $X_i \leftarrow f(X_{i-1});$
  $Y_i \leftarrow f(f(Y_{i-1}));$
  $g \leftarrow \mathsf{GCD}(Y_i - X_i, n);$
  $i \leftarrow i + 1;$
return g

$$\mathsf{GCD}(1148 - 347, 1517) = 1$$

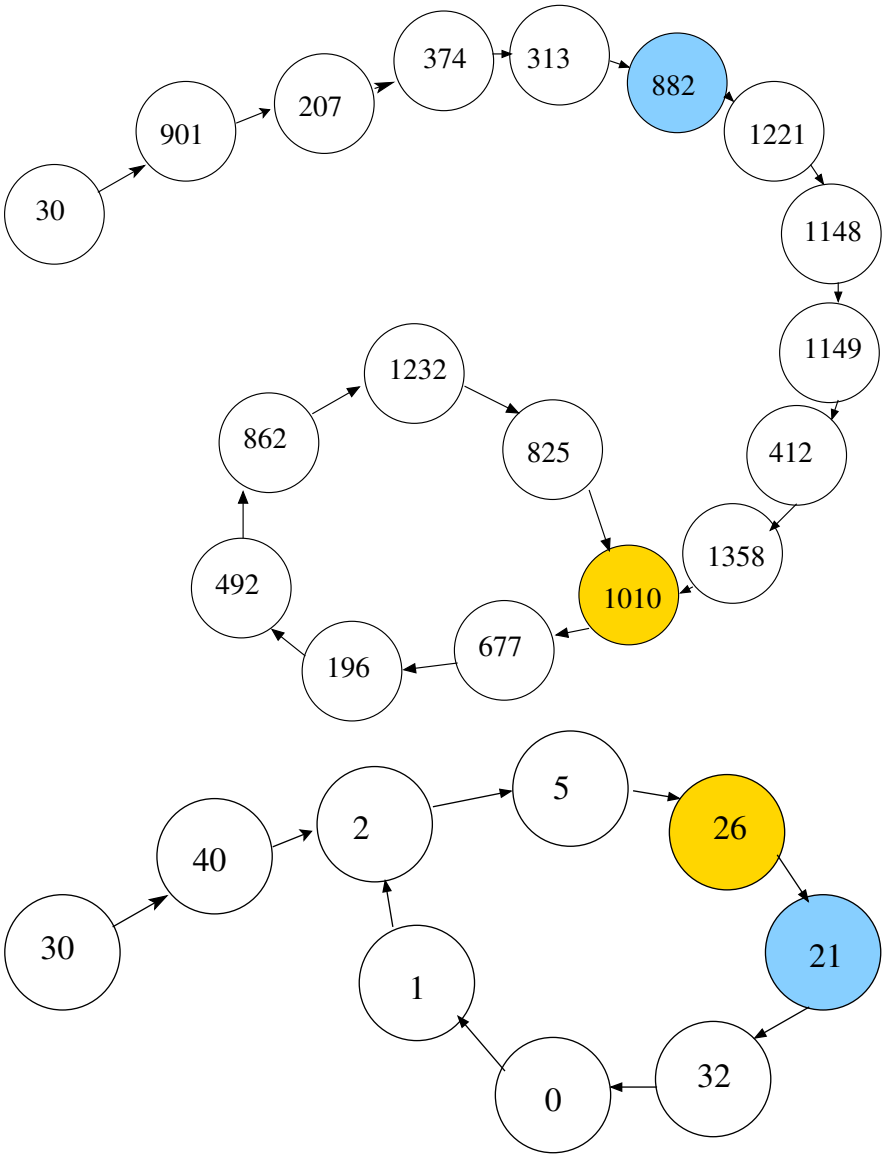
Rho Pollard: illustration



$X_0 \leftarrow \mathsf{rand}(), g \leftarrow 1, i \leftarrow 1;$
$Y_0 \leftarrow f(X_0);$
while $g = 1$ **do**
  $X_i \leftarrow f(X_{i-1});$
  $Y_i \leftarrow f(f(Y_{i-1}));$
  $g \leftarrow \mathsf{GCD}(Y_i - X_i, n);$
  $i \leftarrow i + 1;$
return g

$$\mathsf{GCD}(412 - 313, 1517) = 1$$

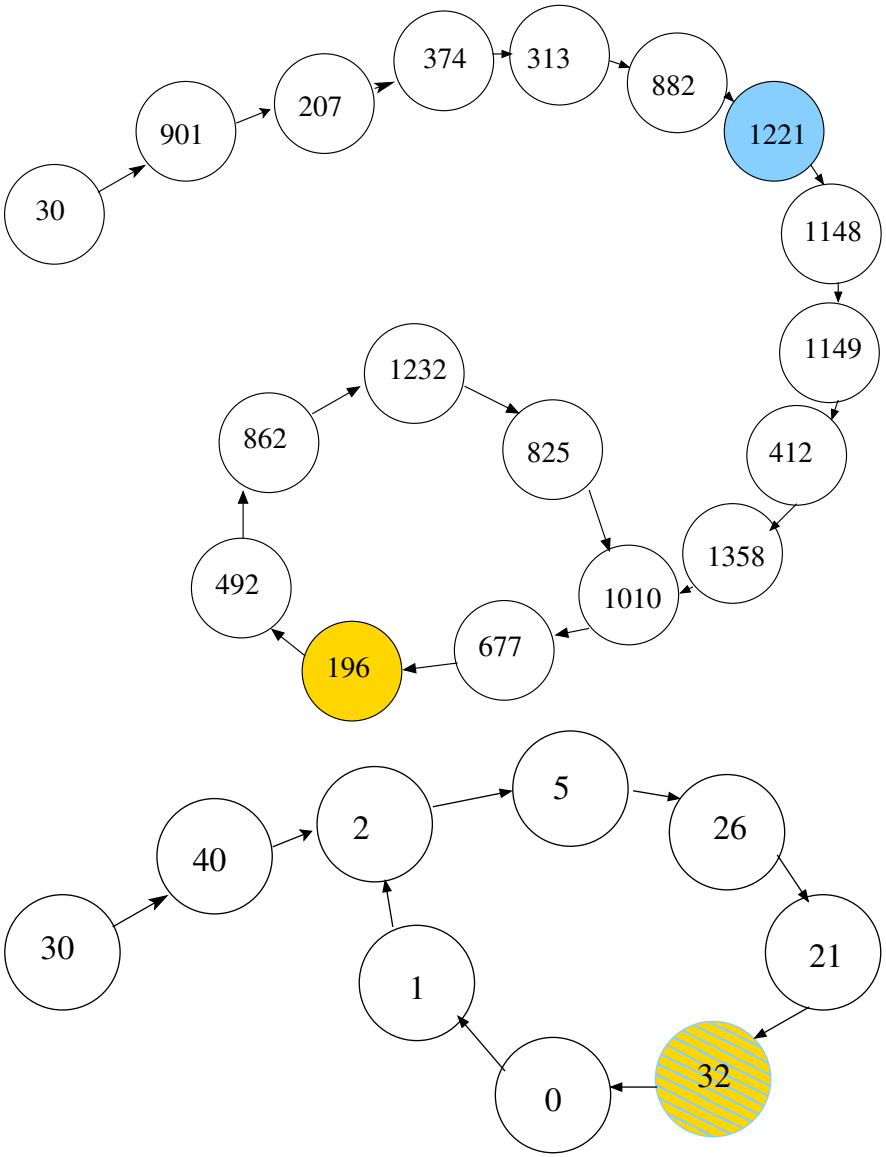
Rho Pollard: illustration



$X_0 \leftarrow \mathsf{rand}(), g \leftarrow 1, i \leftarrow 1;$
$Y_0 \leftarrow f(X_0);$
while $g = 1$ **do**
  $X_i \leftarrow f(X_{i-1});$
  $Y_i \leftarrow f(f(Y_{i-1}));$
  $g \leftarrow \mathsf{GCD}(Y_i - X_i, n);$
  $i \leftarrow i + 1;$
return g

$$\mathsf{GCD}(1010 - 882, 1517) = 1$$

Rho Pollard: illustration



$X_0 \leftarrow \mathsf{rand}(), g \leftarrow 1, i \leftarrow 1;$
$Y_0 \leftarrow f(X_0);$
while $g = 1$ **do**
  $X_i \leftarrow f(X_{i-1});$
  $Y_i \leftarrow f(f(Y_{i-1}));$
  $g \leftarrow \mathsf{GCD}(Y_i - X_i, n);$
  $i \leftarrow i + 1;$
return g

$$\mathsf{GCD}(196 - 1221, 1517) = 41$$

Rho Pollard for the discrete logarithm problem

Solving Discrete logarithm problems

Find x s.t. $y = g^x$ over a group G

Recurring function

For a well chosen partition $G = S_1 \cup S_2 \cup S_3$

$$u_0 = 1 \text{ and } u_{i+1} = f(u_i) = \begin{cases} y \cdot u_i & \text{if } u_i \in S_1 \\ u_i^2 & \text{if } u_i \in S_2 \\ g \cdot u_i & \text{if } u_i \in S_3 \end{cases}$$

Therefore $u_i = g^{a_i} y^{b_i}$ where $a_0 = b_0 = 1$ and

$$a_{i+1} = \begin{cases} a_i & \text{if } u_i \in S_1 \\ 2a_i & \text{if } u_i \in S_2 \\ a_i + 1 & \text{if } u_i \in S_3 \end{cases}, b_{i+1} = \begin{cases} b_i + 1 & \text{if } u_i \in S_1 \\ 2b_i & \text{if } u_i \in S_2 \\ b_i & \text{if } u_i \in S_3 \end{cases}$$

Floyd's cycle detection

At some point

$$\begin{aligned} u_{2i} = u_i \quad &\Rightarrow \quad g^{a_{2i}} y^{b_{2i}} = g^{a_i} y^{b_i} \\ &\Rightarrow \quad g^{a_{2i}-a_i} = y^{b_i-b_{2i}} \\ &\Rightarrow \quad a_{2i} - a_i = (b_i - b_{2i}) \log_g y \end{aligned}$$

With low probability $b_i - b_{2i} = 0$

$$\Rightarrow x = \log_g y = \frac{a_{2i} - a_i}{b_i - b_{2i}}.$$

Outline

Index calculus

Solving Discrete logarithm problems

Find x s.t. $y = g^x$ over $(\mathbb{Z}/p\mathbb{Z})^*$ or $\mathbb{F}_q^*$.

Index calculus

Solving Discrete logarithm problems

Find x s.t. $y = g^x$ over $(\mathbb{Z}/p\mathbb{Z})^*$ or $\mathbb{F}_q^*$.

Index calculus

Solving Discrete logarithm problems

Find x s.t. $y = g^x$ over $(\mathbb{Z}/p\mathbb{Z})^*$ or $\mathbb{F}_q^*$.

Idea

If $y = p_1^{e_1} p_2^{e_2} \dots p_m^{e_m}$ then

$$x = \log_g y = e_1 \log_g p_1 + e_2 \log_g p_2 + \dots + e_m \log_g p_m$$

$\Rightarrow \log_g y$ is a linear combination of the discrete logs of the p_i's.

Index calculus

Solving Discrete logarithm problems

Find x s.t. $y = g^x$ over $(\mathbb{Z}/p\mathbb{Z})^*$ or $\mathbb{F}_q^*$.

Idea

If $y = p_1^{e_1} p_2^{e_2} \dots p_m^{e_m}$ then

$$x = \log_g y = e_1 \log_g p_1 + e_2 \log_g p_2 + \dots + e_m \log_g p_m$$

$\Rightarrow \log_g y$ is a linear combination of the discrete logs of the p_i's.

Outline

1. Select a small subset $S \subset G$ s.t. significantly many elements of G factorize in S.
2. Compute the discrete logs of all elements of S
3. Try to express the unknown DLP as a combination of elements in the database.

Index calculus for the discrete logarithm over $(\mathbb{Z}/p\mathbb{Z})^*$

If $y = p_1^{e_1} p_2^{e_2} \dots p_m^{e_m}$ then

$$\log_g y = e_1 \log_g p_1 + e_2 \log_g p_2 + \cdots + e_m \log_g p_m$$

Database of discrete logs

- Let $S = \{2, 3, 5, 7, \dots\}$ the first m primes

Index calculus for the discrete logarithm over $(\mathbb{Z}/p\mathbb{Z})^*$

If $y = p_1^{e_1} p_2^{e_2} \dots p_m^{e_m}$ then

$$\log_g y = e_1 \log_g p_1 + e_2 \log_g p_2 + \cdots + e_m \log_g p_m$$

Database of discrete logs

- Let $S = \{2, 3, 5, 7, \dots\}$ the first m primes
- Pick random k and compute $y_k = g^k$

Index calculus for the discrete logarithm over $(\mathbb{Z}/p\mathbb{Z})^*$

If $y = p_1^{e_1} p_2^{e_2} \dots p_m^{e_m}$ then

$$\log_g y = e_1 \log_g p_1 + e_2 \log_g p_2 + \dots + e_m \log_g p_m$$

Database of discrete logs

- Let $S = \{2, 3, 5, 7, \dots\}$ the first m primes
- Pick random k and compute $y_k = g^k$
- if y_k factorizes in S: $y_k = \prod_{i=1}^{m} p_i^{e_{k,i}}$
  $\Rightarrow k = \sum_{i=1}^{m} e_{k,i} \log_g p_i$
  $\Rightarrow$one linear relation between the $\log_g p_i$

Index calculus for the discrete logarithm over $(\mathbb{Z}/p\mathbb{Z})^*$

If $y = p_1^{e_1} p_2^{e_2} \dots p_m^{e_m}$ then

$$\log_g y = e_1 \log_g p_1 + e_2 \log_g p_2 + \dots + e_m \log_g p_m$$

Database of discrete logs

- Let $S = \{2, 3, 5, 7, \dots\}$ the first m primes
- Pick random k and compute $y_k = g^k$
- if y_k factorizes in S: $y_k = \prod_{i=1}^{m} p_i^{e_{k,i}}$
  $\Rightarrow k = \sum_{i=1}^{m} e_{k,i} \log_g p_i$
  $\Rightarrow$one linear relation between the $\log_g p_i$
- Iterate until we get m independent equations

Index calculus for the discrete logarithm over $(\mathbb{Z}/p\mathbb{Z})^*$

If $y = p_1^{e_1} p_2^{e_2} \dots p_m^{e_m}$ then

$$\log_g y = e_1 \log_g p_1 + e_2 \log_g p_2 + \dots + e_m \log_g p_m$$

Database of discrete logs

- Let $S = \{2, 3, 5, 7, \dots\}$ the first m primes
- Pick random k and compute $y_k = g^k$
- if y_k factorizes in S: $y_k = \prod_{i=1}^{m} p_i^{e_{k,i}}$
  $\Rightarrow k = \sum_{i=1}^{m} e_{k,i} \log_g p_i$
  $\Rightarrow$one linear relation between the $\log_g p_i$
- Iterate until we get m independent equations
- Solve the linear system

$$\begin{bmatrix} e_{k_1,1} & \dots & e_{k_1,m} \\ \vdots & & \vdots \\ e_{k_m,1} & \dots & e_{k_m,m} \end{bmatrix} \begin{bmatrix} \log_g p_1 \\ \vdots \\ \log_g p_m \end{bmatrix} = \begin{bmatrix} k_1 \\ \vdots \\ k_m \end{bmatrix} \mod p - 1$$

Index calculus for the discrete logarithm over $(\mathbb{Z}/p\mathbb{Z})^*$

If $y = p_1^{e_1} p_2^{e_2} \dots p_m^{e_m}$ then

$$\log_g y = e_1 \log_g p_1 + e_2 \log_g p_2 + \cdots + e_m \log_g p_m$$

Problem: y very likely does not factor in S

Recovery of the unkown discrete log

- Pick k at random and compute $z = g^k y$
- Iterate until z factorizes in S: $z = \prod_{i=1}^{m} p_i^{f_i}$
- Then

$$x = \log_g y = \log_g z - k = \sum_{i=1}^{m} f_i \log_g p_i - k.$$

Index calculus algorithm

- Most work is in the collection of relations
  $\Rightarrow$embarassingly parallel

Index calculus algorithm

- Most work is in the collection of relations
  $\Rightarrow$embarassingly parallel
- Linear algebra phase: compute modulo composite
  $p-1=\prod_{i=1}^{k} q_i^{\ell_i}$

Index calculus algorithm

- Most work is in the collection of relations
  $\Rightarrow$embarassingly parallel
- Linear algebra phase: compute modulo composite
  $p-1=\prod_{i=1}^{k} q_i^{\ell_i}$
  - Chinese remainder modulo each $q_i^{\ell_i}$

Index calculus algorithm

- Most work is in the collection of relations
  $\Rightarrow$embarassingly parallel
- Linear algebra phase: compute modulo composite $p-1=\prod_{i=1}^{k} q_i^{\ell_i}$
  - Chinese remainder modulo each $q_i^{\ell_i}$
  - p-adic lifting from q_i to $q_i^{\ell_i}$

Index calculus algorithm

- Most work is in the collection of relations
  $\Rightarrow$embarassingly parallel
- Linear algebra phase: compute modulo composite
  $p-1=\prod_{i=1}^{k} q_i^{\ell_i}$
  - Chinese remainder modulo each $q_i^{\ell_i}$
  - p-adic lifting from q_i to $q_i^{\ell_i}$
- Once a database is collected, attack becomes much cheaper for each new DLP problem.

Outline

The quadratic sieve

Principle

Find x, y such that

$$\begin{cases} x^2 & = & y^2 & \mod n \\ x & \neq & \pm y & \mod n \end{cases}.$$

Then $g = \mathrm{GCD}(x - y, n) \notin \{1, n\}$ $\Rightarrow$a non-trivial factor of n.

The quadratic sieve

Principle

Find x, y such that

$$\begin{cases} x^2 & = & y^2 & \mod n \\ x & \neq & \pm y & \mod n \end{cases}.$$

Then $g = \text{GCD}(x - y, n) \notin \{1, n\}$ $\Rightarrow$a non-trivial factor of n.

Proof.

Since $x^2 - y^2 = (x - y)(x + y) = kn$,

- if $g = n$ then $x - y = 0 \mod n$
- if $g = 1$ then $x - y$ divides k: $k = (x - y)k'$, and thus $x + y = k'n = 0 \mod n$

□

The quadratic sieve

Example (Factorization of 7429)

$$\begin{aligned} 87^2 &= 1 \times 7429 + 140 \text{ and } 140 = 2^2 \times 5 \times 7 \\ 88^2 &= 1 \times 7429 + 315 \text{ and } 315 = 3^2 \times 5 \times 7 \end{aligned}$$

The quadratic sieve

Example (Factorization of 7429)

$$\begin{aligned} 87^2 &= 1 \times 7429 + 140 \text{ and } 140 = 2^2 \times 5 \times 7 \\ 88^2 &= 1 \times 7429 + 315 \text{ and } 315 = 3^2 \times 5 \times 7 \end{aligned}$$

$$88^2 \times 87^2 = (2 \times 3 \times 5 \times 7)^2 \mod 7429$$

$$\begin{aligned} x &= 87 \times 88 = 227 \mod 7429 \\ y &= 2 \times 3 \times 5 \times 7 = 210 \mod 7429 \end{aligned}$$

The quadratic sieve

Example (Factorization of 7429)

$$\begin{aligned} 87^2 &= 1 \times 7429 + 140 \textit{ and } 140 = 2^2 \times 5 \times 7 \\ 88^2 &= 1 \times 7429 + 315 \textit{ and } 315 = 3^2 \times 5 \times 7 \end{aligned}$$

$$88^2 \times 87^2 = (2 \times 3 \times 5 \times 7)^2 \mod 7429$$

$$\begin{aligned} x &= 87 \times 88 = 227 \mod 7429 \\ y &= 2 \times 3 \times 5 \times 7 = 210 \mod 7429 \end{aligned}$$

$$GCD(227 - 210, 7429) = 17$$

$\Rightarrow$

$$7429 = 17 \times 437$$

The quadratic sieve

	Exp. of 2	Exp. of 3	Exp of 5	Exp. of 7
83^2	2	3	1	0
87^2	2	0	1	1
88^2	0	2	1	1

The quadratic sieve

	Exp. of 2	Exp. of 3	Exp of 5	Exp. of 7
83^2	2	3	1	0
87^2	2	0	1	1
88^2	0	2	1	1

- 87×88 is a square iff L2+L3 is even
- find a vector $x \in \{0,1\}^m$ s.t. $x^T M$ is even
- find a vector x s.t. $x^T M = 0 \mod 2$
- find a non-zero left-kernel vector of M in $\mathbb{Z}/2\mathbb{Z}$.

The quadratic sieve

Sketch of the algorithm

1. Set the factor base as the first t primes $S = \{p_1, p_2, \ldots, p_t\}$
2. Pick x at random near $\lceil \sqrt{n} \rceil$ and compute $b \leftarrow x^2 - n$ until b factorizes in S.
3. Set $a_i = x$ and $b_i = b$ and $v_i = (e_1, \ldots, e_t)$ the multiplicities of S in b
4. Iterate until the collection of relations $\begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix}$ is large enough
5. Find a non zero $x \in \mathbb{F}_2^m$ such that $x^T M = 0 \mod 2$ (e.g. solve $x^T M = b$ for $b = v^T M$ and v random).
6. $X \leftarrow \prod_{i=0}^{m} x_i a_i$
7. $w \leftarrow \frac{1}{2} x^T M$
8. $Y \leftarrow \prod_{i=0}^{m} p_i^{w_i}$
9. Return GCD$(X - Y, n)$

Exercise

Let $n = 23129$.

1. Show that the points $153, 155$ and 197, near $\sqrt{n} \approx 152$ have a smooth square modulo n. Hint: $153^2 = 280 \mod n$ and $197^2 = 2^6 \times 5 \times 7^2 \mod n$ and $586 \times 197 = 22926$.
2. Factor n

Exercise

Let $n = 23129$.

1. Show that the points $153, 155$ and 197, near $\sqrt{n} \approx 152$ have a smooth square modulo n. Hint: $153^2 = 280 \mod n$ and $197^2 = 2^6 \times 5 \times 7^2 \mod n$ and $586 \times 197 = 22926$.
2. Factor n

correction

1. $153^2 = 280 = 2^3 \times 5 \times 7$
2. $155^2 = (153 + 2)^2 = 280 + 612 + 4 = 896 = 2^7 \times 7$
3. $197^2 = 2^6 * 5 * 7^2 \mod n$

Exercise

Let $n = 23129$.

1. Show that the points $153, 155$ and 197, near $\sqrt{n} \approx 152$ have a smooth square modulo n. Hint: $153^2 = 280 \mod n$ and $197^2 = 2^6 \times 5 \times 7^2 \mod n$ and $586 \times 197 = 22926$.
2. Factor n

correction

1. $153^2 = 280 = 2^3 \times 5 \times 7$
2. $155^2 = (153 + 2)^2 = 280 + 612 + 4 = 896 = 2^7 \times 7$
3. $197^2 = 2^6 * 5 * 7^2 \mod n$

Hence $(153 \times 155 \times 197)^2 = 2^{16}5^27^4 = (2^8 \times 5 \times 7)^2 \mod n$
Therefore GCD$(153 \times 155 \times 197 - 2^8 \times 5 \times 7^2, n)$ may be non-trivial.
$153 \times 155 = 153^2 + 2 \times 153 = 280 + 306 = 586$. $586 \times 197 = 22926$.
$2^8 \times 5 \times 7^2 = 16462$. $X - Y = 6464 = 64 \times 101$ and 101 divides n.

Complexity

The $L_n[\alpha, c]$ notation

$$L_n[\alpha, c] = e^{(c+o(1))(\log n)^{\alpha}(\log\log n)^{1-\alpha}}$$

- $L_n[0, c] = (\log n)^{(c+o(1))}$ $\Rightarrow$polynomial in the size $\log n$.
- $L_n[1, c] = n^{(c+o(1))}$ $\Rightarrow$exponential in the size $\log n$.

Complexity

The $L_n[\alpha, c]$ notation

$$L_n[\alpha, c] = e^{(c+o(1))(\log n)^{\alpha}(\log\log n)^{1-\alpha}}$$

- $L_n[0, c] = (\log n)^{(c+o(1))}$ $\Rightarrow$polynomial in the size $\log n$.
- $L_n[1, c] = n^{(c+o(1))}$ $\Rightarrow$exponential in the size $\log n$.

Rho pollard: $L_p[1, 1/2] = \Theta(\sqrt{n})$ exponential time

Complexity

The $L_n[\alpha, c]$ notation

$$L_n[\alpha, c] = e^{(c+o(1))(\log n)^{\alpha}(\log\log n)^{1-\alpha}}$$

- $L_n[0, c] = (\log n)^{(c+o(1))}$ $\Rightarrow$polynomial in the size $\log n$.
- $L_n[1, c] = n^{(c+o(1))}$ $\Rightarrow$exponential in the size $\log n$.

Rho pollard: $L_p[1, 1/2] = \Theta(\sqrt{n})$ exponential time
Index Calculus: $L_p[1/2, c]$ sub-exponential time
Quadratic Sieve: $L_p[1/2, c]$ sub-exponential time

Complexity

The $L_n[\alpha, c]$ notation

$$L_n[\alpha, c] = e^{(c+o(1))(\log n)^{\alpha}(\log\log n)^{1-\alpha}}$$

- $L_n[0, c] = (\log n)^{(c+o(1))}$ $\Rightarrow$polynomial in the size $\log n$.
- $L_n[1, c] = n^{(c+o(1))}$ $\Rightarrow$exponential in the size $\log n$.

Rho pollard: $L_p[1, 1/2] = \Theta(\sqrt{n})$ exponential time
Index Calculus: $L_p[1/2, c]$ sub-exponential time
Quadratic Sieve: $L_p[1/2, c]$ sub-exponential time
Number field sieve: $L_p[1/3, c]$ sub-exponential time

Complexity

The $L_n[\alpha, c]$ notation

$$L_n[\alpha, c] = e^{(c+o(1))(\log n)^{\alpha}(\log\log n)^{1-\alpha}}$$

- $L_n[0, c] = (\log n)^{(c+o(1))}$ $\Rightarrow$polynomial in the size $\log n$.
- $L_n[1, c] = n^{(c+o(1))}$ $\Rightarrow$exponential in the size $\log n$.

Rho pollard: $L_p[1, 1/2] = \Theta(\sqrt{n})$ exponential time
Index Calculus: $L_p[1/2, c]$ sub-exponential time
Quadratic Sieve: $L_p[1/2, c]$ sub-exponential time
Number field sieve: $L_p[1/3, c]$ sub-exponential time

	Factorisation	Discrete logarithm
Rho Pollard	$L_p[1, 1/2]$	$L_p[1, 1/2]$
Index Calculus	N/A	$L_p[1/2, c]$
Quadratic Sieve	$L_p[1/2, c]$	N/A
Number Field Sieve	$L_p[1/3, c]$	$L_p[1/3, c]$

Records in practice

DLP over finite fields with Number Field Sieve

$\mathbb{Z}/p\mathbb{Z}$:

2014 $p \approx 596$ bits in 130 core years

2016 $p \approx 768$ bits in 6600 core years

$GF(q)$

2019: $GF(2^{30750})$ in 2908 cores years

2014: $GF(2^{9234})$ in 45.66 cores years

2013: $GF(2^{6168})$ in 22.9 cores days

DLP over Elliptic curves

2009: modulo 112 bit prime. Rho on 200 PS3 $\times$ 6 months

2016: $GF(2^{117.35})$, Rho on 576 FPGA $\times$ 6 months

Records in practice

Integer factorization with the Number Field Sieve

2005: RSA 640 bits in 4.5 months total

- $36\,000\,000$ columns
- $\approx 7 \times 10^9$ non-zero coeffs

2009: RSA 768 bits in ≈ 2 years (> 1500 core year)

- $192\,796\,550$ columns
- $\approx 27 \times 10^9$ non-zero coeffs

Security parameters and orders of magnitude

From [Lenstra Kleinjung Thomé, 2013]

Security level	Symmetric	Hash	RSA
66	66	132	768
76	76	152	1024
106	106	212	2048

3

4

Security parameters and orders of magnitude

From [Lenstra Kleinjung Thomé, 2013]

[3]	Security level	Symmetric	Hash	RSA
pool security	65	65	130	745
	66	66	132	768
	76	76	152	1024
rain security	80	80	160	1130
lake security	90	90	180	1440
sea	105	105	210	1990
	106	106	212	2048
global[4]	114	114	228	2380

[3] energy required to put that amount of water into boiling status

[4] year of solar energy

Security parameters and orders of magnitude

From [Lenstra Kleinjung Thomé, 2013]

[3]	Security level	Symmetric	Hash	RSA
pool security	65	65	130	745
	66	66	132	768
	76	76	152	1024
rain security	80	80	160	1130
lake security	90	90	180	1440
sea	105	105	210	1990
	106	106	212	2048
global[4]	114	114	228	2380

DLP:

- over prime field: $\approx$ RSA security
- over ECC with strong primes: $\approx$ half the bit size. (Rho Pollard)

[3] energy required to put that amount of water into boiling status

[4] year of solar energy