

# Asymmetric cryptography

## Cryptographic Engineering

---

Clément PERNET

M2 Cyber Security,

UFR-IM<sup>2</sup>AG, Univ. Grenoble-Alpes

ENSIMAG, Grenoble INP



What is a security property

One Way functions

Two fundamental one-way functions

Discrete Logarithm

Integer factorization

Attacking the hard problems

Baby-step / Giant-step

Rho Pollard's algorithm

Index calculus algorithm

Quadratic sieve

# Cryptography is (mainly) about

**Encryption:** To guarantee **confidentiality**

## Encryption scheme

Three functionalities:

**Gen** : key generation  $K_E, K_D \leftarrow \text{Gen}()$

$K_E, K_D$ : keys

**Enc** : encryption  $c \leftarrow \text{Enc}(m, K_E)$

$m$  = plaintext

**Dec** : decryption  $m \leftarrow \text{Dec}(c, K_D)$

$c$  = ciphertext

such that

$$\text{Dec}(\text{Enc}(m, K_E), K_D) = m$$

# Cryptography is (mainly) about

**Encryption:** To guarantee **confidentiality**

**Authentication:** To prove one's identity

## Signature scheme

Three functionalities:

**Gen** : key generation  $K_S, K_V \leftarrow \text{Gen}()$   $K_E, K_D$ : keys

**Sign** : signature  $c \leftarrow \text{Sign}(m, K_S)$   $s$  = signature

**Verify** : decryption  $r \in \{0, 1\} \leftarrow \text{Verify}(s, m, K_V)$

such that

$$\text{Verify}(\text{Sign}(m, K_S), m, K_V) = 1 \text{ and } P[\text{Verify}(s', m, K_V) = 0] > 1 - \epsilon$$

## What is a safe cryptosystem ?

- No attacker can recover the decryption key ?
  - ⇒ Not sufficient e.g.  $Enc(m, K) = m$ .

## What is a safe cryptosystem ?

- No attacker can recover the decryption key ?  
⇒ Not sufficient e.g.  $Enc(m, K) = m$ .
- No attacker can recover the plaintext ?  
⇒ partially recovering a plaintext is already an issue.

# How to assess the security of a cryptosystem

## What is a safe cryptosystem ?

- No attacker can recover the decryption key ?  
⇒ Not sufficient e.g.  $Enc(m, K) = m$ .
- No attacker can recover the plaintext ?  
⇒ partially recovering a plaintext is already an issue.
- No attacker can recover any character of the plaintext ?  
⇒ knowing the number of digits of a salary is already a leak.

# How to assess the security of a cryptosystem

## What is a safe cryptosystem ?

- No attacker can recover the decryption key ?  
⇒ Not sufficient e.g.  $Enc(m, K) = m$ .
- No attacker can recover the plaintext ?  
⇒ partially recovering a plaintext is already an issue.
- No attacker can recover any character of the plaintext ?  
⇒ knowing the number of digits of a salary is already a leak.
- The knowledge of the ciphertext does not provide any additional information about the plaintext to an attacker (besides what he already knows)

# How to define the security of a cryptosystem

## Define the goal of the attacker

**BREAK:** decrypt any message (e.g. by finding the decryption key)

**OW (oneway):** decrypt a given message

**IND (indistinguishability):** distinguish between two ciphertexts

**NM (non malleability):** modify a ciphertext so that it becomes the ciphertext of another plaintext clair

Hierarchy in the power of the goals :

BREAK  $\gg$  OW  $\gg$  IND  $\gg$  NM

# How to define the security of a cryptosystem

## Clarify the means that an attacker can use

**COA (Ciphertext Only Attack):** knows only one/some ciphertexts

**KPA (Known Plaintext Attack):** knows pairs (plaintext, ciphertext)

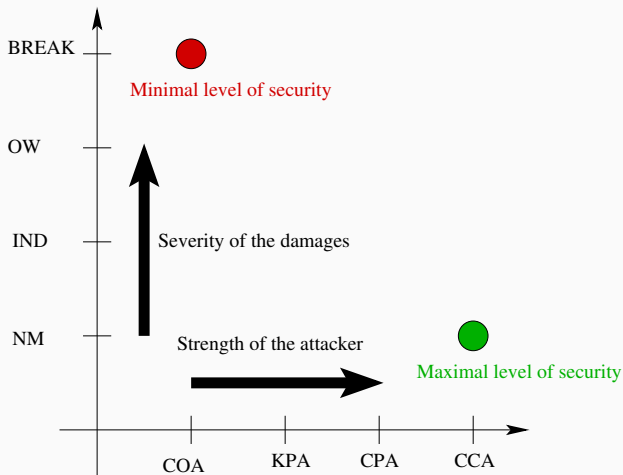
**CPA (Chosen Plaintext Attack):** able to encrypt chosen plaintexts

**CCA (Chosen Ciphertext Attack):** able to decrypt some ciphertexts  
(except the target)

Hierarchy in the level of the means:

$\text{COA} \ll \text{KPA} \ll \text{CPA} \ll \text{CCA}$

# Comparing security levels



## Clarify the underlying hypotheses

- (almost) never an absolute **proof** of security
- Security proofs most often relies on one/some hypotheses
  - Computational Hardness of integer factoring
  - Computational Hardness of solving Decisional Diffie-Hellman, etc

# How to define the security of a cryptosystem

## Clarify the underlying hypotheses

- (almost) never an absolute **proof** of security
- Security proofs most often relies on one/some hypotheses
  - Computational Hardness of integer factoring
  - Computational Hardness of solving Decisional Diffie-Hellman, etc
- State clearly the required hypotheses
- Prefer the *simplest, standard, old* hypotheses

# How to define the security of a cryptosystem

## Proving security

**Theorem:** *Under hypothesis  $H$ , an attacker following the attack model  $M$  with  $x$  resources, can not reach goal  $G$  (with probability  $> \epsilon$ ).*

# How to define the security of a cryptosystem

## Proving security

**Theorem:** *Under hypothesis  $H$ , an attacker following the attack model  $M$  with  $x$  resources, can not reach goal  $G$  (with probability  $> \epsilon$ ).*

## Warning

- A security proof only offers guaranties under the hypotheses being made:
  - Security hypotheses
  - Attacker model
  - Means used

# How to define the security of a cryptosystem

## Proving security

**Theorem:** *Under hypothesis  $H$ , an attacker following the attack model  $M$  with  $x$  resources, can not reach goal  $G$  (with probability  $> \epsilon$ ).*

## Warning

- A security proof only offers guaranties under the hypotheses being made:

- Security hypotheses
- Attacker model
- Means used

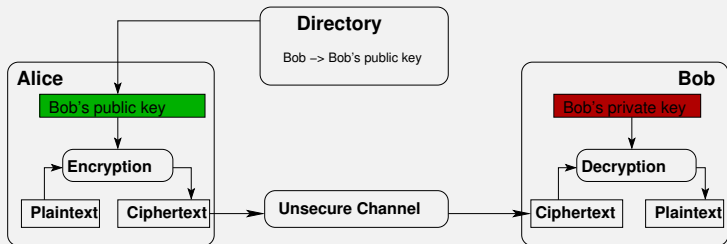
⇒ restricts the perimeter of the possible attacks

⇒ may differ in practice: failing implementation, side-channel, etc

# Asymmetric Cryptography

*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

What is a security property

## One Way functions

Two fundamental one-way functions

- Discrete Logarithm

- Integer factorization

Attacking the hard problems

- Baby-step / Giant-step

- Rho Pollard's algorithm

- Index calculus algorithm

- Quadratic sieve

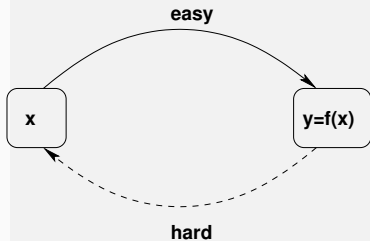
# One way functions

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

## One way function

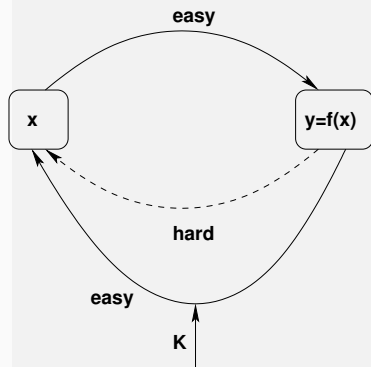


- easy to compute in one way
- hard to invert (other way) ...

# One way functions

- The private key  $K_d$  is fully determined by the public key  $K_e$   
⇒ 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

What is a security property

One Way functions

Two fundamental one-way functions

Discrete Logarithm

Integer factorization

Attacking the hard problems

Baby-step / Giant-step

Rho Pollard's algorithm

Index calculus algorithm

Quadratic sieve

What is a security property

One Way functions

Two fundamental one-way functions

Discrete Logarithm

Integer factorization

Attacking the hard problems

Baby-step / Giant-step

Rho Pollard's algorithm

Index calculus algorithm

Quadratic sieve

# 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 ... k do  
  y ← y × 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**

if  $k=0$  then

└ return 1

Compute recursively

$z \leftarrow x^{\lfloor k/2 \rfloor}$ ;

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  $G$

**begin**

$y \leftarrow 1$ ;

for  $i = 1 \dots k$  do

└  $y \leftarrow y \times x$ ;

$\Rightarrow O(k) = O(|G|)$  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**

```
if  $k=0$  then
  return 1
Compute recursively
 $z \leftarrow x^{\lfloor k/2 \rfloor}$ ;
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  $G$

**begin**

```
 $y \leftarrow 1$ ;
for  $i = 1 \dots k$  do
   $y \leftarrow y \times x$ ;
```

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

**begin**

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

# Reciprocal: the discrete logarithm

## Problem

From  $y, x \in G$ , compute  $k \in \{1 \dots |G|\}$  such that  $y = x^k$

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

## One way function

- $\text{Exp}_x$ : easy (polynomial time complexity)
- $\log_x$ : hard (no known polynomial time algorithm)

# Instances of groups to build a DLP

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

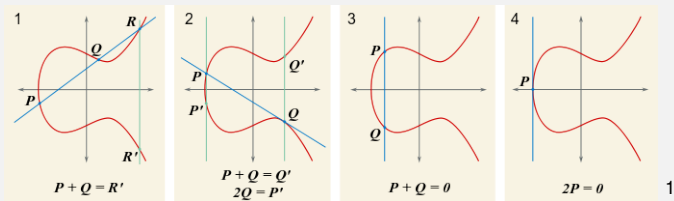
# Instances of groups to build a DLP

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

## Elliptic curve arithmetic

$$(E) : y^2 = ax^3 + b \pmod{p}$$

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

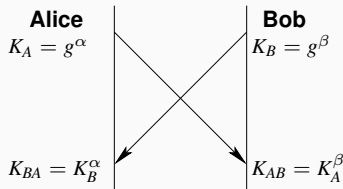


- $(p, k) \longrightarrow kP = \underbrace{P + \dots + P}_{k \text{ times}}$  is easy
- $(P, Y) \longrightarrow k \text{ s.t. } Y = kP$  is hard.

<sup>1</sup>CC-BY-SA-3.0, SuperManu at English Wikipedia

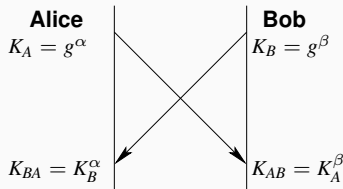
# Diffie Hellman key exchange protocol

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



# Diffie Hellman key exchange protocol

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



# Diffie Hellman key exchange protocol

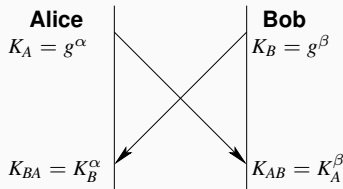
1. Choose publicly a group  $G$  and a generator  $g \in G$

2. Alice:

- chooses a secret  $\alpha \in \{1 \dots |G|\}$
- computes  $K_A = g^\alpha$
- sends  $K_A$  to Bob
- computes  $K_{BA} = K_B^\alpha$

3. Bob:

- chooses a secret  $\beta \in \{1 \dots |G|\}$
- 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  $\alpha, \beta$ .

# Security of the Diffie-Hellmann protocol

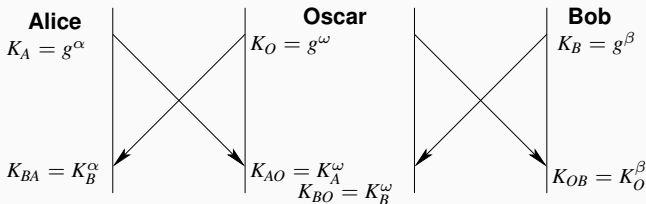
- From  $g, K_A, K_B$  find  $\alpha, \beta \Rightarrow$  need to compute a discrete logarithm

# Security of the Diffie-Hellmann protocol

- From  $g, K_A, K_B$  find  $\alpha, \beta \Rightarrow$  need to compute a discrete logarithm
- From  $g, K_A, K_B$  find  $K_{AB}$  : not clear
  - CDH: Computational Diffie-Hellman (a problem on its own)

# Security of the Diffie-Hellmann protocol

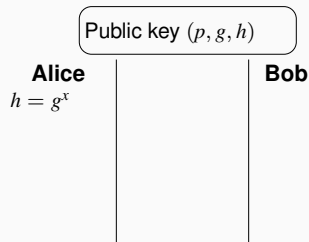
- From  $g, K_A, K_B$  find  $\alpha, \beta \Rightarrow$  need to compute a discrete logarithm
- From  $g, K_A, K_B$  find  $K_{AB}$  : not clear
  - CDH: Computational Diffie-Hellman (a problem on its own)
- Weakness : *Person in the middle attack*



# 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

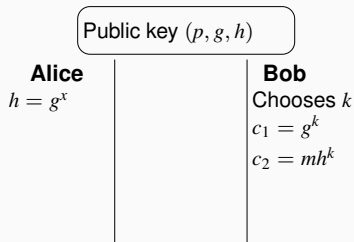
## 1. Alice:

- 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

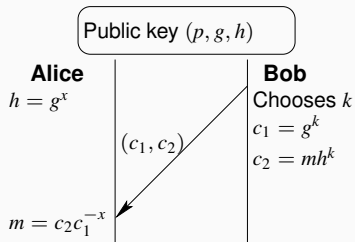
## 1. Alice:

- chooses  $G$  and a generator  $g$
- chooses a secret  $x$
- computes  $h = g^x$
- Public key:  $(g, h)$
- computes  $c_2c_1^{-x}$

## 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

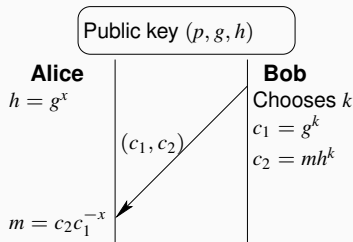
## 1. Alice:

- chooses  $G$  and a generator  $g$
- chooses a secret  $x$
- computes  $h = g^x$
- Public key:  $(g, h)$
- computes  $c_2c_1^{-x}$

## 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



$$c_2c_1^{-x} = mh^k g^{-kx} = mg^{kx} g^{-kx} = m$$

What is a security property

One Way functions

**Two fundamental one-way functions**

Discrete Logarithm

**Integer factorization**

Attacking the hard problems

Baby-step / Giant-step

Rho Pollard's algorithm

Index calculus algorithm

Quadratic sieve

# The RSA cryptosystem: *Rivest Shamir Adelman*

## Theorem (Rivest Shamir Adelman 78)

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

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

Proof:

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

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

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

# The RSA cryptosystem

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

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

## The RSA cryptosystem:

- Public key:  $(e, n)$ , encryption:  $c = m^e \pmod{n}$
- Private key:  $(d, n)$ , decryption:  $c^d \pmod{n}$

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

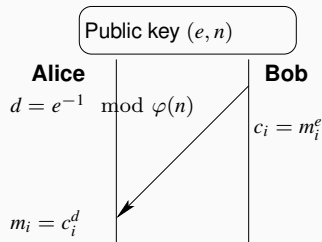
## Warning:

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

# The RSA cryptosystem

## 1. Alice:

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



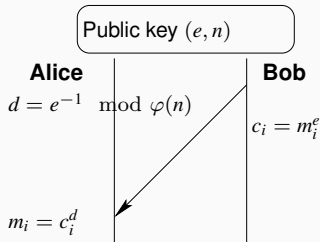
# The RSA cryptosystem

## 1. Alice:

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

## 2. Bob: wants to send the message $m$

- splits  $m = (m_1, \dots, m_k)$  st.  $m_i < n$
- For each  $m_i$
- sends  $c_i = m_i^e \pmod n$  to Alice



## Exercise

Show that RSA is not IND-CPA. [Hint: because  $\text{Enc}$  is deterministic]

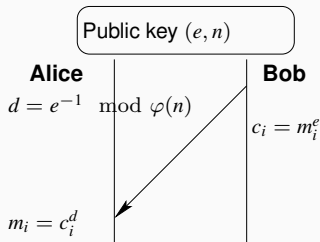
# The RSA cryptosystem

## 1. Alice:

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

## 2. Bob: wants to send the message $m$

- splits  $m = (m_1, \dots, m_k)$  st.  $m_i < n$
- For each  $m_i$
- sends  $c_i = m_i^e \pmod n$  to Alice



## Exercise

Show that RSA is not IND-CPA. [Hint: because  $\text{ENC}$  is deterministic]

# Randomized Padding

Not IND-CPA = Not semantically secure

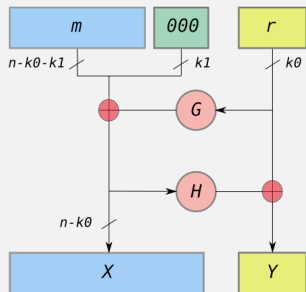
⇒ use randomized padding on each block

# Randomized Padding

Not IND-CPA = Not semantically secure

⇒ use randomized padding on each block

## The OAEP padding<sup>2</sup>



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

<sup>2</sup>CC-BY-SA-3.0, Ozga at English Wikipedia

# RSA algorithmic

- Computing  $p, q$  primes  $\Rightarrow$  prime number generator  $\Rightarrow$  expected  $O(\log^2 n)$
- Computing  $n, \varphi(n) = (p-1)(q-1)$   $\Rightarrow$  Int. multiplication  $O(\log n)$
- Computing  $d = e^{-1} \pmod{\varphi(n)}$   $\Rightarrow$  Ext. Euclidean alg.  $O(\log n)$
- Encryption:  $c = m^e \pmod n$   $\Rightarrow$  Modular exponentiation  $O(\log^2 n)$
- Decryption:  $m = c^d \pmod n$   $\Rightarrow$  Modular exponentiation  $O(\log^2 n)$

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).

## Exercise

1. Write a modular exponentiation algorithm computing  $m^d \pmod n$  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 multiplication). How does it compare to the standard modular exponentiation?

# CRT based RSA decryption

$$m, \quad d \quad \longrightarrow \quad m^d \pmod{n}$$

# CRT based RSA decryption

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

# CRT based RSA decryption

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

# CRT based RSA decryption

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

# CRT based RSA decryption

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

## Algorithm(s)

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

# CRT based RSA decryption

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

## Algorithm(s)

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

### 1. Precompute:

$$1.1 \ a \leftarrow q (q^{-1} \pmod p) \pmod n$$

$$1.2 \ b \leftarrow p (p^{-1} \pmod q) \pmod n$$

$$2. \ m_p \leftarrow m \pmod p; m_q \leftarrow m \pmod q$$

$$3. \ c_p \leftarrow \text{ExpMod}(m_p, d_p, p)$$

$$4. \ c_q \leftarrow \text{ExpMod}(m_q, d_q, q)$$

$$5. \ c \leftarrow ac_p + bc_q$$

# CRT based RSA decryption

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

## Algorithm(s)

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

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

### 1. Precompute:

$$1.1 \ a \leftarrow q (q^{-1} \pmod p) \pmod n$$

$$1.2 \ b \leftarrow p (p^{-1} \pmod q) \pmod n$$

$$2. \ m_p \leftarrow m \pmod p; m_q \leftarrow m \pmod q$$

$$3. \ c_p \leftarrow \text{ExpMod}(m_p, d_p, p)$$

$$4. \ c_q \leftarrow \text{ExpMod}(m_q, d_q, q)$$

$$5. \ c \leftarrow a c_p + b c_q$$

# CRT based RSA decryption

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

## Algorithm(s)

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

### 1. Precompute:

$$1.1 \ a \leftarrow q (q^{-1} \pmod p) \pmod n$$

$$1.2 \ b \leftarrow p (p^{-1} \pmod q) \pmod n$$

$$2. \ m_p \leftarrow m \pmod p; m_q \leftarrow m \pmod q$$

$$3. \ c_p \leftarrow \text{ExpMod}(m_p, d_p, p)$$

$$4. \ c_q \leftarrow \text{ExpMod}(m_q, d_q, q)$$

$$5. \ c \leftarrow ac_p + bc_q$$

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

### 1. Precompute:

$$1.1 \ a \leftarrow p (p^{-1} \pmod q) \pmod n$$

$$2. \ m_p \leftarrow m \pmod p$$

$$3. \ m_q \leftarrow m \pmod q$$

$$4. \ c_p \leftarrow \text{ExpMod}(m_p, d_p, p)$$

$$5. \ c_q \leftarrow \text{ExpMod}(m_q, d_q, q)$$

$$6. \ c \leftarrow m_p + a(c_q - c_p)$$

# 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 successfully 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

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

1. If only one of  $c_p$  or  $c_q$  is incorrect (say  $c_p$ ) then

- the faulty  $c'$  still verifies  $c' = c_q \pmod q$ .
- hence  $c' - c = 0 \pmod q$  and  $q$  divides  $c' - c$
- moreover  $c' - c = a(c'_p - c_p) + \lambda n$ .
- but  $q$  divides  $a$  ( $a = q(q^{-1} \pmod p)$ )
- Therefore  $q$  divides  $c' - c$  but  $p$  does not.
- Thus  $\gcd(c' - c, n) = q$ .

# Fault injection on CRT-RSA

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

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

# Fault injection on CRT-RSA

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

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

## Counter-measures

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

What is a security property

One Way functions

Two fundamental one-way functions

Discrete Logarithm

Integer factorization

Attacking the hard problems

Baby-step / Giant-step

Rho Pollard's algorithm

Index calculus algorithm

Quadratic sieve

# 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

What is a security property

One Way functions

Two fundamental one-way functions

Discrete Logarithm

Integer factorization

**Attacking the hard problems**

**Baby-step / Giant-step**

Rho Pollard's algorithm

Index calculus algorithm

Quadratic sieve

# Baby-step / Giant-step to solve a DLP

## Problem

Given  $x, y \in G$ , find  $k$  such that  $y = x^k$

# Baby-step / Giant-step to solve a DLP

## Problem

Given  $x, y \in G$ , find  $k = k_0 + mk_1$  such that  $y = x^k$

- Writing  $k$  in base  $m = \sqrt{|G|}$

# Baby-step / Giant-step to solve a DLP

## Problem

Given  $x, y \in G$ , find  $k = k_0 + mk_1$  such that  $y = x^k = x^{k_0} (x^m)^{k_1}$

- Writing  $k$  in base  $m = \sqrt{|G|}$
- splits the unknown bits of  $k$  in two equal parts  $k_0, k_1 \in \{0 \dots m - 1\}$

# Baby-step / Giant-step to solve a DLP

## Problem

Given  $x, y \in G$ , find  $k = k_0 + mk_1$  such that  $y = x^k = x^{k_0}(x^m)^{k_1}$

- Writing  $k$  in base  $m = \sqrt{|G|}$
- splits the unknown bits of  $k$  in two equal parts  $k_0, k_1 \in \{0 \dots m-1\}$
- Compute and store all possible  $x^{k_0}$

# Baby-step / Giant-step to solve a DLP

## Problem

Given  $x, y \in G$ , find  $k = k_0 + mk_1$  such that  $y = x^k = x^{k_0}(x^m)^{k_1}$

- Writing  $k$  in base  $m = \sqrt{|G|}$
- splits the unknown bits of  $k$  in two equal parts  $k_0, k_1 \in \{0 \dots m - 1\}$
- Compute and store all possible  $x^{k_0}$
- Iterate through all  $(x^m)^{k_1}$  to find a match

# Baby-step / Giant-step to solve a DLP

## Problem

Given  $x, y \in G$ , find  $k = k_0 + mk_1$  such that  $y = x^k = x^{k_0}(x^m)^{k_1}$

- Writing  $k$  in base  $m = \sqrt{|G|}$
- splits the unknown bits of  $k$  in two equal parts  $k_0, k_1 \in \{0 \dots m-1\}$
- Compute and store all possible  $x^{k_0}$
- Iterate through all  $(x^m)^{k_1}$  to find a match

$m \leftarrow \sqrt{|G|};$

**for**  $i \leftarrow 1 \dots m$  **do**

└ Store  $(x^i, i)$  in a table  $T$ ;

$h \leftarrow (x^m)^{-1};$

**for**  $j \leftarrow 1 \dots m$  **do**

└ **if**  $\exists i, (y \times h^j, i) \in T$  **then**  
└└ **return**  $i + jm$ ;

# Baby-step / Giant-step to solve a DLP

## Problem

Given  $x, y \in G$ , find  $k = k_0 + mk_1$  such that  $y = x^k = x^{k_0}(x^m)^{k_1}$

- Writing  $k$  in base  $m = \sqrt{|G|}$
- splits the unknown bits of  $k$  in two equal parts  $k_0, k_1 \in \{0 \dots m-1\}$
- Compute and store all possible  $x^{k_0}$
- Iterate through all  $(x^m)^{k_1}$  to find a match

$m \leftarrow \sqrt{|G|};$

**for**  $i \leftarrow 1 \dots m$  **do**

└ Store  $(x^i, i)$  in a table  $T$ ;

$h \leftarrow (x^m)^{-1};$

**for**  $j \leftarrow 1 \dots m$  **do**

└ **if**  $\exists i, (y \times h^j, i) \in T$  **then**  
└└ **return**  $i + jm$ ;

## Implementation of $T$

Simple array:  $T[i] = x^i$

- sort  $\Rightarrow O(m \log m)$
- search  $O(\log m) \times m$

Hash table:  $T[x^i] = i$

- search:  $O(1)$

# Baby-step / Giant-step to solve a DLP

## Problem

Given  $x, y \in G$ , find  $k = k_0 + mk_1$  such that  $y = x^k = x^{k_0}(x^m)^{k_1}$

- Writing  $k$  in base  $m = \sqrt{|G|}$
- splits the unknown bits of  $k$  in two equal parts  $k_0, k_1 \in \{0 \dots m-1\}$
- Compute and store all possible  $x^{k_0}$
- Iterate through all  $(x^m)^{k_1}$  to find a match

$m \leftarrow \sqrt{|G|};$

**for**  $i \leftarrow 1 \dots m$  **do**

    Store  $(x^i, i)$  in a table  $T$ ;

$h \leftarrow (x^m)^{-1};$

**for**  $j \leftarrow 1 \dots m$  **do**

**if**  $\exists i, (y \times h^j, i) \in T$  **then**  
        **return**  $i + jm$ ;

## Implementation of $T$

Simple array:  $T[i] = x^i$

- sort  $\Rightarrow O(m \log m)$
- search  $O(\log m) \times m$

Hash table:  $T[x^i] = i$

- search:  $O(1)$

**Cost:**  $O(\sqrt{|G|})$  time and  $O(\sqrt{|G|})$  space

What is a security property

One Way functions

Two fundamental one-way functions

Discrete Logarithm

Integer factorization

**Attacking the hard problems**

Baby-step / Giant-step

**Rho Pollard's algorithm**

Index calculus algorithm

Quadratic sieve

# 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

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

$p$  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

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

$p$  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:

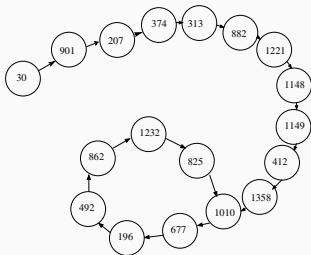
- Recurring sequence mod  $n$ :
$$\begin{cases} X_0 & = \text{rand}(), \\ X_{i+1} & = f(X_i) \bmod n \end{cases}$$
- Finite number of states: ultimately periodic

# Collision in a recurring sequence

## Principle:

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

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



modulo  $1517 = 41 \times 37$

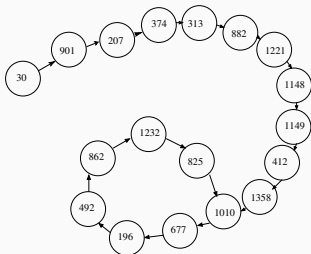
# Collision in a recurring sequence

## Principle:

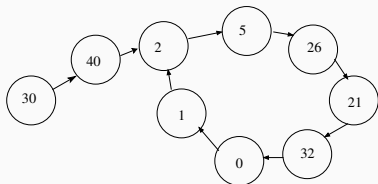
- Recurring sequence mod  $n$ : 
$$\begin{cases} X_0 &= \text{rand}(), \\ X_{i+1} &= f(X_i) \bmod n \end{cases}$$
- Finite number of states: ultimately periodic

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

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



modulo 1517 = 41 × 37

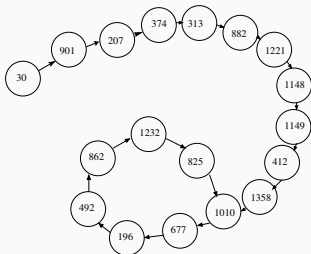


modulo 41

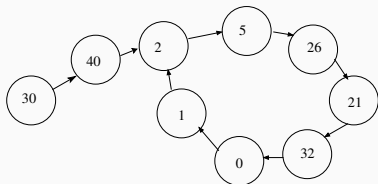
# Collision in a recurring sequence

## Principle:

- Recurring sequence mod  $n$ : 
$$\begin{cases} X_0 &= \text{rand}(), \\ X_{i+1} &= f(X_i) \bmod n \end{cases}$$
- Finite number of states: ultimately periodic
  - mod  $n$ :** period  $\lambda_n$  s.t.  $X_{t+\lambda_n} = X_t \bmod n \forall t \geq \mu_n$
  - mod  $p$ :** period  $\lambda_p \leq \lambda_n$  s.t.  $X_{t+\lambda_p} = X_t \bmod p \forall t \geq \mu_p$
- $\text{GCD}((X_{t+k} - X_t), n) = p \text{ or } q \Rightarrow \text{If } X_{t+k} - X_t \neq 0 \bmod n \Rightarrow \text{get } p$

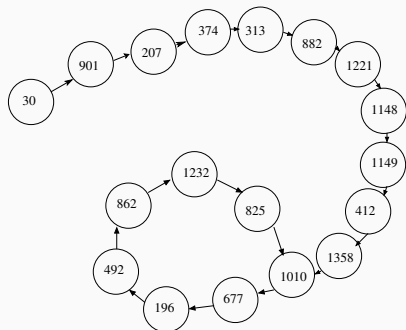


modulo 1517 = 41 × 37

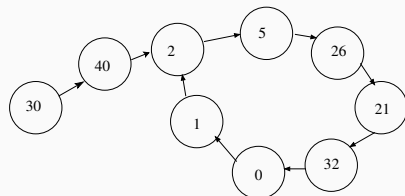


modulo 41

# 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  $X_i = X_j \pmod{p}$ .
2. What is its time and space complexity?

# Floyd's cycle detection

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

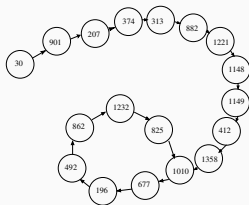
**Tail length:**  $\mu$

**Period:**  $\lambda$

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

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.

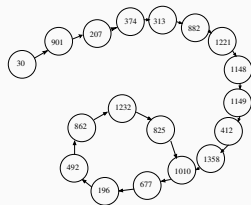
**Tail length:**  $\mu$

**Period:**  $\lambda$

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

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.

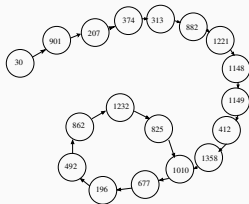
**Tail length:**  $\mu$

**Period:**  $\lambda$

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

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

# Floyd's cycle detection

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

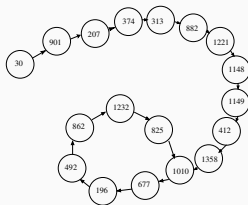
**Tail length:**  $\mu$

**Period:**  $\lambda$

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

Iterate through  $u_i$  with 2 paces:

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



## Lemma

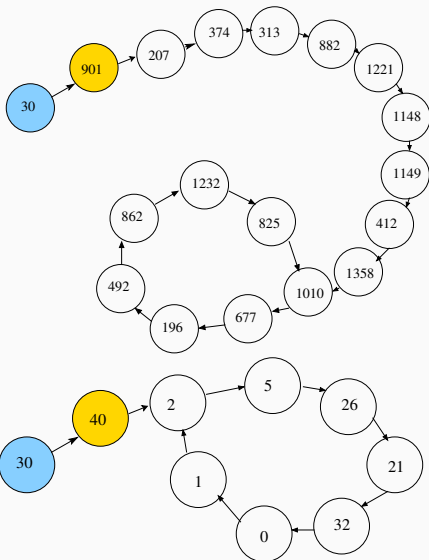
$$u_i = u_{2i} \Leftrightarrow i = k\lambda \geq \mu$$

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

## Cost

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

# Rho Pollard: illustration



$X_0 \leftarrow \text{rand}(), g \leftarrow 1, i \leftarrow 1;$

$Y_0 \leftarrow f(X_0);$

**while**  $g = 1$  **do**

$X_i \leftarrow f(X_{i-1});$

$Y_i \leftarrow f(f(Y_{i-1}));$

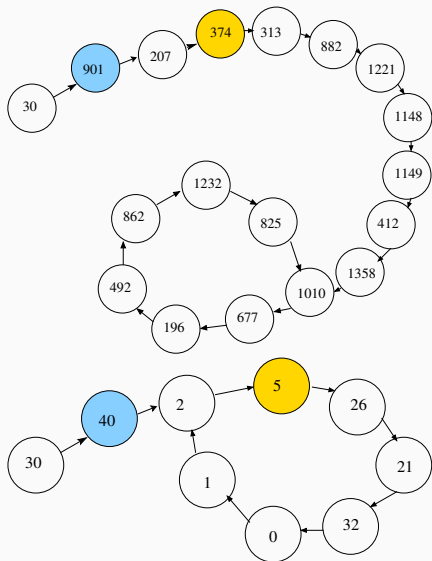
$g \leftarrow \text{GCD}(Y_i - X_i, n);$

$i \leftarrow i + 1;$

**return**  $g$

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

# Rho Pollard: illustration



$X_0 \leftarrow \text{rand}(), g \leftarrow 1, i \leftarrow 1;$

$Y_0 \leftarrow f(X_0);$

**while**  $g = 1$  **do**

$X_i \leftarrow f(X_{i-1});$

$Y_i \leftarrow f(f(Y_{i-1}));$

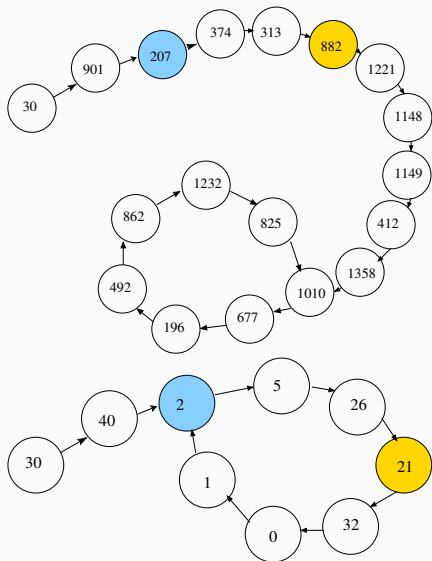
$g \leftarrow \text{GCD}(Y_i - X_i, n);$

$i \leftarrow i + 1;$

**return**  $g$

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

# Rho Pollard: illustration



$X_0 \leftarrow \text{rand}(), g \leftarrow 1, i \leftarrow 1;$

$Y_0 \leftarrow f(X_0);$

**while**  $g = 1$  **do**

$X_i \leftarrow f(X_{i-1});$

$Y_i \leftarrow f(f(Y_{i-1}));$

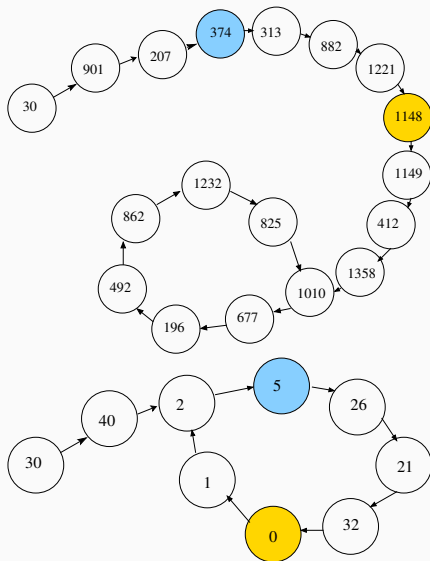
$g \leftarrow \text{GCD}(Y_i - X_i, n);$

$i \leftarrow i + 1;$

**return**  $g$

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

# Rho Pollard: illustration



$X_0 \leftarrow \text{rand}(), g \leftarrow 1, i \leftarrow 1;$

$Y_0 \leftarrow f(X_0);$

**while**  $g = 1$  **do**

$X_i \leftarrow f(X_{i-1});$

$Y_i \leftarrow f(f(Y_{i-1}));$

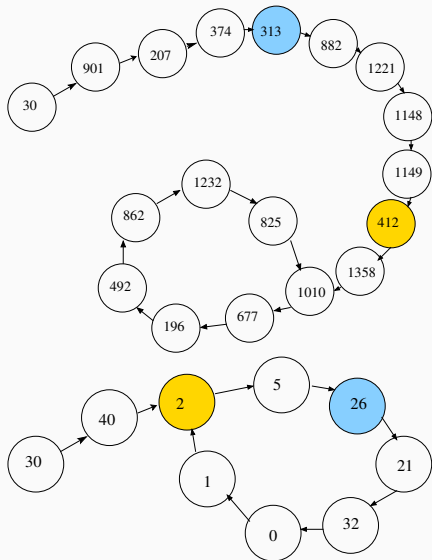
$g \leftarrow \text{GCD}(Y_i - X_i, n);$

$i \leftarrow i + 1;$

**return**  $g$

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

# Rho Pollard: illustration



$X_0 \leftarrow \text{rand}(), g \leftarrow 1, i \leftarrow 1;$

$Y_0 \leftarrow f(X_0);$

**while**  $g = 1$  **do**

$X_i \leftarrow f(X_{i-1});$

$Y_i \leftarrow f(f(Y_{i-1}));$

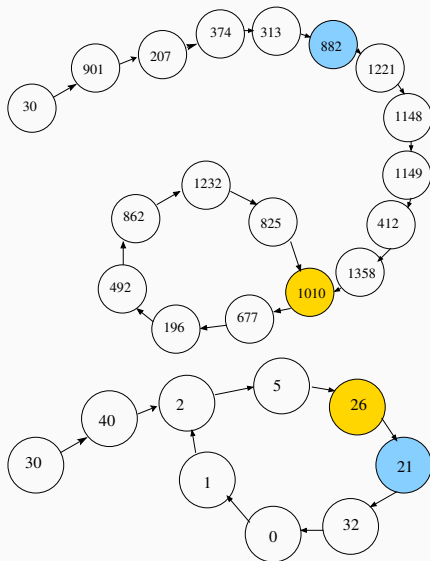
$g \leftarrow \text{GCD}(Y_i - X_i, n);$

$i \leftarrow i + 1;$

**return**  $g$

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

# Rho Pollard: illustration



$X_0 \leftarrow \text{rand}(), g \leftarrow 1, i \leftarrow 1;$

$Y_0 \leftarrow f(X_0);$

**while**  $g = 1$  **do**

$X_i \leftarrow f(X_{i-1});$

$Y_i \leftarrow f(f(Y_{i-1}));$

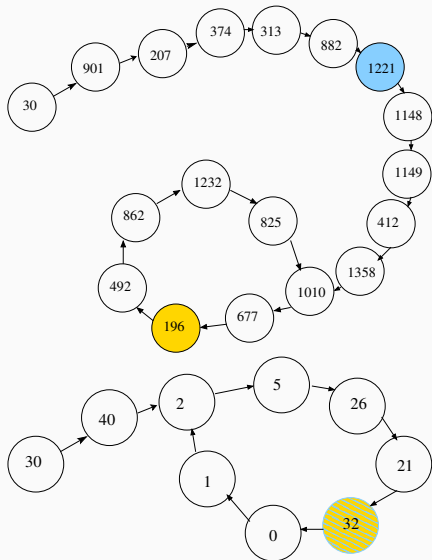
$g \leftarrow \text{GCD}(Y_i - X_i, n);$

$i \leftarrow i + 1;$

**return**  $g$

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

# Rho Pollard: illustration



$X_0 \leftarrow \text{rand}(), g \leftarrow 1, i \leftarrow 1;$

$Y_0 \leftarrow f(X_0);$

**while**  $g = 1$  **do**

$X_i \leftarrow f(X_{i-1});$

$Y_i \leftarrow f(f(Y_{i-1}));$

$g \leftarrow \text{GCD}(Y_i - X_i, n);$

$i \leftarrow i + 1;$

**return**  $g$

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

# 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  $a_0 = b_0 = 0$  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

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

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

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

What is a security property

One Way functions

Two fundamental one-way functions

Discrete Logarithm

Integer factorization

**Attacking the hard problems**

Baby-step / Giant-step

Rho Pollard's algorithm

**Index calculus algorithm**

Quadratic sieve

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

$$x = \log_g y = e_1 \log_g p_1 + e_2 \log_g p_2 + \dots + e_m \log_g p_m \pmod{p-1}$$

$\Rightarrow \log_g y$  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  $y = p_1^{e_1} p_2^{e_2} \dots p_m^{e_m} \pmod{p}$  then

$$x = \log_g y = e_1 \log_g p_1 + e_2 \log_g p_2 + \dots + e_m \log_g p_m \pmod{p-1}$$

$\Rightarrow \log_g y$  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

$$\log_g y = e_1 \log_g p_1 + e_2 \log_g p_2 + \dots + e_m \log_g p_m \pmod{p-1}$$

## 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

$$\log_g y = e_1 \log_g p_1 + e_2 \log_g p_2 + \dots + e_m \log_g p_m \pmod{p-1}$$

## 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

$$\log_g y = e_1 \log_g p_1 + e_2 \log_g p_2 + \dots + e_m \log_g p_m \pmod{p-1}$$

## 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  $S$ :  $y_k = \prod_{i=1}^m p_i^{e_{k,i}}$ 
  - $\Rightarrow k = \sum_{i=1}^m e_{k,i} \log_g p_i \pmod{p-1}$
  - $\Rightarrow$  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

$$\log_g y = e_1 \log_g p_1 + e_2 \log_g p_2 + \dots + e_m \log_g p_m \pmod{p-1}$$

## 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  $S$ :  $y_k = \prod_{i=1}^m p_i^{e_{k,i}}$ 
  - $\Rightarrow k = \sum_{i=1}^m e_{k,i} \log_g p_i \pmod{p-1}$
  - $\Rightarrow$  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

$$\log_g y = e_1 \log_g p_1 + e_2 \log_g p_2 + \dots + e_m \log_g p_m \pmod{p-1}$$

## 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  $S$ :  $y_k = \prod_{i=1}^m p_i^{e_{k,i}}$ 
  - $\Rightarrow k = \sum_{i=1}^m e_{k,i} \log_g p_i \pmod{p-1}$
  - $\Rightarrow$  one linear relation between the  $\log_g p_i$
- Iterate until we get  $m$  independent equations
- Solve the linear system

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

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

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

$$\log_g y = e_1 \log_g p_1 + e_2 \log_g p_2 + \dots + e_m \log_g p_m \pmod{p-1}$$

Problem:  $y$  very likely does not factor in  $S$

## Recovery of the unknown 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  
⇒ embarrassingly parallel

# Index calculus algorithm

- Most work is in the collection of relations  
⇒ embarrassingly 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  
⇒ embarrassingly 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  
⇒ embarrassingly 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  
⇒ embarrassingly 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.

What is a security property

One Way functions

Two fundamental one-way functions

Discrete Logarithm

Integer factorization

**Attacking the hard problems**

Baby-step / Giant-step

Rho Pollard's algorithm

Index calculus algorithm

Quadratic sieve

# The quadratic sieve

## Principle

Find  $x, y$  such that

$$\begin{cases} x^2 = y^2 \pmod{n} \\ x \not\equiv \pm y \pmod{n} \end{cases}$$

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

# The quadratic sieve

## Principle

Find  $x, y$  such that

$$\begin{cases} x^2 = y^2 \pmod{n} \\ x \not\equiv \pm y \pmod{n} \end{cases}$$

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  $x - y = 0 \pmod{n} \Rightarrow x = y \pmod{n}$ , absurd.
- if  $g = 1$  then  $x - y$  divides  $k$ :  $k = (x - y)k'$ ,  
and thus  $x + y = k'n = 0 \pmod{n} \Rightarrow x = -y \pmod{n}$ , absurd.



# The quadratic sieve

## Principle

Like for the Index calculus: 3 phases

1. Select a collection (basis) of small prime numbers  $\mathcal{B}$
2. Collect random numbers  $r_i$  which square factors in  $\mathcal{B}$ 
  - $r_i^2 = \prod_{j \in J_i} p_j$
3. Combine a selection of such numbers in order to form an identity  $x^2 = y^2 \pmod n$ 
  - $x = \prod_{i \in I} r_i$
  - $y = \prod_{i \in I} \prod_{j \in J_i} p_j = \prod_k p_k^{e_k} \Rightarrow$  selection so that each  $e_k$  is even.
  - very likely  $x \not\equiv \pm y \pmod n$

# The quadratic sieve

## Example (Factorization of 7429)

$$87^2 = 1 \times 7429 + 140 \text{ and } 140 = 2^2 \times 5 \times 7$$

$$88^2 = 1 \times 7429 + 315 \text{ and } 315 = 3^2 \times 5 \times 7$$

# The quadratic sieve

## Example (Factorization of 7429)

$$87^2 = 1 \times 7429 + 140 \text{ and } 140 = 2^2 \times 5 \times 7$$

$$88^2 = 1 \times 7429 + 315 \text{ and } 315 = 3^2 \times 5 \times 7$$

$$88^2 \times 87^2 = (2 \times 3 \times 5 \times 7)^2 \pmod{7429}$$

$$x = 87 \times 88 = 227 \pmod{7429}$$

$$y = 2 \times 3 \times 5 \times 7 = 210 \pmod{7429}$$

# The quadratic sieve

## Example (Factorization of 7429)

$$87^2 = 1 \times 7429 + 140 \text{ and } 140 = 2^2 \times 5 \times 7$$

$$88^2 = 1 \times 7429 + 315 \text{ and } 315 = 3^2 \times 5 \times 7$$

$$88^2 \times 87^2 = (2 \times 3 \times 5 \times 7)^2 \pmod{7429}$$

$$x = 87 \times 88 = 227 \pmod{7429}$$

$$y = 2 \times 3 \times 5 \times 7 = 210 \pmod{7429}$$

$$\text{GCD}(227 - 210, 7429) = 17$$

⇒

$$7429 = 17 \times 437$$

# 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

- $87 \times 88$  is a square iff  $L_2+L_3$  is even
- find a vector  $x \in \{0, 1\}^m$  s.t.  $x^T M$  is even
- find a vector  $x$  s.t.  $x^T M = 0 \pmod 2$
- 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  $S = \{p_1, p_2, \dots, p_t\}$
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  $v_i = (e_1, \dots, e_t)$  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  $x^T M = 0 \pmod 2$   
(e.g. solve  $x^T M = c$  for  $c = v^T M$  and  $v$  random and return  $x - v$ ).
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  $\text{GCD}(X - Y, n)$

## 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:  $153^2 = 280 \pmod{n}$  and  $197^2 = 2^6 \times 5 \times 7^2 \pmod{n}$  and  $586 \times 197 = 22926 \pmod{n}$ .
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:  $153^2 = 280 \pmod n$  and  $197^2 = 2^6 \times 5 \times 7^2 \pmod n$  and  $586 \times 197 = 22926 \pmod n$ .
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.  $197^2 = 2^6 * 5 * 7^2 \pmod n$

## 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:  $153^2 = 280 \pmod n$  and  $197^2 = 2^6 \times 5 \times 7^2 \pmod n$  and  $586 \times 197 = 22926 \pmod n$ .
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.  $197^2 = 2^6 * 5 * 7^2 \pmod n$

Hence  $(153 \times 155 \times 197)^2 = 2^{16} 5^2 7^4 = (2^8 \times 5 \times 7)^2 \pmod n$

Therefore  $\text{GCD}(153 \times 155 \times 197 - 2^8 \times 5 \times 7^2, n)$  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  $n$ .

## 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  $\log n$ .

## 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

## 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

## 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

## 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	$L_p[1/3, c]$	$L_p[1/3, c]$

## 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

## 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  $\approx 2$  years ( $> 1500$  core year)

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

**2019:** RSA 795 bits in  $> 900$  core year

**2020:** RSA 829 bits in  $> 2700$  core year

# 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

# Security parameters and orders of magnitude

From [Lenstra Kleinjung Thomé, 2013]

<sup>3</sup>	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 <sup>4</sup>	114	114	228	2380

<sup>3</sup>energy required to put that amount of water into boiling status

<sup>4</sup>year of solar energy

# Security parameters and orders of magnitude

From [Lenstra Kleinjung Thomé, 2013]

<sup>3</sup>	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 <sup>4</sup>	114	114	228	2380

DLP:

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

<sup>3</sup>energy required to put that amount of water into boiling status

<sup>4</sup>year of solar energy