

# Complexity theory and reductions

## Cryptographic Engineering

---

Clément PERNET

M2 Cyber Security,

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

ENSIMAG, Grenoble INP

# Asymmetric crypto and **Provable security**

## **Definition: one-way function**

A bijection (i.e. one-to-one mapping)  $f$  is **one-way** iff

- (i) It is easy to compute  $f(x)$  from  $x$ ;
- (ii) Computation of  $x = f^{-1}(y)$  from  $y = f(x)$  is intractable, i.e. requires too many operations, e.g.  $10^{36} \simeq 2^{120}$

## **How to prove one-way-ness?**

1. Analyze the time complexity of an algorithm that computes  $f$ .
2. Provide a lower bound on the minimum time complexity to compute  $x = f^{-1}(y)$  given  $y$ 
  - very hard to obtain lower bounds in complexity theory
  - it is related both to the problem  $f^{-1}$  and the input  $y$  (i.e.  $x$ )

## **Provable security** [*Contradiction proof, by reduction*]

If computation of  $f^{-1}$  is feasible, then a well-studied and presumed intractable problem could be solved.

P, NP classes and reduction

One way function and asymmetric cryptography

# Definitions : P, NP

Context: Decision problems (answer  $\in \{\text{YES}, \text{NO}\}$ )

## Definition (Class P)

A problem is in the complexity class **P** if there is an algorithm  $A(x)$  that solves it on every instance  $x$  in time polynomial in  $|x|$  (the size of  $x$ ).

# Definitions : P, NP

Context: Decision problems (answer  $\in \{\text{YES}, \text{NO}\}$ )

## Definition (Class P)

A problem is in the complexity class **P** if there is an algorithm  $A(x)$  that solves it on every instance  $x$  in time polynomial in  $|x|$  (the size of  $x$ ).

- closed under composition and polynomially bounded iterations
- Informally:

**P** = set of problems efficiently solvable.

## Definition (Class NP)

A decision problem  $F$  is in the complexity class **NP** if there is a polynomial time algorithm  $V(x, y)$  such that

- $F(x) = 1 \Rightarrow \exists z$  of size poly in  $|x|$  s.t.  $V(x, z) = 1$
- $F(x) = 0 \Rightarrow \forall y V(x, y) = 0$

## Definition (Class NP)

A decision problem  $F$  is in the complexity class **NP** if there is a polynomial time algorithm  $V(x, y)$  such that

- $F(x) = 1 \Rightarrow \exists z$  of size poly in  $|x|$  s.t.  $V(x, z) = 1$
  - $F(x) = 0 \Rightarrow \forall y V(x, y) = 0$
- 
- Set of problems which YES answer can be **verified** in poly-time
  - Informally: **NP** = Set of pb efficiently verifiable.
  - The second input  $y$  is called a certificate.

# Definitions : P, NP

## Definition (Class NP)

A decision problem  $F$  is in the complexity class **NP** if there is a polynomial time algorithm  $V(x, y)$  such that

- $F(x) = 1 \Rightarrow \exists z$  of size poly in  $|x|$  s.t.  $V(x, z) = 1$
- $F(x) = 0 \Rightarrow \forall y V(x, y) = 0$
  
- Set of problems which YES answer can be **verified** in poly-time
- Informally: **NP** = Set of pb efficiently verifiable.
- The second input  $y$  is called a certificate.
- $\mathbf{P} \subset \mathbf{NP}$  but  $\mathbf{NP} \subset \mathbf{P}$  is a 1M\$ question
- **co-NP**: class of problems  $F$  which complement is in **NP**

## Example (IS\_COMPOSITE( $n$ ) $\in$ NP)

IS\_COMPOSITE( $n$ )=YES  $\Leftrightarrow n$  is a composite number

### Verification

- certificate: a number  $a \notin \{n, 1\}$  such that  $a|n \Leftrightarrow n = 0 \pmod a$
- $V(n,a)\{\text{return } (n \pmod a == 0)\}$
- Cost of verification  $(n \pmod a == 0)$  in time  $O(\log_2(n)^2)$

## Example ( $\text{PLOG}_G(x,t) \in \text{NP}$ )

Over a group  $G$  generated by  $g$ ,

$$\text{PLOG}_G(x,t)=\text{YES} \Leftrightarrow \exists i, g^i = x \text{ and } t \leq i < \#G$$

Verification

- certificate: the index  $i$  such that  $g^i = x$
- $V(x, t, i) \{y = g^i ; \text{return } (x==y \ \&\& \ i \geq t)\}$
- Cost of verification  $g^i$  in time  $O(\log(|G|)^3)$

## Definition (Karp reduction for decision problems)

$A \leq_P^{(Karp)} B$  ( $A$  is reducible to  $B$ ) if there is a polynomial time algorithm transforming an input  $x$  for  $A$  into an input  $y$  for  $B$  such that  $A(x) = B(y)$ .

## Definition (Karp reduction for decision problems)

$A \leq_P^{(Karp)} B$  (A is reducible to B) if there is a polynomial time algorithm transforming an input  $x$  for  $A$  into an input  $y$  for  $B$  such that  $A(x) = B(y)$ .

**Oracle:** For a problem  $B$ , an oracle is a imaginary method returning any answer  $B(x)$  in constant time.

## Definition (Turing/Cook reduction)

$A \leq_P^{(Turing)} B$  (A is reducible to B) if there is an algorithm computing  $A(x)$  in a polynomial number ( $|x|^{O(1)}$ ) of operations and calls to an oracle for  $B$ .

# P-reduction

## Definition (Karp reduction for decision problems)

$A \leq_P^{(Karp)} B$  (A is reducible to B) if there is a polynomial time algorithm transforming an input  $x$  for  $A$  into an input  $y$  for  $B$  such that  $A(x) = B(y)$ .

**Oracle:** For a problem  $B$ , an oracle is a imaginary method returning any answer  $B(x)$  in constant time.

## Definition (Turing/Cook reduction)

$A \leq_P^{(Turing)} B$  (A is reducible to B) if there is an algorithm computing  $A(x)$  in a polynomial number ( $|x|^{O(1)}$ ) of operations and calls to an oracle for  $B$ .

## Property

If  $A \leq B$  Then

- $B \in \mathbf{P} \Rightarrow A \in \mathbf{P}$ .
- $A \notin \mathbf{P} \Rightarrow B \notin \mathbf{P}$ .

## Property

$\leq_P^{(Turing)}$  and  $\leq_P^{(Karp)}$  are transitive

# Reductions

## Example ( $\text{PLOG}_G \leq_P \text{LOG}_G$ )

Algorithm PLOG\_Reduction ( $G, x, \text{int } t$ )

1.  $\text{log} = \text{OracleLOG}(x)$ ;
2. return ( $\text{log} \geq t$ );

Since  $0 \leq t, \text{log} \leq \#G$ , and cost of OracleDLOG is constant, cost of PLOG\_Reduction is  $O(\log \#G)$ .

# Reductions

## Example ( $\text{PLOG}_G \leq_P \text{LOG}_G$ )

Algorithm  $\text{PLOG\_Reduction}(G, x, \text{int } t)$

1.  $\text{log} = \text{OracleLOG}(x)$ ;
2. return  $(\text{log} \geq t)$ ;

Since  $0 \leq t, \text{log} \leq \#G$ , and cost of OracleDLOG is constant, cost of  $\text{PLOG\_Reduction}$  is  $O(\log \#G)$ .

## Example ( $\text{LOG}_G \leq_P \text{PLOG}_G$ )

Algorithm  $\text{LOG\_Reduction}(G, x)$  // Binary search

1.  $\text{min} = 0$ ;  $\text{max} = \#G$ ;
2. while  $(\text{min} < \text{max})\{$
3.      $\text{mid} = (\text{min} + \text{max}) / 2$ ;
4.     If  $\text{OraclePLOG}(x, \text{mid})$   $\text{min} = \text{mid}$ ; else  $\text{max} = \text{mid}$ ; }
5. return  $\text{min}$ ;

Cost  $O(\log^2 \#G)$

# NP hardness NP completeness

## Property

**NP** is closed under  $\leq_P^{(Karp)}$  :

$$\left\{ \begin{array}{l} A \leq_P^{(Karp)} B \\ B \in \mathbf{NP} \end{array} \right. \Rightarrow A \in \mathbf{NP}.$$

## Definition

A is **NP-hard** if  $\forall X \in \mathbf{NP} \quad X \leq_P^{(Karp)} A$

## Definition

**NP-complete** = **NP-hard**  $\cap$  **NP**

## Theorem (Cook)

**NP-complete**  $\neq \emptyset$  as 3 - SAT  $\in$  **NP-complete**

# NP-intermediate

## Definition (NP-intermediate)

Problems that are neither in **P** nor in **NP-complete**.

## Theorem (Ladner)

If  $P \neq NP$  then  $NP\text{-intermediate} \neq \emptyset$

# NP-intermediate

## Definition (NP-intermediate)

Problems that are neither in **P** nor in **NP-complete**.

## Theorem (Ladner)

If  $P \neq NP$  then  $NP\text{-intermediate} \neq \emptyset$

## Examples

Good candidates for **NP-intermediate** problems:

- Graph isomorphism
- Integer factorization
- Discrete logarithm

P, NP classes and reduction

One way function and asymmetric cryptography

# One way functions

## Definition (One way function)

Function  $E : \{0, 1\}^n \rightarrow \{0, 1\}^n$  that are

- injective
- easy to compute
- hard to invert:  $x \leftarrow E^{(-1)}(y)$  should be computationally hard

# One way functions

## Definition (One way function)

Function  $E : \{0, 1\}^n \rightarrow \{0, 1\}^n$  that are

- injective
- easy to compute
- hard to invert:  $x \leftarrow E^{(-1)}(y)$  should be computationally hard

## Remark

- *Easy to compute*  $\Rightarrow E \in \mathbf{P}$
- *Therefore*  $D = E^{-1} \in \mathbf{NP}$
- *Hence if one way functions exist then*  $\mathbf{P} \neq \mathbf{NP}$

# One way functions

## Definition (One way function)

Function  $E : \{0, 1\}^n \rightarrow \{0, 1\}^n$  that are

- injective
- easy to compute
- hard to invert:  $x \leftarrow E^{(-1)}(y)$  should be computationally hard

## Remark

- *Easy to compute*  $\Rightarrow E \in \mathbf{P}$
- *Therefore*  $D = E^{-1} \in \mathbf{NP}$
- *Hence if one way functions exist then*  $\mathbf{P} \neq \mathbf{NP}$

Design one way functions by basing  $D$  on difficult problems in **NP**:

- **NP-complete**: subset-sum, knapsack, (Merkle-Hellman, Chor-Rivest)
- **NP-intermediate**: factorization (RSA), discrete log (El-Gamal)

# One way trapdoor functions

Make decryption practical: add a parameter (secret key)

## Definition

- $E$  is one way
  - $E$  is easy to compute
  - $D$  such that  $D(E(x)) = x$  is hard to compute
- but given a trapdoor (secret key),  $D$  is easy to compute

## Example: Knapsack [Merkle Hellman 78]

### Problem (Subset sum $\in$ NP-complete)

**Input:**  $(a_1, a_2, \dots, a_n)$  and  $S$  integers

**Output:** Yes iff  $\exists (x_1, \dots, x_n) \in \{0, 1\}^n, \sum_{i=1}^n x_i a_i = S$

## Example: Knapsack [Merkle Hellman 78]

### Problem (Subset sum $\in$ NP-complete)

**Input:**  $(a_1, a_2, \dots, a_n)$  and  $S$  integers

**Output:** Yes iff  $\exists (x_1, \dots, x_n) \in \{0, 1\}^n, \sum_{i=1}^n x_i a_i = S$

Idea for encoding:  $E(x_1, \dots, x_n) = \sum_{i=1}^n x_i a_i$

## Example: Knapsack [Merkle Hellman 78]

### Problem (Subset sum $\in$ NP-complete)

**Input:**  $(a_1, a_2, \dots, a_n)$  and  $S$  integers

**Output:** Yes iff  $\exists (x_1, \dots, x_n) \in \{0, 1\}^n, \sum_{i=1}^n x_i a_i = S$

Idea for encoding:  $E(x_1, \dots, x_n) = \sum_{i=1}^n x_i a_i$

### Building a trapdoor function

- Easy to solve instance: choose  $(a_1, \dots, a_n)$  super-increasing  
i.e.  $a_i > \sum_{j=1}^{i-1} a_j$ . which algorithm?
- Hiding simplicity:  $b_i = ta_i \pmod m$  with  $t$  and  $m > S$  secret and coprime

## Example: Knapsack [Merkle Hellman 78]

### Problem (Subset sum $\in$ NP-complete)

**Input:**  $(a_1, a_2, \dots, a_n)$  and  $S$  integers

**Output:** Yes iff  $\exists (x_1, \dots, x_n) \in \{0, 1\}^n, \sum_{i=1}^n x_i a_i = S$

Idea for encoding:  $E(x_1, \dots, x_n) = \sum_{i=1}^n x_i a_i$

### Building a trapdoor function

- Easy to solve instance: choose  $(a_1, \dots, a_n)$  super-increasing  
i.e.  $a_i > \sum_{j=1}^{i-1} a_j$ . which algorithm?
- Hiding simplicity:  $b_i = ta_i \pmod m$  with  $t$  and  $m > S$  secret and coprime

**Public key:**  $(b_1, \dots, b_n)$ .

**Encryption:**  $C = E(x_1, \dots, x_n) = \sum_{i=1}^n x_i b_i \pmod m$

**Private key:**  $(a_1, \dots, a_n), t, u = t^{-1} \pmod m$

**Decryption:**  $C.u \pmod m$  and solve the easy problem with the  $a_i$ 's.