Outline Lecture

- Part 1 : Asymmetric cryptography, one way function, complexity
- Part 2 : arithmetic complexity and lower bounds : exponentiation
- **Part 3 : Provable security and polynomial time reduction :**
  - **P, NP classes. One-way function and NP class.**
    1. **NP : definition ,examples**
    2. **P-reduction, NP-hard, NP-complete, NP-intermediate**
    3. **Relationship between asymetric cryptography and NP**
- Part 4 : RSA : the algorithm
- Part 5 : Provable security of RSA
- Part 6 : Attacks and importance of padding.

Polynomial reduction

- Lecture: Polynomial reduction
  - Very short « remind » about P and NP
- Example:
  - Least significant bit of LOG versus all bits of LOG
    - LSB in a cyclic group: input x, output YES iff LOG(x) is odd
- Exercise n. 2 / Form 2:
  - Primes, Big Factor and Factorization

Non deterministic polynomial time

- Problem F is in ***P*** if there is an algorithm A(x) that computes F(x) on input x in time polynomial in the input size, |x|.
  - P is closed under composition and polynomially bounded iterations.
- Decision problem F is in ***NP*** if for all x such that F(x) holds, there exists a polynomial sized certificate c(x) and a verifying algorithm V(x, y) such that V(x, c(x)) computes F(x) in time polynomial in |x|.
  - NP contains P . It is not known if P=? NP
  - Co-NP definition: F is in ***co-NP*** iff Complement(F) is in NP.

Example 1 : discrete log

- G = {g^i, i=0, …, n-1} a cyclic group of order n
- Problem $\mathcal{LOG}_G$:
  - Input: x in G ;
  - Output : $0 \le i < n$ such that $g^i = x$.
- Decision Problem $\mathcal{PLOG}_G$:
  - Input: x in G and an integer t ($0 \le t < n$) ;
  - Output : YES iff $\mathcal{LOG}_G(x) \ge t$.

PLOG is in NP

- PLOG(x, t) = YES iff it exists $0 \le i < n$ such that $g^i = x$ and $x \ge t$.
- PLOG is in NP:
  - Certificate : i an integer
  - Verifying algorithm V(x, t, i) {
    y = BinaryPower(g,i);
    if (y==x) and (i ≥ t) return « OK: PLOG(x,t) is proved »;
    }
  - *Algorithm V is a verifying algorithm for PLOG*
    - Proof: V(x,t,i) returns OK ⇔ PLOG(x,t) = YES
  - *Algorithm V runs in time polynomial in |x|+|t| for all input (x,t) satisfying PLOG(x,t)=YES*
    - Proof: if PLOG(x,t)=YES it exists a polynomial sized certificate i with $|i| \le \log_2 n = |x|$ and V(x,t,i) requires at most O(|i|+|x|+|t|) operations.

NP-class equivalent definitions

- **Def 1.** NP = set of decision problems Q which YES output is verified by a deterministic polynomial time:
  - There exists an algorithm VerificationQ(x , z) :
    - For all x such that Q(x)=YES, it exists z such that VerificationQ(x , z) returns "Q(x)=YES is proved" in polynomial time.
    - For all x such that Q(x)=NO, for all z, VerificationQ(x , z) never returns "Q(x)=YES is proved".
- **Def 2.** NP = set of Decision problems Q that admit a **N**on-deterministic **P**olynomial-time algorithm:
  - If Q output=YES, at least one path returns YES
  - If Q output=NO, no path returns YES
    - (i.e., any path returns NO or infinitely loops)

Non-determinstic polynomial-time algorithm: an example

- Decision problem $PLOG_G$(x, t)
  - Input: w in G and an integer t
  - Output : YES iff it exists i : $g^i = x$ and $i \ge t$.
- NDetAlgo_PLOG (x, t) {

```
  Int i = nonderministic_choice (0, .., |G|-1) ;
  y = BinaryPower( g, i ) ;
  if ( y == x ) return YES ;
  else { while (1) ; /* infinite loop */
}
```

Non-determinstic polynomial-time algorithm: an example

- Decision problem IS_COMPOSITE (N)
  - Input: an integer N
  - Output : YES iff N is composite
- NDetAlgo_IsComposite (N) {

```
  Int a = nonderministic_choice (1, .., √N) ;
  if ( N mod a == 0 ) return YES ;
  return NO;
}
```

- *Remark: another proof*: PRIME is in P.
  So IS_COMPOSITE is in P_{DEC}, which is included in NP.

P-reduction, NP-Hard, NP-Complete

- Let A and B be two problems.
  OracleB(x): oracle that computes B(x) in time |x|.
- **Def**: Polynomial Reduction: A $\leq_P$ B iff there exists an algorithm Algo A that computes A(x) in polynomial time using standard operations (DTM or RAM model) and oracles for B.
  Note: This polynomial reduction is named « Turing-reduction » or « Cook-reduction »)

$PLOG_G \leq_P LOG_G$

- **Algorithm** PLOG_reduction (G x, Int t)

```
{   logx = OracleLOG( x ) ;
    if (logx ≥ t) return YES else return NO;
}
```

- Assuming cost of OracleLOG is constant, and since $0 \leq$ logx $< n$ and $0 \leq t < n$, cost of PLOG_reduction is O(*log* n).
- Thus $PLOG_G \leq_P LOG_G$.

$LOG_G \leq_P PLOG_G$

- **Algorithm** LOG_reduction (G x)

```
{   // computation by binary search in [min, max(
    min = 0 ; max = n ;
    while (min < max)
    { mid = (min + max ) / 2 ;
      if ( OraclePLOG( x, mid)) { min=mid;}  else {max=mid;};
    }
    return min;
}
```

- Cost including calls to the Oracle: O(log^2 n),
  which is polynomial in the input size (|x| = *log* n).
- Thus $LOG_G \leq_P PLOG_G$

Relation between PLOG and LOG

- Theorem: if LOG_G is computationally impossible, then $PLOG_G$ is computationally impossible too.
  - Proof:
- Variants [exercise]:
  - Least significant bit: PLOG-LSB
    Let PLOG-LSB(x) = YES iff LOG(x) mod 2=1.
  - Highest significant bit : PLOG-HSB
    Let PLOG-LSB(x) = YES iff LOG(x) ≥ ($\log_2$ n-1)/2.

NP class and $\leq_{P\_Karp}$ reduction

- Prop. NP is closed under $\leq_{P\_Karp}$
  - i.e. ($A \leq_{P\_Karp} B$ and $B \in NP$) => $A \in NP$.
- Def. A decision problem Q is **NP-hard** iff
  $\forall X \in NP : \quad X \leq_{P\_Karp} Q$.
- Def. NP-complete = NP ∩ NP-Hard
- Theo: SAT∈NP-complete.
  - Def: SAT(F : boolean formula)=YES iff F is not always false.
  - Moreover, 3-SAT∈NP-complete (but 2-SAT∈P)
- Def. coNP: Q∈coNP iff ¬Q∈NP
  - Def: TAUT(F : boolean formula)=YES iff F is always true.
  - Theo: TAUT∈coNP-complete

P-reduction, NP-Hard, NP-Complete

- Let A and B be two problems.
  OracleB(x): oracle that computes B(x) in time |x|.
- **Def**: Polynomial Reduction: $A \leq_P B$ iff there exists an algorithm Algo A that computes A(x) in polynomial time using standard operations (DTM or RAM model) and oracles for B.
  Note: This polynomial reduction is named « Turing-reduction » or « Cook-reduction »)
- **Remark:** The reduction $\leq_P$ is used for security proofs;
  but it is different from the « standard » *many-to-one* reduction (*Karp-reduction*).
  - With Turing reduction: NP $=_P$ co-NP (but open question with Karp reduction)
  - With Turing reduction: it is not known wether NP is closed or not (but NP is closed under Karp-reduction

  This affects the below (non standard) definition of NP-Hard and NP-complete:
  - **Def**: Q is **NP-hard** iff, $\forall X \in NP : X \leq_P Q$
    Q **NP-complete** iff both Q is NP-hard and Q ∈ NP.
  - **Cook theorem** : NP-complete ≠ ∅. SAT and 3-SAT are NP-complete.

NP - Intermediate

- Def: NP-intermediate == problems that are neither in P nor NP-complete.
  - Theorem: If P≠NP, NP-intermediate ≠ ∅
- Good candidates for NP-intermediate problems:
  - P_LOG_G∈NP-intermediate
    - DISCRETE_LOGARITHM $\leq_P$ PLOG [See exercise sheet 2]
  - HAS_BIG_FACTOR ∈NP-intermediate
    - INTEGER_FACTORIZATION $\leq_P$ HAS_BIG_FACTOR [See exercise sheet 2]
  - Graph isomorphism

One-way function and *NP class*

- $E : \{0,1\}^n \rightarrow \{0,1\}^n$ (or $Im(E) \subset \{0,1\}^{n+1}$)
  injective (one-to-one mapping),
  *and **easy to compute** i.e. ~linear time to compute E(X)*
- $D = E^{-1}$: should be computationally impossible
- Does such functions exist? Anyway:
  - E « easy » to compute ⇒ E ∈ *P*
  - Then, since $D=E^{-1}$ ⇒ D ∈ *NP* (non-deterministic)
  - Note: if one-way functions exist, P≠NP
- Then, look for a convenient D among the most difficult problems inside NP… conjectured intractable
  - NP-complete ones: eg subset sum/knapsack [Merkle-Hellman, Chor-Rivest…]
  - Conjectured computationally imposible ones: factorization…

Some «hard » problems used to build one-way functions

- **Subset sum** [NP-complete]
  - Input : S, $(a_1, \ldots, a_n)$; - Output : $(x_1, \ldots, x_n) \in \{0,1\}^n$: $\sum_{i=1}^{n} x_i a_i = S$
- **Discrete logarithm** (NP-intermediate)
  - Input : g, M ; - Output : x such that $g^x = M$
- **Factorization** (NP-intermediate)
  1. Input : N - output : factorization of N
  2. Input: N, M, C ; - output : d s.t $M^d = C \bmod N$
  3. Input : N, e, C ; - output : M s.t. $M^e = C \bmod N$
  4. Input : N, x ; - output : YES iff ∃ y such that $x = y^2 \bmod N$

Example 1 : « Exponential and Discrete logarithm »

- (G, *) : cyclic group of order n; g a generator of G
  - $G = \{ g^i ; i = 0, \ldots, n-1 \}$
- **Exponential** : Exp: { 0, …, n-1} → G defined by $Exp(i) = g^i$

  Computation cost of Exp (i) = O(log (i)) = **O(log n)** [upper and lower bound, lect2]
  Example : 5^{11} [7] = $((5^2)^2\, 5)^2\, 5 = ((4)^2\, 5)^2\, 5 = (2.5)^2\, 5 = 2.5 = 3$
- **Discrete Logarithm**: Log : G → { 0, …, n-1} defined by Log(x) =i s.t. $x= g^i$
  Example : find x / $6^x = 8$ [11]

  (Answer: x = 7)

  Best known algorithms for any G in **O($n^{0.5}$)** [Shanks]
  - Note : INTEGER-FACTORIZATION ≤P DISCRETE-LOGARITHM

  Conjectured hard to compute :
  - Very used in asymmetric cryptography: ex RSA, El Gamal, ECDLP
  - **But** : some specific instances are easy to compute

One-way *trapdoor* function

- Definition:
  - E is one-way
  - D(E(x)) = x [and E(D(x)) = x for signature]
  - But, given a trapdoor (the secret key),
    D is *easy* to compute (almost linear time)
- Provable security:
  - Given c = E(x), computing x is untractable
  - How to prove it? By reduction (contractiction) !
    - assume there exists an algorithm to compute x from c
    - then exhibit an algorithm that computes an untracatable problem !

Example 2 : « knapsack » [Merkle-Hellman,78]

- SUBSETSUM ∈ *NP* -complete
  - Input : $(a_1, \ldots, a_n)$ and *S* integers
  - Output : YES iff it exists $(x_1, \ldots, x_n) \in \{ 0,1 \}^n$: $\sum_{i=1}^{n} x_i a_i = S$
- Idea for an encoding: $E(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i a_i$
- Building a trapdoor function
  - Easy to solve instance; choose $(a_1, \ldots, a_n)$ ***super-increasing.***
    - What is the decoding algorithm?
  - Hiding simplicity $b_i = t.a_i \bmod m$ with t secret and prime to m
  - Public : $(b_1, \ldots, b_n)$ and m : $E(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i b_i \bmod m$
  - Secret : $(a_1, \ldots, a_n)$, t and $u = t^{-1} \bmod m$:
    - Decoding: just compute (S.u mod n) and decode from $(a_1, \ldots, a_n)$

Outline lecture 2

- $P \subset NP \subset$ NP-complete $\subset$ NP-hard
- The (polynomial) complexity of E bounds the complexity of D :

  $(E \in P) \Rightarrow (D \in NP)$
- Conjecture for asymetric cryptography:$P \neq NP$
  - so asymetric cryptograpy is based on NP-intermediate problems.
- Discrete LOG has a complexity polynomially equivalent to LSB_LOG.