Complexity Theoretic Cryptography

JASS 2005

Stefan Neukamm

June 7, 2005

Abstract

In this work we present the basic concept of complexity theoretic cryptography. Definitions of one-way functions, collections of one-way functions, collections of trapdoor functions, hard-core predicates will be discussed and we give a brief introduction to complexity theory.

1 Introduction - The two approaches to cryptography

Traditionally, cryptography has been associated with the problem of providing *secret* communication over insecure media. The setting of this problem consists of two parties communicating through a channel which is possibly tapped by an adversary. To keep the 'wiretapper' as ignorant as possible regarding the contents of the exchanged information, the parties will use a protocol, the encryption scheme, that allows them to communicate *secretly*.

The most important task of cryptography is designing and analysing encryption schemes. A preliminary task is to give the intuitive terms 'security' and 'adversary' proper mathematical definitions. Two approaches to defining security are known.

The classical approach by Shannon in 1949 is *information theoretic*. It is concerned with the information about the plaintext which is present in the ciphertext. Loosely spoken, an encryption scheme is secure if there is no information about the plaintext present in the ciphertext. In his theory Shannon showed that - given an adversary, who is assumed to have unlimited computational resources - secure encryption systems can exist only if the secret key in use is at least as large as the the information to be ever exchanged using the encryption system.

The modern approach, followed in this paper, is based on *computational complexity*. It abandons the assumption of an adversary, supplied with unlimited computational resources and assumes that the adversary's computation is bounded in some reasonable way. In this setting the important question is not if there is plaintext information present in the ciphertext, but rather if this information can be *efficiently* extracted. In the complexity-theoretic approach security is based on a gap between efficient algorithms guaranteed for the legitimate users versus the computational infeasibility of retrieving information for an adversary.

Designing such systems requires the existence of computational *primitives* with certain kinds of computational hardness properties. The most basic primitive is a *one-way* function, which loosely spoken is a function, that is easy to compute but hard to invert.

This paper will discuss the concept of *one-way* and *trapdoor functions*, following the description in [GB01] and [Gol01]. The required background on computational complexity will be briefly introduced. We will discuss *necessary assumptions*, which have to be made for the existence of one-way functions and for proving the one-wayness of common candidates, like RSA or discrete logarithm. Finally the concept of *hard-core predicates* will be introduced and we will construct a *generic hard core predicate* for an arbitrary one-way function.

2 Complexity theory

2.1 Basic definitions

Definition 2.1 (Algorithm)**.** An (deterministic) *algorithm* **A** is a well-defined computational procedure that takes a variable input x and halts with an output, denoted by $\mathbf{A}(x)$.

Notation 2.2. The *size* or *length* of an input or output, denoted by $|\cdot|$ is the total number of bits needed for its ordinary binary notation.

Remark 2.3.

- The term 'well-defined computational procedure' can be formalized by using formal computational models like Turing machines or boolean circuits.
- The number of bits in the binary representation of a positive integer is $|n| = 1 + \lfloor \log_2 n \rfloor$.

Definition 2.4 (Running time)**.** For an algorithm **A** and an arbitrary input x we define the *running time*

$$Time_{\mathbf{A}}(x)$$

by the number of primitive operations or 'steps' executed. For an algorithm **A** and an input length $n \in \mathbb{N}$ we define the *worst case running time*

$$time_{\mathbf{A}}(n) := \max\{Time_{\mathbf{A}}(n) : \ |x| = n\}$$

as the maximum running time of **A** over all inputs with length n.

Normally one is interested in the worst case running time of an algorithm for large input. A useful notation to describe the asymptotic running time is the $\mathcal{O}$-notation:

Definition 2.5. If f is a non-negative function and **A** an algorithm, we write

$$time_{\mathbf{A}}(n) = \mathcal{O}(f(n)) \quad \Leftrightarrow \quad \exists c > 0, \exists n_0 \in \mathbb{N} : \ \forall n > n_0 : \ time_{\mathbf{A}}(n) \leq cf(n).$$

That means the worst case running time grow no faster asymptotically than f to within a constant multiple.

Definition 2.6 (Polynomial time algorithm)**.** An algorithm **A** is called *polynomial time algorithm* (**PT A**), if there exists an polynomial p such that

$$time_{\mathbf{A}}(n) \leq p(n) \qquad \forall n \in \mathbb{N}$$

Remark 2.7. Clearly an algorithm **A** runs in polynomial time iff

$$time_{\mathbf{A}}(n) = \mathcal{O}(n^k)$$

for a positive constant k.

Definition 2.8 (Probabilistic algorithm)**.** A probabilistic or randomized algorithm **A** is an algorithm which makes random decisions at certain points in its execution. Clearly the output of $\mathbf{A}(x)$ must be interpreted as a random variable, where the probability is token over all internal random decisions (sometimes called 'internal coin tosses') made during the execution. $\mathbf{Pr}[\mathbf{A}(x) = y]$ is the probability of y being the output of **A** on input x. If the worst case running time of **A** can be bounded by a polynomial, then **A** is called a probabilistic polynomial time algorithm **PPT A**.

Example 2.9. For illustration, let us compare the running time of three algorithms, deciding if an integer is prime or composite:

1. Sieve of Eratosthenes: This algorithm tests primality by trial division. Clearly if an integer n is not dividable by $2, 3, ..., \lceil\sqrt{n}\rceil$ then n is prime. For verifying primality approximately $\sqrt{n}$ divisions have to be done. Expressed in the length of the input n we get

   $$time_{\mathbf{Eras}}(n) = \mathcal{O}(2^{0.5|n|}).$$

   That means, Eratosthenes' sieve runs in exponential time.

2. Agrawal-Kayal-Saxena AKS algorithm [AKS02] is the only known deterministic polynomial time algorithm and runs assymptotically in $\mathcal{O}(|n|^{11.913})$.

3. Miller-Rabin Primality Test is a randomized algorithm, based on Fermats little Theorem and quadratic residues. It runs in $\mathcal{O}(k|n|^2)$ with an error less than 4^{-k}.

Table 2.1 shows the asymptotic behaviour of the three introduced primality tests. Clearly only the Miller-Rabin primality test is fast enough for real computations. It is used to produce so called 'industrial-grade primes'. The Miller-Rabin test recognizes every prime, but with an error less than 10^{-12} for $k := 20$ the test certifies composite integers to be prime.

n	$2^{0.5\|n\|}$	$\|n\|^{12}$	$k\|n\|^2$; $k := 20$
10 Bit	32	10^{12}	$2 \cdot 10^3$
100 Bit	10^{15}	10^{24}	$2 \cdot 10^5$
1000 Bit	$3 \cdot 10^{149}$	10^{36}	$2 \cdot 10^7$

Table 1: Asymptotic behaviour

Although AKS is not of practical use, because of the big exponent in its asymptotic behaviour, it is the only known deterministic polynomial time primality test and so from great theoretical interest.

2.2 Complexity classes

A main goal of the complexity theory is to classify problems by measuring their computational 'difficulty'. There are different ways to characterize the difficulty. Here we will introduce the most common one, the time complexity. For simplicity the complexity theory restricts its attention to very easy problems, the decision problems, which compute functions with value 0 or 1. This is not to too restrictive in practise, as all the problems that will be encountered here can be reduced to a decision problem of the corresponding complexity class.

Definition 2.10 (Decision problem)**.** An algorithm **A** computes the decision problem given by a set L if

$$\mathbf{A}(x) = \chi_L(x) := \begin{cases} 1 & x \in L \\ 0 & x \notin L \end{cases}$$

Definition 2.11 (**P**-class)**.** The complexity class **P** is the class of all decision problems which are decidable by a **PT** algorithm.

Example 2.12 (Prime)**.** The decision problem given by

$$\textsc{Prime} := \{p : \ p \in \mathbb{N}, \, p \text{ is prime}\}$$

lies in **P**, since the AKS algorithm works in polynomial time.

Definition 2.13 (**NP**-class)**.** The complexity class **NP** is the class of all decision problems L for which there exists a **PT A**, such that

- for all inputs $x \in L$, there exists a witness y with length bounded by a polynomial $p(|x|)$, such that $\mathbf{A}(x, y) = \chi_L(x) = 1$.

Example 2.14.

- $\mathbf{P} \subset \mathbf{NP}$
- For $f : \{0,1\}^* \to \{0,1\}^*$ define:
  $\textsc{Inverse}_f := \{(y, 1^n) : \ \exists x \in \{0,1\}^n \text{ s.t. } f(x) = y\}$.
  If f is computable in polynomial time, then $\textsc{Inverse}_f \in \mathbf{NP}$:
  Let $(y, 1^n) \in \textsc{Inverse}_f$, then
  - $\exists x \in \{0,1\}^n$ s.t. $f(x) = y$
  - $|(y, 1^n)| = n + |y| \geq |x|$. (Length of x is polynomially bounded.)
  - Since $f(x) = y$, we can verify the membership of $(y, 1^n) \in \textsc{Inverse}_f$ in $time_{\mathbf{f}}(n)$.

  $\Rightarrow \textsc{Inverse}_f \in \mathbf{NP}$

Now we like to introduce the randomized complexity class **BPP**. Let L be a set and $\mathbf{A}$ a randomized algorithm for the decision problem given by L. As mentioned in 2.8 for every input x the output $\mathbf{A}(x)$ is a random variable. Hence, we will measure the complexity of L by the expected running time and the 'success' probability of $\mathbf{A}$ computing χ_L:

Definition 2.15 (**BPP**-class)**.** The complexity class **BPP** is the class of all decision problems L for which there exist a **PPT A**, which satisfies

$$\forall x \in L : \mathbf{Pr}\left[\mathbf{A}(x) = 1\right] \geq \frac{2}{3}$$
$$\forall x \notin L : \mathbf{Pr}\left[\mathbf{A}(x) = 0\right] \geq \frac{2}{3},$$

The important feature of the problems in **BPP** is the fact, that the success probability can be bounded away in polynomial time. Hence, the constant $\frac{2}{3}$ can be substituted by any constant greater than $\frac{1}{2}$ without changing the definition. More generally **BPP** can be characterized by the following lemma.

Lemma 2.16. *Let L be a decision problem, than the following definitions are equivalent*

(i) $L \in \mathbf{BPP}$

(ii) $\exists p$ *positive polynomial,* $\exists$ **PPT A** *such that*

$$\forall x : \mathbf{Pr}\left[\mathbf{A}(x) = \chi_L(x)\right] \geq \frac{1}{2} + \frac{1}{p(|x|)}$$

(iii) $\forall p$ *positive polynomial,* $\exists$ **PPT A** *such that*

$$\forall x : \mathbf{Pr}\left[\mathbf{A}(x) = \chi_L(x)\right] \geq 1 - 2^{-p(|x|)}$$

Proof only for $(2 \Rightarrow 1)$. Let $\mathbf{A}$ be an algorithm and p a polynomial according to (ii). For $k \in \mathbb{N}$ we construct an algorithm $\mathbf{M}_k$, which involves algorithm $\mathbf{A}$ k-times and then rules by majority, namely

$$\mathbf{M}_k(x) := \begin{cases} 1 & \text{if } \frac{1}{k}\sum_{i=1}^{k} \mathbf{A}(x) \geq \frac{1}{2} \\ 0 & \text{else} \end{cases}$$

Since $\mathbf{A}$ runs in polynomial time and k can be bounded by a polynomial in the length of x, the constructed algorithm $\mathbf{M}_k$ will run in polynomial time. Let's observe the success probability of $\mathbf{M}_k$.
Let $x \in L$ and define $c > 0$ by the equation $\mathbf{Pr}\left[\mathbf{A}(x) = 1\right] = \frac{1}{2} + c$.

Clearly $p(|x|) \geq 1/c$ holds.
Further we define $X_1, ..., X_k := \mathbf{A}(x)$ and $S_k := \frac{1}{k}\sum_{i=1}^{k} X_i$. $X_1, ..., X_k$ are independent Bernoulli random variables with expectation $\mathbb{E}(X_i) = \frac{1}{2} + c$ and variance σ^2.
Hence, we get $\mathbb{E}(S_k) = \frac{1}{k}\sum_{i=1}^{k} \mathbb{E}(X_i) = \frac{1}{2} + c$ and $\mathrm{Var}(S_k) = \frac{1}{k}\sigma^2$.
Since $x \in L$, it holds that

$$\begin{aligned}
\mathbf{Pr}\left[\mathbf{M}(x) = 1 \,|\, x \in L\right] &= \mathbf{Pr}\left[S_k \geq \frac{1}{2}\right] = \mathbf{Pr}\left[S_k - (\frac{1}{2} + c) \geq -c\right] \\
&\geq \mathbf{Pr}\left[|S_k - \mathbb{E}(S_k)| \geq -c\right] = 1 - \mathbf{Pr}\left[|S_k - \mathbb{E}(S_k)| < c\right] \\
&\geq 1 - \frac{\mathrm{Var}(S_k)}{c} \quad \text{Chebyshov's Inequality} \\
&= 1 - \frac{\sigma^2}{kc}. \qquad (1)
\end{aligned}$$

With $k(|x|) := 3\sigma^2 p(|x|) + 1$ algorithm $\mathbf{M}_{k(|x|)}$ runs in polynomial time and

$$\mathbf{Pr}\left[\mathbf{M}_{k(|x|)}(x) = 1 \,|\, x \in L\right] \geq \frac{2}{3}$$

holds for all $x \in L$. With a similar consideration for $x \notin L$ we get an **PPT** algorithm satisfying (i).
The proof for (3) $\Rightarrow$ (1) uses the Chernoff bound instead of Chebyshov's Inequality. □

Remark 2.17. We conclude that the complexity class **BPP** is the class of problems, which can be recognized by a **PPT** algorithm with a negligible error probability.

Notation 2.18. We will call a function $\nu : \mathbb{N} \to \mathbb{N}$ *negligible*, if for every polynomial p there exists $N \in \mathbb{N}$, such that for all $n > N$

$$\nu(n) < \frac{1}{p(n)}$$

holds.

3 One-way function

3.1 Motivation

The central idea of complexity theoretic cryptography is to implement security by using the gap between efficiently computable problems and the computational infeasibility of certain tasks.

Consider the follwing setting: Alice encrypts a plaintext message $x \in \{0,1\}^n$ by computing the ciphertext y, which is the image of x under the encryption function f. The adversary Eve, who knows only the ciphertext and the encryption function, tries to retrieve the plaintext message, e.g. Eve tries to find an inverse $\tilde{x} \in \{0,1\}^n$ with $f(\tilde{x}) = y$.

Clearly, if Eve just guess $\tilde{x}$, the probability to retrieve the original plaintext will be insignificant. If we assume f to be 1-1, the resulting probability is

$$\mathbf{Pr}\left[f(U_n) = f(x)\right] = \frac{1}{2^n},$$

where U_n denotes a random variable uniformly distributed over $\{0,1\}^n$.

On the other side, if Eve tests every possible plaintext, surely Eve will find the original one. But Eve will have to apply 2^n times the function f to different plaintext candidates. Hence, this approach is very inefficient and almost impossible for large n, since Eves computational resources are bounded.

The obvious questions is, if there exists an algorithm in the 'middle', namely an algorithm finding an inverse $\tilde{x}$

- with significantly high success probability
- in polynomial running time.

If inverting f is hard enough in the sense, that such algorithm does not exist, f will be called one-way.

Hence, the one-wayness of the encryption function is a necessary precondition for security. The task of this chapter is to define a formal frame, which abstracts this concept and enables us to analyse the resulting security in a precise way.

3.2 Definition

Definition 3.1 (One-way function)**.** A function $f : \{0,1\}^* \to \{0,1\}^*$ is called one-way if the following two conditions hold

(a) *easy to compute.* $\exists$ **PT** $\mathbf{A}$ such that $\forall x \in \{0,1\}^* :$ **PT** $\mathbf{A}(x) = f(x)$

(b) *hard to invert.* For every **PPT** $\mathbf{A}'$, every polynomial p and all sufficiently large $n \in \mathbb{N}$

$$\mathbf{Pr}\left[\mathbf{A}'(f(U_n), 1^n) \in f^{-1}(f(U_n))\right] < \frac{1}{p(n)}$$

holds. (U_n denotes a random variable uniformly distributed over $\{0,1\}^n$.)

Remark 3.2 (The model of adversary). In the setting, that is induced by definition 3.1, the model of adversary is called passive adversary. It is assumed, that the adversary is only able to read the ciphertext $f(x)$, generate encryptions of any message on his own (applying f on arbitrary $x \in \{0,1\}^*$)and perform probabilistic polynomial time computation.

A successful attack is interpreted as the event, that a **PPT** $\mathbf{A}'$ outputs a preimage of the cipher $f(x)$. Clearly, if f is 1-1 the preimage will be unique and equal to the original plaintext. The adversary algorithm $\mathbf{A}'$ takes as input the cipher $f(x)$ and the security parameter 1^n, which corresponds to the binary length of x. Since, the encryption function is computable in polynomial time, the size of the ciphertext $|f(x)|$ is bounded by a polynomial in $n = |x|$.

The guarantee of one-wayness is probabilistic. The adversary is not unable to find the original plaintext, but has very low probability to do so. Asymptotically, every adversary algorithm will have only negligible better success probability than simple guessing.

3.3 Useful length convention

For further discussions it is useful to introduce the following convention regarding the length of the preimage and image of one-way functions.

In definition 3.1 the domain of f is the set of all binary words (with finite length). Clearly, we can generalize the definition:

Definition 3.3 (One-way on lengths in I)**.** Let $I \subset \mathbb{N}$ be a polynomial-time enumerable set, i.e. $\forall n \in \mathbb{N} : \ s_I(n) := \min\{i \in I : i > n\}$ is polynomial-time computable and f is defined over lengths in I, i.e. the domain of f is $\bigcup_{n\in I}\{0,1\}^n$. f is called *one-way on lengths in* I if f is computable in polynomial time and f is hard to invert over n's in I.

Given such a function f, we can construct a function $g : \{0,1\}^* \to \{0,1\}^*$ by setting $g(x) := f(x')$ with x' is the longest prefix of x with length in I.

Lemma 3.4. *Let I be a polynomial-time enumerable set and f one-way on lengths in I, then g (constructed above) is one-way according to definition 3.1.*

Proof sketch. Assume there exists an probabilistic polynomial time algorithm $\mathbf{B}'$ which inverts g with not negligible probability and conclude that there exists an **PPT** $\mathbf{A}'$ using $\mathbf{B}'$ and inverting f with not negligible probability. □

Definition 3.5 (Length preserving)**.** A function $f : \{0,1\}^* \to \{0,1\}^*$ is called length preserving if

$$\forall x \in \{0,1\}^* : \ |f(x)| = |x|$$

Given an arbitrary one-way function f we can construct a length-preserving function g, which is one-way:

Lemma 3.6. *If f is a one-way function, then there exists a length preserving one-way function g.*

3.4 Candidates for one-way functions

Since one-way functions can be computed in polynomial time, the breaking task can be performed by a non-deterministic polynomial time machine (cp. 2.14). Hence, a necessary condition for the existence of one-way functions is

$$\mathbf{P} \neq \mathbf{NP}.$$

Further, if **NP** is contained in **BPP**, then also the breaking task can be performed by a probabilistic polynomial time algorithm with not negligible probability. Hence, a more strong necessary condition for the existence of one-way functions is

$$\mathbf{NP} \not\subseteq \mathbf{BPP}.$$

The following functions are conjectured to be one-way, since no efficient inverting algorithms are known.

Example 3.7 (Factoring)**.** The best algorithms known for factoring an integer $N \in \mathbb{N}$ run in time

$$2^{\mathcal{O}(\sqrt{\log P \log\log P})},$$

where P is the second biggest prime factor in N. Hence we can conjecture that the function f_{mult}, which partitions its input into two parts and returns the integer resulting by multiplying these parts, is one-way.

Assuming the intractability of factoring and using the density of primes one can construct a one-way function based on f_{mult}.

Other popular functions like RSA or Rabin-Square are related to integer factorization.

Example 3.8 (Discrete logarithm)**.** Let p be a prime and g a generator of the multiplicative group $\mathbb{Z}_p^*$. The function $\text{Exp}_{g,p} : x \to (g^x \bmod p)$ is assumed to be one-way. The problem to find an inverse of $\text{Exp}_{g,p}$ is called the discrete logarithm problem DLP and the fastest randomized algorithms known today work in subexponential running time.

An interesting problem connected with DLP is to find an algorithm, which will generate a prime p and a generator g of $\mathbb{Z}_p^*$. A deterministic polynomial time algorithm is not known today, only randomized algorithms with expected polynomial running time are known.

The popular ElGamal cryptosystem is closely related to the DLP.

4 Collections of one-way functions

4.1 Motivation

In this section we introduce the concept of one-way collections, which is more suitable to examine in practice.

In definition 3.1 the domain of the one-way function f was a infinite set. In this section we avoid this by talking about a infinite collection of functions, each defined over some finite domain.

As a motivating discussion, let's mentally install a cryptosystem by using $\text{EXP}_{g,p}$ definied in 3.8:

(a) At first we will select the security level by defining the security parameter 1^n.

(b) According to 1^n we will compute (g, p), where p is prime with length n and g a generator of $\mathbb{Z}_p^*$.

(c) Clearly, $\text{EXP}_{g,p}$ is 1-1 if we reduce the domain to $D_p := \{1, ..., p-1\}$. Hence, we split up the plaintext message according to D_p and maintain $x \in D_p$.

(d) Finally we encode x by applying $\text{EXP}_{g,p}$.

Since difficulty to invert $\text{EXP}_{g,p}$ depends on the selection of p (e.g. if $p-1$ has only very small factors, $\text{EXP}_{g,p}$ can be inverted efficiently), the analysis of the security in this setting must consider not only the hard-to-invert property of $\text{EXP}_{g,p}$, but also the probability distribution of the algorithm computing (p, g) in (b) and the sampling algorithm in (c).

This obeservation leads to the following definition.

4.2 Definition

Definition 4.1 (Collection of one-way functions)**.** Let I be a set of indices and $D_i \subset \{0,1\}^*$ finite $\forall i \in I$. A collection of one-way functions is a set

$$F = \{f_i : D_i \to \{0,1\}^*\}_{i \in I}$$

satisfying the following two conditions

1 There exists three **PPT** $\mathbf{S_I}$, $\mathbf{S_D}$, $\mathbf{A}$, such that
   - $\mathbf{S_I}$ on input 1^n outputs an $i \in \{0,1\}^n \cap I$
   - $\mathbf{S_D}$ on input $i \in I$ outputs an $x \in D_i$
   - $\mathbf{A}$ on input $i \in I$ and $x \in D_i$ outputs $f_i(x)$

2 For every **PPT** $\mathbf{A'}$, every polynomial $p(\cdot)$ and sufficiently large n

$$\mathbf{Pr}\left[\mathbf{A'}(f_{I_n}(X_n), I_n) \in f_{I_n}^{-1}(f_{I_n}(X_n))\right] < \frac{1}{p(n)}$$

holds.

I_n, X_n are random variables describing the output distribution of $\mathbf{S_I}$, $\mathbf{S_D}$.

Lemma 4.2. *One-way functions exist if and only if collections of one-way functions exist.*

For a proof we refer to [GB01].

Remark 4.3. Under the strong discrete logarithm assumption (i.e. $\forall$**PPT A** and $\forall k$ sufficiently large: $\mathbf{Pr}\left[\mathbf{A}(g,p,y) = x \text{ such that } g^x \bmod p = y\right]$ is negligible) there exists a collection of one-way functions based on the DLP.

4.3 Collection of trapdoor functions

In this section we will concern about the proceeding of decryption. As the encryption, the decryption of ciphertext should be computable in polynomial time.

Because of the one-wayness of the encryption function, a decryption algorithm needs additional information to compute the plaintext in polynomial-time. This additional information is also called the trapdoor and allows the efficiently inverting of f.

In the concept of one-way collections, we claim that for every $f_i \in F$, F collection of one-way functions, there exist a trapdoor t_i, such that an algorithm on input $(f_i(x), t_i)$ finds x in polynomial time.

Hence, the formal definition is

Definition 4.4 (Collection of trapdoor functions)**.** Let I be a set of indices and $D_i \subset \{0,1\}^*$ finite $\forall i \in I$. A collection of trapdoor functions is a set

$$F = \{f_i : D_i \to \{0,1\}^*\}_{i \in I}$$

satisfying the following two conditions

1 There exists four **PPT** $\mathbf{S_I}$, $\mathbf{S_D}$, $\mathbf{A}$, such that
   - $\mathbf{S_I}$ on input 1^n outputs an $i \in \{0,1\}^n \cap I$ and the trapdoor t_i with $|t_i| < p(n)$ for a polynomial $p(\cdot)$.
   - $\mathbf{S_D}$ on input $i \in I$ outputs an $x \in D_i$
   - $\mathbf{A_1}$ on input $i \in I$ and $x \in D_i$ outputs $f_i(x)$
   - $\mathbf{A_2}$ such that $\mathbf{A_2}(i, t_i, f_i(x)) = x$ for all $i \in I, x \in D_i$.

2 For every **PPT** $\mathbf{A'}$, every polynomial $p(\cdot)$ and sufficiently large n

$$\mathbf{Pr}\left[\mathbf{A'}(f_{I_n}(X_n), I_n) \in f_{I_n}^{-1}(f_{I_n}(X_n))\right] < \tfrac{1}{p(n)}$$

holds.

I_n, X_n are random variables describing the output distribution of $\mathbf{S_I}$, $\mathbf{S_D}$.

Remark 4.5. Possible examples for trapdoor collections like RSA, SQUARE-RABIN can be found in [GB01].

Remark 4.6. The implementation of a public key cryptosystem can be inspired by the following idea. Let $F := \{f_i\}_{i \in I}$ be a collection of trapdoor functions. After randomly choosen an $i \in I$ according to the security parameter, Alice computes the trapdoor information t_i and keeps it secret as the private key. Then Alice distributes the function f_i as a public key. Bob can send a message x to Alice by using the encryption funtion f_i. For every adversary Eve, not knowing the trapdoor t_i, it is infeasible to compute the original plaintext.

Detailed information about the relation between one-way functions (collections) and public key cryptosystems, public key distribution systems can be found in [DH76].

5 Hard-core predicate

5.1 Motivation

In 3.2 we observed that in the context of one-wayness an adversary attack is succesful if $\mathbf{A}'$ finds an $\tilde{x}$ with $f(\tilde{x}) = f(x)$. The situation that an adversary algorithm could find partial information of x, e.g. the least significant bit, is not considered.

As an example let's observe the one-way function candidate EXP defined in 3.8. We show that it is easy to compute the least significant bit of x given only $\mathrm{EXP}(x)$:

Lemma 5.1. *Let p be a prime and g generator of $\mathbb{Z}_n^*$. Let $x \in \mathbb{Z}_n^*$ and define $y := \mathrm{EXP}_{g,p}(x)$. Then the least significant bit of x can be (efficiently) computed by*

$$\textit{least significant Bit of } x := \begin{cases} 1 & \textit{if } y^{\frac{p-1}{2}} \equiv 1 \bmod p \\ 0 & \textit{else} \end{cases}$$

Proof. For p prime we define the map

$$f : \mathbb{Z}_p^* \to \mathbb{Z}_p^*,\ f(x) := x^2 \bmod p$$

and the set of the quadratic residues modulo p

$$QR(p) := f(\mathbb{Z}_p^*) = \{y \in \mathbb{Z}_p^* : \exists x : y \equiv x^2 \bmod p\}$$

Let $y \in QR(p)$. Let $w, x \in \mathbb{Z}_p^*$ with $y \equiv x^2 \bmod p \equiv x^2 \bmod p$. Then it holds that

$$\begin{aligned} & x^2 - w^2 \bmod p \equiv 0 \bmod p \\ \Rightarrow & p|(x-w)(x+w) \\ \Rightarrow & \underbrace{p|(x-w)}_{\Rightarrow w \equiv x \bmod p} \text{ or } \underbrace{p|(x+w)}_{\Rightarrow w \equiv -x \bmod p} \\ \Rightarrow & |f^{-1}(y)| \leq 2 \end{aligned}$$

Clearly, $y = f(x) = f(-x)$ and since p is not even, $x \bmod p \not\equiv -x \bmod p$ holds. Hence, $|f^{-1}(y)| = 2\ \forall y \in QR(p)$ and so $QR(p)$ must have the cardinality $\frac{p-1}{2}$.

Let g be a generator of $\mathbb{Z}_p^*$, then g^x is a quadratic residue if and only if x is even. On the other side

$$a \in QR(p) \Leftrightarrow a^{\frac{p-1}{2}} \equiv 1 \bmod p,$$

since with Euler's criterion it holds, that

$$a \in QR(p) \Rightarrow \exists \alpha : a = g^{2\alpha} \Rightarrow a^{\frac{p-1}{2}} = (g^\alpha)^{p-1} \equiv 1 \bmod p \qquad (2)$$

and if $g^\beta \equiv 1 \bmod p$, we conclude that β must be even.

Since the least significant bit of x is equal to 0 if and only if x is even, we can compute it by testing (2). □

Example 5.1 shows that a one-way function does not necessarily hide everything about x. But obviously there is at least one bit of x which is hard to retrieve from $f(x)$.

Indeed in the case of the DLP, one can show that 'guessing' the most significant bit of x is hard in the sense, that if someone can compute the most significant bit of x with probability not negligible better than $1/2$, then there exists a **PPT** algorithm that solves the DLP with not negligible success probability. (Detailed information can be found in [BM84].)

In general we will call a predicate b a hard-core of f if guessing $b(x)$ from $f(x)$ is as hard as inverting f.

5.2 Definition

Definition 5.2. A hard-core predicate of a function $f : \{0,1\}^* \to \{0,1\}^*$ is a boolean predicate $b : \{0,1\}^* \to \{0,1\}$, such that

(a) $\exists$**PPT** $\mathbf{A} : \forall x : \mathbf{A}(x) = b(x)$

(b) $\forall$**PPT** $\mathbf{G}, \forall p(\cdot)$ polynomial and all sufficiently large n

$$\mathbf{Pr}\left[G(f(U_n)) = b(U_n)\right] < \frac{1}{2} + \frac{1}{p(n)}$$

holds.

5.3 A generic hard-core predicate

In this section we will construct a hard-core predicate for one-way functions of special form and we will show how an arbitrary one-way function can be transformed into the required form without loss in either 'security' nor 'efficiency'.

Theorem 5.3 (Generic hard-core predicate)**.** *Let f be an arbitrary length-preserving one-way function. Define $g : \bigcup_{n\in\mathbb{N}} \{0,1\}^{2n} \to \{0,1\}^*$ by*

$$g(x,r) := (f(x), r) \text{ with } |r| = |x|.$$

Let $b(r,x)$ denote the inner product modulo 2, namely

$$b(x,r) = \sum_{i=1}^{n} (x_i r_i \bmod 2),$$

where $n := |x|$ and r_i, x_i denotes the ith Bit of x respectively r.
Then b is a hard-core predicate of the function g.

Remark 5.4. In other words, b computes the exclusive-or of a subset of the bits of x. The subset is induced by the r. Hence, the theorem states that if f is a one-way function it is infeasible to guess this exclusive-or when given $f(x)$ and the subset itself.

Proof. The proof works by contradiction and uses a 'reducibility argument': We assume that $b(x, r)$ can be predicted efficiently by an algorithm $\mathbf{G}$ with probability not negligible better than $\frac{1}{2}$ and we will conclude the existence of an efficient algorithm inverting f. This contradicts the hypothesis that f is a one-way function.

Let's assume b is not a hard-core predicate. We conclude:

$$\begin{aligned} &\exists \mathbf{PPT}\ \ \mathbf{G} \\ &\exists p' \text{ polynomial} \\ &\exists I' \subset \mathbb{N} \text{ infinite} \end{aligned}$$

such that

$$\mathbf{Pr}\left[G(g(U_n)) = b(U_n)\right] > \frac{1}{2} + \frac{1}{p'(n)} \quad \forall n \in I'$$

According to the structure of b we can transfer this statement to the equivalent one:

$$\begin{aligned} &\exists \mathbf{PPT}\ \ \mathbf{G} \\ &\exists p \text{ polynomial} \\ &\exists I \subset \mathbb{N} \text{ infinite} \end{aligned} \tag{3}$$

such that

$$\mathbf{Pr}\left[G(g(X_n, R_n)) = b(X_n, R_n)\right] > \frac{1}{2} + \frac{1}{p(n)} \quad \forall n \in I$$

We define the advantage $\epsilon(n)$ of $\mathbf{G}$ in predicting $b(x, r)$ from $f(x)$ and r by

$$\epsilon(n) := \mathbf{Pr}\left[G(f(X - n), R_n) = b(X_n, R_n)\right] - \frac{1}{2}.$$

Clearly, $\epsilon(n) > \frac{1}{p(n)} \quad \forall n \in I$. In the sequel we restrict our attention only to $n \in I$.

As a first observation, we conclude that there is subset S_n of words of length n, which has at least the cardinality $\frac{\epsilon(n)}{2}2^n$ and for which $\mathbf{G}$ predicts b with an advantage of at least $\frac{\epsilon(n)}{2}$. The following claim can be proved by applying the Markov Inequality on $s(X_n)$.

Claim. There exists a set $S_n \subseteq \{0,1\}^n$, such that

- $|S_n| \geq \frac{\epsilon(n)}{2}2^n$
- $s(x) := \mathbf{Pr}\left[\mathbf{G}(f(x), R_n) = b(x, R_n)\right] \geq \frac{1}{2} + \frac{\epsilon(n)}{2} \quad \forall x \in S_n.$

In the sequel we restrict our attention to x's in S_n. For those x's we will construct an efficient algorithm that on every input y with $f(x) = y,\ x \in S_n$ will find x with high probability. Since, $\mathbf{Pr}\left[X_n \in S_n\right] \geq \frac{\epsilon(n)}{2}$ contradiction to the one-wayness of f will follow.

Idea 1 As a mental experiment we suppose $\mathbf{G}$ to be such good, that

$$s(x) > \frac{3}{4} + \frac{1}{2p(n)} \quad \forall x \in S_n. \tag{4}$$

Under such assumption, we can easily construct an algorithm $\mathbf{A}$ finding an preimage x.

First, we introduce the notation

$$x \oplus r := w \text{ with } w_i := (x_i + r_i) \bmod 2.$$

If $e_i \subset \{0,1\}^n$ denotes the word, equal to zero except the ith bit$= 1$, then clearly

$$b(x, e_i) = x_i \quad \forall x \tag{5}$$

and

$$\begin{aligned} b(x,r) \oplus b(x, r \oplus e_i) =& b(x, r \oplus r \oplus e_i) = b(x, e_i) \\ =& x_i \quad \forall x, r. \end{aligned} \tag{6}$$

holds.

The probability of the event $\mathcal{H}$, that

$$\mathcal{H}:\ \mathbf{G}(f(x), r) = b(x, r) \text{ and } \mathbf{G}(f(x), r \oplus e_i) = b(x, r \oplus e_i)$$

holds, is

$$\mathbf{Pr}\left[\mathcal{H} | x \in S_n\right] > 1 - \frac{1}{poly(|x|)}.$$

In this case we can compute the ith bit of x by using (6):

$$\mathbf{G}(f(x), r) \oplus \mathbf{G}(f(x), r \oplus e_i) = b(x, r) \oplus b(x, r \oplus e_i) = x_i.$$

Since, $\mathbf{Pr}\left[\mathcal{H}\right]$ is not negligible better than $\frac{1}{2}$, we can retrieve x_i by repeating the above procedure polynomially often and ruling by majority.

The problem of idea 1 is that the assumption made in (4) cannot be weakened. An important point is, that $\mathbf{Pr}\left[\mathcal{H}\right]$ is not negligible better than $\frac{1}{2}$. We obtain this probability by using $\mathbf{G}$ twice to get a 'guess' of $b(x, r)$ and $b(x, r \oplus e_i)$; consequently $\mathbf{Pr}\left[\mathcal{H}\right]$ is the squared success probability of $\mathbf{G}$. In a realistic setting the advantage $\epsilon(n)$ of $\mathbf{G}$ will be significantly smaller than $\frac{1}{4}$, so that $\mathbf{Pr}\left[\mathcal{H}\right] \leq \frac{1}{2}$. That means we cannot improve the result by repeating and ruling by majority.

Idea 2 The second idea is to construct a sequence of $\{r^J\}_{J\in M}$ and only using $\mathbf{G}$ once for each r^J to get a prediction of $b(x, r^J \oplus e_i)$. We will construct $\{r^J\}_{J\in M}$ in such a way, that on one side the sequence of the r^J's is 'sufficiently random' and on the other side 'structured', so that we are able to make sufficiently good predictions of the predicates $b(x, r^J)$.

The algorithm The following algorithm, denoted $\mathbf{A}$, inverts f with non negligible probability. Let y be the input of $\mathbf{A}$, then

1. $\mathbf{A}$ sets $n := |y|$, $l := \lceil \log_2(2n\, p(n)^2 + 1 \rceil$, where $p(\cdot)$ is defined in (3).
2. $\mathbf{A}$ uniformly and independently selects $s_1, ..., s_l \in \{0,1\}^n$
3. $\mathbf{A}$ uniformly and independently selects $\sigma_1, ..., \sigma_l \in \{0,1\}$
4. For every non-empty set $J \subset \{1, 2, ..., l\}$ $\mathbf{A}$ computes
   $r^J := \oplus_{j\in J} s_j$ and
   $\rho^J := \oplus_{j\in J} \sigma_j$
5. For every $i \in \{1, ..., n\}$ and every non-empty set $J \subset \{1, ..., l\}$, $\mathbf{A}$ computes
   $z_i^J := \rho^J \oplus \mathbf{G}(y, r^J \oplus e_i)$
   and outputs the majority of z_i^J as the ith bit of x.

Analysis of the success probability of A For the analysis of the success probability, we define the event $\mathcal{F}$ by

$$\mathcal{F} : \forall k = 1, ..., l : \sigma_k = b(x, s_k)$$

and the event $\mathcal{E}$ by

$$\mathcal{E} : \text{for the majority of } J \subset \{0, ..., l\} : \mathbf{G}(f(x), r^J \oplus e_i) = b(x, r^J \oplus e_i).$$

Clearly, if $\mathcal{F}$ holds, then

$$\forall J \subset \{1, ..., l\} \text{ non-empty} : \rho^J = b(x, r^J).$$

Hence, $\mathcal{F}$ can be interpreted as the event that 'our guess' ρ^J is always correct.

If $\mathcal{F}$ and $\mathcal{E}$ holds, than for a majority of the non-empty subsets $J \subset \{1, ..., l\}$ it holds that

$$\rho^J \oplus \mathbf{G}(f(x), r^J \oplus e_i) = b(x, r^J) \oplus b(x, r^J \oplus e_i) = x_i.$$

In this case $\mathbf{A}$ outputs a correct ith bit.

Independently from $\mathcal{F}$ we observe the probability of $\mathcal{E}$:

Claim. For every $x \in S_n$ and every $1 \leq i \leq n$,

$$\mathbf{Pr}\left[\left|\{J : b(x, r^J) \oplus \mathbf{G}(f(x), r^J \oplus e_i) = x_i\}\right| > \frac{1}{2}(2^l - 1)\right] > 1 - \frac{1}{2n}$$

Proof sketch. For every J define a 0-1 random variable

$$X_J := \begin{cases} 1 & \text{if } b(x, r^J) \oplus \mathbf{G}(f(x), r^J \oplus e_i) = x_i \\ 0 & \text{else} \end{cases}.$$

Since the r^J are uniformly distributed over $\{0,1\}^n$ it follows that

$$\mathbf{Pr}\left[X_J = 1\right] = s(x) \geq \frac{1}{2} + \frac{1}{2p(n)}.$$

Further the X_J are pairwise independent. Hence, with Chebyschev's Inequality we get

$$\mathbf{Pr}\left[\sum_J X_J \leq \frac{1}{2}(2^l - 1)\right] < \frac{1}{2n}.$$

The claim follows. □

Now, we are able to find a lower bound for the success probability of algorithm $\mathbf{A}$. If the events $\mathcal{F}$ and $\mathcal{E}$ hold and $x \in S_n$, then $\mathbf{A}$ surely is successful. Hence, we conclude

$$\begin{aligned} \mathbf{Pr}\left[\mathbf{A}(y) = x \text{ such that } f(x) = y\right] &\geq \mathbf{Pr}\left[\mathcal{E} \wedge \mathcal{F} \wedge X_n \in S_n\right] \\ &= \mathbf{Pr}\left[\mathcal{E}\right] \mathbf{Pr}\left[\mathcal{F}\right] \mathbf{Pr}\left[X_n \in S_n\right] \\ &\geq \frac{1}{2} \cdot 2^{-l} \cdot \frac{|S_n|}{2^n} \\ &= \frac{1}{8np(n)^3 + p(n)} \end{aligned}$$

The success probability is not negligible! Since, $\mathbf{A}$ invokes $\mathbf{G}$ for $2n \cdot p(n)^2$ times, $\mathbf{A}$ is a probabilistic polynomial time algorithm, that inverts f with not negligible success probability. This is a contradiction to the hypothesis that f is one-way.

We conclude, that the 'guessing' algorithm $\mathbf{G}$ does not exist. Hence, b is a hard-core predicate of g. □

References

[GB01] Shafi Goldwasser and Mihir Bellare, *Lecture notes on cryptography*, 2001

[Gol01] Oded Goldreich, *Foundations of cryptography: Basic applications*, Cambridge University Press, 2004.

[DH76] W. Diffie and M.E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory (1976)

[BM84] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo-random bits.* SIAM J. Computing, 13(4):850-863, November 1984

[AKS02] M. Agrawal, N. Kayal, N. Saxena, *PRIMES is in P.* IIT Kanpur, www.cse.iitk.ac.in/news/primality.html