Appendix B

Complexity Theory for Cryptology

For (at least) three reasons "ordinary" complexity theory (using TURING machines) is insufficient for cryptologic needs:

1. Complexity theory primarily addresses the question whether the *worst case* of a problem is hard (i. e. a solution is not efficiently computable). However to preclude an efficient cryptanalysis we want the *normal case* to be hard. We saw that the existence of strong basic cryptographic functions implies $\mathbf{P} \neq \mathbf{NP}$, but conversely this inequality (if true) would *not* suffice to prove the existence of strong cryptography.

   From other parts of mathematics we are warned that a worst case scenario may not suffice to make a problem hard. For instance the NEWTON algorithm for determining roots of polynomials and the simplex method for linear optimization are hard in the worst case, but very efficient in the normal case.

2. The cryptanalyst is free to use *probabilistic* algorithms ("Monte Carlo" algorithms) that are very efficient but don't give a correct result in all cases. We saw several examples for number theoretic problems.

   The exact mathematical treatment uses concepts from probability theory: Parts of the input are taken from a probability space Ω. The results are statements on the distribution of the output.

3. Moreover the cryptanalyst is free to adapt her methods to the concrete problem at hand. She doesn't necessarily need a universal algorithm that is efficient for *all* instances of her problem. For example she could choose a different algorithm depending on the key length. Thus we have to consider *non-uniform* models of computation.

As a consequence the ordinary theory of TURING machines is insufficient for formalizing complexity theory as it is needed in cryptology. We could

remedy this shortage by considering families of TURING machines—say a different one for each input length—, and also admitting probabilistic input.

However an alternative model of computation, using Boolean circuits, has a simpler, more intuitive description and a more direct and elegant application: families of probabilistic circuits (FPC for short). Realizing common algorithms by circuits is distinctly simpler and more intuitive than programming a TURING machine.

B.1 Probabilistic Boolean Circuits

A Boolean circuit describes an algorithm in the form of a flow chart that connects the single bit operations, see Appendix C.12 of Part II. It has two supplemental generalizations:

a probabilistic circuit formalizes probabilistic algorithms,

a family of circuits allows to express the complexity of an algorithm for increasing input sizes.

First we formalize the concept of a probabilistic algorithm for computing a map

$$f\colon A \longrightarrow \mathbb{F}_2^s$$

on a set A. To this end we consider maps (to be represented by circuits)

$$C\colon A \times \Omega \longrightarrow \mathbb{F}_2^s$$

where Ω is a probability space. We look at the probabilities that C "computes" $f(x)$ or f:

$$P(\{\omega \mid C(x,\omega) = f(x)\}) \quad (\text{"locally" at } x) \text{ and}$$

$$P(\{(x,\omega) \mid C(x,\omega) = f(x)\}) \quad (\text{"globally"})$$

that we want to be "significantly" $> \frac{1}{2^s}$, the probability of hitting a value in $\mathbb{F}_2^s$ by pure chance. In the local case we average over Ω for fixed x, in the global case we average also over $x \in A$. In general we assume that the probability spaces Ω and $A \times \Omega$ are finite and (in most cases) uniformly distributed.

In order to describe probabilistic algorithms we need circuits with *three* different types of input nodes:

- r **deterministic input nodes** that are seeded by an input tuple $x \in \mathbb{F}_2^r$, or x from a subset $A \subseteq \mathbb{F}_2^r$,
- some **constant input nodes**, each a priori set to 0 or 1,
- k **probabilistic input nodes** that are seeded by an element ("event") of the LAPLACEan probability space $\Omega = \mathbb{F}_2^k$ (corresponding to k "coin tosses"), or by an element of a subset $\Omega \subseteq \mathbb{F}_2^k$.—Sometimes also other probability distributions on Ω, different from the uniform distribution, might be taken into account.

The theory aims at statements on the probabilities of the output values $y \in \mathbb{F}_2^s$.

Examples

1. Searching a quadratic non-residue for an n bit prime module p. Here we choose a random $b \in [1 \dots p-1]$ and compute $(\frac{b}{p})$ (the LEGENDRE symbol that is 1 for quadratic residues, -1 for quadratic non-residues). The success probability is $\frac{1}{2}$, the cost $\mathrm{O}(n^2)$ (see Appendix A.8).

   More generally we ask whether an h-tuple

   $$(b_1, \dots, b_h) \in \Omega = [1 \dots p-1]^h$$

   of independently choosen elements contains a quadratic non-residue. There is a probabilistic circuit (for the given p) without deterministic input nodes (but with some constant input nodes to input p),

   $$C : \mathbb{F}_2^{hn} \longrightarrow \mathbb{F}_2^n,$$

   $$C(\omega) = \begin{cases} b_i, & \text{the first } b_i \text{ that is a quadratic non-residue,} \\ 0 & \text{if none of the } b_i \text{ is a quadratic non-residue,} \end{cases}$$

   of size $\mathrm{O}(hn^2)$ that outputs a quadratic non-residue with probability $1 - \frac{1}{2^h}$. Note the deviation of this example from the definition above: Here C doesn't compute an explicitly given function f but provides output with a certain property.

2. The strong pseudoprime test: Here the input is taken from the set $A \subseteq [3 \dots 2^n - 1]$ of odd integers. We want to compute the primality indicator function

   $$f : A \longrightarrow \mathbb{F}_2, \quad f(m) = \begin{cases} 1 & \text{if } m \text{ is composite,} \\ 0 & \text{if } m \text{ is prime.} \end{cases}$$

   The probabilistic input consists of a base $a \in \Omega = [2 \dots 2^n - 1]$. The strong pseudoprime test is represented by a circuit

   $$C : \mathbb{F}_2^n \times \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$$

   of size $\mathrm{O}(n^3)$, and yields the result 1 if m fails (then m is proven to be composite), 0 if m passes (then m is possibly prime). Thus C outputs the correct result only with a certain probability.

Now we formalize the property of a (probabilistic) circuit C of computing the correct value of $f(x) \in \mathbb{F}_2^s$ with a probability that "significantly" differs from a random guess: Given $\varepsilon \geq 0$, a circuit

$$C : \mathbb{F}_2^r \times \Omega \longrightarrow \mathbb{F}_2^s$$

(with r deterministic input nodes) has an **ε-advantage** for the computation of $f(x)$ or f if

$$P(\{\omega \in \Omega \mid C(x,\omega) = f(x)\}) \geq \frac{1}{2^s} + \varepsilon \quad (\text{“local case”}) \text{ or}$$

$$P(\{(x,\omega) \in A \times \Omega \mid C(x,\omega) = f(x)\}) \geq \frac{1}{2^s} + \varepsilon \quad (\text{“global case”}).$$

Thus in the global case the probability with respect to ω of getting a correct result is additionally averaged over $x \in A$. The advantage 0, or the probability $\frac{1}{2^s}$, corresponds to randomly guessing the result.

C has an **error probability** δ for computing $f(x)$ or f if

$$P(\{\omega \in \Omega \mid C(x,\omega) = f(x)\}) \geq 1 - \delta \text{ or}$$

$$P(\{(x,\omega) \in A \times \Omega \mid C(x,\omega) = f(x)\}) \geq 1 - \delta.$$

Examples

1. For searching a quadratic non-residue $\bmod p$ we have

   $$P(\{\omega \in \Omega \mid C(\omega) \text{ is a quadratic non-residue}\}) = 1 - \frac{1}{2^h}.$$

   Thus the circuit has an $(\frac{1}{2} - \frac{1}{2^h})$-advantage and an error probability of $\frac{1}{2^h}$.

2. For the strong pseudoprime test we have for fixed m

   $$P(\{\omega \in \Omega \mid C(m,\omega) = f(m)\}) \begin{cases} \geq \frac{3}{4} & \text{if } m \text{ is composite,} \\ = 1 & \text{if } m \text{ is prime.} \end{cases}$$

   Averaging over m we get

   $$P(\{(m,\omega) \in A \times \Omega \mid C(m,\omega) = f(m\}) \geq \frac{3}{4},$$

   hence an $\frac{1}{4}$-advantage and an error probability of $\frac{1}{4}$. (Since the number of composite integers is much larger than the number of primes, the value $\frac{1}{4}$ is not significantly changed by averaging over m.)

B.2 Polynomial Size Families of Circuits

A circuit has a fixed number of input nodes. Therefore it can process inputs of a fixed length only, in contrast with a TURING machine. However to assess the efficiency of an algorithm in general we have to estimate the increase of cost for increasing input sizes.

To this end we consider families $(C_n)_{n\in\mathbb{N}}$ of circuits with an increasing number of deterministic input nodes. Then the cost of a computation may be expressed as a function of the length of the input.

More exactly we define: A **family of probabilistic circuits (FPC)** is a family $C = (C_n)_{n\in\mathbb{N}}$,

$$C_n \colon \mathbb{F}_2^{r(n)} \times \mathbb{F}_2^{k(n)} \longrightarrow \mathbb{F}_2^{s(n)}, \tag{1}$$

where the circuit C_n has $r(n)$ deterministic input nodes, and $k(n)$ probabilistic ones. Of course, if all $k(n) = 0$, we speak of a family of deterministic circuits.

A **polynomial size family of probabilistic circuits (PPC)** is an FPC $C = (C_n)_{n\in\mathbb{N}}$, such that $\#C_n \leq \alpha(n)$ for all $n \in \mathbb{N}$ with a polynomial $\alpha \in \mathbb{N}[X]$ (non-negative integer coefficients). In particular the number of input nodes of all kinds, as well as the number $s(n)$ of output nodes, is polynomially bounded. (We don't require that the functions r, k, s themselves are polynomials.)

Even in the deterministic case this model of computation might be able to compute more functions than the common model of TURING machines (and it is in fact), since it allows to choose a different algorithm for each input length. For this reason we also speak of a "non-uniform computational model". On first sight this feature seems not so pleasant. Nevertheless it is particularly realistic for cryptanalysis: Depending on the input size n the cryptanalyst may choose a suitable algorithm.

If a TURING machine computation in polynomial time is possible, then for the same problem there is a PPC also. The reverse statement is *not* true, although we only know "artificial" counterexamples.

Should any NP-complete problem be computable by a PPC, then so would be all the other ones. Virtually nobody believes in this possibility.

Non-uniform complexity may be modelled by TURING machines also, simply admitting a different TURING machine for each input length. Moreover we could also define probabilistic TURING machines. After all preferring the SHANNON model of circuits over TURING machines is a matter of taste.

A computational problem is called **hard** if there is no PPC that solves it with a distinguished advantage; we'll make this definition more precise in the next sections. The "hard number theoretic problems" from Chapter 5, such as prime decomposition, are conjectured to be hard in this sense.

We know already that the basic operations on integers are computable by (even deterministic) PPC's. And therefore so are all algorithms on integers

that are efficiently computable in the naive sense, using only "polynomially many" elementary arithmetic operations.

B.3 Efficient Algorithms

To generalize the results from Section B.1 we first define the concepts of advantage and error probability for PPCs.

Let $L \subseteq \mathbb{F}_2^*$ be a language over the binary alphabet $\mathbb{F}_2$, and set $L_n := L \cap \mathbb{F}_2^n$. Let f be a map

$$f\colon L \longrightarrow \mathbb{F}_2^* \quad \text{with } f(L_{r(n)}) \subseteq \mathbb{F}_2^{s(n)} \tag{2}$$

where $r(n)$ is the monotonically increasing sequence of indices i with $L_i \neq \emptyset$. We want to compute this map by a PPC as in (1).

Examples

1. The function $f(x, y, z) := xy \bmod z$ for n-bit integers x, y, z is computable by a (deterministic) circuit

   $$C_n\colon \mathbb{F}_2^{3n} \longrightarrow \mathbb{F}_2^n$$

   of size $\#C_n = \mathrm{O}(n^3)$ (with error probability 0). Here $r(n) = 3n$ and $s(n) = n$.

2. Let L be the set of (binary encoded) odd integers ≥ 3, and $f\colon L \longrightarrow \mathbb{F}_2$ be the primality indicator as in Section B.1. There we saw a PPC for the strong pseudoprime test of size $\mathrm{O}(n^3)$ with advantage $\frac{1}{4}$ and error probability $\frac{1}{4}$ (constant with respect to n). Using t bases we get a size of $\mathrm{O}(tn^3)$, and an error probability of $\frac{1}{4^t}$.

Definition 1 A function $\varphi\colon \mathbb{N} \longrightarrow \mathbb{R}_+$ is called **(asymptotically) negligible** if for each nonconstant polynomial $\eta \in \mathbb{N}[X]$

$$\varphi(n) \leq \frac{1}{\eta(n)} \quad \text{for almost all } n \in \mathbb{N}.$$

In other words, $\varphi(n)$ tends to 0 faster than the inverse of any polynomial.

Example An obvious example is $\varphi(n) = 2^{-n}$.

Definition 2 Sei $f\colon L \longrightarrow \mathbb{F}_2^*$ be as in (2). Let C be a PPC that computes f on $L_{r(n)}$ with an error probability of ε_n. Assume ε_n is a negligible function of n. Then C is called an **efficient probabilistic algorithm** for f.

f is called **(probabilistically) efficiently computable** if there is an efficient algorithm for f.

This definition substantiates the idea of an algorithm that is "efficient for almost all input tuples" (or input strings if the input is taken from a language L).

For RABIN's primality test, that is the repeated execution of the strong pseudoprime test, we satisfy this requirement by letting the number t of bases grow with n. In order to get a polynomial family we upgrade t to a polynomial $\tau \in \mathbb{N}[X]$. Then C_n has n deterministic input nodes, and $n\tau(n)$ probabilistic ones. The size is $\mathrm{O}(n^3\tau(n))$, and the error probability, $\frac{1}{4^{\tau(n)}}$. Thus we have shown:

Proposition 28 RABIN*'s primality test is an efficient probabilistic algorithm for deciding primality.*

B.4 Hard Problems

Exactly defining what a hard problem is is somewhat more tricky. We want to characterize a problem that has *no efficient solution for almost all* input tuples (or strings). Simply negating the property "efficient" is clearly insufficient. Somewhat better is the requirement that the advantage of an algorithm approaches 0 with increasing n. But also this is not yet a suitable definition since the advantage describes a lower bound only.

A better requirement is the non-existence of an advantage that approaches 0 too slowly. "Too slowly" is

$$\frac{1}{\eta(n)} \quad \text{with an arbitrary polynomial } \eta \in \mathbb{N}[X].$$

"Slow enough" is for instance the inverse exponential finction $1/2^n$.

Moreover there should be "almost no" exceptions, or the set of exceptions should be "sparse". Now we try to translate these ideas into an exact definition.

For $x \in L_{r(n)}$ we consider the probability

$$p_x := P(\{\omega \in \Omega_{k(n)} \mid C_n(x, \omega) = f(x)\}),$$

and the set of input strings x for which C_n has an ε-advantage:

$$L_{r(n)}(\varepsilon) := \{x \in L_{r(n)} \mid p_x \geq \frac{1}{2^{s(n)}} + \varepsilon\}.$$

For a polynomial $\eta \in \mathbb{N}[X]$ the set $L_{r(n)}(\frac{1}{\eta(n)})$ consists of the input strings x for which C computes $f(x)$ with advantage $\frac{1}{\eta(n)}$. Thus the exceptional set for η is

$$L^{[f,C,\eta]} := \bigcup_{n \in \mathbb{N}} L_{r(n)}(\frac{1}{\eta(n)}).$$

We denote it as "**advantageous set for f, C, η**". Its components should become more and more marginal with increasing n. The definition is:

Definition 3 A subset $A \subseteq L$ is called **sparse** if

$$\frac{\#A_n}{\#L_n}$$

is negligible.

Remarks and Examples

1. If $\#A_n = c$ is constant, and $L_n = \mathbb{F}_2^n$, then A is sparse in L for

$$\frac{\#A_n}{\#L_n} = \frac{c}{2^n}.$$

2. If $\#A_n$ grows at most polynomially, but $\#L_n$ grows faster than any polynomial, then A is sparse in L.

3. If $\#A_n = c \cdot \#L_n$ is a fixed proportion, then A is not sparse in L.

4. If $L = \mathbb{N}$, and A is the set of primes (in binary coding), then by the prime number theorem

$$\#A_n \approx \frac{2^{n-1}}{n \cdot \ln(2)} = \frac{\#L_n}{n \cdot \ln(2)}.$$

   Hence the set of primes is not sparse in $\mathbb{N}$.

5. No known efficient algorithm is able to factorize a non-sparse subset of the set M of all products of primes whose lengths differ by at most one bit.

Definition 4 Let f be as in (2). Then f is called **hard** if for each PPC as in (1) and for each polynomial $\eta \in \mathbb{N}[X]$ the advantageous set $L^{[f,C,\eta]}$ is a sparse subset of L.

Examples

1. The conjecture that prime decomposition of integers is hard makes sense by remark 5.

2. **Quadratic residuosity conjecture:** Let B be the set of Blum integers (products of two primes $\equiv 3 \pmod 4$),

$$L = \{(m, a) \mid m \in B, a \in \mathbb{M}_n^+\},$$

   (for $\mathbb{M}_n^+$ see Appendix A.5) and let

$$f \colon L \longrightarrow \mathbb{F}_2$$

   be the indicator function

$$f(m, a) = \begin{cases} 1 & \text{if } a \text{ is a quadratic residue mod } m, \\ 0 & \text{else.} \end{cases}$$

   Then f is hard. (A forteriori when we more generally admit $a \in \mathbb{M}_n$.)

B.5 Basic Cryptographic Functions

Now the theoretic basis suffices for an exact definition of one-way functions and strong symmetric ciphers. Note that the "functions" or "maps" in these definitions are infinite families with growing input size. There is no mathematically sound definition of one-way or hash functions, or of strong symmetric ciphers, for a fixed input size, as we assumed in treating these concepts in a naive way in Section 4.1 and Chapters 5 and 6.

Definition 5 Let $f: L \longrightarrow \mathbb{F}_2^*$ be as in (2). A right inverse of f is a map $g: f(L) \longrightarrow L \subseteq \mathbb{F}_2^*$ with $f(g(y)) = y$ for all $y \in f(L)$. In other words g finds pre-images of f. We call f a **one-way function** if each right inverse of f is hard.

Adapting this definition the conjecture that the discrete exponential function in finite prime fields is hard makes sense.

Now for the definition of a strong cipher. An "ordinary" block cipher is a map

$$F: \mathbb{F}_2^r \times \mathbb{F}_2^q \longrightarrow \mathbb{F}_2^r.$$

The corresponding decryption function is a map

$$G: \mathbb{F}_2^r \times \mathbb{F}_2^q \longrightarrow \mathbb{F}_2^r$$

with $G(F(x,k),k) = x$ for all $x \in F_2^r$ and $k \in F_2^q$.

An attack with known plaintext finds a key $k \in F_2^q$ with $F(x,k) = y$, given $x, y \in \mathbb{F}_2^r$. We formalize this by a map

$$H: \mathbb{F}_2^r \times \mathbb{F}_2^r \longrightarrow \mathbb{F}_2^q$$

with $F(x, H(x,y)) = y$ for all $x, y \in \mathbb{F}_2^r$ with $y \in F(x, \mathbb{F}_2^q)$ ("possible pairs" (x,y)).

Exercise Give an exact definition of a possible pair.

A more general attack uses several, say s, plaintext blocks. So it defines a map

$$H: \mathbb{F}_2^{rs} \times \mathbb{F}_2^{rs} \longrightarrow \mathbb{F}_2^q$$

with $F(x_i, H(x_i, y_i)) = y_i$ for $i = 1, \ldots s$ for all possible $x, y \in \mathbb{F}_2^{rs}$.

Now we give a definition in terms of complexity theory.

Definition 6 A **symmetric cipher** is a family $F = (F_n)_{n \in \mathbb{N}}$ of block ciphers

$$F_n: \mathbb{F}_2^{r(n)} \times \mathbb{F}_2^{q(n)} \longrightarrow \mathbb{F}_2^{r(n)}$$

with strictly monotonically increasing functions r and q, such that $F_n(\bullet, k)$ is bijective for each $k \in \mathbb{F}_2^{q(n)}$, and

- F is efficiently computable,
- there is an efficiently computable family $G = (G_n)_{n\in\mathbb{N}}$ of corresponding decryption functions.

Definition 7 An **known plaintext attack** on a symmetric cipher F is a family $H = (H_n)_{n\in\mathbb{N}}$ of maps

$$H_n \colon \mathbb{F}_2^{r(n)s(n)} \times \mathbb{F}_2^{r(n)s(n)} \longrightarrow \mathbb{F}_2^{q(n)}$$

with

$$F_n(x_i, H_n(x_i, y_i)) = y_i \quad \text{for } i = 1, \ldots, s(n)$$

for all possible pairs $x, y \in \mathbb{F}_2^{r(n)s(n)}$.

F is called a **strong symmetric cipher** if each known plaintext attack on F is hard.

Defining a hash function is even more tricky. We omit it.