**Public-key Encryption** November 4, 2012

# Lecture 1

*Lecturer: Dominique Schröder*

## 1 Introduction

These notes should only supplement the lectures giving in class. They consist mainly of the technical aspects and do not cover all discussions and ideas.

## 2 Preliminaries

We review some basic notations and definitions that we will use during the entire course. The security parameter $\lambda$ is an additional input to all algorithms. To guarantee that the algorithms run in polynomial time in their input length, we provide $\lambda$ in unary and it is denoted by $1^\lambda$.

**Definition 1** An algorithm $A$ that gets as input a string of size $\lambda$ is a polynomial-time algorithm, if it runs in time $\mathcal{O}(\lambda^c)$ for some constant $c$. We write $y \leftarrow A(x)$ to denote the output of $A$ on input $x$. $\diamondsuit$

Now, we define algorithms that run in polynomial-time, but make random choices during their executions. Although these algorithms make random choices, they always terminate after a polynomial number of steps.

**Definition 2** A polynomial-time algorithm $A$, often dented by PPT, is a probabilistic polynomial-time algorithm, if it gets as input a string $x$ and some randomness $r$. We write $y \leftarrow A(x; r)$ to denote the output of $A$ on input $x$ using randomness $r$. $\diamondsuit$

### 2.1 Negligible Functions

In many cases, we need to bound the probability that some undesired event happens. Such an event could be that the adversary guesses a message or a secret key. In cryptography, we often cannot prevent these events, but the probability that they happen is so small that one could never notice it. To bound this probability, we recall the notion of negligible functions. A function $f$ is negligible if it grows smaller than any inverse polynomial. We formalize this notion as follows:

**Definition 3** A function $f$ is negligible in terms of $\lambda$, denoted by $\mathsf{negl}(\lambda)$, if for all integer $c$, there exists a $\lambda'$, such that for all $\lambda > \lambda'$ it holds that $f(\lambda) < 1/\lambda^c$. $\diamondsuit$

**Example 4** The following functions are all negligible in $\lambda$:

- $\delta(\lambda) = 2^{-\lambda}$
- $\delta(\lambda) = \lambda^{-\log \lambda}$.

These functions are $\mathsf{not}$ negligible in $\lambda$:

- $\delta(\lambda) = \mathsf{const} > 0$
- $\delta(\lambda) = 1/p(\lambda)$.

$\diamondsuit$

Negligible functions fulfill the following properties:

**Proposition 1** *Let $f_1$ and $f_2$ be negligible functions. Then,*

1. *The function $f_3$ defined by $f_3(\lambda) = f_1(\lambda) + f_2(\lambda)$ is negligible.*
2. *For any positive polynomial $p$, the function $f_4$ defined by $f_4(\lambda) = p(\lambda) \cdot f_1(\lambda)$ is negligible.*

The second property is important for cryptographic applications as it implies that repeating the algorithm polynomially many times (in the input length) yields an algorithm that has also negligible success probability.

## 3 One-Way and Trapdoor Functions

We review the well known definitions of one-way functions, permutations, and trapdoor permutations.

### 3.1 One-Way Functions and Permutations

A one-way function (OWF) is a function that is easy to compute, but hard to invert. OWFs are the most basic primitive and they are necessary to realize most cryptographic schemes. Unfortunately, researchers have not been able to prove that one-way functions exist. Hence, the existence of one-way functions is a necessary assumption. We begin with a syntactic definition of a function family and define one-wayness afterwards.

**Definition 5** A function family is a tuple of algorithms $(\mathsf{Gen}, \mathsf{Sample}, \mathsf{Eval})$, where

$\mathsf{Gen}(1^\lambda)$**:** The generation algorithm is a PPT algorithm that takes as input the security parameter. It outputs a value $i$ (one could think of $i$ as indexing a function $f_i$ over some domain $D_i$).

$\mathsf{Sample}(i)$**:** If $i$ was generated by $\mathsf{Gen}$, then the probabilistic sampling algorithm outputs an element $x$ that is uniformly distributed in $D_i$. Formally, this means that the distribution $\{\mathsf{Sample}(i)\}$ is equal to the uniform distribution over $D_i$.

$\mathsf{Eval}(i, x)$**:** If $i$ was generated by $\mathsf{Gen}$ and $x \in D_i$, then the deterministic evaluation algorithm outputs an element $y \in D_i$.

$\diamondsuit$

**Definition 6** A function family $(\mathsf{Gen}, \mathsf{Sample}, \mathsf{Eval})$ is one-way if for all PPT algorithms $\mathcal{A}$ the probability that the experiment $\mathsf{OW}_{\mathcal{A}}$ evaluates to 1 is negligible in $\lambda$, where

**Experiment** $\mathsf{OW}_{\mathcal{A}}(\lambda)$
```
    $i \leftarrow \mathsf{Gen}(1^\lambda)$
    $x \leftarrow \mathsf{Sample}(i)$
    $y \leftarrow \mathsf{Eval}(i, x)$
    $x' \leftarrow \mathcal{A}(i, y)$
Return 1 iff $\mathsf{Eval}(i, x') = y$.
```

$\diamondsuit$

We will use the notation of experiments extensively throughout this course. The experiment represents the probability that a particular event happens after a certain sequence of executions. The first step of the experiment is to pick an index $i$ of the function and to seelect a random element $x$ from the set $\{0,1\}^\lambda$. In particular, $x \leftarrow \{0,1\}^\lambda$ denotes selecting $x$ uniformly at random from $\{0,1\}^\lambda$ (or more general, from an arbitrary set $S$). The adversary $\mathcal{A}$ is run with uniformly-chosen randomness $r$. We often use $A(y)$ as an abbreviation of $A(y; r)$.

**Remark 7** Even moderate assumptions, such as $\mathcal{P} \neq \mathcal{NP}$, are (currently) not sufficient to prove the existence of one-way functions. However, it is widely believed the OWF exist and there is no *a-priori* reason to believe that they do not exist. A long sequence of outstanding results shows that one-way functions are sufficient to realize all primitives belonging to Impagliazzo's *minicrypt* world, including private-key encryption, messages authentication codes, and signature schemes. In the following we will discuss some (good) candidates. $\diamondsuit$

A slightly stronger primitive are one-way *permutations* (OWP). Loosely speaking, a one-way permutation is a one-way function with the property that every image $y$ has a unique pre-image.

**Definition 8** A function $f$ is a one-way permutation if $f$ is a one-way function and $f$ is a permutation. $\diamondsuit$

## 3.2 Trapdoor Permutations

Trapdoor permutations (TDP) are similar to one-way permutations, but TDPs have the additional property that there exists a secret information, the trapdoor, that allows the efficient inversion of the function. Following [1], we give two different definitions of TDPs. The first definition is rather formal, but maps to all known TDP candidates. The second definition is somewhat easier to understand and work with, but does not necessarily cover the (conjectured) instantiations. The generic constructions that we discuss in this class, however, can easily be modified to hold for the first definition as well.

**Definition 9** A *trapdoor permutation* family $\mathcal{F}$ is a tuple of PPT algorithms ($\mathsf{Gen}$, $\mathsf{Sample}$, $\mathsf{Eval}$, $\mathsf{Invert}$) such that:

$\mathsf{Gen}(1^\lambda)$**:** The key generation algorithm outputs a pair $(i, \mathsf{td})$, where $i$ defines a particular permutation $f_i$ over some domain $D_i$ and $\mathsf{td}$ represents some "trapdoor".

$\mathsf{Sample}(i)$**:** is defined analogously to Definition 3.1.

$\mathsf{Eval}(i, x)$**:** The deterministic evaluation algorithm outputs an element $y \in D_i$ (assuming $i$ was output by Gen and $x \in D_i$). Furthermore, for all $i$ output by Gen, the function $\mathsf{Eval}(i, \cdot) : D_i \mapsto D_i$ is a permutation. (Thus, one can view $\mathsf{Eval}(i, \cdot)$ as corresponding to a permutation $f_i$ mentioned above.)

$\mathsf{Invert}(\mathsf{td}, y)$**:** The deterministic inversion algorithm returns an element $x_i \in D_i$, where $(i, \mathsf{td})$ is a possible output of Gen.

The *correctness* condition is as follows: We require that for all $\lambda$, all $(i, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda)$, and all $x \in D_i$ we have $\mathsf{Invert}(\mathsf{td}, \mathsf{Eval}(i, x)) = x$. $\diamondsuit$

Note that the correctness condition guarantees that the inverse function $f_i^{-1}$ is *efficiently* computable given the trapdoor td. In general, however, computing the inversion function without the trapdoor is not possible in polynomial-time.

The following definition formalizes the intuition of being hard to invert.

**Definition 10** A trapdoor function family $\mathcal{F} = (\mathsf{Gen}, \mathsf{Sample}, \mathsf{Eval}, \mathsf{Invert})$ is one-way if for all PPT algorithms $\mathcal{A}$ the probability that the experiment $\mathsf{OW}_\mathcal{A}$ evaluates to 1 is negligible in $\lambda$, where

```
Experiment $\mathsf{OW}_\mathcal{A}(\lambda)$
    $(i, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda)$
    $x \leftarrow \mathsf{Sample}(i)$
    $y \leftarrow \mathsf{Eval}(i, x)$
    $x \leftarrow \mathcal{A}(i, y)$
Return 1 iff $\mathsf{Eval}(i, x) = y$.
```

$\diamondsuit$

## 3.3 A Simplified Definition of TDPs

In the following, we give a second definition of trapdoor permutations that is simpler to work with. While this definition maps our intuition for trapdoor permutations, it sometimes does not cover the trapdoor permutations that are used in practice.

The first simplification is that we assume that all $D_i$ are the same for a given security parameter $\lambda$ and that $D_i = \{0, 1\}^\lambda$ (i.e., the set of strings with length $\lambda$). The second simplification is to let the key generation algorithm return the function $f$ and it's inverse $f^{-1}$ instead of an index $i$ that maps to $f_i$. Technically, one could think of $f$ as being a description of the function). Also, even if we abuse the notations writing $f^{-1}$ it should be clear that the mathematical inversion function is not efficiently computable without knowing the trapdoor (i.e., the function exists, but it cannot be computed in polynomial time without the trapdoor).

**Definition 11** A *trapdoor permutation* family $\mathcal{F}$ is a tuple of PPT algorithms $\mathcal{F} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Invert})$ such that:

$\mathsf{Gen}(1^\lambda)$**:** The key generation algorithm outputs a pair $(f, f^{-1})$, where $f$ is a permutation over $\{0, 1\}^\lambda$.

$\mathsf{Eval}(f, x)$**:** The deterministic evaluation algorithm outputs an element $y \in \{0,1\}^\lambda$ (assuming $f$ was output by $\mathsf{Gen}$ and $x \in \{0,1\}^\lambda$). We often write $f(x)$ instead of $\mathsf{Eval}(f, x)$.

$\mathsf{Invert}(f^{-1}, y)$**:** The deterministic inversion algorithm returns an element $x_i \in \{0,1\}^\lambda$, where $(\cdot, f^{-1})$ is a possible output of $\mathsf{Gen}$ and we will write often $f^{-1}(y)$ instead of $\mathsf{Invert}(f^{-1}, y)$.

The *correctness* condition is as follows: We require that for all $\lambda$, all $(f, f^{-1}) \leftarrow \mathsf{Gen}(1^\lambda)$, and all $x \in \{0,1\}^\lambda$ we have $f^{-1}(f(x)) = x$. ♢

With this simplified definition, we also obtain a simpler definition of one-wayness that looks as follows.

**Definition 12** A trapdoor function family $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Invert})$ is one-way if for all PPT algorithms $\mathcal{A}$ the probability that the experiment $\mathsf{OW}_\mathcal{A}$ evaluates to 1 is negligible in $\lambda$, where

```
Experiment $\mathsf{OW}_\mathcal{A}(\lambda)$
    $(f, f^{-1}) \leftarrow \mathsf{Gen}(1^\lambda)$
    $x \leftarrow \{0,1\}^\lambda$
    $y \leftarrow f(x)$
    $x \leftarrow \mathcal{A}(i, y)$
Return 1 iff $f(x) = y$.
```

♢

## Further Reading

We discussed in class the relation between OWF and TDP. For additional reading about it, I suggest the paper [2].

## References

[1] Jonathan Katz, Lecture Notes on Advanced Topics in Cryptography, University of Maryland, USA.

[2] Russel Impagliazzo, A Personal View of Average-Case Complexity, *SCT '95 Proceedings of the 10th Annual Structure in Complexity Theory Conference (SCT'95).*