ETH Zürich
Department of Computer Science

Cryptography Foundations

Ueli Maurer

Spring 2019

Preface

Cryptography, known as a fascinating but difficult field, is a mathematical science providing the foundations and tools for constructing secure information systems.

The cryptography literature consists of a large body of proposed schemes and protocols, attacks against such schemes, security definitions, and security proofs. This course focuses on foundations of cryptography, i.e., on aspects of a general theory underlying the field. Our goal is to modularize statements and proofs so that the individual modules are simple, easy to understand, and reusable in different contexts, and so that they fit into a general theoretical framework. This is achieved by taking an abstract approach. We identify general principles and patterns that occur in different places in the literature on cryptography and aim at unifying them in a concise treatment.

This course builds on several other courses in the Computer Science curriculum at ETH (or taught elsewhere in a similar form). We need a good understanding of *Probability Theory*, basic algebraic concepts (e.g. groups) from *Discrete Mathematics*, and a basic understanding of algorithms, complexity, and reductions from *Theoretical Computer Science*. Moreover, some basic cryptographic schemes were discussed in *Information Security*. Despite these prerequisites, the course will be mostly self-contained, but certain topics that were treated in previous courses will be discussed quite rapidly, with references to the Appendix or to other lecture notes.

This course has benefitted enormously from discussions with many present and former postdocs and Ph.D. students. I would like to thank the following people for many helpful discussions and contributions to the course: Divesh Aggarwal, Joel Alwen, Christian Badertscher, Fabio Banfi, Sandro Coretti, Grégory Demay, Peter Gaži, Martin Hirt, Thomas Holenstein, Daniel Jost, David Lanzenberger, Chen-Da Liu Zhang, Christoph Lucas, Christian Matt, Marta Mularczyk, Krzysztof Pietrzak, Christopher Portmann, Dominik Raub, Pavel Raykov, Renato Renner, Guilherme Teixeira Rito, Gregor Seiler, Björn Tackmann, Stefano Tessaro, Zuzana Trubini, Daniel Tschudi, Stefan Wolf, Jiamin Zhu, and Vassilis Zikas.

The lecture notes will be made available as the course evolves. They contain only few figures; many figures accompanying the text will be drawn on the blackboard. It is highly recommended to take notes (especially figures) during the course. The lectures, the exercises, and the copies of slides handed out constitute an integral part of this course, also for the exam.

The course does not make use of textbooks, but we encourage the students to consult several books to get a more detailed and sometimes complementary view on the various topics. Some useful books on cryptography, each with a different focus, are listed below:

- O. Goldreich, *Foundations of Cryptography, Basic Tools*, Cambridge University Press, 2001
- J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, Chapman & Hall, 2015.
- W. Mao, *Modern Cryptography – Theory and Practice*, Prentice Hall, 2004.
- A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, Boca Raton: CRC Press, 1997.
- D.R. Stinson, *Cryptography – Theory and Practice*, CRC Press, 1995.
- N. P. Smart, *Cryptography Made Simple*, Springer Verlag, 2016.

Zurich, February 2019 Ueli Maurer

Chapter 1

Introduction

1.1 What Cryptography is About

1.1.1 The Role of Cryptography in Information Security

Cryptography[1] provides the core foundations for building secure information systems. It is an interdisciplinary mathematical science drawing on several fields, most importantly on computer science (theoretical computer science, complexity theory, algorithms, hardware, software), mathematics (algebra, number theory, probability theory, information theory, combinatorics, statistics, etc.), and physics (e.g. quantum physics, noise models, side-channel analysis, etc.).

To understand the mission of cryptography within information security, we observe that the field of information security can be classified coarsely into the following two sub-areas:

- **Correct system implementation.** Most of the security problems reported in the media relate to system failures and vulnerabilities exploited by an entity, often called intruder or adversary. The problem of building a secure system is hence, in principle, the same as more generally in software design, consisting of:
  - *specification* of the system,
  - *implementation* of the specification, ideally with a proof that the implementation meets the specification.

  This long-studied problem is intrinsically hard since the complexity of many systems (e.g. an operating system) is too high to be really fully specified. The lack of a clear specification and/or the lack of an implementation provably meeting the specification are at the core of today's security problems.
- **Virtual system emulation.** This can be seen as the **mission of cryptography**. The goal is to construct, by a cryptographic protocol, a system that behaves like a virtual system, in a world where such a system does not actually exist. A few quite diverse examples of such virtual systems are:
  - A **secure communication channel** which is emulated (using encryption) in a world where all communication channels are insecure (e.g. based on Internet communication).
  - A **voting system** to which every voter can send his/her vote and which tallies the votes and announces the result to the public. In a world where one does not trust a single computer system because it could be ill-designed or badly managed, cryptography can nevertheless emulate such a virtual voting system.
  - A **public ledger system** which can be achieved using distributed protocols often subsumed under the term "block-chain".

[1] One sometimes uses the term *cryptography* only for the science of *designing* cryptographic mechanisms, *cryptanalysis* for the science of *breaking* cryptographic systems, and *cryptology* as the term comprising both.

1.1.2 Cryptography – Science of Paradoxes

One can see cryptography as a *science of paradoxes*. It is hard to find another scientific discipline that has produced so many paradoxical discoveries.

Public-key cryptography is an apparently paradoxical concept. How should it be possible that two entities can communicate securely, without sharing a secret key initially, only by communication over a channel accessible to an adversary? Similarly, many of the achievements of modern cryptography are quite paradoxical:

- **Digital signatures.** A user can issue a bit-string as a digital signature for a certain digital contract such that everyone can verify the signature, yet nobody can forge a signature.
- **Interactive proofs and zero-knowledge protocols.** One can prove a mathematical statement (or the claim that one knows a certain secret value) without leaking any information about the proof (or the secret).
- **Secure multi-party computation.** An illustrative example of this general paradigm is the so-called millionaires' problem: Two millionaires want to find out who is richer, without telling each other anything about how much they own, except what is implied by who is richer. They want to achieve this solely by talking to each other or, in our terminology, by a protocol between the two. While this example is only a curiosity, many similar protocols have important potential applications, for example in e-voting.

- **Computing on encrypted data.** A server can perform computations on data stored on the server even though the data is encrypted and the server does not know the secret key. This can be possible due to a homomorphic property of the encryption.
- **Security against adversaries with infinite computing power.** In many cryptographic settings one can achieve security against an adversary with infinite computing power. In other words, even trying all posible keys does not help to break the system.
- **Anonymous digital cash.** A bank can issue bit-strings as digital bank notes or coins. Forging coins is impossible. The digital money is anonymous in the sense that, although the bank issues the money to a known customer, it cannot recognize a coin when it is deposited and hence cannot trace where a customer spent the money.

1.2 The Need for a Rigorous Mathematical Treatment

1.2.1 Security is not Testable

A main problem in research and practice in information security, and more specifically in cryptography, is that *one cannot test security*. One can only *prove* security, which requires (in the first place) a meaningful definition of security. There are at least two reasons why one can generally not test the security:

- Security is, informally, a statement of the form that no adversary in a certain class $\mathcal{A}$ of adversaries can cause a problem. Since the set $\mathcal{A}$ of such potential adversaries is infinite, one cannot test the system for each adversary.
- Even for a single adversary, testing a system may be impossible because security-relevant properties might not be observable in the system. For example, information leakage generally does not change the system in an observable manner.

1.2.2 Bugs and Flaws

Many real-world cryptographic security protocols, like TLS, were designed in a quite ad-hoc manner, and they have a long history of being broken by novel attacks, against which patches were subsequently proposed. This cycle of breaks and patches must be interrupted by a rigorous security protocol design methodology that achieves provable security.

There are at least three types of flaws leading to a system being insecure:

- **Implmentation bugs**, e.g. buffer overflow, insecure operating systems, etc. This is a topic for software design methodologies, not for cryptography. This course in *not* about avoiding implementation bugs.
- **Protocol flaws**, e.g. the wrong order of authentication and encryption, a signature being computed over the wrong fields of a data record, or missing identifiers in protocol messages.[2] This course is about avoiding such flaws. The reason for protocol design flaws is the lack of understanding of what exactly cryptographic schemes achieve and how they can be composed in a sound manner.
- **Mismatch between a theoretical model and the real world**, e.g. timing attacks or other physical leakage of information (side channels). Every model abstracts away certain aspects of reality (e.g. time). This happens on purpose to modularize the analysis of complex systems. It should be possible to refine models to take into account more and more aspects of the physical world. Such issues are addressed in this course.

1.2.3 Towards a Rigorous Approach

In this course we take a rigorous approach to understanding cryptography and designing cryptographic protocols. The goal is to precisely specify the security goals a protocol must achieve and to design a protocol that can be mathematically proven to be secure, usually based on a list of well-defined assumptions (like the hardness of a certain computational problem). This approach should be contrasted with the approach often taken in practice where a protocol design is based on heuristics and the security goals are not even precisely specified, let alone the security proved. Security statements are mathematical statements and require a proof.

A rigorous approach must answer the following three questions:

- How can we *define* security of a cryptographic scheme (e.g. encryption)?
- How can we *prove* the security of a cryptographic scheme (or protocol)?
- How can we *use* a secure scheme in a larger context?

There are many reasons for taking a rigorous approach, including:

- It is necessary to be relevant in practice.
- It is no fun to teach a course which explains only intuitive statements and "how it's done in practice" without clear justification.
- We want to move from a "bag of tricks" towards a unified science.

[2] These are flaws in the protocol *specification* and are hence also present in any (even correct) implementation.

1.3 Some Terminology

1.3.1 Cryptographic Schemes and Protocols

We discuss a few important cryptographic terms and their relation.

A *cryptographic scheme* consists of one or several functions. For example, a symmetric encryption scheme consists of a pair of functions, an encryption function and a decryption function. Other types of cryptographic schemes are message authentication schemes, hash functions, public-key encryption schemes, digital signature schemes, commitment schemes, etc.

A *cryptographic protocol* for a given set of parties consists of, for each party, a precisely specified behavior in the interaction with the other parties, for example in the form of a program that must be executed by the party. A cryptographic protocol typically makes use of several cryptographic schemes in a well-defined manner. For example, a secure communication protocol for two parties, Alice and Bob, specifies, typically, how Alice uses an encryption function and a message authentication function to transform a message before sending it over the Internet. Other examples are e-voting protocols, anonymous e-cash protocols, and so-called secure multi-party computation protocols.

The purpose of applying a cryptographic protocol is to achieve an *application goal*, for example to obtain a secure communication channel. The security of a protocol can only be judged relative to a declared (application) goal and relative to a specification of which parties must be tolerated to be dishonest.

1.3.2 Requirements

Cryptographic schemes and protocols need to satisfy the following three types of requirements:

- **Correctness:** The specification of what must be achieved, not considering an adversary. For example, an encryption scheme is correct if encryption followed by decryption results in the initial message.
- **Security:** The specification of what must be achieved when considering an adversary. For example, the adversary must not obtain any information about a message (which needs to be precisely defined).
- **Practicality:** The scheme must be usable. The most fundamental practicality requirement is **efficiency**: The functions (of a scheme) must be computable sufficiently quickly on a given computer.

Correctness is often included as a condition in the definition of a cryptographic scheme since it does not make sense to consider incorrect schemes.

In contrast, security is often considered as a separate definition, not included in the basic definition of a scheme. The reason is that one can define security in

several different ways, which can all be meaningful. One can then compare different security definitions for a given type of scheme.

1.3.3 A Discussion of the Efficiency Requirement

For the most part of this course, practicality is obvious or implicit. For example, it is obvious how the relevant functions can efficiently be computed, or they are even specified in terms of an efficient algorithm. Practicality will rarely be an explicit topic in this course.

This is in contrast to much of the cryptographic literature, where often *the efficiency condition is part of a definition*. For example, a cryptographic scheme is defined to consist of *efficient algorithms* (as opposed to functions). This approach has little benefit but two very significant drawbacks:

- It requires a mathematical definition of an algorithm, including a specific computational model. The typical model used in the literature is that of a Turing machine which, however, is never precisely defined. This implies that the statements must hold for a large class of computational models, which in turn implies that the statements are unavoidably *asymptotic*.
- It requires a mathematical definition of *efficiency*, i.e., a clear distinction between an algorithm that is efficient and one that is not, and such a distinction can (again) only be meaningfully made in asymptotic terms, for instance by defining efficient as *polynomial-time*.[3]

It is important to distinguish between the concept of a *function* and the concept of an *algorithm* for computing the function. For the most part of this course, we consider functions rather than algorithms.

Let us explain the significance of the separation between function and algorithm. In many definitions one finds in the literature, the practicality condition is included as the requirement that the involved functions must be *efficiently computable*. However, this efficient computability condition is not exactly what is required (in a mathematical sense) since one actually needs an efficient algorithm for computing the function, not just that such an algorithm exists (possibly without knowing one explicitly). This is one of many examples that illustrates the subtleties and difficulties in developing a meaningful and precise mathematical theory of cryptography.

1.4 Towards Defining Security

In this section we discuss a number of conceptual aspects arising when trying to define security.

[3] There are actually several different ways in which one can define polynomial-time for interactive systems.

1.4.1 Security of Schemes vs. Security of Applications

Most security definitions in the literature are security definitions for *schemes*, for example for an encryption scheme or a digital signature scheme. Such a definition is useful as the target which the designer of a cryptographic scheme has to meet. The definition serves as the interface between the designers of such schemes and the designers of cryptographic protocols making use of such schemes.

However, what one is actually interested in is that the *application* is secure, for example that transmitted messages are inaccessible for an adversary. It is generally not clear whether and how the security definitions for the *schemes* used imply the security of the application. This obviously depends on the particular way in which the scheme is used in the application and does not depend on the scheme (and its security definition) alone. For example, whether transmitting encrypted messages is secure generally depends on whether the transmission channel is authenticated or not.

In fact, for a given type of scheme there are usually several different security definitions, which shows that the required security definitions for the schemes used in an application may depend on the concrete application. Often, such security definitions can be compared in the sense that one definition is strictly stronger than another one, or that they are (mathematically) equivalent. Some of the definitions proposed in the literature may turn out to be useless (for applications) while other definitions may turn out to be too strong.

In this course we put strong emphasis on defining and proving the security of applications and understanding exactly which security definition of an underlying cryptographic scheme is required.

1.4.2 Attack-game-based vs. "Ideal-world" Security Definitions

Roughly speaking, one can distinguish the following approaches to making precise security definitions in cryptography.

- **Attack-based (or game-based) security definitions.** One defines an *attack game* capturing what an adversary might try to achieve (e.g. performing a so-called chosen-ciphertext attack and trying to guess some bit of a confidential message). Security means that no adversary from a specific class of considered adversaries (see Section 1.4.3) can win this game with substantial probability.

- **Ideal-world security definitions.** One defines an ideal system which is secure by definition, i.e., where an adversary is by definition limited appropriately (for example the adversary can, by definition, not receive the confidential messages). The actual protocol runs in the so-called real world. Security means that whatever the adversary can achieve in the real world, where the protocol is run, she can also achieve in the ideal world where the ideal system is used, thus showing that the real world is not worse (for the honest parties) than the ideal world. This argument is made by means of a so-called *simulator* allowing the adversary to emulate the real-world behavior in the ideal world.

  Constructive cryptography takes a new approach (but the ideal-world paradigm can also be expressed in constructive cryptography as a special type of statement). One gives an abstraction (or over-specification) of the actual real-world system by a system (which can be thought of as an idealized system) which is secure by definition. Hence one has proven that the real system is also secure (because it satisfies the specification of the idealized system). Such statements can naturally composed.

Attack-based security definitions focus on the adversary. Often it is not explicitly specified what the honest entities achieve, only what the adversary *can not* achieve. It is often difficult to guarantee that all relevant attacks have been included in the model, in particular because the relevant attacks might depend on the context where a system is used. One can criticize an attack-based security definition by exhibiting a cryptographic scheme satisfying the definition as well as an application in which using the scheme is insecure.

1.4.3 The Adversary – A Hypothetical Entity

The *adversary*[4] is a hypothetical entity introduced to define security. Security is defined via a thought experiment showing that such a hypothetical adversary, with certain assumed capabilities, is not able to do harm to a system (or, more precisely, to do harm to the honest entities using the system).

From a mathematical perspective, the adversary stands for a universal quantifier ($\forall$) used to state that for all elements in a certain set (the set of all considered adversaries), a certain statement is true.

More generally, every entity or party in a protocol can be considered as being potentially dishonest. This is often captured in cryptography by considering a (central) adversary who can *corrupt* parties. But more generally one wants to also capture the fact that the entities can have conflicting goals and that there can therefore be several independent misbehaving entities. However, in this course we concentrate mostly on the setting with a single potentially dishonest party. Moreover, information-theoretically secure systems can be used as building blocks in systems that are, overall, only computationally secure.

For any security consideration to be meaningful, one must specify the assumptions about the hypothetical adversary. Such an adversary specification

[4] The adversary is sometimes also called *attacker, opponent, eavesdropper, cryptanalyst*, or Eve or Mallory. Sometimes a distinction is made between Mallory, an actively cheating adversary, and Eve who only intercepts the communication without actively cheating. For us, Eve is the general adversary.

includes at least the following parameters:

- **Computing power** (see Section 1.4.4 below).
- **Memory capacity.** One usually allows the adversary unbounded memory, except in the so-called *bounded-storage model*.
- **Attack capability.** An adversary may be able to perform certain attacks, for instance a so-called chosen-message attack in which he can choose a plaintext and obtain the corresponding ciphertext (for the secret key actually in use).

An important principle was stated by A. Kerckhoffs (1835-1903):

> **Kerckhoffs' principle:** "A cryptographic system should be designed to be secure when the adversary knows all details of the system, except for the values explicitly declared to be secret (e.g. secret keys)."

1.4.4 Information-theoretic vs. Computational Security

The adversary's computing power can be assumed to be bounded or unbounded:

- **Infinite computing power.** A system secure without any assumption on the adversary's computing power is called *information-theoretically secure* or sometimes *unconditionally secure*.
- **Bounded computing power.** One specifies an upper bound on the computing power, in some model of computation, and considers the time required to break the system, using the fastest algorithm. A system secure under a reasonable assumption on the adversary's computing power is called *computationally secure*.

Information-theoretic security is of course more desirable than computational security, but it is usually more difficult to achieve and the practicality of such systems is often quite limited. (It may come as a surprise that such systems exist at all.) Therefore, most practical systems are not information-theoretically secure, and one must resort to computational security, based on some intractability assumption (e.g. that factoring large integers is hard). However, it is important to note that none of these intractability assumptions has been proved, and hence no cryptosystem has been proven to be computationally secure.

1.4.5 Modularity and Composability

We want to understand cryptography as a constructive discipline. A central paradigm in any constructive discipline is the decomposition of a complex system into simpler component systems or modules, each of which may consist

of yet simpler modules, and so on. This paradigm, sometimes called step-wise refinement, is useful only if the composition of modules is well-defined and preserves the relevant properties of the modules. For example, in software design, the composition operation must preserve correctness of the modules, i.e., a system consisting of correct modules must itself be correct.

In the context of cryptography, composition should mean that a protocol proved secure in isolation should remain secure if plugged into higher-level protocols or if run in parallel with other protocols. This can only be the case if the security definitions are carefully chosen.

For attack-based security definitions, the composition property is often unclear, or does not hold, or at best is highly non-trivial. One consequence of the lack of composability is that proposed security protocols are often not well-motivated, sometimes unnecessarily complex, and sometimes insecure.

1.4.6 The Role of Assumptions

The security of every cryptographic system depends on certain assumptions about the physical world. Three obvious assumptions, which are usually not even stated explicitly, are:

- **Randomness exists.** This assumption is needed, for example, for the generation of secret keys.[5]
- **Independence exists.** Observations at different locations can be independent. For example, the outcome of flipping a coin is assumed to be independent of anything an adversary can observe.
- **The laws of physics are correct**, i.e., non-physical phenomena, such as reading somebody's mind, do not exist.

More generally, we make the assumption that our understanding of physics is sufficiently correct. Some security proofs are based explicitly on *physical assumptions*, for instance that quantum theory is correct, that certain devices are tamper-resistant, or that a communication channel is subject to at least a minimal noise level.

Other crucial assumptions made explicitly in cryptography are:

- **Trustworthiness** of certain entities (e.g. a certification authority).
- **Computational hardness** of certain computational problems.[6]

The goal of cautious cryptographic design is to make these assumptions both explicit and **as weak as possible**. Reducing the required assumptions and trust

[5] Quantum theory was the first physical theory in which randomness is an inherent feature.

[6] We will introduce a way of capturing security against computationally bounded adversaries without the need to define the hardness of a problem, which is a binary definition (a problem is hard, or it is not).

requirements for achieving a certain security objective is a major theme of research in cryptography.

1.5 Categories of Cryptography Research

The goals and tasks of research in cryptography can be roughly classified into the following four categories:

- **Propose cryptographic schemes and protocols**, for example a new public-key cryptosystem. Often, the claim that the cryptosystem is secure corresponds to making a new computational assumption.
- **Security proofs** for particular schemes or protocols.
- **Theoretical foundations.** A goal in itself is to develop a general theory of security statements and reductions. This course presents basic elements of such a theory.
- **Cryptanalysis** is concerned with proving that a certain assumption is false, (e.g. by exhibiting an efficient algorithm for breaking a proposed cryptographic scheme).

Chapter 2

Preliminaries

2.1 Three Types of Computational Problems in Cryptography

In this section we discuss three types of computational problems of interest in cryptography, and more generally in computer science. A more formal discussion will follow later. The object attemting to solve a given computational problem is called a *solver*.

2.1.1 Games

A very important type of computational problem in computer science are search problems where the task of a solver is to find a solution in a certain search space. An example is the problem of finding a Hamiltonian cycle in a graph. Another example is the integer factorization problem.

A game is a generalization of a search problem and is defined by a system G (containing internal randomness) to which a solver (in the context of games also called a winner) W has access (see Figure 2.1). For the game G a winning condition is defined, shown in Figure 2.1 as a binary output A on the right side of G. The quantity of interest is defined as follows.

Definition 2.1. The winning probability of W for game G is

$$\Gamma^W(G) := \mathsf{Pr}^{WG}(A = 1).$$

Examples of games are the collision-finding problem for a hash function and the discrete logarithm problem, both discussed later.

Figure 2.1: The winner W connected to the game G, producing a binary output A indicating whether the game is won.

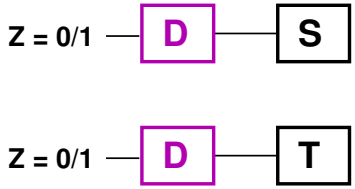

Figure 2.2: The distinction problem of distinguishing S and T. The goal is that D outputs 1 if connected to T and 0 if connected to S.

2.1.2 Distinction Problems

Two examples of distinction problems discussed later are:

- distinguishing the output of a PRG from a uniform random string, and
- distinguishing a block cipher (with a secret random key) from a uniform random function.

A distinction problem (see Figure 2.2) is defined by two systems, say S and T. A so-called distinguisher D is connected to one of the systems and produces a binary output Z. The task of a distinguisher D is to distinguish S and T. The goal of a distinguisher is to always output $Z = 1$ if connected to T and to never output $Z = 1$ (i.e., always output $Z = 0$) if connected to S. This is captured in the following definition defining how good a distinguisher is. This interpretation will be discussed in the next section.

Definition 2.2. The *advantage* of D in distinguishing S and T is[1]

$$\Delta^D(S,T) \;:=\; \mathsf{Pr}^{DT}\big(Z=1\big) - \mathsf{Pr}^{DS}\big(Z=1\big). \tag{2.1}$$

Remark: Later we will denote the described distinction problem by a special symbol, namely $\langle S \,|\, T \rangle$. Consequently, one could actually write the distinguishing advantage of distinguisher D for distinction problem $\langle S \,|\, T \rangle$ as $\Delta^D\big(\langle S \,|\, T \rangle\big)$, and we can understand $\Delta^D(S,T)$ as a short-hand notation for this.

[1]Note that, as indicated, the two probabilities in this definition are defined in different random experiments, namely the random experiment of selecting D and S independently, and the random experiment of selecting D and T independently.

Distinguisher classes. In the cryptographic literature, one usually considers the advantage of the *best* distinguisher, where possibly one considers a restricted set $\mathcal{D}$ of distinguishers (e.g. the class of feasibly implementable distinguishers, for some notion of feasibility).

Definition 2.3. The maximal advantage of distinguisher class $\mathcal{D}$ is

$$\Delta^{\mathcal{D}}(S,T) \;:=\; \sup_{D\in\mathcal{D}} \Delta^D(S,T).$$

Moreover, if the class of *all* distinguishers is considered, then we write

$$\Delta(S,T) \;:=\; \sup_{D} \Delta^D(S,T).$$

One can easily show that probabilistic distinguishers are not more powerful than deterministic ones in the sense that the best deterministic distinguisher has the same advantage as the best probabilistic distinguisher.

In the literature, the distinguishing advantage of D is often defined as the *absolute value* of $\Delta^D(S,T)$, but is nevertheless written as $\Delta^D(S,T)$.

Distinguishing random variables. The distinguishing concept is perhaps best illustrated for the problem of distinguishing two random variables X_0 and X_1 with range $\mathcal{X}$. As the class $\mathcal{D}$ of distinguishers we can consider the set of all functions $\mathcal{X} \to \{0,1\}$.

Definition 2.4. The *statistical distance* of two random variables X and Y over a finite set $\mathcal{X}$, denoted $\delta(X,Y)$ (or sometimes $\|P_X - P_Y\|$), is defined as

$$\delta(X,Y) \;:=\; \tfrac{1}{2}\sum_{x\in\mathcal{X}} \big|P_X(x) - P_Y(x)\big|\,.$$

Lemma 2.1. *The advantage of the best distinguisher for X and Y is the statistical distance $\delta(X,Y)$ between X and Y, i.e.,*

$$\Delta(X,Y) \;=\; \delta(X,Y).$$

Proof. Left as an exercise. Note that a distinguisher for such random variables is a (possibly probabilistic) function $\mathcal{X} \to \{0,1\}$. □

The following lemma, which has a straight-forward proof, allows to bound the distinguishing advantage between two systems S_0 and S_k in terms of the distinguishing advantages of intermediate (hybrid) systems. The application of this lemma is often called a *hybrid argument*.

Lemma 2.2. *For any D,*

$$\Delta^D(S_0,S_k) \;=\; \Delta^D(S_0,S_1) + \Delta^D(S_1,S_2) + \cdots + \Delta^D(S_{k-1},S_k).$$

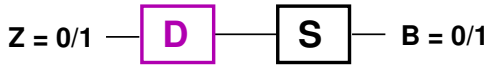

Figure 2.3: The bit-guessing problem (S, B) and a distinguisher D trying to guess the bit B by the bit Z, the goal being that $Z = B$.

2.1.3 Bit-guessing Problems

An example of a bit-guessing problem discussed later is the so-called IND-CPA game occurring in a security definition of encryption schemes.

game discussed above is an example of a specific type of computational problem arising in cryptography, which we call bit-guessing problem.[2]

A bit-guessing problem (see Figure 2.3) consists of a system S and a binary random variable B (correlated with S), which can be thought of being output on the right side of S. Note that, in contrast to a game, the bit B is fixed from the beginning, independently of the interaction on the left side of S.

The task of a distinguisher D, with access to S, is to guess B. Since a random guess yields success probability $\frac{1}{2}$, we want to measure the performance of D only in terms of by how much the success probability exceeds $\frac{1}{2}$. Moreover, we calibrate the advantage so that it is 1 if D is always correct and -1 if it never correct.

Definition 2.5. The *advantage* of D (in guessing bit B) is[3]

$$\Lambda^D\big([\![S;B]\!]\big) \;:=\; 2\cdot\Big(\mathsf{Pr}^{D(S,B)}(Z=B)-\tfrac{1}{2}\Big)\,.$$

2.1.4 The Relation between Distinction and Bit-guessing Problems

Bit-guessing problems are closely related to distinction problems. To demonstrate this, consider two systems S_0 and S_1 and the bit-guessing problem $[\![S_U;U]\!]$, where U is an unbiased random bit (independent of anything else). More precisely, the pair (S_U, U) is chosen by first choosing a uniformly random bit U and then, if $U = 0$ (or if $U = 1$), choosing S_0 (or S_1) according to its distribution. The task of D is hence to guess whether it is connected to S_0 or S_1 when one of the two is randomly selected. Then we have:

[2]The term "game" (often used in the literature) is used here only in the sense of Section 2.1.1. Deciding between only two options is a bit-guessing problem.

[3]Recall that $\mathsf{Pr}^{D(S,B)}$ stands for probabilities in the random experiment of selecting D and the (correlated) pair (S, B) of random variables independently of D. Moreover, D can only interact with S and has no access to B.

Lemma 2.3.

$$\Lambda^D\big([\![S_U;U]\!]\big) \;=\; \Delta^D(S_0,S_1).$$

Proof. For any D we have, according to Definition 2.5,

$$\begin{aligned}
\Lambda^D\big([\![S_U;U]\!]\big) &= 2\cdot\Big(\mathsf{Pr}^{D(S_U,U)}(Z=U)-\tfrac{1}{2}\Big)\\
&= 2\cdot\Big(\tfrac{1}{2}\mathsf{Pr}^{DS_0}(Z=0)+\tfrac{1}{2}\mathsf{Pr}^{DS_1}(Z=1)-\tfrac{1}{2}\Big)\\
&= 2\cdot\Big(\tfrac{1}{2}\big(1-\mathsf{Pr}^{DS_0}(Z=1)\big)+\tfrac{1}{2}\mathsf{Pr}^{DS_1}(Z=1)-\tfrac{1}{2}\Big)\\
&= \mathsf{Pr}^{DS_1}(Z=1)-\mathsf{Pr}^{DS_0}(Z=1)\\
&= \Delta^D(S_0,S_1),
\end{aligned}$$

where we have used Definition 2.2 in the last step. □

Conversely, consider a bit-guessing problem $[\![S;B]\!]$. We can consider distinguishers that have access to a system and a bit and the problem of distinguishing the combined system,[4] denoted $[S, B]$ from the combined system $[S, U]$ where the bit U is uniform and independent of S. In the former case the distinguisher sees S and the bit B shown on the right side of the system S. In other words, the distinguisher is given access to S as well as a bit which is either the bit B correlated with S or an independent uniformly random bit U. The following lemma states (in concrete terms) that if the advantage in guessing bit B when given access to system S is small, then also the advantage in distinguishing the pair $[S, B]$ from the pair $[S, U]$ is small. Therefore one can use the bit B more or less like as if it were an independent (of S) uniform bit. In other words, a hard-to-guess bit behaves, in a computational sense, essentially like an independent uniform random bit.

Lemma 2.4. *For any distinguisher D and for a simple mapping[5] $D \mapsto D'$ we have*

$$\Delta^D\big([S,B],[S,U]\big) \;=\; \tfrac{1}{2}\Lambda^{D'}\big([\![S;B]\!]\big).$$

Proof. Left as an exercise. □

2.2 Computational Hardness and Reductions

Most security definitions in crytography are relative to the hardness of a certain computational problem. In this section we explain this way of thinking, but we

[4]We denote pairs (or more general tuples) with square brackets if we think of them as being a single object giving access to both (sub-)objects, here the sub-objects S and B. Note that the double square bracket used for denoting bit-guessing problems is *different* from the single square bracket used to denote combined systems.

[5]A formal version of this lemma would have to specify this mapping, but here we leave finding it as an exercise.

point out that later we will present a way of thinking that does not require the definition of computational hardness.

Informally, a problem is hard if no feasible algorithm (for an appropriate notion of feasibility) can solve the problem with success probability more than a negligible value (for an appropriate notion of negligible). In other words, any algorithm with a non-negligible success probability is infeasible.[6]

2.2.1 Hardness Implications and Weakening Assumptions

Since actual hardness proofs for computational problems of interest and for relevant computational models remain unachievable for the foreseeable future, possibly forever, one is usually interested in proving *implications* of the form:

If problem P is hard, then problem Q is hard.

Such a statement is of interest if it appears (*a priori*) more credible that P is hard than that Q is hard. Such implications can be composed.

A relevant research goal in cryptography is to prove the hardness of breaking a certain cryptographic scheme under an apparently very weak assumption. One therefore tries to prove hardness *implications*, as explained above, where P is a computational problem whose hardness is much more credible than the hardness of breaking the considered cryptographic scheme.

To weaken cryptographic assumptions is both a fundamental challenge in the theory of cryptography and at the same time a major goal for the cautious and prudent design of practical cryptographic systems.

2.2.2 Computational Reductions

Hardness implication results are proved by so-called *reductions*, a concept we now briefly discuss. A more thorough treatment will be given in Chapter 5.

Consider two computational problems, P and Q, and the following statements A and B:

$$\begin{aligned} A &= \text{“}P \text{ is computationally hard”} \\ B &= \text{“}Q \text{ is computationally hard”} \end{aligned}$$

A typical setting is that A is an assumption and B should be proved under this assumption. In other words, one wants to prove the implication

$$A \implies B.$$

[6] In the cryptographic literature, generally no distinction is made between the concept of efficiency in the sense of practicality (for the users) and the notion of feasibility, implying the notion of infeasibility (for an adversary), and both are called *efficient* and instantiated in the same concrete manner, namely as polynomial-time computable.

Such a statement is usually proved using that fact that $A \implies B$ is logically equivalent to

$$\neg B \implies \neg A.$$

Here $\neg B \implies \neg A$ can be read as follows: If there exists a feasible algorithm for solving Q, then there also exists a feasible algorithm for solving P. This proof is usually carried out in a constructive manner, by exhibiting a feasible algorithm R, called a *reduction*, which can invoke any algorithm solving Q, and thereby solve P.

The relevance of a reduction is crucially determined by two factors:

- the loss (or gain) in success probability incurred by R, and
- the loss in efficiency incurred by R.

For example, whether R invokes the given algorithm for Q a billion times or only once makes a significant difference for the strength of the corresponding hardness implication statement.

We will discuss reductions in Chapter 5.