

# Introduction to Cryptography

## 89-656

Yehuda Lindell<sup>1</sup>

October 19, 2006

<sup>1</sup>This is an *outdated draft* of lecture notes written for an undergraduate course in cryptography at Bar-Ilan University, Israel. The notes are replaced by the textbook *Introduction to Cryptography* by Jonathan Katz and myself. There is a significant difference between the presentation in these notes and in the textbook (and thus how we will teach in class).

© Copyright 2005 by Yehuda Lindell.

Permission to make copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that new copies bear this notice and the full citation on the first page. Abstracting with credit is permitted.

# Abstract and Course Syllabus

## Abstract

The aim of this course is to teach the basic principles and concepts of modern cryptography. The focus of the course will be on cryptographic problems and their solutions, and will contain a mix of both theoretical and applied material. We will present definitions of security and argue why certain constructions meet these definitions. However, these definitions and arguments will be rather informal. (A rigorous treatment of the theory of cryptography will be given in course 89-856 next semester.) There is no one text book that covers all of the material in the course (nor one that presents the material in the same way as we do). However, much of the material can be found in the textbooks of [36] and [37] in the library.

## Course Syllabus

1. (a) **Introduction:** what is modern cryptography (what problems does it attempt to solve and how); the heuristic versus the rigorous approach; adversarial models and principles of defining security.  
(b) **Historical ciphers and their cryptanalysis**
2. **Perfectly secret encryption:** definitions, the one-time pad and its proof of security; proven limitations, Shannon's theorem.
3. (a) **Pseudorandomness:** definition, pseudorandom generators and functions.  
(b) **Private-key (symmetric) encryption schemes:** Definition of security for eavesdropping adversary, stream ciphers (construction from pseudorandom generators).
4. **Private-key encryption schemes:**
  - (a) **Block ciphers:** CPA-secure encryption from pseudorandom permutations/functions.
  - (b) **The Data Encryption Standard (DES).**
5. **Private-key encryption (continued):**
  - (a) **DES (continued):** Attacks on reduced-round DES; double DES and triple DES.
  - (b) **Modes of operation:** how to encrypt many blocks.
6. **Collision-resistant hash functions:** definition, properties and constructions; the random oracle model
7. **Message authentication:** definition, constructions, CBC-MAC, HMAC
8. (a) **Combining encryption and authentication:** how and how not to combine the two.

- (b) **CCA-secure encryption:** definition and construction.
  - (c) **Key management:** the problem, key distribution centers (KDCs), key exchange protocols
9. **Public-key (asymmetric) cryptography:** introduction and motivation, public-key problems and mathematical background (Discrete Log, Computational and Decisional Diffie-Hellman, Factoring, RSA), Diffie-Hellman key agreement
  10. **Public-key (asymmetric) encryption schemes:** the model and definitions, the El-Gamal encryption scheme, the RSA trapdoor one-way permutations, RSA in practice
  11. **Attacks on RSA:** common modulus, broadcast, timing attacks
  12. **Digital signatures and applications:** definitions and constructions (in the random oracle model), certificates, certificate authorities and public-key infrastructures.
  13. **Secure protocols:** SSL, secret sharing

Much of the material in the course (but far from all of it) can be found in [36] and [37]. Other texts of relevance are [17] and [28].

**A note regarding references:** Some references are presented throughout the lecture notes. These reference are not supposed to be citations in the classic sense, but rather are pointers to further reading that may be of interest to those who wish to understand a topic in greater depth. As such, the references are not complete (in fact, there are many glaring omissions).

# Contents

<b>1</b>	<b>Introduction and Historical Ciphers</b>	<b>5</b>
1.1	Introduction . . . . .	5
1.2	Historical Ciphers and Their Cryptanalysis . . . . .	7
<b>2</b>	<b>Information Theoretic Security</b>	<b>11</b>
2.1	Preliminaries . . . . .	11
2.2	Perfect Secrecy . . . . .	12
2.3	The One-Time Pad (Vernam's Cipher) . . . . .	13
2.4	Limitations of Perfect Secrecy . . . . .	14
2.5	Shannon's Theorem . . . . .	14
<b>3</b>	<b>Pseudorandomness and Private-Key Encryption Schemes</b>	<b>17</b>
3.1	Pseudorandomness . . . . .	17
3.2	Symmetric (Private-Key) Encryption . . . . .	18
3.2.1	Defining Security . . . . .	18
3.2.2	Constructing Secure Encryption Schemes . . . . .	19
<b>4</b>	<b>Private-Key Encryption Schemes and Block Ciphers</b>	<b>21</b>
4.1	Pseudorandom Functions and Permutations . . . . .	21
4.2	Private-Key Encryption Schemes from Pseudorandom Functions . . . . .	22
4.3	DES – The Data Encryption Standard . . . . .	24
<b>5</b>	<b>Block Ciphers (continued)</b>	<b>27</b>
5.1	Attacks on DES . . . . .	27
5.1.1	Single-Round DES . . . . .	27
5.1.2	Two-Round DES . . . . .	28
5.1.3	Three-Round DES . . . . .	28
5.1.4	Brute Force Search . . . . .	29
5.1.5	The Best Known Attacks on Full DES . . . . .	29
5.1.6	Further Reading . . . . .	29
5.2	Increasing the Key-Size for DES . . . . .	29
5.2.1	First Attempt – Alternating Keys . . . . .	29
5.2.2	Second Attempt – Double DES . . . . .	30
5.2.3	Triple DES (3DES) . . . . .	31
5.3	Modes of Operation . . . . .	31

<b>6</b>	<b>Collision-Resistant (Cryptographic) Hash Functions</b>	<b>35</b>
6.1	Definition and Properties . . . . .	35
6.2	Constructions of Collision-Resistant Hash Functions . . . . .	37
6.3	Popular Uses of Collision-Resistant Hash Functions . . . . .	38
6.4	The Random Oracle Model . . . . .	39
<b>7</b>	<b>Message Authentication</b>	<b>41</b>
7.1	Message Authentication Codes – Definitions . . . . .	42
7.2	Constructions of Secure Message Authenticate Codes . . . . .	43
7.3	Practical Constructions of Message Authentication Codes . . . . .	44
<b>8</b>	<b>Various Issues for Symmetric Encryption</b>	<b>47</b>
8.1	Combining Encryption and Message Authentication . . . . .	47
8.2	CCA-Secure Encryption . . . . .	49
8.3	Key Management . . . . .	50
<b>9</b>	<b>Public-Key (Asymmetric) Cryptography</b>	<b>53</b>
9.1	Motivation . . . . .	53
9.2	Public-Key Problems and Mathematical Background . . . . .	54
9.3	Diffie-Hellman Key Agreement [18] . . . . .	55
<b>10</b>	<b>Public-Key (Asymmetric) Encryption</b>	<b>57</b>
10.1	Definition of Security . . . . .	57
10.2	The El-Gamal Encryption Scheme [19] . . . . .	58
10.3	RSA Encryption . . . . .	59
10.4	Security of RSA . . . . .	60
10.5	Hybrid Encryption . . . . .	61
<b>11</b>	<b>Attacks on RSA</b>	<b>63</b>
11.1	Private and Public-Key Reversal . . . . .	63
11.2	Textbook RSA . . . . .	63
11.3	Common Modulus Attack . . . . .	64
11.4	Simplified Broadcast Attack . . . . .	64
11.5	Timing Attacks . . . . .	65
<b>12</b>	<b>Digital Signatures and Applications</b>	<b>67</b>
12.1	Definitions . . . . .	67
12.2	Constructions . . . . .	68
12.3	Certificates and Public-Key Infrastructure . . . . .	71
12.4	Combining Encryption and Signatures – SignCryption . . . . .	72
<b>13</b>	<b>Secure Protocols</b>	<b>73</b>
13.1	The SSL Protocol Version 3.0 . . . . .	73
13.2	Secret Sharing . . . . .	75

# Lecture 1

## Introduction and Historical Ciphers

### 1.1 Introduction

Classic cryptography dealt exclusively with the problem of secure communication. Modern cryptography is a much broader field that deals with all adversarial threats facing parties who wish to carry some task in a network. More specifically, the aim of cryptography is to provide secure solutions to a set of parties who wish to carry out a distributed task and either do not trust each other or fear an external adversarial threat.

**The heuristic versus rigorous approach to security.** The heuristic approach follows a build/break/fix cycle. That is, a cryptographic solution is proposed, its weaknesses are then discovered and the solution is fixed to remove these weaknesses. At this point, new weaknesses are found and again fixed. An inherent problem with this approach is that it is unclear when the cycle has concluded. That is, at the point at which no new weaknesses are found, it is unclear whether this is **(a)** because the solution is really secure or **(b)** because we just haven't found any weaknesses *yet*. We note that there have been a number of examples of protocols that were well-accepted as secure, until weaknesses were found years later (the Needham-Schroeder protocol is a classic example). On the other hand, it is widely believed that the problem of constructing efficient and secure block ciphers (using the heuristic approach) is well understood.<sup>1</sup> In contrast to the heuristic approach, the rigorous approach (or the approach of “provable security”) provides mathematical proofs of the security of cryptographic constructions. Typically, these proofs are by reduction to basic problems that are *assumed to be hard*. (This breakdown in the rigorous nature of the proof is inevitable until major breakthroughs are made in the theory of computer science. In particular, the existence of secure solutions for most cryptographic tasks implies, among other things, that  $\mathcal{P} \neq \mathcal{NP}$ .) An important focus of this approach is **(a)** reducing the assumptions to their minimum, and **(b)** proving the security of even very *complex constructions* by reducing them to a *simple assumption* regarding the feasibility of solving a certain problem in polynomial-time.

I am a strong proponent of the rigorous approach to cryptography and security. However, together with this, I believe that some heuristic solutions are necessary in practice (given our current state of knowledge regarding rigorous constructions). This belief is reflected in this course, where

---

\* Lecture notes for an undergraduate course in cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

<sup>1</sup>My personal position is that heuristic constructions of low-level primitives make more sense than for more complex protocols. Indeed, even theoretical cryptography relies on “heuristic” constructions of primitives (since there is no proof that factoring is hard, this belief is arguably no different to the belief that DES is a pseudorandom permutation).

practical and heuristic constructions are presented together with theory.<sup>2</sup> I stress that it is my *strong belief* that the use of heuristics should be minimalized as much as possible. Furthermore, although heuristics may sometimes be unavoidable with respect to actual constructions, it is imperative that rigorous definitions of both the adversarial model and what it means that the construction is secure are always provided.

**Defining security.** In order to define what it means for a construction to be “secure”, there are two distinct issues that must be explicitly considered. The first is the **power of the adversary** and the second the **break**. The *power of the adversary* relates to assumptions regarding its computational power and what actions it is allowed to take. For example, do we assume that the adversary can merely eavesdrop on encrypted messages, or do we assume that it can also actively request encryptions (or even decryptions). The *break* relates to what the adversary must do in order to “succeed”. For example, if encryption is being considered, then adversarial success could be defined by the ability of obtaining the plaintext of an encrypted message (this would be a very weak notion of security, but this is not the place to discuss how “good” such a definition would be). Once the adversarial power and break have been defined, a construction is said to be secure if no adversary of the specified power can succeed in breaking the construction (beyond a specified small probability).

An important issue to note in relation to the above is that defining security essentially means providing a mathematical definition. However, if the adversarial power that is defined is too weak (and in practice adversaries have more power), then “real security” is not obtained, even if a “mathematically secure” construction is used. Likewise, if the break is not carefully defined, then the mathematical definition may not provide us with the level of security that is needed in the real world. In short, a definition of security must accurately model the real world security needs in order for it to deliver on its mathematical promise of security.

**Parenthetical remark.** Turing faced a similar problem in [38]. Specifically, he was concerned with the question of whether or not the mathematical definition of “computable numbers” indeed includes all numbers that would be considered computable. A quote from [38, Section 9] follows:<sup>3</sup>

*No attempt has yet been made to show that the “computable” numbers include all numbers which would naturally be regarded as computable. All arguments which can be given are bound to be, fundamentally, appeals to intuition, and for this reason rather unsatisfactory mathematically. The real question at issue is “What are the possible processes which can be carried out in computing a number?”*

*The arguments which I shall use are of three kinds.*

- (a) *A direct appeal to intuition.*
- (b) *A proof of the equivalence of two definitions (in case the new definition has a greater intuitive appeal).*
- (c) *Giving examples of large classes of numbers which are computable.*

*Once it is granted that computable numbers are all “computable” several other propositions of the same character follow. In particular, it follows that, if there is a general*

---

<sup>2</sup>Given that this is an undergraduate course, more attention is given to constructions than to the theory underlying theoretical cryptography. This does *not* reflect my position regarding the relative importance of theory. In fact, the opposite is true. Rather, I focus on theory in my graduate course on the “foundations of cryptography”.

<sup>3</sup>We thank Hugo Krawczyk for showing us this quote.

*process for determining whether a formula of the Hilbert function calculus is provable, then the determination can be carried out by a machine.*

It seems that this issue is inevitable when one attempts to present mathematical models of real-life processes.

## 1.2 Historical Ciphers and Their Cryptanalysis

This section focuses on ciphers that were widely used in the past, and how they can be broken. Although this is the focus, we also present some central principles of cryptography which can be learned from the weaknesses of these schemes. In this section, plaintext characters are written in `lower case` and ciphertext characters are written in `UPPER CASE`.

**Caesar’s cipher and Kerckhoff’s principle** [24]. Encryption is carried out in this cipher by rotating the letters of the alphabet by 3. That is: `a` is encrypted to `D`, `b` to `E` and so on. For example:

`attack today` → `DWDF WRGDB`

The secrecy of this cipher is in the algorithm only. Once the method is discovered (or leaked), it is trivial to decipher. This weakness brings us to one of the central principles of modern cryptography, known as **Kerckhoff’s principle**:

*A cryptosystem should be secure even if everything about the system, except the **key**, is public knowledge.*<sup>4</sup>

This principle stands in contrast to the *security by obscurity* school of thought, which is unfortunately still very popular in practice. Some of the advantages of “open design” are as follows:

1. Published designs undergo public scrutiny and are therefore likely to be stronger.
2. It is better that security flaws are revealed (by “ethical hackers”) and made public, than having these flaws only known to malicious parties.
3. If the security of the system relies on the secrecy of the algorithm, then reverse engineering of code poses a serious threat to security. (This is in contrast to a key which should never be hardwired into the code, and so is not vulnerable to reverse engineering.)
4. Public design enables the establishment of standards.

We now return to the topic of historical ciphers.

**The shift cipher and the sufficient key space principle.** Caesar’s cipher suffered from the fact that encryption and decryption are always the same, and there is no secret key. In the shift cipher, the key  $K$  is a random number between 0 and 25. Encryption takes place by shifting each letter by  $K$ , and rotating when reaching the letter `z`. Caesar’s cipher is equivalent to the shift cipher when  $K = 3$ . The security of this cipher is very weak, since there are only 26 possible

---

<sup>4</sup>In fact, this is just one of six principles laid down by Kerckhoff; the others are also interesting and some of them just as central. See <http://encyclopedia.thefreedictionary.com/Kerckhoff's%20principle> for a short dictionary entry on the topic.

keys. Therefore, all possible keys can be tested, and the key that decrypt the ciphertext into a plaintext that “makes sense” is most likely the correct one. This brings us to a trivial, yet important principle: *Any secure scheme must have a key space that is not vulnerable to exhaustive search.* In today’s age, an exhaustive search may use very powerful computers, or many thousands of PC’s that are distributed around the world.

*Question:* it has been decided to use a shift cipher in order to send a message whose meaning is whether or not to attack on the following day. The plaintext y (for yes) will be used to represent the message “attack” and the plaintext n (for no) will be used to represent the message “don’t attack”. Is this use of the shift cipher secure? Why or why not?

**Mono-alphabetic substitution.** In order to increase the key space, each plaintext character is mapped to a *different* ciphertext character. The mapping is 1–1 in order to enable decryption (i.e., the mapping is a permutation). The key is therefore a permutation of the alphabet (the size of the key space is therefore  $26!$ ). For example, the key

```
a b c d e f g h i j k l m n o p q r s t ...
X E U A D N B K S M R O C Q F S Y H W G ...
```

results in the following encryption:

attack today  $\rightarrow$  XGGZR GFAXL

A brute force attack on the key no longer works for this cipher – even by computer. Nevertheless, it is easy to break the scheme by using the statistical patterns of the language. The two properties that allow the attack are **(a)** in this cipher, the mapping of each letter is fixed, and **(b)** the probability distribution of text in English (or any other language) is known. The attack works by building the probability distribution of the ciphertext (i.e., simply counting how many times each letter appears), and then comparing it to the probability distribution of English. The letter with the highest frequency in the ciphertext is likely mapped from e, and so on. Combining this with general knowledge that we have of the English language, it is easy to decipher. See the probability distribution of English letters frequencies in Figure 1.1. This simple attack works remarkably well, even on relatively short texts.

*Problems:* Decipher the following two ciphertexts:

- BPMZO XS AFZHAWFC FX OSSK: "XPH ZFOT SY XPH PSEAH MAC'X MC. APH'A FX PHK SYMBH KECCMCL PHK BSWNFCT. M'ZZ LHX XPH WFC SY XPH PSEAH."
- FXHFMK WT OLXTDFBO LG VTLVXT HNL EYKFIDTT HYON MLA. ONTM NFUT F DYINO OL ONTYD LHB DYEYPAXLAK LVYBYLBK.

**The Vigenère (poly-alphabetic substitution) cipher.** The problem with the substitution cipher above is that each plaintext character is always mapped to the same ciphertext character. In this cipher, different instances of the same letter are mapped to different characters. The method:

1. Choose a period  $t$  and then choose  $t$  random permutations  $\pi_1, \dots, \pi_t$  over the alphabet.
2. Characters in positions  $i, t + i, 2t + i, \dots$  are mapped using permutation  $\pi_i$  (for  $i = 1, \dots, t$ ).

The statistical attack described above no longer works because the different permutations smooth out the probability distribution. The key here contains  $t$  (chosen at random within some determined range) and the random permutations  $\pi_1, \dots, \pi_t$ . We describe a three step attack:

1. Determine the period  $t$
2. Break the ciphertext into  $t$  separate pieces
3. Solve each piece using the simple statistical attack above (note this is more difficult than before because we need to combine all guesses together in order to utilize our knowledge of the language)

The question here is how to determine  $t$ . We now describe **Kasiski's method** for solving this problem. First, we identify repeated patterns of length 2 or 3 in the ciphertext. These are likely to be due to certain bigrams or trigrams that appear very often in the English language. Now, the distance between these appearances should be a multiple of the period length. Thus, taking the *greatest common divisor* of all distances between the repeated sequences should be a good guess for the length of  $t$ .

Other ways of determining  $t$  are as follows:

1. *Brute force search*: for  $k = 1, 2, \dots$  build the probability distributions for the letters  $i, k + i, 2k + i$  and so on (for each  $i$ ). If  $t = k$ , then these distributions should look like English.
2. *Measure smoothness*: Denote by  $p_i$  the frequency of the  $i^{\text{th}}$  letter in the ciphertext. Then, compute  $\sum_{i=1}^{26} (p_i - \frac{1}{26})^2$ ; this value is the statistical distance of the probability distribution from the uniform distribution. Now, the larger the value  $t$ , the closer the distribution should be to the uniform distribution. (It is possible to build a table of expected statistical distances for different values of  $t$ .)

**Code-book cipher.** In this cipher, an unrelated text is used as a key. That is, a book and a page number is chosen, and encryption works by “adding” the two texts together. In order to break this, we use the fact that English language is not uniformly distributed. So, we start by subtracting popular trigrams like **the** or **ing** from each point in the text. Sometimes, we will obtain a combination that makes sense and we will be able to complete the word. Obtaining enough words may enable us to guess the text that is used.

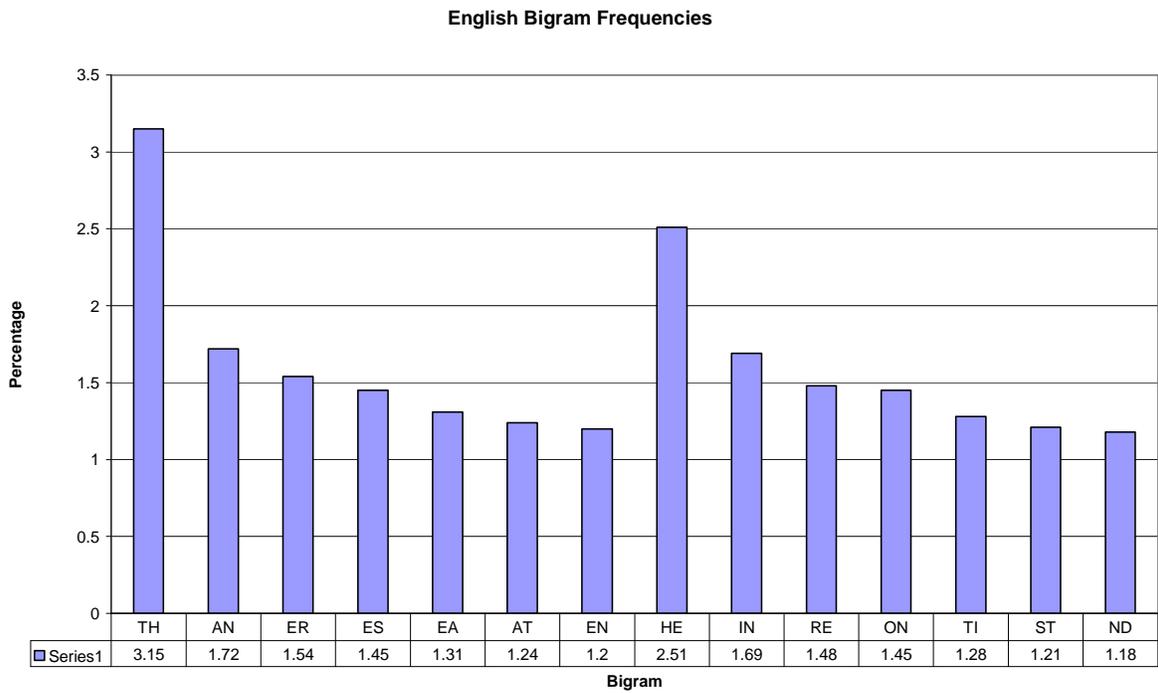
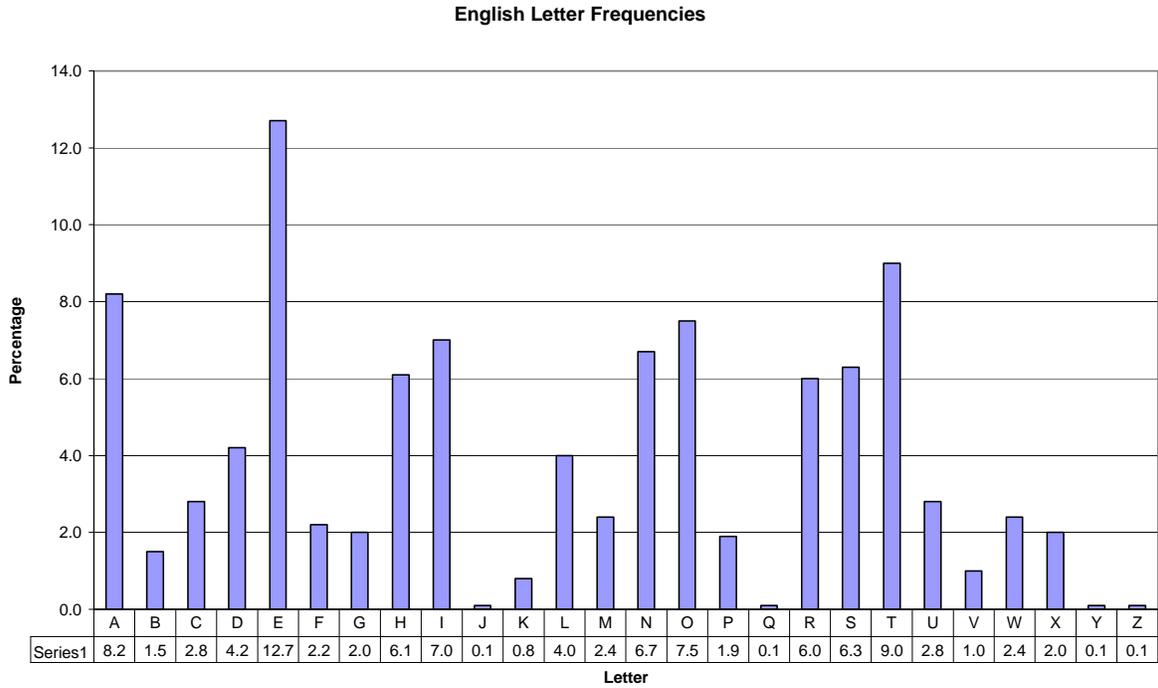


Figure 1.1: English Frequencies

## Lecture 2

# Information Theoretic Security

In general, we consider an encryption scheme to be secure if the amount of time (computing resources) that it takes to “break” it is well beyond the capability of any entity given today’s technology and the technology of the foreseeable future. In this lecture, we consider the notion of perfect or information-theoretic security of schemes that cannot be broken even if an adversary has infinite computational power and resources. Unfortunately, there are proven limitations on the constructions of perfectly secure encryption and we will prove these limitations as well.

### 2.1 Preliminaries

**Brief reminders from probability:**

1.  $X$  and  $Y$  are independent if

$$\Pr[X = x \ \& \ Y = y] = \Pr[X = x] \cdot \Pr[Y = y]$$

2. The conditional probability of  $X$  given  $Y$  is

$$\Pr[X = x \mid Y = y] = \frac{\Pr[X = x \ \& \ Y = y]}{\Pr[Y = y]}.$$

Restating the above,  $X$  and  $Y$  are independent if and only if  $\Pr[X = x \mid Y = y] = \Pr[X = x]$  for all  $x$  and  $y$ .

3. Bayes’ theorem states that if  $\Pr[Y = y] > 0$ , then

$$\Pr[X = x \mid Y = y] = \frac{\Pr[X = x] \cdot \Pr[Y = y \mid X = x]}{\Pr[Y = y]}.$$

**Perfectly-secret encryption.** Before defining the notion of perfectly-secret encryption, we present some notation:

1. Let  $\mathcal{P}$  denote the set of all possible plaintexts (the message space).
2. Let  $\mathcal{K}$  denote the set of all possible keys (the key space).

---

\* Lecture notes for an undergraduate course in cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

3. Let  $\mathcal{C}$  denote the set of all possible ciphertexts.

We stress that the set  $\mathcal{P}$  is defined independently of the encryption scheme and equals the set of messages that can be obtained with non-zero probability. In contrast, the set  $\mathcal{K}$  depends only on the encryption scheme (and not  $\mathcal{P}$ ), and the set  $\mathcal{C}$  may in principle depend both on the encryption scheme and the set  $\mathcal{P}$ . As for  $\mathcal{P}$ , the sets  $\mathcal{K}$  and  $\mathcal{C}$  are minimal in that they only include values that can be obtained with non-zero probability.

An encryption scheme  $(E, D)$  is a pair of functions such that  $E : \mathcal{K} \times \mathcal{P} \rightarrow \mathcal{C}$ ,  $D : \mathcal{K} \times \mathcal{C} \rightarrow \mathcal{P}$ , and for every  $m \in \mathcal{P}$  and every  $k \in \mathcal{K}$  it holds that  $D(k, (E(k, m))) = m$  (i.e., the decryption process always succeeds). We typically write  $E_k(\cdot)$  and  $D_k(\cdot)$ . We also note that  $E$  may be *probabilistic* (in which case it is a function from  $\mathcal{K} \times \mathcal{P} \times \mathcal{R}$  where  $\mathcal{R}$  is a uniformly chosen string of a specified length). We assume that the distributions over the sets  $\mathcal{P}$  and  $\mathcal{K}$  are independent (i.e., keys are chosen independently of the message). This is reasonable because the key is typically fixed before the message is known.

We write  $\Pr[\mathcal{K} = k]$  to mean the probability that the specific key being used is  $k$ ; likewise for  $\Pr[\mathcal{C} = c]$  and  $\Pr[\mathcal{P} = m]$ . For example, if the message space is  $\{a, b, c\}$  and the message  $a$  is sent with probability  $1/4$ , then we write  $\Pr[\mathcal{P} = a] = 1/4$ . We note that the probability space for  $\Pr[\mathcal{C} = c]$  includes the random choice of the key  $k$ , the distribution over the message space  $\mathcal{P}$  and any random coin tosses (possibly) made by  $E$  during encryption.

Denote the set of ciphertexts under a given key  $k$  by  $\mathcal{C}(k) = \{E_k(x) \mid x \in \mathcal{P}\}$ . We therefore have the following fact (which relies on the independence of  $\mathcal{P}$  and  $\mathcal{K}$ ):

$$\Pr[\mathcal{C} = c] = \sum_{k \in \mathcal{K} \text{ s.t. } c \in \mathcal{C}(k)} \Pr[\mathcal{K} = k] \cdot \Pr[\mathcal{P} = D_k(c)] \quad (2.1)$$

We can also compute the probability that  $\mathcal{C} = c$  given some plaintext  $m$  (i.e., the probability that the encryption of  $m$  will equal  $c$ ):

$$\Pr[\mathcal{C} = c \mid \mathcal{P} = m] = \sum_{k \in \mathcal{K} \text{ s.t. } m = D_k(c)} \Pr[\mathcal{K} = k] \quad (2.2)$$

## 2.2 Perfect Secrecy

We are now ready to define the notion of perfect secrecy. Intuitively, this is formulated by saying that the probability distribution of the plaintext *given* the ciphertext is that same as its a priori distribution. That is,

**Definition 2.1** (perfect secrecy): *An encryption scheme has perfect secrecy if for all plaintexts  $m \in \mathcal{P}$  and all ciphertexts  $c \in \mathcal{C}$ :*

$$\Pr[\mathcal{P} = m \mid \mathcal{C} = c] = \Pr[\mathcal{P} = m]$$

An equivalent formulation of this definition is as follows.

**Lemma 2.2** *An encryption scheme has perfect secrecy if and only if for all  $m \in \mathcal{P}$  and  $c \in \mathcal{C}$*

$$\Pr[\mathcal{C} = c \mid \mathcal{P} = m] = \Pr[\mathcal{C} = c]$$

**Proof:** This lemma follows from Bayes' theorem. That is, assume that

$$\Pr[\mathcal{C} = c \mid \mathcal{P} = m] = \Pr[\mathcal{C} = c]$$

Then, multiply both sides of the equation by the value  $\Pr[\mathcal{P} = m]/\Pr[\mathcal{C} = c]$  and you obtain

$$\frac{\Pr[\mathcal{C} = c \mid \mathcal{P} = m] \cdot \Pr[\mathcal{P} = m]}{\Pr[\mathcal{C} = c]} = \Pr[\mathcal{P} = m]$$

Then, by Bayes' theorem, it follows that the left-hand-side equals  $\Pr[\mathcal{P} = m \mid \mathcal{C} = c]$ . Thus,  $\Pr[\mathcal{P} = m \mid \mathcal{C} = c] = \Pr[\mathcal{P} = m]$  and the scheme has perfect secrecy.

In the other direction, assume that the scheme has perfect secrecy and so  $\Pr[\mathcal{P} = m \mid \mathcal{C} = c] = \Pr[\mathcal{P} = m]$ . Then, multiplying both sides by  $\Pr[\mathcal{C} = c]/\Pr[\mathcal{P} = m]$  we obtain

$$\frac{\Pr[\mathcal{P} = m \mid \mathcal{C} = c] \cdot \Pr[\mathcal{C} = c]}{\Pr[\mathcal{P} = m]} = \Pr[\mathcal{C} = c]$$

By Bayes' theorem, the left-hand side equals  $\Pr[\mathcal{C} = c \mid \mathcal{P} = m]$ , completing the lemma. ■

Note that in the above proof, we use the fact that  $\Pr[\mathcal{C} = c] > 0$  and the fact that  $\Pr[\mathcal{P} = m] > 0$ . Given the above, we obtain a useful property of perfect secrecy:

**Lemma 2.3** *For any perfectly-secret encryption scheme, it holds that for all  $m, m' \in \mathcal{P}$  and all  $c \in \mathcal{C}$ :*

$$\Pr[\mathcal{C} = c \mid \mathcal{P} = m] = \Pr[\mathcal{C} = c \mid \mathcal{P} = m']$$

**Proof:** Applying Lemma 2.2 we obtain that  $\Pr[\mathcal{C} = c \mid \mathcal{P} = m] = \Pr[\mathcal{C} = c]$  and likewise  $\Pr[\mathcal{C} = c \mid \mathcal{P} = m'] = \Pr[\mathcal{C} = c]$ . Thus, they both equal  $\Pr[\mathcal{C} = c]$  and so are equal to each other. ■

## 2.3 The One-Time Pad (Vernam's Cipher)

In 1917, Vernam patented a cipher that obtains perfect secrecy. Let  $a \oplus b$  denote the *bitwise exclusive-or* of  $a$  and  $b$  (i.e., if  $a = a_1 \cdots a_n$  and  $b = b_1 \cdots b_n$ , then  $a \oplus b = a_1 \oplus b_1 \cdots a_n \oplus b_n$ ). In Vernam's cipher, a key  $\mathcal{K}$  is only allowed to be used once (thus, it is called a one-time pad). In this cipher, we set  $\mathcal{K}$  to be the uniform distribution over  $\{0, 1\}^n$ , and we set  $\mathcal{P} = \{0, 1\}^n$  and  $\mathcal{C} = \{0, 1\}^n$ . Then,

$$c = E_k(m) = k \oplus m \quad \text{and} \quad m = D_k(c) = k \oplus c$$

Intuitively, the one-time pad is perfectly secret because for every  $c$  and every  $m$ , there exists a key for which  $c = E_k(m)$ . We prove this intuition formally in the next theorem.

**Proposition 2.4** *The one-time pad is perfectly secret.*

**Proof:** We prove this using Eq. (2.2) and Lemma 2.3. Let  $m, m' \in \{0, 1\}^n$  be any two plaintexts and let  $c \in \{0, 1\}^n$  be a ciphertext. Then, by Eq. (2.2) it follows that

$$\Pr[\mathcal{C} = c \mid \mathcal{P} = m] = \sum_{k \in \mathcal{K} \text{ s.t. } m=k \oplus c} \Pr[\mathcal{K} = k] = \frac{1}{2^n}$$

where the second equality follows from the fact that there exists exactly one key  $k$  for which  $m = k \oplus c$ , and the keys are chosen at uniform from the set  $\{0, 1\}^n$ . The above calculation is independent of the distribution  $\mathcal{P}$ , and therefore it also holds that

$$\Pr[\mathcal{C} = c \mid \mathcal{P} = m'] = \sum_{k \in \mathcal{K} \text{ s.t. } m' = k \oplus c} \Pr[\mathcal{K} = k] = \frac{1}{2^n}$$

Thus,  $\Pr[\mathcal{C} = c \mid \mathcal{P} = m] = \Pr[\mathcal{C} = c \mid \mathcal{P} = m']$ , as required. ■

We therefore have that perfect secrecy is indeed attainable. However, the one-time pad is severely limited in that each key can only be used once.

## 2.4 Limitations of Perfect Secrecy

Unfortunately, the above-described limitation of the one-time pad is inherent. That is, we now prove that in order to obtain perfect secrecy, the key-space must be at least as large as the message-space. Before stating the theorem, we recall that we assume that the message-space  $\mathcal{P}$  is such that for every  $m \in \mathcal{P}$ ,  $\Pr[\mathcal{P} = m] > 0$ . Otherwise, one should redefine  $\mathcal{P}$  without  $m$  and apply the theorem to this new  $\mathcal{P}$ . We now state the theorem.

**Theorem 2.5** *Assume that there exists a perfectly secret encryption scheme for sets  $\mathcal{K}$ ,  $\mathcal{P}$  and  $\mathcal{C}$ . Then,  $|\mathcal{K}| \geq |\mathcal{P}|$ .*

**Proof:** Let  $c$  be a ciphertext and consider the set  $D(c) = \{m \mid \exists k \text{ s.t. } m = D_k(c)\}$ . Then, by perfect secrecy it must hold that  $D(c) = \mathcal{P}$ ; otherwise, we obtain that for some message  $m$  it holds that  $\Pr[\mathcal{P} = m \mid \mathcal{C} = c] = 0$ , whereas  $\Pr[\mathcal{P} = m] > 0$ . Since  $D(c) = \mathcal{P}$  it follows that for every  $m$ , there exists at least one key  $k$  such that  $E_k(m) = c$  (otherwise,  $m \notin D(c)$ ). Now, let  $m$  and  $m'$  be two different plaintexts, let  $k$  be such that  $E_k(m) = c$ , and let  $k'$  be such that  $E_{k'}(m') = c$  (we have just proven the existence of  $k$  and  $k'$ ). Then, it must hold that  $k \neq k'$ ; otherwise correct decryption will not hold. We conclude that for every plaintext  $m$  there is at least one key  $k$  that is unique to  $m$  such that  $E_k(m) = c$ . Thus, the number of keys is at least as great as the number of messages  $m$ . That is,  $|\mathcal{K}| \geq |\mathcal{P}|$ . ■

We remark that the above proof holds also for *probabilistic* encryption schemes.

**One-time-pad security at a lower price.** Very often, encryption schemes are developed by companies who then claim to have achieved the security level of a one-time pad, at a “lower price” (e.g., with small keys, with public-keys etc.). The above proof demonstrates that such claims *cannot* be true; the person claiming them either knows little of cryptography or is blatantly lying.

## 2.5 Shannon’s Theorem

Shannon [34] provided a characterization of perfectly secure encryption schemes. As above, we assume that the sets  $\mathcal{P}$ ,  $\mathcal{C}$  and  $\mathcal{K}$  are such that all elements are obtained with non-zero probability (and  $\mathcal{K}$  is independent of  $\mathcal{P}$ ).

**Theorem 2.6** (Shannon’s theorem): *Let  $(E, D)$  be an encryption scheme over  $(\mathcal{P}, \mathcal{C}, \mathcal{K})$  where  $|\mathcal{P}| = |\mathcal{K}| = |\mathcal{C}|$ . Then, the encryption scheme provides perfect secrecy if and only if:*

1. Every key is chosen with equal probability  $1/|\mathcal{K}|$ ,

2. For every  $m \in \mathcal{P}$  and every  $c \in \mathcal{C}$ , there exists a unique key  $k \in \mathcal{K}$  such that  $E_k(m) = c$ .

**Proof:** We first prove that if an encryption scheme is perfectly secret over  $(\mathcal{P}, \mathcal{K}, \mathcal{C})$  such that  $|\mathcal{P}| = |\mathcal{K}| = |\mathcal{C}|$ , then items (1) and (2) hold. We have already seen that for every  $m$  and  $c$ , there exists at least one key  $k \in \mathcal{K}$  such that  $E_k(m) = c$ . For a fixed  $m$ , consider now the set  $\{E_k(m) \mid k \in \mathcal{K}\}$ . By the above,  $|\{E_k(m) \mid k \in \mathcal{K}\}| \geq |\mathcal{C}|$  (because by fixing  $m$  first we obtain that for every  $c$  there exists a  $k$  such that  $E_k(m) = c$ ). In addition, by simple counting,  $|\{E_k(m) \mid k \in \mathcal{K}\}| \leq |\mathcal{K}|$ . Now, since  $|\mathcal{K}| = |\mathcal{C}|$  it follows that  $|\{E_k(m) \mid k \in \mathcal{K}\}| = |\mathcal{K}|$ . This implies that for every  $m$  and  $c$ , there do not exist any two keys  $k_1$  and  $k_2$  such that  $E_{k_1}(m) = E_{k_2}(m) = c$ . That is, for every  $m$  and  $c$ , there exists at most one key  $k \in \mathcal{K}$  such that  $E_k(m) = c$ . Combining the above, we obtain item (2).

We proceed to show that for every  $k \in \mathcal{K}$ ,  $\Pr[\mathcal{K} = k] = 1/|\mathcal{K}|$ . Let  $n = |\mathcal{K}|$  and  $\mathcal{P} = \{m_1, \dots, m_n\}$  (recall,  $|\mathcal{P}| = |\mathcal{K}| = n$ ), and fix a ciphertext  $c$ . Then, we can label the keys  $k_1, \dots, k_n$  such that for every  $i$  ( $1 \leq i \leq n$ ) it holds that  $E_{k_i}(m_i) = c$ . (This labelling can be carried out because by fixing  $c$ , we obtain that for every  $m$  there exists a single  $k$  such that  $E_k(m) = c$ ; for  $m_i$  we label the key by  $k_i$ . We note that this labelling covers all of the keys in  $\mathcal{K}$ . This holds because for  $i \neq j$ , the key  $k_i$  such that  $E_{k_i}(m_i) = c$  is different to the key  $k_j$  such that  $E_{k_j}(m_j) = c$ ; otherwise, unique decryption will not hold.) Then, by perfect secrecy we have:

$$\begin{aligned} \Pr[\mathcal{P} = m_i] &= \Pr[\mathcal{P} = m_i \mid \mathcal{C} = c] \\ &= \frac{\Pr[\mathcal{C} = c \mid \mathcal{P} = m_i] \cdot \Pr[\mathcal{P} = m_i]}{\Pr[\mathcal{C} = c]} \\ &= \frac{\Pr[\mathcal{K} = k_i] \cdot \Pr[\mathcal{P} = m_i]}{\Pr[\mathcal{C} = c]} \end{aligned}$$

where the second equality is by Bayes' theorem and the third equality holds by the labelling above. From the above, it follows that for every  $i$ ,

$$\Pr[\mathcal{K} = k_i] = \Pr[\mathcal{C} = c]$$

Therefore, for every  $i$  and  $j$ ,  $\Pr[\mathcal{K} = k_i] = \Pr[\mathcal{K} = k_j]$  and so all keys are chosen with the same probability. We conclude that  $\Pr[\mathcal{K} = k_i] = 1/|\mathcal{K}|$ , as required.

The other direction of the proof is left for an exercise. ■



## Lecture 3

# Pseudorandomness and Private-Key Encryption Schemes

In this lecture we introduce the notion of computational indistinguishability and pseudorandomness (generators and functions). In addition, we define secure symmetric (private-key) encryption, and present stream ciphers, block ciphers and modes of operation. In general, we use a “concrete-security analysis” approach (rather than an asymptotic one); however, we use a simplified and rather informal presentation. Our aim in this approach is to simplify the technical aspects of the definitions, while also raising awareness of the need in practice to have a concrete analysis in order that specific parameters may be chosen. Once again, a rigorous treatment of this material will be given in Semester II (course 89-856).

### 3.1 Pseudorandomness

In this section, we present the notion of indistinguishability and pseudorandomness. Initially, it may not be clear why these seemingly very abstract concepts are of relevance. However, we will soon show how to use them to construct secure encryption schemes that do not suffer from the inherent limitations on key-size that hold for perfectly-secret encryption schemes.

**Conventions.** Let  $X$  be a probability distribution and  $A$  an algorithm. Then, when we write  $A(X)$  we mean the output of  $A$  when given a random output drawn according to the distribution  $X$ . If  $X$  appears twice or more in the same equation, then each appearance should be interpreted as a single instance (i.e., all values are the same). We denote by  $U_n$  the uniform distribution over  $\{0, 1\}^n$ .

**Computational indistinguishability and pseudorandomness.** Loosely speaking, two ensembles are  $(t, \epsilon)$ -indistinguishable if for every algorithm (Turing machine) that runs for at most  $t$  steps, the probability of distinguishing between them (via a single sample) is at most  $\epsilon$ .

**Definition 3.1** ( $(t, \epsilon)$ -indistinguishability): *Let  $\epsilon \in [0, 1]$  be a real value, let  $t \in \mathbf{N}$ , and let  $X$  and  $Y$  be probability distributions. We say that the  $X$  and  $Y$  are  $(t, \epsilon)$ -indistinguishable, denoted  $X \stackrel{t, \epsilon}{\equiv} Y$ ,*

---

\* Lecture notes for an undergraduate course in cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

if for every distinguisher  $D$  running for at most  $t$  steps,

$$|\Pr[D(X) = 1] - \Pr[D(Y) = 1]| < \epsilon$$

A probability distribution is pseudorandom if it is indistinguishable from the uniform distribution.

**Definition 3.2** ( $(t, \epsilon)$ -pseudorandomness): Let  $X_n$  be a probability distribution ranging over strings of length  $n$ . We say that  $X_n$  is  $(t, \epsilon)$ -pseudorandom if it is  $(t, \epsilon)$ -indistinguishable from  $U_n$ .

**Pseudorandom generators.** A pseudorandom generator (PRG) takes short strings and “stretches” them so that the result is pseudorandom. Intuitively, such an algorithm enables us to use less randomness. Thus, for example, we could obtain the effect of a one-time pad while using a key that is shorter than the message.

**Definition 3.3** ( $(t, \epsilon)$ -pseudorandom generator): Let  $G_{n,\ell} : \{0, 1\}^n \rightarrow \{0, 1\}^\ell$  be a polynomial-time algorithm. We say that  $G_{n,\ell}$  is a  $(t, \epsilon)$ -pseudorandom generator if  $\ell > n$  and  $G_{n,\ell}(U_n) \stackrel{t,\epsilon}{\equiv} U_\ell$ .

A pseudorandom generator is of interest (in the context of cryptography) if it can be efficiently computed (in time significantly less than  $t$ ). The input  $r$  to a generator is called the *seed*.

## 3.2 Symmetric (Private-Key) Encryption

### 3.2.1 Defining Security

As we discussed in the first lecture, defining secure encryption involves specifying two components: the *adversary* and the *break*. Intuitively, an encryption scheme is secure if a ciphertext reveals no information about the plaintext. We present a formalization based upon *indistinguishability*. That is, we say that the encryption scheme is secure if for every two plaintexts  $x$  and  $y$ , it is hard to distinguish the case that  $c = E(x)$  and  $c = E(y)$ .<sup>1</sup> The ciphertext  $c$  is called the *challenge ciphertext*. Regarding the adversary’s power, we consider three cases:

1. *Eavesdropping attack*: In this attack, the adversary is just given the ciphertext and must attempt to distinguish.
2. *Chosen-plaintext attack (CPA)*: In this attack, the adversary is allowed to obtain encryptions of any messages that it wishes. This models the fact that sometimes it is possible to influence the messages that are encrypted. By giving the adversary full power, we model this in the most general way.
3. *Chosen-ciphertext attack (CCA)*: In this attack, the adversary is even allowed to obtain decryptions of any ciphertext that it wishes (except for the challenge ciphertext). This is a very strong attack. However, it models information that may be obtained about the plaintext of ciphertexts. Such information may practically be obtained, for example, by querying a server with ciphertexts that should be sent as part of the protocol. The response of the server may reveal information about the encrypted message. (Most simply, the server may just reply with whether or not the ciphertext was valid. This information was actually used in real attacks.)

---

<sup>1</sup>We note that this is equivalent to a definition that states that the ciphertext reveals no partial information about the plaintext. Namely, this definition states that for every function  $g$  (representing information to be learned about the plaintext), the probability of computing  $g(x)$  from  $E(x)$  is comparable (i.e., within  $\epsilon$ ) of the probability of computing  $g(x)$  even without receiving  $E(x)$ .

**The syntax.** An encryption scheme is a tuple of algorithms  $(K, E, D)$  such that  $K$  is a probabilistic algorithm that generates keys,  $E$  is the encryption algorithm and  $D$  is the decryption algorithm. For now, we will assume that  $K$  just uniformly chooses a key from some set, and so we will sometimes refer to  $K$  as a set (rather than as an algorithm). We require that for every  $k \in K$  and every  $x$  it holds that  $D_k(E_k(x)) = x$ .

**The eavesdropping indistinguishability game  $\text{Expt}_{\text{EAV}}$ .** The adversary  $\mathcal{A}$  outputs a pair of messages  $m_0, m_1$ . Next, a random key is chosen  $k \leftarrow K$  and a random bit  $b \in_R \{0, 1\}$ . The adversary  $\mathcal{A}$  is then given  $c = E_k(m_b)$  and finally outputs a bit  $b'$ . We say that  $\mathcal{A}$  succeeded if  $b' = b$  and in such a case we write  $\text{Expt}_{\text{EAV}}(\mathcal{A}) = 1$ .

**Definition 3.4** *An encryption scheme  $E$  is  $(t, \epsilon)$ -indistinguishable for an eavesdropper if for every adversary  $\mathcal{A}$  that runs for at most  $t$  steps,*

$$\Pr[\text{Expt}_{\text{EAV}}(\mathcal{A}) = 1] < \frac{1}{2} + \epsilon$$

Notice that this means that the only advantage  $\mathcal{A}$  has over a random guess is  $\epsilon$ . If  $\epsilon$  is sufficiently small, this is “ideal”. We stress that typically  $t$  will be taken so that it takes the most powerful computer (or network of computers) today many hundreds of years to reach this number of steps. On the flip side,  $\epsilon$  is taken to be a number so small that it is more likely that all computers in the world crash than this actually yielding an advantage. For example,  $\epsilon$  may be taken to be  $2^{-80}$ .

Note also that Definition 3.4 is a natural relaxation of perfect secrecy. Namely, consider the setting of perfect secrecy and define  $\mathcal{P} = \{m_0, m_1\}$  with  $\Pr[\mathcal{P} = m_0] = \Pr[\mathcal{P} = m_1] = \frac{1}{2}$ . Then, perfect secrecy dictates that for every  $c$ ,

$$\Pr[\mathcal{P} = m_0 \mid \mathcal{C} = c] = \Pr[\mathcal{P} = m_0] = \frac{1}{2} = \Pr[\mathcal{P} = m_1] = \Pr[\mathcal{P} = m_1 \mid \mathcal{C} = c]$$

Thus, given a ciphertext  $c$ , the probability that  $\mathcal{P} = m_0$  equals the probability that  $\mathcal{P} = m_1$ . Stated differently, given  $c$ , the probability that any adversary will correctly guess  $b$  in the above game equals  $1/2$  exactly. Now, Definition 3.4 says exactly the same thing, except that:

1. The adversary is limited to running  $t$  steps, and
2. The adversary is “allowed” a small advantage of  $\epsilon$ .

Thus, there are really two relaxations in this definition (and they are both essential).

### 3.2.2 Constructing Secure Encryption Schemes

We show how to construct an encryption scheme that is indistinguishable for an eavesdropping adversary.

**A stream cipher.** We first show how to use a pseudorandom generator to obtain a secure encryption scheme, as long as only a single message is encrypted (i.e., as specified in the eavesdropping game above). Let  $G_{n,\ell}$  be a pseudorandom generator. The construction is as follows:

- *Key generation:* choose a random  $k \in_R \{0, 1\}^n$

- Encryption of  $m \in \{0, 1\}^\ell$ :  $c = E_k(m) = G_{n,\ell}(k) \oplus m$
- Decryption of  $c \in \{0, 1\}^\ell$ :  $m = D_k(c) = G_{n,\ell}(k) \oplus c$

In this context,  $G_{n,\ell}$  is called a **stream cipher**. We now claim that this scheme is secure for an eavesdropping adversary. The intuition for proving this claim is due to the fact that a pseudorandom generator outputs a string that is “essentially” the same as a uniformly distributed string. Therefore, the security should follow from the fact that when a uniformly distributed string is used, perfect security is obtained. The actual proof is by reduction.

**Theorem 3.5** *If  $G_{n,\ell}$  is a  $(t, \epsilon)$ -pseudorandom generator, then the encryption scheme  $(K, E, D)$  is  $(t', \epsilon)$ -indistinguishable for an eavesdropper, where  $t' = t - O(\ell)$ .*

**Proof:** We prove the theorem by reduction. That is, we show that if an adversary  $\mathcal{A}$  that runs for  $t'$  steps succeeds in the eavesdropping game with probability greater than or equal to  $\frac{1}{2} + \epsilon$ , then we can construct a distinguisher  $D$  that runs for  $t$  steps and distinguishes  $G_{n,\ell}(U_n)$  from  $U_\ell$  with advantage at least  $\epsilon$ .

The machine  $D$  receives a string  $R$  (that is either chosen according to  $G_{n,\ell}(U_n)$  or  $U_\ell$ ).  $D$  then invokes  $\mathcal{A}$ , obtains  $m_0, m_1$ , chooses  $b \in_R \{0, 1\}$  and hands  $\mathcal{A}$  the value  $c = R \oplus m_b$ . The distinguisher  $D$  then outputs 1 if and only if  $\mathcal{A}$  outputs  $b$ .

We begin by computing  $D$ 's running-time. Notice that  $D$  just invokes  $\mathcal{A}$ , reads  $m_0, m_1$  and computes  $m_b \oplus R$ . Therefore,  $D$ 's running-time equals  $\mathcal{A}$ 's running-time plus  $O(\ell)$  additional steps. We now proceed to analyze  $D$ 's probability of successfully distinguishing a random string from a pseudorandom one. First note that if  $R$  is uniformly chosen, then the probability that  $\mathcal{A}$  outputs  $b$  equals  $1/2$  exactly. On the other hand, if  $R$  is chosen according to  $G_{n,\ell}$ , then  $D$  outputs 1 with exactly the probability that  $\mathcal{A}$  succeeds. That is,

$$\Pr[D(G_{n,\ell}) = 1] - \Pr[D(U_\ell) = 1] = \Pr[\text{Expt}_{\text{EAV}}(\mathcal{A}) = 1] - \frac{1}{2} \geq \epsilon$$

We conclude that if  $\mathcal{A}$  runs in time  $t' = t - O(\ell)$  and succeeds with probability at least  $1/2 + \epsilon$ , then  $D$  runs in time  $t$  and distinguishes with probability at least  $\epsilon$ . Thus, if the generator is  $(t, \epsilon)$ -pseudorandom, it follows that  $(K, E, D)$  is  $(t - O(\ell), \epsilon)$ -indistinguishable for an eavesdropper. ■

Notice that we have now shown that the lower bounds on information-theoretic security can be “broken”, assuming the existence of pseudorandom generators.

**Stream ciphers in practice.** We note that in practice, stream ciphers are extremely fast and are therefore widely used. However, they are also less well-understood than block ciphers (see the next lecture), and thus are more often broken. We will not present practical constructions of stream ciphers in this course.

## Lecture 4

# Private-Key Encryption Schemes and Block Ciphers

The main problem with stream ciphers is that each key can only be used once (or else, some synchronization mechanism is needed to know which point in the stream is being currently used). Block ciphers are a different type of construction and are very widely used. Essentially, a block cipher is a pseudorandom permutation. We will show how to encrypt with a block cipher, and will concentrate here on both theory and practice. In this lecture, we will first show how to construct CPA-secure schemes from pseudorandom functions (or permutations). We will then proceed to show how block ciphers are constructed and used in practice.

### 4.1 Pseudorandom Functions and Permutations

A pseudorandom function is a function that cannot be distinguished from a truly random function. Note that the length of the description of an arbitrary function from  $\{0, 1\}^n$  to  $\{0, 1\}^\ell$  is  $2^{n\ell}$  bits. Thus, the set of all functions of these parameters is of size  $2^{2^{n\ell}}$ ; a random function is a function that is chosen according to the uniform distribution from this set. Equivalently, a random function can be chosen by choosing the image of every value uniformly at random (and independently of all other values).

In the following definition, we use the notion of an oracle machine (or algorithm). Such an algorithm is given access to an oracle  $\mathcal{O}$  that it can then query with any value  $q$ , upon which it receives back the result  $\mathcal{O}(q)$ . Furthermore, this computation is counted as a single step for the algorithm.

**Definition 4.1** ( $(t, \epsilon)$ -pseudorandom function): *Let  $F_{n,\ell}$  be a distribution over functions that map  $n$ -bit long strings to  $\ell$ -bit long strings. Let  $H_{n,\ell}$  be the uniform distribution over all functions mapping  $n$ -bit long strings to  $\ell$ -bit long strings. Then, the distribution  $F_{n,\ell}$  is called  $(t, \epsilon)$ -pseudorandom if for every probabilistic oracle machine  $D$  running for at most  $t$  steps,*

$$\left| \Pr[D^{F_{n,\ell}} = 1] - \Pr[D^{H_{n,\ell}} = 1] \right| < \epsilon$$

where the probabilities are taken over the coin-tosses of  $D$ , as well as the choice of the functions.<sup>1</sup>

\* Lecture notes for an undergraduate course in cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

<sup>1</sup>We stress that the order of steps in this “game” is as follows. First, a specific function is chosen uniformly from the distribution (either  $F_{n,\ell}$  or  $H_{n,\ell}$ ). Then,  $D$  is given oracle access to the chosen (and now fixed) function. Finally,  $D$  outputs a bit, in which it attempts to guess whether the function was chosen from  $F_{n,\ell}$  or  $H_{n,\ell}$ .

As with pseudorandom generators, pseudorandom functions are of interest if they can be efficiently computed. We will therefore consider distributions of functions for which a function is specified via a “short” random key of length  $k$ . That is, the functions we consider are defined by  $F_{n,\ell} : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^\ell$  where the key is chosen randomly from  $\{0, 1\}^k$ . (Notice that for  $n = k$ , for example, the description of such a pseudorandom function is only  $k$  bits, whereas the description of a truly random function is of length  $2^n \ell = 2^k \ell$ .)

**Convention.** From now on, we will consider pseudorandom functions for which  $n = \ell = k$ . That is,  $F$  will be a family of  $2^n$  functions; each function is specified by choosing a random  $n$ -bit key  $k \in_R \{0, 1\}^n$ . Then, we denote by  $F_k(\cdot)$  the deterministic function that maps  $n$ -bit strings to  $n$ -bit strings, where the key  $k$  is fixed.

## 4.2 Private-Key Encryption Schemes from Pseudorandom Functions

We begin by defining the CPA (chosen-plaintext attack) indistinguishability games.

**The CPA indistinguishability game  $\text{Expt}_{\text{CPA}}$ .** A random key is chosen  $k \leftarrow K$  and the adversary  $\mathcal{A}$  is then given oracle access to the encryption algorithm  $E_k(\cdot)$ . At some stage, the algorithm  $\mathcal{A}$  outputs a pair of message  $m_0, m_1$  of the same length. A bit  $b \in_R \{0, 1\}$  is then randomly chosen and  $\mathcal{A}$  is given  $c = E_k(m_b)$  (this value is called the challenge ciphertext). Adversary  $\mathcal{A}$  continues to have oracle access to  $E_k(\cdot)$ . Finally,  $\mathcal{A}$  outputs a bit  $b'$ . We say that  $\mathcal{A}$  succeeded if  $b' = b$  and in such a case we write  $\text{Expt}_{\text{CPA}}(\mathcal{A}) = 1$ .

**Definition 4.2** An encryption scheme  $E$  is  $(t, \epsilon)$ -indistinguishable under chosen-plaintext attacks (CPA secure) if for every adversary  $\mathcal{A}$  that runs for at most  $t$  steps,

$$\Pr[\text{Expt}_{\text{CPA}}(\mathcal{A}) = 1] < \frac{1}{2} + \epsilon$$

We note that according to this definition, an encryption scheme *must* be probabilistic. Question: why is it important that a message encrypted twice does not result in the same ciphertext (e.g., bombing locations by Japanese in WW2)?

**Constructing CPA-secure encryption schemes.** The idea behind our construction of a CPA-secure scheme is to use the pseudorandom function to hide the message. That is, let  $F$  be a pseudorandom function mapping  $n$ -bit strings to  $n$ -bit strings (recall that the key  $k$  is also of length  $n$ ). The construction is as follows:

- *Key generation:*  $k \in_R \{0, 1\}^n$
- *Encryption of  $m \in \{0, 1\}^n$ :* choose  $r \in_R \{0, 1\}^n$ ; compute  $F_k(r) \oplus m$ ; output  $c = (r, F_k(r) \oplus m)$ .
- *Decryption of  $c = (r, f)$ :* compute  $m = F_k(r) \oplus f$

Intuitively, security holds because an adversary cannot compute  $F_k(r)$  and so cannot learn anything about  $f$ . More exactly, like for the stream cipher, if we replace  $F_k(r)$  with a truly random function, then the scheme obtained would be perfectly secret, unless the same  $r$  is used twice. Barring this “bad event”, security will once again follow from the perfect secrecy of the one-time pad.

**Theorem 4.3** *If  $F_k$  is a  $(t, \epsilon)$ -pseudorandom function, then the encryption scheme  $(K, E, D)$  is  $(t', \epsilon')$ -indistinguishable under chosen-plaintext attack, where  $t' = O(t/n)$  and  $\epsilon' = \epsilon + t'/2^n$ .*

**Proof Sketch:** We present a proof sketch only. Our proof follows a general paradigm for working with pseudorandom functions. First, we analyze the security of the scheme in an idealized world where a truly random function is used. Next, we replace the random function with a pseudorandom one, and analyze the difference.

We begin by analyzing the security in the case that  $F$  is a truly random function. In this case, the only way that the adversary can learn anything about  $b$  is if the challenge ciphertext equals  $(r, F_k(r) \oplus m_b)$  and  $r$  has been seen before. Since  $r$  is chosen uniformly from  $\{0, 1\}^n$ , the probability that this  $r$  will appear in at least one other call to the encryption oracle is at most  $t'/2^n$ . (This follows from the fact that the random string  $r$  is chosen uniformly in each oracle call to  $E_k(\cdot)$  and the adversary has no control over it. Then, the probability that one of these uniformly chosen strings equals the  $r$  used in generating the challenge equals  $1/2^n$  for each call.<sup>2</sup> Since there at most  $t'$  oracle calls, the probability follows.) We conclude that the encryption adversary's probability of success in this case equals  $1/2 + t'/2^n$  where  $t'$  is the adversary's running-time.

Next, we replace the random function with a pseudorandom function. This can only change the success of the adversary by at most  $\epsilon$  (otherwise, the pseudorandom function can be distinguished from random with probability greater than  $\epsilon$ ). Therefore, the probability of outputting  $b' = b$  here is at most  $1/2 + \epsilon + t'/2^n$ . The above holds as long as the distinguisher constructed from the adversary (that we use to contradict the security of the pseudorandom function) runs for at most  $t$  steps. It therefore remains to compute the overhead for simulating the CPA-game for the adversary  $\mathcal{A}$ . Essentially, this simulation involves calling the function oracle and computing XORs. So, if the encryption adversary runs in time  $t'$ , the distinguisher will run in time at most  $t = O(t'n)$ . We conclude that if  $F$  is a  $(t, \epsilon)$ -PRF then,  $(K, E, D)$  is  $(t', \epsilon')$ -pseudorandom, where  $t' = O(t/n)$  and  $\epsilon' = \epsilon + t'/2^n$ . ■

**Suggested exercise:** *Write a more formal proof of the above theorem.*

### Remarks:

1. *Extending the analysis:* Our above analysis includes parameters  $t$  and  $\epsilon$  only. Typically, the number of queries to the encryption or pseudorandom function oracle is also included (we just used  $t'$  as an upper-bound on this value). The reason for this is that it may be realistic to run for  $2^{40}$  steps, but it would be much harder to obtain so many encryptions. This is especially the case, for example, if encryption takes place on a (relatively slow) smart card.
2. *Efficiency of the scheme:* Notice that the encryption scheme above has the drawback that the bandwidth is doubled. This can be removed if **(a)** a counter is used (this requires storing and remembering state) or **(b)** it is guaranteed that the same message is never encrypted twice. However, the practice of encrypting by  $E_k(x) = F_k(x)$  is in general *not* secure.

---

<sup>2</sup>In order to clarify this further, one can think of changing the CPA experiment in the following way. First choose  $r$  to be used in generating the challenge  $E_k(m_b)$ . Then, start the experiment and answer the oracle calls as before. When the challenge is to be generated, use  $r$  and continue as before. This is exactly the same because all strings  $r$  are uniformly and independently distributed. However, in this context it is clear that the probability of obtaining  $r$  again is  $t'/2^n$ .

**Block ciphers.** From here on, we will call a pseudorandom permutation that is used for encryption a block cipher. We remark that by “permutation” here, we mean that the function  $F_k : \{0, 1\}^n \rightarrow \{0, 1\}^n$  is 1–1. Furthermore, given  $k$  it is possible to efficiently invert  $F_k(x)$  and obtain  $x$ . The block size of such a cipher is  $n$  (i.e., this is the number of bits in its domain and range).

We note that the “attack model” is extended for pseudorandom permutations by providing the distinguisher with both access to  $F_k(\cdot)$  and  $F_k^{-1}(x)$ .

### 4.3 DES – The Data Encryption Standard

DES was developed in the 1970s at IBM (with “help” from the NSA), and adopted in 1976 as the US standard, and later as an international standard. The standard was recently replaced by the Advanced Encryption Standard (AES). Nevertheless, DES is still used almost everywhere. We will now study the DES construction.

**Feistel structure.** In order to construct a block cipher that is invertible, we have one of two options:

1. *Use invertible operations throughout:* this has the drawback of limiting our freedom.
2. *Use operations that are not all invertible:* however, the overall structure must be invertible.

A Feistel structure is a way of using non-invertible operations in the construction of an invertible function. See Figure 4.1; the function  $f$  in the diagram is not invertible and is where the key is used.

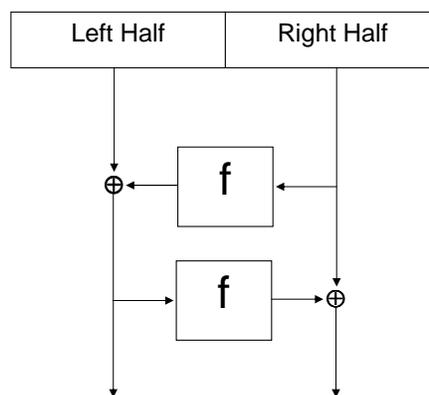


Figure 4.1: The Feistel Structure

Notice that the overall structure can be inverted by working from the bottom-up, and always computing  $f$  and not  $f^{-1}$ . Each invocation of  $f$  is called a round of the structure. Notice that a single round of a Feistel structure only hides half of the input. We therefore repeat this at least twice. Actually, we repeat it many times until sufficient security is obtained.

**The DES structure.** The DES block cipher is a Feistel structure. It has a 64 bit input/output (i.e., the block size is 64 bits) and it uses a key of size 56 bits (it is actually 64 bits, but 8 bits are used for redundancy for the purpose of error detection). The non-invertible function in the Feistel

structure of DES is  $f : \{0, 1\}^{32} \rightarrow \{0, 1\}^{32}$  and uses a 48 bit subkey. There are exactly 16 rounds; in round  $i$ , a subkey  $k_i$  is used which is a subset of 48 out of the 56 bits of the full key  $K$ . The way  $k_i$  is chosen in each round is called the key schedule.

We note that before and after the Feistel structure, DES actually applies a *fixed and known* permutation to the input and output. This slows down software implementations (the computation of this permutation takes about 1/3 of the running time, in contrast to hardware where it takes no time). We will ignore the permutation as it plays no security role, beyond slowing down attackers who use software.

**The DES  $f$  function.** The  $f$  function is constructed as follows. Let  $E$  be an expansion function,  $E : \{0, 1\}^{32} \rightarrow \{0, 1\}^{48}$ . This expansion works by simply duplicating half of the bits that enter  $f$ . Now, for  $x \in \{0, 1\}^{32}$  that enters  $f$ , the first step is to compute  $E(x) \oplus k$  where  $k$  is the subkey of this round (note that this is where the dependence on the key is introduced). Next, the result is divided into 8 blocks of size 6 bits each. Each block is then run through a lookup table (called an S-box), that takes 6 bits and outputs 4 bits. There are 8 S-boxes, denoted  $S_1, \dots, S_8$ . Notice that these boxes are *not* invertible. The result of this computation is  $8 \times 4 = 32$  bits (because the S-boxes reduce the size of the output). The computation of  $f$  is then completed by permuting the resulting 32 bits. We note that the expansion function  $E$ , the S-boxes and the final permutation are all known and fixed. The only unknown quantity is the key  $K$ . See Figure 4.2 for a diagram of the construction.

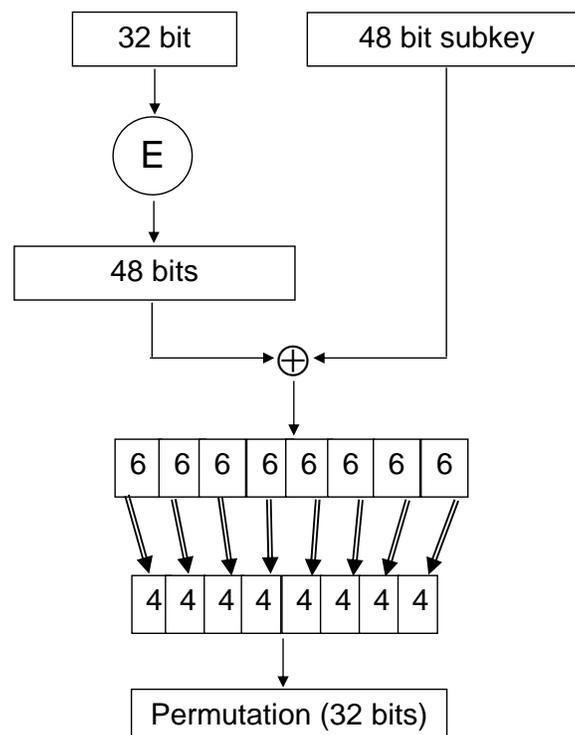


Figure 4.2: The DES  $f$  function

The key schedule works by dividing the 56 bits of the key into two 28 bit blocks. Then, in each round, 24 bits are chosen from each block; the first 24 bits affect only the choices in S-boxes  $S_1, \dots, S_4$  and the second 24 bits affect only the choices in S-boxes  $S_5, \dots, S_8$ .

**The S-boxes.** The definition of the S-boxes is a crucial element of the DES construction. We will now describe some properties of these boxes:

1. Each box can be described as a 4 x 16 table (64 entries corresponding to  $2^6$  possibilities), where each entry contains 4 bits.
2. The 1st and last input bit are used to choose the *row* and the 2-5th bits are used to choose the *column*.
3. Each row in the table is a permutation of  $0, 1, 2, \dots, 15$ .
4. Changing *one input* bit, always changes at least *two output* bits.
5. The S-boxes are very far from any linear function from 6 bits to 4 bits (the hamming distance between the output mapping and the closest linear function is very far).

The above properties are very important, in order to enhance the so-called **avalanche effect**.

**Avalanche effect:** small changes to the input (or key) lead very quickly (i.e., within a few rounds) to major changes in the output. We will now analyze how DES achieves this effect. In order to do this, we will trace the hamming distance between the ciphertexts of two plaintexts that differ by just a single bit.

If the difference is in the side of the input that does not enter the first  $f$ -function, then the hamming distance remains 1. This part of the input is next used as input to  $f$ . If this is not expanded, then the hamming distance remains 1 after the expansion function and XOR with the key. Now, due to the above-described properties of the S-boxes, we have that at least 2 output bits are changed. The permutation then spreads these outputs into far-away locations, so that in the next round the inputs are different in at least 2 S-boxes. Continuing in the above line, we have that at least 4 output bits are modified and so on.

We can see that the avalanche effect grows exponentially and is complete after 4 rounds (i.e., every bit has a chance of being changed after 4 rounds - this is not the case after 3 rounds).

A complete description of the DES standard can be obtained from the FIPS (Federal Information Processing Standards Publications) website of NIST (National Institute of Standards and Technology). The actual document can be obtained from:

<http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>

## Lecture 5

# Block Ciphers (continued)

### 5.1 Attacks on DES

In this part of the lecture we will understand more about the security of DES by considering attacks on it. Our attacks will be mainly for reduced-round variants of DES. Recall that DES has a 16-round Feistel structure; we will show attacks on DES when the number of rounds is significantly reduced.

In all of our attacks, we assume that the adversary has a number of plaintext/ciphertext pairs that were generated by an “encryption” oracle for DES. Furthermore, in Section 5.2.1, we will assume that the adversary can query the oracle for encryptions. This is analogous to either a chosen-plaintext attack, or more appropriately, a distinguisher for a pseudorandom permutation. (Recall that DES, or any block cipher, within itself is supposed to be a pseudorandom permutation and not a secure encryption scheme.) The lack of clear distinctions between encryption schemes and pseudorandom permutations is very problematic. We do not clarify it here because we wish to use the terminology of block ciphers as encryption that is used in practice. Nevertheless, we hope that it is clear that when we refer to block ciphers, we always mean pseudorandom permutations (even if the block cipher is denoted  $E_k$ ).

#### 5.1.1 Single-Round DES

In this case, if we are given a plaintext  $x = x_1, x_2$  and its corresponding ciphertext  $y = y_1, y_2$ , then we know the input and output to the  $f$ -function completely. Specifically, the input to  $f$  equals  $x_2$  and the output from  $f(x_2)$  equals  $y_1 \oplus x_1$ . In order to obtain the key, we will look inside the S-boxes. Notice that given the exact output of  $f$ , we can immediately derive the output of each S-box. As we have seen, each row of an S-box is a permutation over  $0, \dots, 15$ . Therefore, given the output of an S-box, the only uncertainty remaining is from which row it came. Thus, we know the only 4 possible input values that could have occurred. Notice that since we also know the input value to  $f$ , this gives us 4 possible choices for the portion of the key entering the S-box being analyzed. The above is true for each of the 8 S-boxes, and so we have reduced the possible number of keys to  $4^8 = 2^{16}$ . We can try all of these, or we can try a second message and use the same method. The probability of an intersection is very low.

---

\* Lecture notes for an undergraduate course in cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

### 5.1.2 Two-Round DES

Let  $x = x_1, x_2$  be the input and  $y = y_1, y_2$  the output of the two-round DES on  $x$ . Then, we know that the input into  $f$  in the first round equals  $x_2$  and the output of  $f$  in the first round equals  $y_1 \oplus x_1$ . Furthermore, we know that the input into  $f$  in the second round equals  $y_1$  and the output is  $y_2 \oplus x_2$ . We therefore know the inputs and outputs of  $f$  in both rounds, and we can use the same method as for single-round DES. (It is useful to draw a two-round Feistel structure in order to be convinced of the above.)

### 5.1.3 Three-Round DES

In order to describe the attack here, we first denote the values on the wires as in Figure 5.1 below.

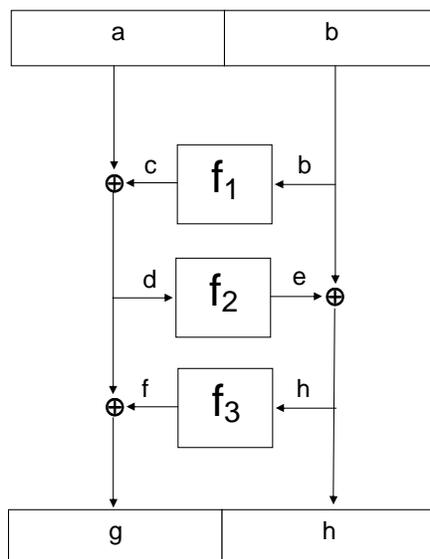


Figure 5.1: Three round DES

We now show an important relation between the values on these wires. First, note that  $a = c \oplus d$  and  $g = f \oplus d$ . Therefore,  $a \oplus g = c \oplus f$ . Since  $a$  and  $g$  are known (from the input/output), we have that  $c \oplus f$  is known.

We conclude that we know the inputs to  $f_1$  and  $f_3$ , and the XOR of their outputs. Namely, have the following relation  $f_1(b) \oplus f_3(h) = a \oplus g$ , where all of  $b, h, a$  and  $g$  are known.

**The attack.** The subkeys in each round are generated by rotating each half of the key. The left half of the key always affects only  $S_1, \dots, S_4$  and the right half of the key always affects only  $S_5, \dots, S_8$ . Since the permutation after the S-boxes is fixed, we also know which bits come out of which S-box.

Now, there are  $2^{28}$  possible half-keys. Let  $k_L$  be a guess for the left half of the key. We know  $b$  and so can therefore compute the output of  $S_1, \dots, S_4$  for the value  $c$  in the case that  $k_L$  is the left half-key. Likewise, we can compute the same locations for the value  $f$  by using input  $h$  and working with the guessed half-key  $k_L$ . We can now compute the XOR of the values obtained by this guess, and see if they match the appropriate bits in  $a \oplus g$  which is known (recall that above we have seen that  $c \oplus f = a \oplus g$ ). Since we consider 16 bits of output, an incorrect key is accepted

with probability approximately  $2^{-16}$ . There are  $2^{28}$  keys and so approximately  $2^{12}$  keys will pass the test as a potential left half of the key.

We can carry out the above separately for each half in time  $2 \cdot 2^{28}$  and obtain approximately  $2^{12}$  candidates for the left half and  $2^{12}$  candidates for the right half. Overall, we remain with  $2^{24}$  candidate keys and we can run a brute-force search over them all (since anyway  $2^{24} < 2^{28}$ ).

The total complexity of the attack is  $2^{28} + 2^{28} + 2^{24} \approx 2^{29}$ .

#### 5.1.4 Brute Force Search

We conclude with the trivial attack that works for all encryption schemes: a brute force search over the key space. Such an attack works by trying each possible key  $k$  and comparing the result of  $\text{DES}_k(m)$  to the known ciphertext  $c$  for  $m$ . If they match, then with high probability this is the correct key. (We can try one or two more plaintexts to be very certain.) For DES, such an attack requires  $2^{55}$  encryptions. Unfortunately, this amount of work is no longer beyond reach for organizations with enough money to build powerful computers, or for large groups of people who are willing to distribute the work over many computers.

#### 5.1.5 The Best Known Attacks on Full DES

The only attacks on the full 16 round DES that go below the  $2^{55}$  time barrier are due to Biham and Shamir [7], and Matsui [27]. Biham and Shamir's attack is based on differential cryptanalysis and requires obtaining  $2^{47}$  chosen plaintext/ciphertext pairs. This was a breakthrough result in that it was the first to beat an exhaustive search on the full 16-round DES. However, in practice, it is far more feasible to run an exhaustive search than to obtain so many chosen plaintext/ciphertext pairs. Later, Matsui used linear cryptanalysis to further reduce the attack to one whereby it suffices to see  $2^{43}$  plaintext/ciphertext pairs. (Note that Matsui's attack reduces both the amount of pairs, but also does not require *chosen* pairs.) Once again, an exhaustive search on the key space is usually more feasible than this.

#### 5.1.6 Further Reading

In this course, we do not have time to show more sophisticated attacks on DES. Two important attack methodologies are *differential cryptanalysis* and *linear cryptanalysis*. We strongly recommend reading [7]. It suffices to read the shorter conference version from CRYPTO'90 (a few hours of work should be enough to get the idea). If you have difficulty obtaining it, I will be happy to send you a copy. However, first check in the library in the Lecture Notes in Computer Science series (LNCS), Volume 537, Springer-Verlag, 1991.

## 5.2 Increasing the Key-Size for DES

As we have seen, DES is no longer secure enough due to the fact that its key size is too small. We now consider potential ways of increasing this size.

### 5.2.1 First Attempt – Alternating Keys

Instead of using one 56-bit key, choose two independent 48 bit keys  $k_1$  and  $k_2$ . Then, use  $k_1$  in the odd rounds (1, 3, 5 etc.) of the Feistel structure, and use  $k_2$  in the even rounds. Notice that

decryption is exactly the same except that  $k_2$  is used in the odd rounds and  $k_1$  in the even rounds. The total key size obtained is 96 bits.

We now show that it is possible to find the key in  $2^{49}$  steps, using oracle queries to the DES encryption machine. In order to see this, first assume that we have found  $k_1$ . We can then eliminate the first round because we can compute  $f_{k_1}(x_2) \oplus x_1$  (where the plaintext equals  $x = x_1, x_2$ ). More specifically, for input  $x = x_1, x_2$  we wish to find an input  $x' = x'_1, x'_2$  such that after the first round, the values on the wires are  $x = x_1, x_2$ . This can be achieved by setting  $x'_2 = x_2$  and  $x'_1 = f_{k_1}(x'_2) \oplus x_1$ . Thus, after the first round we have  $x_2$  still on the right wire and  $f_{k_1}(x'_2) \oplus x'_1$  on the left wire. But,  $f_{k_1}(x'_2) \oplus x'_1 = f_{k_1}(x'_2) \oplus f_{k_1}(x'_2) \oplus x_1 = x_1$  as required. Likewise, we can add another round of  $k_1$  at the end by computing  $y'_1 = y_1 \oplus f_{k_1}(y_2)$  and  $y'_2 = y_2$ .

It is now possible to use the encryption oracle to essentially decrypt a value. That is, recall that decryption is the same as encryption, except that the order of  $k_1$  and  $k_2$  is reversed. By adding the rounds as above, we achieve a decryption oracle from the encryption oracle. Furthermore, this decryption oracle can be used to test our guess of  $k_1$ .

The actual attack works as follows. First, choose an input  $m$  and obtain a ciphertext  $c$  by querying the oracle. Next, guess  $k_1$  and ask for an encryption of  $c'$  where  $c'$  is computed by removing the first round using  $k_1$ . Upon obtaining the result, use  $k_1$  again to add another round and see if the result equals  $m$ . If yes, then  $k_1$  is correct. Continue in this way until the correct  $k_1$  is found. Once  $k_1$  is found, use a brute force search on  $k_2$ . We obtain that the entire key is found in  $2^{49}$  invocations of DES, and is thus even weaker than the original DES.

### 5.2.2 Second Attempt – Double DES

In Double-DES, two independent 56-bit keys  $k_1$  and  $k_2$  are chosen. Then, encryption is obtained by  $E_{k_1, k_2}(m) = \text{DES}_{k_2}(\text{DES}_{k_1}(m))$ . Thus, we obtain a key-size of  $2^{112}$ , which is much too large for any brute force search.

In this section, we describe a generic attack on any method using “double encryption”. The attack is called a “meet-in-the-middle” attack and finds the keys approximately  $2^{56}$  time and  $2^{56}$  space. (We note that this much space is most probably not feasible, but nevertheless, the margin of security is definitely not large enough.) Since the attack is generic, we will refer to an arbitrary encryption scheme  $E$  of key-size  $n$  and show an attack requiring  $O(2^n)$  time and  $O(2^n)$  space. We will denote  $c = E_{k_2}(E_{k_1}(m))$ ,  $c_1 = E_{k_1}(m)$ , and  $c_2 = E_{k_2}^{-1}(c)$ . (Thus, for the correct pairs of keys,  $c_1 = c_2$ .)

The attack works as follows. Let  $(m, c)$  be an input/output pair from the double-encryption scheme. Then, build a data structure of pairs  $(k_1, c_1)$  sorted by  $c_1$  (this is built by computing  $c_1 = E_{k_1}(m)$  for every  $k_1$  where  $m$  is as above). The size of this list is  $2^n$ . Similarly, we build a list of pairs  $(k_2, c_2)$ , sorted by  $c_2$ , by computing  $c_2 = E_{k_2}^{-1}(c)$  for every  $k_2$  using the  $c$  above. Note that sorting these lists costs  $2^n$  only because we can use a *counting sort*. That is, we start with a table of size  $2^n$  and use the  $c_1$  and  $c_2$  values as indices into the table. (The entries in the table contain the appropriate  $k_1$  and  $k_2$  values.)

Next, we find all the pairs  $(k_1, c_1)$  and  $(k_2, c_2)$  such that  $c_1 = c_2$ . Since the lists are sorted according to  $c_1$  and  $c_2$ , it is possible to do this in time  $2 \cdot 2^n$ . Now, we expect in the order of  $2^n$  pairs because the range of a random function is most of  $2^n$ . Therefore, each value  $\hat{c} \in \{0, 1\}^n$  should appear approximately once in each table. This implies that each  $\hat{c}$  in the first table should have approximately one match in the second table. This yields approximately  $2^n$  candidates.

The attack is concluded by testing all of these candidate pairs of keys on a new pair  $(m', c')$  obtained from the double-encryption oracle. Given a few such input/output pairs, it is possible to

be sure of the correct pair  $(k_1, k_2)$  with very high probability.

### 5.2.3 Triple DES (3DES)

The extension of choice that has been adopted as a standard is Triple-DES (or 3DES). There are two variants:

1. Choose 3 independent keys  $k_1, k_2, k_3$  and compute  $E_{k_1, k_2, k_3}(m) = \text{DES}_{k_3}(\text{DES}_{k_2}^{-1}(\text{DES}_{k_1}(m)))$ .
2. Choose 2 independent keys  $k_1, k_2$  and compute  $E_{k_1, k_2}(m) = \text{DES}_{k_1}(\text{DES}_{k_2}^{-1}(\text{DES}_{k_1}(m)))$ .

The reason for this strange alternation between encryption and decryption is to enable one to set  $k_1 = k_2 = k_3$  and then obtain single-DES encryption with a triple-DES implementation.

It is strongly believed that 3DES is secure. Its drawback is that it is quite slow since it requires 3 full encryption operations. Its lack of flexibility with respect to key size, its speed and its size are all factors that led to the introduction of AES (the Advanced Encryption Standard) in 2000. We will not have time to describe the AES encryption scheme here.

## 5.3 Modes of Operation

Until now, we have considered encryption of a single block (for DES, the block size is 64 bits but it could be any value); we denote the block-size from here on by  $n$ . An important issue that arises for block ciphers is how to encrypt messages of arbitrary length. The first issue to deal with is to obtain a ciphertext of length that is a multiple of  $n$ . This is obtained by padding the end of a message by  $10 \cdots 0$ . (Actually, there remains a problem in the case that a message is of length  $c \cdot n$  or  $c \cdot n - 1$  for some value  $c$  because in this case some bits may be lost. This is solved by adding an additional block in this case.) A **mode of operation** is a way of encrypting many blocks with a block cipher. We present the four most popular modes of operation, even though some of them are clearly not secure. All four modes are presented in Figure 5.2.

**Mode 1 – Electronic Code Book Mode (ECB).** In this mode, all blocks are encrypted separately. We note that this mode is clearly not secure, first and foremost because the encryption is deterministic and we know that probabilistic encryption is necessary. Furthermore, the blocks are of a small size, so repeated blocks can be detected. This is not just a “theoretical problem” because headers of messages are often of a fixed format. Therefore, the type of message can easily be detected. We note one plus here in that an error that occurs in one block affects that block only.

There are other issues related to data integrity and authentication that are typically raised in the context of modes of operation. However, this is a different issue and our position is that these issues should not be mixed. We will discuss message authentication in Lecture 7.

**Mode 2 – Cipher Block Chaining Mode (CBC).** In this mode, the  $i^{\text{th}}$  block is encrypted by XORing it with the  $i - 1^{\text{th}}$  ciphertext block and then encrypting it. The *IV* value is an initial vector that is chosen uniformly at random. We stress that it is not kept secret and is sent with  $c$ . Here, the encryption is already probabilistic. It has been shown that if  $E_k$  is a pseudorandom permutation, then the CBC model yields a CPA-secure encryption scheme [3]. Some drawbacks of this mode are: encryption must be carried out sequentially (unlike decryption which may be executed in parallel), and error in transmission affects the garbled block and the next one.

**Mode 3 – Output Feedback Mode (OFB).** Essentially, this is a way of turning a block cipher into a stream cipher. As before, a random IV is used each time and sent together with the ciphertext. Here, encryption and decryption are strictly sequential, but most of the encryption work can be carried out independently of the plaintext. Regarding errors, a flipped bit affects only a single bit of the output (unless the error is in the IV).

**Mode 4 – Cipher Feedback Mode (CFB).** Here the stream depends on the plaintext as well as the IV. An advantage of CFB over OFB is that decryption can be carried out in parallel. However, encryption is strictly sequential and depends also on the plaintext. Regarding errors, a single bit affects a single bit in the current block and the entire next block.

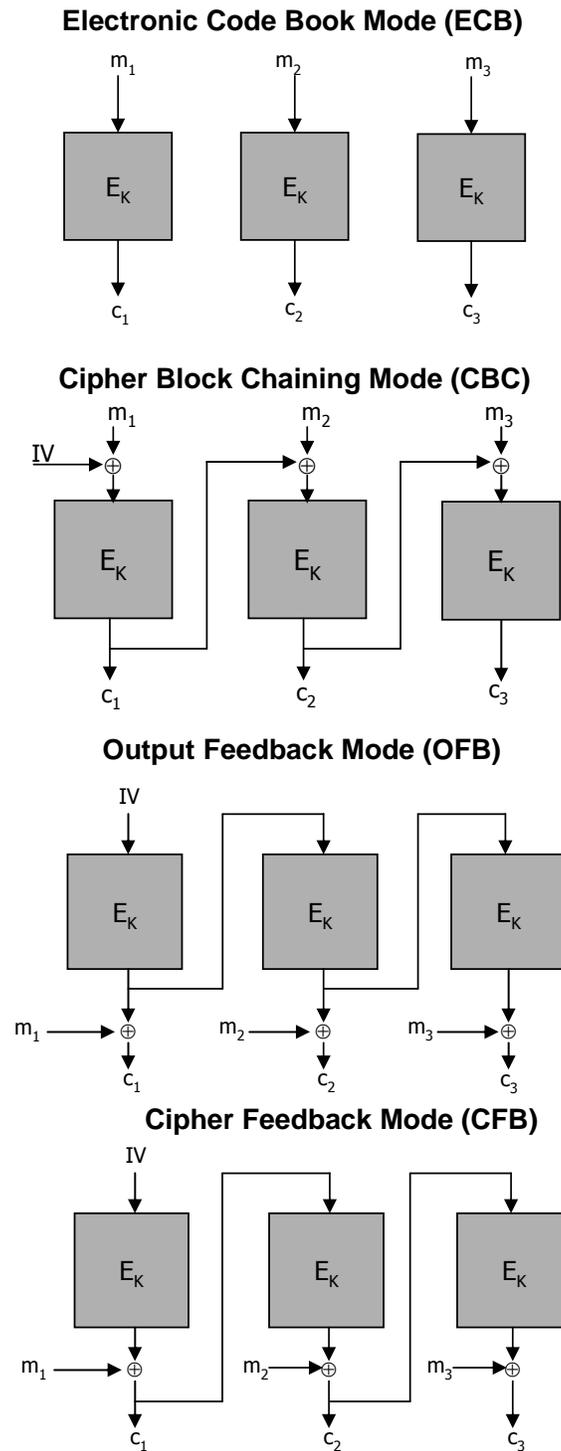


Figure 5.2: Modes of Operation



## Lecture 6

# Collision-Resistant (Cryptographic) Hash Functions

In this lecture, we introduce a new primitive: collision-resistant hash functions. Recall that hash functions (as used in data structures and algorithms) are functions that take arbitrary-length strings and *compress* them into shorter strings. In data structures, the aim is for these short strings to be used as indices in a table; as such the output of the hash function is very short. Furthermore, it is desired that the hash function yields as few collisions as possible (so that only a few elements will end up in each entry in the table). We remark that a truly random function would do the best job here (however, such functions require exponential storage).

Collision-resistant hash functions are also compression functions that take arbitrary-length strings and output strings of a fixed shorter length. However, the “desire” in data structures to have few collisions is converted into a mandatory requirement. Specifically, a *collision-resistant* hash function is secure if no adversary (running in a specified time) can find a collision. That is, let  $h$  be the function. Then, no adversary should be able to find  $x \neq y$  such that  $h(x) = h(y)$ . As in the case of data structures, a random function provides the best collision resistance, because seeing the output of the function in one place is of no help in seeing the output in a different place.

### 6.1 Definition and Properties

We now provide a definition of *collision-resistance* for cryptographic hash functions. We note that any collision-resistant hash function must have a key of some sort. This is due to the fact that finding collisions must be hard for *all* adversaries. In particular, it must be hard for all adversaries in the family of algorithms  $\{A_{x,y} \mid x, y \in \{0,1\}^*\}$  where each  $A_{x,y}$  just outputs the pair  $x$  and  $y$ . Now, for every fixed function  $h$ , there exist  $x'$  and  $y'$  such that  $h(x') = h(y')$ ; therefore, the adversary  $A_{x',y'}$  with this  $x'$  and  $y'$  will “find” a collision. However, when keys are used, it is possible to require that every adversary will succeed with low probability; we note that this probability is also over the choice of the key.

We stress that *unlike for encryption and other cryptographic tasks*, the hash key is *public knowledge* once it has been chosen. As previously, for simplicity we will assume that the key for a hash function  $h : \{0,1\}^* \rightarrow \{0,1\}^n$  is of length  $n$ .

**Definition 6.1** (collision-resistant hash functions): *Let  $H = \{h_k\}_{k \in \{0,1\}^n}$  be a polynomial-time*

---

\* Lecture notes for an undergraduate course in cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

computable family of functions such that for every  $k \in \{0, 1\}^n$ ,  $h_k : \{0, 1\}^* \rightarrow \{0, 1\}^n$ . We say that  $H$  is  $(t, \epsilon)$ -collision resistant if for every adversary  $A$  running for at most  $t$  steps

$$\Pr_{k \leftarrow \{0, 1\}^n} [A(k) = (x, y) \text{ such that } h_k(x) = h_k(y) \ \& \ x \neq y] < \epsilon$$

**Inherent limitations on collision resistance.** As we have mentioned, the “best” collision resistance is obtained by random functions. We therefore begin by studying the collision resistance of a truly random function  $h$  such that  $h : \{0, 1\}^* \rightarrow \{0, 1\}^n$ .

First, it is clear that any adversary can find a collision in  $h$  in time  $2^n + 1$ . This holds by the pigeon-hole principle (by computing  $h(x_i)$  for  $2^n + 1$  different values of  $x_i$ , we must obtain  $x_i \neq x_j$  such that  $h(x_i) = h(x_j)$ ).

However, by the birthday paradox, a collision can actually be found in time  $\sqrt{2^n} = 2^{n/2}$ . That is, by computing  $h(x_i)$  for  $\sqrt{2^n}$  random values  $x_i$ , we obtain that with probability over  $1/2$  we will have obtained  $x_i \neq x_j$  such that  $h(x_i) = h(x_j)$ . We therefore conclude that an inherent limitation on the security of collision-resistant hash functions is that the running-time  $t$  of the adversary must be considerably lower than  $\sqrt{2^n}$ .

**The Birthday Paradox.** For the sake of completeness, we provide a proof of the birthday paradox. Actually, we will prove that the expected number of collisions with  $\sqrt{2^n}$  random values equals 1; this is simpler to prove.

Consider a game with  $N$  bins and  $q$  balls. The  $q$  balls are thrown randomly into the bins (i.e., with the uniform distribution), and we ask for what value of  $q$  do we obtain that the expected number of bins with 2 or more balls is at least 1. In the context of hash functions, the number of bins equals the size of the range of the hash function (i.e.,  $N = 2^n$ ) and throwing a ball uniformly into a bin is equivalent to the process of choosing a random  $x$  and computing  $h(x)$ , for a random function  $h$ .

Denote the balls by  $b_1, \dots, b_q$  and define a boolean random variable  $X_{i,j}$  such that  $X_{i,j} = 1$  if and only if balls  $i$  and  $j$  fall in the same bin. The probability that two *given* balls  $i$  and  $j$  fall in the same bin equals  $1/N$  exactly. Therefore, by the definition of expectation:

$$E[X_{i,j}] = 1 \cdot \frac{1}{N} + 0 \cdot \left(1 - \frac{1}{N}\right) = \frac{1}{N}$$

By the linearity of expectations, the expected number of pairs of balls that fall in the same bin is given by:

$$\sum_{i=2}^q \sum_{j=1}^{i-1} E[X_{i,j}] = \binom{q}{2} \frac{1}{N} = \frac{q(q-1)}{2N}$$

Therefore, when  $q \approx \sqrt{2N}$ , we obtain that the expected number of collisions equals 1.

**Collision-resistant hash functions are one-way.** An important and interesting property of one-way hash functions is that they are hard to invert; such functions are called **one-way**. We will not provide a formal definition of such functions in this course. Informally speaking, a function  $f$  is **one-way** if it can be efficiently computed, and if for sufficiently large inputs, the probability of finding some preimage for  $f(x)$  where  $x$  is *uniformly chosen*, is very small.

We claim now that collision-resistant hash functions are one-way. This follows from the fact that if it is possible to find a preimage for  $h(x)$ , then we can also find a collision. In order to see this, notice that for any pair  $x \neq y$  such that  $z = h(x) = h(y)$ , it is impossible to know if  $z$  was

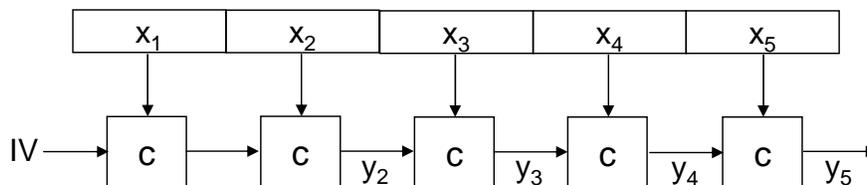


Figure 6.1: The Merkle-Damgård Construction

obtained by computing  $h(x)$  or  $h(y)$ . Now, consider a collision-finding algorithm that chooses a random  $x$ , computes  $z = h(x)$  and gives  $z$  to the preimage-finding algorithm. Upon obtaining back a preimage  $w$  such that  $f(w) = z$ , if  $w = x$  then no collision was found. However, if  $w = y$ , then a collision has been found (because  $x \neq y$  and  $h(x) = h(y) = z$ ). Since  $z$  could have been produced from  $x$  or  $y$  with the same probability (this value is randomly chosen), the probability that the preimage is the same value as was originally chosen is at most  $1/2$ .

**Un-keyed hash functions.** As we have seen, any collision-resistant hash function must have keys (even though they are public). Nevertheless, in practice, unkeyed collision-resistant hash functions are used. The best of these functions (e.g., SHA-1) have withstood years of attack and is widely believed to be collision-resistant.<sup>1</sup>

## 6.2 Constructions of Collision-Resistant Hash Functions

Very often, collision-resistant hash functions are constructed in two stages. First, a collision-resistant compression/hash function  $c(\cdot)$  is constructed, where  $c : \{0, 1\}^{n+t} \rightarrow \{0, 1\}^n$ . That is,  $c(\cdot)$  compresses messages of a fixed length  $n + t$  to messages of length  $n$ ; this is called a **fixed-length hash function**. Next, a collision-resistant hash function  $h : \{0, 1\}^* \rightarrow \{0, 1\}$  is constructed using  $c(\cdot)$ ; note that  $h$  takes inputs of any arbitrary length. The key is used in choosing the function  $c(\cdot)$ ; we will ignore this from here on.

Due to time constraints, we will only have time to show how to extend a fixed-length compression function to a full-fledged collision-resistant hash function. Constructions of fixed-length hash functions can be found in [15].

**The Merkle-Damgård construction [16, 29].** The idea of the construction is to break the input up into blocks, and iteratively apply the fixed-length hash function to the new block, and the result of the previous application; see Figure 6.1. The output of the function is the output of the last iteration *and* the length of the input.

For simplicity, we will define the Merkle-Damgård construction for the case that  $t = n$  (i.e., the compression function is such that its input is twice the length of its output). Let  $c(\cdot) : \{0, 1\}^{2n} \rightarrow \{0, 1\}^n$  be a compression function. Then, upon receiving any input  $x$  of length  $N$ , define  $x = x_1, x_2, \dots, x_\ell$  where  $\ell = \lceil N/n \rceil$ . That is, each block is of length  $n$  (the last block can be padded with zeros). Furthermore, let  $y_0 = IV$  be a *fixed* initial vector of length  $n$  (this can even be  $0^n$ ). Then, for every  $i = 1, \dots, \ell$ , compute  $y_i = c(y_{i-1}, x_i)$ . Finally, define  $h(x) = (y_\ell, |x|)$ . Note that the  $IV$  is used in order to ensure that  $h$  is applied on inputs of length  $2n$  each time (although we could just as easily applied  $h$  to  $x_1$  and  $x_2$ ).

<sup>1</sup>We note, however, that some widely-used hash functions, like MD5, are known to *not* be collision-resistant.

**Theorem 6.2** Assume that  $c : \{0, 1\}^{2n} \rightarrow \{0, 1\}^n$  is  $(t, \epsilon)$ -collision-resistant. Then, the Merkle-Damgård construction described above yields a  $(t', \epsilon')$ -collision-resistant hash function  $h : \{0, 1\}^* \rightarrow \{0, 1\}$ , for  $\epsilon' = \epsilon$  and  $t' = t/2$ .<sup>2</sup>

**Proof:** Assume by contradiction that there exists an adversary  $\mathcal{A}$  that runs in time  $t'$  as above, and with probability  $\epsilon'$  outputs  $x \neq y$  such that  $h(x) = h(y)$ . We differentiate between two cases in which a collision is indeed found by  $\mathcal{A}$ :

1. *Case 1* –  $|x| \neq |y|$ : In this case, it must be that  $h(x) \neq h(y)$  because the lengths  $|x|$  and  $|y|$  are in the output of  $h$ .
2. *Case 2* –  $|x| = |y|$ : Let  $\alpha_1, \dots, \alpha_\ell$  and  $\beta_1, \dots, \beta_\ell$  be the outputs of  $c(\cdot)$  while computing  $h(x)$  and  $h(y)$ , respectively. (That is,  $\alpha_i = c(\alpha_{i-1}, x_i)$  as described above.) Now, since  $h(x) = h(y)$  it follows that  $c(\alpha_{\ell-1}, x_\ell) = c(\beta_{\ell-1}, y_\ell)$ . If  $(\alpha_{\ell-1}, x_\ell) \neq (\beta_{\ell-1}, y_\ell)$ , then we have finished because we have found a collision in  $c$ . Otherwise, we continue backwards until we find some  $i$  for which  $c(\alpha_{i-1}, x_i) = c(\beta_{i-1}, y_i)$  and yet  $(\alpha_{i-1}, x_i) \neq (\beta_{i-1}, y_i)$ . Note, that such an  $i$  must exist because if  $(\alpha_{i-1}, x_i) = (\beta_{i-1}, y_i)$  for every  $i$ , then  $x = y$ . For this  $i$ , once again, we have found a collision.

This yields an adversary  $\mathcal{A}_c$  that finds a collision in  $c$ . Specifically,  $\mathcal{A}_c$  invokes  $\mathcal{A}$ . Then, upon receiving  $\mathcal{A}$ 's output  $x$  and  $y$ , adversary  $\mathcal{A}_c$  first checks if  $h(x) = h(y)$ . If not,  $\mathcal{A}_c$  halts. If yes, it computes the series  $\alpha_1, \dots, \alpha_\ell$  and  $\beta_1, \dots, \beta_\ell$  and finds the index  $i$  as above (as argued above, such an  $i$  must exist).  $\mathcal{A}_c$  then outputs the pair of messages  $x' = (\alpha_{i-1}, x_i)$  and  $y' = (\beta_{i-1}, y_i)$  and halts.

Clearly,  $\mathcal{A}_c$  finds a collision with probability  $\epsilon = \epsilon'$ . Furthermore, the running-time of  $\mathcal{A}_c$  equals the running-time of  $\mathcal{A}$  with the additional overhead of computing  $h$  twice and comparing  $\ell = |x|/n$  strings of length  $n$ . Therefore, under the assumption that  $\mathcal{A}$ 's running-time is significantly longer than the time to compute  $h$  twice (see Footnote 2), we have that  $t < 2t'$ , as required.

This completes the proof. ■

**Efficiency considerations.** Notice that the hash function  $h$  can be computed by carrying out one pass on the input, while storing a small amount of state information. This makes computation of  $h$  very efficient, even for very large files.

**Hash functions in practice.** The most popular collision-resistant hash functions used today are MD5 and SHA-1. Collisions have been found in MD5, and so this hash function should not be used from now on. SHA-1 is the accepted standard of NIST; it has an output of size 160 bits and so is not vulnerable to a birthday attack. For information about MD5 and SHA-1, see [31] and [40], respectively.

### 6.3 Popular Uses of Collision-Resistant Hash Functions

One problem that arises in practice is how and where to store the password file that is used for login. Clearly, if this file is stolen, then all user's passwords are revealed. In Unix, the solution

---

<sup>2</sup>We assume for simplicity that the running-time of any adversary for the Merkle-Damgård construction is significantly greater than the time it takes to compute  $h$  on its two output strings  $x$  and  $y$ .

to this problem is to store only *hashed* passwords in the password file. That is, each entry in the table is  $(user_i, h(pwd_i))$  where  $h$  is a collision-resistant hash function. Then, when the Unix server receives a request for logon of  $\langle user_i, pwd_i \rangle$ , it computes  $h(pwd_i)$  and compares it with the entry in the table.

The idea behind this construction is that, as we have seen, collision-resistant hash functions are one-way. Therefore, obtaining the password file does not reveal the individual passwords which are needed for login. (Notice that knowing  $h(pwd_i)$  does not help for logging into the server, because the server expects to receive  $pwd_i$  and not  $h(pwd_i)$ .)

**Dictionary attacks on passwords.** The idea of using a hash function to protect the password file is OK as a secondary measure. Specifically, if the passwords are uniformly chosen from a large set, then it is possible to formally justify this use. However, passwords are typically *not uniformly distributed*, in which case one-wayness is not guaranteed. Furthermore, they typically have low entropy (i.e., are based on a short amount of randomness). Therefore, it is possible to launch a dictionary attack that takes the file and tries all “likely” passwords. This works very well in practice (except for the select few users who use long and random passwords).

In the Unix system, this attack is somewhat limited by actually storing  $\langle user_i, salt_i, h(salt_i, pwd_i) \rangle$  where  $salt_i$  is a random string chosen independently for each user. Since the “salt” is public, it does not slow down the dictionary attack. However, it does have the effect of forcing an attacker to repeat the dictionary attack for every entry. That is, it is not possible to build one big table and then just do lookup.

## 6.4 The Random Oracle Model

Collision-resistant hash functions that are used in practice are often considered to behave like random functions. Of course, this is absurd because these functions have very short descriptions and so do not look at all like a random function. Nevertheless, in practice, this view has been used to provide justification for the security of some schemes that otherwise cannot be proven secure. Such security claims are called “proofs in the random oracle model”.

In the random oracle model, hash functions (like SHA-1) are used in constructions of different cryptographic schemes. Then, a proof of security is provided that holds when the hash functions are replaced with truly random functions. Such claims are essentially heuristic proofs of security for the original scheme; they are *heuristic* because the original scheme that does *not* use a random function, but rather a specific collision-resistant hash function. We warn that such reasoning is very problematic, and counter-examples have been demonstrated. Nevertheless, in some cases where the only efficient protocols that are known are proven secure in the random oracle model, this can be a useful tool. For more information on this issue see [5] (for arguments in favour of the model) and [12] (for arguments against).



## Lecture 7

# Message Authentication

In this lecture, we are going to consider a new security problem. Until now, the only application that we have seen is that of *encryption* where the aim of the parties is to communicate privately over an open channel. However, there are problems that may arise from communication over an open channel that are not connected to privacy. For example, consider the case that a party  $A$  sends an email to party  $B$  that contains a large order for some merchandise. Upon receiving such an order over an open communication line, party  $B$  has to ask itself two questions:

1. Did party  $A$  really send the order?
2. If yes, is the order that  $B$  received the order that  $A$  actually sent (and not, for example, that the size of the order was somehow changed).

Such examples are very common. For example, an order for a bank transfer may not be secret. However, we should hope that it is not possible to change the order so that instead of transferring \$1,000, the sum is changed to \$100,000. This problem is a problem of *message authentication* and is distinct from the problem of *privacy*.

**First attempt – encryption.** At first sight, it seems that encryption should immediately solve the problem of authentication as well. That is, if an encrypted message  $E_k(m)$  looks like “random garbage” to an observer, how is it possible for the message to be tampered with. The easiest way to answer this question is to look at concrete examples. First, consider the one-time pad encryption scheme (or equivalently any encryption via a stream cipher):  $E_k(m) = k \oplus m$ . Then, given a ciphertext  $c$  that encrypts a message  $m$  it is easy to generate a new ciphertext that encrypts  $\bar{m}$  (i.e., the bitwise complement of  $m$ ): simply compute  $\bar{c}$ . Likewise, it is possible to flip the last bit of  $m$  and so on. Next, consider the ECB (electronic code-book) mode of encrypting multiple blocks with a block cipher. Here, blocks can be flipped, thereby changing the encrypted message. Furthermore, blocks can be garbled without the receiver knowing that this wasn’t the intended message.

We conclude that in general, in order to achieve authentication, there must be some sort of additional mechanism that will indicate whether or not the message was tampered with.

**Second attempt – encryption with error correction.** Another possible solution is to apply an error correcting code  $C$  to the message and then encrypt  $m$  together with  $C(m)$ . It is easy to

---

\* Lecture notes for an undergraduate course in cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

see that this is still not sufficient. Once again, consider the one-time pad and consider the simple parity error correcting code (i.e.,  $C(m) = \bigoplus_{i=1}^{|m|} m_i$ , or in other words, the parity of  $m$ ). Then, given  $c = (m, C(m)) \oplus k$ , it is possible to flip the first and last bits of  $c$  (or just to flip an even number of bits in the first part of the ciphertext). It is clear that the parity bit is still legal.

**Third attempt – encryption with collision-resistant hash.** In this solution, the message  $m$  is encrypted together with  $h(m)$ , where  $h$  is a collision-resistant hash function. In order to show that this doesn't work, we remark that in the case of message authentication, privacy is not an issue. Therefore, the message  $m$  may be known to the adversary (and it would not be wise to assume that it is not known). Now, let  $c = E_k(m) = (m, h(m)) \oplus k$  and denote  $c = c_1, c_2$  where  $c_1$  “contains” the encryption of  $m$  and  $c_2$  “contains” the encryption of  $h(m)$ . Then, given  $c = c_1, c_2$  and given knowledge of the message  $m$ , it is possible to compute  $c'_1 = c_1 \oplus m \oplus m'$  and  $c'_2 = c_2 \oplus h(m) \oplus h(m')$ , for any message  $m'$  of the same length as  $m$ . It is easy to see that the message  $c' = c'_1, c'_2$  is a legal encryption of  $m'$ .

## 7.1 Message Authentication Codes – Definitions

We now provide a (fairly informal) definition of message authentication codes. We are still considering the “symmetric key” world, and therefore we assume that the communicating parties  $A$  and  $B$  share a secret key  $k$ . A message authentication code (MAC) is comprised of the following algorithms:

1. A *Key Generation* algorithm that returns a secret key  $k$ . We assume for simplicity that  $k \in_R \{0, 1\}^n$  and so will not refer to this algorithm in the future.
2. A *Tagging* algorithm that given a key  $k \in \{0, 1\}^n$  and an arbitrary-length message  $m \in \{0, 1\}^*$ , returns a tag  $t = \text{MAC}_k(m)$ . (For simplicity, assume  $t \in \{0, 1\}^n$ .)
3. A *Verification* algorithm that given a key  $k$ , a message  $m$  and a candidate tag  $t$ , returns a bit  $b = \text{Verify}_k(m, t)$ .

It is required that for every key  $k$  and every message  $m$ , it holds that  $\text{Verify}_k(m, \text{MAC}_k(m)) = 1$ . We note that if the tagging algorithm  $\text{MAC}$  receives only messages of a fixed size (say,  $n$ ), then we say that the scheme is a **fixed-length MAC**.

**The message authentication game  $\text{Expt}_{\text{MAC}}$ .** The aim of the adversary in attacking a MAC is to generate a legal MAC-tag on a message  $m'$  that was not sent by the parties. (Of course, it is always possible to copy a tag that was generated by the legitimate parties. We remark that this raises a very important issue of *replay*; however, we don't have time to cover this issue here.) In order to model the possible influence that the adversary may have on the messages sent and tagged by the legitimate parties, we allow it full control over the messages that are tagged. The experiment is defined as follows:

A random key  $k$  is chosen. The adversary  $\mathcal{A}$  is then given full oracle access to the tagging algorithm  $\text{MAC}_k(\cdot)$ . At the end,  $\mathcal{A}$  outputs a pair  $(m', t')$ . Let  $Q$  be the set of messages  $m$  that  $\mathcal{A}$  queried its oracle during the experiment. Then, we say that  $\mathcal{A}$  succeeded, denoted  $\text{Expt}_{\text{MAC}}(\mathcal{A}) = 1$ , if  $m' \notin Q$  and  $\text{Verify}_k(m', t') = 1$ .

**Definition 7.1** An message authentication scheme MAC is  $(t, \epsilon)$ -secure if for every adversary  $\mathcal{A}$  that runs for at most  $t$  steps,

$$\Pr[\text{Expt}_{\text{MAC}}(\mathcal{A}) = 1] < \epsilon$$

We remark that the above attack model that we have defined is called a *chosen message attack* because the adversary is given the power to obtain a tag for any message that it chooses.

## 7.2 Constructions of Secure Message Authenticate Codes

It is immediate to see that a pseudorandom function serves as a good message authentication code. However, in the typical case that the pseudorandom function receives messages of a fixed length  $n$ , the straightforward application of this yields a *fixed-length* message authentication code. We therefore have:

**Theorem 7.2** Let  $F_k$  be a  $(t, \epsilon)$ -pseudorandom function, and for  $m \in \{0, 1\}^n$ , define  $\text{MAC}_k(m) = F_k(m)$  and  $\text{Verify}_k(m, t) = 1$  if and only if  $F_k(m) = t$ . Then, the scheme defined by MAC and Verify is a fixed-length  $(O(t), \epsilon + 2^{-n})$ -secure message authentication code.

**Proof:** Left for an exercise. ■

Of course, fixed-length message authentication codes are somewhat unsatisfactory. Fortunately, we can use collision-resistant hash functions in order to construct arbitrary-length MACs from fixed-length MACs. In the construction below, we assume that the range of the hash function is  $\{0, 1\}^n$ . We have:

**Theorem 7.3** Let  $(\text{MAC}, \text{Verify})$  be a  $(t_1, \epsilon_1)$ -secure MAC and let  $H$  be a  $(t_2, \epsilon_2)$ -collision-resistant family of hash functions. Furthermore, let  $t_{\text{MAC}}$  equal the time that it takes to compute MAC and let  $t_h$  be the time that it takes to compute  $h$ . Then the scheme  $(\text{MAC}', \text{Verify}')$  defined by  $\text{MAC}'_{k,k'}(m) = \text{MAC}_k(h_{k'}(m))$  and  $\text{Verify}'_{k,k'}(m, t) = \text{Verify}_k(h_{k'}(m), t)$ , is an arbitrary-length  $(t_3, \epsilon_3)$ -secure message authentication code, where  $t_3 = \max\{\frac{t_1}{t_h}, \frac{t_2}{t_{\text{MAC}}+t_h}\}$  and  $\epsilon_3 = \epsilon_1 + \epsilon_2$ .

**Proof:** The idea behind the proof of this theorem is as follows. If the MAC adversary finds a collision in  $h$  with probability greater than  $\epsilon_2$ , then this contradicts the security of  $H$ . However, if no such collision is found, then the security of  $\text{MAC}'$  reduces immediately to the security of the fixed-length scheme MAC. (If no collisions are found, then it is equivalent to the case that only messages of length- $n$  are used.) The proof follows.<sup>1</sup>

Let  $\mathcal{A}_{\text{MAC}}$  be an adversary attacking the MAC scheme and running in time at most  $t_3$ . Furthermore, let  $\epsilon$  be the probability that  $\mathcal{A}_{\text{MAC}}$  succeeds in  $\text{Expt}_{\text{MAC}}$ . Now, consider the following two disjoint events:

1.  $\mathcal{A}_{\text{MAC}}$  succeeds in  $\text{Expt}_{\text{MAC}}$  and the pair  $(m', t')$  output by  $\mathcal{A}_{\text{MAC}}$  is such that there exists a message  $m \in Q$  for which  $h(m) = h(m')$ .
2.  $\mathcal{A}_{\text{MAC}}$  succeeds in  $\text{Expt}_{\text{MAC}}$  and the pair  $(m', t')$  output by  $\mathcal{A}_{\text{MAC}}$  is such that for every  $m \in Q$  it holds that  $h(m) \neq h(m')$ .

Since the above events are disjoint, and yet cover all possibility of success in  $\text{Expt}_{\text{MAC}}$ , it follows that  $\epsilon = \epsilon' + \epsilon''$  where  $\epsilon'$  is the probability that the first event occurs and  $\epsilon''$  is the probability that the second event occurs. We now bound the values of  $\epsilon'$  and  $\epsilon''$ .

<sup>1</sup>This proof will not be given in class, but is provided for completeness. Students are strongly encouraged to read and understand the proof.

**THE REDUCTION TO COLLISION RESISTANCE.** We begin by constructing an adversary  $\mathcal{A}_h$  for the hash function.  $\mathcal{A}_h$  receives a key  $k'$  for the hash function, chooses a key  $k \in_R \{0,1\}^n$  for the MAC scheme, and invokes  $\mathcal{A}_{\text{MAC}}$ . For every query  $m$  output by  $\mathcal{A}_{\text{MAC}}$ , the adversary  $\mathcal{A}_h$  computes  $t = \text{MAC}_k(h_{k'}(m))$ , adds  $m$  to the set  $Q$  (initialized at  $\phi$ ), and returns  $t$  to  $\mathcal{A}_{\text{MAC}}$ . Then, after  $\mathcal{A}_{\text{MAC}}$  outputs a pair  $(m', t')$ , the adversary  $\mathcal{A}_h$  checks if there exists a message  $m \in Q$  such that  $h(m) = h(m')$ . If yes,  $\mathcal{A}_h$  outputs  $(m, m')$  and halts. (Otherwise, it outputs fail.) First, note that  $\mathcal{A}_h$ 's running-time equals the running-time of  $\mathcal{A}_{\text{MAC}}$ , plus the time that it takes to compute all of the MAC and hash values. Using a coarse upper-bound, we have that the number of queries is bounded by  $\mathcal{A}_{\text{MAC}}$ 's running-time. Therefore, if  $\mathcal{A}_{\text{MAC}}$  runs in time  $t(\mathcal{A}_{\text{MAC}})$ , the adversary  $\mathcal{A}_h$  runs in time  $t(\mathcal{A}_{\text{MAC}}) \cdot (t_{\text{MAC}} + t_h)$ . Since  $H$  is  $(t_2, \epsilon_2)$ -collision-resistant, we must have that  $t(\mathcal{A}_{\text{MAC}}) \cdot (t_{\text{MAC}} + t_h) < t_2$ . However,  $t(\mathcal{A}_{\text{MAC}}) \leq t_3$  and  $t_3 \leq t_2 / (t_{\text{MAC}} + t_h)$ ; this therefore holds. Given that this holds, we have that  $\mathcal{A}_h$  can succeed with probability at most  $\epsilon_2$ . Since  $\mathcal{A}_h$  succeeds in finding a collision with probability exactly  $\epsilon'$ , it follows that  $\epsilon' \leq \epsilon_2$ .

**THE REDUCTION TO THE FIXED-LENGTH MAC.** As above, we construct an adversary  $\mathcal{A}_{\text{FLM}}$  for the fixed-length MAC scheme.  $\mathcal{A}_{\text{FLM}}$  chooses a key  $k'$  for the hash function and invokes  $\mathcal{A}_{\text{MAC}}$ . For every query  $m$  output by  $\mathcal{A}_{\text{MAC}}$ , the adversary  $\mathcal{A}_{\text{FLM}}$  computes  $h(m)$ , queries its own MAC oracle, and returns the reply  $t$  that it received back to  $\mathcal{A}_{\text{MAC}}$ . Then, after  $\mathcal{A}_{\text{MAC}}$  outputs a pair  $(m', t')$ , the adversary  $\mathcal{A}_{\text{FLM}}$  checks if there exists a message  $m \in Q$  such that  $h(m) = h(m')$ . If yes, it outputs fail. Otherwise, it outputs  $(h(m'), t')$  and halts. First, note that  $\mathcal{A}_{\text{FLM}}$ 's running-time equals the running-time of  $\mathcal{A}_{\text{MAC}}$  plus the time it takes to compute the hash values. Therefore, we have that  $\mathcal{A}_{\text{FLM}}$ 's running-time is bounded by  $t(\mathcal{A}_{\text{MAC}}) \cdot t_h$ . Since the fixed-length MAC is  $(t_1, \epsilon_1)$ -secure, we must have that  $t(\mathcal{A}_{\text{MAC}}) \cdot t_h < t_1$ . However,  $t(\mathcal{A}_{\text{MAC}}) \leq t_3$  and  $t_3 \leq t_1 / t_h$ ; this therefore holds. Given that this holds, we have that  $\mathcal{A}_{\text{FLM}}$  can succeed with probability at most  $\epsilon_1$ . Since  $\mathcal{A}_{\text{FLM}}$  succeeds in  $\text{Expt}_{\text{MAC}}$  with probability exactly  $\epsilon''$ , it follows that  $\epsilon'' \leq \epsilon_1$ .

**CONCLUDING THE PROOF.** We have seen that for the value of  $t_3$  specified in the theorem statement, it must hold that  $\epsilon' \leq \epsilon_2$  and  $\epsilon'' \leq \epsilon_1$ . Therefore,  $\epsilon = \epsilon' + \epsilon'' \leq \epsilon_1 + \epsilon_2 = \epsilon_3$ . That is,  $\mathcal{A}_{\text{MAC}}$  can succeed with probability at most  $\epsilon_3$ , as required. ■

### 7.3 Practical Constructions of Message Authentication Codes

We discuss two constructions here: CBC-MAC and HMAC. Both constructions are industry standards and are in wide use.

**Fixed-length CBC-MAC.** This construction is based on the CBC mode of encryption; see Figure 7.1. Note that no initial vector  $IV$  is used here. The security of this construction is given in the following theorem:

**Theorem 7.4** *Let  $\ell$  be any fixed value. If the block-cipher  $E_k$  is a pseudorandom function, then CBC-MAC is a pseudorandom function for the domain  $\{0,1\}^\ell$ .*

We stress that the theorem holds only for inputs of a *fixed length* (it doesn't matter what the length is, as long as all messages that are used of this length). Of course, if  $\ell$  is not a multiple of  $n$  then some padding must be used; it suffices to concatenate a single 1 followed by zeros.

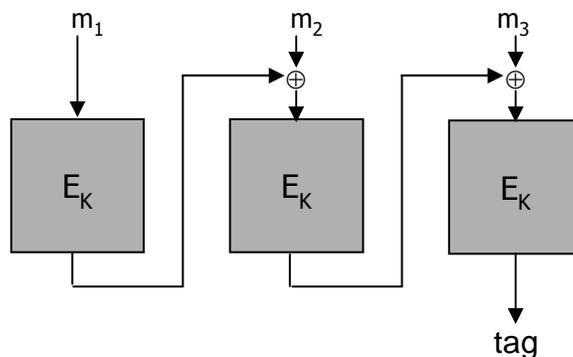


Figure 7.1: The CBC-MAC construction

**Variable-length CBC-MAC.** We now show that CBC-MAC is *not secure* for variable-length messages by constructing an adversary  $\mathcal{A}$ . The adversary  $\mathcal{A}$  chooses any message  $m \in \{0, 1\}^n$  of length  $n$  (recall  $n$  is the domain and range of  $E_k$ ). Then,  $\mathcal{A}$  queries its MAC oracle with  $m$  and obtains back a tag  $t_m$ . Next,  $\mathcal{A}$  queries its MAC oracle with  $t_m$  and receives back  $t_{t_m}$ . Finally,  $\mathcal{A}$  outputs  $(m', t')$  where  $m' = m \| 0^n$  and  $t' = t_{t_m}$ . By the iterative nature of the CBC construction, this is a correct pair.

In order to obtain an arbitrary-length secure MAC via the CBC construction, there are three options:

1. Apply the pseudorandom function (permutation) to the block length  $\alpha$  of the input message in order to obtain a key  $k_\alpha$ . Then, compute the original CBC-MAC using the key  $k_\alpha$  and send the resulting tag along with the block length.
2. *Prepend* the message with its block length, and then compute the CBC-MAC (the first block contains the number of blocks to follow). We stress that appending the message with its block length is *not secure*.
3. Use two different keys  $k_1$  and  $k_2$ . Then, compute the CBC-MAC using  $k_1$  and apply the pseudorandom function to the resulting tag, but this time using  $k_2$ . This has the advantage that the length of the message need not be known until the end.

The interested student is referred to [4] for more details.

**HMAC.** Due to lack of time, we will not see the HMAC construction. Loosely speaking, this construction is based on applying collision-resistant hash functions to the message and a secret key. We stress that there is no solid reasoning for computing a MAC by  $\text{MAC}_k(m) = h(k \| m)$ ; the HMAC construction is more involved than this. See [2] for more details.



## Lecture 8

# Various Issues for Symmetric Encryption

In this lecture, we will deal with three distinct topics: how to combine encryption and authentication, the notion of CCA-secure encryption, and the problem of key management in the world of private-key encryption.

### 8.1 Combining Encryption and Message Authentication

Until now, we have seen how it is possible to encrypt messages, thereby guaranteeing *privacy*. We have also showed how to generate secure message authentication codes, thereby guaranteeing *data authenticity* or *integrity*. The question that we deal with here is how to obtain both of these goals simultaneously.

There are three common approaches to combining encryption and message authentication:

1. *Encrypt-and-authenticate*: Here encryption and message authentication are computed and sent separately. This methodology is deployed in SSH.
2. *Authenticate-then-encrypt*: Here a MAC is first computed and then the message and MAC are encrypted together. This methodology is used in SSL.
3. *Encrypt-then-authenticate*: Here, the message is first computed and a MAC is then computed on the encrypted message. This methodology is used in IPsec.

We now discuss each of these approaches; for a full treatment, see [26].

**Encrypt-and-authenticate.** In this approach, an encryption and message authentication code are computed and sent separately. That is, let  $E$  be an encryption scheme and  $\text{MAC}$  a message authentication code. Then, the combined message is simply  $(E_{k_1}(m), \text{MAC}_{k_2}(m))$ .

This combination does *not* yield a secure encryption scheme. First, notice that at least by definition, a secure MAC does not necessarily imply privacy. Specifically, if  $\text{MAC}$  is a secure message authentication code, then so is the scheme  $\text{MAC}'_k(m) = (m, \text{MAC}_k(m))$ . Therefore, the combination  $(E_{k_1}(m), \text{MAC}_{k_2}(m))$  may actually reveal  $m$ . This is enough because any secure methodology should be secure for *any* instantiation of the underlying building blocks. (Otherwise, at some later stage

---

\* Lecture notes for an undergraduate course in cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

the MAC may be changed with catastrophic results. Notice, that this change may also take place within some library, and without touching the protocol that uses encrypt-and-authenticate.) For anyone not convinced by this argument because message authentication codes used in practice do not reveal information about the message, we note that they are nevertheless typically *deterministic* functions of the message and key. It is therefore possible to detect immediately if two encryptions are of the same message. (Recall that any CPA-secure encryption scheme must not reveal this information.) In addition, privacy may be compromised due to the same problem as will be outlined in the authenticate-then-encrypt methodology.

**Authenticate-then-encrypt.** In this approach, a message authentication code  $\text{MAC}_{k_2}(m)$  is first computed, next the pair  $(m, \text{MAC}_{k_2}(m))$  is encrypted, and finally the ciphertext  $E_{k_1}(m, \text{MAC}_{k_2}(m))$  is sent. This combination also does *not* yield a secure encryption scheme. In order to see this, consider the following encryption scheme:

- Let  $\text{Transform}(m)$  be as follows: any 0 in  $m$  is transformed to 00, and any 1 in  $m$  is transformed randomly to 01 or 10. (Note that encodings of this type may be used for any number of reasons.)
- Define  $E'_k(m) = E_k(\text{Transform}(m))$ , where  $E$  is a stream cipher that works by generating a new pseudorandom stream for each message to encrypt, and then XORs the stream with the input message. (For example, this can be obtained by using a secure block cipher in OFB mode with a new random IV for each encryption.) Note that both the encryption schemes  $E$  and  $E'$  are CPA secure.

We now show an attack on the authenticate-then-encrypt methodology with this encryption scheme. This attack works under the assumption that an adversary can find out if a given ciphertext is valid (this assumption is very real in practice where a server returns an error message in case it receives an invalid message). Given a ciphertext  $c = E'_{k_1}(m) = E_{k_1}(\text{Transform}(m, \text{MAC}_{k_2}(m)))$ , the attacker simply flips the first two bits of  $c$  (i.e., takes their complement), and queries if the resulting ciphertext is valid. If yes, then the adversary knows that the first bit of the message sent equals 1; otherwise it is 0. We conclude that this scheme does not preserve privacy. (We remark that this same counter-example works for encrypt-and-authenticate as well.)

We stress that this demonstrates that the above combination is *not always* secure. However, there are some instantiations that are secure (for example, the specific encryption scheme and MAC used within SSL are secure); see [26]. Nevertheless, as mentioned above, it is bad practice to use a methodology whose security depends on specific implementations.

**Encrypt-then-authenticate.** In this approach, an encryption  $c = E_{k_1}(m)$  is first computed, next a message authentication code  $\text{MAC}_{k_2}(c)$  is computed on  $c$ , and finally  $(c, \text{MAC}_{k_2}(c))$  is sent. For *any* encryption scheme  $E$  that is CPA-secure and *any* secure MAC, this combination *is* secure. Intuitively, privacy is preserved because the MAC is computed over  $c$ . Therefore, no function of  $m$  is computed that could reveal information. Furthermore, the security of the MAC ensures that the adversary cannot cause the legitimate parties to accept any ciphertext (and so any message) that was not generated by them. The combination of encryption and authentication in this way yields so-called **secure channels**, that provide both authenticity and privacy. For more information on this topic (and a formal proof that encrypt-then-authenticate yields the desired result), see [13].

**Independent keys.** We conclude by stressing a basic principle of security and cryptography: *different security goals should always use different keys.*<sup>1</sup> That is, if an encryption scheme and a message authentication scheme are both needed, then independent keys should be used for each one. In order to illustrate this here, consider what happens to the *encrypt-then-authenticate* methodology when the same key  $k$  is used both for the encryption scheme  $E$  and for the message-authentication code  $MAC$ . We provide a concrete example using pseudorandom-function based encryption and a pseudorandom-function based MAC. That is, let  $f$  be a pseudorandom permutation. Then, it follows that both  $f$  and  $f^{-1}$  are pseudorandom functions. Define:  $E_k(m) = f_k(r, m)$  for a random  $r \in_R \{0, 1\}^n$ , and  $MAC_k(c) = f_k^{-1}(c)$ . Now, the combined encryption and authentication of a message  $m$  with the same key  $k$  equals:

$$\langle c = E_k(m), MAC_k(c) \rangle = \langle f_k(m, r), f_k^{-1}(f_k(m, r)) \rangle = \langle f_k(m, r), (m, r) \rangle$$

Thus, the message  $m$  is revealed in the output.

## 8.2 CCA-Secure Encryption

Until now, we have seen two notions of secure encryption: eavesdropping security and chosen-plaintext security. The third notion, called *chosen-ciphertext security*, is more stringent than the previous two. Specifically, the adversary's power is extended by giving it access to a decryption oracle as well as an encryption oracle. Apart from this, the definition is the same as for CPA-security. That is, the adversary is given access to an encryption and decryption oracle. At some stage, it outputs a pair  $m_0, m_1$  and it is then given an encryption  $c$  of one of the two messages. The adversary's aim is to guess if  $c$  is an encryption of  $m_0$  or  $m_1$ . We note that the adversary may continue using the decryption oracle after it receives  $c$ , and is allowed to ask for the decryption of any ciphertext other than  $c$ . (Otherwise, security clearly cannot be achieved.)

Chosen-ciphertext security seems very strong. However, it models attacks where an adversary may obtain *some* information about plaintexts from ciphertexts. For example, in real life, it is possible for an adversary to know whether a ciphertext is valid or invalid (for example, by seeing if a server outputs an error message). Furthermore, such information is sometimes enough to execute a successful attack on CPA-secure schemes. Another setting where chosen-ciphertext security may be needed is in the case that a server would behave differently, depending on what type of message it receives. If this behaviour is observable to an outside attacker, then it can learn something about the message that is encrypted by presenting the server with a ciphertext. Any encryption scheme that is secure against a chosen-ciphertext attack is secure in these settings. It is therefore often important to use schemes that achieve this level of security.

**The CCA indistinguishability game  $\text{Expt}_{\text{CCA}}$ .** A random key is chosen  $k \leftarrow K$  and the adversary  $\mathcal{A}$  is then given oracle access to the encryption and decryption algorithms  $E_k(\cdot)$  and  $D_k(\cdot)$ . At some stage, the algorithm  $\mathcal{A}$  outputs a pair of message  $m_0, m_1$ . A bit  $b \in_R \{0, 1\}$  is then randomly chosen and  $\mathcal{A}$  is given  $c = E_k(m_b)$ . Adversary  $\mathcal{A}$  continues to have oracle access to both  $E_k(\cdot)$  and  $D_k(\cdot)$ ; however only queries  $c' \neq c$  are answered by  $D_k(\cdot)$ . Finally,  $\mathcal{A}$  outputs a bit  $b'$ . We say that  $\mathcal{A}$  succeeded if  $b' = b$  and in such a case we write  $\text{Expt}_{\text{CCA}}(\mathcal{A}) = 1$ .

---

<sup>1</sup>We note that it is sometimes possible to use the same key for different goals; however, an explicit proof is needed for such cases.

**Definition 8.1** An encryption scheme  $E$  is  $(t, \epsilon)$ -indistinguishable under chosen-ciphertext attacks (CCA secure) if for every adversary  $\mathcal{A}$  that runs for at most  $t$  steps,

$$\Pr[\text{Expt}_{\text{CCA}}(\mathcal{A}) = 1] < \frac{1}{2} + \epsilon$$

We note that this definition is actually for that of *adaptive* chosen-ciphertext attacks, typically denoted CCA2.

**Achieving CCA-secure encryption.** In order to achieve CCA-security, we will construct an encryption scheme with the property that the adversary will not be able to obtain *any* valid ciphertext that was not generated by the legitimate parties. This will have the effect that the decryption oracle will be rendered useless. The scheme that we will use is exactly the *encrypt-then-authenticate* scheme discussed above, except that we will assume that the MAC scheme has *unique tags*. That is, the MAC algorithm is deterministic. (We note that different requirements can be added to the MAC so that the theorem will hold, but this suffices for us here.) We have the following theorem:

**Theorem 8.2** Let  $E$  be a CPA-secure encryption scheme and let MAC be a secure message authentication code with unique tags. Then,  $E'_{k_1, k_2}(m) = \langle c, \text{MAC}_{k_2}(c) \rangle$ , where  $c = E_{k_1}(m)$ , is a CCA-secure encryption scheme.

The idea behind the proof of this theorem is as follows. Since MAC is a secure message authentication code, we can assume that all queries to the decryption oracle return *invalid*, unless the queried ciphertext was previously obtained by the adversary querying the encryption oracle. Therefore, the security of  $E'$  is reduced to the CPA-security of  $E$  (because the decryption oracle is effectively useless).

In the above theorem statement, we have not include the parameters  $t$  and  $\epsilon$ . We leave the computation of these parameters and a proof of the theorem as *an advanced exercise*. (I will be happy to help any student who wishes to attempt this by him or herself.)

**Secure channels versus CCA-security.** We note that although we use the same construction for achieving CCA-security and combining privacy and encryption, the security goals in both cases are different. Namely, here we are not necessarily interested in obtaining authentication. Rather, we wish to ensure privacy even in a strong adversarial setting where the adversary is able to obtain some information about a plaintext from a given ciphertext.

This difference becomes much clearer in the public-key world (see later lectures). In particular, CCA-security can be achieved in a setting where only the *receiver* has a public key, whereas secure channels requires that both the *sender and receiver* have public keys.

### 8.3 Key Management

In this section, we discuss issues relating to the management of private keys. Until now, we have seen that the security goals of privacy and authentication can be achieved, *assuming that the communicating parties share a secret key*. The first question that must be dealt with here is how do the parties obtain such a shared secret key to begin with. In a closed environment (like the military, intelligence agencies etc.) it is possible for the parties to physically meet and share the key. However, this does not work in open systems like the Internet. This raises a problem that,

unfortunately, does not have a solution within the private-key world. Next week, we will begin studying the topic of public-key cryptography, which addresses this exact issue.

Assume for now, that it *is* possible for parties to somehow obtain initial secret keys. Even in this case, an important issue that must be dealt with is how to manage such keys. First, such keys must be kept in secure databases (so that adversaries cannot read them). Furthermore, they must be refreshed at reasonable intervals (so that if the security is at some stage compromised, this can be limited to the interval in question). These requirements mean that having a large number of different keys is very problematic. However, if  $n$  parties wish to communicate with each other, then each party needs to hold  $n - 1$  keys (and overall there need to be  $n(n - 1)/2$  different secret keys in the system). For  $n = 1,000$  this is already a huge amount. We will therefore briefly discuss how to reduce the number of keys needed.

**Key distribution centers and session-key generation.** The main idea is to have the parties interact via a key distribution center (KDC) in order to set up a session-key. More specifically, all parties share a secret key with the KDC (so each party holds one secret key and the KDC holds  $n$  secret keys). Then, in order for parties  $A$  and  $B$  to communicate, they ask for “help” from the KDC. At the conclusion of their interaction with the KDC and with each other, they will obtain a shared secret key that is used only for the current interaction (or session). This key is called a **session key**.

The basic idea can be thought of as follows. If  $A$  wishes to communicate with  $B$ , it sends a request  $(A, B, \text{key})$  to the KDC. The KDC then chooses a random session-key  $sk$  and sends  $(A, B, E_{k_A}(sk))$  to  $A$  and  $(A, B, E_{k_B}(sk))$  to  $B$ , where  $k_A$  and  $k_B$  are  $A$  and  $B$ 's respective secret keys that are shared with the KDC. Notice that only  $A$  and  $B$  can decrypt  $sk$  and so they can use this as a secret key to communicate.

We stress that, although this sounds very simple, there are a number of delicate issues that arise. Due to lack of time, we will not be able to present these schemes. However, *it is highly recommended to at least read pages 127–132 in [36]*. Note that the Kerberos system, presented on page 132 is widely implemented (including as the basis of the shared-secret logon protocol in Windows 2000 and later). For more information on Kerberos, see [23].

**Choosing keys.** We stress that keys must be chosen at random. Different methods exist for generating random coin tosses. First, it is possible to purchase computers today that have hardware-based random bit generators. Other methods include users typing random noise or moving the mouse around at random. We warn that using a memorable password as a key is very dangerous and should be avoided wherever possible.



## Lecture 9

# Public-Key (Asymmetric) Cryptography

### 9.1 Motivation

One of the central aims of public-key cryptography is to solve the problem of key agreement or distribution. In the private-key world, encryption and authentication can only be carried out if the communication parties can somehow obtain the same, uniformly distributed keys. There are many applications in which such key distribution is feasible; however, there are also many in which it is not. For example, almost all electronic commerce between vendors and customers over the Internet would not be possible if the customer had to physically obtain a secret key from the vendor. This issue led Diffie and Hellman to essentially revolutionize the field of cryptography with the invention of *public-key cryptography* in 1976 [18]. We strongly recommend reading the original paper of Diffie and Hellman (it can be found online easily). The basic idea is that some operations are inherently asymmetric. In particular, everyone should be able to send me a letter, but only I should be able to open and read my mail. Furthermore, there are mathematical operations that are also asymmetric. For example, it is easy to multiply two large (e.g., 1024 bit) prime numbers together, but no polynomial-time algorithm is known for inverting this process and finding the prime factors of a large (e.g., 1024 bit) number  $n$  that is a multiple of two primes.

In a public-key encryption scheme, the encryption and decryption keys are not the same. Rather, the encryption key is made public, enabling everyone to encrypt messages. However, the crucial point is that knowing this encryption key does not make it possible to decrypt ciphertexts. This capability is left only to the legitimate receiver who has the decryption key. This makes it possible for parties to publish their public-keys so that everyone can communicate privately with them. We stress that this distribution is also not trivial, and we will discuss later how this should be done.

Other important public-key tasks that we will consider are *digital signatures* and *key agreement*. In the setting of digital signatures, there are distinct signing and verification keys with the property that anyone can verify the validity of a signature, but only the legitimate signer can sign. (This is in contrast to message authentication codes where the application of the code and its verification used the same key.) Likewise, the public-key version of key agreement enables parties to use public/private key pairs in order to agree on a shared secret key.

---

\* Lecture notes for an undergraduate course in cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

**Advantages and disadvantages of public-key cryptography.** Some of the advantages of using public-key cryptography are that key distribution becomes much easier (but again, as we have mentioned, still remains non-trivial), less keys need to be stored (it suffices for every party to have a single key), and other parties' public keys may be kept in public, but reliable, storage (that is, each party needs only to keep its own personal private key in secure storage).

We also note one disadvantage of public-key cryptography: all known schemes are *far slower* than their equivalent in the private-key world. For this reason, “hybrid” schemes are often used that combine between public and private-key encryption (e.g., first a symmetric key is encrypted using the public-key, and then the message is encrypted with the symmetric key).

## 9.2 Public-Key Problems and Mathematical Background

As we have mentioned, the technical heart of public-key cryptography is the fact that some mathematical operations seem to be inherently asymmetric. More specifically, there exist functions that can be efficiently computed, but for which inversion seems to be hard (i.e., no polynomial-time inversion algorithms are known to exist). We will present some of the most famous of these problems here. Our focus here is not on the mathematical technicalities in these functions and we will therefore skip many important details. These details can be found in many cryptographic texts; in particular, we recommend [25] and [35]. We also assume knowledge of basic algebra and number theory like modular arithmetic.<sup>1</sup>

**The Discrete-Logarithm problem (DL).** Let  $p$  be an  $n$ -bit prime, and let  $g$  be a generator of the cyclic group  $G$  defined over  $\{1, \dots, p-1\}$  with the group operation of multiplication. (That is,  $g$  is a value such that for every  $a \in \{1, \dots, p-1\}$  there exists a unique value  $b \in \{1, \dots, p-1\}$  such that  $a = g^b \bmod p$ . We note that such a generator always exists and it can be efficiently found given the factorization of  $p-1$ .)

The discrete-log problem is: given a prime  $p$ , a generator  $g$  and a random value  $a \in G$ , find a value  $b \in G$  such that  $g^b = a \bmod p$  ( $b$  is called the discrete-log of  $a$  in base  $g$ ). The discrete-log assumption is that for random  $n$ -bit primes  $p$  (for large enough  $n$ ), it is *infeasible* to solve the discrete-log problem (i.e., it is assumed that there does not exist a polynomial-time algorithm that solves this problem). We note that the best known algorithm today runs in approximately  $O\left(e^{\sqrt[3]{n \log n}}\right)$  time. So, taking  $t$  to be significantly less than this time, we can set  $\epsilon$  to be somewhere in the region of  $2^{-n}$  for a concrete hardness assumption. Note that  $n$  should be greater than or equal to 1024, and for high-security applications it is wise to set  $n = 2048$ . In order to parameterize the assumption, we say that the DL problem is  $(t, \epsilon)$ -hard if every algorithm running for at most  $t$  steps can succeed in solving the problem with probability at most  $\epsilon$ . This same parameterization can be used for all the problems described below.

Note that it is *easy* to compute  $a = g^b \bmod p$  given the tuple  $(p, g, b)$ . The *hard part* is to invert this operation and compute  $b$  from  $(p, g, a)$ . We also note that  $a$  does not “hide all information” about  $b$ . In fact, it is possible to efficiently compute the least significant bit of  $b$  from  $(p, g, a)$ . However, it can be shown that computing the most significant bit of  $b$  from  $(p, g, a)$  is “as hard as” computing  $b$  itself.

---

<sup>1</sup>Just to be safe, we say that  $x \equiv y \bmod n$  iff  $x$  and  $y$  have the same remainder when divided by  $n$ . In other words,  $x \equiv y \bmod n$  iff  $n$  divides  $x - y$ .

**The Computational and Decisional Diffie-Hellman problems [18].** The Diffie-Hellman problem is based on the hardness of the computing the discrete log. The computational Diffie-Hellman (CDH) assumption is the following:

Given a random  $n$ -bit prime  $p$  (for large enough  $n$ ), a generator  $g$  of  $G$  and a pair  $(g^a \bmod p, g^b \bmod p)$  where  $a, b \in_R G$ , it is hard to compute  $g^{ab} \bmod p$ .

The only way that we know how to compute  $g^{ab} \bmod p$  today is to first compute the discrete log of  $g^a \bmod p$  thereby obtaining  $a$ , and then compute  $g^{ab} \bmod p = (g^b \bmod p)^a \bmod p$ . This shows that the Discrete-Log problem is at least as hard as the Diffie-Hellman problem (i.e., any solution to DL also solves CDH); the reverse direction is not known.

The CDH assumption is related to the difficulty of computing  $g^{ab}$ . The decisional Diffie-Hellman (DDH) assumption relates to the difficulty of knowing if a value  $\alpha$  equals  $g^{ab}$  or equals  $g^c$  for some random  $c$ . That is:

For large enough random  $n$ -bit primes  $p$ , it is hard to distinguish tuples of the form  $(p, g, g^a \bmod p, g^b \bmod p, g^{ab} \bmod p)$  from tuples of the form  $(p, g, g^a \bmod p, g^b \bmod p, g^c \bmod p)$  where  $a, b, c \in_R G$ .

The DDH assumption seems significantly stronger than the CDH and DL assumptions (as above, a solution to DL or CDH clearly solves DDH, but the reverse is unknown). We note that there are groups where DDH is known to be easy but CDH is still assumed to be hard. See [10] for a survey on the DDH assumption.

**Factoring, Rabin and RSA.** Let  $p$  and  $q$  be large random  $n$ -bit primes. Then, computing  $N = pq$  is easy, but the best known algorithm for computing  $p$  and  $q$  from  $N$  runs in time approximately  $O\left(e^{\sqrt[3]{n \log n}}\right)$ . The Rabin function [30] can be informally defined by  $f(x) = x^2 \bmod N$ . The important point about the Rabin function is that it is *equivalent* to factoring. That is, it is possible to invert the function if and only if it is possible to factor  $N$  into its prime factors  $p$  and  $q$ .

The RSA function [32] is defined as follows. Let  $p, q$  and  $N$  be as above. Euler's function  $\varphi(N)$  is defined as the number of values  $b$  in  $\{0, \dots, N-1\}$  that are co-prime to  $N$  (i.e.,  $\gcd(b, N) = 1$ ). Then, for  $N = pq$ , we have that  $\varphi(N) = (p-1)(q-1)$ . Now, a random value  $e$  is chosen between 1 and  $\varphi(N)$  such that  $e$  is co-prime to  $\varphi(N)$  (i.e.,  $\gcd(e, \varphi(N)) = 1$ ). Finally, the inverse  $d$  of  $e$  modulo  $N$  is computed: i.e.,  $d$  is such that  $e \cdot d = 1 \bmod \varphi(N)$ . The RSA function is  $f_{e,N}(x) = x^e \bmod N$ . Note that for  $y = f_{e,N}(x)$ , it is possible to compute the inverse by

$$y^d \equiv x^{ed} \equiv x^{1+c\varphi(N)} \equiv x \cdot \left(x^{\varphi(N)}\right)^c \equiv x \bmod N$$

where the last equivalence is due to Euler's generalization of Fermat's little theorem that states that if  $\gcd(a, N) = 1$  then  $a^{\varphi(N)} \equiv 1 \bmod N$ .

### 9.3 Diffie-Hellman Key Agreement [18]

The key agreement problem is how to establish a secret key over a public channel. In order to illustrate this, consider the following protocol that uses two-lock boxes (i.e., boxes that can be locked independently with two padlocks) and the postal service. Alice takes a two-lock box, places the secret key  $k$  inside the box, locks it with her padlock and sends it to Bob. Bob receives the box, places his padlock on the box and returns it to Alice. Alice removes her padlock and sends it once again to Bob. Bob now removes his padlock and obtains the key  $k$ . Notice that only Alice

and Bob can obtain the key  $k$ , as long as no party can intercept the boxes and impersonate Bob to Alice. This idea is the basis of the Diffie-Hellman key agreement protocol.

Initially, the parties have an agreed-upon prime  $p$  and generator  $g$  (this information is public). The protocol works as follows:

1. Alice chooses a random  $a \in G$ , computes  $x = g^a \bmod p$  and sends  $x$  to Bob.
2. Bob chooses a random  $b \in G$ , computes  $y = g^b \bmod p$  and sends  $y$  to Alice.
3. Alice and Bob both compute  $g^{ab} \bmod p$  and set this to be their secret key.

Note that assuming that no party modifies their message en route, the security of this protocol follows from the decisional Diffie-Hellman assumption. This is due to the fact that given  $g^a \bmod p$  and  $g^b \bmod p$ , the DDH assumption states that the secret key  $g^{ab} \bmod p$  is indistinguishable from  $g^c \bmod p$  where  $c \in_R G$  (and so this latter key truly is random).

It is important to stress here that if the communication channel between Alice and Bob is not reliable (i.e., it is possible for an adversary to modify messages being sent), then it is possible to completely break the above protocol. Namely, the adversary can simply compute  $y' = g^{b'} \bmod p$  and  $x' = g^{a'} \bmod p$  and replace Alice's message to Bob with  $x'$  and Bob's reply to Alice with  $y'$ . The result is that Alice will conclude with a key  $g^{a'b'} \bmod p$ , Bob will conclude with a key  $g^{a'b} \bmod p$ , and the adversary will know them both. This is called a *man-in-the-middle attack* and will be discussed in more detail later.

**Diffie-Hellman with authenticated keys.** Consider the case that there is a reliable and trusted database that holds pairs of the form  $(\text{Alice}, g^a \bmod p)$ , where it is verified that  $g^a \bmod p$  is indeed Alice's key. Furthermore, it is assumed that it is possible to obtain values from the database without adversarial interference. Then, if Bob wishes to communicate with Alice it can look up Alice's key in the database and then use the key  $k = g^{ab} \bmod p$  where  $(\text{Bob}, g^b \bmod p)$  is its own entry in the database. Since these keys are verified, or "authenticated", this ensures secure communication between Alice and Bob. We note that in practice, Alice and Bob will not actually use  $k$ , but will rather generate a fresh *session key* every time that they wish to communicate.

## Lecture 10

# Public-Key (Asymmetric) Encryption

As we have mentioned in the previous lecture, the main idea behind public-key encryption is to split the key into two distinct parts: the *public encryption key* and the *secret decryption key*. The security of the encryption scheme is formalized in an almost identical way as in the private-key world, except that the adversary is also given the public encryption key. Notice that this means that by default, a chosen-plaintext attack can always be carried out (since the adversary knows the encryption key, it can encrypt any message that it wishes).

### 10.1 Definition of Security

**The syntax.** An encryption scheme is a tuple of algorithms  $(G, E, D)$  such that  $G$  is a probabilistic algorithm that generates keys,  $E$  is the encryption algorithm and  $D$  is the decryption algorithm. Unlike the case of private-key encryption, we cannot assume that the keys are just uniformly distributed strings. Rather,  $G$  receives a parameter  $1^n$  (that indicates in unary the length of the keys to be generated), and outputs a pair of keys  $(pk, sk)$ . The encryption algorithm  $E$  receives the public-key  $pk$  and a plaintext message, and generates a ciphertext. The decryption algorithm  $D$  receives the secret-key  $sk$  and a ciphertext, and outputs a plaintext. We require that, except with negligible probability over the choice of the keys, it holds that for every  $x$ ,  $D_{sk}(E_{pk}(x)) = x$ .

For the sake of completeness, we repeat the definition of security under chosen-plaintext attack:

**The Public-Key CPA indistinguishability game**  $\text{Expt}_{\text{CPA}}^{\text{Pub}}$ . A random pair of keys is chosen  $(pk, sk) \leftarrow G(1^n)$  and the adversary  $\mathcal{A}$  is given the public-key  $pk$ . Next, the algorithm  $\mathcal{A}$  outputs a pair of message  $m_0, m_1$  of the same length. A bit  $b \in_R \{0, 1\}$  is then randomly chosen and  $\mathcal{A}$  is given  $c = E_{pk}(m_b)$  (this value is called the challenge ciphertext). Following this,  $\mathcal{A}$  outputs a bit  $b'$ . We say that  $\mathcal{A}$  succeeded if  $b' = b$  and in such a case we write  $\text{Expt}_{\text{CPA}}^{\text{Pub}}(\mathcal{A}) = 1$ .

**Definition 10.1** A public-key encryption scheme  $E$  is  $(t, \epsilon)$ -indistinguishable under chosen-plaintext attacks (CPA secure) if for every adversary  $\mathcal{A}$  that runs for at most  $t$  steps,

$$\Pr[\text{Expt}_{\text{CPA}}^{\text{Pub}}(\mathcal{A}) = 1] < \frac{1}{2} + \epsilon$$

As for private-key encryption, a secure public-key encryption scheme *must* be probabilistic. Otherwise, it is possible for the adversary to just compute  $E_{pk}(m_0)$  and  $E_{pk}(m_1)$  by itself and compare

---

\* Lecture notes for an undergraduate course in cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

the result to the challenge ciphertext. Note that unlike in the private-key setting, there is no need for any oracle in order to carry out this attack. The analogous definition of chosen-ciphertext security for public-key schemes is left for an exercise.

## 10.2 The El-Gamal Encryption Scheme [19]

This encryption scheme is based on the Decisional Diffie-Hellman assumption (DDH) and is a direct extension of the Diffie-Hellman key-exchange protocol.

### The Scheme:

1. *Key Generation*  $G$ : choose a group  $\mathcal{G}$  in which it is assumed that the DDH problem is hard, and choose a generator  $g$  for  $\mathcal{G}$ . Choose a value  $a \in_R \mathcal{G}$ , compute  $h = g^a$  and output the public-key  $pk = (\langle \mathcal{G} \rangle, g, h)$ , where  $\langle \mathcal{G} \rangle$  is a (short) description of  $\mathcal{G}$ . Output the secret-key  $sk = (\langle \mathcal{G} \rangle, g, a)$ . (Note that all operations here and below are in the group  $\mathcal{G}$ .)
2. *Encryption*: Given  $pk = (\langle \mathcal{G} \rangle, g, h)$  and a message  $m \in \mathcal{G}$  (i.e., the message  $m$  is encoded as an element of  $\mathcal{G}$ ), choose  $r \in_R \mathcal{G}$ , compute  $v = g^r$ ,  $w = h^r \cdot m$  and output  $c = (v, w)$ .
3. *Decryption*: Given  $sk = (\langle \mathcal{G} \rangle, g, a)$  and  $c = (v, w)$ , compute  $m = w/v^a$ .

First, notice that decryption always works. That is, for  $(v, w)$  that is correctly computed, it holds that  $v^a = g^{ra} = (g^a)^r = h^r$ . Therefore,  $w/v^a = (h^r \cdot m)/h^r = m$ . As we have mentioned, this encryption scheme is a direct extension of the Diffie-Hellman key-exchange protocol. In order to see this, set  $v = g^r$  to be “ $A$ ’s message” and  $h = g^a$  to be “ $B$ ’s message” in the Diffie-Hellman protocol. Then, the key that is generated from the protocol equals  $g^{ar} = h^r$ , which is exactly the “secret key” used by the sender to mask the message  $m$ . Finally, we remark that encryption here is probabilistic, and two encryptions of the same message look completely different (because the masking value  $r$  is completely different every time).

We now prove that the security of the El-Gamal encryption scheme reduces directly to the DDH assumption.

**Theorem 10.2** *Assume that the DDH problem is  $(t, \epsilon)$ -hard. Then, the El-Gamal encryption scheme is  $(O(t), \epsilon)$ -indistinguishable under a chosen-plaintext attack.*

**Proof:** Assume by contradiction that there exists an adversary  $\mathcal{A}$  that runs in time  $O(t)$  and succeeds in the public-key CPA-experiment for El-Gamal with probability greater than or equal to  $1/2 + \epsilon$ . We construct a distinguisher  $D$  for the DDH problem.

The distinguisher  $D$  is given a tuple  $(\langle \mathcal{G} \rangle, g, g^a, g^b, \alpha)$  where  $\alpha = g^{ab}$  if the tuple is of the Diffie-Hellman type, and  $\alpha = g^c$  otherwise (recall,  $a, b, c \in_R \mathcal{G}$ ).  $D$  sets  $pk = (\langle \mathcal{G} \rangle, g, h)$  where  $h = g^a$  (i.e.,  $h$  is the third term in  $D$ ’s input tuple). Then,  $D$  invokes the adversary  $\mathcal{A}$  with the key  $pk$ . When  $\mathcal{A}$  outputs two messages  $(m_0, m_1)$ , the distinguisher  $D$  chooses  $\beta \in_R \{0, 1\}$ , sets  $v = g^b$  (i.e.,  $v$  is the fourth term in  $D$ ’s input tuple) and  $w = \alpha \cdot m_\beta$ .  $D$  then hands  $\mathcal{A}$  the ciphertext  $c = (v, w)$ . If  $\mathcal{A}$  outputs  $\beta' = \beta$ , then  $D$  outputs 1 (i.e., the input was a DH tuple); otherwise,  $D$  outputs 0 (i.e., the input was not a DH tuple).

Now, if  $D$  received a DH tuple and so  $\alpha = g^{ab} = h^b$ , then the ciphertext generated by  $D$  is correct. That is,  $v = g^b$  and  $w = h^b \cdot m$  (since  $b \in_R \mathcal{G}$ , this is the same as ordinary El-Gamal encryption). Therefore, by the contradicting assumption,  $\mathcal{A}$  outputs  $\beta' = \beta$  with probability that is greater than or equal to  $1/2 + \epsilon$ .

In contrast, if  $D$  received a non-DH tuple and so  $\alpha = g^c$  for  $c \in_R \mathcal{G}$ , then  $w = g^c \cdot m_\beta$ . Therefore, the value  $\beta$  is *perfectly hidden* by  $w$ . (Notice that  $w$  is an encryption of  $m_0$  if and only if  $c$  is such that  $g^c = w/m_0$ , and  $w$  is an encryption of  $m_1$  if and only if  $c$  is such that  $g^c = w/m_1$ . Since  $g$  is a generator, and  $c \in_R \mathcal{G}$ , both of these possible choices of  $c$  occur with the same probability. Therefore, we have “perfect secrecy” here.) Thus, by an information-theoretic argument, it holds that  $\mathcal{A}$  outputs  $\beta' = \beta$  with probability at most  $1/2$ .

We conclude that:

$$\left| \Pr[D(\langle \mathcal{G} \rangle, g, g^a, g^b, g^{ab}) = 1] - \Pr[D(\langle \mathcal{G} \rangle, g, g^a, g^b, g^c) = 1] \right| \geq \frac{1}{2} + \epsilon - \frac{1}{2} = \epsilon$$

and so  $D$  solves the DDH problem with probability greater than or equal to  $\epsilon$ . Assuming that  $t$  is significantly larger than the overhead of  $D$  (which essentially involves one multiplication) and setting  $\mathcal{A}$ 's running-time to  $t/2$ , we have that  $D$  runs in time  $t$ . Thus,  $D$  contradicts the  $(t, \epsilon)$  hardness assumption on the DDH problem. This completes the proof. ■

**CCA security.** We remark that the El-Gamal encryption scheme is *not* CCA-secure. Namely, upon receiving the challenge ciphertext  $c = (v = g^r, w = h^r m_b)$ , it is possible for the adversary to request a decryption of  $(v, 2 \cdot w)$ . The plaintext that is returned equals  $2 \cdot m_b$ , and so it is easy to tell if the ciphertext  $m_0$  or  $m_1$  was encrypted. This also shows that the El-Gamal encryption scheme is easily malleable (see Exercise 4).

A CCA-secure encryption scheme that is based on the DDH assumption was presented by Cramer and Shoup in [14]. This is the most efficient CCA-secure encryption scheme that is known to date.

## 10.3 RSA Encryption

The RSA encryption scheme is the most popular algorithm used in practice. Last week, we saw how the basic RSA function works. To recap:

### The Plain RSA Scheme:

1. *Key Generation  $G$* : upon input  $1^n$ , choose two random primes of length  $n/2$  and compute  $N = pq$ . Choose  $e$  such that  $\gcd(e, \varphi(N)) = 1$ . Find  $d$  such that  $ed \equiv 1 \pmod{\varphi(N)}$ . Output  $pk = (e, N)$  and  $sk = (d, N)$ .
2. *Encryption*: Given  $pk = (e, N)$  and a message  $m \in \{1, \dots, N - 1\}$ , compute and output  $c = x^e \pmod{N}$ .
3. *Decryption*: Given  $sk = (d, N)$  and  $c$ , compute  $m = c^d \pmod{N}$ .

Recall that last week we saw that decryption works. The first important observation about the above-described scheme is the following:

**Claim 10.3** *Plain RSA is not indistinguishable under a CPA-attack.*

This trivially follows from the fact that encryption is *deterministic*.

**RSA with random padding.** The typical way of encrypting with RSA is to add “enough” random padding after the message. The PKCS #1 standard (version 1.5) employs the following method:

1. *Key generation:* as with plain RSA.
2. *Encryption:* Let  $m$  be the message to be encrypted. The message  $m$  is padded as follows. The first (i.e., most significant) two bytes equal 0002 (in hexadecimal), the next series of *at least* 8 bytes are random, the next byte is 00, and finally the message is written in the least significant portion. This padded value is then encrypted as in plain RSA.
3. *Decryption:* as with plain RSA, followed by the removal of the padding.

We note that there is no security analysis of this this padding method. Nevertheless, it is widely used and believed to be secure *under a chosen-plaintext attack*. Importantly, it was shown by Bleichenbacher in [8] that the PKCS #1 standard (v. 1.5) is *not secure under a chosen-ciphertext attack*, or even under an attack where the only information provided is whether the padding of the ciphertext is correctly formatted. This attack by Bleichenbacher is very important in that it demonstrates that in many setting, CPA-security does not suffice.

In order to achieve CCA-security for RSA encryption, a padding method called OAEP (Optimal Asymmetric Encryption Padding) is used [6]. The security of the scheme relies on the random oracle methodology (see Lecture 6) and is therefore heuristic. Due to lack of time, we do not describe the scheme. We remark that the new PKCS #1 standard (version 2.0) uses OAEP encryption. In practice, if RSA encryption is to be used, this seems to be the best methodology available today.

**Provably secure RSA.** There exist constructions of secure public-key encryption that can be provably reduced to the RSA problem. Unfortunately, these schemes are less efficient than heuristic ones, and are therefore not used. We will learn more about provably-secure constructions in the course “Foundations of Cryptography” (89-856) in the next semester.

**Implementation issues.** RSA encryption and decryption is relatively slow. However, a number of optimizations exist that can significantly speed up the computation. For example, using the Chinese Remainder Theorem, it is possible to carry out most of the computation modulo  $p$  and  $q$ , and only then combine the results to obtain the computation modulo  $N = pq$ .

## 10.4 Security of RSA

The RSA function (and encryption scheme) has withstood 25 years of scrutiny and the only known way of “breaking” RSA (when encryption is carried out properly) is by factoring the modulus. Since we believe that factoring is hard, we may conclude that RSA is secure. However, we stress that there is no proof of this fact, and it is unknown whether or not the RSA assumption can be proven under the assumption that factoring is hard. Nevertheless, there are some tasks related to breaking RSA that can be proven to be equivalent to the problem of factoring.

**Computing  $\varphi(N)$  from  $(e, N)$ .** Assume that we are given  $N$  and  $\varphi(N)$ . Then, it follows that we have two equations with two unknowns  $p$  and  $q$ :

$$\begin{aligned} N &= pq \\ \varphi(N) &= (p-1)(q-1) = N - p - q + 1 \end{aligned}$$

These equations can be solved by substituting  $q = N/p$  into the second equation, yielding:

$$\varphi(N) - N + p + \frac{N}{p} - 1 = 0$$

Multiplying both sides by  $p$  and re-arranging the terms, we obtain a quadratic equation in the unknown value  $p$ :

$$0 = p^2 - (N - \varphi(N) + 1)p + N = 0$$

The two roots of the above polynomial will be the two factors of  $N$ . Therefore, using the standard equation for finding the roots of a quadratic polynomial, we can obtain the factors  $p$  and  $q$  of the modulus  $N$ . This demonstrates that finding  $\varphi(N)$  is no easier than factoring  $N$  (in particular, if one can find  $\varphi(N)$ , then using the above algorithm it is possible to go one step further and find  $p$  and  $q$ ).

**Private-key recovery and factoring.** Next, we will show that the problem of finding the RSA secret-key  $d$  given the public-key  $(e, N)$  is *equivalent* to the problem of factoring  $N$  into its prime factors. We stress that this should not be interpreted as meaning that RSA is secure as long as factoring is hard. Rather, it just means that in such a case, it is hard to recover the secret-key. This is a very different statement, because it may be possible to learn information about a plaintext (or even fully decrypt) without ever knowing the secret-key.

**Fact 10.4** *Let  $(N, e)$  be an RSA public key. There exists an efficient algorithm that receives the public and private keys  $(e, N)$  and  $d$ , and factors the modulus  $N$ . In addition, there exists an efficient algorithm that receives  $e$  and the factorization of  $N$ , and outputs  $d$ .*

**Proof:** We will only present the proof of a simplified case where  $e = 3$ . That is, we show that given  $(N, e, d)$  where  $e = 3$  it is possible to efficiently find the prime factors  $p$  and  $q$  of  $N$ . Recall that  $N = pq$  and  $ed \equiv 1 \pmod{\varphi(N)}$  where  $\varphi(N) = (p-1)(q-1)$ . Therefore,  $ed - 1 = c(p-1)(q-1)$  for some constant  $c$ . Now, since  $1 \leq e \leq 3$  and  $1 \leq d \leq (p-1)(q-1)$ , we have that  $1 \leq c \leq 3$ . Assume that we know the exact value of  $c$  (we can just try for each of the 3 possibilities). We then have the following equation:  $p + q = N + 1 - (ed - 1)/c$ ; denote  $s = N + 1 - (ed - 1)/c$ . Define the following polynomial  $f(x) = (x-p)(x-q) = x^2 - sx + N$ . Notice that  $s$  and  $N$  are known to us, and therefore we can fully define this polynomial. Furthermore,  $p$  and  $q$  are roots of the polynomial, so finding the roots gives us the factors of  $N$ . (Of course, finding roots is straightforward, using the standard equation.) The general case (for any  $e$ ) is a bit more involved, and we will not present it here.

The “in addition” part of the fact follows immediately from the fact that given  $p$  and  $q$ , simply compute  $\varphi(N) = (p-1)(q-1)$  and find  $e^{-1} \pmod{\varphi(N)}$ . ■

## 10.5 Hybrid Encryption

Due to the fact that public-key encryption is computationally expensive, when long messages are to be encrypted, a hybrid between public and private key encryption is used. That is, first a random symmetric key  $K$  is encrypted using a public-key encryption scheme. Then, the long message is encrypted using a highly efficient private-key encryption scheme, with key  $K$ .



# Lecture 11

## Attacks on RSA

In this lecture, we will describe a number of attacks on RSA that are known. All of these attacks can easily be avoided. However, studying them increases our understanding of the RSA algorithm and its security. It also demonstrates that it is not a good idea to implement RSA yourself without first understanding all of the possible issues involved. In addition, if you do implement it yourself, take care to follow the exact specification. We note that we cover only some of the known attacks. We refer the interested student to [9] for a survey of attacks on RSA.

### 11.1 Private and Public-Key Reversal

Notice that in RSA encryption is fast (because one can choose  $e$  in any way desired, and in particular such that  $e$  is small), but decryption is slow (because  $d$  must just be the inverse of  $e$  modulo  $\varphi(N)$ ). In the case of a server carrying out many decryptions, we would prefer the reverse. One may be tempted to therefore first take  $d$  to be *small*, and then find  $e$  such that  $e \cdot d \equiv 1 \pmod{\varphi(N)}$ . and such that  $d$  is *small* (thereby making decryption faster).

The above “optimization” makes it trivial to recover  $d$  given the public-key  $(e, N)$ . In particular, given that it is known that  $d$  is small (e.g.,  $d \leq 65,537$ ), it is possible to simply try all possible  $d$ 's, and check for each one whether or not it is correct.

### 11.2 Textbook RSA

We have already shown that plain RSA is not secure, by the fact that it is deterministic. One may be tempted to claim that indistinguishability, as we have defined it, is necessary for encrypting regular messages that may not have inherent randomness. However, if RSA is used for encrypting a random symmetric key  $K$  (as in hybrid encryption; see Section 10.5), then plain RSA should suffice. Notice that in such a case, a different  $K$  is used for every encryption. Furthermore, since the key-space is large, it is not possible to decrypt by computing  $K^e \pmod N$  for all possible  $K$ 's and comparing the result to the ciphertext.

We now show that the above claim is false. In particular, we show that with good probability, a 64-bit encrypted key can be “extracted” from an RSA ciphertext in time approximately  $2^{40}$ . Let  $K \in_R \{0, 1\}^{64}$  (i.e.,  $K \approx 2^{64}$ ) and let  $c = K^e \pmod N$ . Suppose now that  $K = K_1 \cdot K_2$  where

---

\* Lecture notes for an undergraduate course in cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005. We note that this lecture was skipped in 2004-05; its current format is therefore tentative.

$K_1, K_2 < 2^{34}$ ; this holds about 18% of the time (to see why this holds, and a full description of this attack, see [11]). If this holds, then we have that  $c/K_1^e = K_2^e \pmod N$ .

Now, build a table  $(1, c/1^e), (2, c/2^e), (3, c/3^e), \dots, (2^{34}, c/2^{34e})$  containing  $2^{34}$  pairs and sort it by the second element. Then, for each possible value of  $K_2$ , check if  $K_2^e$  is in the table (i.e., is a second element in a pair). If yes, then we have found  $K_1$  and  $K_2$ . Specifically, we have  $K_2$  and a pair  $(K_1, K_2^e) = (K_1, c/K_1^e)$ . The time that it takes to carry out this computation and find  $K = K_1 \cdot K_2$  is  $2^{34} + 2^{34}34 \approx 2^{40}$ , which is much smaller than  $2^{64}$  (and is small enough to be feasible).

### 11.3 Common Modulus Attack

This attack is due to a negligent use of RSA. Specifically, Assume that a number of parties wish to all use the same RSA modulus  $N = pq$ , where every party has a different encryption and decryption exponent  $e$  and  $d$ . That is, let  $P_1, \dots, P_k$  be  $k$  different parties and let  $(e_i, N)$  and  $(d_i, N)$  be the respective public and private keys for party  $P_i$ .

The first observation here is that party  $P_i$  can decrypt the messages of party  $P_j$  for all  $j$ . This follows from Fact 10.4. Specifically, given  $(e_i, N)$  and  $d_i$ , party  $P_i$  can factor  $N$  into  $p$  and  $q$ . Then, given  $p$  and  $q$ , it can compute  $\varphi(N)$  and then  $d_j$  for all  $j$  (it can do this because  $e_j$  is public and known and all that is needed is to compute  $d_j = e_j^{-1} \pmod{\varphi(N)}$ ).

**Sharing a modulus between trusting parties.** Above we have shown that all parties sharing the same modulus can decrypt each other's messages. This still leaves the possibility that a shared modulus can be used in the case of mutually trusting parties. (Of course, in such a case, they may as well just use the same public key. Here we will see what happens if they do not.)

Let  $P_1$  and  $P_2$  be two parties with respective public keys  $(e_1, N)$  and  $(e_2, N)$ , and assume that a party sends the same encrypted message to both parties. That is, the adversary sees  $c_1 = m^{e_1} \pmod N$  and  $c_2 = m^{e_2} \pmod N$ . Given these ciphertexts, the adversary can compute

$$\begin{aligned} t_1 &= e_1^{-1} \pmod{e_2} \\ t_2 &= (t_1 e_1 - 1)/e_2 \end{aligned}$$

(The inverse of  $e_1$  modulo  $e_2$  can be found efficiently using the extended Euclidean algorithm.) Now, given these values it is possible to compute  $m$  as follows:

$$\begin{aligned} c_1^{t_1} \cdot c_2^{-t_2} &= m^{e_1 t_1} \cdot m^{-e_2 t_2} \\ &= m^{1+e_2 t_2} m^{-e_2 t_2} \\ &= m^{1+e_2 t_2 - e_2 t_2} \\ &= m \end{aligned}$$

It is therefore “more secure” for all parties to use the same public-key, than for them to use a shared modulus.

### 11.4 Simplified Broadcast Attack

Assume a party wishes to encrypt a message  $m$  and send it to a number of parties  $P_1, \dots, P_k$  with respective public keys  $(e_1, N_1), \dots, (e_k, N_k)$ . Assume also that  $m < N_i$  for every  $i$  (i.e.,  $m$  is a

message that fits into a single application of RSA for encryption). A naive way of carrying out this encryption is to simply compute  $c_i = m^{e_i} \bmod N_i$  for every  $i$ , and then send out  $c_1, \dots, c_k$ .

Now, assume that there are at least 3 public exponents  $e_i$  that are equal to 3; without loss of generality, let them be  $e_1 = e_2 = e_3 = 3$ . In this case, it is easy to recover  $m$  from  $c_1, c_2$  and  $c_3$ . Specifically, an eavesdropper is given  $c_1 = m^3 \bmod N_1$ ,  $c_2 = m^3 \bmod N_2$  and  $c_3 = m^3 \bmod N_3$ . Then, assuming that all of  $N_1, N_2$  and  $N_3$  are relatively prime (otherwise, it is possible to factor them), it is possible to compute  $c = m^3 \bmod N_1 N_2 N_3$ . (This computation is easy to carry out using the Chinese remainder theorem.) Since  $m < N_i$  for all  $i$ , we have that  $c = m^3$  over the integers (and not just  $\bmod N_1 N_2 N_3$ ). Therefore, the cube root of  $c$  (over the integers) returns the original message  $m$ .

This attack provides another reason why randomized encryption must be used (if each message was padded with a different random pad, this attack would not be successful).

## 11.5 Timing Attacks

In order to describe this attack, we first need to explain how modular exponentiation is carried out. In particular, we need to compute  $c^d \bmod N$  in order to decrypt. A naive algorithm that simply runs in a loop  $d$  times would not finish in our lifetime (because it iterates in the order of  $2^{1024}$  times). Thus, exponentiation is carried out in a different way. In order to get an idea of how the algorithm works, think about computing  $c^{2^t} \bmod N$  for some  $t$ . In this case, it is possible to use the following algorithm, that is called *repeated squaring*:

1. Set  $x \leftarrow c$
2. For  $i = 1$  to  $t$ , compute  $x \leftarrow x^2$
3. Output  $x$

In the general case, we use a similar idea. Let  $d = d_t d_{t-1} \dots d_0$  be the binary representation of  $d$  (e.g.,  $t = 1024$ ). Since  $d = 2^0 d_0 + 2^1 d_1 + \dots + 2^t d_t$ , we have that

$$c^d \bmod N = \prod_{i=0}^n c^{2^i d_i} \bmod N$$

This leads us to the following algorithm:

1. Set  $x \leftarrow 1$  and  $z \leftarrow c$
2. For  $i = 0$  to  $t$  do:
  - (a) If  $d_i = 1$ , set  $x \leftarrow x \cdot z \bmod N$
  - (b) Set  $z \leftarrow z^2$
3. Output  $x$

Notice that at stage  $i$ , if  $d_i = 0$  then the value  $x$  is not modified. However, if  $d_i = 1$ , then we multiply the previous result by  $c^{2^i}$ . This is in line with the formula  $c^d \bmod N = \prod_{i=0}^n c^{2^i d_i} \bmod N$ . (Notice that in the beginning of iteration  $i$ , the value  $z$  equals  $z^{2^i}$ .)

The timing attack of Kocher [20] uses the fact that when  $d_i = 1$ , an additional multiplication takes place. Now, assume that an attacker holds a smartcard that decrypts (or equivalently,

computes signatures). Then, the attacker asks it to decrypt a large number of random messages  $c_1, \dots, c_k \in_R \mathbb{Z}_N^*$  and measures the time  $T_i$  that it takes to compute  $c_i^d \bmod N$ . These timing measurements are now used to obtain  $d$ , one bit at a time. Since  $d$  is odd, we know that  $d_0 = 1$ . Regarding the second iteration (for  $d_1$ ), initially  $z = c^2 \bmod N$  and  $x = c$ . Thus, if  $d_1 = 1$ , the smartcard computes the product  $x \cdot z = c \cdot c^2 \bmod N$ ; otherwise, it does not. Let  $t_i$  be the time that it takes to compute  $c_i \cdot c_i^2$  where  $c_i$  is one of the random messages that was initially chosen (this value can be computed using the smartcard or another one of the same type).

Kocher showed that the ensembles  $\{t_i\}$  and  $\{T_i\}$  are correlated in the case that  $d_1 = 1$ , but behave independently in the case that  $d_1 = 0$ . By measuring the correlation between these ensembles, it is possible for the adversary to determine if  $d_1$  equals 1 or 0. Given  $d_0$  and  $d_1$ , it is possible to do the same thing for  $d_2$  and so on.

**Defenses.** Two known defenses are *delaying* and *blinding*. In a delaying defense, the smartcard just ensures that every decryption takes a fixed amount of time that is independent of  $d$ . In a blinding defense, the smartcard first chooses a random  $r$  and computes  $c' = c \cdot r^e \bmod N$ . Then,  $m' = c'^d \bmod N$  is computed and finally  $m = m'/r \bmod N$  is given as the output. Blinding is effective because the decryption process is applied to unknown values (whereas the fact that the  $c_i$ 's are known is crucial for the attack).

**Extensions.** Timing attacks were initially considered to be problematic for smartcard-type scenarios. However, it has been demonstrated that they can also be carried out against servers over the Internet, or between processes on the same machine. Other attacks against smartcards, called *power attacks*, are even more effective and cause a serious problem in implementations.

## Lecture 12

# Digital Signatures and Applications

Digital signatures are the analogue of message authentication codes in the public-key world. As in public-key encryption, there are distinct *signing* and *verification* keys. Only the party who knows the signing key can generate a valid signature, but anyone knowing the verification key can check the validity of a signature. Thus, it is possible to verify the authenticity of a message by checking whether or not the signature on the message is valid. This is crucial in many settings, including for successful electronic commerce. In particular, the property needed is that of *non-repudiation*, which means that the signer cannot later deny that he signed the document. Thus, a signed order for products cannot later be denied, and must be paid for. Another extremely important application of digital signatures that is in wide use today is that of digital certificates for a public-key infrastructure. We will discuss this application more at the end of the lecture.

### 12.1 Definitions

**The syntax.** A signature scheme is a triple of algorithms  $(G, S, V)$  as follows:

1.  $G$  is a *key generation* algorithm that receives  $1^n$  for input and returns a pair of keys  $(sk, vk)$  of length  $n$ . The key  $sk$  is called the *signing key* and the key  $vk$  is called the *verification key*.
2.  $S$  is a *signing* algorithm that receives a key  $sk$  and a message  $m$  and outputs a signature  $\sigma = S(sk, m)$ . We will often write  $S_{sk}(m)$  instead of  $S(sk, m)$ .
3.  $V$  is a *verification* algorithm that receives a key  $vk$ , a message  $m$  and a candidate signature  $\sigma$ , and returns a bit  $b = V(vk, m, \sigma)$ . As with  $S$ , we will often write  $V_{vk}(m, \sigma)$ .

It is required that except with negligible probability over the choice of a key, for every message  $m$  it holds that  $V_{vk}(m, S_{sk}(m)) = 1$ . We note that if the signing algorithm  $S$  receives only messages of a fixed size (say,  $n$ ), then we say that the signature scheme is *fixed-length*.

**Security.** As is to be expected, the security requirement of a signature scheme states that it should be hard to forge signatures. This should hold even when the adversary  $\mathcal{A}$  is given oracle access to a signing oracle (this oracle represents valid signatures that  $\mathcal{A}$  may obtain in a real attack). As in the MAC experiment, in order for  $\mathcal{A}$  to succeed, it must generate a valid signature on a message that was not queried to the signing oracle. We define the following experiment:

---

\* Lecture notes for an undergraduate course in cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

**The signature game**  $\text{Expt}_{\text{SIG}}$ . The generator  $G$  is run, outputting a key-pair  $(vk, sk)$ . Then,  $\mathcal{A}$  is given  $vk$  and oracle access to the signing oracle  $S(sk, \cdot)$ . At the conclusion of the experiment,  $\mathcal{A}$  outputs a pair  $(m^*, \sigma^*)$ . Let  $Q$  be the set of oracle queries made by  $\mathcal{A}$ . Then, we say that  $\mathcal{A}$  has succeeded, denoted  $\text{Expt}_{\text{SIG}}(\mathcal{A}) = 1$ , if  $V(vk, m^*, \sigma^*) = 1$  and  $m^* \notin Q$ . That is,  $\mathcal{A}$  outputs a message along with a valid signature, and  $\mathcal{A}$  did not query its oracle with this message. We are now ready to present the definition of security for signature schemes:

**Definition 12.1** A signature scheme  $(G, S, V)$  is  $(t, \epsilon)$  existentially secure against chosen-message attacks (or just  $(t, \epsilon)$ -secure) if for every adversary  $\mathcal{A}$  that runs for at most  $t$  steps

$$\Pr[\text{Expt}_{\text{SIG}}(\mathcal{A}) = 1] < \epsilon$$

Security here is called “existential” because it suffices to just come up with a forgery on any message, and the attack is called “chosen-message” because the adversary has oracle access to the signing algorithm and so can request signatures on any message that it chooses.

Notice that the security guarantee provided by digital signatures is far stronger than that of regular handwritten ones. First, we know that handwritten signatures can be forged with some expertise. Second, a handwritten signature does not verify that the content of the document was not somehow modified. In contrast, a digital signature scheme ensures that only the exact message signed can be accepted. Of course, this is under the assumption that a party’s signing key is kept secret.

**Fixed-length versus variable-length signature schemes.** We note that variable-length signature schemes can be obtained from fixed-length signature schemes in the same way as for message authentication codes. Specifically, if  $(G, S, V)$  is a fixed length scheme, then define  $S'_{sk}(m) = S_{sk}(h(m))$ , where  $h$  is a collision-resistant hash function (whose output is of the length to be input to  $S$ ). The proof that this yields a secure variable-length signature scheme is almost identical to our proof of the analogous claim for message authentication codes. We remark that it is actually possible to use a weaker type of hash function, called a universal one-way hash function.

## 12.2 Constructions

The basic idea behind many constructions of secure signature schemes is that, *in some sense* the signature operation is very reminiscent of the decryption operation. That is, in the setting of digital signatures, only the party knowing the secret key should be able to sign. Likewise, in the setting of encryption, only the party knowing the secret key should be able to decrypt. Therefore, the “private-key decryption operation” in encryption serves as a good starting point as the “signing operation” for signatures. Likewise, the “public-key encryption operation” in encryption serves as a good starting point as the “verification operation” for signatures. We stress that saying that one signs by decrypting and verifies by encrypting is very misleading; such terminology (although very popular) should not be used.

**A naive implementation.** In light of the above idea, let us consider the following scheme that uses plain RSA. (Note that since we do not aim to hide the message being signed, our objections to plain RSA in the context of encryption do not seem to be valid any more. Indeed, it is possible to construct secure deterministic encryption schemes.)

1. *Key generation*: let the verification key  $vk$  be the public RSA key  $(e, N)$  and let the signing key  $sk$  be the secret RSA key  $(d, N)$ .
2. *Signing algorithm*: upon input  $(d, N)$  and  $m$ , compute  $\sigma = m^d \bmod N$ .
3. *Verification*: upon input  $(e, N)$  a pair  $(m, \sigma)$ , compute  $\sigma^e \bmod N$  and output 1 if and only if the result equals  $m$ .

We call the above plain RSA signatures. This signature scheme is not secure. Specifically, given two signatures  $(m_1, \sigma_1)$  and  $(m_2, \sigma_2)$ , it is possible to compute  $\sigma_1 \cdot \sigma_2 \bmod N$  and thereby obtain a valid signature on the new message  $m_1 \cdot m_2 \bmod N$ . This is due to the “multiplicative” property of RSA (i.e.,  $m_1^d \bmod N \cdot m_2^d \bmod N = (m_1 \cdot m_2)^d \bmod N$ ).

**Secure constructions.** There are many different constructions of secure signature schemes. As with encryption, these range from schemes that have formal proofs of security (with reductions to assumptions like RSA, factoring and so on), to schemes that have proofs of security in the random-oracle model, and finally to schemes that have no formal analysis at all (but are just believed to be secure).

Due to lack of time, we will see only one construction here. This construction is called the full domain hash and only has a proof of security in the random-oracle model. Nevertheless, given the lack of time, this seems to be a good example to see.

**RSA with a full domain hash (RSA-FDH).** The basic idea here is that the multiplicative attack on plain RSA is possible due to the fact that  $m^d \bmod N$  has a simple structure that can be utilized. In contrast, if signing is carried out by  $h(m)^d \bmod N$ , then in order to carry out a multiplicative attack as described above, it must be possible to find three values  $m$ ,  $m_1$  and  $m_2$  such that  $h(m) = h(m_1) \cdot h(m_2) \bmod N$  (given these three values it is possible to multiply signatures on  $m_1$  and  $m_2$  in order to obtain a signature on  $m$ ). However, this seems to be a difficult task. In order to prove this, we need to rely on the random-oracle modelling of hash functions. In this model, it is assumed that the hash function is really an external oracle that holds a truly random function. As we have mentioned, a hash function is no such thing. However, this heuristic seems to be useful for gaining some confidence in the security of a scheme.

The actual construction is as follows:

1. *Key generation algorithm  $G$* : run the RSA key generation algorithm, obtaining  $e$ ,  $d$  and  $N$ . Then, find a collision-resistant hash function  $h$  with range  $\{1, \dots, N-1\}$  (where this function can also be appropriate for the random-oracle modelling). Finally, output  $sk = (d, N, h)$  and  $vk = (e, N, h)$ .  
We note that the range of typical hash functions (like SHA-1) does not equal  $\{1, \dots, N-1\}$ . Nevertheless, it is possible to modify them appropriately; see [5] for details.
2. *Signing algorithm  $S$* : given  $sk = (d, N, h)$  and  $m$ , compute  $\sigma = S_{sk}(m) = h(m)^d \bmod N$ .
3. *Verification algorithm  $V$* : given  $vk = (e, N, h)$ ,  $m$  and  $\sigma$ , compute  $\sigma^e \bmod N$  and compare the result to  $h(m)$ . Output 1 if and only if they are equal. In other words, define  $V_{vk}(m, \sigma) = 1$  if and only if  $\sigma^e \bmod N = h(m)$ .

We have the following theorem (for the sake of simplicity, we state and prove the theorem informally and without using concrete analysis).

**Theorem 12.2** *Replace the function  $h$  in the RSA-FDH construction by a random oracle. Then, if there exists a polynomial-time adversary  $\mathcal{A}$  that can forge a signature with non-negligible probability, then there exists a polynomial-time algorithm that can invert the RSA function.*

**Proof Sketch:** The basic idea behind the proof here is that if it is possible to forge a signature, then this involves computing  $x^d \bmod N$  for a *random value*  $x$  (the reason that  $x$  is random is that  $x = h(m)$  and  $h$  is modelled as a random oracle). Therefore, computing such a signature is equivalent to inverting RSA on a random point.

More formally, let  $\mathcal{A}$  be a polynomial-time adversary that successfully generates a forgery  $(m^*, \sigma^*)$  with non-negligible probability. Furthermore, let  $t$  be an upper bound on the number of queries that  $\mathcal{A}$  makes to the random oracle. Without loss of generality, we assume that every query made by  $\mathcal{A}$  to the signing oracle is first asked to the random oracle.

Let  $M$  be a machine that receives  $(e, N, y)$  where  $y \in_R \{1, \dots, N - 1\}$ , and attempts to find  $x$  such that  $x^e = y \bmod N$  (i.e.,  $M$  attempts to invert RSA on a random point). The machine  $M$  invokes  $\mathcal{A}$  with input  $vk = (e, N)$ , chooses a random index  $i \in \{1, \dots, t\}$  and works as follows:

1. *Random-oracle queries:* The  $i^{\text{th}}$  query  $m_i$  that  $\mathcal{A}$  makes to its random-oracle is answered with the value  $y$  from  $M$ 's input.<sup>1</sup> All other queries  $m_j$  are answered by choosing a random value  $r_j$  and replying with  $s_j = r_j^e \bmod N$ . (These values are chosen in this way in order to enable  $M$  to properly answer a signing-oracle query for the message  $m_j$ ; see below.)
2. *Signing queries:* Let  $m$  be a query that  $\mathcal{A}$  asks to its signing oracle. If  $m = m_i$ , then  $M$  halts with output failure (because in this case  $m$  cannot be the message for which a forgery is generated). If  $m = m_j$ , then  $M$  replies with the value  $r_j$  that was chosen for  $m_j$  when it was queried to the random oracle. (Recall that by our assumption, all queries to the signing oracle are first queried to the random oracle.)

Let  $(m^*, \sigma^*)$  be  $\mathcal{A}$ 's output. If  $m^*$  was never queried to the random oracle, then the probability that  $\sigma^*$  is a valid signature is at most  $1/N$ , which is negligible (this holds because without querying  $m^*$ , the value of  $h(m^*)$  is uniformly distributed given  $\mathcal{A}$ 's view). We therefore ignore this event. We have the following three possible events:

1.  $m^* = m_j$  for some  $j \neq i$ : in this case,  $M$  outputs failure. (A forgery here is of no help in inverting  $y$ .)
2.  $m^* = m_i$  but  $\sigma^*$  is not a valid signature: here too,  $M$  outputs failure.
3.  $m^* = m_i$  and  $\sigma^*$  is a valid signature: this implies that  $(\sigma^*)^e = y \bmod N$  and therefore  $\sigma^*$  is the inverse of  $y$ . In this case  $M$  has succeeded and so it outputs  $x = \sigma^*$  and stops.

It remains to analyze the probability that  $M$  succeeds. Ignoring the negligible probability that  $\mathcal{A}$  generates a successful forgery for  $m^*$  without querying  $m^*$  to the random oracle, we have that  $M$  succeeds with probability  $\epsilon/t$  where  $\epsilon$  is the probability that  $\mathcal{A}$  generates a successful forgery. This holds because  $M$  chooses the index  $i$  for which  $\mathcal{A}$  generates the forgery with probability  $1/t$ . Thus, success is achieved with probability  $\epsilon \cdot 1/t$ . Since  $\mathcal{A}$  is polynomial, so is  $t$ . Therefore, the success of  $M$  is non-negligible. ■

---

<sup>1</sup>The hope is that  $\mathcal{A}$  will attempt to forge a signature for the message  $m_i$  and will therefore invert  $y$  when it generates its forgery.

**A remark on the random-oracle model.** Having seen our first “random-oracle proof”, it is worth remarking on the meaning of the proof. First, we stress that formally, such a theorem says *nothing* about the security of the scheme when a concrete hash function is used. Thus, any argument saying that security is implied, is heuristic. Nevertheless, the intuition here is that if the hash function itself is not attacked, then an attack on RSA-FDH *should* yield an inversion algorithm for RSA. Any attack on RSA-FDH that does not yield such an inversion algorithm *should* imply some sort of attack on the “random properties” of the hash function that is used. As you can see, such arguments employ a lot of handwaving and cannot be made rigorous. Nevertheless, they are arguably better than nothing.

**Partial domain hash.** Notice that the proof of Theorem 12.2 relies heavily on the fact that the range of  $h$  is the entire RSA domain (or at least, a polynomial fraction of it). Otherwise, the input  $y$  that  $M$  receives may not be in the range of  $h$ , and so cannot be set to be the reply of the random oracle. We note that in some standards, the hash-and-sign paradigm as in RSA-FDH is used, but the output of the hash is much smaller than the size of the modulus (e.g., it could be the output of SHA-1 which is only 160 bits). There is no proof of security for such schemes (even in the random oracle); the best that is known is that for the case of  $e = 2$  (i.e., the Rabin function), security in the random-oracle model holds as long as the output of  $h$  is larger than  $2/3$  of the size of the modulus.

## 12.3 Certificates and Public-Key Infrastructure

One very important use of digital signatures is for generating certificates that are used in a public-key infrastructure. Very briefly, the problem that we wish to solve here is that of distributing *public keys* in a reliable way. That is, we have seen that just having the receiver send its public key to the sender is dangerous because an adversary can carry out a man-in-the-middle attack. However, if the public key could somehow be *certified* so that the sender would know that it is correct, this would solve the problem.

**Certificates.** A digital certificate is a “document” containing a party’s identity, its public key (for signatures or encryption), and other information (like an expiration date). Thus, my certificate would contain my name “Yehuda Lindell”, my identity number (or social security number), and my public key  $pk$ . This document is then signed by a **certificate authority (CA)** who is trusted to verify the identity of a party before generating a certificate. (In reality, it is possible to download certificates over the Internet. However, these have very low reliability; good certificates must be paid for and require providing some proof of identity to the CA.)

Note that a certificate *binds a person’s identity to a specified public-key*. This is the property that is needed for key distribution, and also for ensuring proper verification of digital signatures.

**Public-key infrastructure.** Obtaining a certificate does not help at all if the certificate authority’s verification key is not known to the involved parties. In a public-key infrastructure, it is assumed that the CA’s verification key *is* known. Therefore, upon receiving a certificate from Alice, it is possible to first verify the name on the certificate (to see that it indeed belongs to Alice), and then check that the CA’s signature is correct. If yes, we can be confident that the key received really belongs to Alice and thus it can be used for encrypting messages to send to Alice. Likewise, if the key involved is a verification key, then we can be confident that a given signature was actually generated by Alice and no one else.

Of course, we have now just moved the problem to that of obtaining the CA's public verification key. However, since there are only a limited number of CA's, this is a much smaller problem. In practice, Windows's web browsers have CA verification keys hardwired into them (and buried deep inside the operating system so that they cannot be modified). In order to see this in Explorer, go to **Tools** → **Internet Options** → **Content** → **Certificates** → **Trusted Root Certification Authorities**. (It is worthwhile looking at the certificates and the information in them.) Thus, assuming that we trust the certificate authorities (or at least some of them), and that we trust Windows (for this purpose, this is definitely reasonable), this solves the problem of key distribution. We note that secure Internet connections use certificates, and they are automatically verified by your browser. In order to see this, go to <https://hb2.bankleumi.co.il/H/Login.html> (this is the customer login site of Israel's Bank Leumi). Then, double-click on the small yellow padlock on the bottom-right of your browser. The certificate information will appear.

**Other issues.** There are many other issues to be discussed on this topic. First, it is important to note that, in general, a public-key infrastructure is actually a tree of trust. For example, a company may obtain one certificate from a CA (for its IT manager) and then use that certificate to generate certificates for all its employees. Verifying the validity of a certificate then requires verifying the chain of certificates from the one received to a trusted "root certificate". Another issue that is important is that of revocation. Once a certificate has been issued, how is it possible to revoke it in the case that the associated secret key has been leaked (or some other problem). For this purpose there are "certificate revocation lists"; we will not discuss this more here. Finally, we note that all of the above assumes that secret keys are never leaked. In practice, this means that a party's secret key should never be kept on his or her computer (because this would make it vulnerable in the case that a virus successfully invades the machine). Therefore, smartcards should be used: a smartcard is a small processor that resides in a tamper-proof environment. Keys should be generated on the card and the secret key should *never* be exported. Rather, any decryption or signing operation should take place on the card.

## 12.4 Combining Encryption and Signatures – SignCryption

We conclude our treatment of digital signatures with the analogous problem to that of "encrypt and authenticate". As with the private-key scenario, naive solutions fail. Indeed just encrypting-then-authenticating (or the reverse) leads to many problems. For example, naive encrypt-then-sign enables an attacker to take an encrypted message from someone else and resign it with its own key. That is, let  $pk_i$  and  $sk_i$  be the respective public-encryption and secret-signing keys of Party  $i$ . Then, receiving  $\sigma = S_{sk_1}(E_{pk_2}(m))$  should not at all convince Party 2 that Party 1 knows the message  $m$ . Indeed, Party 1 could just have taken it from Party 3 who generated  $S_{sk_3}(E_{pk_2}(m))$ . We note that these problems can be overcome, for example, by including the identity of the sender and receiver with the message. See [1] for just one paper on this topic.

# Lecture 13

## Secure Protocols

In this lecture we will study some secure protocols. In particular, we will describe the SSL authentication and session-key generation protocol, and we will present Shamir's secret-sharing protocol. Our presentation here is very ad-hoc. We do not present formal definitions or proofs of security, even though they are essential (as in all fields of cryptography).

### 13.1 The SSL Protocol Version 3.0

The SSL protocol is used in order to provide authentication and secure communication over the Internet. It is used in almost all secure Internet transactions, and so is worthwhile studying. We note that a newer version of SSL (called TLS) is also used very often. However, the parts of the protocol that we will describe here are almost the same in both cases.

The SSL protocol is made up of two parts: **(1)** the **handshake protocol** which is used for authentication, generating secret session keys, and agreeing on certain parameters, and **(2)** the **record layer**, which is used for encrypting and authenticating communication (using the session keys that were generated in the handshake protocol). We will focus exclusively on the handshake protocol here.

**The handshake protocol.** In this protocol, the parties carry out a number of tasks. First, they agree on the SSL protocol version and the cryptographic algorithms to be used. Furthermore, they authenticate each other (optionally) and use public-key encryption (or Diffie-Hellman) in order to generate secret session keys. We will assume the default scenario where the server is to be authenticated to the client, but the client is not authenticated to the server.<sup>1</sup> We will also only consider the initial handshake protocol, and not the protocol used for *resuming* SSL sessions.

In the first step of the protocol, the parties exchange `serverHello` and `clientHello` messages. These messages include the protocol version, a unique session identifier (chosen by the server), the cryptographic algorithms to be used later (this is called the **cipher suite**), the compression method to be used, and respective random values denoted `serverHello.random` and `clientHello.random`. The client sends its hello message first, after which the server replies with its hello message and its *digital certificate* for authentication.

---

\* Lecture notes for an undergraduate course in cryptography. Yehuda Lindell, Bar-Ilan University, Israel, 2005.

<sup>1</sup>The authentication of the client to the server is often carried out on the application level, by having the client send a password over an SSL connection.

At this point, there are two possibilities, depending on the type of digital certificate owned by the server:

1. *The server's digital certificate contains an RSA public encryption key  $(N, e)$ :* In this case, the client chooses a random 48 byte pre-master secret  $R_{pms}$  and sends  $c = RSA_{N,e}(R_{pms})$  to the server. PKCS encryption should be used here (and in particular OAEP that is secure against chosen-ciphertext attacks).
2. *The server's digital certificate contains a public verification key for signatures:* In this case, the client and server run a key-exchange protocol. Here too there are two possibilities:

- (a) *RSA key exchange:* The server chooses a temporary pair of RSA keys  $(pk, sk)$ , signs on the public-key  $pk$  with the signing key that is bound to its certificate, and sends  $\langle RSA, pk, \sigma \rangle$  to the client, where  $\sigma$  is the signature on  $pk$ .

The client verifies the signature using the public verification-key from the certificate (that is also verified by the client). If it is correct, the client chooses a random 48 byte pre-master secret  $R_{pms}$  as above, and sends  $c = RSA_{N,e}(R_{pms})$  to the server.

- (b) *Diffie-Hellman key exchange:* The server chooses a random  $(p, g, x_s)$  as in the Diffie-Hellman key exchange protocol, and sends the client the message  $\langle DH, (p, g, y_s), \sigma \rangle$  where  $y_s = g^{x_s} \bmod p$  and  $\sigma$  is a signature on  $(p, g, y_s)$ .

The client receives the tuple, verifies the signature, chooses a random  $x_c$  and sends  $y_c = g^{x_c} \bmod p$  to the server.

Both parties set the pre-master secret to  $R_{pms} = g^{x_s x_c} \bmod p$ .

At this stage of the protocol, the client and server both hold the same pre-master secret. They both now use the pre-master secret in order to derive the master secret  $R_{MS}$ . The master secret is derived by applying a series of hashes to the pre-master secret and the `clientHello.random` and `serverHello.random` messages. (This series of hashes is similar to the HMAC construction, but is not exactly the same. Presumably, the assumption is that this serves as a pseudorandom function, with the pre-master secret as the key.) Following this, the protocol is concluded by each sending finish messages that are an HMAC of all the traffic of the handshake protocol, using the master secret as the key. Specifically, the server sends  $HMAC_{R_{MS}}(\overline{m}_s, \text{server})$  where  $\overline{m}_s$  is the series of all the messages that the server sent and received in the handshake protocol (from the `hello` up until but not including the `finish` messages). Likewise, the client sends  $HMAC_{R_{MS}}(\overline{m}_c, \text{client})$ .

These values are checked, and if they are correct the parties proceed to derive the session keys from the master secret. Specifically, the master secret is hashed together with client and server random messages, in the same way as above. (That is, the hashing is similar to the HMAC construction. However, here the key to the pseudorandom function is the master secret, and not the pre-master secret.) The result is then divided into a client MAC key, a server MAC key, a client encryption key, a server encryption key, a client IV (for CBC encryption) and a server IV. This concludes the description of the handshake protocol.

**The security of SSL.** Consider first the default case, where the server's certificate contains an RSA encryption key. In this case, the main point to observe is that the client receives the server's key (without modification), and encrypts the pre-master secret with that key. Since only the server can decrypt this secret, this implies that no adversary (even one launching a man-in-the-middle attack) can learn the pre-master secret. We note that the server has no idea who the client is; as we have mentioned, such client authentication is optional in SSL but is typically dealt with at the

level of the application if necessary, with passwords and the like. (Client authentication in SSL is problematic because it requires the client to have a certificate.) Nevertheless, the important point is that the *client* knows that it shares a secret with the correct server, and that no adversary knows that secret. Thus, the client is guaranteed to have established a secure channel with the server.

One attack of importance is where the adversary causes two parties who both support SSL version 3.0 to use version 2.0 (which has many problems). Likewise, the adversary may cause the parties to use extremely weak 40 bit encryption, even though they support stronger encryption. However, in order to do this, it must change at least one of the `clientHello` or `serverHello` messages. Since the parties apply a MAC over the entire transcript of the handshake protocol, this will be detected at the end. In general, it is good practice to apply a MAC over the entire transcript in key exchange protocols.

**Signing-only certificates.** We note that this mode was mainly introduced to deal with the problem of export regulations. If SSL is to be run in a country where only short encryption keys are allowed, then it is far better to use a strong signing key as the long-term key in the certificate (strong keys for signing are allowed), and then generate “short” keys for one-time use. Today this issue is less problematic since export regulations have been significantly relaxed.

**SSL weaknesses.** We describe one security hole in SSL that was pointed out by [39]. Notice that in the key-exchange mode, the server signs on the RSA key or Diffie-Hellman parameters. However, the server does *not* sign on the tag that says which key exchange is to be used (i.e., RSA or DH). This mistake can be utilized in the following attack.

Assume that a server chooses Diffie-Hellman key-exchange mode and sends  $\langle \text{DH}, (p, g, y_s), \sigma \rangle$  to the client. A man-in-the-middle can modify this message to  $\langle \text{RSA}, (p, g, y_s), \sigma \rangle$  and then send it to the client. Notice that the signature will still be correct (since the DH tag is not signed). Now, the SSL specification does not require the client to check the length of the values, and as such it may not detect any malicious behaviour. In such a case, the client is likely to interpret  $p$  as the RSA modulus and  $g$  as the RSA exponent. It then sends the server the ciphertext  $c = (R_{pms})^g \bmod p$ . The adversary receives  $c$  and can easily decrypt because  $g$  is known and  $p$  is a prime (in such a field, extracting  $g^{\text{th}}$  roots can be carried out efficiently). Furthermore, the adversary can just send the server a Diffie-Hellman value  $y$  for which it knows  $x$  such that  $y = g^x \bmod p$ . The final result is that the adversary knows both the pre-master secret  $R_{pms}$  that is held by the client and the pre-master secret  $g^{x \cdot x} \bmod p$  that is held by the server.

## 13.2 Secret Sharing

The problem that we wish to consider here is the following. Assume that there are  $n$  parties who wish to share a secret  $s$  so that any subset of  $t$  or more parties can reconstruct the secret, yet any subset of less than  $t$  parties learn nothing about the secret. In more detail, there exists one party (the dealer) who holds a secret  $s$ . The dealer generates messages for each of the parties, and privately sends each party its designated message. Following this preprocessing stage, the above privacy and reconstruction properties must hold. A scheme that fulfills the above requirements is called a  $(t, n)$ -threshold secret-sharing scheme.

This problem has many applications in the constructions of secure protocols. For example, one may wish to construct a signature scheme so that only a subset of  $t$  parties can sign. These schemes (and many others) use secret sharing as a basic building block.

**Shamir's secret sharing protocol [33].** This scheme is based on the fact that for any for  $t$  points on the two dimensional plane  $(x_1, y_1), \dots, (x_t, y_t)$  there exists a unique polynomial  $q(x)$  of degree  $t - 1$  such that for every  $i$ ,  $q(x_i) = y_i$ . Furthermore, it is possible to find this polynomial efficiently, using interpolation. We now describe the scheme itself:

1. *Dealer instructions:* Upon input  $s$ , choose a prime  $p$  such that  $p$  is greater than both  $s$  and  $n$ . Then, choose a random polynomial  $q(\cdot)$  of degree  $t - 1$  in  $\mathbb{Z}_p[x]$ , under the constraint that  $q(0) = s$ . More specifically, choose coefficients  $a_1, \dots, a_{t-1}$  such that each  $a_i$  is uniformly distributed in  $\{0, \dots, p - 1\}$ . Then, let  $q(x) = s + a_1x + a_2x^2 + \dots + a_{t-1}x^{t-1}$ .

For each party  $P_i$ , send party  $P_i$  the pair  $(i, q(i))$ . (Note that here  $n$  different values are needed. This is the reason that  $p$  must be greater than both  $s$  and  $n$ .)

2. *Reconstruction:* Given  $t$  values  $(i_j, q(i_j))$ , use polynomial interpolation in order to obtain the polynomial  $q(\cdot)$ . Then, output  $s = q(0)$ .

It is clear that the reconstruction procedure works. It remains to show that any subset of less than  $t$  parties obtain no information about the secret  $s$ . In order to see this, assume that we are given  $t - 1$  shares; i.e., points  $(i_1, q(i_1)), \dots, (i_{t-1}, q(i_{t-1}))$ . Then, for every possible choice of  $s \in \{0, \dots, p - 1\}$ , there exists a unique polynomial of degree  $t$  that passes through the  $t - 1$  given points and the point  $(0, s)$ . Since all polynomials are chosen with equal probability, this means that the value of  $s$  is perfectly hidden.

**Verifiable secret sharing (VSS).** A significant problem with the above secret-sharing scheme occurs in the case that some of the parties are malicious. Such parties can present false values (i.e., can lie about the value of the shares that they received), thereby causing the reconstructed value to be false. Notice that the dealer may also behave maliciously. A verifiable secret sharing scheme is one that prevents malicious behaviour by malicious participants.

We describe briefly the VSS scheme of Feldman [21]. The idea behind the scheme is to have the dealer publish encryptions of the coefficients that enable the recipients to check the validity of their shares. Specifically, in addition to sending the Shamir shares, the dealer chooses a random prime  $p$  and generator  $g$  and broadcasts  $C_0 = g^s \bmod p$  and  $C_i = g^{a_i} \bmod p$ , for  $i = 1, \dots, t - 1$ . Now, upon receiving a share  $(j, y_j)$ , the party  $P_j$  checks that

$$g^{y_j} = \prod_{i=0}^t (C_i)^{j^i} \quad (13.1)$$

Notice that  $y_j$  is supposed to equal  $q(j) = s + \sum_{i=1}^t a_i \cdot j^i$ . Therefore,  $g^{y_j}$  is supposed to equal  $g^{q(j)} = g^s \cdot \prod_{i=1}^t g^{a_i \cdot j^i}$ . If the values do not match, the party broadcasts an "accusation". The dealer must then respond by broadcasting a new pair of points  $(j', q(j'))$  so that Eq. (13.1) holds. For reconstruction, the new point  $j'$  is used instead.

Note that if the dealer cheats, then it will receive accusations, which it must then answer correctly. If it does not, all parties will know that it is cheating, because these values must be *broadcast*. On the other hand, if malicious participants falsely publish accusations, they will be able to obtain additional shares. This problem is solved by setting the threshold  $t$  to be greater than  $\frac{2n}{3}$ , and assuming that the number of dishonest parties  $f$  is less than  $\frac{n}{3}$ . Since each dishonest party can publish at most one accusation, they cannot obtain  $\frac{2n}{3}$  shares, and so still cannot learn the secret  $s$ .

We note that this scheme reveals  $C_0 = g^s \bmod p$  which does contain some information about  $s$ . Nevertheless, under the discrete log assumption (and assuming that  $s$  is a random secret), this implies that some bits of  $s$  are fully hidden. (We will not elaborate on this point here.)



# Bibliography

- [1] J.H. An, Y. Dodis and T. Rabin. On the Security of Joint Signature and Encryption. In *EUROCRYPT 2002*, Springer-Verlag (LNCS 2332), pages 83–107, 2002.
- [2] M. Bellare, R. Canetti, and H. Krawczyk. Keying Hash Functions for Message Authentication. In *CRYPTO'96*, Springer-Verlag (LNCS 1109), pages 1–15, 1996. For a lighter version, see *RSA Laboratories' CryptoBytes* Vol. 2, No. 1, Spring 1996.
- [3] M. Bellare, A. Desai, E. Jorjipii and P. Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *38th FOCS*, pages 394–403, 1997. Full version available from <http://www.cs.ucsd.edu/users/mihir/papers/sym-enc.html/>.
- [4] M. Bellare, J. Kilian and P. Rogaway. The Security of the Cipher Block Chaining Message Authentication Code. *Journal of Computer and System Sciences*, 61(3):362–399, 2000.
- [5] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In the *1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993. Available at <http://www.cs.ucsd.edu/users/mihir/papers/ro.html>.
- [6] M. Bellare and P. Rogaway. Optimal asymmetric encryption – How to encrypt with RSA. In *EUROCRYPT'94*, Springer-Verlag (LNCS 950), pages 92–111, 1995.
- [7] E. Biham and A. Shamir. Differential Cryptanalysis of DES-Like Cryptosystems. In *CRYPTO'90* and the *Journal of Cryptology*, 4(1):3–72, 1991.
- [8] D. Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. In *CRYPTO'98*, Springer-Verlag (LNCS 1462), pages 1–12, 1994.
- [9] D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the American Mathematical Society* (AMS), 46(2):203–213, 1999.
- [10] D. Boneh. The Decisional Diffie-Hellman Problem. Online paper can be downloaded from <http://crypto.stanford.edu/~dabo/abstracts/DDH.html>.
- [11] D. Boneh, A. Joux, and P. Nguyen. Why Textbook ElGamal and RSA Encryption are Insecure. In *AsiaCrypt 2000*, Springer-Verlag (LNCS 1976), pages 30–44, 2000.
- [12] R. Canetti, O. Goldreich and S. Halevi. The Random Oracle Methodology, Revisited. In the *30th STOC*, 209–218, 1998. Available at: [http://www.wisdom.weizmann.ac.il/~oded/p\\_rom.html](http://www.wisdom.weizmann.ac.il/~oded/p_rom.html).

- [13] R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *EUROCRYPT 2001*, Springer-Verlag (LNCS 2045), pages 453–474, 2001.
- [14] R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. In *CRYPTO'98*, Springer-Verlag (LNCS 1462), pages 13–25, 1998.
- [15] I. Damgård. Collision Free Hash Functions and Public Key Signature Schemes. In *EUROCRYPT'87*, Springer-Verlag (LNCS 304), pages 203–216, 1988.
- [16] I. Damgård. A Design Principle for Hash Functions. In *CRYPTO'89*, Springer-Verlag (LNCS 435), pages 416–427, 1990.
- [17] H. Delfs and H. Knebl. *Introduction to Cryptography, Principles and Applications*. Springer-Verlag, 2002.
- [18] W. Diffie, and M.E. Hellman. New Directions in Cryptography. *IEEE Trans. on Info. Theory*, IT-22, pages 644–654, 1976.
- [19] T. El Gamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *CRYPTO'84*, Springer-Verlag (LNCS 196), pages 10–18, 1985.
- [20] P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO 1996*, Springer-Verlag (LNCS 1109), pages 104–113, 1996.
- [21] P. Feldman. A Practical Scheme for Non-interactive Verifiable Secret Sharing In *28th FOCS*, pages 427–437, 1987.
- [22] Shai Halevi and Hugo Krawczyk. Public-Key Cryptography and Password Protocols. In *ACM Transaction on Information and System Security*, 2(3):230–268, 1999.
- [23] Kerberos Papers and Documentation. <http://web.mit.edu/kerberos/papers.html>
- [24] A. Kerckhoffs. La Cryptographie Militaire. *Journal des Sciences Militaires*, vol. IX, pages 5–83, Jan. 1883, pages 161–191, Feb. 1883.
- [25] N. Koblitz. *A Course in Number Theory and Cryptography (Second Edition)*. Graduate Texts in Mathematics, Springer-Verlag, 1994.
- [26] H. Krawczyk. The Order of Encryption and Authentication for Protecting Communication (or: How Secure is SSL?). In *CRYPTO'01*, Springer-Verlag (LNCS 2139), pp. 310–331, 2001.
- [27] M. Matsui. Linear Cryptoanalysis Method for DES Cipher. In *EUROCRYPT'93*, pages 386–397, 1993.
- [28] A. Menezes, P. Van Oorschot and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997. Available for free download from <http://ww.cacr.math.uwaterloo.ca/hac/>.
- [29] R.C. Merkle. One-Way Hash Functions and DES. In *CRYPTO'89*, Springer-Verlag (LNCS 435), pages 428–446, 1990.

- [30] M. Rabin. Digitalized Signatures and Public Key Functions as Intractable as Factoring. TR-212 LCS, MIT, 1979.
- [31] R. Rivest. The MD5 Message-Digest Algorithm. *Internet RFC 1321*, 1992. Available for download from: <http://www.ietf.org/rfc.html>.
- [32] R.L. Rivest, A. Shamir, and L.M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [33] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.
- [34] C.E. Shannon. Communication Theory of Secrecy Systems. *Bell Systems Technical Journal*, Vol. 28, pages 656-715, 1949.
- [35] V. Shoup. A Computational Introduction to Number Theory and Algebra. Available online from <http://shoup.net/ntb/>.
- [36] N. Smart *Cryptography, An Introduction*. McGraw-Hill, 2003.
- [37] D. Stinson *Cryptography, Theory and Practice*. CRC Press, 2002.
- [38] A. Turing. On Computable Numbers, With an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–254, 1937.
- [39] D. Wagner and B. Schneier. Analysis of the SSL 3.0 Protocol. In the *2nd USENIX Workshop on Electronic Commerce*, 1996.
- [40] The Secure Hash Standard. *FIPS Publication 180-1*, 1995. Available for download from: <http://www.itl.nist.gov/fipspubs/fip180-1.htm>.
- [41] The SSL Protocol Version 3.0, <http://wp.netscape.com/eng/ss13/draft302.txt>.
- [42] The TLS Protocol Version 1.0, RFC 2246, <http://www.ietf.org/rfc/rfc2246.txt>.