6 Introduction to Cryptography

This section gives a short introduction to cryptography. It is based on the recent tutorial by Jörg Rothe. For an in-depth treatment of cryptography, please consult the Handbook of Applied Cryptography whose reference is given at the end of this section.

The notion of a cryptosystem is formally defined as follows:

Definition 6.1 *A* cryptosystem *is a quintuple* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ *such that*

1. $\mathcal{P}$*,* $\mathcal{C}$*, and* $\mathcal{K}$ *are finite sets, where*

   - $\mathcal{P}$ *is the* plain text space *or* clear text space*,*
   - $\mathcal{C}$ *is the* cypher text space*, and*
   - $\mathcal{K}$ *is the* key space*.*

   Elements of $\mathcal{P}$ *are referred to as* plain text*, and elements of* $\mathcal{C}$ *are referred to as* cypher text*. A* message *is a string of plain text symbols.*

2. $\mathcal{E} = \{E_k \mid k \in \mathcal{K}\}$ *is a family of functions* $E_k : \mathcal{P} \to \mathcal{C}$ *that are used for encryption, and* $\mathcal{D} = \{D_k \mid k \in \mathcal{K}\}$ *is a family of functions* $D_k : \mathcal{C} \to \mathcal{P}$ *that are used for decryption.*

3. *For each key* $e \in \mathcal{K}$ *there exists a key* $d \in \mathcal{K}$ *such that for each* $p \in \mathcal{P}$*:*

$$D_d(E_e(p)) = p\ .$$

A cryptosystem is called symmetric *(or* private key*) if* $d = e$*, or if* d *can at least be "easily" computed from* e*. A cryptosystem is called* asymmetric *(or* public key*) if* $d \neq e$ *and it is "computationally infeasible in practice" to compute* d *from* e*. Here,* d *is the* private key *and* e *is the* public key*.*

In the following, we will present and discuss some symmetric and asymmetric cryptosystems and their applications.

6.1 Symmetric cryptosystems

Symmetric cryptosystems are much older than asymmetric cryptosystems. Consider the English alphabet $\Sigma = \{\text{A,B,}\ldots\text{,Z}\}$. To carry out the arithmetic modulo 26 with letters as if they were numbers, we identify Σ with $\mathbb{Z}_{26} = \{0, 1, \ldots, 25\}$. Thus, 0 represents A, 1 represents B, and so on. We start with an example called *Caesar cipher*, named after the Roman emperor Julius Caesar.

Caesar cypher

Let $\mathcal{K} = \mathbb{Z}_{26}$ and let $\mathcal{P} = \mathcal{C} = \Sigma$. The Ceasar cypher encrypts messages by shifting (modulo 26) each character of the plain text by the same number k of letters in the alphabet, where k is the key. Shifting each character of the cipher text back using the same key k reveals the original message:

- For each $e \in \mathbb{Z}_{26}$, define the encryption function $E_e : \Sigma \rightarrow \Sigma$ by
$$E_e(p) = (p + e) \bmod 26$$
where addition with e modulo 26 is carried out character-wise, that is, each character $m_i \in \Sigma$ of a message $m \in \Sigma^*$ is shifted by e positions to $m_i + e$ modulo 26. For example, using the key $e = 11 = L$, the message "SUMMER" will be encrypted as "DFXXPC".

- For each $d \in \mathbb{Z}_{26}$, define the decryption function $D_d : \Sigma \rightarrow \Sigma$ by
$$D_d(c) = (c - d) \bmod 26$$
where subtraction by d modulo 26 again is carried out character-wise. Hence, $d = e$. For example, decrypting the cipher text "DNSZZW" with the key $d = 11$ reveals the plain text "SCHOOL".

Since the key space is very small (there are only 26 possible keys), breaking the Ceasar cypher is very easy.

Hill cypher

Since the Roman empire, more and more elaborate cryptosystems were developed by people such as the Italian mathematician Leon Battista Alberti (born in 1404), the German abbot Johannes Trithemius (born in 1492), the French cryptographer and diplomat Blaise de Vigenère (1523-1596), or Lester Hill, who invented the Hill cipher in 1929. The Hill cipher is based on linear algebra and, like the Vigenère cipher, is an affine linear block cipher.

For a fixed $n \in \mathbb{N}$, the key space $\mathcal{K}$ is the set of all invertible $n \times n$ matrices in $\mathbb{Z}_{26}^{n \times n}$. $\mathcal{P} = \mathcal{C} = \mathbb{Z}_{26}^{n}$. Messages $m \in \mathbb{Z}^*$ that are longer than n are split into blocks of length n and are encrypted block-wise. All arithmetic operations are carried out modulo 26. The *Hill cipher* is defined as follows:

- For each $K \in \mathcal{K}$, define the encryption function $E_K : \mathbb{Z}_{26}^{n} \rightarrow \mathbb{Z}_{26}^{n}$ by
$$E_K(p) = K \cdot p \bmod 26$$
where "$\cdot$" denotes matrix multiplication modulo 26.

- Letting K^{-1} denote the inverse matrix of K, the decryption function $D_{K^{-1}} : \mathbb{Z}_{26}^{n} \rightarrow \mathbb{Z}_{26}^{n}$ is defined by
$$D_{K^{-1}}(c) = K^{-1} \cdot c \bmod 26 \; .$$

Since K^{-1} can easily be computed from K, the Hill cypher is a symmetric cryptosystem. It is also the most general linear block cipher. Affine linear block ciphers are easy to break by known-plain-text attacks. That is, for an attacker who knows some sample plain texts with the corresponding encryptions, it is not too hard to find the key used to encrypt these plain texts. Even encryptions of unknown texts can be decrypted with not too much of work. The method of frequency counts is often useful here. It exploits the redundancy of the natural language used for plain text messages. For example, in many languages the letter "E" occurs most frequently, with a percentage of 12.31% in English, of 15.87% in French, and even of 18.46% in German.

In light of many successful methods for breaking symmetric cryptosystems, it is natural to ask whether there exist any cryptosystems that guarantee *perfect* secrecy. Next we present such a system.

Vernam's one-time pad

To discuss perfect secrecy of cryptosystems in mathematical terms, we need some preliminaries from elementary probability theory.

Definition 6.2 *Let A and B be events with* $\Pr[B] > 0$.

- *The probability that A occurs under the condition that B occurs is defined by*

$$\Pr[A \mid B] = \frac{\Pr[A \cap B]}{\Pr[B]} \ .$$

- *A and B are* independent *if* $\Pr[A \cap B] = \Pr[A] \cdot \Pr[B]$ *(or equivalently, if* $\Pr[A \mid B] = \Pr[A]$*).*

Definition 6.3 *A cryptosystem* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ *provides* perfect secrecy *if and only if*

$$\forall p \in \mathcal{P}\ \forall c \in C: \ \Pr[p \mid c] = \Pr[p] \ .$$

That is, a cryptosystem achieves perfect secrecy if the event that some plain text p is encrypted and the event that some cipher text c is received are independent. In this case, an eavesdropper learns nothing about p from knowing c. Shannon was able to characterize precisely what it means for a cryptosystem to provide perfect secrecy.

Theorem 6.4 (Shannon) *Let $S = (\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ be a cryptosystem with $|\mathcal{C}| = |\mathcal{K}|$ and $\Pr[p] > 0$ for each $p \in \mathcal{P}$. Then, S provides perfect secrecy if and only if*

1. *$\Pr_{\mathcal{K}}$, the probability distribution on $\mathcal{K}$, is the uniform distribution, and*
2. *for each $p \in \mathcal{P}$ and for each $c \in \mathcal{C}$, there exists a unique key $k \in \mathcal{K}$ with $E_k(p) = c$.*

The *Vernam one-time pad* is a symmetric cryptosystem that does provide perfect secrecy. It was invented by Gilbert Vernam in 1917[1], and is defined as follows. Let $\mathcal{P} = \mathcal{C} = \mathcal{K} = \{0,1\}^n$ for some $n \in \mathbb{N}$. For $k \in \{0,1\}^n$, define

- the encryption function $E_k : \{0,1\}^n \to \{0,1\}^n$ by

$$E_k(p) = p \oplus k \bmod 2 \ , \quad \text{and}$$

- the decryption function $D_k : \{0,1\}^n \to \{0,1\}^n$ by

$$D_k(c) = c \oplus k \bmod 2 \ ,$$

[1] Slightly differing from the system here, Vernam's actual invention was a system with a finite period and hence did not provide perfect secrecy.

where $\oplus$ denotes the bit-wise addition modulo 2 (also known as XOR). The keys are uniformly distributed on $\{0,1\}^n$. Note that for each plain text p a new key k is chosen from $\{0,1\}^n$.

By Shannon's theorem, the one-time pad provides perfect secrecy, since for each plain text $p \in \mathcal{P}$ and for each cipher text $c \in \mathcal{C}$ there exists a unique key $k \in \mathcal{K}$ with $c = p \oplus k$, namely the string $k = c \oplus p$. However, the one-time pad has major disadvantages that make it impractical to use in most situations: To obtain perfect secrecy, every key can be used only once, and it must be at least as long as the plain text to be transmitted. Despite these drawbacks, for the perfect secrecy it provides, the one-time pad has been used in real-world applications such as, allegedly, for the communication between Moscow and the Russian embassy in Washington. Also, recent work by Ding and Rabin suggests that certain strategies based on one-time pads might be feasible when assuming that an adversary only has a bounded amount of storage.

Although perfect secrecy is expensive to obtain, symmetric cryptosystems would work reasonably well if the secret keys necessary for the cryptosystem could be exchanged without an eavesdropper getting to know them. The next subsection shows how to do this even when only having an insecure channel.

6.2 Protocols for secret-key agreement

Consider the situation where Alice and Bob want to exchange messages over an insecure channel such as a public telephone line. In order to keep their messages secret, they may decide to use a symmetric cryptosystem in which they both possess the same key for encryption and decryption. But then, how can they agree on a joint secret key when they can communicate only over an insecure channel? If they were to send an encrypted message containing the key to be used in subsequent communications, which key should they use to encrypt *this* message?

This paradoxical situation is known as the *secret-key agreement* problem, and it was considered to be unsolvable since the beginning of cryptography. It was quite a surprise when, in 1976, Whitfield Diffie and Martin Hellman solved this long-standing, seemingly paradoxical problem by proposing the first secret-key agreement protocol. We describe their protocol below. It was the Diffie-Hellman protocol that later inspired Rivest, Shamir, and Adleman to invent the popular RSA public-key cryptosystem. That is, Diffie et al's key idea to solve the secret-key agreement problem opened the door to modern public-key cryptography, which no longer requires sending secret keys over insecure channels.

Strangely enough, the reverse happened in the non-public sector. The Communications Electronics Security Group (CESG) of the British Government Communications Head Quarters (GCHQ) claims to have invented the RSA system prior to Rivest, Shamir, and Adleman and the Diffie-Hellman secret-key agreement scheme independently of Diffie and Hellman. And they did so in reverse order. James Ellis first discovered the principle possibility of public-key cryptography in the late sixties. In 1973, Clifford Cocks developed the mathematics necessary to realize Ellis' ideas and formulated what four years later became known as the RSA system. Soon thereafter, inspired by Ellis' and Cocks' work, Malcolm Williamson invented what became known as the Diffie-Hellman secret-key agreement scheme, around the same time Diffie and Hellman succeeded.

The Diffie-Hellman scheme

First we need some notation. The *greatest common divisor* of two integers a and b is denoted by $\gcd(a, b)$. For $n \in \mathbb{N}$, define the set

$$\mathbb{Z}_n^* = \{i \in \{1, \dots, n-1\} \mid \gcd(i, n) = 1\} .$$

The *Euler function* ϕ is defined as $\phi(n) = |\mathbb{Z}_n^*|$. Note that $\mathbb{Z}_n^*$ is a group (with respect to multiplication) of order $\phi(n)$. The following useful properties of ϕ follow from the definition:

- $\phi(m \cdot n) = \phi(m) \cdot \phi(n)$ for all $m, n \in \mathbb{N}$ with $\gcd(m, n) = 1$, and
- $\phi(p) = p - 1$ for all primes p.

In particular, $\phi(n) = (p-1) \cdot (q-1)$ if n is the product of two primes p and q.

For $n \in \mathbb{N}$, a *primitive root of* $\mathbb{Z}_n^*$ is any element $a \in \mathbb{Z}_n^*$ satisfying that, for each d with $1 \leq d \leq \phi(n)$, it holds that

$$a^d \not\equiv 1 \bmod n .$$

Equivalently, a primitive root of $\mathbb{Z}_n^*$ is a generator of $\mathbb{Z}_n^*$.

For example, $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$ and $\mathbb{Z}_4^* = \{1, 3\}$, so $\phi(4) = 2$, and the two primitive roots of $\mathbb{Z}_5^*$ are 2 and 3, because $2^1 = 2$, $2^2 = 4$, $2^3 = 3$, and $2^4 = 1$ modulo 5. Not every integer has a primitive root: 8 is the smallest such example. It is known from elementary number theory that an integer n has a primitive root if and only if n is 1 or 2 or 4 or is of the form q^k or $2q^k$ for some odd prime q.

Let p be a prime and let g be a primitive root of $\mathbb{Z}_p^*$. The function $\alpha_{(g,p)} : \mathbb{Z}_{p-1} \to \mathbb{Z}_p^*$ that is defined by

$$\alpha_{(g,p)}(a) = g^a \bmod p$$

is called the *modular exponential function with base g and modulus p*. Its inverse function, which for fixed p and g maps $\alpha_{(g,p)}(a)$ to $a = \log_g \alpha \bmod p$, is called the *discrete logarithm*.

Figure 1 shows how the Diffie-Hellman secret-key agreement protocol works. It is correct, because

$$k_A = \beta^a = g^{b \cdot a} = g^{a \cdot b} = \alpha^b = k_B .$$

Thus, the keys computed by Alice and Bob indeed are the same. Furthermore, Figure 2 shows how to transform a symmetric cryptosystem into a public-key cryptosystem using the Diffie-Hellman protocol.

Computing discrete logarithms is considered to be a hard problem. No efficient algorithms are known for solving it. However, unfortunately, it has not been possible so far to establish a connection between the hardness of solving the discrete logarithm and complexity-theoretical statements such as $P = NP$.

6.3 Asymmetric cryptosystems

The problem of using the cryptosystem in Figure 2 is that it is very vulnerable to man-in-the-middle attacks, i.e. against someone that just pretends to be Alice or Bob. Fortunately, asymmetric cryptosystems do not require to choose a new random public key for each session.

1. Alice and Bob agree upon a large prime p and a primitive root g of $\mathbb{Z}_p^*$; p and g are public.
2. Alice chooses a large number a at random and computes $\alpha = g^a \bmod p$, and Bob chooses a large number b at random and computes $\beta = g^b \bmod p$.
3. Alice sends α to Bob and Bob sends β to Alice.
4. Alice computes her key $k_A = \beta^a \bmod p$, and Bob computes his key $k_B = \alpha^b \bmod p$.

Figure 1: The Diffie-Hellman secret-key agreement protocol.

1. Alice and Bob agree upon a large prime p and a primitive root g of p; p and g are public.
2. Bob chooses a large number b at random as his private key, computes $\beta = g^b \bmod p$, and sends β to Alice.
3. Alice chooses a large number a at random, computes $\alpha = g^a \bmod p$, the key $k = \beta^a \bmod p$, and the cipher text $c = E_k(m)$, where m is the message to be sent, and sends α and c to Bob.
4. Bob computes $k = \alpha^b \bmod p$ and $m = D_k(c)$.

Figure 2: A public-key cryptosystem based on the Diffie-Hellman protocol, which uses the encryption and decryption algorithms E_k and D_k of a given symmetric cryptosystem.

Instead, both Alice and Bob may choose some fixed public/private key pair and then use the public key for authentication as well as encryption. This at least stops the man in the middle from reading the messages. The most well-known asymmetric cryptosystem is the RSA system, named after its inventors Rivest, Shamir, and Adleman. This system is supposed to be secure if the assumption that prime factorization is hard is true, i.e. it is hard to determine the prime factors of a number $n \in \mathbb{N}$. As with discrete logarithm, also for prime factorization no connection has been established yet to complexity-theoretical statements, and it is therefore not clear how hard it is to solve the prime factorization problem.

Only recently, first public-key cryptosystems have been proposed that are based on complexity-theoretical hardness assumptions. The *shortest lattice vector* problem, denoted by SVP, is the problem of finding a shortest lattice vector in the lattice generated by a given lattice basis. That is, given a set of vectors $V = \{v_1, v_2, \ldots, v_k\} \subseteq \mathbb{R}^d$, the problem is to find coefficients $c_1, c_2, \ldots, c_k \in \mathbb{Z}$ so that

$$||\sum_{i=1}^{k} c_i \cdot v_i||_2$$

is minimized, where $||v||_2 = \sqrt{\sum_i v_i^2}$ is the Euclidean length of vector v.

Ajtai and Dwork designed a public-key cryptosystem in 1997 whose cryptographic security depends on the hardness of SVP. However, the hardness of SVP had remained an open problem for a long time. Solving this problem in 1998, Ajtai established the NP-hardness of SVP under randomized reductions. His result was recently strengthened by Micciano (2001), who also simplified Ajtai's proof.

6.4 Digital signatures and zero-knowledge protocols

Apart from being able to securely transmit messages, it is also important to make sure that a message indeed came from the sender and that a sender is really the sender that it pretends to be. (Just to present the public key is not enough, since this would still allow someone to impersonate the real owner of the public key, even though he would not be able to decrypt the messages.) For the first case, we need a *digital signature scheme*, and for the second case we need an *authentication scheme*.

Digital signature schemes

To provide digital signatures is relatively easy. Suppose we have a public-key cryptosystem with the property that

$$D_d(E_e(m)) = E_e(D_d(m))$$

which is, for example, the case for the RSA system. If Bob wants to convince Alice that he really sent message m, he can send $m' = D_d(m)$ over to Alice (using, for example, Alice's public key), and Alice can then check that Bob indeed sent the message m by computing $E_e(m')$. Note that only Bob can produce $D_d(m)$ because only Bob knows the private key d.

Authentication schemes

Suppose that Alice and Bob want to exchange messages. How can Bob be certain that it is Alice he is communicating with and not someone else that may just pretend to be Alice? In other words, how can Alice prove her identity to Bob beyond any doubt? One way to achieve this goal is to assign to Alice's identity some secret information such as her PIN (Personal Identification Number) or any other private information that nobody else knows. We refer to the information proving Alice's identity as Alice's *secret*.

But here is the catch. Alice would like to convince Bob of her identity by proving that she knows her secret. Ideally, however, she should not disclose her secret because then it would not be a secret any more. If Bob, for example, knew Alice's secret, he could pretend to be Alice when communicating with somebody else. So the question is: *How can one prove the knowledge of a secret without telling the secret?* This is precisely what zero-knowledge protocols are about.

Goldreich et al. found such a zero-knowledge protocol that is based on the graph isomorphism problem.

In the *graph isomorphism* problem we are given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, and the problem is to decide whether there is a way of renaming the nodes in V_1, i.e. a permutation (or isomorphism) $\pi : V_1 \rightarrow V_1$, so that $\pi(G_1) = G_2$, where $\pi(G_1)$ is the graph resulting from this renaming on G_1.

Although it is not known whether the graph isomorphism problem is NP-hard, it is considered to be a hard problem (like prime factorization and discrete logarithm).

The zero-knowledge protocol is given in Figure 3. Alice's secret is the isomorphism π she has chosen. The protocol is correct, since Alice knows her secret π and also her random permutation ρ. Hence, she can easily compute the isomorphism σ with $\sigma(G_b) = H$ to prove her identity to Bob. Since ρ is chosen uniformly at random, σ has a uniform probability distribution, and therefore σ does not reveal any information about π.

Suppose now that some eavesdropper, say Eric, wants to cheat by pretending to be Alice. Eric does not know her secret isomorphism π, but he does know the public isomorphic graphs G_1 and G_2. Eric wants to convince Bob that he knows Alice's secret, which corresponds to (G_1, G_2). If, by coincidence, Bob's bit b equals his previously chosen bit a, Eric wins. However, if $b \neq a$, computing $\sigma = \rho \circ \pi$ or $\sigma = \rho \circ \pi^{-1}$ requires knowledge of π. Without knowing π, computing π from the public graphs G_1 and G_2 seems to be impossible for Eric, since graph isomorphism is a hard problem. Thus, he will fail provided that the graphs are chosen large enough.

Since Eric cannot do better than guessing the bit b, he can cheat with probability at most $1/2$. Hence, if Bob demands, say, k independent rounds of the protocol to be executed, he can make the cheating probability as small as 2^{-k}, and thus is very likely to detect any cheater.

1. Before the protocol, Alice chooses a large graph G_1, a random permutation π on G_1's vertices, and computes the graph $G_2 = \pi(G_1)$. (G_1, G_2) are public, π is private.

2. The protocol starts with Alice choosing a random permutation ρ on $V(G_1)$ and a bit $a \in \{1, 2\}$, computing $H = \rho(G_a)$, and sending H to Bob.

3. Bob chooses a bit $b \in \{1, 2\}$ at random, sends b to Alice, and wants to see an isomorphism between G_b and H.

4. Alice computes the permutation

$$\sigma = \begin{cases} \rho & \text{if } b = a \\ \rho \circ \pi & \text{if } 1 = b \neq a = 2 \\ \rho \circ \pi^{-1} & \text{if } 2 = b \neq a = 1 \end{cases}$$

   satisfying $\sigma(G_b) = H$ and sends σ to Bob.

5. Bob verifies that indeed $\sigma(G_b) = H$ and accepts accordingly.

Figure 3: The Goldreich-Micali-Wigderson zero-knowledge protocol for graph isomorphism.

The zero-knowledge protocol above belongs to a class of protocols called *interactive proof systems* that will be considered in more depth in the next section.

6.5 References

- Y. Z. Ding and M. O. Rabin. Hyper-encryption and everlasting security. In *19th Symp. on Theoretical Aspects of Computer Science (STACS)*, pp. 1-26, 2002.

- A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press Series on Discrete Mathematics and Its Applications, 1997.

- J. Rothe. Some facets of complexity theory and cryptography: A five-lecture tutorial. *ACM Computing Surveys* 34(4), pp. 504–549, 2002.