Lecture Notes: Computer Networks CSE1405
Cryptography and Network Security

Stefanie Roos

May 30, 2020

1 Introduction

Security deals with protecting a computer system from *intentional misbehaviour*. Thus, this chapter of the course differs from previous ones that consider *random* faults and errors.

This chapter is divided into 3 sections. In the first section, we start by introducing key definitions and describe how to properly disclose discovered security vulnerabilities. Section 2 describes cryptographic primitives that form the basis for secure protocols. In particular, we discuss encryption, which allows users to send messages without revealing the content of these messages. In addition to encryption, cryptographic also enables integrity, meaning that users can be sure that the information that they receive has not been tampered with. Section 3 then describes protocols that achieve security for different layers of the network architecture.

Achieving security entails achieving the following three properties in the presence of malicious parties:

1. *Confidentiality*: Ensure that only authorized parties can access information
2. *Integrity*: Ensure that users can verify that they have the correct information
3. *Availability*: Ensure that information and services are available when needed

These three goals are often referred to as the *CIA triad* of confidentiality, integrity, and availability. An example for confidentiality is hiding the content of a message despite adversarial parties observing the communication. Similarly, giving the receiver of the message a guarantee that the content of the message was not changed during transmission is an example of integrity. Availability comes into play when considering a webserver that has to provide its service to non-malicious participants while malicious participants overload it with fake requests. Availability is frequently the hardest goal to achieve. Often, it is impossible to give universal guarantees for availability. For instance, if an attacker has the ability to cut wires, there is only a limited number of redundant wires you have reasonably have to mitigate the attack. In this course, we focus on achieving confidentiality and integrity.

In addition to these three main goals, there are a number of less prominent goals and properties that relate to security. Some of them, like *authentication*, are specific goals that fall under the above three categories, while others, like *privacy*, share some aspects of the above goals but also contain aspects not covered by them.

Authentication refers to ensuring that a party is who they claim to be. As such, authentication falls under integrity, because it ensures that information regarding a party's identity is correct. An example for an authentication method is passwords. Note that authentication differs from *identification*, which refers to providing an identity such as a username. Identification on its own does not require a proof of identity, which is why it is usually applied together with authentication.

Privacy is frequently associated with security, despite the fact that it is a different and sometimes even opposing concept. Privacy refers to an individual's ability to remaining in control over their personal information. Thus, privacy is closely related to all three security goals: Confidentiality naturally includes hiding personal information, while integrity includes ensuring that personal information is not tampered with. As being in control of personal information does imply the right to publish such information as well as hiding it, privacy also has a link to availability. However, there are aspects to privacy that are not covered by security. For instance, the confidentiality aspect of security ensures that only authorized parties can access information. However, security measures do not generally impose guidelines on how authorized parties deal with the information they have access to. In contrast, privacy aims to provide control over personal information even after it has been shared. Similarly, there are security issues that do not relate to privacy such as keeping business secrets hidden from competitors. Indeed, security and privacy can be opposing goals. In the content of computer networks, an example is keeping records of IP addresses from clients that accessed a server. Since IP addresses might be traced back to people, keeping a record of activities associated by an IP can be a violation of their privacy. However, keeping such records allows servers to detect potentially malicious behavior, e.g., starting a high number of requests from the same IP to overload the server.

A key aspect of privacy research is *anonymity*, i.e., hiding the identity of an individual. Anonymity is hence an aspect of confidentiality, though it is harder to achieve than only hiding the content of message. In the context of the Internet, anonymity implies hiding IP addresses, which is non-trivial given that Internet routing heavily relies on those addresses.

Another concept often associated with security is *trust*. Trust is a multi-faceted concept without a universal definition. For the purpose of this work, we only use trust in the sense it is used by cryptographic protocols: A *trusted* party is a party that will not behave maliciously. As such, it is not necessary to protect a network against misbehavior by such a party.

Last, note that the English language[1] makes a difference between security and *safety*. Safety refers to protection against random accidents while security implies the existence of malicious intent. For instance, an error detection code is likely to detect a transmission error caused by an unreliable transmission medium. However, a malicious party will simply change the value of the detection code when modifying the message. Thus, error detection codes are a safety measure but do not provide security.

Sources The lecture notes are partially inspired by Module 1, 5, and 7 of the course 'Computer Security and Privacy' [1] at University of Waterloo, in particular the version taught by Stefanie Roos and Urs Hengartner in Fall 2017. Furthermore, the books 'Security in Computing' [6] by Charles and Shari Pfleeger and 'Introduction to Cryptography' [3] by Johannes Buchmann were extremely helpful in compiling these notes. Additional sources for individual topics are cited in the corresponding sections.

1.1 Disclosing Security Vulnerabilities

When identifying security problems in real-world software, it is important to research these vulnerabilities in a legally and ethically responsible manner. In this section, we first discuss the legal situation with regard to hacking into someone's computer. Then, we discuss how to disclose discovered security vulnerabilities.

Legal Situation Circumventing protection mechanisms of a network or computer without the explicit permission of the owner constitutes a crime in most countries around the world. In the Netherlands, the consequences may include fines of up to several thousands of euros or up to four years in prison. The severity of the punishment depends on whether any confidential data was downloaded and whether the system was damaged by the intruder's actions. If the actions are linked to terrorism, more severe punishments are possible [5].

Disclosure In contrast to intrusion, checking someone's software or hardware for vulnerabilities on your own machines or on request by the owner is not punishable by law. However, revealing any discovered vulnerabilities to the public is a sensitive issue. In the following, we denote the party who developed a program with a vulnerability as the developer. The party who discovered the vulnerability is called the investigator.

There are three approaches to disclosure:

- Non-disclosure: Not disclosing a security vulnerability (or only disclosing it to the developer without following up, even if the developers do not fix the issue) does generally not entail legal consequences. Indeed, sometimes contracts prevent employees from disclosing vulnerabilities publicly. Legal consequence for non-disclosure are only possible if i) the person was responsible for finding and disclosing a vulnerability, and ii) the non-disclosure had far-reaching consequences such as loss of life. However, even in the absence of legal consequences, not disclosing a known vulnerability is ethically questionable. It leaves users vulnerable to be exploited by a malicious party.
- Full disclosure: The investigator announces the vulnerability publicly, e.g., via a mailing list, as soon as it is discovered without consulting the developers first. Full disclosure ensures that users are informed of the vulnerability as fast as possible. However, it also informs malicious parties who might have been unaware of it before. These parties can now abuse the vulnerabilities as it has not yet been fixed.
- Responsible (or coordinated) disclosure: In a responsible disclosure process, the investigator first informs the developer only. The investigator sets a deadline until which the vulnerability has to be fixed and announced by the developer. If the developer does not adhere to this deadline, the investigator reveals the vulnerability. The investigator may involve Computer Emergency Response Teams (CERTs) that have experience in setting deadlines and dealing with developers. In the Netherlands, there are five CERTs, each handling different security-critical businesses[2]. Responsible disclosure gives the developer time to fix the issue without immediately informing malicious parties.

There is some discussion on whether responsible or full disclosure is more ethical. Most of the recent disclosures of severe vulnerabilities have followed the responsible disclosure process.

[1] in contrast to Dutch, which uses 'veiligheid' for both safety and security

[2] `https://www.cert.nl/english`

1.2 Attacker model

An *attacker or adversary model* denotes a description of the goals and capabilities of malicious parties. Generally, a system protects against all malicious parties following a certain adversary model. Protection against malicious parties with capabilities outside of the adversary model is not guaranteed. As the provided guarantees should hold for *all attacks* an adversary with the defined capabilities can come up with, attacker models do not include attack strategies.

The need for adversary models arises from the fact that it is impossible to protect an omnipotent adversary, meaning that security researchers have to define limitations on the adversary. These limitations depend on the concrete scenario and should mirror the real-world situation.

The following terminology is typically used to characterize adversaries. The first four terms — internal, external, global, and local — describe the positioning of the attacker in a network. The next four terms — passive, active, static, and adaptive — describe adversarial behaviour while the last two — computationally unbounded and polynomially bounded — characterize the computational power of the attacker.

- *internal*: An internal adversary controls participants in a system. A corrupt server is an example of an internal adversary.
- *external*: In contrast to an internal adversary, an external adversary observes the system from the outside. An example is a malicious party listening on the wire.
- *global*: An attacker that can observe or control the complete system. For instance, an Internet service (ISP) can observe all the links and routers within its autonomous system (AS). So, if the scenario is restricted to the AS, we would characterize the ISP as a global adversary.
- *local*: An attacker that can observe or control only a part of the system. In the previous example, the ISP is a local adversary if the scenario considers the complete Internet.
- *passive*: A passive adversary does not manipulate the protocol. An external adversary listening to the wire without modifying any transmission is a passive adversary. A second example is a malicious service provider who follows the protocol but additionally tries to gain confidential information. The latter is also called an *honest-but-curious* attacker.
- *active*: An active adversary manipulates information. An adversary replacing a message on the wire with a modified message is an example of an active adversary.
- *static*: A static adversary does not change its behaviour over time. For instance, an attacker in a P2P network who always chooses a random set of peers to establish connections with is a static adversary.
- *adaptive*: An adaptive attacker changes its behaviour dynamically based on its observations. In the previous example, an adaptive attacker might start connecting to the most active nodes rather than a random set to observe more of the overall traffic.
- *computationally unbounded*: The attacker has unlimited computational power at their disposal. In practice, such an attacker is unrealistic but constitutes a worst-case assumption.
- *polynomially bounded*: An attacker with polynomially bounded computational power can only compute algorithms that have polynomial complexity. As algorithms are indeed infeasible to compute if they have higher complexity and sufficiently large input sizes, even with all of today's computational power, all realistic attackers are polynomially bounded.

Cryptographic algorithms that protect against computationally unbounded provide *information-theoretic security* while algorithms protecting against polynomially bounded adversaries provide *computational security*.

In the context of computer networks, the most common adversary model is the Dolev-Yao model, which defines a polynomially bounded active adversary who can overhead, discard, create, modify, delay, and replay messages.

2 Cryptography

Traditionally, cryptography is equated with encryption, i.e., turning meaningful text into seemingly random characters. However, today's cryptographic algorithms are also concerned with integrity. They ensure that intentional changes to information can be detected.

Formally, a *cryptosystem* describes a set of algorithms with guaranteed security properties. *Cryptographic keys* determine which concrete algorithms from the set to use. *Kerckhoff's principle* states that the security of a cryptosystem should not depend on the secrecy of the algorithms. Rather, only the keys should remain secret. As changing keys is

easy, accidentally revealing a key does not prevent further use of the cryptosystem. In contrast, if the security relies on the actual algorithms remaining secret, leakage of the algorithms implies that a complete new set of algorithms must be designed, which is a time-consuming process.

The probably simplest example of cryptosystem is the Caesar cipher, an encryption algorithm dating back to the ancient Romans. For an alphabet $\mathbb{A}$ of n letters, the Caesar cipher substitutes every letter with a different letter. The replacement letter is chosen such that the i-th letter of the alphabet is replaced by letter $i + k \mod n$ for some $k \in \{0, \ldots, n-1\}$. The original text can then be reconstructed by replacing the j-th letter of the alphabet with letter $j - k \mod n$. Here, k is the cryptographic key and the cryptosystem is the algorithm for substituting the letters. When using the English alphabet and the key $k = 3$, the word 'pterodactyl' becomes 'swhurgdfwbo'.

Cryptography is the research area concerned with creating cryptosystems whereas *cryptanalysis* denotes research into breaking cryptosystems. Breaking a cryptosystems means that an attacker can undermine the goal of the cryptosystem without being given the key. For an encryption algorithm, breaking the algorithm implies that the attacker can reconstruct (parts of) the original text. When cryptography aims to achieve integrity, breaking the algorithm implies allowing an attacker to change information without being detected. The combined research of cryptography and cryptanalysis is called *cryptology*.

Note that the Caesar cipher presented above is not secure and an attacker can easily reconstruct the original text. A particular weakness of the Caesar cipher is that it preserves the frequency distribution of letters. More precisely, the encrypted text has the same frequency counts as the original but for different letters. With a high probability, the most frequent letter in the encrypted text is the substitution of the most frequent letter in the language the original text has been written in. An attacker can hence obtain the key k by computing the difference between the most frequent letter in the real language and the most frequent letter in the encrypted text in terms of their position in the alphabet. For instance, the most frequent letter in 'vuljhuulclyohclluvbnozvjrz' is 'l' whereas the most frequent letter in the English language is 'e'. Shifting 'e' to 'l' is a shift by 7. Setting $k = 7$ gives 'onecanneverhaveenoughsocks'. As the Caesar cipher is not secure, we require more complicated protocols, which we will discuss in the remainder of the chapter.

2.1 Encryption: Setting and Terminology

A cryptosystem used for encryption is called a *cipher*. Ciphers make use of three algorithms: the key generation algorithm $KeyGen$, the encryption algorithm Enc, and the decryption algorithm Dec. We focus on the latter two here. The encryption algorithm takes a key $K1$ and the message M, called the *plaintext*, as input. We write $Enc_{K1}(M)$ or $Enc(K1, M)$ to denote the encryption of M using $K1$. The encrypted text is called the *ciphertext*. The decryption algorithm takes a key $K2$ and a ciphertext C as input and computes the corresponding plaintext (or an error message if the ciphertext is not a valid ciphertext). We write $Dec_{K2}(C)$ or $Dec(K2, C)$.

Encryption assumes a passive adversary Eve who aims to break confidentiality. Typically, Eve listens on the connection between two parties who exchange confidential information. If she manages to obtain any information about the content of any message that she did not have before, Eve's attack is considered successful. The only information Eve is allowed to learn is the length of the messages.

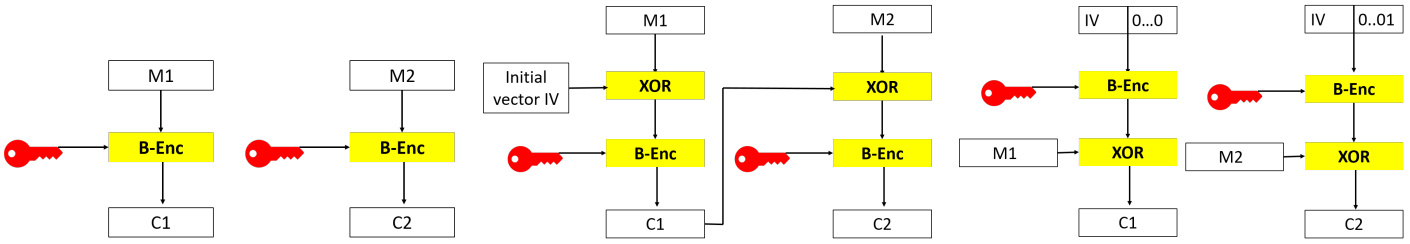
More specifically, let M be a message and $prior(M)$ give the probability that Alice sends M to Bob. *prior* is Eve's knowledge before observing the communication between Alice and Bob. If an encryption algorithm is secure, Eve's posterior distribution *post* after observing Alice sent a ciphertext C differs only negligibly from $prior_{|length(C)}$, i.e., the prior distribution given that the ciphertext is of a certain length. Let's consider one concrete example. Alice sends one encrypted bit, so 0 or 1, to Bob. If Eve thinks both are equally likely before observing the ciphertext, she should still think they are approximately equally likely after observing the ciphertext. Typically, some messages, e.g., those corresponding to text that makes sense, are more likely than others, which means from Eve's point of view, there can be messages that are more likely even for a secure encryption algorithm. However, observing the ciphertext should not help Eve in making a better guess than she could have made without observing the ciphertext.

The two main approaches to encryption are symmetric-key encryption, which uses $K1 = K2$, and asymmetric-key encryption, which uses $K1 \neq K2$. In the next subsections, we explain the idea of both concepts and introduce the most common algorithms of each class.

2.2 Symmetric-key encryption

Symmetric-key encryption uses one key $K = K1 = K2$ for encryption and decryption. Two parties Alice and Bob have to agree on K and keep it secret. We say that Alice and Bob have to communicate via a *secure channel* to ensure the secrecy of the key. One possibility of a secure channel is meeting in person. More commonly, asymmetric-key encryption enables a secure channel, as we will discuss in Section 2.3. Symmetric-key encryption is also referred to as *secret-key encryption* and is a subfield of *secret-key or symmetric-key cryptography*.

Two dimensions classify the type of a symmetric-key cipher. The first dimension describes the manner they process plaintexts. Here, we distinguish *stream* and *block ciphers*. Stream ciphers act on each character of the plaintext indi-



(a) Electronic Codebook (b) Cipherblock Chaining (c) Counter

Figure 1: Modes of operation

vidually whereas block ciphers divide the plaintext in blocks of equal length and then act on each block. The second dimension categorizes the manner the encryption modifies the plaintext. We distinguish *substitution* and *transposition* (or *permutation*) *ciphers*. A substitution cipher replaces a character or block with another character or block whereas a transposition cipher changes the order of characters. The Caesar cipher is an example of both a stream and a substitution cipher.

In modern cryptography, transposition ciphers are only used in combination with substitution. As transposition ciphers maintain the frequency of letters of the plaintext, they can usually be broken by using certain linguistic properties. At the very least, Eve can determine if a certain word was not part of the plaintext if she cannot find all letters of the word in the ciphertext. However, transposition ciphers used after or before substitution ciphers can enhance the security.

Stream Ciphers As stated above, a stream cipher acts on each character individually. Typically, the characters are bits rather than letters to allow for a generic binary data.

The archetype of the stream cipher is the One-Time Pad (OTP). Alice and Bob exchange a key K via a secure channel that has at least the same length as the message they want to send. At a later point in time, Alice wants to share M with Bob. She computes $C = M\ XOR\ K$ and sends C to Bob. If Bob receives C unmodified, he can compute the plaintext $M = C\ XOR\ K$.

OTP is information-theoretically secure as any message M' of length $length(M)$ can be encrypted to C with a key $K' = C\ XOR\ M'$. Hence, Eve cannot gain any information about the plaintext based on the ciphertext besides the length.

However, sharing a key of the same length as the message via a secure channel is infeasible for most scenarios. Using the same key K to encrypt two messages M_1 and M_2 creates security issues. By computing the XOR of the corresponding ciphertexts $C_1 = M_1\ XOR\ K$ and $C_2 = M_2\ XOR\ K$, we get $C_1\ XOR\ C_2 = M_1\ XOR\ M_2$. As the two messages are not random, one can likely gain some information about them from their XOR. For instance, if Eve correctly guesses that M_1 starts with 'hello', she can now tell the first letters of M_2.

Stream ciphers that are actually in use today apply a modified version of OTP. These ciphers are not information-theoretically secure but allow for shorter and reusable keys. They should be computationally secure.

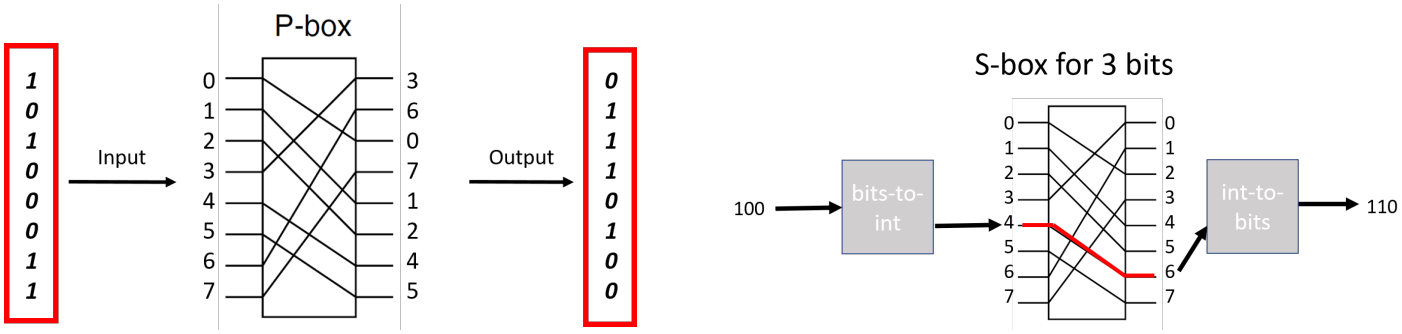
Modified versions of OTP rely on a pseudo-random number generator $PRNG$. Given a short input S called seed, $PRNG(S)$ outputs a bit stream of arbitrary length. For a computationally bounded attacker, it should be impossible to i) differentiate the output of $PRNG$ from a randomly created bit stream, and ii) derive the seed S used to generate the bit stream. Note that $PRNG$ is deterministic in S, i.e., S always entails the same output.

The encryption algorithm now leverages $PRNG$ as follows. Alice and Bob agree on a key K whose length corresponds to the seed length of $PRNG$. If one of them, without loss of generality Alice, wants to send a message M, she chooses a random *nonce* n of the same length as the key. A nonce is a random number or character string used only once for randomization, i.e., the nonce ensures that the same keystream is not used twice even if the same key is used multiple times. She then computes $S = K\ XOR\ n$ as the seed for $PRNG$. Afterwards, she obtains the first $length(M)$ bits KS of the keystream $PRNG(S)$ and computes $C = M\ XOR\ KS$. She sends both C and n to Bob.

For decryption, Bob also computes the XOR of K and n. From that, he derives the corresponding key stream KS using the same pseudo-random number generator $PRNG$. He decrypts $M = C\ XOR\ KS$.

Most stream ciphers used in practice follow the above idea. They differ in the pseudo-random number generator they use. Traditionally, RC4 was the most common stream cipher, but various security weaknesses of RC4 have been discovered over the years. Hence, RC4 is in the process of being replaced with more secure options such as ChaCha or Salsa20.

Block ciphers A block cipher operates on blocks of a fixed length l. The plaintext is divided into blocks such that the i-th block M_i corresponds to bit $(i-1)*l+1$ to $i*l$ (starting to count at 0 from the left). Padding ensures that the plaintext length is a multiple of l. The encryption is determined by block encryption and decryption algorithms B-Enc and B-Dec as well as the *mode of operation*, which determines how blocks are combined.



(a) P-box (b) S-box

Figure 2: Components of block ciphers

We start by introducing three common modes of operation: *Electronic Codebook* Mode, *Cipherblock Chaining* Mode, and Counter *Mode*, as depicted in Figure 1. For all of them, assume that Alice and Bob agreed upon a shared key K using a secure channel.

Electronic Codebook Mode is the most straight-forward mode. Here, K is the key for B-Enc and each block M_i is encrypted to $C_i = \text{B-Enc}(K, M_i)$. The complete ciphertext is the concatenation of blocks $C_1, C_2, \ldots, C_n$ for n plaintext blocks. Decryption then computes $M_i = \text{B-Dec}(K, C_i)$.

Electronic Codebook Mode reveals pattern in the plaintext and hence leads to security issues. More precisely, blocks that are identical have an identical ciphertext, so it is possible to detect which parts of the plaintext are the same. Knowing one of the plaintext blocks implies that other blocks with the same ciphertext are also known to the attacker.

Cipherblock Chaining Mode is a mode that ensures that identical plaintext blocks have different encryptions. For this purpose, Alice picks a random initial vector IV of length l, which serves a similar purpose as a nonce in that it randomizes the encryption. For the first plaintext block M_1, Alice computes the XOR of IV and M_1. She then encrypts $C_1 = \text{B-Enc}(K, M_1 \; XOR \; IV)$. For any plaintext block M_i with $i > 1$, Alice repeats the process using the ciphertext block C_{i-1} in place of IV. The complete ciphertext C is then the concatenation of the blocks. Alice sends both C and IV to Bob. Bob decrypts the ciphertext as follows:

$$M_i = \text{B-Dec}(K, C_i) \; XOR \begin{cases} IV, & \text{if i=1} \\ C_{i-1}, & \text{otherwise} \end{cases} .$$

Cipherblock chaining resolves the security issues of the Electronic Codebook Mode but the encryption cannot be parallelized.

In contrast, Counter mode is both secure and parallelizable. It assumes that the block length l is even. Counter mode requires an initial value IV of length $l/2$ in contrast to Cipherblock Chaining whose initial vector is of length l. To encrypt the block M_i, Alice concatenates IV with the $l/2$-bit binary representation of $i-1$, e.g., a binary string of $l/2$ 0s for $i = 1$. She then encrypts the concatenation with key K and derives C_i as the XOR of the encryption and M_i. Formally, let $(i)_2^{l/2}$ denote the binary representation of i and $||$ denote concatenation. Then, the i-th ciphertext block is

$$C_i = M_i \; XOR \; \text{B-Enc}(K, IV||(i-1)_2^{l/2}).$$

After encrypting and concatenating the ciphertext blocks, Alice sends C and IV to Bob. For decryption, Bob consequently switches the roles of M_i and C_i:

$$M_i = C_i \; XOR \; \text{B-Enc}(K, IV||(i-1)_2^{l/2}).$$

There are additional modes of operation. Some of them combine encryption with integrity measures.

Now, it remains to discuss the nature of the block encryption and decryption algorithms B-Enc and B-Dec. Block encryption is a combination of multiple *P-boxes*, which correspond to transposition ciphers, and *S-boxes*, which correspond to substitution ciphers.

A P-box takes a binary input of length l and changes the order of the bits according to a fixed permutation. As depicted in Figure 2a, a graphical representation of a Pbox is a box with l incoming and outgoing lines enumerated 0 to $l-1$. Each incoming line is connected to exactly one outgoing line. A connection between the i-th incoming line and the j-th outgoing line indicates that the i-th bit of the input is the j-th bit of the output. Encryption hence maps all bits of the input to the corresponding output positions. Decryption reverses the process by mapping outgoing lines to incoming lines.

In contrast, an S-box maps a l-bit input to a different l-bit output using a box with 2^l incoming and outgoing lines, as shown in Figure 2b. As for P-boxes, each incoming line is connected to exactly one outgoing line. The binary input is first encoded as a number r between 0 and $2^l - 1$. Following the r-th incoming line to the corresponding outgoing line gives us a different number q. Decoding q to a binary string of length l gives the encryption. Decryption again reverses the process by treating outgoing lines the same way encryption treats incoming lines.

Block encryption algorithms apply multiple P-boxes and S-boxes, possibly repeating the process several times, to guarantee security. The first widely used block encryption algorithm was DES, Data Encryption Standard, which uses 16 iterations of a combination of S-boxes and P-boxes. However, DES' key length is too short to protect against today's computational power. DES uses keys of length 64 and block length $l = 64$. Yet, 8 of these bits are redundant, so that the actual key length is 56, which can be solved by bruteforce, i.e., trying all combinations, within acceptable time.

A solution that overcomes the short key length of DES is Triple-DES. Triple-DES uses three keys K_1, K_2, and K_3. For encryption, it first executes a DES encryption using K_1, followed by a DES decryption with K_2, and a DES encryption with K_3. The reason for choosing a decryption as the second step is backward compatibility: If a communication partner does only use DES and not Triple-DES, executing Triple-DES with $K_1 = K_2 = K_3$ gives the same result as a simple DES encryption. By construction, Triple-DES also acts on blocks of length $l = 64$ but has a key length of 192 bits in total and an effective key length, i.e., without redundant bits, of 168.

The successor of DES is AES, Advanced Encryption Standard, which won a competition for the best proposal of a block cipher in 2001. AES' design is public and it is implementable in both software and hardware. Furthermore, it supports various key lengths, namely 128, 192, and 258.

All symmetric-key encryption algorithms share the need for a secure channel to exchange keys. Asymmetric-key encryption can create such a channel and hence make encryption feasible in modern settings, where communication partners usually do not meet in person. Usually, asymmetric-key encryption is applied for the exchange of keys whereas the actual communication then uses symmetric-key encryption with the previously exchanged keys.

2.3 Asymmetric-key Encryption

Asymmetric-key encryption uses different keys for encryption and decryption, respectively. The encryption key is called the public key and denoted K_{pub}. The decryption key is only known to its owner and denoted by K_{pri}. We call K_{pub} the *public key* and K_{pri} the *private* or *secret key*. Asymmetric-key encryption is also referred to as *public-key encryption* and is a subfield of *public-key cryptography*.

We will first discuss the general principle of asymmetric-key encryption. Then, we explain RSA, the most common asymmetric-key encryption algorithm, followed by two examples of using asymmetric-key encryption for exchanging a shared secret key.

When Bob wants to receive encrypted messages, he first generates a *key pair* consisting of a public key K_{pub} and a corresponding private key K_{pri}. Corresponding here means that messages encrypted with K_{pub} can only be decrypted using K_{pri}. Bob publishes K_{pub} such that others can find it, e.g., on a *key server* dedicated to storing people's public keys.

If Alice wants to send Bob an encrypted message, she first looks up his public key. After obtaining K_{pub}, she encrypts the plaintext M with K_{pub} to get the ciphertext $C = Enc(K_{pub}, M)$. Alice sends the ciphertext to Bob and Bob decrypts $M = Dec(K_{pri}, C)$. The key challenges lies in finding an algorithm such that the attacker can not derive the private key K_{pri} from the known public key K_{pub}.

RSA *RSA*, named after its inventors Rivest, Shamir, and Adleman, is both the oldest and most used asymmetric-key encryption algorithm. RSA's security relies on the difficulty of factorizing large numbers. Its functionality follows from Euler's theorem, a centuries-old result in number theory that did not have any practical applications before the advent of modern cryptography. Note that we use $=$ for congruence, i.e., for indicating that two integers are equal modulo a number.

Theorem 1. ***[Euler's Theorem]*** *Let n be an integer. Let further $\phi(n)$ be the number of integers k such that $1 \leq k < n$ with greatest common divisor $gcd(n, k) = 1$. For all integers a with $gcd(a, n) = 1$, we have*

$$a^{\phi(n)} \mod n = 1. \tag{1}$$

RSA uses the above result for a number $n = p \cdot q$ with p, q being prime. Then $gcd(n, k) = 1$ for all k that are not multiples of either p or q. The total number of integers k with $1 \leq k < n$ is $n - 1$. Of these $n - 1$ numbers, $q - 1$ are multiples of p and $p - 1$ are multiples of q. Hence, $\phi(n) = n - 1 - (q - 1) - (p - 1) = p \cdot q - (q + p) + 1 = (p - 1) \cdot (q - 1)$.

When generating a key pair, Bob chooses two large primes p and q, typically of length at least 1024. Nowadays, 2048 bits are common. He computes n and $\phi(n)$. Afterwards, he chooses an integer d with $gcd(\phi(n), d) = 1$ and uses

the Extended Euclidean algorithm to derive the multiplicative inverse modulo $\phi(n)$, i.e., an integer e such that $d \cdot e \mod \phi(n) = 1$. In other words,

$$d \cdot e = y \cdot \phi(n) + 1 \tag{2}$$

for an integer y. Bob now publishes $K_{pub} = (e, n)$ as the public key and keeps $K_{pri} = (d, p, q)$ as his private key.

When Alice encrypts a plaintext M to Bob, she first encodes the plaintext as a sequence of numbers $i_1, \ldots, i_m$ with $i_j \in \{1..n\}$. for all $j = 1..m$, she computes $x_j = i_j^e \mod n$. Last, she sends $x_1, \ldots, x_m$ to Bob.

Bob computes $x_j^d \mod n$, which is equal to the original i_j because of Eq. 1 and Eq. 2. In more detail,

$$\begin{aligned}
x_j^d \mod n &= (i_j^e)^d \mod n \\
&= i_j^{e*d} \mod n \\
&= i_j^{y*\phi(n)+1} \mod n \text{ (by Eq. 2)} \\
&= (i_j^{\phi(n)})^y \cdot i_j \mod n \\
&= ((i_j^{\phi(n)} \mod n)^y \mod n) \cdot (i_j \mod n) \mod n \\
&\text{(as } a \cdot b \mod n = (a \mod n) \cdot (b \mod n) \mod n) \\
&= 1^y \cdot i_j \mod n \text{ (by Eq. 1)} \\
&= i_j
\end{aligned}$$

Table 1 displays an example.

In practice, RSA is not used in the above manner, which is often referred to as *textbook RSA*. In the above description, the encryption is deterministic, indicating that identical messages always lead to identical ciphertexts. In real-world implementations, a nonce is applied for randomization, as we have seen for symmetric ciphers in Section 2.2.

We have shown that RSA encryption works in the sense that the decryption indeed recovers the plaintext. It is computationally secure under the assumption that it is computationally unfeasible to i) factorize n, and ii) find the e-th root of an arbitrary number modulo n, i.e., derive i_j from $i_j^e \mod n$. There is good evidence that both i) and ii) are true for traditional computers, but quantum computers in theory enable algorithms for fast factorization. Hence, there is a lot of research effort dedicated to developing alternative algorithms that can withstand quantum computers.

Encryption				Decryption		
Plaintext	i_j	$i_j^3 \mod 33$		x_j	$x_j^7 \mod 33$	Plaintext
I	9	3		3	9	I
	0	0		0	0	
A	1	1		1	1	A
M	13	19		19	13	M
	0	0		0	0	
B	2	8		8	2	B
A	1	1		1	1	A
T	20	14		14	20	T
M	13	19		19	13	M
A	1	1		1	1	A
N	14	5		5	14	N

Table 1: Textbook RSA example with $p = 3$, $q = 11$, $n = 33$, $\phi(n) = 20$, $d = 7$, $e = 3$;
Each letter of the alphabet is encoded as its position in the alphabet and whitespace is encoded as 0.

A key issue of RSA is that it requires long keys and computationally expensive operations such as exponentiation. These two aspects entail slow computation, especially for long plaintexts. As a consequence, many algorithms use RSA only for key exchange and use symmetric encryption for the actual plaintext.

Hybrid encryption Hybrid encryption combines an asymmetric-key encryption algorithm with a symmetric-key algorithm. Let Enc^A and Dec^A denote the encryption and decryption algorithm for the asymmetric-key encryption and Enc^S and Dec^S the encryption and decryption algorithm for the symmetric-key encryption.

If Alice wants to send a message M to Bob, she

1. looks up Bob's public key K_{pub},

2. chooses a key K for the symmetric-key encryption,
3. encrypts the key K with Bob's public key: $C_1 = Enc^A(K_{pub}, K)$,
4. encrypts the message M with K: $C_2 = Enc^S(K, M)$, and
5. sends $C = (C_1, C_2)$ to Bob.

Knowing C, Bob

1. decrypts the key K using his private key K_{pri}: $K = Dec^A(K_{pri}, C_1)$, and
2. decrypts M using K: $M = Dec^S(K, C_2)$.

In this manner, Alice and Bob only require expensive asymmetric key encryption to agree upon a key for a more efficient symmetric-key cipher.

Diffie-Hellman Key Exchange An alternative method for exchanging keys is the *Diffie-Hellman (DH) key exchange*. In contrast to hybrid encryption, DH is interactive and hence both parties need to be online.

Similar to RSA, Diffie-Hellman relies upon the assumption that a mathematical problem can not be solved using polynomial-time algorithms. The problem underlying DH is the *Discrete Log Problem*: Consider the group $\mathbb{Z}_n$, i.e., the integers modulo n, and let $g \in \mathbb{Z}_n$ be a generator of $\mathbb{Z}_n$, i.e., $\mathbb{Z}_n = \{g^k : k \in \{1..n\}\}$. The Discrete Log Problem is: given g and $g^a \mod n$, find $a \in \mathbb{Z}_n$. Solving the Discrete Log Problem is assumed to be computationally unfeasible. Diffie-Hellman key exchange relies on a slight modification of this assumption, the so called *computational Diffie-Hellman (CDH) assumption*: It states that given only n, g, $g^a \mod n$, and $g^b \mod n$ with $a, b \in \mathbb{Z}_n$, it is computationally unfeasible to derive $g^{ab} \mod n$.

The DH key exchange now utilizes this assumption. Assume that Alice and Bob want to agree on a key in an interactive manner and n and g are publicly known. Without loss of generality, Alice starts the key exchange by choosing $a \in \mathbb{Z}_n$ and sending $A = g^a \mod n$ to Bob. Similarly, Bob chooses $b \in \mathbb{Z}_n$ and sends $B = g^b \mod n$ to Alice. Alice now computes $K = B^a \mod n = g^{ba} \mod n$ and Bob computes the same K as $K = A^b \mod n = g^{ab} \mod n$. Now, Alice and Bob have a shared secret key, which they can use for a symmetric-key encryption algorithm (after using a publicly known encoding scheme to get a key of the form required by the symmetric-key cipher).

We have discussed both symmetric and asymmetric encryption algorithms, which achieve confidentiality against an eavesdropper Eve. However, encryption does not prevent an active attacker Malory from modifying messages. Next, we discuss how to use cryptographic methods to achieve integrity.

2.4 Message Authentication Codes

A *message authentication code (MAC)*, sometimes also called a message integrity code (MIC), appends a tag to a message. The tag provides the following properties:

- Message integrity: If Alice sends a message to Bob, Bob can detect any modification made to the message by a third party.
- Authentication: If Alice sends a message to Bob, Bob can be sure that Alice sent the message.
- Repudiation: Nobody, including Alice and Bob, can prove that Alice sent a message to Bob (based on the message and the corresponding tag) to a third party.

Note that while authentication and repudiation seem contradictory on the first glance, the key difference is that authentication is concerned with Alice and Bob whereas repudiation is concerned with outsiders. So, while Bob can be sure that Alice sent the message, he is unable to prove that to another party.

Message authentication codes rely on symmetric-key or secret-key cryptography. Formally, a message authentication code consists of a key generation algorithm $KeyGen$ and an algorithm MAC that takes a key and a message as input.

Alice and Bob agree upon a secret key K using a secure channel. When Alice wants to send a message M, she computes the tag $MAC(K, M)$. She then sends both M and $MAC(K, M)$ to Bob.

Bob computes $MAC(K, M)$ himself and then checks if his result is identical to the received tag. If that is the case, Bob assumes that the message has not been tampered with and is indeed from Alice. Otherwise, Bob knows that the message has been changed and discards it.

In order to achieve integrity, it has to be computationally infeasible for the attacker to find a message N such that $MAC(K, N) = MAC(K, M)$. Otherwise, Malory could exchange a message M with N without being detected. In order to achieve authentication, it has to be computationally infeasible for the attacker to compute $MAC(K, N)$ for a message N

without knowing the key. Repudiation follows from the fact that both Alice and Bob know the key K. As a consequence, nobody can tell if a message and the corresponding tag is from Alice or Bob.

Message authentication codes internally use *cryptographic hash functions* to achieve integrity. Like a hash function, a cryptographic hash function maps inputs of arbitrary length to fixed-length outputs. In addition, a cryptographic hash function h has the following three properties:

- Collision-resistance: It is computationally infeasible to find $x \neq y$ with $h(x) = h(y)$.
- Preimage resistance: Given z, it is computationally infeasible to find x with $h(x) = z$.
- Second preimage resistance: Given $x, z = h(x)$, it is computationally infeasible to find $y \neq x$ with $h(y) = z$.

Common cryptographic hash functions are MD5, SHA1, SHA2, and SHA3. For both MD5 and SHA1, collisions have been found, therefore they should no longer be used.

The manner in which message authentication codes use cryptographic hash functions depends on the construction principle of the message authentication code. Common constructions are Merkle-Damgaard and HMAC. Furthermore, block ciphers can include MACs, usually in combination with an encryption algorithm.

Message authentication codes are useful to assure Alice and Bob that Malory does not tamper with their communication. Due to the property of repudiation, they are not applicable when someone has to prove that a communication took place. We discuss next how to provide non-repudiation, i.e., the ability to prove that a certain party send a message.

2.5 Digital Signatures

Similar to MACs, digital signatures append tags to messages. They provide the following properties:

- Message integrity: If Alice sends a message to Bob, Bob can detect any modification made to the message by a third party.
- Authentication: If Alice sends a message to Bob, Bob can be sure that Alice sent the message.
- Non-repudiation: Based on a message and the corresponding tag by Alice, everyone can prove that Alice sent the message.

Digital signatures use asymmetric-key cryptography to achieve the above properties. They rely on three algorithms, a key generation algorithm $KeyGen$, a signing algorithm $Sign$, and a verification algorithm $Verify$. $Sign$ takes a key and a message as input and outputs a tag. $Verify$ takes a key, a message, and a tag as input and outputs a binary value. Alice first generates a key pair consisting of a public *verification key* K_{pub} and a private signature key K_{pri}. She then publishes K_{pub}.

When Alice sends a signed message to Bob, she computes the *signature* $s = Sign(K_{pri}, M)$ for her message M. She sends M and s to Bob. After retrieving Alice's public verification key, Bob (and everyone else) can compute $Verify(K_{pub}, M, s)$. If s is indeed a signature of Alice for M, the verification function returns *true*. Otherwise, it returns $false$ to indicate tampering.

In order to achieve message integrity, authentication, and non-repudiation, the digital signature algorithm has to guarantee that it is computationally infeasible to compute a signature without knowledge of K_{pri}. RSA achieves the desired properties.

Hash-then-sign Like asymmetric-key encryption, digital signatures are slow. In contrast, cryptographic hash functions are fast to compute. Thus, *hash-then-sign* is a method to speed up signing and verification. Instead of directly signing a message, Alice signs $h(M)$ for a publicly known cryptographic hash function h. After receiving the message and signature, Bob also computes $h(M)$ and then executes the verification with $h(M)$ rather than M.

Note that all algorithms discussed up to now fail to answer the following question: How does Bob (or anyone else) know that they have the correct public key for Alice? What if Malory replaces Alice's key with her own?

2.6 Public Key Infrastructure

A *public key infrastructure (PKI)* ensures that adversarial parties can not easily replace the keys of other users with their own.

Hierarchical PKIs PKI typically use certificate authorities. These certificate authorities follow a hierarchical structure. *Root authorities* act as trusted parties. There can be one or several root authorities, though several root authorities are more common in large-scale international PKIs.

Each root authority has a public key pair consisting of a public verification key and a private signature key. Hierarchical PKIs assume that there is a method for ensuring that everyone knows the verification keys of the root authorities. For instance, they can be shipped with the operating system. A root authority can now sign the public keys of other users, together with some information associated with the key that identifies the owner. The combination of the public key, the identifying information, and the signature is called a *certificate*. Users with certificates can act as certificate authorities as well. They can hence sign the public keys of other users, who can again sign the public keys of more users.

The result is a hierarchical structure of multiple levels. *First-level certificate authorities* are those whose certificates have been signed directly by a root authority. To verify the certificate of a first-level authority, users check the correctness of the root authority's signature on their certificate.

A *second-level certificate authority* is an authority whose certificate is signed by a first-level certificate authority. The second-level authority has to present both its own certificate and the certificate of the first-level certificate authority that signed the certificate, also called the parent authority. Users then verify that the second-level authority is who they claim to be by i) verifying the certificate of the parent authority, and ii) verifying the certificate of the second-level authority using the public verification key of the first-level authority.

In general, a k-th level certificate authority CA has a certificate signed by a $k-1$-th level authority. For verification, CA has to present its certificate and all the certificates its parent authority has to present. The user then verifies all of the certificates, starting with the one of the first-level authority and ending with CA's certificate. Each certificate is verified using the public key of the parent authority. If any verification step fails, the user does not accept the certificate. The set of all the certificates involved in verifying a certificate of an entity U is called U's *certificate chain*.

While certificates can theoretically have an arbitrary form, there is a standard format, namely $X.509$. It specifies the information a certificate should include such as version number, used signature algorithm, and expiry date.

Web-of-Trust In the absence of trusted parties that can act as roots, hierarchical PKIs are not feasible. An alternative is the *Web-of-Trust*. Here, people sign the public keys of participants that they know in the real-world. Users verify a new public key by checking if it has been signed by one or more public keys that i) they have signed or ii) have been signed by a public key they consider legitimate.

We have now discussed the theoretical background of many secure applications used in computer networks. In the remainder of this work, we exemplify some of these applications.

3 Network Security: Key Protocols

In this section, we cover various protocols that make use of the previously discussed cryptographic primitives. In doing so, we explore security issues of each layer and present solutions.

On the physical layer, wireless communication is particularly vulnerable to attacks. In contrast to cables, wireless are easy is listen in to and manipulate.

Moving on to the network layer, we consider the problem of organizations with internal networks in different physical locations that communicate by the Internet. Virtual Private Networks (VPNs) aim to enable treating all these separate networks as one private network in terms of security.

One of the most successful security protocols is Transport Layer Security (TLS), which enables confidentiality and integrity for communication between a client and a server. Furthermore, TLS authenticates the web server to the client. TLS underlies almost all Internet communication of today that follows the client-server model.

Most widely used applications have their own security protocols. Here, we focus on email encryption, secure DNS, and methods for authenticating users to web servers.

3.1 Secure Wireless Communication

Securing a wireless network typically implies the following goals:

- Confidentiality: Nobody but client and access point should be able to access the content of the communication.
- Integrity: Nobody should be able to modify communication between the access point and the client.
- Access Control: Only authorized users should be able to use the network.

The algorithms discussed for wireless networks in previous chapters do not provide any of these properties. We now discuss various protocols for wireless security.

Wired Equivalent Privacy (WEP) WEP was the first protocol aiming to provide secure communication over wireless channels. It lacked a proper external reviewing process before its roll-out. Unsurprisingly, it failed to achieve all the above security goals [2].

In a nutshell, WEP uses a *challenge-response protocol* for access control, a cryptographically weak stream cipher called $RC4$ for confidentiality, and a Cyclic Redundancy Check (CRC) for integrity.

The challenge-response protocol works as follows: All users of the wireless network receive the same password, which corresponds to a secret key K of 40 bits. If a user U wants to access the network, it sends a connection request to an access point AP. AP replies with a random bit string c. c is called the *challenge*. U first concatenates the 40-bit key K with a nonce v of 24 bits. They then use the concatenation as the input for $RC4$ to generate a keystream s of the same length as the challenge c. Last, U sends $c\ XOR\ s$ and the nonce v to AP. AP uses v and K to generate the keystream and corresponding XOR. If AP's result agrees with the received XOR, they grant U access to the network.

This challenge-response protocol can easily be misused by an attacker Malory to gain access to the network without knowing the password. Malory observes AP sending c and U sending $c\ XOR\ s$ and v. She can hence compute $c\ XOR(c\ XOR\ s) = s$. Afterwards, she contacts AP for a challenge d and returns $d\ XOR\ s$ and v.

After authentication, both U and AP can send messages to each other. Without loss of generality, assume that U sends a message M to AP as follows:

1. U computes $CRC(M)$ and $M||CRC(M)$ with $||$ denoting concatenation.
2. U chooses a nonce v and computes $RC4(K||v)$ to obtain a keystream s of the same length as $M||CRC(M)$.
3. U computes $C = (M||CRC(M))\ XOR\ s$.
4. U sends C and v to AP.

The above protocols fails to achieve both confidentiality and integrity. Its failure to achieve confidentiality is due to two reasons: First, the initial key length of 40 bits is too short, so that an attacker can easily bruteforce the key. Second, the pseudo-random number generator used in $RC4$ has multiple security weaknesses, including i) predictable patterns in the keystream and ii) correlations between keystream and key. Both i) and ii) enable the partial reconstruction of original messages and keys from the ciphertext.

The lack of integrity follows from the use of a CRC. Note that both the CRC is a linear operation in the sense that $CRC(M\ XOR\ N) = CRC(M)\ XOR\ CRC(N)$. Malory can now change the sent message M without being detected. Concretely, Malory can determine the respective length of M and $CRC(M)$ from $C = (M||CRC(M))\ XOR\ s$ as the algorithm for CRC computation is publicly known. Consequently, Malory divides C into

1. $C_1 = M\ XOR\ s_1$, which is the part of the ciphertext corresponding to M with s_1 being the first $length(M)$ bits of the keystream s, and
2. $C_2 = CRC(M)\ XOR\ s_2$, the part of the ciphertext corresponding to $CRC(M)$ with s_2 being the remaining $length(CRC(M))$ bits of s.

For a message N of the same length as M, Malory computes

$$\begin{aligned} C_1' &= C_1\ XOR\ N = M\ XOR\ s_1\ XOR\ N = (M\ XOR\ N)\ XOR\ s_1 \\ C_2' &= C_2\ XOR\ CRC(N) = CRC(M)\ XOR\ s_2\ XOR\ CRC(N) = (CRC(M)\ XOR\ CRC(N))\ XOR\ s_2 \\ &= (CRC(M\ XOR\ N)\ XOR\ s_2 \end{aligned}$$

Hence, C_1' is the encryption of $M\ XOR\ N$ and C_2' the encryption of $CRC(M\ XOR\ N)$, which is what AP expects. Hence, AP will accept the modified message and not detect that Malory changed it.

In addition to the above vulnerabilities, giving all users the same password inadvertently leads to problems: It becomes cumbersome to revoke access rights as everyone has to change their password. Furthermore, for large groups, it is likely that at least one person leaks the secret.

Wi-Fi Protected Access (WPA) WPA was a short-term replacement for WEP. It used a MAC instead of a CRC, it furthermore uses an encryption protocol with 128 bits.

WPA2 was the carefully researched successor of WPA and is still widely used. It relies on AES as an encryption protocol and uses a different interactively generated key for each message after the initial 4-way handshake. Furthermore, WPA2 comes with an Enterprise mode that allows to assign users individual passwords rather than one general password. Since 2006, WiFi devices are required to implement WPA2.

Despite the existence of security proofs, there was an attack on WPA2's handshake protocol in 2017 [9]. The attack was possible because the models used in the proofs failed to consider replay attacks during the handshake. While there is a patch available, the patch does not work for all devices. WPA3 should resolve this and other issues but attacks on WPA3 in Spring 2019 [10] have delayed its large-scale deployment.

3.2 Virtual Private Networks (VPNs)

In this section, we address the scenario of an organization that wants to maintain a private networks such that:

1. Communication between two private networks in physically different places that has to traverse an untrusted network should be encrypted and authenticated.

2. Communication with authorized parties outside of the organization's networks, e.g., employees working from home, should be encrypted and authenticated.

Optimally, communication between parties in different physical locations that requires using an untrusted network such as the Internet should have the same security guarantees as communication within one private network. However, that is not quite possible in this generality. Sending content via an untrusted network naturally reveals some information, even if the communication is encrypted and anonymized. Furthermore, organizations can do little to improve the availability of the untrusted network. As a consequence, VPNs focus on encrypted and authentication. A large number of VPNs relies on *IPSec* to achieve these goals but other protocols exist.

IPSec As the name indicates, IPSec, or IP Security, provides security for IP packets, both for IPv4 and IPv6. Concretely, IPSec uses symmetric-key encryption for confidentiality and message authentication codes for integrity. The cryptographic algorithms vary. The encryption algorithm is either a block cipher, AES or less commonly TripleDES, or a stream cipher, typically Chacha20. Commonly used message authentication codes is a HMAC with SHA1 or SHA2 as the hash function or a block cipher with a mode that combines integrity and confidentiality [4].

In the context of IPSec, (security) gateways play a key role. In a protected and private network, gateways process all traffic from and to the outside. Apart from their role for IPSec, which is described below, they typically also act as a firewall and identify undesired traffic.

IPSec uses different modes, depending on the role of the communication partners:

- *Transport mode* handles communication between two (non-gateway) hosts that are not in any private network or one gateway and a hosts outside of the private network.

- *Tunnel mode* addresses the communication between two gateways in different private networks.

IPSec makes use of the following three components [8]:

- *Security Associations (SA)* define the parameters used to achieve confidentiality and integrity. They also specify the key exchange protocol.

- *Authenticated Header (AH)* provides integrity of the IP header and payload.

- *Encapsulated Security Payload (ESP)* provides confidentiality and integrity for an IP packet.

We focus on ESP here. Assume that two parties A and B, be it gateways or normal hosts, are communicating. They have exchanged keys and now, A wants to send an IP packet to B. The original IP packet consists a header and a payload, which are neither confidential nor authenticated.

Transport mode does only protect the payload of the packet. A applies IPSec by inserting a ESP header with IPSec-related header fields between the IP header and the payload. A encrypts the payload and appends a message authentication code to the payload. A also modifies the header fields that are affected by the changes to the payload, e.g., the packet size. Note that any router on the path will consider the combination of ESP header, encrypted payload, and MAC as the payload of the modified packet. In this manner, the payload is encrypted and authenticated. The header is neither encrypted nor authenticated.

In contrast, tunnel mode protects the complete IP packet. A first creates a new IP header. They then send a new IP packet to B consisting of:

1. the new IP header,

2. ESP header,

3. the encrypted old IP header,

4. the encrypted payload,

5. a message authentication code that guarantees integrity of both the old IP header and the payload.

IPSec-based VPNs VPNs leverage IPSec to ensure confidentiality and integrity. Like IPSec, VPNs utilize transport and tunnel mode.

VPNs apply the transport mode when a user outside of the private network communicates with hosts inside the private network. Let U be the outside user, H the host inside the network U communicates with, and G the gateway. Assume that U and G have successfully completed a key exchange protocol. U now has an original IP packet with its IP address as the source and H's IP address as the destination. When using the VPN, U modifies the IP packet as described above for IPSec's transport mode. The private network is set up such that all traffic goes via the gateway G. G checks that the message authentiation code is correct and decrypts the payload. Then, G forwards the packet to H, for which it looks like a normal IP packet. Similarly, for any IP packet going from H to U, G performs the necessary modification for IPSec's transport mode. U executes the corresponding decryption and verification operations.

In tunnel mode, we consider two hosts H_1 and H_2 that communicate. Both are within the private network but in physically different locations such that their communication has to traverse the untrusted Internet. Each location has a different gateway, which we denote by G_1 and G_2 for H_1 and H_2, respectively. When H_1 wants to contact H_2, H_1 sends a normal IP packet with a header and a payload. The gateway G_1 inspects the packet and modifies with using IPSec's tunnel mode. The new IP header has the IP address of G_1 as the source and the IP address of G_2 as the destination. In this manner, the IP addresses of the two communicating hosts are hidden, providing a low degree of anonymity. When receiving the new IP packet, G_2 will check that the message authentication code is valid and then decrypt the original IP packet. If the check and the decryption succeed, G_2 sends the original IP packet to H_2.

In summary, VPNs provide confidentiality and integrity when used in transport mode. Tunnel mode in addition hides the IP addresses of the communicating hosts from parties in the untrusted network.

3.3 Transport Layer Security (TLS)

TLS provides confidentiality and integrity for communication between clients and web servers. Furthermore, it authenticates servers. Optionally, TLS can also authenticate clients but the option is barely used outside of server-to-server communication. TLS runs on top of a reliable transport layer protocol, usually TCP, and HTTPS, the combination of HTTP of TLS, is probably the most widely used application layer protocol. As a consequence, TLS is probably the most successful security solution. One of its key properties is that it is seamlessly integrated into browsers.

The development of TLS goes back to the 90s of the previous century. Initially, the protocol was named SSL, Secure Socket Layer. Over the years, researchers and developers have improved TLS considerably. In 2017, the roll-out of TLS 1.3 started, which included a number of new cryptographic algorithms and finally deprecated ciphers whose use has been discouraged for decades [7].

A hierarchical public key infrastructure with multiple root certificate authorities enables server authentication. Servers are required to buy a certificate to use TLS, which limits the use of TLS for applications such as P2P. Furthermore, unreliable certificate authorities can lead to incorrect certificates, i.e., a certificate that allows a web server to authenticate as an url that is not actual its own url.

An additional problem of TLS is the ability of servers to obtain valid certificates for urls that are similar to popular urls, e.g., `amazonn.com`. If users accidentally types `amazonn.com`, the authentication succeeds and the user proceeds to have a confidential and authenticated communication but with a different party than intended.

TLS establishes a connection between a user U and a server S by first executing a handshake protocol to establish the identity of the server and agree upon cryptographic algorithms and keys. In this handshake protocol, U first sends a connection requests, which includes its TLS version and the different *ciphersuites*, i.e., combinations of encryption and message authentication protocols, that U can use. The web server S replies with i) a chosen ciphersuite, namely the most secure ciphersuite that both U and S know and ii) its certificate chain, as defined in Section 2.6. After U has validated the certificate chain, U and S execute a key exchange for symmetric keys for both encryption and message authentication.

The method used for key exchange varies. Options include but are not limited to various version of Diffie-Hellman. The preferred ciphersuites offered by modern TLS versions are AES-based block ciphers integrating message authentication and Chacha20 with the Poly1305 message authentication code (see Section 5.5 of the TLS 1.3 RFC [7]).

3.4 Secure DNS (DNSSEC)

In its original form, DNS is highly vulnerable to *DNS spoofing* and *DNS poisoning* attacks.

In a *spoofing attack*, an active attacker Malory claims that an object is a different object. In the context of DNS, spoofing indicates that Malory claims an IP address corresponds to a domain name without that being the case. In a *poisoning attack*, Malory corrupts the records of an honest entity, e.g., by spoofing. In the context of DNS, Malory convinces a DNS server to include incorrect mappings from domain names to IP addresses.

In the absence of additional security protocols, DNS spoofing and poisoning is simple. If Malory aims to corrupt the record for the domain name dn at the DNS server S, she first asks ds. Unless S has the record for dn in its local cache, S will forward the query using UDP as the transport layer protocol. Malory can now send S a fake DNS record for dn

mapping it to an incorrect IP address, most likely an address under her control. As S stores the record in its local cache, users that requests dn in the future will receive incorrect information.

The impact of this information on the user's security depends on the circumstances. If the actual web server uses TLS, whose certificates are bound to the domain name, an incorrect web server cannot provide the necessary certificates. Unless users ignore the corresponding warnings, the effect of DNS spoofing is a lack of availability. Without the use of TLS, Malory's web server can successfully pretend to be a different entity and extract confidential information such as passwords from the user.

DNSSEC uses a hierarchical PKI to enable authentication of domains. The hierarchical PKI mirrors the hierarchical structure of DNS: Name servers for one domain sign the public keys of all servers in their domain. In order to retrieve DNS certificates, DNSSEC adds various new record types, the most important ones being:

- *RRSIG* is the *resource record signature*, i.e., a signature over a set of DNS records
- *DNSKEY* contains the public key that should be used to check signatures.

Note that DNSSEC only provides authentication and not confidentiality.

Despite the danger of DNS spoofing and poisoning attacks, DNSSEC is not widely deployed in comparison with TLS. One particular problem hampering deployment is backward compatibility.

3.5 Pretty Good Privacy

Pretty Good Privacy (PGP) is a program for email security that relies on hybrid encryption for confidentiality and digital signatures for integrity and authentication. In a nutshell, PGP first executes hybrid encryption, followed by hash-then-sign.

In PGP, a user U maintains two key pairs. For authentication and integrity, U has the public verification key K^U_{VERIFY} and the private signature key K^U_{SIGN}. For encryption, U has a public encryption K^U_{ENC} and a private decryption key K^U_{DEC}. We denote the asymmetric-key encryption function by Enc^A and the corresponding decryption function by Dec^A. For the digital signature, PGP uses functions $Sign$ and $Verify$ as introduced in Section 2.5. In addition, PGP makes use of a cryptographic hash function h and a symmetric-key encryption algorithm with encryption function Enc^S and decryption function Dec^S.

While U can use the same key pair for both encryption and digital signatures, it is not advised. In general, users should change encryption keys regularly but keep long-term signature keys. The public verification key is then linked to the identity. If the secret signing key leaks, the user U can revoke the key. All signatures created after revocation are invalid. Thus, the damage of the leakage is low assuming that the user detected it immediately. In contrast, if U accidentally leaks a long-term decryption key, an attacker who recorded previous message to U can decrypt all of these messages. Thus, the longer a person maintains a key pair for encryption, the larger the damage.

Assume that both A and B have created their respective keys and know each other's public keys. When sending a message to B, A

1. computes $h(M)$,
2. computes $s = Sign(h(M), K^A_{SIGN})$ using A's private signature key,
3. chooses a key K for the symmetric-key encryption algorithm,
4. computes the ciphertext $C_1 = Enc^A(K^B_{ENC}, K)$, the encryption of K under B's public encryption key, and
5. computes the ciphertext $C_2 = Enc^S(K, M||s)$, the encryption of the concatenation of M and the signature s under the key K.

A sends both C_1 and C_2. After receiving the ciphertext, B

1. decrypts the secret key $K = Dec^A(K^B_{DEC}, C_1)$ using B's private decryption key,
2. decrypts the concatenation $M||s = Dec^S(K, C_2)$,
3. computes $h(M)$,
4. computes $Verify(K^A_{VERIFY}, h(M), s)$ using A's public verification key.

If the verification succeeds, B accepts the message.

In contrast to TLS, PGP relies on the Web-of-Trust as a decentralized PKI.

3.6 User Authentication

As discussed in Section 3.3, TLS authenticates web servers to clients but not clients to web servers. For applications such as online banking, correct client authentication is essential.

Strong client authentication should rely on *multi-factor authentication*, i.e., multiple checks based on different criteria should be applied. Factors can be classified as:

1. *Something you know*: passwords, pins, passphrases, etc
2. *Something you have*: phone number, credit card, physical keys, etc
3. *Something you are (i.e., biometrics)*: finger prints, iris scan, etc

Typically, authentication should at least use two of the above. An additional weaker factor is *somewhere you are*, which uses e.g., the country associated with an IP address. However, this factor is mainly used to detect irregularities such as unusual places the users logs from in.

References

[1] Computer security and privacy, university of waterloo.

[2] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications: the insecurity of 802.11. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, 2001.

[3] Johannes Buchmann. *Introduction to cryptography*. Springer Science & Business Media, 2013.

[4] Paul Hoffman. Cryptographic suites for ipsec. Technical report, 2005.

[5] Bert-Jaap Koops. Cybercrime legislation in the netherlands. In *Netherlands reports to the eighteenth international congress of comparative law*, 2010.

[6] Charles P Pfleeger and Shari Lawrence Pfleeger. *Security in computing*. Prentice Hall Professional Technical Reference, 2002.

[7] Eric Rescorla. The transport layer security (tls) protocol version 1.3. Technical report, 2018.

[8] Rodney Thayer, Naganand Doraswamy, and Rob Glenn. Ip security document roadmap. Technical report, 1998.

[9] Mathy Vanhoef and Frank Piessens. Key reinstallation attacks: Forcing nonce reuse in wpa2. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[10] Mathy Vanhoef and Eyal Ronen. Dragonblood: A security analysis of wpa3's sae handshake. *IACR Cryptology ePrint Archive*, 2019, 2019.