Chapter 12

Security

Every day people order and pay online – but is it secure?

12.1 Transport Layer Security

Protocol 12.1 (Transport Layer Security, TLS)**.** *TLS is a network protocol in which a client and a server exchange information in order to communicate in a secure way. Common features include a key exchange protocol (Section 12.2), the authentication of the server to the client (12.3), a bulk encryption algorithm (12.4), and a message authentication algorithm (12.5).*

Remarks:

- TLS is the successor of Secure Sockets Layer (SSL). However, sometimes in practice the term SSL includes (the newer) TLS as well.
- HTTPS (Hypertext Transfer Protocol Secure) is not a protocol on its own, but rather denotes the usage of HTTP via TLS or SSL.
- SSH (Secure Shell), even though close in name to SSL, is something different: It is a protocol to allow a client to remotely access a server, e.g., for a command-line interface.

12.2 Key Exchange

How to agree on a common secret key in public, if you never met before?

Definition 12.2 (Primitive Root)**.** *Let $p \in \mathbb{N}$ be a prime. $g \in \mathbb{N}$ is a primitive root of p if the following holds: For every $h \in \mathbb{N}$, with $1 \leq h < p$, there is a $k \in \mathbb{N}$ s.t. $g^k = h \mod p$.*

Algorithm 12.3 Diffie-Hellman Key Exchange

Input: Publicly known prime p and a primitive root g of p.
Result: Alice and Bob agree on a common secret key.

1: Alice picks k_A, with $1 \leq k_A \leq p-2$ and sends $g^{k_A} \mod p$ to Bob
2: Bob picks k_B, with $1 \leq k_B \leq p-2$ and sends $g^{k_B} \mod p$ to Alice
3: Alice calculates $\left(g^{k_B}\right)^{k_A} \mod p = g^{k_B k_A} \mod p$
4: Bob calculates $\left(g^{k_A}\right)^{k_B} \mod p = g^{k_A k_B} \mod p$
5: Alice & Bob have a common secret key $g^{k_A k_B} \mod p = g^{k_B k_A} \mod p$

Remarks:

- In crypto, it's always Alice and Bob, with a possible attacker Eve.
- Also, we will use k for keys, m for messages, p for primes, g for primitive roots, and c for ciphertext (encrypted messages). Generally speaking, an encryption algorithm encrypts a plain message m by applying a key k, resulting in ciphertext c.
- Small (not so secure) example for prime $p = 5$ and primitive root $g = 2$: $2^1 = 2 \mod 5$, $2^2 = 4 \mod 5$, $2^3 = 3 \mod 5$, $2^4 = 1 \mod 5$. One more primitive root for $p = 5$ exists. There are sophisticated methods to quickly find primitive roots, but they are beyond the material covered in this lecture.
- Algorithm 12.3 with $p = 5$ and $g = 2$: Alice picks $k_A = 2$ with $2^2 = 4 \mod 5$, and Bob picks $k_B = 3$ with $2^3 = 3 \mod 5$. Thus, Bob receives 4 and Alice receives 3. Then, Bob calculates $4^3 = 4 \mod 5$, and Alice calculates $3^2 = 4 \mod 5$. Hence, Alice and Bob have agreed on the common secret key of 4.
- How secure is Algorithm 12.3?

Definition 12.4 (Discrete Logarithm Problem)**.** *Let $p \in \mathbb{N}$ be a prime, and let $g, a \in \mathbb{N}$ with $1 \leq g, a < p$. The discrete logarithm problem is defined as finding an $x \in \mathbb{N}$ with $g^x = a \mod p$.*

Remarks:

- Intuitively, the best approach to calculate the common secret key of Algorithm 12.3 from the publicly known p, g, g^{k_A}, g^{k_B} is to solve the discrete logarithm problem. This is also the best known attack.
- However, for some classes of primes there are better attacks, which is why one often resorts to so-called safe primes p, where $p' = (p-1)/2$ is also a prime.
- How to find big enough primes though? Deterministic methods are still too slow in practice. Thus, let's go probabilistic with the following primality test.

Algorithm 12.5 Probabilistic Primality Testing

Input: An odd number $p \in \mathbb{N}$.
Result: Is p a prime?

1: Let $j, r \in \mathbb{N}$ and j odd with $p - 1 = 2^r j$
2: Select $x \in \mathbb{N}$ uniformly at random, $1 \leq x < p$
3: Set $x_0 = x^j \mod p$
4: **if** $x_0 = 1$ or $x_0 = p - 1$ **then**
5: Output "p `is probably prime`" and **stop**
6: **end if**
7: **for** $i = 1, \ldots r - 1$ **do**
8: Set $x_i = x_{i-1}^2 \mod p$
9: **if** $x_i = p - 1$ **then**
10: Output "p `is probably prime`" and **stop**
11: **end if**
12: **end for**
13: Output "p `is not prime`"

Lemma 12.6. *Algorithm 12.5 is correct with probability* 75% *if it outputs "p is probably prime", and* 100% *correct if it outputs "p is not prime".*

Corollary 12.7. *The runtime of Algorithm 12.5 is* $O(r) \in O(\log p)$

Remarks:

- The proof for the probabilistic correctness of the primality test in Algorithm 12.5 goes beyond the material covered in this lecture.
- Algorithm 12.5 is a Monte Carlo algorithm as its (fast) runtime is deterministic, but the output can be wrong with bounded probability. However, running the algorithm again on the same p, but with different x, produces an independent result, allowing to bound the error probability by $\frac{1}{4^r}$ in r runs.
- A simple method to find big primes is thus as follows: Pick a big random number p, with p being odd. Run Algorithm 12.5 until p is prime with the desired probability of $1 - \varepsilon$. If p is not prime, pick another p. According to the prime number theorem, the average distance between two primes of size at most n is just $\ln n$, i.e., there is a good chance to find a big prime.

Definition 12.8 (Man in the Middle Attack)**.** *A man in the middle attack is defined as an attacker Eve deciphering or changing the messages between Alice and Bob, while Alice and Bob believe they are communicating directly with each other.*

Theorem 12.9. *The Diffie-Hellman Key Exchange from Algorithm 12.3 is vulnerable to a man in the middle attack.*

Proof. Assume that Eve can intercept and relay all messages between Alice and Bob. That alone does not make it a man in the middle attack, Eve needs to be able to decipher or change messages without Alice or Bob noticing. However,

Eve can emulate Alice's and Bob's behavior to each other, by picking her own k'_A, k'_B, and then agreeing on common keys $g^{k_A k'_B}$, $g^{k_B k'_A}$ with Alice and Bob, respectively. Thus, Eve can relay all messages between Alice and Bob while deciphering and (possibly) changing them, while Alice and Bob believe they are securely communicating with each other. □

Remarks:

- It is a bit like concurrently playing chess with two grandmasters: If you play white and black respectively, you can essentially let them play against each other by relaying their moves.
- How do we fix this? One idea is to personally meet in private first, exchange a common secret key $k_{A,B}$, and then use this key for secure communication. Now a man in the middle cannot change the key.

Definition 12.10 (Forward Secrecy)**.** *A sequence of secured communication rounds has the property of forward secrecy, if discovering the secret key(s) of a single communication round does not reveal the content of past communication rounds.*

Remarks:

- So Alice and Bob cannot use the same secret key multiple times.

Algorithm 12.11 Diffie-Hellman Key Exchange with Forward Secrecy

Input: Alice's and Bob's common secret key $k_{A,B}$.
Result: A Diffie-Hellman key exchange not vulnerable to a man in the middle attack, and with forward secrecy.

1: Alice and Bob run Algorithm 12.3 to obtain round key $g^{k_A k_B}$
2: Alice sends a random number $1 \leq x_A \leq p - 2$ encrypted with $k_{A,B}$ as c_A to Bob, and Bob sends Alice a random number $1 \leq x_B \leq p - 2$ encrypted with $k_{A,B}$ as c_B a challenge, respectively
3: Alice and Bob decrypt the respective messages, and Alice sends $x_B + 1$ encrypted with $k_{A,B}$ to Bob as a response (and Bob as well with $x_A + 1$)
4: If decryption yields $x_A + 1$ for Alice, or $x_B + 1$ for Bob, respectively, they accept the round key $g^{k_A k_B}$

Lemma 12.12. *Algorithm 12.11 has the property of forward secrecy and is not vulnerable to a man in the middle attack, if encryption with* $k_{A,B}$ *is secure.*

Proof. For a man in the middle attack, Eve needs to be able to decrypt and encrypt with $k_{A,B}$ to convince Alice and Bob that they directly communicated with each other, which is a contradiction to the security assumption.

Regarding forward secrecy, if the attacker Eve gathers the secret key $g^{k_A k_B}$ of a communication round, she can decrypt the messages of this communication round. Even if Eve gains access to $k_{A,B}$, she cannot gain access to the keys generated in past communication rounds. □

Remarks:

- Observe that forward secrecy only applies to communication rounds in the past. If Eve gains access to $k_{A,B}$, she can perform man in the middle attacks in future communication rounds.
- However, we have a new inconvenience: Alice and Bob need to agree on a secret key $k_{A,B}$ beforehand. Furthermore, with n participants, everyone needs $n-1$ different keys.

12.3 Public Key Cryptography

"*Love all, trust a few.*" – William Shakespeare

Definition 12.13 (Public Key Cryptography)**.** *A public key cryptography system uses two keys: A public key k_p, to be disseminated to everyone, and a secret (private) key k_s, only known to the owner. A message encrypted with the secret key can be decrypted with the corresponding public key. Analogously, a message encrypted with the public key can be decrypted with the corresponding secret key.*

Remarks:

- Popular public key cryptosystem include RSA and elliptic curve cryptography.
- With public key cryptography, we have reduced the number of keys – everyone just needs a secret and a public key.
- A conceptual way to think of public key cryptography is as follows: The secret key is a physical (secret) key that opens a specific type of padlock, and this type of padlock is freely available. The public key is a physical key too, freely available, but it opens only a (secret) specific type of padlock. If Alice wants to send Bob an encrypted message, she applies his public padlock to the message container, and only Bob can open it. Similarly, if Alice wants to authenticate her message to Bob, she locks the container with her secret padlock, and only Alice's public key can unlock it. Lastly, if Alice wants to ensure both encryption and authentication, she applies both her own secret padlock and Bob's public padlock to the message container.
- We will now extend the Diffie-Hellman algorithm to public key cryptography.

Algorithm 12.14 Elgamal Public Secret Key Generation

Input: Publicly known prime p and a primitive root g of p.
Result: Alice generates a public and a secret key

1: Alice picks random k_s with $1 \leq k_s \leq p-2$ as her secret key
2: Alice calculates $k_p = g^{k_s} \mod p$ as her public key

Remarks:

- Alice can publish p, g, k_p, but should keep k_s to her own.
- We will now start with encryption, before covering authentication.

Algorithm 12.15 Elgamal Public Key Encryption and Decryption

Input: Alice and Bob know p, g, k_p, Alice knows k_s.
Result: Bob sends Alice an encrypted message, which she can decrypt.

1: Bob picks a message $1 \leq m \leq p-2$ and a random $1 \leq x \leq p-2$
2: Bob sends $g^x \mod p$ and $c = m \cdot k_p^x \mod p$ to Alice
3: Alice first calculates $y = (g^x)^{p-k_s-1} \mod p$
4: Alice then obtains $m = y \cdot c \mod p$

Theorem 12.16 (Fermat's little theorem)**.** *Let p be a prime number. Then, for any $a \in \mathbb{N}$ holds: $a^p = a \mod p$. If a is not divisible by p, then $a^{p-1} = 1 \mod p$.*

Lemma 12.17. *Algorithm 12.15 is correct.*

Proof.

$$\begin{aligned} y \cdot c &= (g^x)^{p-k_s-1} \left(m \cdot k_p^x\right) \mod p \\ &= (g^x)^{p-k_s-1} \left(m \cdot \left(g^{k_s}\right)^x\right) \mod p \quad \text{(using } k_p = g^{k_s} \mod p\text{)} \\ &= (g^x)^{p-k_s-1} \, m \cdot \left(g^x\right)^{k_s} \mod p \\ &= m \, (g^x)^{p-1} \mod p \\ &= m \mod p \quad \text{(using Theorem 12.16).} \end{aligned}$$

□

Remarks:

- We can now send someone an encrypted message using public key cryptography, but what about authentication?
- Again, we first need some number theoretic preliminaries.

Definition 12.18 (Greatest Common Divisor, gcd)**.** *The greatest common divisor (gcd) of two integers i_1, i_2 is the largest integer that divides i_1 and i_2 without a remainder.*

Theorem 12.19. *Let p be a prime and i be an integer with $gcd(i, p) = 1$. Let $a_1, a_2 \in \mathbb{N}$. If $a_1 = a_2 \mod (p-1)$, then $i^{a_1} = i^{a_2} \mod p$.*

Algorithm 12.20 Elgamal Authentication

Input: Alice and Bob know p, g, k_p, Alice knows k_s.
Result: Alice signs a message $1 \leq m \leq p-2$, which Bob authenticates.

1: Alice picks a random $1 \leq x \leq p-1$, with $gcd(x, p-1) = 1$
2: Alice calculates $a = g^x \mod p$ and $b = x^{-1} \mod (p-1)$
3: Alice calculates $d = (m - ak_s)b \mod (p-1)$
4: Alice sends the message m and the signature (a, d) to Bob
5: Bob checks if $1 \leq a \leq p-1$, else he rejects
6: Bob accepts Alice's signature for m if $k_p^a a^d = g^m \mod p$

Remarks:

- A multiplicative inverse modulo p (in this algorithm: $x^{-1} \mod p$), can be calculated using, e.g., the extended Euclidean algorithm.

Lemma 12.21. *Algorithm 12.20 is correct.*

Proof. With $d = (m - ak_s)x^{-1} \mod (p-1)$, it follows that:

$$dx = m - ak_s \mod (p-1) \Rightarrow m = dx + ak_s \mod (p-1).$$

Using Theorem 12.19, we now obtain $g^{dx+ak_s} = g^m \mod p$. Hence,

$$k_p^a a^d \mod p = \left(g^{k_s}\right)^a (g^x)^d = g^{ak_s} g^{dx} \mod p = g^m \mod p.$$

□

Remarks:

- The security of the Elgamal public key cryptography again depends on the hardness of the discrete logarithm problem.
- We can now authenticate a message using public key cryptography, e.g., we can check that the public key of Alice corresponds to Alice's secret key.
- However, we are back still at our old problem: How do I know that Alice's public key really belongs to Alice? Maybe Eve pretended to be Alice? To use a famous saying by Peter Steiner: "*On the Internet, nobody knows you're a dog*".
- What can we do, unless we personally meet with everyone to exchange secret keys? The answer lies in trusting a few, in order to trust many: Let's say that you don't know Alice, but both Alice and you know Doris. If you trust Doris, then Doris can verify Alice's public key for you. In the future, you can ask Alice to vouch for her friends as well, etc.
- Trust is not limited to real persons though, especially since Alice and Doris are represented by their keys. Take a website like PayPal for example. How do you know that you give them your credit card information, and not some infamous Nigerian princess Eve? You probably don't know anybody who personally knows PayPal...

Definition 12.22 (Web of Trust)**.** *Let $G = (V, E)$ be a graph, where an edge between two nodes u, v represents trust between u, v. For any two nodes u, w, we say u trusts w if there is a path from u to w in G.*

Remarks:

- Hence, if you want someone to authenticate themselves, you need to find a path in the Web of Trust to them.
- In practice, the Web of Trust is a bit more sophisticated, as you can assign various levels of trust – and you might only trust someone in short distance.
- The whole situation is a bit of a chicken and egg dilemma though. In the beginning, you don't trust anyone, and nobody trusts you. You may want to find some well-connected nodes and gain their trust. This is the motivation for certificate authorities.

Definition 12.23 (Certificate Authority, CA)**.** *A certificate authority is a node in a web of trust that is trusted by many other nodes.*

Remarks:

- A main distinction between a CA (or nodes in general) and your real-life friends is that trust is not needed to be mutual, edges in the web of trust can also be directed. As such a node u might trust v, but v does not necessarily need to trust u.
- You will find trust for some certificate authorities pre-installed on your system/browser, known as root certificates. When you want to know if you can trust a node, the node can supply you with a path (chain of trust) from the CA. More specifically, you will be supplied with signatures which you can check (as you trust the CA).
- Again, one can implement various levels of trust, e.g., you might only trust short paths.
- Moreover, a CA might get compromised. This leads to the idea of key revocation, where one can check if a key for a signature has been compromised – the corresponding certificate can be generated by anyone holding the respective secret key. Another idea is to also generate expiration dates for keys.
- A totally different problem is that your own set of root certificates might be compromised, e.g., if malicious software adds new root certificates to one's device.

12.4 Secret Sharing & Bulk Encryption

"*Three may keep a secret, if two of them are dead.*" – Benjamin Franklin

Definition 12.24 (Perfect Secrecy)**.** *An encryption algorithm has perfect secrecy, if the encrypted message reveals no information to an attacker, except for the possible maximum length of the message.*

Definition 12.25 (Threshold Secret Sharing)**.** *Let $t, n \in \mathbb{N}$ with $1 \leq t \leq n$. An algorithm that distributes a secret among n participants such that t participants need to collaborate to recover the secret is called a (t,n)-threshold secret sharing scheme.*

Algorithm 12.26 (n, n)-Threshold Secret Sharing

Input: A secret k, encoded in binary representation of length $l(k)$.

Secret distribution

1: Generate $n-1$ random binary numbers k_i of length $l(k)$ and distribute them among $n-1$ participants
2: Give participant n the value k_n as the result of *XOR* of k and $k_1, \ldots, k_{n-1}$, i.e., $k_n = k_1 \oplus k_2 \oplus \cdots \oplus k_{n-1}$

Secret recovery

1: Collect all n values $k_1, \ldots, k_n$ and obtain $k = k_1 \oplus k_2 \oplus \cdots \oplus k_{n-1} \oplus k_n$

Theorem 12.27. *Algorithm 12.26 has perfect secrecy even if $n-1$ participants collaborate.*

Proof. The theorem holds as applying the *XOR* operation $\oplus$ to a random bitstring and k results in a random bitstring. □

Remarks:

- How can we achieve a (t, n)-threshold secret sharing scheme with perfect secrecy?

Algorithm 12.28 (t, n)-Threshold Secret Sharing

Input: A secret k, represented as a real number.

Secret distribution

1: Generate $t-1$ random $a_1, \ldots, a_{t-1} \in \mathbb{R}$
2: Obtain a polynomial f of degree $t-1$ with $f(x) = k + a_1 x + \cdots + a_{t-1} x^{t-1}$
3: Generate n distinct $x_1, \ldots, x_n \in \mathbb{R} \setminus 0$
4: Distribute $(x_1, f(x_1))$ to participant P_1, ..., $(x_n, f(x_n))$ to P_n

Secret recovery

1: Collect t pairs $(x_i, f(x_i))$ from at least t participants
2: Use Lagrange's interpolation formula to obtain $f(0) = k$

Remarks:

- With at most $t-1$ pairs $(x_i, f(x_i))$, there are infinitely many possible polynomials with different values for $f(0)$.
- There are many other (t, n)-threshold secret sharing schemes, e.g., with intersecting hyperplanes.

- Note that in practice, a finite field of prime order instead of real numbers is used.
- We can now use the ideas in this section so far to develop a bulk encryption algorithm with perfect secrecy.

Definition 12.29 (Bulk Encryption Algorithm)**.** *A bulk encryption algorithm can securely encrypt a message of any size.*

Algorithm 12.30 One-Time Pad

Input: A message m known to Alice, and a symmetric key k (as a random bitstring) of length $l(k)$ known by both Alice and Bob.

Encryption

1: Alice sends $c = m \oplus k$ to Bob

Decryption

1: Bob obtains m by $m = c \oplus k$

Corollary 12.31. *Algorithm 12.30 has perfect secrecy.*

Remarks:

- Note that Algorithm 12.30 has one big disadvantage – Alice and Bob need to agree on a large random number first! While this is feasible for, e.g., secret agents, it is quite impractical for everyday usage.
- One can use padding to also remove information about the length of the message, e.g., by adding random bits to the secret.

Definition 12.32 (Electronic Code Book, ECB)**.** *Given a method to encrypt a block of x bits, ECB encrypts a message of length rx by splitting the message into r blocks of length x, encrypting each block separately.*

Remarks:

- Do we now have a secure method to easily encrypt a large message, if we can encrypt small blocks, each using the same one-time pad?
- Suppose you have two message blocks m_1, m_2 of the same length, encrypted with k, resulting in c_1, c_2. However, you can obtain $m_1 \oplus m_2 = c_1 \oplus c_2$, giving you information about m_1 and m_2.

Definition 12.33 (Cipher Block Chaining, CBC)**.** *Given a method f to encrypt a block of x bits, CBC encrypts a message of length rx by splitting the message into r blocks of length x, $m_1, m_2, \ldots, m_r$, encrypting (the plaintext of) each block XORed with the previous encrypted block, i.e., $c_i = f(m_i \oplus c_{i-1})$. The first block c_0 is initialized randomly.*

Remarks:

- Are we secure now? Using the same technique as in the last remark, you can again get, e.g., $m_4 \oplus m_5$.
- CBC is still one of the standard techniques though when encrypting blocks successively, as more advanced algorithms are not susceptible to this simple attack for one-time pads. An example would be the advanced encryption standard (AES). Using AES with CBC is an example of a bulk encryption algorithm. The operation of AES is beyond the scope of this short chapter however.

12.5 Message Authentication & Passwords

"*I've been imitated so well I've heard people copy my mistakes.*" – Jimi Hendrix

Definition 12.34 (Replay Attack)**.** *In a replay attack a previously valid message from Alice to Bob is sent again from an eavesdropper Eve to Bob.*

Remarks:

- An easy way to prevent replay attacks is to include time stamps in messages. Bob can detect a replay attack, if the time stamp is too old or multiple messages with the same time stamp arrive. Another idea is to use *nonces* (numbers only used once), with the sender and receiver keeping track of the nonces used so far.
- Another issue is that an attacker could change an encrypted message without knowing the content

Definition 12.35 (Malleability)**.** *If ciphertext c can be changed to c' such that the receiver decrypts it into a different message m' without noticing, the encryption algorithm is malleable.*

Remarks:

- The Elgamal encryption Algorithm 12.15 is malleable: An attacker can relay $c = m \cdot k_p^x \mod p$ as $z \cdot c$, resulting in a valid decryption of zm.
- Thus, we need a way to ensure that the messages cannot be changed by an attacker. A natural solution are hash functions. However the hash functions described in Chapter 6 are not secure.

Definition 12.36 (One-Way Hash Function)**.** *A hash function $h : U \to S$ is called one-way, if for a given $z \in S$ it is computationally hard to find an element $x \in U$ with $h(x) = z$.*

Definition 12.37 (Collision Resistant Hash Function)**.** *A hash function $h : U \to S$ is called collision resistant, if it is computationally hard to find elements $x \neq y$, $x, y \in U$, with $h(x) = h(y) \in S$.*

Remarks:

- It can be shown that a collision resistant hash function is also a one-way hash function.

Theorem 12.38 (Example for a Collision Resistant Hash Function)**.** *Let $p = 2q + 1$ be a safe prime, with primitive roots $g_1 \neq g_2$ of p. The hash function $h : \{0, \ldots, q-1\} \times \{0, \ldots, q-1\} \to \mathbb{Z} \setminus \{0\}$ with $h(x_1, x_2) = g_1^{x_1} g_2^{x_2} \mod p$ is a collision resistant hash function.*

Remarks:

- Popular hash functions used in cryptography include the Secure Hash Algorithm (SHA) and the Message-Digest Algorithm (MD).
- For a small example, let us pick $p = 5$ with primitive roots $g_1 = 2$ and $g_2 = 3$. We choose $x_1 = 3$ and $x_2 = 4$, obtaining the hash $h(3, 4) = 2^3 3^4 \mod 5 = 3 \mod 5$.
- It can be shown that finding a collision for the hash function described in Theorem 12.38 is equivalent to solving the discrete logarithm problem for $\log_{g_1} g_2$. Thus, the hash function is a collision resistant hash function, as we assume the discrete logarithm problem to be computationally hard.
- One might think that using a collision resistant hash function is good enough to store passwords for a service. E.g., store the hash of each password, and then compare it to the input of the user. Even if the hashes are leaked, an attacker Eve cannot recover the passwords – or can she?
- In practice, many users use short passwords, trading security for convenience. Eve can sample the hashes of common passwords such as "*password*", revealing the passwords of all users using these simple passwords. To counter this attack, one uses a technique called *salting*: The service adds a random bitstring (the *salt*) to each password before storing the hash (or, less secure, but simpler, the username). Even if the salt is known for each user, Eve needs to attack the hash of each user individually.
- To make life for Eve even harder, it is good practice to use hash functions that provably need a lot of computation and memory to execute. However, there is still a trade off as the real user wants to log in fast as well.
- Many web services already offer secure two-factor authentication (e.g., via mobile phones) instead of just passwords or challenge-response systems. However, there is a trade-off between security and convenience.
- Are we resistant against malleability now, if we include a hash of the encrypted message? No: An attacker changing the message can change the hash as well, as the hash function is not assumed to be secret. How do we prevent the hash from being modified without being noticed? The answer are HMACs:

Definition 12.39 (Message Authentication Code, MAC)**.** *A message authentication code is a bitstring that verifies that a message comes from the desired sender and was not changed until reaching the receiver.*

Definition 12.40 (Hash-Based Message Authentication Code, HMAC)**.** *A hash-based message authentication code is a MAC that uses a collision resistant hash function in combination with a secret key.*

Algorithm 12.41 Hash-Based Message Authentication Code Generation

Input: An encrypted message c, to be sent from Alice to Bob, the publicly known hash function h from Theorem 12.38, and a secret key $1 \leq k \leq c$ known to Alice and Bob.
Result: An HMAC for c, checkable by Bob.

1: Alice computes $h_A = h(k, h(k, c))$, and sends c, h_A to Bob
2: Bob computes $h_B = h(k, h(k, c))$, and checks if $h_A = h_B$

Remarks:

- In practice, if $k > c$, then k will be hashed to have a smaller size. Also, the key will be padded for extra security.
- If an attacker wants to change the message, he needs to change the HMAC too. To change the HMAC, he needs to know the secret key k
- Algorithm 12.41 can be also used with any other collision resistant hash function.

Chapter Notes

The Diffie-Hellman Key Exchange was published in the seminal paper [6], parallel unpublished work also existed from Ellis et al. at the British intelligence service GCHQ. For some works showing the hardness of breaking the Diffie-Hellman key exchange, we refer to, e.g., [5], [10], [17]. For some more recommendations on how to choose the parameters of the Diffie-Hellman key exchange see RFC 3526 at `http://tools.ietf.org/html/rfc3526`. The currently fastest algorithms to solve the discrete logarithm problem still have non-practical runtime, e.g., [1]. The idea of challenging the other party to return an encrypted version of one's random number incremented by one in Algorithm 12.11 is taken from the Kerberos protocol. The Elgamal cryptosystem was published by Elgamal in 1984 [8], some years after RSA [14].

The first deterministic polynomial primality test, by Agrawal, Kayal, and Saxena, was published in [9], with an improved runtime of $\tilde{O}(\log^6 p)$ available at `https://math.dartmouth.edu/~carlp/aks041411.pdf`. The Miller-Rabin primality test is from Rabin [13] and Miller [11]. For an introduction to number theory, we recommend, e.g., [15].

The idea for the web of trust was proposed by Zimmermann in 1992. For certificate chains and key revocation, we refer to RFC 5280 at `http://tools.ietf.org/html/rfc5280`.

The Chaum-van-Heijst-Pfitzmann hash function described in Theorem 12.38 was published in [4] by Chaum et al., for the reduction to the discrete logarithm problem see, e.g., [18]. However, the runtime of the Chaum-van-Heijst-Pfitzmann hash function is too high in practice, it is chosen in this chapter as it is easier to understand compared to other related work. The subsequently described HMAC Algorithm 12.41 is from RFC 2104 at `https://tools.ietf.org/html/rfc2104`, with further security updates in RFC 6151, cf. `https://tools.ietf.org/html/rfc6151`.

The secret sharing variant discussed in this chapter is from Shamir [16], Blakley developed similar work in parallel in 1979 [3], and also discussed its relation to one-time pads [2].

While CBC seems superior to ECB, there is one downside: Decryption of ECB can be parallelized, but the decryption of CBC has to be sequential. The in this context mentioned AES encryption is a symmetric key algorithm, based on the Rijndael cipher of Daemen and Rijmen. Details of the Advanced Encryption Standard can be found in `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`. AES, with a key length of 128,192, or 256 bits, replaced DES (Data Encryption Standard), as its key length of just 56 was no longer secure enough against brute-force attacks.

For a general overview of the topic of computer security, we recommend [12] and [7]. Lastly, as a very general recommendation, we urge you not to implement your own cryptosystem unless you really know what you are doing – there is just too much that can easily be missed.

This chapter was written in collaboration with Klaus-Tycho Förster.

Bibliography

[1] Leonard Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, SFCS '79, pages 55–60, Washington, DC, USA, 1979. IEEE Computer Society.

[2] G. R. Blakley. One time pads are key safeguarding schemes, not cryptosystems fast key safeguarding schemes (threshold schemes) exist. In *Proceedings of the 1980 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 14-16, 1980*, pages 108–113. IEEE Computer Society, 1980.

[3] G.R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the 1979 AFIPS National Computer Conference*, pages 313–317, Monval, NJ, USA, 1979. AFIPS Press.

[4] David Chaum, Eugène van Heijst, and Birgit Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 470–484. Springer, 1991.

[5] Bert den Boer. Diffie-hillman is as strong as discrete log for certain primes. In Shafi Goldwasser, editor, *Advances in Cryptology - CRYPTO '88, 8th*

Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings, volume 403 of *Lecture Notes in Computer Science*, pages 530–539. Springer, 1988.

[6] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.

[7] Niels Ferguson and Bruce Schneier. *Practical cryptography*. Wiley, 2003.

[8] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

[9] Nitin Saxena Manindra Agrawal, Neeraj Kayal. PRIMES Is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[10] Ueli M. Maurer. Towards the equivalence of breaking the diffie-hellman protocol and computing discrete algorithms. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 271–281. Springer, 1994.

[11] Gary L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. Syst. Sci.*, 13(3):300–317, December 1976.

[12] Josef Pieprzyk, Thomas Hardjono, and Jennifer Seberry. *Fundamentals of computer security*. Springer, 2003.

[13] M.O. Rabin. Probabilistic algorithms for testing primality. *J. Number Theory*, 12:128 – 138, 1980.

[14] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[15] Winfried Scharlau and Hans Opolka. *From Fermat to Minkowski: lectures on the theory of numbers and its historical development*. Undergraduate Texts in Mathematics. Springer, New York, 1985.

[16] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[17] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997.

[18] Douglas R. Stinson. *Cryptography - theory and practice*. Discrete mathematics and its applications series. CRC Press, 1995.