Summer of Science

CRYPTOGRAPHY

Endterm Report

Mentored by: Ankit Kumar Misra

Malay Kedia

June, 2024

Contents

1 Introduction

Modern cryptography involves the study of mathematical techniques for securing digital information, systems, and distributed computations against adversarial attacks.
Though historically, the major consumers of cryptography were military organizations and governments, but nowadays, cryptography is everywhere! We use it everywhere, from typing a password, purchasing something by credit card over the Internet, or downloading a verified update for your operating system.

1.1 Definitions

Encryption Scheme enable two parties to send messages while keeping those messages hidden from an eavesdropper who can monitor all communication between them. Security of all classical encryption schemes relies on a secret—a key— shared by the communicating parties in advance and unknown to the eavesdropper. This scenario, in which the communicating parties share some secret information in advance, is known as the private-key (or shared-/secret-key) setting of encryption.
In the context of private-key encryption, two parties share a key and use that key when they want to communicate secretly. One party can send a message, or plaintext, to the other by using the shared key to encrypt the message and thus obtain a ciphertext that is transmitted to the receiver. The receiver uses the same key to decrypt the ciphertext and recover the original message. The same key is used to convert the plaintext into a ciphertext and back; that is why this setting is also known as the symmetric-key setting.
Note that the two parties may be two parties are separated in space, or same party communicating with itself over time.

1.2 The Syntax of Encryption

Formally, a private-key encryption scheme is defined by specifying a message space M along with three algorithms: a procedure for generating keys (Gen), a procedure for encrypting (Enc), and a procedure for decrypting (Dec). The message space M defines the set of "legal" messages, i.e., those supported by the scheme. The algorithms of the scheme have the following functionality:

1. The key-generation algorithm Gen is a probabilistic algorithm that outputs a key k chosen according to some distribution.

2. The encryption algorithm Enc takes as input a key k and a message m and outputs a ciphertext c. We denote by $Enc_k(m)$ the encryption of the plaintext m using the key k.
   The encryption algorithm may be probabilistic, ie ($Enc_k(m)$ might output a different ciphertext when run multiple times), and we write $c \leftarrow$

$Enc_k(m)$ to denote the possibly probabilistic process by which message m is encrypted using key k to give ciphertext c.

3. The decryption algorithm Dec takes as input a key k and a ciphertext c and outputs a plaintext m. We denote the decryption of the ciphertext c using the key k by $Dec_k(c)$.
   Dec is deterministic without loss of generality, since $Dec_k(c)$ must give the same output every time it is run.

An encryption scheme must satisfy the following correctness requirement: for every key k output by Gen and every message $m \in M$, it holds that

$$Dec_k(Enc_k(m)) = m$$

The set of all possible keys output by the key-generation algorithm is called the key space and is denoted by K.

1.3 Kerckhoffs' principle

`The cipher method must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience.`

The principle states that security should not rely on the encryption scheme being secret, but solely on secrecy of the key.
The reasons for the same are:

1. It is significantly easier to maintain secrecy of a short key than to keep secret a (more complicated) encryption scheme.

2. In case the honest parties' shared, secret information is ever exposed, it will be much easier for them to change the key than to replace the encryption scheme.

3. There is a significant benefit to encouraging public review of that scheme in order to check for possible weaknesses. In fact, it is very dangerous to use a proprietary, "home-brewed" algorithm (i.e., a non-standardized algorithm designed in secret) since published designs undergo public peer review and are therefore likely to be stronger.

2 Historical Ciphers

We discuss some historical encryption schemes and show that they are insecure. Hence we aim:

1. to highlight the weaknesses of heuristic approaches to cryptography
2. to demonstrate that simple approaches to achieving secure encryption are unlikely to succeed

2.1 The Mono-Alphabetic Substitution Cipher

2.1.1 Mathematical Analysis

- The message space consists of arbitrary length strings of English letters with punctuation, spaces, and numerals removed, and with no distinction between upper and lower case.
- Algorithm Gen outputs an arbitrary map on the alphabet subject only to the constraint that it be one-to-one (so that decryption is possible). Hence the key is a bijection of the alphabet $f : A_1\{a, \ldots, z\} \to A_2\{A \ldots Z\}$.
- Algorithm Enc takes the key and a plaintext and replaces each letter of the plaintext $c_i to f(c_i)$.
- Algorithm Dec takes the key and a ciphertext and replaces each letter of the plaintext $c_i to f^{-1}(c_i)$.

2.1.2 Security Analysis

- The key space consists of all bijections, or permutations, of the alphabet. Hence, there are $26! \approx 2^{88}$ keys now, making a brute force attack infeasible.
- Assume English-language text is being encrypted (i.e., the text is grammatically correct English writing, not just text written using characters of the English alphabet). The mono-alphabetic substitution cipher can then be attacked by utilizing statistical properties of the English language. The frequency distribution of individual letters in English-language text is known. Of course, very short texts may deviate from this distribution, but even texts consisting of only a few sentences tend to have distributions that are very close to it.

  The attack works by tabulating the frequency distribution of characters in the ciphertext, i.e., recording that A appeared 12% of the time, B appeared 3% of the time, and so on. These frequencies are then compared to the known letter frequencies of normal English text. One can then guess parts of the mapping defined by the key based on the observed frequencies.

  Some of the guesses may be wrong, but enough of the guesses will be correct to enable relatively quick decryption (especially utilizing other

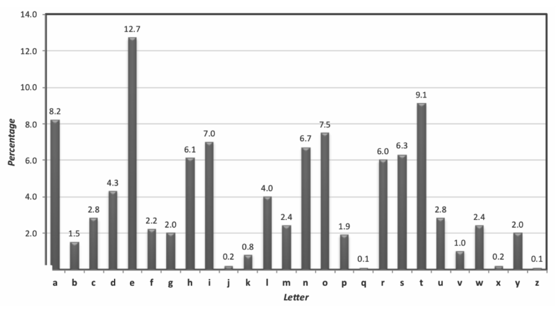

Figure 1: Average letter frequencies for English-language text

knowledge of English, such as the fact that q is generally followed by u, and that h is likely to appear between t and e).

This attack is rather difficult to automate, because a computer cannot describe what "makes sense" in English text.

2.2 Caeser's Cipher

A Caesar cipher is a simple method of encoding messages. Caesar ciphers use a substitution method where letters in the alphabet are shifted by some fixed number of spaces to yield an encoding alphabet. A Caesar cipher with a shift of 1 would encode an A as a B, an M as an N, and a Z as an A, and so on. The method is named after Roman leader Julius Caesar, who used it in his private correspondence.

2.2.1 Mathematical Analysis

The Caeser's Cipher is also called Shift cipher in modern sense.

- The message space consists of arbitrary length strings of English letters with punctuation, spaces, and numerals removed, and with no distinction between upper and lower case.
- Algorithm Gen outputs a uniform key $k \in \{0, \ldots, 25\}$.
- Algorithm Enc takes a key k and a plaintext and shifts each letter of the plaintext forward k positions (wrapping around at the end of the alphabet).
- Algorithm Dec takes a key k and a ciphertext and shifts every letter of the ciphertext backward k positions.

2.2.2 Security Analysis

The Caeser cipher is not a secure encryption scheme.

- The key space is small (only 26 possible keys), and an adversary can easily try all possible keys. The correct plaintext will certainly be on this list; moreover, if the ciphertext is "long enough" then the correct plaintext will likely be the only candidate on the list that "makes sense".
- Frequency analysis can be used to attack the Caesar cipher.
- We associate the letters of the English alphabet with $0, \ldots, 25$. Let p_i, with $0 \leq p_i \leq 1$, denote the frequency of the ith letter in normal English text. Calculation gives

  $$\sum_{i=0}^{25} p_i^2 \approx 0.065$$

  Now, say we are given some ciphertext and let q_i denote the frequency of the ith letter of the alphabet in this ciphertext; i.e., q_i is simply the number of occurrences of the ith letter of the alphabet in the ciphertext divided by the length of the ciphertext. If the key is k, then pi should be roughly equal to $q_i + k$ for all i because the ith letter is mapped to the (i + k)th letter. (We use i+k instead of the more cumbersome [i+k mod 26].) Thus, if we compute

  $$I_q = \sum_{i=0}^{25} p_i q_{i+j}$$

  for each value of $j \in 0, \ldots, 25$, then we expect to find that $I_k \approx 0.065$ (where k is the actual key), whereas I_j for $j \neq k$ will be different from 0.065.

  This leads to a key-recovery attack that is easy to automate: compute I_j for all j, and then output the value k for which I_k is closest to 0.065.

2.3 The Vigenere Cipher

The Vigenere cipher is a method of encrypting alphabetic text where each letter of the plaintext is encoded with a different Caesar cipher, whose increment is determined by the corresponding letter of another text, the key. It is hence a poly-alphabetic shift cipher.

For example, encryption of the message tellhimaboutme using the key cafe would work as follows:

Plaintext	tellhimaboutme
Key	cafecafecafeca
Ciphertext	VEQPJIREDOZXOE

Poly-alphabetic substitution ciphers "smooth out" the frequency distribution of characters in the ciphertext and make it harder to perform statistical analysis.

2.3.1 Security Analysis

If the key is sufficiently long, cracking this cipher appears daunting. Indeed, it had been considered by many to be "unbreakable," and although it was invented in the 16th century, a systematic attack on the scheme was only devised hundreds of years later.

- If the length of the key is known, then attacking the vignere cipher is relatively easy. Specifically, say the length of the key, also called the period, is t. Write the key k as $k = k_1 \dots k_t$ where each k_i is a letter of the alphabet. An observed ciphertext $c = c_1 c_2 \dots$ can be divided into t parts where each part can be viewed as having been encrypted using the shift cipher with key k_i. The key-recovery attack is then to treat each part as a shift cipher and use the attack on the shift cipher to recover the key k_i for each i. This attack is successful because the shift cipher is insecure.

  Now, as long as the maximum length T of the key is not too large, we can simply repeat the above attack T times (once for each possible value $t \in 1, \dots, T$). This leads to at most T different candidate plaintexts, among which the true plaintext will likely be easy to identify. So an unknown key length is not a serious obstacle.

- There are also more efficient ways to determine the key length from an observed ciphertext. One is to use Kasiski's method, published in the mid-19th century. The first step here is to identify repeated patterns of length 2 or 3 in the ciphertext. These are likely the result of certain bigrams or trigrams that appear frequently in the plaintext. For example, consider the common word "the." This word will be mapped to different ciphertext characters, depending on its position in the plaintext. However, if it appears twice in the same relative position, then it will be mapped to the same ciphertext characters. For a sufficiently long plaintext, there is thus a good chance that "the" will be mapped repeatedly to the same ciphertext characters.

- An alternative approach, called the index of coincidence method, is more methodical and hence easier to automate. The frequencies of the characters in the sequence of any stream are expected to be identical to the character frequencies of standard English text in some shifted order.

  In more detail: let q_i denote the observed frequency of the ith letter in this stream; this is simply the number of occurrences of the ith letter of the alphabet divided by the total number of letters in the stream. If the shift used here is j (i.e., if the first character k1 of the key is equal to j), then for all i we expect $q_{i+j} \approx p_i$, where p_i is the frequency of the ith letter of the alphabet in standard English text. (Once again, we use q_{i+j} in place of $q_{[i+j mod 26]}$.) But this means that the sequence $q0, \dots, q_{25}$ is just the sequence $p_0, \dots, p_{25}$ shifted j places. As a consequence,

$$\sum_{i=0}^{25} q_i^2 \approx \sum_{i=0}^{25} p_i^2 \approx 0.065$$

This leads to a nice way to determine the key length t. For $\tau = 1, 2, \ldots, T$, look at the sequence of ciphertext characters $c_1, c_{1+\tau}, c_{1+2\tau}, \ldots$ and tabulate $q_0, \ldots, q_{25}$ for this sequence. Then compute

$$S_\tau = \sum_{i=0}^{25} q_i^2$$

When $\tau = t$ we expect $S_\tau \approx 0.065$, as discussed above. On the other hand, if τ is not a multiple of t we expect that all characters will occur with roughly equal probability in the sequence $c_1, c_{1+\tau}, c_{1+2\tau}, \ldots$, and so we expect $q_i \approx 1/26$ for all i. In this case we will obtain

$$S_\tau \approx \sum_{i=0}^{25} \frac{1}{26}^{2} \approx 0.038$$

The smallest value of τ for which $S_\tau \approx 0.065$ is thus likely the key length.

One can further validate a guess τ by carrying out a similar calculation using the second stream $c_2, c_{2+\tau}, c_{2+2\tau}, \ldots$, etc

2.4 Conclusions

The above attacks on the Vigenere cipher require a longer ciphertext than the attacks on previous schemes. For example, the index of coincidence method requires $c_1, c_{1+t}, c_{1+2t}, \ldots$ (where t is the actual key length) to be sufficiently long in order to ensure that the observed frequencies are close to what is expected; the ciphertext itself must then be roughly t times larger. Similarly, the attack we showed on the mono-alphabetic substitution cipher requires a longer ciphertext than the attack on the shift cipher (which can work for encryptions of even a single word). This illustrates that a longer key can, in general, require the cryptanalyst to obtain more ciphertext in order to carry out an attack.

(Indeed, the Vigenere cipher can be shown to be secure if the key is as long as what is being encrypted. We will see a related phenomenon in the next chapter.)

The learnings from the Historic Ciphers adopted into the modern cryptography are:

- We need to have formal definitions to quantify our goals and understand when we have achieved them.

- We need to have precise assumptions to prove the same.

3 Modern Definitions

Our security goal should be: regardless of any information an attacker already has, a ciphertext should leak no additional information about the underlying plaintext. This definition of secure encryption scheme is suitable for all potential applications in which secrecy is required.
The following are several plausible options for the threat model in the context of encryption; standard ones, in order of increasing power of the attacker:

1. **Ciphertext-only attack**: This is the most basic attack, where the adversary just observes a ciphertext (or multiple ciphertexts) and attempts to determine information about the underlying plaintext (or plaintexts).

2. **Known-plaintext attack**: Here, the adversary is able to learn one or more plaintext/ciphertext pairs generated using some key. The aim of the adversary is then to deduce information about the underlying plaintext of some other ciphertext produced using the same key.

3. **Chosen-plaintext attack**: In this attack, the adversary can obtain plaintext/ciphertext pairs, as above, for plaintexts of its choice.

4. **Chosen-ciphertext attack**: The final type of attack is one where the adversary is additionally able to obtain (some information about) the decryption of ciphertexts of its choice, e.g., whether the decryption of some ciphertext chosen by the attacker yields a valid English message. The adversary's aim, once again, is to learn information about the underlying plaintext of some other ciphertext (whose decryption the adversary is unable to obtain directly) generated using the same key.

NOTE: This specifies what "power" the attacker is assumed to have, but does not place any restrictions on the adversary's strategy. This is an important distinction: we specify what we assume about the adversary's abilities, but we do not assume anything about how it uses those abilities.

4 Perfectly Secret Encryption

An encryption scheme is defined by three algorithms Gen, Enc, and Dec, as well as a specification of a message space M with $|M| > 1$.We denote by K the (finite) key space, i.e., the set of all possible keys that can be output by Gen. We let C denote the set of all possible ciphertexts that can be output by $Enc_k(m)$, for all possible choices of $k \in K$ and $m \in M$ (and for all random choices of Enc in case it is randomized).
In the definitions and theorems below, we refer to probability distributions over K, M, and C.

Theorem 4.1. *By redefining the key space, we may assume that the key-generation algorithm Gen chooses a uniform key from the key space, without changing $Pr[C = c|M = m]$ for any m, c.*

Proof. Let R be the set of random tapes used by the algorithm Gen to produce a key $k \in K$.

Assume that all the tapes have a uniform distribution over R. The probability of choosing a particular key k_i fed to the algorithm Enc is still $Pr[K = k_i]$.

Now, let us change Enc to take as input a key k' from R and then internally calculate k previously calculated by Gen, followed by using k to encrypt the message. This change does not affect the distribution of the ciphertexts C.

This shows that $Pr[C = c|M = m]$ remains unchanged when using the new key space R. Hence, by redefining the key space, we may assume that the key-generation algorithm Gen chooses a uniform key from the key space, without changing $Pr[C = c|M = m]$ for any m, c. □

We let K be the random variable denoting the value of the key output by Gen; thus, for any $k \in K$, $Pr[K = k]$ denotes the probability that the key output by Gen is equal to k.

Similarly, we let M be the random variable denoting the message being encrypted, so $Pr[M = m]$ denotes the probability that the message takes on the value $m \in M$.

The probability distribution of the message is not determined by the encryption scheme itself, but instead reflects the likelihood of different messages being sent by the parties using the scheme, as well as an adversary's uncertainty about what will be sent.

K and M are required to be independent, i.e., what is being communicated by the parties must be independent of the key they share.

Fixing an encryption scheme and a distribution over M determines a distribution over the space of ciphertexts C given by choosing a key $k \in K$ (according to Gen) and a message $m \in M$ (according to the given distribution), and then computing the ciphertext $c \leftarrow Enc_k(m)$. We let C be the random variable denoting the resulting ciphertext and so, for $c \in C$, write $Pr[C = c]$ to denote the probability that the ciphertext is equal to the fixed value c.

We now define the notion of perfect secrecy for an encryption scheme. We imagine an adversary who knows the probability distribution of M (likelihood that different messages will be sent), the encryption scheme being used, but doesnt know the key shared by the parties. The adversary can eavesdrop on the parties' communication, and thus observe this ciphertext. (That is, this is a ciphertext-only attack, where the attacker sees only a single ciphertext). For a scheme to be perfectly secret, observing this ciphertext should have no effect on the adversary's knowledge regarding the actual message that was sent; in other words, the a posteriori probability that some message $m \in M$ was sent, conditioned on the ciphertext that was observed, should be no different from the a priori probability that m would be sent.

This means that the ciphertext reveals nothing about the underlying plaintext, and the adversary learns absolutely nothing about the plaintext that was encrypted. Formally,

Definition 4.1. *An encryption scheme (Gen,Enc,Dec) with message space M is perfectly secret if for every probability distribution for M, every message $m \in M$,* *and every ciphertext $c \in C$ for which $Pr[C = c] > 0$:*

$$Pr[M = m|C = c] = Pr[M = m]$$

(The requirement that $Pr[C = c] > 0$ is a technical one needed to prevent conditioning on a zero-probability event)

An equivalent way to state the definition of perfect secrecy is that an encryption scheme is perfectly secret iff the distribution of the ciphertext does not depend on the plaintext.Formally:

Definition 4.2. *An encryption scheme (Gen,Enc,Dec) with message space M is perfectly secret iff for any two messages $m, m' \in M$, the distribution of the ciphertext when m is encrypted should be identical to the distribution of the ciphertext when m' is encrypted. I.e. for every $m, m' \in M$, and every $c \in C$, we have for every $m, m' \in M$ and every $c \in C$:*

$$Pr[EncK(m) = c] = Pr[EncK(m') = c]$$

(where the probabilities are over choice of K and any randomness of Enc)

Proof. We will prove that the two definitions are equivalent.

The key observation is that for any scheme, any distribution on M, any $m \in M$ for which $Pr[M = m] > 0$, and any $c \in C$, we have

$$\begin{aligned} Pr[C = c|M = m] &= Pr[Enc_K(M) = c|M = m] \\ &= Pr[Enc_K(m) = c|M = m] \\ &= Pr[Enc_K(m) = c] \end{aligned}$$

Take the uniform distribution over M. If the scheme is perfectly secret then $Pr[M = m|C = c] = Pr[M = m]$, and so $Pr[C = c|M = m] = Pr[C = c]$. Since m and c were arbitrary, this shows that for every $m, m' \in M$ and every $c \in C$,

$$Pr[Enc_K(m) = c] = Pr[Enc_K(m') = c]$$

.

Conversely, if the scheme satisfies the second definition, then for every $m, m' \in M$ and every $c \in C$, we have $Pr[Enc_K(m) = c] = Pr[Enc_K(m') = c]$. This implies that for every $m \in M$ and every $c \in C$, we have $Pr[C = c|M = m] = Pr[C = c]$. Hence, for every $m \in M$ and every $c \in C$, we have $Pr[M = m|C = c] = Pr[M = m]$. □

4.1 Perfect indistinguishability

Consider an adversary A first specifies two arbitrary messages $m_0, m_1 \in M$. Next, a key k is generated using Gen. Then, one of the two messages specified by A is chosen (each with probability 1/2) and encrypted using k; the resulting ciphertext is given to A. Finally, A outputs a "guess" as to which of the two messages was encrypted; A succeeds if it guesses correctly. An encryption scheme is perfectly indistinguishable if no adversary A can succeed with probability better than 1/2. (Note that, for any encryption scheme, A can succeed with probability 1/2 by outputting a uniform guess; the requirement is simply that no attacker can do any better than this.) Formally, let $\Pi = (Gen, Enc, Dec)$ be an encryption scheme with message space M. Let A be an adversary, which is formally just a (stateful) algorithm. We define an experiment $PrivK^{eav}_{A,\Pi}$, based, on A and Π, as follows:

1. The adversary A outputs two messages $m_0, m_1 \in M$.
2. A key k is generated using Gen.
3. A uniform bit $b \in \{0, 1\}$ is chosen. Cipher text $c \leftarrow Enc_k(m_b)$ is computed and given to A. We refer to this as the challenge ciphertext.
4. A outputs a guess $b' \in \{0, 1\}$.
5. The output of the experiment is defined to be 1 if $b' = b$ and 0 otherwise.

Definition 4.3. *An encryption scheme* $\Pi = (Gen, Enc, Dec)$ *with message space M is perfectly indistinguishable if for every adversary A, the probability that A succeeds in the experiment* $PrivK^{eav}_{A,\Pi}$ *is at most 1/2:*

Theorem 4.2. *Encryption scheme* Π *is perfectly secret if and only if it is perfectly indistinguishable.*

4.2 The One-Time Pad

The one-time pad is an encryption scheme described as follow. The key space is the set of all strings of the same length as the message space, and the encryption and decryption algorithms are defined by the bitwise XOR operation.

The one-time pad is perfectly secret because the distribution of the ciphertext is independent of the plaintext.

The drawbacks of the same are:

1. The key must be as long as the message, and the key must be random.
2. The key must be shared between the parties in advance.
3. The key can be used only once.

4.3 Limitations of Perfect Secrecy

Any perfectly secret encryption scheme must have a key space that is at least as large as the message space. This is an inherent limitation of perfect secrecy and not a limitation of any particular scheme.

Theorem 4.3. *If (Gen,Enc,Dec) is a perfectly secret encryption scheme with message space M and key space K, then* $|K| \geq |M|$.

Proof. Assume to the contrary, that $|K| < |M|$. Let M(c) be the set of all messages m such that $Dec_k(c) = m$ for some key k for a given ciphertext c. Hence, $|M(c)| \leq |K| < |M|$.

So, $\exists m' \in M$ such that $m' \notin M(c)$. This implies that $Pr[M = m'|C = c] = 0 \neq Pr[M = m']$. So, the encryption scheme is not perfectly secret. $\square$

A related theorem is Shannon's Theorem, which provides a characterization of a perfectly secret encryption scheme. This characterization says that, under certain conditions, the key-generation algorithm Gen must choose the key uniformly from the set of all possible keys; moreover, for every message m and ciphertext c there is a unique key mapping m to c.

The theorem as stated here assumes —M—= —K—= —C—, meaning that the sets of plaintexts, keys, and ciphertexts all have the same size.

Theorem 4.4. *Let $\Pi = (Gen, Enc, Dec)$ be an encryption scheme with message space M, key space K, and ciphertext space C, where $|M| = |K| = |C|$. The scheme is perfectly secret if and only if:*

1. *The key-generation algorithm Gen outputs a key $k \in K$ uniformly at random.*
2. *For every $m \in M$ and $c \in C$, there exists a unique key $k \in K$ such that $Enc_k(m) = c$.*

Proof. We first prove that if the encryption scheme satisfies conditions 1 and 2, then it is perfectly secret.

Fix arbitrary $c \in C$ and $m \in M$. Let k be the unique key, guaranteed by condition 2, for which $Enc_k(m) = c$. Then,

$$Pr[EncK(m) = c] = Pr[K = k] = 1/|K|$$

where the final equality holds by condition 1. Since this holds for arbitrary m and c, the scheme is perfectly secret.

For the second direction, we assume that the scheme is perfectly secret and prove that it satisfies conditions 1 and 2.

Fix an arbitrary $c \in C$. There must be some message m^* for which $Pr[Enc_K(m^*) = c] \neq 0$. Hence, $Pr[Enc_K(m) = c] \neq 0$ for every $m \in M$. In other words, if we let $M = \{m_1, m_2, \dots\}$, then for each $m_i \in M$ we have a nonempty set of keys $K_i \subset K$ such that $Enc_k(m_i) = c$ if and only if $k \in K_i$. Moreover, when $i \neq j$, K_i and K_j must be disjoint or else correctness fails to hold. Since $|K| = |M|$, we see that each K_i contains only a single key k_i, as required by condition 2. Now, for any $m_i, m_j \in M$ we have

$$Pr[K = k_i] = Pr[Enc_K(m_i) = c] = Pr[Enc_K(m_j) = c] = Pr[K = k_j]$$

Since this holds for all $1 \leq i, j \leq |M| = |K|$, and $k_i \neq k_j$, we have $Pr[K = k_i] = Pr[K = k_j] = 1/|K|$. $\square$

5 Computational Security

While perfect secrecy is a worthwhile goal, it is also unnecessarily strong. Perfect secrecy requires that absolutely no information about an encrypted message is leaked, even to an eavesdropper with unlimited computational power. For all practical purposes, however, an encryption scheme would still be considered secure if it leaked information with some tiny probability to eavesdroppers with bounded computational power.

Computational security definitions take into account computational limits on an attacker, and allow for a small probability that security is violated, in contrast to notions (such as perfect secrecy) that are information-theoretic in nature.

The relaxations we now make are:

1. Security is only guaranteed against efficient adversaries that run for some feasible amount of time. This means that given enough time (or sufficient computational resources) an attacker may be able to violate security.

2. Adversaries can potentially succeed (i.e., security can potentially fail) with some very small probability

5.1 Concrete approach to Computational Security

The concrete approach to computational security quantifies the security of a cryptographic scheme by explicitly bounding the maximum success proba- bility of a (randomized) adversary running for some specified amount of time or, more precisely, investing some specified amount of computational effort.

Thus, a concrete definition of security takes the following form:

Definition 5.1. *A scheme is (t, ϵ)-secure if any adversary running for time atmost t succeeds in breaking the scheme with probability at most ϵ.*

It may be more convenient to measure running time in terms of CPU cycles. A typical scheme can be such that no adversary using at most 2^{80} cycles can break the scheme with probability better than 2^{-60}.

The concrete approach is important in practice, since concrete guarantees are what users of a cryptographic scheme are ultimately interested in. However, precise concrete guarantees are difficult to provide. Moreover, we need to precisely define what we mean by “running time” and “computational effort” in order to make the definition of security precise.

5.2 Asymptotic approach to Computational Security

The issues above are dealed by precise definitions of running time and computational effort, and by providing guarantees that hold for all possible adversaries. This is the asymptotic approach to computational security.

We introduce an integer-valued security parameter (denoted by n) that parameterizes both cryptographic schemes as well as all involved parties (i.e., the honest parties as well as the attacker). When honest parties use a scheme (e.g., when they generate a key), they choose some value for the security parameter; for the purposes of this discussion, one can view the security parameter as corresponding to the length of the key. We also view the running time of the adversary, as well as its success probability, as functions of the security parameter rather than as fixed, concrete values. Then:

1. We equate "efficient adversaries" with randomized (i.e., probabilistic) algorithms running in time polynomial in n. This means there is some polynomial p such that the adversary runs for time at most p(n) when the security parameter is n. We also require—for real-world efficiency—that honest parties run in polynomial time, although we stress that the adversary may be much more powerful (and run much longer) than the honest parties.

2. We equate the notion of "small probabilities of success" with success probabilities smaller than any inverse polynomial in n. Such probabilities are called negligible

For formally, a function f is called negligible if:

Definition 5.2. *A function f from the natural numbers to the non-negative real numbers is negligible if for every polynomial p there is an N such that for all $n > N$ it holds that $f(n) < \frac{1}{p(n)}$.*

A related result is:

Theorem 5.1. *For any polynomial p, the function $negl_2(n) = p(n) \cdot negl_1(n)$ is negligible.*

Note that this implies that if a certain event occurs with only negligible probability in some experiment, then the event occurs with negligible probability even if that experiment is repeated polynomially many times.

Let PPT stand for "probabilistic polynomial-time." A definition of asymptotic security then takes the following general form:

Definition 5.3. *A scheme is secure if any PPT adversary succeeds in breaking the scheme with at most negligible probability.*

One advantage of using (probabilistic) polynomial time as our notion of efficiency is that this frees us from having to specify our model of computation precisely, since the extended Church-Turing thesis states that all "reasonable" models of computation are polynomially equivalent. Thus, we need not specify whether we use Turing machines, boolean circuits, or random-access machines; we can present algorithms in high-level pseudocode and be confident that if our analysis shows that an algorithm runs in polynomial time, then any reasonable implementation of that algorithm will run in polynomial time.

This notion of security is asymptotic since it depends on a scheme's behavior for sufficiently large values of n.

We can view the security parameter as a mechanism that allows the honest parties to "tune" the security of a scheme to some desired level. (Increasing the security parameter also increases the time required to run the scheme, as well as the length of the key, so the honest parties will want to set the security parameter as small as possible subject to defending against the class of attacks they are concerned about.)

The ability to "increase security" by increasing the security parameter has important practical ramifications, since it enables honest parties to defend against increases in computing power. Note that the effect of a faster computer has been to make the adversary's job harder.

5.2.1 Necessity of the Relaxations in definitions of Computational Security

Computational secrecy introduces two relaxations of perfect secrecy: first, secrecy is guaranteed only against efficient adversaries; second, secrecy may "fail" with small probability. Both these relaxations are essential for achieving practical encryption schemes, and in particular for bypassing the negative results for perfectly secret encryption.

Assume we have an encryption scheme where the size of the key space K is smaller than the size of the message space M. (As shown in the previous chapter, this means the scheme cannot be perfectly secret.) Regardless of how the encryption scheme is constructed, following results hold:

- Given a ciphertext c, an adversary can decrypt c using all keys $k \in K$. This gives a list of all the messages to which c can possibly correspond. Since this list cannot contain all of M (because $|K| < |M|$), this attack leaks some information about the message that was encrypted.

  Moreover, say the adversary carries out a known-plaintext attack and learns that ciphertexts $c_1, \ldots, c_l$ correspond to the messages $m_1, \ldots, m_l$, respectively. The adversary can again try decrypting each of these ciphertexts with all possible keys until it finds a key k for which $Dec_k(c_i) = m_i$ for all i. Later, given a ciphertext c that is the encryption of an unknown message m, it is almost surely the case that $Dec_k(c) = m$.

  Brute-force attacks like the above allow an adversary to "succeed" with probability ≈ 1 in time $O(|K|)$.

- Consider again the case where the adversary learns that ciphertexts $c_1, \ldots, c_l$ correspond to messages $m_1, \ldots, m_l$. The adversary can guess a uniform key $k \in K$ and check whether $Dec_k(c_i) = m_i$ for all i. If so, then, as above, the attacker can use k to decrypt anything subsequently encrypted by the honest parties.

  Here the adversary runs in constant time and "succeeds" with nonzero probability $1/|K|$.

Nevertheless, by setting $|K|$ large enough we can hope to achieve meaningful secrecy against attackers running in time much less than $|K|$ (so the attacker does not have sufficient time to carry out a brute-force attack), except possibly with small probability on the order of $1/|K|$.

6 Formally Defining Computational Secure Encryption

We now explicitly take into account the security parameter n. We also make two other changes: we allow the decryption algorithm to output an error (e.g., in case it is presented with an invalid ciphertext), and let the message space be the set $\{0,1\}^*$ of all (finite-length) binary strings by default.

Definition 6.1. *A private-key encryption scheme consists of three probabilistic polynomial-time algorithms (Gen,Enc,Dec) such that:*

- *The key-generation algorithm Gen takes as input 1^n (i.e., the security parameter written in unary) and outputs a key k; we write $k \leftarrow Gen(1^n)$ (emphasizing that Gen is a randomized algorithm). We assume without loss of generality that any key k output by $Gen(1^n)$ satisfies $|k| \geq n$.*
- *The encryption algorithm Enc takes as input a key k and a plaintext message $m \in \{0,1\}^*$, and outputs a ciphertext c. Since Enc may be randomized, we write this as $c \leftarrow Enc_k(m)$.*
- *The decryption algorithm Dec takes as input a key k and a ciphertext c, and outputs a message $m \in \{0,1\}^*$ or an error. We denote a generic error by the symbol $\perp$.*

It is required that for every n, every key k output by $Gen(1^n)$, and every $m \in \{0,1\}^*$, it holds that $Dec_k(Enc_k(m)) = m$.

If (Gen,Enc,Dec) is such that for k output by $Gen(1^n)$, algorithm Enc_k is only defined for messages $m \in \{0,1\}^{l(n)}$, then we say that (Gen,Enc,Dec) is a fixed-length private-key encryption scheme for messages of length l(n).

Almost always, $Gen(1^n)$ simply outputs a uniform n-bit string as the key. When this is the case, we omit Gen and define a private-key encryption scheme to be a pair of algorithms (Enc,Dec). Without significant loss of generality, we assume Dec is deterministic throughout this book, and so write $m := Dec_k(c)$.

The above definition considers stateless schemes, in which each invocation of Enc is independent of all prior invocations (and similarly for Dec). We assume encryption schemes are stateless (as in the above definition) unless explicitly noted otherwise.

6.1 EAV Security

The most basic security definition for private-key encryption is called EAV security (for "existential unforgeability under an adaptive chosen-ciphertext attack"). It aims to define security against a ciphertext-only attack where the adversary observes only a single ciphertext or, equivalently, security when a given key is used to encrypt just a single message.

We define the security goal by recalling that the idea is that the adversary should be unable to learn any partial information about the plaintext from the ciphertext.

6.1.1 Indistinguishability in the presence of an eavesdropper

We first begin by the idea of indistinguishability.

Consider our last experiment of perfect indistinguishability. Now, we keep the experiment $PrivK^{eav}_{A,\Pi}$ almost exactly the same (except for some technical differences discussed below), but introduce two important modifications in the definition itself:

- We now consider only adversaries running in polynomial time, whereas our previous considered even adversaries with unbounded running time.
- We now concede that the adversary might determine the encrypted message with probability negligibly better than 1/2.

We write $PrivK^{eav}_{A,\Pi}(n)$ to denote the experiment being run with security parameter n, and write $Pr[PrivK^{eav}_{A,\Pi}(n) = 1]$ to denote the probability that the output of experiment $PrivK^{eav}_{A,\Pi}(n)$ is 1.

A second difference is that we now explicitly require the adversary to output two messages m_0, m_1 of equal length. This means that, by default, we do not require a secure encryption scheme to hide the length of the plaintext.

We now give the formal definition, beginning with the experiment outlined above. The experiment is defined for a private-key encryption scheme $\Pi = (Gen, Enc, Dec)$, an adversary A, and a value n for the security parameter:

The adversarial indistinguishability experiment $PrivK^{eav}_{A,\Pi}(n)$:

1. The adversary A is given input 1^n, and outputs a pair of messages m_0, m_1 with $|m_0| = |m_1|$.
2. A key k is generated by running Gen(1^n), and a uniform bit $b \in \{0,1\}$ is chosen. Ciphertext $c \leftarrow Enc_k(m_b)$ is computed and given to A. We refer to c as the challenge ciphertext.
3. A outputs a bit b'.
4. The output of the experiment is defined to be 1 if b' = b, and 0 otherwise. If $PrivK^{eav}_{A,\Pi}(n) = 1$, we say that A succeeds.

The definition of indistinguishability states that an encryption scheme is secure if no PPT adversary A succeeds in guessing which message was encrypted in the above experiment with probability significantly better than random guessing (which is correct with probability 1/2).

Definition 6.2. *A private-key encryption scheme $\Pi = (Gen, Enc, Dec)$ has indistinguishable encryptions in the presence of an eavesdropper, or is EAV-secure, if for all probabilistic polynomial-time adversaries A there is a negligible function negl such that, for all n,*

$$\Pr[PrivK^{eav}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

The probability above is taken over the randomness used by A and the randomness used in the experiment (for choosing the key and the bit b, as well as any randomness used by Enc).

Note that this definition is weaker that the perfect indistinguishability definition, and hence any perfectly secure encryption scheme is also EAV-secure.

An equivalent definition of indistinguishability is that every PPT adversary behaves the same whether it observes an encryption of m_0 or of m_1.Since A outputs a bit, "behaving the same" means it outputs 1 with almost the same probability in each case.

To formalize this, define $PrivK^{eav}_{A,\Pi}(n, b)$ as above except that the fixed bit $b \in \{0, 1\}$ is used (rather than being chosen at random). Let $out_A(PrivK^{eav}_{A,\Pi}(n, b))$ denote the output bit b' of A in this experiment.

Definition 6.3. *A private-key encryption scheme $\Pi = (Gen, Enc, Dec)$ has indistinguishable encryptions in the presence of an eavesdropper if for all PPT adversaries A there is a negligible function negl such that*

$$\left|\Pr[out_A(PrivK^{eav}_{A,\Pi}(n, 0)) = 1] - \Pr[out_A(PrivK^{eav}_{A,\Pi}(n, 1)) = 1]\right| \leq negl(n)$$

The default notion of secure encryption does not require the encryption scheme to hide the plaintext length and, in fact, all commonly used encryption schemes reveal the plaintext length (or a close approximation thereof).

The main reason for this is that it is impossible to support arbitrary length messages while hiding all information about the plaintext length.

6.1.2 Semantic Security

We define secure encryption by a definition called semantic security which formalizes exactly the notion of an adversary being unable to learn any partial information about the plaintext from the ciphertext. We build up to that definition by first introducing two weaker notions and showing that they are implied by EAV-security.

1. EAV-security implies that ciphertexts leak no information about individual bits of the plaintext.

   Formally, we show that if an EAV-secure encryption scheme (Enc,Dec) (if Gen is omitted, the key is a uniform n-bit string) is used to encrypt a uniform message $m \in \{0,1\}^l$, then for any i it is infeasible for an attacker given the ciphertext to guess the i^{th} bit of m (here denoted by m_i) with probability much better than 1/2.

   Theorem 6.1. *Let $\Pi = (Enc, Dec)$ be a fixed-length private-key encryption scheme for messages of length l that is EAV-secure. Then for any ppt algorithm A there is a PPT algorithm A' such that for any distribution D over $\{0,1\}^l$ and any function $f : \{0,1\}^l \to \{0,1\}$, there is a negligible function negl such that:*

   $$|\Pr[A(1^n, Enc_k(m)) = f(m)] - \Pr[A'(1^n) = f(m)]| \leq negl(n),$$

   where the probability is taken over uniform $m \in \{0,1\}^l$ and $k \in \{0,1\}^n$, the randomness of A, and the randomness of Enc.

   Proof. The idea behind the proof of this theorem is that if it were possible to determine the ith bit of m from $Enc_k(m)$, then it would also be possible to distinguish between encryptions of messages m_0 and m_1 whose ith bits differ.

   □

2. EAV-security implies that no PPT adversary can learn any function f of the plaintext m from the ciphertext, regardless of the distribution D of m. This requirement is not trivial to define formally, because it needs to distinguish information that the attacker knows about the message due to D from information the attacker learns about the message from the ciphertext. (For example, if D is only over messages for which f evaluates to 1, then it is easy for an attacker to determine f(m). But in this case the attacker is not learning f(m) from the ciphertext.)

   This is taken into account in the definition by requiring that if there exists an adversary who, with some probability, correctly computes f(m) when given $Enc_k(m)$, then there exists an adversary that can correctly compute f(m) with almost the same probability without being given the ciphertext at all (and knowing only the distribution D of m).

   Theorem 6.2. *Let $\Pi = (Enc, Dec)$ be a fixed-length private-key encryption scheme for messages of length l that is EAV-secure. Then for any ppt algorithm A there is a PPT algorithm A' such that for any distribution D over $\{0,1\}^l$ and any function $f : \{0,1\}^l \to \{0,1\}$, there is a negligible function negl such that:*

   $$|\Pr[A(1^n, Enc_k(m)) = f(m)] - \Pr[A'(1^n) = f(m)]| \leq negl(n),$$

where the first probability is taken over the choice of m according to D, uniform choice of $k \in \{0,1\}^n$, and the randomness of A and Enc, and the second probability is taken over the choice of m according to D and the randomness of A'.

Proof. Consider the probability that A successfully computes f(m) given $Enc_k(m)$. We claim that A should successfully compute f(m) given $Enc_k(1^l)$ with almost the same probability; otherwise, A could be used to distinguish between $Enc_k(m)$ and $Enc_k(1^l)$. The distinguisher is easily constructed: choose m according to D, and output $m_0 = m$, $m1 = 1^l$. When given a ciphertext c that is an encryption of either m_0 or m_1, invoke $A(1^n, c)$ and output 0 if and only if A outputs f(m). If A outputs f(m) when given an encryption of m with probability that is significantly different from the probability that it outputs f(m) when given an encryption of 1^l, then the described distinguisher violates def 6.3.

The above suggests the following algorithm A' that does not receive an encryption of m, yet computes f(m) almost as well as A does: $A'(1^n)$ chooses a uniform key $k \in \{0,1\}^n$, invokes A on $c \leftarrow Enc_k(1^l)$, and outputs whatever A does. By the above, we have that A outputs f(m) when run as a subroutine by A' with almost the same probability as when it receives $Enc_k(m)$. Thus, A' fulfills the property required by the theorem. □

The full definition of semantic security guarantees considerably more than above. The definition allows arbitrary (efficiently sampleable) distributions over messages, generated by some polynomial-time sampling algorithm Samp. The definition also takes into account arbitrary "external" information h(m) about the message m that may be available to the adversary via other means (e.g., because the message is used for some other purpose as well). It also allows messages of varying lengths, although it assumes the message length is revealed.

Definition 6.4. *A private-key encryption scheme $\Pi = (Enc, Dec)$ is semantically secure in the presence of an eavesdropper if for every PPT algorithm A there exists a PPT algorithm A' such that for any PPT algorithm Samp and polynomial-time computable functions f and h, the following is negligible:*

$$|\Pr[A(1^n, Enc_k(m), h(m)) = f(m)] - \Pr[A'(1^n, |m|, h(m)) = f(m)]|$$

where the first probability is taken over m output by $Samp(1^n)$, uniform choice of $k \in \{0,1\}^n$, and the randomness of Enc and A, and the second probability is taken over m output by $Samp(1^n)$ and the randomness of A'.

The adversary A is given the ciphertext $Enc_k(m)$ as well as the external information h(m), and attempts to guess the value of f(m). Algorithm A' also attempts to guess the value of f(m), but is given only the length of m and h(m). The security requirement states that A's probability of correctly guessing f(m) is about the same as that of A'. Intuitively, then, this means that the ciphertext $Enc_k(m)$ does not reveal any information about f(m) except for $|m|$.

Theorem 6.3. *A private-key encryption scheme has indistinguishable encryptions in the presence of an eavesdropper (i.e., is EAV-secure) if and only if it is semantically secure in the presence of an eavesdropper.*

6.2 Security for Multiple Encryptions

The previous subsection deals with the case where the communicating parties transmit a single ciphertext that is observed by an eavesdropper. It would be convenient, however, if the communicating parties could securely send multiple ciphertexts to each other—all generated using the same key—even if an eavesdropper might observe all of them. For such applications we need an encryption scheme secure for the encryption of multiple messages.

We begin with an appropriate definition of security for this setting. We first introduce an appropriate experiment defined for any encryption scheme Π, adversary A, and security parameter n:

The multiple-message eavesdropping experiment $PrivK^{mult}_{A,\Pi}(n)$:

1. The adversary A is given input 1^n, and outputs a pair of equal-length lists of messages $\vec{M}_0 = (m_{0,1}, \ldots, m_{0,t})$ and $\vec{M}_1 = (m_{1,1}, \ldots, m_{1,t})$, with $|m_{0,i}| = |m_{1,i}|$ for all i.
2. A key k is generated by running Gen(1^n), and a uniform bit $b \in \{0,1\}$ is chosen. For all i, the ciphertext $c_i \leftarrow Enc_k(m_{b,i})$ is computed and the list $\vec{C} = (c_1, \ldots, c_t)$ is given to A.
3. A outputs a bit b'.
4. The output of the experiment is defined to be 1 if b' = b, and 0 otherwise.

The definition of security is the same as before, except that it now refers to the above experiment.

Definition 6.5. *A private-key encryption scheme $\Pi = (Gen, Enc, Dec)$ has indistinguishable multiple encryptions in the presence of an eavesdropper if for all PPT adversaries A there is a negligible function negl such that*

$$\Pr[PrivK^{mult}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

Any scheme that has indistinguishable multiple encryptions in the presence of an eavesdropper clearly also satisfies Definition 6.2, since experiment $PrivK^{eav}_{A,\Pi}(n)$ corresponds to the special case of $PrivK^{mult}_{A,\Pi}(n)$ where the adversary outputs two lists containing only a single message each. In fact, our new definition is strictly stronger than Definition 6.2.

Theorem 6.4. *If* Π *is a encryption scheme in which Enc is a deterministic function of the key and the message, then* Π *cannot have indistinguishable multiple encryptions in the presence of an eavesdropper.*

Proof. Consider an adversary A attacking the scheme Π (in the sense defined by experiment $PrivK^{mult}_{A,\Pi}(n)$): A outputs $\vec{M}_0 = (0^l, 0^l)$ and $\vec{M}_1 = (0^l, 1^l)$. (The first contains the same message twice, while the second contains two different messages.) Let $\vec{C} = (c_1, c_2)$ be the list of ciphertexts that A receives. If $c_1 = c_2$, then A outputs $b' = 0$; otherwise, A outputs $b' = 1$.

We now analyze the probability that $b' = b$. Since the algorithm is deterministic, encrypting the same message twice (using the same key) yields the same ciphertext. Thus, if $b = 0$ then we must have $c_1 = c_2$ and A outputs 0 in this case. On the other hand, if $b = 1$ then a different message is encrypted each time; hence $c_1 \neq c_2$ and A outputs 1.

This means that A always outputs the correct value of b with probability 1, which violates the definition of indistinguishable multiple encryptions. $\square$

The above might appear to show that Definition 6.5 is impossible to achieve using any encryption scheme. This is true as long as the encryption scheme is (stateless and) deterministic,and so encrypting the same message multiple times using the same key always yields the same result.

To construct a scheme secure for encrypting multiple messages, we must design a scheme in which encryption is randomized, so that when the same message is encrypted multiple times different ciphertexts can be produced. While achieving security for the encryption of multiple messages is important, we do not extensively consider this definition itself but instead focus on the stronger definition that we introduce in the following section.

6.3 CPA-Security

Chosen-plaintext attacks capture the ability of an adversary to exercise (partial) control over what the honest parties encrypt. Imagine a scenario in which two honest parties share a key k, and the attacker can influence those parties to encrypt messages $m_1, m_2, \ldots$ (using k) and send the resulting ciphertexts over a channel that the attacker can observe. At some later point in time, the attacker observes a ciphertext corresponding to some unknown message m encrypted using the same key k; let us even assume that the attacker knows that m is one of two possibilities m_0, m_1. Security against chosen-plaintext attacks means that even in this case the attacker cannot tell which of those two messages was encrypted with probability significantly better than random guessing. (For now we revert back to the case where the eavesdropper is given only a single encryption of an unknown message. Shortly, we will return to consideration of the multiple-message case.)

Note that chosen-plaintext attacks also encompass known-plaintext attacks-in which the attacker knows some of the messages being encrypted, even if it does not get to choose them-as a special case.

We define the chosen-plaintext security experiment $PrivK^{cpa}_{A,\Pi}(n)$ as follows:

We model chosen-plaintext attacks by giving the adversary A access to an encryption oracle, viewed as a “black box” that encrypts messages of A’s choice using a key k that is unknown to A. That is, we imagine A has access to an "oracle" $Enc_k(\cdot)$; when A queries this oracle by providing it with a message m as input, the oracle returns a ciphertext $c \leftarrow Enc_k(m)$ as the reply. (If Enc is randomized, the oracle uses fresh randomness each time it answers a query.) The adversary can interact with the encryption oracle adaptively, as many times as it likes.

Consider the following experiment defined for any encryption scheme $\Pi =$ (Gen,Enc,Dec), adversary A, and value n for the security parameter:

1. A key k is generated by running $Gen(1^n)$.
2. The adversary A is given input 1^n and oracle access to $Enc_k(\cdot)$, and outputs a pair of messages m_0, m_1 of the same length.
3. A uniform bit $b \in \{0, 1\}$ is chosen. Cipher text $c \leftarrow Enc_k(m_b)$ is computed and given to A.
4. The adversary A continues to have oracle access to $Enc_k(\cdot)$, and outputs a bit b'.
5. The output of the experiment is defined to be 1 if $b' = b$ and 0 otherwise.

Definition 6.6. *A private-key encryption scheme* $\Pi = (Gen, Enc, Dec)$ *has indistinguishable encryptions under a chosen-plaintext attack, or is CPA-secure if for all PPT adversaries A there is a negligible function negl such that*

$$\Pr[PrivK^{cpa}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

where the probability is taken over the randomness used by A, as well as the randomness used in the experiment.

6.4 CPA-Security for Multiple Encryptions

Definition 6.6 can be extended to the case of multiple encryptions in the same way that Definition 6.2 is extended to give Definition 6.5, i.e., by using lists of plaintexts.

Here, we take a different approach that is somewhat simpler and has the advantage of modeling attackers that can adaptively choose pairs of plaintexts to be encrypted. Specifically, we now give the attacker access to a "left-or-right" oracle $LR_{k,b}$ that, on input a pair of equal-length messages m_0, m_1, computes the ciphertext $c \leftarrow Enc_k(m_b)$ and returns c. That is, if b = 0 then the adversary always receives an encryption of the "left" plaintext, and if b = 1 then it always receives an encryption of the "right" plaintext. The bit b is a uniform bit chosen at the beginning of the experiment, and as in previous definitions the goal of the attacker is to guess b.

Consider the following experiment defined for any encryption scheme $\Pi =$ (Gen, Enc, Dec), adversary A, and value n for the security parameter:

The LR-oracle experiment $PrivK^{LR\text{-}cpa}_{A,\Pi}(n)$:

1. A key k is generated by running $Gen(1^n)$.
2. A uniform bit $b \in \{0,1\}$ is chosen.
3. The adversary A is given input 1^n and oracle access to $LR_k, b(\cdot,\cdot)$, as defined above.
4. The adversary A outputs a bit b'.
5. The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. In the former case, we say that A succeeds.

Definition 6.7. *Private-key encryption scheme Π has indistinguishable encryptions under a chosen-plaintext attack, or is CPA-secure for multiple encryptions, if for all PPT adversaries A there is a negligible function negl such that*

$$\Pr[PrivK^{LR\text{-}cpa}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + negl(n)$$

where the probability is taken over the randomness used by A and the randomness used in the experiment.

Note that an attacker given access to $LR_{k,b}$ can simulate access to an encryption oracle: to obtain the encryption of a message m, the attacker simply queries $LR_{k,b}(m,m)$. Given this observation, it is immediate that if Π is CPA-secure for multiple encryptions then it is also CPA-secure. It should also be clear that if Π is CPA-secure for multiple encryptions then it has indistinguishable multiple encryptions in the presence of an eavesdropper. In other words, Definition 6.7 is at least as strong as Definitions 6.6 and 6.5.

Theorem 6.5. *Any private-key encryption scheme that is CPA-secure is also CPA-secure for multiple encryptions.*

Proof. TODO □

7 Message Authentication Codes

7.1 Secrecy vs Integrity

A basic goal of cryptography is to enable parties to communicate securely.

Secure communication entails mechanisms that can be used to prevent a passive eavesdropper from learning anything about messages sent over an open channel.

However, not all security concerns are related to secrecy, and not all adversaries are limited to passive eavesdropping. In many cases, it is of equal or greater importance to guarantee message integrity (or message authentication) against an active adversary who can inject messages on the channel or modify messages in transit. We consider two motivating examples as follows:

1. Imagine first a user communicating with her bank over the Internet. When the bank receives a request to transfer $1,000 from the user's account to the account of some other user X, the bank has to consider the following:

   (a) Is the request authentic? That is, did the user in question really issue this request, or was the request issued by an adversary (perhaps X itself) who is impersonating the legitimate user?

   (b) Assuming a transfer request was issued by the legitimate user, is the request received by the bank exactly the same as what was sent by that user? Or was, e.g., the transfer amount modified as the request was sent across the Internet?

   Note that standard error-correction techniques do not suffice for the second concern. Error-correcting codes are only intended to detect and recover from "random" errors that affect a small portion of the transmission, but they do nothing to protect against a malicious adversary who can choose exactly where to introduce an arbitrary number of changes

2. The HTTP protocol used for web traffic is stateless, so when a client and server communicate in some session (e.g., when a user [client] shops at a merchant's [server's] website), any state generated as part of that session (e.g., the contents of the user's shopping cart) is often placed in a "cookie" that is stored by the user and included along with each message the user sends to the merchant. Assume the cookie stored by some user includes the items in the user's shopping cart along with a price for each item, as might be done if the merchant offers different prices to different users (reflecting discounts and promotions, or user-specific pricing).

   It would be undesirable for the user to be able to modify the cookie it stores so as to alter the prices of the items in its cart. The merchant thus needs a technique to ensure the integrity of the cookie that it stores at the user. Note that the contents of the cookie (namely, the items and their prices) are not secret and, in fact, must be known by the user. The problem here is purely one of integrity.

In general, one cannot assume the integrity of communication without taking specific measures to ensure it.

7.2 Encryption vs. Message Authentication

Encryption does not (in general) provide any integrity, and encryption should not be assumed to ensure message authentication unless it is specifically designed with that purpose in mind.

One might mistakenly think that encryption solves the problem of message authentication. (In fact, this is a common error.) This is due to the fuzzy, and incorrect, reasoning that since a ciphertext completely hides the contents of the message, an adversary cannot possibly modify an encrypted message in any meaningful way. Despite its intuitive appeal, this reasoning is completely false.

Consider the example of one-time pad, which is perfectly secure. Ciphertexts in this case are very easy to manipulate: flipping any bit in the ciphertext results in the same bit being flipped in the message that is recovered upon decryption. This simple attack can have severe consequences. As an example, consider the case of a user encrypting some dollar amount she wants to transfer from her bank account, where the amount is represented in binary. Flipping the least significant bit has the effect of changing this amount by $1, and flipping the 11th least significant bit changes the amount by more than $1,000! Interestingly, the adversary does not necessarily learn whether it is increasing or decreasing the initial amount, i.e., whether it is flipping a 0 to a 1 or vice versa.

7.3 MACs: Definitions

Message authentication code (MAC) is a mechanism which enables the communicating parties to know whether or not a message was tampered with. The aim of a message authentication code is to prevent an adversary from modifying a message sent by one party to another, or from injecting a new message, without the receiver detecting that the message did not originate from the intended party.

As in the case of encryption, this is only possible if the communicating parties have some secret information that the adversary does not know (otherwise nothing can prevent an adversary from impersonating the party sending the message). Here, we continue to consider the private-key setting where the communicating parties share a secret key.

7.3.1 Syntax of Message Authentication Code

Two users who wish to communicate in an authenticated manner begin by generating and sharing a secret key k in advance of their communication. When one party wants to send a message m to the other, she computes a tag t based on the message and the shared key, and sends the message m along with t to the other party. The tag is computed using a tag-generation algorithm Mac; thus, rephrasing what we have just said, the sender of a message m computes $t \leftarrow Mac_k(m)$ and transmits (m,t) to the receiver. Upon receiving (m,t), the second party verifies whether t is a valid tag on the message m (with respect to the shared key) or not. This is done by running a verification algorithm Vrfy that takes as input the shared key as well as a message m and a tag t, and indicates whether the given tag is valid.

Formally:

Definition 7.1. *A message authentication code (MAC) consists of three probabilistic polynomial-time algorithms* $(Gen, Mac, Vrfy)$ *such that:*

1. *The key-generation algorithm Gen takes as input the security parameter* 1^n *and outputs a key* k *with* $|k| \geq n$.

2. *The tag-generation algorithm Mac takes as input a key k and a message $m \in \{0,1\}^*$, and outputs a tag t. Since this algorithm may be randomized, we write this as $t \leftarrow Mac_k(m)$.*

3. *The deterministic verification algorithm $Vrfy$ takes as input a key k, a message m, and a tag t. It outputs a bit b, with $b = 1$ meaning valid and $b = 0$ meaning invalid. We write this as $b := Vrfy_k(m,t)$.*

It is required that for every n, every key k output by $Gen(1^n)$, and every $m \in \{0,1\}^*$, it holds that $Vrfy_k(m, Mac_k(m)) = 1$.

As with private-key encryption, $Gen(1^n)$ almost always simply chooses a uniform key $k \in \{0,1\}^n$, and we omit Gen in that case.

7.3.2 Security for Message Authentication Codes

The intuitive idea behind the definition is that no efficient adversary should be able to generate a valid tag on any "new" message that was not previously sent (and authenticated) by one of the communicating parties.

In the setting of message authentication, an adversary observing the communication between the honest parties may be able to see all the messages sent by those parties along with their corresponding tags. The adversary may also be able to influence the content of those messages, whether directly or indirectly (if, e.g., external actions of the adversary affect the messages sent by the parties). This is true, for example, in the web cookie example from earlier, where the user's own actions influence the contents of the cookie being stored on his computer.

To model the above, we allow the adversary to request tags for any messages of its choice. Formally, we give the adversary access to a MAC oracle $Mac_k(\cdot)$; the adversary can repeatedly submit any message m of its choice to this oracle, and is given in return a tag $t \leftarrow Mac_k(m)$.

An attacker "breaks" the scheme if it succeeds in outputting a forgery, i.e., if it outputs a message m along with a tag t such that:

1. t is a valid tag on the message m (i.e., $Vrfy_k(m,t) = 1$)

2. the message m was not previously authenticated by the honest parties (i.e., the adversary did not previously request a tag on the message m from its oracle)

These conditions imply that if the adversary were to send (m,t) to one of the honest parties, then that party would be mistakenly fooled into thinking that m originated from the other legitimate party (since $Vrfy_k(m,t) = 1$) even though it did not.

A MAC that cannot be broken in the above sense is said to be existentially unforgeable under an adaptive chosen-message attack. "Existentially unforgeable" refers to the fact that the adversary is unable to forge a valid tag on any message; this should hold even if the attacker can carry out an "adaptive chosen-message attack" by which it is able to obtain tags on arbitrary messages chosen adaptively during its attack.

The above discussion leads us to consider the following experiment for a message authentication code $\Pi = (Gen, Mac, Vrfy)$, an adversary A, and security parameter n:

The message authentication experiment $Mac - forge_{A,\Pi}(n)$:

1. A key k is generated by running Gen(1^n).
2. The adversary A is given input 1^n and oracle access to $Mac_k(\cdot)$. The adversary eventually outputs (m,t). Let Q denote the set of all queries that A submitted to its oracle.
3. A succeeds if and only if (1) $Vrfy_k(m,t) = 1$, (2) m was not previously queried to the oracle. In that case the output of the experiment is defined to be 1.

A MAC is secure if no efficient adversary can succeed in the above experiment with non-negligible probability.

Definition 7.2. *A message authentication code $\Pi = (Gen, Mac, Vrfy)$ is existentially unforgeable under an adaptive chosen-message attack, or just secure, if for all PPT adversaries A, there is a negligible function negl such that*

$$\Pr[Mac - forge_{A,\Pi}(n) = 1] \leq negl(n)$$

NOTE: The above definition, and MAC by itself, offers no protection against replay attacks. In a replay attack, an adversary records a message and its corresponding tag, and later sends the same message and tag to the receiver. This is why MACs are often used in conjunction with timestamps or nonces to prevent replay attacks.

8 CCA Security and Authenticated Encryption

Our analysis so far misses a case- secrecy in the presence of an active adversary. This adversary can not only eavesdrop on the communication between the parties, but can also modify the messages being sent.

Consider a scenario in which a sender encrypts a message m and then transmits the resulting ciphertext c. An attacker who can tamper with the communication can modify c to generate another ciphertext c' that is received by the other party. This receiver will then decrypt c' to obtain a message m'. If $m' \neq m$ (and $m' \neq \perp$), this is a violation of integrity. What is of interest to us here, however, is the potential impact on secrecy. In particular, if the attacker learns partial information about m'-say, from subsequent behavior of the receiver-might that reveal information about the original message m?

This type of attack, in which an adversary causes a receiver to decrypt ciphertexts that the adversary generates, is called a chosen-ciphertext attack. Chosen-ciphertext attacks are possible, in principle, any time an attacker has the ability to inject traffic on the channel between the sender and receiver.

8.1 Formal Definition: CCA Security

The adversary should have the ability not only to obtain the encryption of messages of its choice (as in a chosen-plaintext attack), but also to obtain the decryption of ciphertexts of its choice (with one exception discussed later). Formally, we give the adversary access to a decryption oracle $\mathrm{Dec}_k(\cdot)$ in addition to an encryption oracle $\mathrm{Enc}_k(\cdot)$.

Consider the following experiment for any private-key encryption scheme $\Pi = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$, adversary A, and value n for the security parameter.

The CCA indistinguishability experiment $\mathrm{PrivK}^{\mathrm{cca}}_{A,\Pi}(n)$:

1. A key k is generated by running $\mathrm{Gen}(1^n)$.
2. A is given input 1^n and oracle access to $\mathrm{Enc}_k(\cdot)$ and $\mathrm{Dec}_k(\cdot)$. It outputs a pair of equal-length messages m_0, m_1.
3. A uniform bit $b \in \{0,1\}$ is chosen, and then a challenge ciphertext $c \leftarrow \mathrm{Enc}_k(m_b)$ is computed and given to A.
4. The adversary A continues to have oracle access to $\mathrm{Enc}_k(\cdot)$ and $\mathrm{Dec}_k(\cdot)$, but is not allowed to query the latter on the challenge ciphertext itself. Eventually, A outputs a bit b'.
5. The output of the experiment is 1 if $b' = b$, and 0 otherwise. If the output of the experiment is 1, we say that A succeeds.

Definition 8.1. *A private-key encryption scheme Π has indistinguishable encryptions under a chosen-ciphertext attack, or is CCA-secure, if for all probabilistic polynomial-time adversaries A there is a negligible function negl such that:*

$$\Pr\left[PrivK^{cca}_{A,\Pi}(n) = 1\right] \leq \frac{1}{2} + negl(n),$$

where the probability is taken over all randomness used in the experiment.

8.2 Authenticated Encryption

CCA-security is extremely important, but is subsumed by an even stronger notion of security we introduce here. Until now, we have considered how to obtain secrecy (using encryption) and integrity (using message authentication codes) separately. The aim of authenticated encryption, defined below, is to achieve both goals simultaneously. It is best practice to always ensure secrecy and integrity by default in the private-key setting. Indeed, in many applications where secrecy is required it turns out that integrity is essential also. Moreover, a lack of integrity can sometimes lead to a breach of secrecy, as illustrated in the previous section.

8.3 Formal Definition: Authenticated Encryption

Consider the following experiment defined for a private-key encryption scheme $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$, adversary A, and value n for the security parameter:

The unforgeable encryption experiment $\text{Enc-Forge}_{A,\Pi}(n)$:

1. A key k is generated by running $\text{Gen}(1^n)$.
2. The adversary A is given input 1^n and access to an encryption oracle $\text{Enc}_k(\cdot)$. The adversary eventually outputs a ciphertext c. Let $m := \text{Dec}_k(c)$ and let Q denote the set of all queries that A submitted to its oracle.
3. A succeeds if and only if (1) $m \neq \perp$ and (2) $m \notin Q$. In that case the output of the experiment is defined to be 1.

Definition 8.2. *A private-key encryption scheme* Π *is unforgeable if for all probabilistic polynomial-time adversaries* A*, there is a negligible function negl such that:*

$$\Pr[\textit{Enc-Forge}_{A,\Pi}(n) = 1] \leq \textit{negl}(n).$$

We may now define an authenticated encryption scheme.

Definition 8.3. *A private-key encryption scheme is an authenticated encryption (AE) scheme if it is CCA-secure and unforgeable.*

It is also possible to capture both the above requirements in a definition involving a single experiment. The experiment is somewhat different from previous experiments we have considered, so we provide some motivation before giving the details.

The idea is to consider two different scenarios, and require that they be indistinguishable to an attacker. In the first scenario, which can be viewed as corresponding to the real-world context in which the adversary operates, the attacker is given access to both an encryption oracle and a decryption oracle. In the second case, which can be viewed as corresponding to an "ideal" scenario, these two oracles are changed as follows:

- In place of an encryption oracle, the attacker is given access to an oracle that encrypts a 0-string of the correct length. Formally, the attacker is given access to an oracle $\text{Enc}^0_k(\cdot)$ where $\text{Enc}^0_k(m) = \text{Enc}_k(0^{|m|})$. I.e., when requesting an encryption of m, the attacker is instead given an encryption of a 0-string of the same length as m.
- In place of a decryption oracle, the attacker is given access to an oracle $\text{Dec}^{\perp}(\cdot)$ that always returns the error symbol $\perp$.

If an attacker cannot distinguish the first scenario from the second, then this means (1) any new ciphertexts the attacker generates in the real world will be invalid (i.e., will generate an error upon decryption). This not only implies a strong form of integrity, but also makes chosen-ciphertext attacks useless.

Moreover, (2) the attacker cannot distinguish a real encryption oracle from an oracle that always encrypts 0s, which implies secrecy.

Formally, for a private-key encryption scheme Π, adversary A, and value n for the security parameter, define the following experiment: The authenticated-encryption experiment $\mathrm{PrivK}^{\mathrm{ae}}_{A,\Pi}(n)$:

1. A key k is generated by running $\mathrm{Gen}(1^n)$.
2. A uniform bit $b \in \{0,1\}$ is chosen.
3. The adversary A is given input 1^n and access to two oracles:
   - If $b = 0$, then A is given access to $\mathrm{Enc}_k(\cdot)$ and $\mathrm{Dec}_k(\cdot)$.
   - If $b = 1$, then A is given access to $\mathrm{Enc}^0_k(\cdot)$ and $\mathrm{Dec}^{\perp}(\cdot)$.

   A is not allowed to query a ciphertext c to its second oracle that it previously received as the response from its first oracle.
4. The adversary outputs a bit b'.
5. The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. In the former case, we say that A succeeds.

In the experiment, the attacker is not allowed to submit ciphertexts to the decryption oracle that it received from its encryption oracle, since this would lead to a trivial way to distinguish the two scenarios. We remark that in the "real" case (i.e., when $b = 0$) the attacker already knows the decryption of those ciphertexts, so there is not much point in making such queries, anyway.

Definition 8.4. *A private-key encryption scheme is an authenticated encryption (AE) scheme if for all probabilistic polynomial-time adversaries A there is a negligible function negl such that*

$$\Pr[\mathit{PrivK}^{ae}_{A,\Pi}(n) = 1] \leq \frac{1}{2} + \mathit{negl}(n).$$

We have given two definitions of authenticated encryption. Fortunately, the definitions 8.3 and 8.4 are equivalent:

Theorem 8.1. *A private-key encryption scheme satisfies definition 8.3 if and only if it satisfies definition 8.4.*

8.4 CCA Security vs. Authenticated Encryption

One can imagine applications where CCA-security is needed but authenticated encryption is not. One example might be when private-key encryption is used for key transport. As a concrete example, say a server gives a tamper-proof hardware token to a user, where the token stores a long-term key k. The server can share a fresh, short-term key k' with the token (that will remain unknown to the user) by giving the user $\mathrm{Enc}_k(k')$; the user is supposed to give this ciphertext to the token, which will decrypt it to obtain k'. CCA-security is necessary here because chosen-ciphertext attacks can be easily carried out by the user in this context. On the other hand, not much harm is done if the user can generate a valid ciphertext that causes the token to use some arbitrary key k'' that is uncorrelated with k'. (Of course, this depends on what the token does with this short-term key.)

Notwithstanding the above, most applications of private-key encryption in the presence of an active adversary do require integrity.

9 Key Distribution

We will now look into the question of *"How can the parties share a secret key in the first place?"*

Clearly, the key cannot simply be sent over the insecure communication channel because an eavesdropping adversary would then be able to observe the key and it would no longer be secret. Some other mechanism must be used.

In some situations, the parties may have access to a secure channel that they can use to reliably share a secret key. One common example is when the two parties are physically co-located at some point in time, during which they can share a key. Alternatively, the parties might be able to use a trusted courier service as a secure channel. We stress that even if the parties have access to a secure channel at some point, this does not make private-key cryptography useless: in the first example, the parties have a secure channel at one point in time but not later; in the second example, utilizing the secure channel might be slower and more costly than communicating over an insecure channel.

Relying on a secure channel to distribute keys, however, does not work well in many other situations. For example, consider a large, multinational corporation in which every pair of employees might need the ability to communicate securely, with their communication protected from other employees as well. It will be inconvenient, to say the least, for each pair of employees to meet so they can securely share a key; for employees working in different cities, this may even be impossible. Even if the current set of employees could somehow share keys with each other, it would be impractical for them to share keys with new employees who join after this initial sharing is done.

Even assuming these N employees are somehow able to securely share keys with each other, another significant drawback is that each employee would have to manage and store $N-1$ secret keys (one for each other employee in the company). In fact, this may significantly under-count the number of keys stored by each user, because employees may also need keys to communicate securely with remote resources such as databases, servers, printers, and so on. The proliferation of so many secret keys is a significant logistical problem. Moreover, all these keys must be stored securely. The more keys there are, the harder it is to protect them, and the higher the chance of some keys being stolen by an attacker. Computer systems are often infected by viruses, worms, and other forms of malicious software that can steal secret keys and send them quietly over the network to an attacker. Thus, storing keys on employees' personal computers is not always a safe solution.

9.1 Key Distribution Centers

One way to address some of the concerns from the previous section is to use a key-distribution center (KDC) to establish shared keys. Consider again the case of a large corporation where all pairs of employees must be able to communicate securely. In such a setting, we can leverage the fact that all employees may trust some entity—say, the system administrator—at least with respect to the security of work-related information. This trusted entity can then act as a KDC and help all the employees share pairwise keys.

The KDC can be used in an online fashion to generate keys "on demand" whenever two employees wish to communicate securely. KDC will share a (different) key with each employee, something that can be done securely on each employee's first day of work. Say the KDC shares key k_A with employee Alice, and k_B with employee Bob. At some later time, when Alice wishes to communicate securely with Bob, she can simply send the message "I, Alice, want to talk to Bob" to the KDC. (If desired, this message can be authenticated using the key shared by Alice and the KDC.) The KDC then chooses a new, random key—called a session key —and sends this key k to Alice encrypted using k_A, and to Bob encrypted using k_B. Once Alice and Bob both recover this session key, they can use it to communicate securely. When they are done with their conversation, they can (and should) erase the session key because they can always contact the KDC again if they wish to communicate at some later time.

Consider the advantages of this approach:

1. Each employee needs to store only one long-term secret key (namely, the one they share with the KDC). Employees still need to manage and store session keys, but these are short-term keys that are erased once a communication session concludes.

   The KDC needs to store many long-term keys. However, the KDC can be kept in a secure location and be given the highest possible protection against network attacks.

2. When an employee joins the organization, all that must be done is to set up a key between this employee and the KDC. No other employees need to update the set of keys they hold.

The drawbacks for the same are:

1. A successful attack on the KDC will result in a complete break of the system: an attacker can compromise all keys and subsequently eavesdrop on all network traffic. This makes the KDC a high-value target. Note that even if the KDC is well-protected against external attacks, there is always the possibility of an insider attack by an employee who has access to the KDC (for example, the IT manager).

2. The KDC is a single point of failure: if the KDC is down, secure communication is temporarily impossible. If employees are constantly contacting the KDC and asking for session keys to be established, the load on the KDC can be very high, thereby increasing the chances that it may fail or be slow to respond.

A simple solution to the second problem is to replicate the KDC. This works (and is done in practice), but also means that there are now more points of attack on the system. Adding more KDCs also makes it more difficult to add new employees, since updates must be securely propagated to every KDC.

9.1.1 Needham-Schroeder protocol

There are a number of protocols for secure key distribution using a KDC. We mention in particular the Needham-Schroeder protocol, which is commonly used.

We only highlight one feature of this protocol. When Alice contacts the KDC and asks to communicate with Bob, the KDC does not send the encrypted session key to both Alice and Bob as we have described earlier. Instead, the KDC sends to Alice the session key encrypted under Alice's key in addition to the session key encrypted under Bob's key. Alice then forwards the second ciphertext to Bob. The second ciphertext is sometimes called a ticket, and can be viewed as a credential that allows Alice to talk to Bob (and allows Bob to be assured that he is talking to Alice).

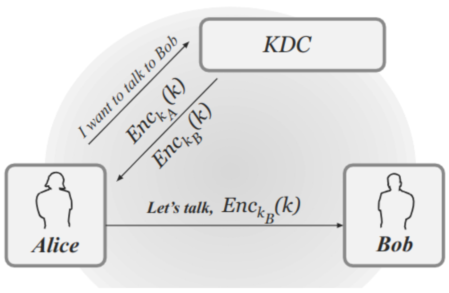

Figure 2: A general template for key-distribution protocols

The protocol was designed in this way to reduce the load on the KDC. In the protocol as described, the KDC does not need to initiate a second connection to Bob, and need not worry whether Bob is on-line when Alice initiates the protocol. Moreover, if Alice retains the ticket (and her copy of the session key), then she can re-initiate secure communication with Bob by simply re-sending the ticket to Bob, without the involvement of the KDC at all. (In practice, tickets expire and eventually need to be renewed. But a session could be re-established within some acceptable time period.)

10 Key Exchange

KDCs and protocols like Kerberos still require, at some point, a private and authenticated channel that can be used to share keys. (In particular, we assumed the existence of such a channel between the KDC and an employee on his or her first day.) Thus, they still cannot solve the problem of key distribution in open systems like the Internet, where there may be no private channel available between two users who wish to communicate.

To achieve private communication without ever communicating over a private channel, a radically different approach is needed. In 1976, Whitfield Diffie and Martin Hellman published a paper on asymmetry in the world; in particular, there are certain actions that can be easily performed but not easily reversed. For example, it is easy to multiply two large primes but difficult to recover those primes from their product (This is called the factoring problem). Diffie and Hellman realized that such phenomena could be used to derive interactive protocols for secure key exchange that allow two parties to share a secret key, via communication over a public channel, by having the parties perform operations that an eavesdropper cannot reverse. The existence of secure key-exchange protocols is quite amazing. It means that you and a friend could agree on a secret by simply shouting across a room (and performing some local computation); the secret would be unknown to anyone else, even if they had listened to everything that was said.

In this section we present the Diffie–Hellman key-exchange protocol. We prove its security against eavesdropping adversaries or, equivalently, under the assumption that the parties communicate over a public but authenticated channel (so an attacker cannot interfere with their communication). Security against an eavesdropping adversary is a relatively weak guarantee, and in practice key-exchange protocols must satisfy stronger notions of security.

10.1 Formal Definitions

We consider a setting with two parties—traditionally called Alice and Bob—who run a probabilistic protocol Π in order to generate a shared, secret key; Π can be viewed as the set of instructions for Alice and Bob in the protocol. Alice and Bob begin by holding the security parameter 1^n; they then run Π using (independent) random bits. At the end of the protocol, Alice and Bob output keys $k_A, k_B \in \{0,1\}^n$, respectively. The basic correctness requirement is that $k_A = k_B$. Since we will only deal with protocols that satisfy this requirement, we will speak simply of the key $k = k_A = k_B$ generated in some honest execution of Π. (Since Π is randomized the key will, in general, be different every time Π is run.)

We now turn to defining security. Intuitively, a key-exchange protocol is secure if the key output by Alice and Bob is completely hidden from an eavesdropping adversary. This is formally defined by requiring that an adversary who has eavesdropped on an execution of the protocol should be unable to distinguish the key k generated by that execution (and now shared by Alice and Bob) from a uniform key of length n. This is much stronger than simply requiring that the adversary be unable to guess k exactly, and this stronger notion is necessary if the parties will subsequently use k for some cryptographic application (e.g., as a key for a private-key encryption scheme).

Formalizing the above, let Π be a key-exchange protocol, A an adversary, and n the security parameter. We have the following experiment:

The key-exchange experiment $\mathbf{KE}^{A,\Pi}_{\mathbf{eav}}(n)$:

1. Two parties holding 1^n execute protocol Π. This results in a transcript trans containing all the messages sent by the parties, and a key k output by each of the parties.
2. A uniform bit $b \in \{0,1\}$ is chosen. If $b = 0$ set $\hat{k} := k$, and if $b = 1$ then choose uniform $\hat{k} \in \{0,1\}^n$.
3. A is given trans and $\hat{k}$, and outputs a bit b'.
4. The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. (In case $\mathrm{KE}^{A,\Pi}_{\mathrm{eav}}(n) = 1$, we say that A succeeds.)

A is given trans to capture the fact that A eavesdrops on the entire execution of the protocol and thus sees all messages exchanged by the parties. In the real world, A would not be given any key; in the experiment the adversary is given k only as a means of defining what it means for A to "break" the security of Π. That is, the adversary succeeds in "breaking" Π if it can correctly determine whether the key $\hat{k}$ is the real key corresponding to the given execution of the protocol, or whether $\hat{k}$ is a uniform key that is independent of the transcript. As expected, we say Π is secure if the adversary succeeds with probability that is at most negligibly greater than 1/2. That is:

Definition 10.1. *A key-exchange protocol* Π *is secure in the presence of an eavesdropper if for all probabilistic polynomial-time adversaries* A *there is a negligible function negl such that*

$$\Pr[KE^{A,\Pi}_{eav}(n) = 1] \leq \frac{1}{2} + negl(n).$$

The aim of a key-exchange protocol is almost always to generate a shared key k that will be used by the parties for some further cryptographic purpose, e.g., to encrypt and authenticate their subsequent communication using, say, an authenticated encryption scheme.

10.2 Diffie-Hellman key-exchange protocol

Let G be a probabilistic polynomial-time algorithm that, on input 1^n, outputs a description of a cyclic group G, its order q (with $\|q\| = n$), and a generator $g \in G$.

The Diffie-Hellman key-exchange protocol is as follows:

Common input: The security parameter 1^n

The protocol:

1. Alice runs $G(1^n)$ to obtain (G, q, g).
2. Alice chooses a uniform $x \in Z_q$, and computes $h_A := g^x$.
3. Alice sends (G, q, g, h_A) to Bob.
4. Bob receives (G, q, g, h_A). He chooses a uniform $y \in Z_q$, and computes $h_B := g^y$. Bob sends h_B to Alice and outputs the key $k_B := h_A^y$.
5. Alice receives h_B and outputs the key $k_A := h_B^x$.

In our description, we have assumed that Alice generates (G, q, g) and sends these parameters to Bob as part of her first message. In practice, these parameters are standardized and known to both parties before the protocol begins. In that case, Alice need only send h_A, and Bob need not wait to receive Alice's message before computing and sending h_B.

It is not hard to see that the protocol is correct: Bob computes the key

$$k_B = h_A^y = (g^x)^y = g^{xy}$$

and Alice computes the key

$$k_A = h_B^x = (g^y)^x = g^{xy},$$

and so $k_A = k_B$.

11 Public Key Cryptography

Public key cryptography is a method of secure communication that uses two keys, a public one which can be used by anyone to encrypt a message and a private key which is used to decrypt messages or to "sign" documents, that is, to add a digital signature to a document as a proof of it's authenticity. This method ensures that only the intended recipent can decrypt the message and that the signed message came from the claimed sender. Although there are cases in which the use of the keys might be switched. Public key cryptography is an essential part of services like confidential messaging, verification of data authenticity and for secure internet communications.
A cryptographic key is a parameter to control operations to ensure the sanctity of the data and the privacy of digital transactions and communications. The key difference between public key encryption and plain encryption is that in the former we have a different one has to be used for decryption. It must be ensured that the search for the private key from the public one with a decrypted message is a doomed quest. The applications of PKC include digital signature, key management and secure transmission of data.

There are a few vulnerabilities of the public key encryption:

- It is vulnerable to the brute force algorithm.

- In case a user misplaces their private key, this very method becomes the most vulnerable method.
- It is also very weak towards the third party (the man in the middle attack) wherein the third party could disrupt the public key communication by altering the public keys.
- If user private key used for certificate creation higher in the Public Key Infrastructure (PKI) server hierarchy is compromised, or accidentally disclosed, then a "man-in-the-middle attack" is also possible, making any subordinate certificate wholly insecure. This is also the weakness of public key Encryption.

The most widely used public key cryptography technique is Rivest-Shamir-Adleman or the **RSA** method.

11.1 RSA method

The RSA method relies on the difficulty in factorizing a large integer. In this method, the public key consists of two numbers where one number is a multiplication of two large prime numbers and private key is also derived from the same two prime numbers. So if somebody can factorize the public key, the private key is compromised, therefore the strength of encryption is based on the key size. If we were to double or triple it, this security would increase exponentially. RSA keys are typically 1024 or 2048 bits long (128 or 256 bytes big) factorising whom remains an infeasible task for now. The process of RSA is explained below.

11.1.1 Key Generation

- We select two prime numbers, say P=53 and Q=89.
- Now the first part of the public key is $n = P * Q$ which means n=4717.
- We now calculate a Euler's Totient Function, $\phi(n) = (P-1)(Q-1)$ which is 4717. Note: Carmichael's totient function is also used sometimes.
- We also require a small integral exponent e which is an integer and **not** a factor of $\phi(n)$, say e=2. Together, n and e make our public key.
- For the private key d, it is the modular multiplicative inverse of e modulo $\phi(n)$. Here, d=2359.

11.1.2 Encryption

To encrypt a message m, we need to convert it to an integer between 0 and n-1. This can be done using a reversible encoding scheme, such as ASCII or UTF-8. Once we have the integer representation of the message, we compute the ciphertext c as c $=m^e$ (mod n). This can be done efficiently using modular exponentiation algorithms, such as binary exponentiation.

11.1.3 Decryption

To decrypt the ciphertext c, we compute the plaintext m as m = c^d (mod n). Again, we can use modular exponentiation algorithms to do this efficiently.

11.1.4 Code

Here is a python code implementing RSA method.

```
import random
import math

prime = set()

public_key = None
private_key = None
n = None


def addPrimes():
    sieve = [True] * 250
sieve[0] = False
sieve[1] = False
for i in range(2, 250):
for j in range(i * 2, 250, i):
sieve[j] = False

for i in range(len(sieve)):
if sieve[i]:
prime.add(i)


def pickAPrime():
global prime
k = random.randint(0, len(prime) - 1)
it = iter(prime)
for _ in range(k):
next(it)

ret = next(it)
prime.remove(ret)
return ret


def setkeys():
global public_key, private_key, n
prime1 = pickAPrime()
```

```
prime2 = pickAPrime()

n = prime1 * prime2
fi = (prime1 - 1) * (prime2 - 1)

e = 2
while True:
if math.gcd(e, fi) == 1:
break
e += 1

public_key = e

d = 2
while True:
if (d * e) % fi == 1:
break
d += 1

private_key = d

def encrypt(message):
global public_key, n
e = public_key
encrypted_text = 1
while e > 0:
encrypted_text *= message
encrypted_text %= n
e -= 1
return encrypted_text

def decrypt(encrypted_text):
global private_key, n
d = private_key
decrypted = 1
while d > 0:
decrypted *= encrypted_text
decrypted %= n
d -= 1
return decrypted


def code(message):
encoded = []
# Calling the encrypting function in encoding function
for letter in message:
```

```
encoded.append(encrypt(ord(letter)))
return encoded


def decode(encoded):
s = ’’
# Calling the decrypting function decoding function
for num in encoded:
s += chr(decrypt(num))
return s


if _name_ == ’_main_’:
addPrimes()
setkeys()
message = "SoS was Fun"
coded = code(message)

print("Original message",message)
print("Encoded Message",’’.join(str(p) for p in coded))
print("Message decoded!",’’.join(str(p) for p in decode(coded)))
```

12 Zero Knowledge Proofs

Zero-knowledge proofs are those that prove a statement without yielding any additional information, for example:

1. In the example of colour blind friend and two balls, we have May and June out of which May is color blind. June has two balls and needs to prove that the balls are two different colors. The latter switches the balls randomly behind her and shows them to the former who must tell her if the balls have been switched. The possibility of June answering correctly is 50%. When this experiment is carried out over and over again, the possibility of June being able to answer correctly with false information is pretty low. So we verify June’s (**prover**) claim without the colors ever being revealed to May(**verifier**).

2. Imagine a cave with a single entrance but two pathways (path A and B) that connect at a common door locked by a passphrase. Alice wants to prove to Bob she knows the code to the door but without revealing the code to Bob. To do this, Bob stands outside of the cave and Alice walks inside the cave taking one of the two paths (without Bob knowing which path was taken). Bob then asks Alice to take one of the two paths back to the entrance of the cave (chosen at random). If Alice originally chose to take path A to the door, but then Bob asks her to take path B back, the only way to complete the puzzle is for Alice to have knowledge of the

code for the locked door. This process can be repeated multiple times to prove Alice has knowledge of the door's code and did not happen to choose the right path to take initially with a high degree of probability. After multiple repetitions of this, Bob can be sufficiently confident that Alice knows the code without ever knowing the code himself.

These examples demonstrate the properties of zero knowledge of the fact to the verifier, completeness of the proof and the soundness of the proof (that is if the prover isn't truthful they won't be able to convince the verifier). So what is the significance of ZKPs in cryptography? The use of ZKP protocols allows us to cryptographically prove to someone that they know about a piece of information without actually revealing the information thus maintaining privacy. It's use is on the rise in areas like public blockchain networks. The two types of zero knowledge proofs are discussed below.

12.1 Interactive Zero Knowledge Proofs

Interactive ZKP requires the verifier to constantly ask a series of questions about the "knowledge" the prover possess. Like in the first example, the May asked June about the switch status of the balls. A real life example of this was seen in Nuclear Disarmament drive in 2016, when Princeton University and one of it's departments demonstrated a technique which might allow inspectors to verify whether an object is nuclear or not without revealing sensitive information that might compromise the details of the internal workings or recording and sharing any information.

Some problems with this method of cryptography are that the transfer of the encrypted information becomes problematic because you essentially need to provide the proof to every single verifier which means the process has to repeated over and over too many times and also, both the prover and verifier need to be available at the same time for the process to be able to work.

12.2 Non Interactive Zero Knowledge Proofs

NIZKPs have emerged as the main concept in the world of cryptography and engineering mathematics. These proofs enable one party to prove the validity of a statement to another party without revealing any additional information beyond the validity of the statement itself.Fiat and Shamir invented Fiat-Shamir heuristic and changed interactive zero-knowledge proof to non-interactive zero-knowledge proof. Fiat–Shamir heuristic is technique for taking an interactive proof of knowledge and creating digital signature based on it. This way ' witness' or fact can be verified publicly without prover being online all the time.

NIZKs do not require interaction between the prover and the verifier, making them highly efficient for various applications in secure communications, blockchain, and privacy-preserving technologies. The way they work is that a trusted third party generates and distributes a common reference string (CRS) to both the prover and the verifier (**Setup Phase**). Then the prover uses the

CRS to create a proof that demonstrates the truth of the statement without revealing any additional information which is sent to the verifier (Proof generation; The generation process can be computationally intensive, but it eliminates the need for interactive communication). Finally in the verification stage, the verifier uses the CRS and the proof received from the prover to check the validity of the statement. If the proof is valid, the verifier is convinced that the statement is true, without learning anything else.

This method is transferable easily, it is efficient, maintains privacy and is easily scalable. It's applications include digital signatures by allowing the signer to prove the authenticity of their signature without exposing the signed message. This is crucial in scenarios where the content of the message must remain confidential.

13 References

Introduction to Modern Cryptography: Katz and Lindell