Summer of Science

CRYPTOGRAPHY

Midterm Reort

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 $f : A_1\{a, \dots, z\} \rightarrow A_2\{A \dots Z\}$.
- 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

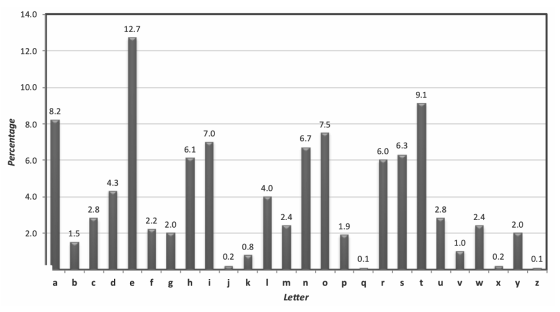

Figure 1: Average letter frequencies for English-language text

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 $c = c_1c_2\dots$ 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 $q_{[i+jmod26]}$.) 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. □

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 $Pr[K = k_i] = Pr[K = k_j] = 1/|K|$. □

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 1n (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 $|m0| = |m1|$.
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.

7 References

Introduction to Modern Cryptography: Katz and Lindell

8 Updated PoA

I severely overestimated how much content I can cover in a week, especially because the resource I am following (Katz-Lindell) is extremely elaborate and mathematically rigorous.

Hence, I am a behind where I was supposed ti be by now. I will be covering the following topics in the upcoming weeks: (I have also accounted for the fact that in the coming month, I am not present at home for 2 weeks)

- **Week 5:** Public-Key (Asymmetric) Cryptography
- **Week 6:** Hash Functions, Digital Signatures and Remaining topics
- **Week 7:** Lattice Based Cryptography
- **Week 8:** Zero Knowledge Proofs