Technical University Darmstadt

– Department of Computer Science –

Introduction to Cryptography

A Summary of the Course Contents

Author:

Jonas Weßner

Teaching professor	:	Prof. Dr. Marc Fischlin
Semester	:	Winter semester 2022/2023

CONTENTS

LIST OF ABBREVIATIONS

1 INTRODUCTION

Cryptography is the science of the processes for securing information, data and systems.

Typically we talk about certain key concepts, which differ from book to book. In this course, we define them as the following:

C Confidentiality: Attackers cannot read a message.

I Integrity: Attackers cannot modify the content of a message (in a narrow sense), cannot fake the message's sender address (authenticity) and receivers cannot claim, they have received the message (non-repudiation).

A Availability: A system is always functional (less relevant in this course)

Often in this lecture, we take a three-step approach to cryptographic processes:

1. **Abstract Object:** We define what interface our cryptographic process should work with. Often, we define one process for encryption, one for decryption and maybe others like one for key generation. We do not yet define their implementation.
2. **Security Model:** We define how our abstract object is used and in which cases our abstract object is secure. Questions to be answered are e.g. when is an attack successful and what is information the attacker is allowed to possess without introducing insecurity?
3. **Security Proof a Specific process:** When given a concrete implementation of our security model, we check the security of that model. For example, we prove that an attacker must calculate at least n hash sums to compromise the system.

Cryptographic processes can be classified in a matrix as follows:

	Private-Key	**Public-Key**
Confidentiality	Symmetric processes	Asymmetric processes
Integrity	Message Authentication Codes (MACs)	Digital signatures

Furthermore, there are elementary processes serving as building blocks to build cryptographic processes:

- Pseudo random number generators (PRNGs)
- Hash functions

- Blockcipher
- Number theory

2 PRIVATE-KEY CRYPTOGRAPHY

2.1 ONE TIME PAD

The One Time Pad works as follows:

1. Alice has a message $m \in \{0,1\}^n$ to be sent to Bob. Furthermore, Alice and Bob possess a common key $k \in \{0,1\}^n, n \in N$. n is called the security parameter.
2. Alice computes a cipher text $c \in \{0,1\}^n$ as $c = m \oplus k$ and sends it to Bob.
3. Bob reconstructs the plain message as $m = c \oplus k = m \oplus k \oplus k = m \oplus \{0\}^n$.

Therefore, the abstract object for this cryptographic process consists of the following functions[1]:

- $kGen(1^n) \to k \in \{0,1\}^n$
- $enc(m,k) \to c, \quad |m| = |k| = |c| = n$
- $dec(c,k) \to m \quad || \quad \perp$

The functional correctness is then defined as follows:
For all security parameter $n \in N$, for all messages m, for all keys $k \leftarrow kGen(1^n)$, for all keys $k \leftarrow kGen(m,k)$ and for all ciphertexts $c \leftarrow enc(m,k)$ applies $dec(c,k) = m$. That means that we must be able to decrypt all encrypted messages for all possible input parameters.

Security of Shannon's OTP

Independently of a bit m_i in the message m, the OTP flips this bit with the probability of $\frac{1}{2}$ creating a distribution of ciphtertexts c that is independent of the messages m. And because m and c are independent, it is intuitive that knowing c cannot reveal anything about m. We can also prove this mathematically because the probability for two independent events to take place is the product of the individual probabilities: $Pr[M = m|C = c] = \frac{Pr[M=m \wedge C=c]}{Pr[C=c]} = \frac{Pr[M=m] \cdot Pr[C=c]}{Pr[C=c]} = Pr[M = m]$.

1 The symbol $\perp$ is indicating an error.

In practice, perfect security is not feasible because this would mean that for transferring each message m a key k with at least the same length $|k| \geq |m|$ would have to be transferred (see definition of perfect security in section 3.2). For this reason, we define a weakened security definition, which comes in two variants – Concrete security and asymptotic security. In this course, we look at asymptotic security.

Concrete Security:	**Asymptotic Security**
A process is $(t, \mathcal{E})$-secure if no attacker A can break it with at most t steps with a probability of $\mathcal{E}$.	A process is secure, if no efficient (polynomial limited by security parameter n) algorithm breaks it with non-negligible (less than $\frac{1}{poly}$) probability.

The relevant terms are defined as follows, where n is the security parameter (e.g. key size):

- *Efficient Algorithm:* Algorithm with polynomial runtime with regard to n.
- *Non-negligible Probability:* An inversely polynomial probability ($\frac{1}{poly}$) with regard to n.
- *Negligible Probability:* Smaller than an inversely polynomial function (i.e. less than non-negligible) $\leq \frac{1}{poly}$. Mathematically speaking, a function $\mathcal{E} := N \rightarrow R$ if there can be found a value $limit \in N$, such that for all polynomial functions $poly$ applies $\mathcal{E}(n) \leq \frac{1}{poly} \quad | \quad n \geq limit$. This means that, if the security parameter is high enough, its value is smaller than any polynomial function. Stating that a function $\mathcal{E}$ is negligible can be written in three ways: $\mathcal{E} \leq neg(n)$, $\mathcal{E} = neg(n)$ or $\mathcal{E} \approx 0$. Furthermore, if a the function $\mathcal{E}_1$ and the function $\mathcal{E}_2$ differ by a negligible difference, we can write $\mathcal{E}_1 \approx \mathcal{E}_2$.

Examples for Negligibility of probability functions

- $\mathcal{E} = 2^{-n}$ is *negligible* because exponential functions grow faster than polynomial functions.
- $\mathcal{E} = n^{-5}$ is *not negligible* because there are polynomials that have smaller values as n approaches infinity e.g. n^{-6}.
- $\mathcal{E} = \begin{cases} 2^{-n} & \text{if } n \text{ is even} \\ n^{-5} & \text{if } n \text{ is uneven} \end{cases}$ is *not negligible* because for some values (all uneven values) it behaves like a polynomial function, such that e.g. n^{-6} would have greater values regardless of a chosen $limit$.

- $\mathcal{E} = \frac{1}{8}$ is *not negligible* because it does not approach zero and therefore there are many polynomial functions that are smaller than $\mathcal{E}$ for high n

Arithmetic Laws for Negligible Functions

If $mathcalE_1$ and $mathcalE_2$ are negligible functions $mathcalE_1, mathcalE_2 \approx 0$ and q is a polynomial function $q = poly$ and ω ia a non-negligible $\omega \not\approx 0$ function, then applies:

$$\mathcal{E}_1(n) + \mathcal{E}_2(n) \approx 0 \tag{2.1}$$
$$\mathcal{E}_1(n) \cdot q(n) \approx 0 \tag{2.2}$$
$$\omega(n) - \mathcal{E}_1(n) \not\approx 0 \tag{2.3}$$

More intuitively phrased:

(2.1) Executing a negligible function and another negligible function after each other will still result in something negligible.

(2.1) Executing a negligible function for any polynomial number of times will still result in a negligible value in sum because the negligible function approaches zero faster than any polynomial function could grow towards positive values.

(2.1) Subtracting something negligible from something non-negligible does not make a difference – the result is still non-negligible.

2.2.1 *Semantic Security – A Simulation-Based Definition*

A cryptographic process is semantically secure if for all PPT attacker algorithms A, which calculate information $f(m)$ about a length-invariant message m (message of given length n) with the help of the message's ciphertext c with a certain probability, there is a simulator algorithm S, which can compute the same information with a negligibly different probability:

$$Pr[A(1^n, c) = f(m)] \approx Pr[S(1^n) = f(m)]$$

More intuitively speaking, everything which can be computed with the knowledge about the ciphtertext of a message can also be computed with nearly the same precision without knowledge about the ciphertext.

This is the simulation-based definition of asymptotic security.

2.2.2 *Indistinguishability (IND) Security – a Game-Based Definition*

This is a game-based definition of asymptotic security, which is equivalent to the semantic security definition (2.2.1).

A cryptographic process $\mathcal{E}$ is indistinguishable (IND) if for all PPT attackers A applies:

$$Pr[Exp_{\mathcal{E},A}^{IND}(n) = 1] \approx \frac{1}{2}$$

, where $Exp_{\mathcal{E},A}^{IND}(n)$ is defined as the following algorithm:

$$KGen(1^n) \rightarrow k \tag{2.4}$$
$$\{0,1\} \rightarrow b \tag{2.5}$$
$$A(1^n) \rightarrow (st, m_0, m_1)| \quad |m_0| = |m_1| = n \tag{2.6}$$
$$Enc(m_b, k) \rightarrow c \tag{2.7}$$
$$A(1^n, st, c) \rightarrow a \tag{2.8}$$
$$return \quad a == b \tag{2.9}$$

More intuitive worded, this means that (2.4) a secret key of length n is chosen (2.5) a secret random bit is chosen (2.6) the attacker selects two messages of his choice (2.7) one through b randomly selected message is encrypted (2.8) the attacker looks at the ciphertext of one of its messages and decides which messages ciphertext that is and (2.9) the experiment returns true if the attacker has been able to identify the message and otherwise false. If the output of this experiment is negligible close to $\frac{1}{2}$ for all attackers A, then the attackers have no significant advantage over simply guessing and, therefore, the cryptographic process $\mathcal{E}$ is secure.

An example of an insecure process $\mathcal{E}$ could be an algorithm, which simply flips all bits: $Enc(m,k) = \sim (m)$. The attacker could then just use the Enc algorithm (which is not secret) with any key and get the same result because the Enc algorithm does not use the key. Therefore, he could distinguish messages with the probability of $1 \not\approx \frac{1}{2}$.

An example of a secure algorithm is Shannon's OTP. The reason is that without knowing the key, the attacker cannot distinguish two messages, because the OTP flips bits with the probability of $\frac{1}{2}$, such that the cipthertext is completely random, just like the key.

2.3 Pseudo Random (Number) Generators (PR(N)G)

An algorithm G is a secure PRG, if for all PPT attackers A applies:

$$Pr[Exp^{PRG}_{G,A}(n) = 1] \approx \frac{1}{2}$$

, where $Exp^{PRG}_{G,A}(n)$ is defined as:

$$KGen(1^n) \to k \tag{2.10}$$
$$\{0,1\} \to b \tag{2.11}$$
$$G(k) \to y_0 \tag{2.12}$$
$$\{0,1\}^{|y_0|} \to \tag{2.13}$$
$$A(1^n, y_b) \to a \tag{2.14}$$
$$return \quad a == b \tag{2.15}$$

More intuitively speaking, this means if we generate a pseudo-random bitstring y_0 using G and a fully random bitstring y_1, no attacker algorithm can distinguish them.

There is another definition of PRG, where the secret bit b is set from the beginning. In this case the experiment will not return the expression $a == b$ but instead directly return the attacker's guess a. The IND security of the PRG is then defined as follows: The behavior of the attacker (i.e. its return value) is not significantly influenced by whether we give it the output of the PRG or a random bitstring (i.e. it does not depend on the value b):

$$Pr[Exp^{PRG}_{G,A,0}(n) = 1] \approx Pr[Exp^{PRG}_{G,A,1}(n) = 1]$$

As a side note, following Kerckhoff's principle, a potential attacker could also access our PRG. He could, therefore, calculate the pseudo-random strings for all possible input values in $|K_n|$. This means, that a generated value y only has the security of 2^k instead of $2^{|y|}$. However, as $|K_n|$ grows exponentially, such an attacker would not be efficient and is, therefore, asymptotically irrelevant.

2.4 PRGs for key expansion

We have seen that the OTP is not practicable because for using it we must first exchange a key that is just as long as the message itself. Now that we have looked into PRGs from an abstract point of view, we could use PRGs to generate a long key from a small key and then use that for encryption:

$$KGen(1^n) \rightarrow k \qquad | \quad n \in \{0,1\}^n \tag{2.16}$$
$$Enc(k, m) \rightarrow m \oplus G(k) \qquad | \quad M_n = \{0,1\}^{l(n)} \tag{2.17}$$
$$Dec(k, c) \rightarrow c \oplus G(k) \qquad or \ \perp if \ |c| \neq l(n) \tag{2.18}$$

Given the assumption that the Generator G is IND-secure, we must now prove that the encryption method $\mathcal{E}$, which utilizes it, is also secure. A suitable method of proof for this is reduction.
The main idea of using reduction for proving the IND security of a cryptographic method $\mathcal{E}$ is the following:

Assume that $\mathcal{E}$ is not IND-secure and, therefore, we could find an attacker algorithm $A_{\mathcal{E}}$ which breaks $\mathcal{E}$. Then construct a new attacker $A_{\mathcal{E}'}$ utilizing $A_{\mathcal{E}}$, which breaks another cryptographic method $\mathcal{E}'$, which is known to be secure. The existence of the newly constructed attacker would then contradict the knowledge about $\mathcal{E}'$. Hence, $\mathcal{E}$ must be IND-secure.

To prove the IND security of the newly constructed cryptographic method $\mathcal{E}$ we reduce it to the IND-security of the utilized PRG as shown in figure 2.1. We have a PRG challenger that challenges us to decide if its output is a random number or the output of the PRG. Then we take the challenger's output y_b and use it to compute a ciphertext c of a randomly chosen message. Assuming the PRG challenger gives us y_0, what we did until now is exactly what a challenger of an IND experiment for $\mathcal{E}$ would do. Because we assume that $\mathcal{E}$ is not IND-secure, the attacker $A_{\mathcal{E}}$ will then be able to efficiently decide what message has been encrypted by us (choose x == a). But in case the PRG challenger gives us y_1, the process of creating c is equivalent to an OTP, which we know is IND-secure, and can therefore not be decided by $A_{\mathcal{E}}$. That means, if there was an efficient attacker for the IND experiment for $\mathcal{E}$, this reduction would build an efficient IND attacker for the PRG. Hence, given the fact the PRG is IND secure, this would contradict. It follows that $\mathcal{E}$ is IND secure if the used PRG is IND secure and otherwise not.

2.5 Indistinguishability under chosen plaintext attack (IND-CPA)

CPA is an attack model in which the attacker can retrieve ciphertexts for arbitrarily chosen plain texts as many times as he wants. But because the attacker must be efficient, he can just retrieve a polynomial amount of ciphertexts. The game-based definition of IND-CPA security is identical to the definition of IND security but allows requesting c_b multiple times for different message pairs (m_0, m_1). But the encryption oracle will have to choose b once in the beginning and then keep it the same for each request.

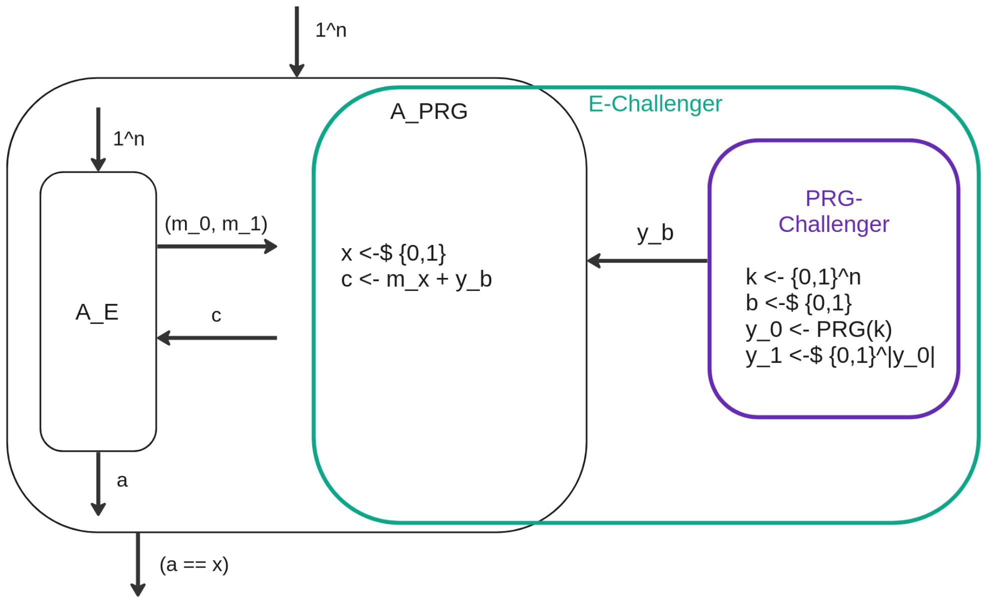

Figure 2.1: Reduction for proving the IND-security of $\mathcal{E}$ if the IND-Security of PRG is given.

Every IND-CPA-secure method is also IND-secure. But IND-secure methods are not automatically IND-CPA secure. For example, the OTP with the same key for each message is IND secure but not IND-CPA secure: An attacker could request the ciphertext for two message pairs $(m_0, m_1) \rightarrow c_1$ and $(m_0, m_2) \rightarrow c_2$. The attacker knows that if $b = 0$, the oracle has encrypted m_0 two times and will therefore return 0 if $c1 == c2$. If $b = 2$, the oracle must have encrypted two different messages, which must because of correctness be mapped to different ciphertexts. Therefore, the attacker returns 1 if $c_1 \neq c_2$.

An approach for making an encryption method using a secure PRG IND-CPA secure could be the following:
For each message to encrypt, call the PRG first to generate a key that is twice as long as the fixed-size message length. Then use the first half of the PRG output for encryption and save the second part. The first seed for the PRG will be taken from a *KGen* algorithm. The following keys will be the second half of the PRG in the previous iteration. Because the PRG output is pseudorandomized, it is also a secure seed.
This approach is nice, but it is not practicable, because the sender and the receiver must be synchronized, which might fail because of insecure connections with message retransmissions.

2.6 PSEUDO RANDOM FUNCTIONS (PRF)

A PRF is a function $F(k, x) \rightarrow y$, which takes a (secret) key and a seed an x-value as input and produces a pseudo-random output with the following characteristics:

- y_0 and y_1, generated as $F(k, x_0) \rightarrow y_0$ and $F(k, x_0) \rightarrow y_0$, are different, if $x_0 \neq x_1$ and are equal if $x_0 = x_1$.
- A CPA attacker cannot distinguish the output of a PRF from a truly random function, which would do the same thing but in a truly randomized fashion.

Defining a (secure) PRF using a game would involve creating a challenger which randomly chooses a hidden bit b and, depending on that bit, calls the PRF or a truly random function. For both, the truly random function and the PRF, the outputs must be deterministic (consistent), such that if the same x value is provided as input, the same y value will be emitted.

2.7 DATA ENCRYPTION STANDARD (DES)

DES is a block cipher, which was used until about the year 2000. Because of its short keys, it was later replaced with DES3 and eventually by AES.

DES defines an encryption method for encrypting 64-bit plain texts using 56-bit keys (the key is 64 bits long, but 8 bits are used for parity). It uses a so-called Feistel function in 16 rounds, where in each round a part of the key is used.

The Feistel function uses another function F to operate on half of the 64-bit block at a time. Its most important characteristic is, that it is reversible, even if the internally used function F is not reversible (e.g. a one-way hash function, which also has collisions).
In each iteration of the Feistel function, the plain text will be split in half ($L_i || R_i$) and transformed to a new intermediate ciphertext $L_{i+1} || R_{i+1}$ as follows:

$$
\begin{aligned}
L_{i+1} &:= R_i \\
R_{i+1} &:= F(k_i, R_i) \oplus L_i
\end{aligned}
$$

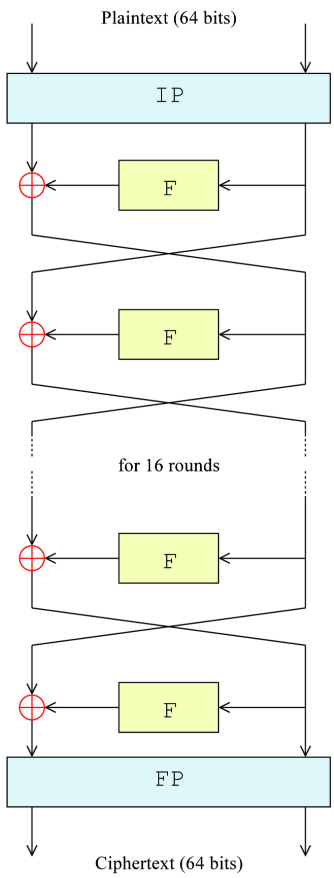

Figure 2.2: Feistel Function

The exception is the last iteration, in which the sides will not be switched, such that $Li + 1 = F(k_i, R_i) + L_i$ and $R_{i+1} = R_i$.

Regardless of the function F, the Feistel function can be reversed by applying it again with reversed key order. That is possible because for each iteration we can compute L_i from $R_{i+1} + L_{i+1}$ and can take R_i directly from L_{i+1}.

The Feistel function provides the reversibility of the function, however, it does not provide the security itself. The security is provided by the internally utilized function F. In DES this function is using so-called S- and P-boxes, whose details go beyond the scope of this course. What is important, however, is that F implements Shannon's Concept of *confusion and diffusion* and that it takes a part of the key k_i and a bitstring as an input. The function F is, in fact, a non-reversible function.

56 bits (2^{56} possibilities) were soon not sufficient for security against brute-force attackers on modern hardware. What could have been done to increase its security would be creating a new encryption algorithm, which works like DES but uses a longer key. But to reduce the effort required to increase DES's security, another approach was used – applying DES multiple times with different keys.

The first approach would be to compute DES two times with two different keys to achieve the security of 2^{112}, because brute force a ciphertext, the attacker would have to compute all possible combinations of keys of the first DES and keys of the second DES $2^{56} \cdot 2^{56}$. But if an attacker would know the plain text m and the ciphertext c of a message, he could use a *meet-in-the-middle* attack to drastically reduce the computational effort for brute forcing. In the following, the computation of double DES is shown:

$$m \rightarrow DES_1(key_1, m) \rightarrow c' \rightarrow DES_2(key_2, c') \rightarrow c$$

An attacker could then first compute all possible intermediate ciphertexts c' for all possible values of k_1 from m and save the results in a hashmap. Then he could compute all possible intermediate ciphertexts c' with all possible values of k_2 from c using DES_2^{-1}. The computation required is equal to $2^{56} + 2^{56} = 2 \cdot 2^{56} = 2^{57}$ instead of 2^{112}. To obtain the correct combination of k_1 and k_2, the attacker would then search for collisions i.e. for pairs (k_1, k_2) that produced the same intermediate ciphertext c'.

For this reason, double DES is not useful. Instead, Triple DES is used, which has the security of 2^{112}, because the attacker can use a meet-in-the-middle attack but has to calculate two DES on one of the sides. Triple DES uses the inverse operation (reverse order key parts of k_2) of DES_2 for the second

DES. The reason is that there was uncertainty about whether DES could be a *group*, such that there might be one DES_x operation, which would be equal to the combination of DES one to three. Later, however, it was discovered, that DES is not a group and inverting the middle DES was, therefore, not required:

$$m \rightarrow DES_1(key_1, m) \rightarrow c' \rightarrow DES_2^{-1}(key_2, c') \rightarrow c'' \rightarrow DES_3(key_3, c'') \rightarrow c$$

3 GLOSSARY

In this chapter, basic terms will be defined and explained.

3.1 KERCKHOFFS'S PRINCIPLE

Security should not require the system, but merely the key to be secret.

3.2 PERFECT SECURITY

A cryptographic process is perfectly secure if for all messages in the set of possibles messages $m \in M$ and all possibly producible ciphertexts $c \in C \mid Pr[C = c] > 0$ the probability that m occurs is equal no matter if the ciphertext is known or unknown: $Pr[M = m] = Pr[M = m|C = c]$. More simply put, this means that an attacker cannot gain any knowledge about the message when seeing its ciphertext regardless of the message's and the ciphertext's concrete values.

3.3 CONDITIONAL PROBABILITY

Conditional probability $Pr[A = a|B = b]$ is the probability that the event $a \in A$ occurs if it is already known that the event $b \in B$ occurs. It is calculated as $Pr[A = a|B = b] = \frac{Pr[A=a] \wedge Pr[B=b]]}{Pr[B=b]}$. An example is a probability rolling a 6-sided dice results in the number 6 if it is already known that the result is an even number: $Pr[is6|iseven] = \frac{Pr[is6] \wedge Pr[iseven]}{iseven} = \frac{\frac{1}{6}}{\frac{1}{2}} = \frac{1}{3}$.

Perfect security can only be reached for processes with deterministic a decryption function if the key space is at least as big as the message space $|K| \geq |M|$. This is because then a given ciphtertext c could be decrypted to $|K|$ different messages m (determinism), which minimizes the search space for an attacker from $|M|$ possibilities to $|K|$ possibilities and therefore also decrease the attacker's uncertainty..

3.4 Hiding Message Length

Completely hiding the length of a message is impossible if the length of possible messages is unlimited. Hiding messages' lengths would require changing their length for transmission. The message length cannot be decreased because this would result in data loss. Therefore, messages would have to be extended up to at least the size of the longest possible message to make all messages appear to have the same length. If the message size is unlimited, we cannot expand messages to that nonexisting limit.

3.5 Insecurity of Shift-Ciphers

Shift-ciphers, like Ceasar, are not perfectly secure, which can be shown by contraposition: If the possible messages are defined as $M = \{aa, ab\}$, a shift cipher shifting each character by some number of characters in an alphabet would always create ciphertexts as follows $Dec(aa) \rightarrow c_0c0$ and $Dec(ab) \rightarrow c_0c_1 \quad | \quad c_0 \neq c_1$ Therefore, an attacker's probability when seeing the ciphertexts to guess the correct message $m \in M$ is equal to 1 and greater than the initial probability, which was $\frac{1}{2}$. A concrete example would be: $Pr[M = aa | C = c_0c_0] = 1 \neq Pr[M = aa] = \frac{1}{2}$. We see that limiting the set of possible messages can be a helpful method for contradictions of perfect correctness.

3.6 Criteria for Correct Reduction Proofs

If using reduction for proving the security of a cryptographic process, the following points must be addressed:

1. Show the reduction algorithm as text, pseudo-code or picture
2. Show that the reduction algorithm is efficient
3. Show that the reduction algorithm wins the outer game with non-negligible advantage.