Applied Cryptography

Ivan Valentini
Telegram: @IvanV1337
Github: https://github.com/IvanValentini/XXX

October 9, 2022

Contents

Chapter 1

Introduction to Cryptography

1.1 Introduction

Cryptography refers to hidden writing. Its goal is to enable a secure communication between two users (Alice and Bob), and making it impossible for an eavesdropper to understand the information being exchanged.

By channel we mean any physical or logical medium of communication from one user to another. A channel becomes secure when the information exchanged over it cannot be overheard or tampered with by eavesdroppers. By default a channel is considered insecure, so the intent of cryptography is to make secure, an insecure channel.

To transmit data in a secure way Alice and Bob rather than transmitting the message in a plain form they first covert it to a disguised form. Formally these are called plaintext ($\mathcal{P}$) and ciphertext ($\mathcal{C}$). The idea is to transform a plaintext into a ciphertext, so that Alice sends the latter to Bob, and Bob is able to reconstruct the plaintext from the received ciphertext while this is very difficult (almost impossible) for Eve.

In practice there is a pair of functions:

- $enc : \mathcal{P} \to \mathcal{C}$
- $dec : \mathcal{C} \to \mathcal{P}$

Such that $dec(enc(m)) = m$ for every $m \in \mathcal{P}$. For this to work Alice and Bob need to agree on what encryption and decryption scheme to use without disclosing it to Eve. Eve will only be able to observe ciphertexts, but notice that if the sets of plaintexts and ciphertexts are too small, then Eve

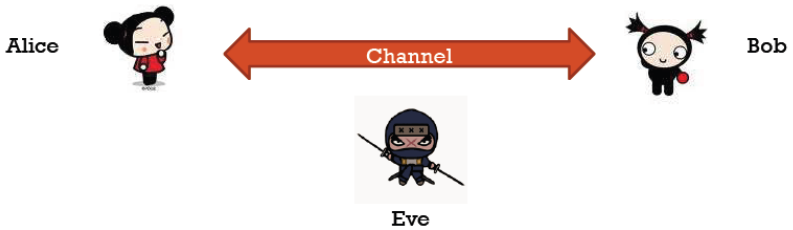

Figure 1.1

can try all the plaintext-ciphertext pairs (exhaustive search) or even if the sets of plaintexts and ciphertexts are large enough to make exhaustive search impractical, encryption and decryption can be defined in obvious way to allow Eve to easily reconstruct them (guessing). There is a problem with this formalization: these two functions, must be kept secret. These function can be used to create a secure channel, but how do you share these function? We need a secure channel, but if we had a secure channel, why not use it in the first place?

Since otherwise we would have to define and encryption and decryption functions for each pair of people that want to communicate securely we introduce the concept of a cryptographic key. We denote the set of (cryptographic) keys with $\mathcal{K}$, and consider a map $\varphi : \mathcal{P} \times \mathcal{K} \to \mathcal{C}$ such that for every key $k \in \mathcal{K}$ the function $\varphi(\cdot, k) : \mathcal{P} \to \mathcal{C}$ is an encryption function.

There are some key differences with respect to the old functions:

- The definition of φ (encryption algorithm) can be very complex but it can be public(i.e. known to anyone) and Alice and Bob can agree on it over an insecure channel. Before the encryption algorithm had to be private.
- The only component that must be kept secret(and thus exchanged over a secure channel) is the cryptographic key k, that defines the encryption function to use

The Kerchkhoff principle states that a cryptosystem should be secure even if everything about the system, except the key, is public knowledge. The advantages to only exchanging cryptographic keys rather than ciphers are that it is easier to keep secret k than φ, and if the key is discovered it is sufficient to choose another key. The main disadvantage is that the attacker only needs to find the key to break the system.

1.2 Attacks

Let's now consider some useful attack models expressed in terms of what the attacker can observe and what queries they can make to the cipher. A query for our purposes is the operation that sends an input value to some function and gets the output in return, without exposing the details of that function. An encryption query, for example, takes a plaintext and returns a corresponding ciphertext, without revealing the secret key. We call these models black-box models, because the attacker only sees what goes in and out of the cipher. There are several different black-box attack models. Note that the higher the attacker skills and complexity of the attack the less (computational) effort the attacker needs to put to break the system.

1.2.1 Ciphertext-only attackers (COA)

Ciphertext-only attackers (COA), or known-ciphertext attackers, observe ciphertexts but don't know the associated plaintexts, and don't know how the plaintexts were selected. Attackers in the COA model are passive and can't perform encryption or decryption queries.The task of the attacker is very difficult and a lot of computational power is required to mount such an attack, this is because the attacker needs to check for every possibile key of if the decrypted ciphertext is meaningful.

In some cases the attacker only needs to know the probability distribution of the plaintexts, it could obtain a lot of information merely by observing some ciphertexts. This holds under the assumption that all plaintexts are encrypted with the same cipher and the same key. The method may be difficult (if possible at all) to apply to short messages or messages that contain words with many occurrences of letters with low frequencies. More information can be found here.

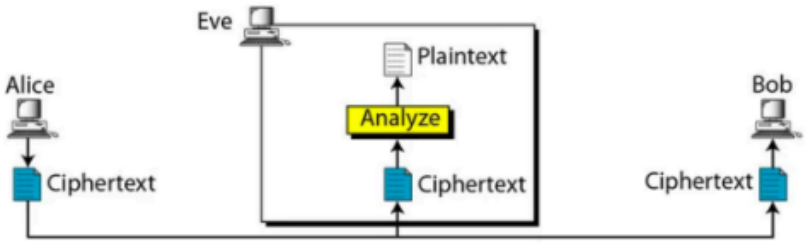

Figure 1.2

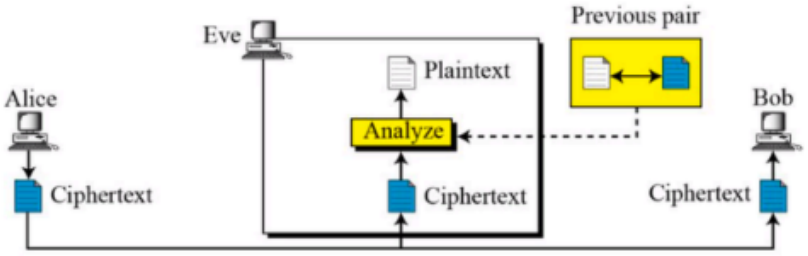

Figure 1.3

1.2.2 Known-plaintext attackers (KPA)

Known-plaintext attackers (KPA) observe ciphertexts and do know the associated plaintexts. Attackers in the KPA model thus get a list of plaintext–ciphertext pairs, where plaintexts are assumed to be randomly selected. Again, KPA is a passive attacker model, thus the attacker can not choose a plaintext to encrypt.

Essentialy the attacker will attempt to do what is known as a key recovery attack. But recovering the key might be a difficult problem to solve, so Eve might try to discover a functionally equivalent algorithm for encryption and decryption, or else design cryptographic algorithm that, even without knowing the key k, produces the same result as those of the cipher with the key k§This kind of attacks are known as Global Deduction/Reconstruction attacks.

1.2.3 Chosen-plaintext attackers (CPA)

Chosen-plaintext attackers (CPA) can perform encryption queries for plaintexts of their choice and observe the resulting ciphertexts. This model captures situations where attackers can choose all or part of the plaintexts that are encrypted and then get to see the ciphertexts. Unlike COA or KPA, which are passive models, CPA are active attackers, because they influence the encryption processes rather than passively eavesdropping.

With this the attacker may try to guess previously unknown plaintext-ciphertext pairs

1.2.4 Chosen-ciphertext attackers (CCA)

Chosen-ciphertext attackers (CCA) can both encrypt and decrypt; that is, they get to perform encryption queries and decryption queries. The CCA model may sound ludicrous at first—if you can decrypt, what else do you need?—but like the CPA model, it aims to represent situations where

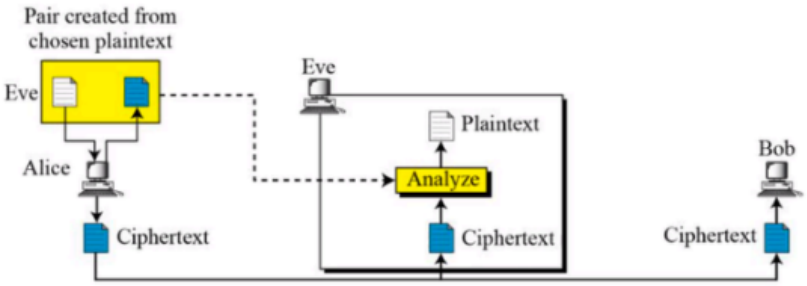

Figure 1.4

attackers can have some influence on the ciphertext and later get access to the plaintext. Moreover, decrypting something is not always enough to break a system.

1.3 Shannon Theorem

A cipher is broken if a method of determining the plaintext from the ciphertext is found without being legitimately given the decryption key. Any cryptosystem can be broken by an exhaustive key search. There exist ciphers that cannot be broken that are called perfect, and even exhaustive key search is of limited use for these ciphers. A cipher is perfect if, after seeing the ciphertext, an attacker gets no extra information about the plaintext other than what was known before the ciphertext was observed. The attacker might know the kind of content hidden but not the content itself, the point is that the knowledge of the attacker about the plaintext is not increased after intercepting the ciphertext .

Clever boy assumption states that plaintext and the keys are indipendent. So we do not select a particular key for a particular plaintext, for any plaintext we can choose any key.

$$\forall m \in \mathcal{P} and \forall c \in \mathcal{C} \Longrightarrow P(m) = P(m|c)$$

So the probability of having the plaintext m is equal to the conditional probability of observing m over the channel once you have seen the ciphertext c. So seeing the ciphertext c does not add any knowledge about the plaintext message sent.

Let us fix an integer n. Assume that: keys, plaintext, ciphertext have all n bits. Assume also that the plaintexts and the keys are independent (Clever boy assumption) and any n-bit string may be either a key or a plaintext. Then φ is a perfect cipher if and only if both the following conditions hold:

1. the keys are perfectly random
2. for any pair (m,c) of plaintext-ciphertexts, there is one and only one key k such that $c = \varphi(m,k)$

1.3.1 Consequences of Shannon Theorem

If φ is a perfect cipher, then all ciphertexts have the same probability to be received. Even if Eve knows the probability distribution of the plaintexts, she observes is a perfectly random cipher. Hence Eve cannot recover any information on the sent message from the intercepted ciphertext. But we

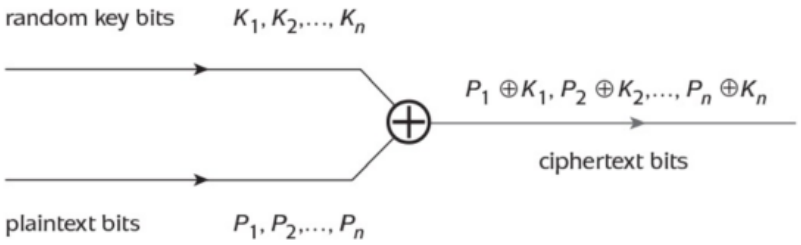

Figure 1.5

made a very big assumpion: this is true only if Eve can intercept **one** ciphertext. If Eve intercepts more ciphertexts encrypted with the same key, then the Shannon theorem no longer guarantees perfect secrecy. So in practical sense, a perfect cipher is not unbreakable, so a perfect cipher is not necessarily ideal, because every time we need to use a different key also the key has to be as large as a message transfered.

1.4 Vernam Cipher - One Time Pad

The One Time Pad is an example of a perfect cipher. We work under the same assumptions of the Shannon Theorem. In the Vernam cipher we define encryption by means of the bitwise XOR operation:

$$c = \varphi_k(m) = k + m$$

Only if the key used is randomly chosen, this cipher is perfect. But there are a few problems: choosing the key randomly, how is the key shared, the length of the key used that has to beh as long as the message shared and having to use a key just once. Let's say that Eve known a single plaintext-ciphertext pair (m_1,c_1), the key can be easily recovered as $m_1 + c_1 = m_1 + (m_1 + k) = (m_1 + m_1) + c_1 = 0 + k = k$.

1.5 From perfect to ideal

Requirement: ability to encrypt a long message (e.g., a file of several megabytes) using a short key (e.g., a few hundred bits). Do not consider all possible adversaries, but only computationally feasible adversaries, that is, "real world" adversaries that must perform their calculations on real computers using a reasonable amount of time and memory. This leads to a weaker definition of security called semantic security. Since the focus is on the "practical", instead of the "mathematically possible", one shall also insist that the encryption and decryption functions are themselves efficient algorithms and not just arbitrary functions. So we'll study ciphers are regarded as secure in practice because the known theoretical attacks take too much time to conduct. In other words, to implement such theoretical attacks requires resources which are unrealistic for any attacker.

1.5.1 Practical secure ciphers

Characterizing the notion of practical security is not an easy task as it must consider several different aspects including:

- Cover time: the time window in which a plaintext must be kept secret. This suggests that no attack on the cipher can be conducted in less than the cover time and implies that an exhaustive key search takes longer than the cover time. Evaluation should be repeated if a new attack is discovered or other parameters are changed such as the available computation power.
- Computational complexity: what computational processes are involved in known attacks on the cryptosystem how much time it takes to conduct these processes. measuring the time taken to perform the processes requires a way of measuring the time it takes to run a process, or a function and this is expressed as a function of the size of the input that expresses the number of elementary operations performed, known as the Big O notation.

Establishing the complexity of any known attacks is important and useful, but brings no guarantees of practical security and this is because there are undiscovered theoretical attacks, problems with the implementation, key management issues and so on.

Chapter 2

Introduction to Stream and Block Ciphers

2.1 The problem with Vernam cipher

There are few issues: generating a truly random key as long as the message, find a secure channel for transportation of the key to the message recipient and do this for every single message to be exchanged. In summary, the problem becomes to securely transfer large quantities of secure keys. There are two approaches that are seen as an improvement to Vernam cipher: stream ciphers and block ciphers.

2.2 Symmetric encryption

In stream ciphers we take a seed (a small vector of a few random bits) that must be kept secret, then build a keystream (a very long sequence of pseudorandom bits), finally xor the keystream with the plaintext bitwise to calculate the ciphertext. The problems are how are we going to generate a perfectly random keys of arbitrary size and how can we replicate the random streams of bytes for decryption. In block ciphers we use the same key multiple times in a way that does not compromise the cipher. The problems are how can we reuse multiple times the same key without enabling an attacker to perform cipher-text only attacks and how can we avoid attackers to exploit the block structure.

These two ciphers are known as symmetric encryption algorithms, where stream ciphers perform operations in a way such that the plaintext is processed one bit at a time, and the algorithm selects one bit of plaintext, performs a series of operations on it, and then outputs one bit of ciphertext, block ciphers perform operations in a way such that the plaintext is processed in blocks (groups) of bits at a time, and the algorithm selects a block of plaintext bits (typically 64 bits), performs a series of operations on them, and then outputs a block of ciphertext bits. Notice that a stream cipher can be seen as a block cipher with blocksize set to 1 bit, but there are also stream ciphers that process data in bytes, and hence could be regarded as block ciphers with a block size of 8, as a rule of thumb if the blocksize is less than 64 bits we talk about stream ciphers otherwise we talk about block ciphers.

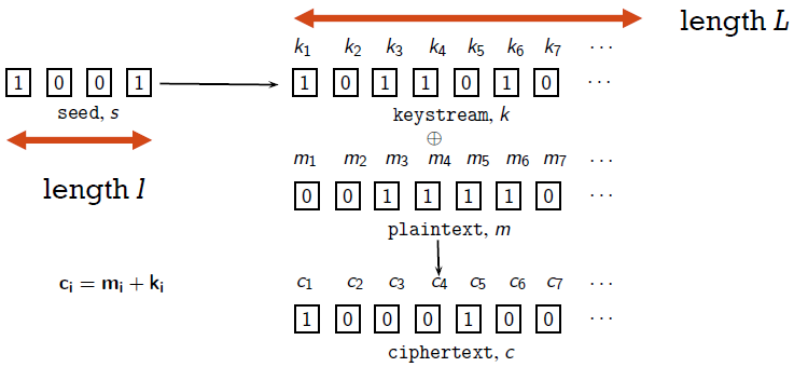

Figure 2.1

2.2.1 Stream ciphers

The real work in designing a good stream cipher goes into designing the keystream generator. Keystream generators produce output which appears to be randomly generated, but is actually not randomly generated, these are referred to as pseudorandom generators. In many cases, stream ciphers combine the keystream with the plaintext in more complex ways than a simple bitwise xor operation.

For a stream cipher to be good, Eve should not be able to: recover the seed by making the set of possible seeds so large that and exhaustive search is very hard in practice and also predict the rest of the keystream by eliminating any patterns from the keystream. More precisely: choose a short l-bit (much smaller than the lenght L of the plaintext to be encrypted) seed s as the encryption key and stretch the seed into a longer L-bit string (the key) that is used to mask the message and decrypt the ciphertext. The seed s is stretched using some efficient, deterministic algorithm G that maps l-bit strings (seeds) to L-bit strings(keys). Formally:

- Encryption: $G(s) + m$ for any seed s (of size l) and plaintext m (of size L)
- Encryption: $G(s) + c$ for any seed s (of size l) and ciphertext c (of size L) where G is called a pseudo-random generator

Notice that if $l < L$, then by Shannon's Theorem, stream ciphers cannot be perfect, however, if G has certain properties, then stream ciphers are secure in practice. Suppose s is a random l-bit string and r is a random L-bit string, if Eve cannot effectively tell the difference between $G(s)$ and r, then it should not be able to tell the difference between stream ciphers and one-time pad. Since the one-time pad latter cipher is secure, so should be the stream cipher.

A stream-cipher is well equipped to encrypt a single message from Alice to Bob. If two messages are encrypted with the same key there may be problems. As an example consider the case in which Alice and Bob want to exchange messages m_1 and m_2, let $c_1 = m_1 + G(s)$ and $c_2 = m_2 + G(s)$. If Eve is able to intercept both ciphertexts, then it is able to calculate $c_1 + c_2 = (m_1 + G(s)) + (m_2 + G(s)) = (m_1 + m_2) + (G(s) + G(s)) = m_1 + m_2$, and as english text contains enough redundancy that given $m_1 + m_2$, Eve can recover both m1 and m2 in the clear by using frequency analysis.

Stream-ciphers are said to be malleable since an attacker can cause predictable changes to the plaintext, this is because and attacker can intercept ciphertext c and forwarding $c^{'=c+d}$, effectively the receiver will get $m' = c' + G(s) = (c + d) + G(s) = (c + G(s)) + d = m + d$. In other words

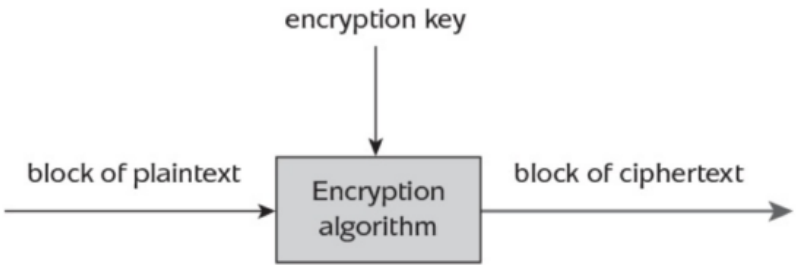

Figure 2.2

knowing neither m nor s, Eve was able to cause the decrypted message to become $m + d$ for d of its choice.

In summary stream-ciphers do not give rise to error propagation as each bit in the ciphertext depends on just one bit in the plaintext and 1 bit transmission errors will result in 1 bit error in the plaintext, also they are very fast making them ideal for real time applications (e.g., mobile communication services) and easy to implement in hardware and don't require large memory capabilities. Since stream ciphers process data bitwise, it is crucial that sender and receiver keep their keystreams in perfect synchronization and 1 bit data loss may have catastrophic consequences as decryptions are performed on the wrong bits after the receiver is out of sync of the sender and re-synchronization mechanisms must be put in place to avoid these problems.

2.2.2 Vigenère cipher

It can be seen as a variant of Vernam cipher whereby the key is a sequence of bits of fixed length. The key, and plaintext are a string of bytes, to encrypt: XOR each character in the plaintext with the next character of the key and wrap around in the key as needed.

Vigenere cipher can be attacked by first determining the key length and determining each byte of the key by using frequency analysis.

2.2.3 Block ciphers

Replace a block of N bits from the plaintext with a block of N bits from the ciphertext.

The relationship between the input blocks and the output block is completely random. It must be invertible for decryption to work. Thus, it has to be one-to-one, i.e. each input block is mapped to a unique output block. Usually, N=64, 128, 256. If the block size is too small, then the number of different plaintext blocks that can ever be encrypted may be too small for an attacker to launch a type of dictionary attack by building up a dictionary of plaintext/ciphertext pairs, if the block size is too large, then the block cipher becomes inefficient to operate, particularly for plaintexts smaller than the block size as they need padding. The encryption key for the ideal block cipher is the table (also called the codebook) that shows the relationship between the input and the output blocks. Think of each possible input block as one of 2^N integers and for each such integer we can specify an output N-bit block. If N is 64 then the codebook will be of size $N * (2^N)$ which is around $10^(21)$, but this is not practical since we can can not share such keys.

To make this practical a block cipher is a keyed family of pseudorandom permutations For each key, we have a single permutation that is independent of all the others. Think of each key as

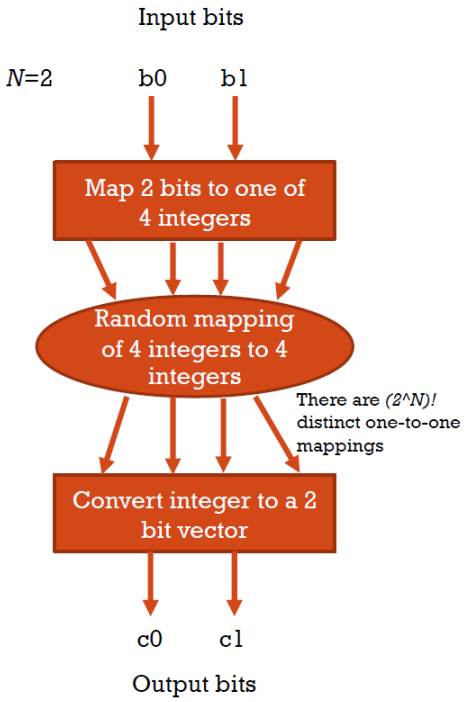

Figure 2.3

corresponding to a different codebook and our strategy is to choose 2^K permutations uniformly at random from the set of all $(2^N)!$ permutations

Block ciphers are slower than stream ciphers but are generally considered more secure than the latter. Block ciphers have a property called diffusion: 2 blocks differing in a single bit shall generate 2 very different ciphertexts. So even a small transmission error will give rise to errors in around half of the bits in the plaintext.

2.3 Examples of symmetric ciphers

2.3.1 Examples of stream ciphers

- RC4: Simple and fast stream cipher with a relatively low level of security, probably the most widely implemented stream cipher in software and widely used in SSL/TLS, WEP, and Microsoft Office
- A5/1: One of the stream cipher algorithms used in GSM to secure the communication channel over the air from a mobile phone to the nearest base station
- E0: The stream cipher algorithm used to encrypt Bluetooth communications

Type	Cipher	Applications
Stream	A5/1, A5/2, A5/3	Phone (GSM)
	RC4	Internet
	E0	Bluetooth
Block	DES	(old)
	3DES	Smart cards
	AES	Everywhere
	PRESENT	sensor networks

Figure 2.4

2.3.2 Examples of block ciphers

- DES: The default block cipher of the 1980s and 1990s, but now considered broken due primarily to its small key size. The two variants based on repeated DES applications commonly known as 3DES are still respected block ciphers, although there are now faster block ciphers available.
- AES: A default block cipher based on the encryption algorithm Rijndael which won the AES design competition by NIST to identity the successor of DES
- Serpent: A respected block cipher with a block size of 128 bits and key lengths of 128, 192, or 256, which was a finalist with AES in the NIST competition. Considered slower but somehow more secure than AES but not as widely adopted

2.4 Implementation of stream ciphers

First we want to tackle the problem of generating a randomly a key (known as keystream)that is as long as possible. Keystream generators should be fast (as in computable in polynomial time as function of number l of bits in the seed) and be secure, so intuitively, a string of L bits produced by a keystream generator should look random. I.e. it should be impossible in a polynomial amount of time in l to distinguish between a truly random bit string of length L and a string of the same length returned by the keystream generator. The main components are:

- States: vector of bits organized in registers
- Update function: function mapping a state to the next state (clock function)
- Output function: function extracting a bit from a state. Concatenating all bits returned by this function, it is possible to obtain the keystream
- Key loading: function that takes the seed (secret) and a (public) Initialization Vector (IV) to compute the initial state for the update function. Each IV should be used only once.

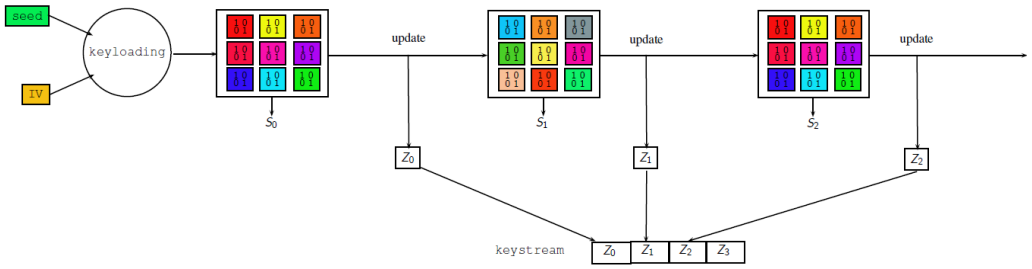

Figure 2.5

2.5 Warm Up

The first output bits strongly depend on the initial state. To avoid potential problems, it is customary to run a warm-up phase before starting encryption. This preliminary phase consists in applying the update function several times without outputting any bits of keystream and it is a highly recommended security best practice. Given any initial state, the states are periodic, since they are in a finite number and at some point we will obtain again one of the previous states. The keystream is also periodic... this is impossible to avoid. The smallest number i such that $update(...(update(S))) = S$ is called the period of the keystream (it depends on the initial state), and as a requirement is that the period of the keystream shall be quite large, regardless of the initial state, and can be achieved by a suitable design of the update function. As an example let's take an un update function as follows $f : (x, y, z) \Rightarrow (y + z, x, y)$. One can see that the repeated application of f to the inital state $(1, 0, 1)$ will yield $(1, 0, 1) \Rightarrow (1, 1, 0) \Rightarrow (1, 1, 1) \Rightarrow (0, 1, 1) \Rightarrow (0, 0, 1) \Rightarrow (1, 0, 0) \Rightarrow (0, 1, 0) \Rightarrow (1, 0, 1)$ with a period of 7.