University of Illinois, Urbana Champaign
CS/ECE 498AC3/4 Applied Cryptography

Instructor: Dakshita Khurana
Scribe: James Wang & Alex Morris
Date: September 1, 2020

LECTURE
3

Stream Cipher Examples, Block Cipher Introduction

Last lecture, we talked about another way to encrypt messages called stream ciphers. These ciphers allow us to use keys that are much smaller than the messages they encrypt. While this is not as secure as perfect secrecy, it is much more practical. The stream ciphers use a small initial "seed" of randomness and expand it into a larger string of pseudorandomness using a pseudorandom generator (PRG). This PRG generates a key that is as long as the message. We can then XOR the message and this key to create the ciphertext, much like the encryption scheme for a One Time Pad (OTP).
However, also like the OTP, stream ciphers have drawbacks as well. Stream ciphers reliance on the linear operation of XOR in order to encrypt the messages creates vulnerabilities. We will go over how these vulnerabilities can be exploited to retrieve the original message if you know enough information about the beginning of the message.
These vulnerabilities are what motivate us to look at ciphers which introduce nonlinear components. One type of which is known as **block ciphers**. Like stream ciphers, block ciphers allow us to encrypt messages with keys smaller than the message, but remove the vulnerabilities of the linear XOR.
To start off, we will first have to look at the framework that allows us to create block ciphers, **pseudorandom functions** (PRF's) and **pseudorandom permutations** (PRP's). We will then look at a new definition of security involving PRF's. Lastly, we will see how these components are used to create a block cipher in the Data Encryption Standard (DES) using a Feistel Network.
Therefore, in this lecture, we will first highlight the flaws of stream based ciphers, underscoring the need for a different encryption scheme. We will then introduce the framework of this new type of cipher know as block ciphers. Lastly, we will explore an example of how these components are used to create a block cipher.

3.1 Stream Ciphers and General Weaknesses

Before we go over different examples and vulnerabilities of stream ciphers, let's first review what symmetric ciphers are. Symmetric ciphers consist of efficient two algorithms, an encryption and decryption algorithm. The encryption algorithm encrypts a message using

a key, outputting a ciphertext. The decryption algorithm decrypts a ciphertext, also using a key. This decryption algorithm outputs a message. We want the decryption algorithm to "undo" the encryption algorithm, as long as they use the same key. Otherwise, this scheme would not be useful, since the receiver of the hidden message could not retrieve the original message, even with the correct key. More formally, we can define symmetric ciphers as follows:

DEFINITION 3.1. A cipher defined over (M, K, C) is a pair of efficient algorithms (**E, D**) where
E: K x M $\rightarrow$ C
D: K x C $\rightarrow$ M
satisfy the correctness property:
$\forall$ k $\in$ K, m $\in$ M, **D**(k, **E**(k,m)) = m.

In other words, the correctness property guarantees that the decryption algorithm can return the ciphertext to the original message, if it uses the same key as the encryption algorithm. Additionally, we want these ciphers to be secure. Since the key space can be smaller than the message space, we defined a different type of security, known as semantic security. In this definition, we do not want an PPT TM (probabilistic polynomial time Turing machine) to be able to tell the difference between the distributions formed by the outputs of the encryption algorithm on two different different messages, given random samples. More formally, we can define semantic security as follows:

DEFINITION 3.2. A cipher has semantic security when $\forall m_0, m_1 \in \mathcal{M}$: **E**(k, m_0) $\approx_c$ **E**(k, m_1)

Stream ciphers are a type of symmetric cipher which use pseudorandom generators to expand a short random key into a much longer pseudorandom string, which is the length of the message. The initial key is random and secretly shared between the two communicating parties. The pseudorandom string is then XOR with the message to produce the ciphertext. Using this, we can define the encryption and decryption algorithms of a stream cipher as follows:

DEFINITION 3.3. Let **E** and **D** represent the encryption and decryption algorithms of a stream cipher, respectively. Also, let PRG represent a pseudorandom function. Then
E(k,m) = PRG(k) $\oplus$ m
D(k,c) = PRG(k) $\oplus$ c
where k $\in$ K, m $\in$ M, c $\in$ C

Previously, we had looked at the disadvantages of the One-Time Pad, namely that the key had to be as long as the message. This is what led us to explore stream ciphers. So what are some disadvantages associated with stream ciphers?

CLAIM 3.4. *In stream ciphers, given **E**(k,m), it is trivial to find **E**(k,m $\oplus$ 1)*

Proof. Recall that in a stream cipher
E(k,m) = PRG(k) $\oplus$ m
Therefore, XOR **E**(k,m) with 1 yields
PRG(k) $\oplus$ m $\oplus$ 1

$= \text{PRG}(k) \oplus (m \oplus 1)$
$= \mathbf{E}(k, m \oplus 1)$

□

This is not ideal because an attacker learns about the ciphertexts of two different messages simply by seeing one ciphertext.

3.2 RC4 and Content Scramble System Stream Cipher

While this highlights a vulnerability of stream ciphers as a whole, some stream ciphers have specific vulnerabilities. For example, RC4 was a stream cipher widely used in HTTPS, WEP, and even by Google. One weakness of this stream cipher was that it was not truly pseudorandom. This means that the distribution formed by the output of the encryption algorithm was not computationally indistinguishable from the random distribution. Why does this matter? Because of this, it was possible for an attacker to recover the original message and key by sampling enough of the encryption distribution of a single message. One common place this attack could be used was to get retrieve the cookies from your web browser. [2] Another weakness of the RC4 stream cipher is it was vulnerable to related key attacks. This means the attacker did not know the specific values of the keys, but could discern some relationship between them after observing the ciphertexts the produced. This was another way for an attacker to retrieve the original message.

Another stream cipher with its own vulnerability is the Content Scramble System (CSS) stream cipher. The CSS stream cipher was used in DVD's and Bluetooth systems. It was designed to be easily compatible with hardware. Because of this, the CSS stream cipher relied on two linear feedback shift registers (LFSR's).

An n-bit LFSR is an array of n bits. On every clock cycle, the last bit is outputted and all the bits are shifted over to the right by 1. The LFSR is used as a PRG since the outputted bit is pseudorandom and the initial values in the array are determined by the key. To replace the missing first bit, bits in certain positions (called tap positions) were XOR together, and the result was inserted in the leftmost position.

Since the CSS stream cipher used a 40-bit key, the two LFSR's were 17-bit and 25-bit. The initial string in the 17-bit register was the first 16 bits of the key with a 1 bit appended to the front. Similarly, the initial string in the 25-bit register was the last 24 bits of the key with a 1 bit appended in the front. The 1 bit appended to the front of each LFSR was to prevent them from being all 0's. This would cause an endless cycles of all 0's in the register. For this cipher, each LFSR ran for eight clock cycles. The resulting eight-bit strings were added together, along with a carry bit that was set to 0 to start. This sum was computed modulo 256. If the sum was greater than 256 before the modulo was applied, the carry bit would become 1. Otherwise, the carry bit was 0. This sum modulo 256 was used as the next part of the PRG. Then the registers would shift, as described above, and the process would repeat. For the sake of simplification, this modulo 256 creation on the PNG will be treated as an XOR operation between the outputs of the LFSR's. However, the same underlying principles will hold in either case.

The main vulnerability of the CSS stream cipher comes from the fact that attackers often know the first bytes of the message being encrypted. How does the attacker know this? For example, say an attacker knew that the message was in fact a png file. A certain amount of the first bytes of the plaintext will be the set up and lines of code that make the png file a

png. Assume, for the sake of example, that the amount of initial message known is the first twenty bytes.
Since the CSS stream cipher is a stream cipher, it produces its ciphertext by XOR the output of the PRG (created by the LFSR's in this case) with the message. This means that the XOR of the first twenty bytes of the ciphertext and the original message will yield the first twenty bytes of the PRG. The attacker can then try every combination of the starting 17 bits in the 17-bit LFSR. Then, the attacker can use the first 20 bytes produced by the 17-bit LFSR and XOR this with the first twenty bytes of the PRG. Since the PRG is created by the XOR of the two LFSR's, thus XOR will result in a suspected output of the 25-bit LFSR.
However, it is easy to check if this 20 byte string is an output of the 25-bit LFSR, since the 25-bit LFSR has only a few possible outputs. This means once we have the correct starting string in the 17-bit LFSR, we can easily find the starting string in the 25-bit LFSR. This allows us to find the key used, by combining the initial values in the LSFR's, minus their first bits. Therefore, we could retrieve the key used in the amount of time it takes us to try all 17-bit starting strings in the 17-bit LFSR. Since each bit is either a 1 or a 0, there are only 2^16 possibilities to check because the first bit is always set to 1 initially. This is a much better approach than the naive approach of exhaustive key analysis. Since the initial key was size 40 bits, there would have been 2^40 combinations to check.

3.3 Modern Day Stream Ciphers

Stream ciphers currently in use attempt to protect their vulnerabilities created by their linear operations by introducing randomness into their encryption scheme. Now, their new encryption algorithm is in the form **E**(k, m; r), where the PRG incorporates randomness by becoming a function of the key and some other random input. This randomness is introduced into the system by sampling the machine's surroundings. For instance, the randomness can be the humidity, or the last time there was a keyboard interrupt. This string of randomness is outputted alongside the ciphertext, otherwise the intended receiver could not recreate the PRG output to retrieve the original message. This pair of key and random input is used only once. Otherwise, this scheme would have have the same vulnerabilities as now broken stream ciphers.

3.4 Introduction to Block Ciphers

Unlike stream ciphers which operate on the entire plaintext at once, block ciphers operate on chunks of the plaintext. Block ciphers improve upon stream ciphers by performing more complex operations on the plaintext.

Definition 3.5. [1] A **block cipher** consists of a set of algorithms (E, D), defined over key set $\mathcal{K}$ and finite message/ciphertext set $\mathcal{X}$.

Block ciphers transform the given block of plaintext by invertible functions in iterations known as **rounds**. Let Rd be the round function, m be the initial plaintext block, j be the number of rounds, and k_i be the ith key derived from a expansion function applied to key k. Then, a block cipher will execute the following steps to get the corresponding ciphertext c.

$$
\begin{aligned}
m_1 &= Rd(k_1, m) \\
m_2 &= Rd(k_2, m_1) \\
&\vdots \\
c &= Rd(k_j, m_{j-1})
\end{aligned}
$$

Examples of block ciphers include 3DES and AES. 3DES has a 168 bit key size and a 64 bit message and ciphertext size. AES has a message and ciphertext size of 128 bits, but has multiple different key sizes of 128, 192, and 256 bit. These different key sizes help AES remain secure, even as computer processing power increases.

3.5 Security for Block Ciphers

We've shown the components that make up a block cipher, but how do we know if this is secure? Recall that the foundation for defining the security of a stream cipher relies on the security of a PRG, namely that it should be difficult for an adversary to distinguish an m-bit output of a PRG given a random seed from a uniformly random m-bit string. We need to do something similar for block ciphers.
Since block ciphers operate on blocks, we naturally would want to define our security for some function that produces a seemingly random n-bit output for any n-bit string for a k-bit key. Additionally, we want to guarantee that for any key k, the permutation $E(k, .)$ also looks like a random permutation [1]. We also require that this function be efficiently computable and invertible so that we can decrypt our ciphertexts. This differentiates our ideal type of function from a PRG, since a PRG simply expands a given n-bit string to a longer, pseudorandom m-bit string.

Definition 3.6. A **pseudorandom function** (PRF) is a function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{C}$, such that $F(k, m)$ is efficiently computable for every k and m and is indistinguishable from random.

Definition 3.7. A **pseudorandom permutation** (PRP) is a function $F : K \times X \to X$ such that
$F(k, x)$ is efficiently computable for every k and x,
$F(k, .)$ has domain = image and is one-to-one, and
$F^{-1}(k, y)$ is efficiently computable for every k and y where $F^{-1}(k, F(k, x)) = x$

Any secure block cipher must be a PRP due to its guarantees. Real-world examples of block ciphers include AES and DES, both of which are PRPs. $|\mathcal{X}|$ is 2^{64} for 3DES and 2^{128} for AES.
To define the security of a PRF, we can apply the concept of running games against adversaries.

Definition 3.8. (PRF Security as a Game) A PRF F, is secure if for all $x \leftarrow \mathcal{X}$ and probabilistic polynomial time adversary A

$$|Pr[A(y) = 1|Game\ 0] - Pr[A(y) = 1|Game\ 1]| < \epsilon$$

where $y = F(k, x)$, $k \leftarrow \mathcal{K}$ for Game 0, y is chosen at random for Game 1, and ϵ is a negligible function.

One component of this definition that is different than previous definitions of security as a game is the adversary's ability to choose an input in game 0. This means that we want our PRP's to be secure, no matter the inputs that the attacker uses. Because of this, we do not want the y outputted in Game 1 to be random all the time. If the adversary inputs the same string x more than once, we always want to output the same string y in return. More formally:

Remark 3.9. Index every input x by the adversary and random string y outputted to the adversary in game 1, such that y_i corresponds to x_i. Then, if $x_j = x_i$ for any j > i, $y_j = y_i$ $\forall i, j \in \mathbb{N}$.

Example 3.10. Now we can consider a few examples of PRFs and PRPs and make some statements about each of them.

- Given that $F : K \times X \to X$ is a secure PRP, F is also a secure PRF. This is because a PRP is simply a stricter version of a PRF.
- Given that $F : K \times X \to \{0,1\}^{128}$ is a secure PRF, the function

$$G(k, x) = \begin{cases} 1^{128} & \text{if } x = 0 \\ F(k, x) & \text{otherwise} \end{cases}$$

  is not a secure PRF. This is because we can create an adversary A that queries $x = 0$ and returns 0 if it receives 1^{128}, 1 otherwise. For a game 0 that uses G and a game 1 that is completely random, A would be able to distinguish G in almost all situations.
- Given that $F : K \times X \to \{0,1\}^{128}$ is a secure PRF, we can build a PRG $G : K \to \{0,1\}^{4096}$ from F. If $X \in \{0,1\}^8$, we can define PRG as follows:

$$G(k) = F(k, 0^8) \mid\mid F(k, 0^7 1) \mid\mid F(k, 0^6 10) \ ...$$

  This will suffice as long as there are $\frac{4096}{128} = 2^5$ terms that make up G.

In the next section, we will look at how the block cipher known as DES works.

3.6 Data Encryption Standard (DES)

The **Data Encryption Standard** (DES) is a block cipher that relies on a structure known as a **Feistel network**.

Definition 3.11. A Feistel network is a structure that consists of an invertible function $F : \{0,1\}^{2n} \to \{0,1\}^{2n}$ given j functions $f_1, ...f_j : \{0,1\}^n \to \{0,1\}^n$ which may not be invertible. Given F, we encrypt message x by computing

$$\begin{aligned} L_1 &= R_0, \ R_1 = L_0 \oplus f_1(R_0) \\ L_2 &= R_1, \ R_2 = L_1 \oplus f_2(R_1) \\ &\vdots \\ L_j &= R_{j-1}, \ R_j = L_{j-1} \oplus f_j(R_{j-1}) \end{aligned}$$

where L_0 concatenated with R_0 is equal to the initial message. The final ciphertext is $c = L_j||R_j$.

It might seem unintuitive that a function consisting of non-invertible functions is itself invertible, but this is possible because of the XOR operations we perform. For example, given L_j and R_j, we can get that $R_{j-1} = L_j$ and $L_{j-1} = R_j \oplus f_j(R_{j-1})$. We can continue this pattern all the way back to L_0 and R_0.
DES is a Feistel network where n= 64 bits, k = 56 bits, j = 16, and k_j = 48 bits. Each function $f_i = F(k_i, x)$ performs the following steps:

- Expands x from 32-bits to 48-bits.
- XORs x with k_i and splits it into 8 equal parts of 6 bits each.
- Runs each partition of x through an S-box (substitution lookup table) that maps each 6-bit input to a 4-bit output.
- Collects all 32-bits from the previous step and rearranges it according to a P-box, producing the final output.

The S-boxes and P-box are important components of F since they provide **non-linearity**, a property that makes it harder for an adversary to distinguish F from a PRF.

3.7 Conclusion

In this lecture, we started out by looking at some examples of stream ciphers. In doing so, we noted some vulnerabilities they have because of their linear operations. This motivated us to look at a new type of cipher: block ciphers. We first looked at pseudorandom functions and pseudorandom permutations, since they are the components we use to create block ciphers. We then showed how block ciphers were secure by looking at the security of pseudorandom functions (and, by extension, pseudorandom permutations). Lastly, we looked the inner workings of DES, a block cipher designed as Feistel network.

Acknowledgement

These scribe notes were prepared by editing a light modification of the template designed by Alexander Sherstov.

References

[1] D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography.* Cambridge University Press, 0.5 edition, 2020.

[2] M. Green. *Attack of the week: RC4 is kind of broken in TLS*, 2013 (accessed September 3, 2020).