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

Instructor: Dakshita Khurana
Scribe: Jeremy Poynton, Hamilton Silberg
Date: September 3, 2020

LECTURE

4

Block Ciphers II

In the previous lecture, we introduced block ciphers as a solution to the problems of stream ciphers. We looked into the definitions of PRFs and PRPs and analyzed the games that can be played to determine the semantic security of these devices. As a continuation of last lecture, we will continue to look into the significance of PRFs and PRPs, especially in the case of using a block cipher as a PRP to create a semantically secure cipher.

Last lecture introduced the DES algorithm with the Feistel network structure, with the ability to create invertible functions out of uninvertible components. We will investigate attacks on the DES algorithm and introduce the evolution of DES into further modified alogrithms to improve security.

Expanding on the example of DES, we will also look at the AES block cipher. AES has a substitution-permutation network, distinguishing it from DES. Comprising entirely of many invertible steps, we see the non-linearity we identified as being critical to a secure cipher implemented within the various steps of the algorithm. Along with going through its implementation, we will compare its security with DES through their known best attacks.

Finally, we will look into a few modes of operation for block ciphers. These modes each define a way to use the block cipher to encrypt and decrypt a plaintext which comprises of many blocks. Each of these modes have different properties which will affect the security and performance of the overall process. While specifically used for AES, these modes can be applied to many block ciphers.

4.1 Building a PRF from a PRG

Last lecture we talked about how you can construct a PRG from a PRF by concatenating the outputs of the PRF with an incrementing counter. The result is a secure PRG because every output from the PRF is guaranteed to be pseudorandom.

We will now talk about how a PRG can be constructed from a PRF. Similar to how we concactenated PRF input to create a PRG, we can partition PRG outputs to create a PRF. That is, given a secure PRG G that doubles the key, we can define a PRF F as:

$$F(K, x \in \{0,1\}) = G(K)[x]$$

As noted, however, this will only work with $x \in \{0,1\}$, as it assigns both halves of the output from the PRG. What happens if we want to extend past that range of x? It turns out, we can construct a binary tree by feeding each half of the PRG's output back into the PRG. This will still be guaranteed pseudorandom because the initial output is pseudorandom. Now given a binary input x, we can follow the corresponding edge in the graph for each bit of x. More formally, with x_n representing the nth bit of x:

$$F(K, x \in \{0,1\}^2) = G(G(K)[x_0])[x_1]$$

This will allow us to create a tree for any input size of x desired. We can simply add more layers and recursive calls to the tree. For a tree of depth a, this method will allow us to cover up to 2^a values of x. Example for a three bit size of x:

$$F(K, x \in \{0,1\}^3) = G(G(G(K)[x_0])[x_1])[x_2]$$

This process can be extended as needed. While inefficient in nature, this construction does prove a secure PRF because each leaf in the tree is guaranteed to be pseudorandom, as they are made by feeding pseudorandom previous inputs into the PRG. And thus, the existence of a secure PRG implies a secure PRF.

4.2 DES attacks

DES is vulnerable to certain brute force attacks with known plaintext. That is, with two sets of plaintext and their corresponding encryption, we can iterate through the entire keyspace. If a key is found that encrypts both plaintexts to the correct ciphertexts, there is high probability that this is the only correct key.

More formally, given two pairs (m_1, c_1), (m_2, c_2) where $c_1 = E(k_1, m_1)$, $c_2 = E(k_2, m_2)$, with high probability there is at most one key k such that $c_1 = E(k, m_1)$ and $c_2 = E(k, m_2)$

Because DES uses a key of 56 bits, running this exhaustive key search would take time 2^{56}. With modern computing available, this level of security can be broken quickly with ease. For example, in a series of challenges in 1999 called DES Challenge III, the key to a DES encryption was found in just over 22 hours using plain brute force.

Due to the low level of security guaranteed by DES, there have been multiple extensions to try and increase security against this type of attack.

4.3 Triple-DES

The first intuitive way of increasing security against an exhaustive key search is to increase the key size. Triple-DES uses a key-size of 168 bit, breaking down to three separate DES

keys. The method uses a DES encryption followed by a decryption, followed by another encryption.

$$3E((k_1, k_2, k_3), m) = E(k_1, D(k_2, E(k_3, m)))$$

Tripling the size of the keyspace would increase the time complexity from a key search to 2^{168}. However, due to the algorithm's vulnerability against meet in the middle attacks, the effective security it provides is around 2^{118}. Meet in the middle attacks will be discussed shortly.

One thing to note in the structure of Triple-DES is that the middle operation is flipped from encypt to decrypt. This is a unique choice that improves security in situations where the three keys are not completely independent.

One may consider why Triple-DES is used first before trying Double-DES or 2-DES. It turns out 2-DES was considered, however, it is particularly vulnerable.

4.4 2-DES

2-DES uses a key size of 112 bits with an encryption followed by an encryption, and is defined as follows.

$$2E((k_1, k_2), m) = E(k_1, E(k_2, m))$$

If 2-DES was not vulnerable to attack, it would take 2^{112} time to break via exhaustive search. This is better than the 2^{56} time in the original DES. However, the meet in the middle attack makes 2-DES significantly easier to attack. The time of the attack can be taken down to about $2^{56}\log(2^{56})$, slightly better than the 2^{56} of original DES. The specifics of this meet in the middle attack are described in the next section.

4.5 Meet in the middle

The meet in the middle attack is an extension of the known plaintext brute force algorithm described earlier. Meet in the middle describes brute forcing an encryption key and a decryption key and looking for matches. It can be described as working forwards from the plaintext, and working backwards from the ciphertext, until a match is found. Hence, the term "meet in the middle."

In the context of 2-DES, the attacker takes the plaintext and stores an encrypted intermediary text for every possible key. Similarly, the ciphertext is decrypted using every possible key. If there exists an intermediary text that matches from the encrpytion and decryption, it is highly likely that those two keys were the keys used for 2-DES.

Using this attack, the entire 112 bit keyspace does not have to be searched. Instead, two separate 56 bit spaces are searched, and the resulting lists are scanned for matches. As such, this reduces the time consideration to $2^{56}\log(2^{56})$. The extra log is the time needed to sort through the intermediary texts. It is worth noting that this attack does have a space requirement to store the intermediary texts.

Meet in the middle can be extended to Triple-DES also. Like in 2-DES, the ciphertext is decrypted using every possible key. However, the first two operations are treated as a single step. That is, an intermediary text is generated for every possible (k1, k2) pair. Then the intermediary texts are searched for matches. The attacker now must searched through a 2^{112} space and a 2^{56} space, bringing the time complexity to attack Triple-DES to $2^{112} \log 2^{112}$. Again, the log term is used for scanning for matches.

Because of the way DES is structured, it is clear that no matter how many encryption or decrpytion operations are present, a meet in the middle attack would still be possible with known ciphertexts. That is, a theoretical 4-DES or 5-DES could save time from an exhaustive key search by storing intermediary texts into tables and looking for matches. At that point, the problem becomes a trade-off between space and time concerns.

4.6 DESX

DESX is a variant of DES encyption that is less vulnerable to meet in the middle attack than its multiple DES counterparts. DESX uses a triple keyspace, like Triple-DES, but instead uses various xor operations with a single DES encryption.

$$EX((k_1, k_2, k_3), m) = k_1 \oplus E(k_2, m \oplus k_3)$$

The process of using xor before and after an encryption has been coined *key whitening*, and DESX was the first scheme to employ this technique. k_1 and k_2 are 64 bit keys, increasing the key size to 184 bits. However, the effective security of DESX has been determined to be that of time complexity 2^{120}.

4.7 The AES algorithm

The Advanced Encryption Standard or **AES algorithm**, also known as the Rjindael algorithm, derived from competition to select a standard encryption algorithm. It has a **substitution-permutation network** design, which means it interleaves substitution and permutation functions as its method of non-linearity. Another name used to describe this type of structure is **confusion-diffusion network**. One important detail of the AES algorithm is the multiple key size options. The three levels provide a trade off between performance and security, with performance improvements for smaller keys due to less rounds, and security improvements for larger keys due to increased key space. These options within the algorithm allow for more simple user control of security as compared to DES, where a separate system of 3DES or DESX must be created in order to improve security.

The specific AES structure follows a series of rounds accomplishing this substitution and permutation. Between each round, a generated round key is XOR'd with the output of the prior round as the input to the next round. The amount of rounds is determined by the size of the key, with larger keys causing more rounds. There are 10 rounds for a 128 bit key, 12 for 192, and 14 for a 256 bit one. The key expansion works by expanding the input key into several **round keys** through a relatively simple process of shifting and XORing with specified values. The larger keys will produce larger round keys, but only the first 128 bits are used as the true round keys.

During the algorithm, the 128 bit input data block is represented by a 4x4 grid of bytes, in which this layout is particularly relevant for some of the steps in the process. For each round, the first step is a substitution box (or **S-box**) transformation where each byte is substituted with another following a lookup table. This is one of the key points of non-linearity introduced into the algorithm. The specific lookup table is specified by the AES procedure and is specially designed to be non-linear. In particular, no inputs will match their output nor be the exact complement. The result is a one-to-one mapping that is non-linear and invertible.

After the S-box, there is a **shift rows** step. Since the data block is represented by the four-wide-four-tall matrix, each row is cycled independently. Row i is shifted left i times, where the first row is index 0. This step is also invertible.

$$ShiftRows \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{bmatrix}$$

The final step of the round is the **mix columns** step. Here each column is independently mixed using its four elements. A specified matrix describes the weights to use. The matrix multiplication that occurs here is within $GF(2^8)$, where each byte represents a polynomial where $x^0, x^1, x^2 ... x^7$ may either be present or absent. The resulting output of this operation becomes the input for the next round of the cipher once the round key is XOR'd. A small detail of AES is that mix columns is not applied to the final round. This was done to make the encryption and decryption process more similar. [1]

$$\begin{bmatrix} 1 & 1 & 2 & 3 \\ 1 & 2 & 3 & 1 \\ 2 & 3 & 1 & 1 \\ 3 & 1 & 1 & 2 \end{bmatrix} * \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} = \begin{bmatrix} s_{0,c} + s_{1,c} + 2s_{2,c} + 3s_{1,c} \\ s_{0,c} + 2s_{1,c} + 3s_{2,c} + s_{1,c} \\ 2s_{0,c} + 3s_{1,c} + s_{2,c} + s_{1,c} \\ 3s_{0,c} + s_{1,c} + s_{2,c} + 2s_{1,c} \end{bmatrix}$$

At the end of the algorithm, we are left with a ciphertext of 128 bits, using our key of 128, 192, or 256 bits. AES has been a standard for many years, and has undergone significant cryptanalysis for attacks and vulnerabilities.

4.8 Attacks on AES

AES is empirically stronger when using DES or 3DES as a comparison point. While the key space for AES can be significantly larger ($2^{256} > 2^{56}$) the best general key recovery attack for AES_{128} is in 2^{122}, which is slightly better than searching the entire key space, compared to the general key recovery attack on 3DES, which is 2^{118}, a significantly smaller value than the entire key space of 2^{168}.

There are better attacks, however the input they require is quite specialized. Due to the key expansion being simple relative to the rest of the algorithm, there exists an attack on a set of keys related in specific ways where the keys can be found in only 2^{99} time. This

however does not affect the overall security of AES given proper implementation, as keys used should otherwise not be related in this manner. Generally speaking, the AES encrypt function is treated as a CPA secure PRP, converting an input to an output of the same size.

4.9 Block cipher modes of operation

Given we have AES acting as our secure PRP, we want to create semantically secure encryption. Recall semantically secure encryption, where for two given messages $m0, m1 \in 0,1^n$ and PRP $\mathcal{E}$ and key $k \leftarrow 0,1^m$, $\mathcal{E}(k, m0) \approx \mathcal{E}(k, m1)$.

Consider the simplest approach of applying AES to a given message: break the message into the necessary blocks and for each block m_i find $c_i = AES(k, m_i)$ using the key k, and concatenate the c_i. This method is known as electronic code book, or **ECB**. This method is not semantically secure, and in fact can be shown to be discovered different from random by a very simple adversary leveraging a property of ECB: if $m_i = m_j$, then $c_i = c_j$.

EXAMPLE 4.1. Consider the CPA game, where we have PRP $\mathcal{E} : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{C}$ and key $k \leftarrow \{0,1\}^m$

The adversary will supply us with ordered inputs x_i.

In game 0 we supply the adversary $\mathcal{A}$ with $\mathcal{E}(k, x_i)$, while in game 1 we supply a random output in $\{0,1\}^n$.

Suppose $x_i = a||a$ where a matches the size of the PRP block size. If we are following ECB mode for our PRP, then in game 0 our output will be $\mathcal{E}(k, a)||\mathcal{E}(k, a)$, which will be readily distinguishable from a randomly generated output, and thus a PPT adversary can reliably win the CPA game, showing ECB to not be semantically secure.

Familiar problems are present, reminiscent of the basic substitution ciphers. Important information about the data can be extracted through frequency analysis or the likes, clearly information available that we want hidden.

Improvements can be made with adding a bit of variety to the blocks. One method of doing this is utilizing a PRF as block cipher with a counter. This method is called **deterministic counter mode**.

Given PRF $F : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{M}$

DEFINITION 4.2. $E_{DETCTR}(k, m) = m_0 \oplus F(k, 0)||m_1 \oplus F(k, 1)||...||m_L \oplus F(k, L)$

This mode fixes our issue with matching input blocks having matching output blocks. However, this methodology runs into similar problems as the stream cipher : this key should only be used once. Since the generated string to XOR the message with is the same given the same key, this opens up any additionally uses of the key to a basic attack. Given messages m_0 and m_1, if they are encrypted with key k and PRG G made from a string of PRF outputs from the counter, we will see that $m0 \oplus G(k) = c_0$ and $m_1 \oplus G(k) = c_1$. Thus, $c_0 \oplus c_1 = m_0 \oplus m_1$, resulting in security concerns if we allow adversaries to query the key more than once. Looking forward, we will use different modes of operation to try to fix our current problem of security for reused keys.

4.10 Conclusion

In this lecture we continued our journey through block ciphers, looking at specific examples in practice of DES and AES and how they relate to the discussion of core security structures such as PRPs. We looked at the implementations of these algorithms to get a stronger sense of what is important for a cipher with our desired attributes, and in some cases how some properties of the algorithm can remove semantic security. We also introduced modes of operations for block ciphers and their implications, and how depending on how a secure PRP is applied can make or break a secure encryption. Looking forward to future lectures, we will cover further modes of operation that achieve different properties, such as key reuse and better performance in machines through the option of parallelization of the encryption and decryption algorithms.

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.