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

Instructor: Dakshita Khurana
Scribe: Yiran Li, Maxiwell Luo
Date: September 17, 2020

LECTURE
8

Advanced Symmetric Encryption

Last lecture, we look in-depth at Authenticated Symmetric Encryption. We first defined the Authenticated Encryption that must satisfy two properties: the cipher should be semantically secure under CPA, and cipher (E, D) must has ciphertext integrity. The Authenticated Encryption has two implication that it ensure the Authenticity and it provides the security against chosen ciphertext attack. However, Authenticated Encryption cannot prevent replay attacks and cannot account for side channels. To solve the problems, we can apply pad-then-encrypt-then-MAC: MAC is checked first and cipherext discarded if invalid. Or we can apply MAC-then-CBC, but padding oracle can destroy it even though it provides A.E..

In this lecture, we talked about Advanced Symmetric Encryption. We introduced Deterministic Encryption, Format-preserving Encryption, and Number theory (Public key encryption). Deterministic Encryption is the encryption does not use the randomness beyond using secret key which use the randomness. Format-preserving Encryption is the encryption which can efficiently map the database that we can construct a PRP from a secure PRF. It will provide same security as Feistel construction with no integrity. Number theory is the preparation of the public key encryption. Before this section, we have only looked at encryption systems with semantic key, share key, and private key.

8.1 Deterministic Encryption

Why we use the Deterministic Encryption? Suppose there is a server want to save its record on the data base. The server has the data entries that wanted to be encrypted and keep these data entries on the encrypted database, and database owner is untrusted. The server does want the data known by the database owner and server also want to efficiently retrieve the data. Therefore in this case, deterministic encryption will be useful. In short Deterministic Encryption will be useful in the situation where people specifically want the message to the same ciphertext every time you encrypted.

For example, if the server want to fetch the all data corresponding to Alice, it need to send the ciphertext, E(k_1, "Alice"), corresponding to plain text Alice, and get all encrypted data from database, so the server learn nothing in this struct.

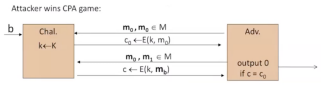

FIGURE 8.1: Attack wins CPA.

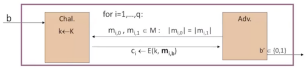

FIGURE 8.2: Deterministic CPA.

This encryption is problematic that it will leak information, since attack can tell when two ciphertexts encrypt the same message. Though it is the feature in Deterministic Encryption, it will lead to significant attacks when message space M is small. Besides, Deterministic Encryption is not CPA-secure, and See Figure 8.1 for an example how an attack wins CPA game. The attacker can ask two encrytion message m_0, and get back the encryption of m_0. Then it gives another message, m_0 or m_1, so attacker can tell which of the two is being encrypted. Therefore by repeat the messages, attacker can break the encryption.

THEOREM 8.1. *$E = (E, D)$ is semantic secure under deterministic CPA if for all efficient A: $ADV_{dCPA}[A, E] = \left| Pr[EXP(0) = 0] = Pr[EXP(1) = 1] \right|$ is negligible.*

To ensure the deterministic CPA game, we need to make sure that the attacker can never submitted two messages that is same, $m_0 \neq m_1$. Like Figure 8.2, where $m_{1,0}$, ..., $m_{q,0}$ are distinct and $m_{1,1}$, ..., $m_{q,1}$ are distinct.

Is counter mode with a fix IV, as Figure 8.3 deterministic CPA secure (IV is a set to a fix string which is not randomized)? No, the answer is no and we can find an adversary as Figure 8.4 to win the CPA game.

Therefore how should we construct a cipher which is semantic secure under deterministic CPA? The answer is Synthetic IV (SIV). Let (E, D) be a CPA-secure encryption, $E(k, m; r)$ where r is random IV, and let F: K x M $\rightarrow$ R be a secure PRF, so we can define $E_{det}((k_1, k_2), m)$ by:

- First compute $r = F(k_2, m)$
- Output $E(k, m; r)$ by apply PRF to messages

THEOREM 8.2. *E_{det} is semantic secure under deterministic CPA*

FIGURE 8.3: Counter mode with fixed IV.

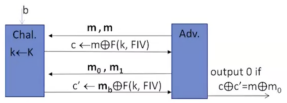

FIGURE 8.4: Adversary which can break deterministic IV.

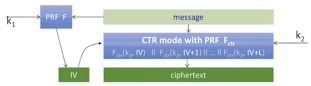

FIGURE 8.5: SIV-CRT.

Since the cipher can get distinct messages implies all r's are indistinct from random, and it well suited for messages longer than one AES block.

We've explained the Deterministic Encryption, let's move to Deterministic Authenticated Encryption. Deterministic Authenticated Encryption is the encryption provides the deterministic security and cipher integrity with deterministic guarantee that every pairs of identical messages are encrypted into identical ciphertexts. We can look at a special case of SIV as the example constructing Deterministic Authenticated Encryption. The special case is SIV-CTR which is a SIV where cipher is counter mode with random IV, and it should work as Figure 8.5. You have two keys, $k_1 and k_2$, and a message m, so you for a encryption, $Enc(k_1, k_2, m)$. Then you apply the PRF to the message to derive initialization vector(IV), and use this IV under the counter mode. Run the counter mode with encryption keys to get the ciphertexts. For the decryption as , we have IV and ciphertexts and we can perform counter mode decryption with k_2 to get message; then given the message and PRF, we can check if the IV matches.

THEOREM 8.3. *If F is a secure PRF and CRT from F_{ctr} is CPA-secure then SIV-CTR from (F, F_{ctr}) provides DAE.*

Another construction is using PRP. Let (E, D) be a secure PRP, $E : KxX \to X$. The proof will be like following:

- Let $f : X \to X$ be a truly random invertible function (q: random values in X)
- Then in EXP(0), adv. sees: $f(m_{1,0})$, ..., $f(m_{q,0})$
- Then in EXP(1), adv. sees: $f(m_{1,1})$, ..., $f(m_{q,1})$

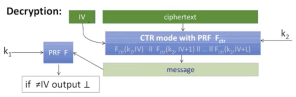

FIGURE 8.6: Decryption of SIV-CRT.

Deterministic CPA secure encryption for 16 bytes messages, and it need PRPs to secure longer messages.

8.2 Format Preserving Encryption

Format preserving encryption, like the name implies, ensures that the ciphertext of a message matches the format of the message. Format preserving encryption is useful when integrating encryption to an existing data flow. Say for example that there is some process that expects a credit card number, and passes that credit card to the next process and so on. If we wanted to add encryption to this workflow, but didn't want to modify any of the intermediate steps, we would need to make sure that the ciphertext passed in looks like a valid credit card.

Describing the process more formally, we want to map a message taken from some message space into the same space. We have already seen a construct that gives us something very similar. That is a pseudo-random permutation (PRP). Essentially, we wish to build a PRP on a message space of s ($\{0, ..., s-1\}$), where $0 < s \leq 2^n$, using a secure PRF $F : K \times \{0,1\}^n \to \{0,1\}^n$. The PRF is some encryption scheme such as AES.

The steps to accomplish format preserving encryption then are as follows:

- use the PRF to create a PRP on a space at least as large as $\{0, ..., s-1\}$
- use the PRP repeatedly on the message until the output is valid

8.2.1 Step 1: PRF to PRP

The goal in this step is to use a PRF $F : K \times \{0,1\}^n \to \{0,1\}^n$ to construct a PRP $G : K \times \{0, ..., s-1\} \to \{0, ..., s-1\}$. In fact, we have already seen how to construct a PRP from a PRF in lecture 4, when learning about DES. Recall that a Feistel Network constructs an invertible function $F : \{0,1\}^{2n} \to \{0,1\}^{2n}$ from a series of non-invertible functions (PRFs) $f_1, ..., f_j : \{0,1\}^n \to \{0,1\}^n$. It does this by splitting the input into a right and left half. In each round, it passes the left half to the right half without modification, but XORs the left half with the function applied to the right half to get the left half for the next round. See Figure 8.7 for a visual description.

For our construction, we truncate F to operate on half of the string ($F' : K \times \{0,1\}^{t/2} \to \{0,1\}^{t/2}$), and use it for each round. t is defined to be the smallest exponent of two that is greater than or equal to s ($2^{t-1} < s \leq 2^t$). The PRP constructed using this Feistel Network is defined as $G : K \times \{0,1\}^t$. Note that because 2^t can be greater than s, the space is likely larger than $\{0, ..., s-1\}$, the space of the original input. We solve this problem in the next step.

8.2.2 Step 2: Repeatedly Apply PRP

Now that we have constructed a PRP, we want to use it to get some output of the message that's in the same space as the input. However, because our PRP operates on a space larger than $\{0, ..., s-1\}$, we will not always get a valid output by only applying the PRP once to the input. However, recall that the space of the PRP, $\{0,1\}^{t/2}$, is defined where ($2^{t-1} < s \leq 2^t$). This means that $\{0, ..., s-1\}$ covers at least half of the space of the PRP. The expected number of necessary iterations to get a valid output is thus 2.

It is quite simple to see that this construction is indeed difficult for an adversary to attack. The PRP is a random permutation over a larger space, which, when done iteratively until a

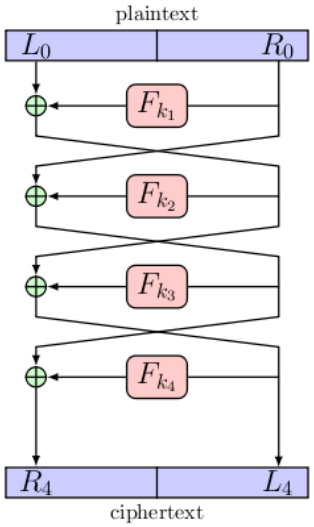

FIGURE 8.7: Feistel Network Diagram.

value in a smaller space is reached, is the same as a random permutation on the smaller space. The intuition is the same as starting with a number 1-3 on a six-sided die, and continually rolling the die until a number 1-3 is rolled again. Because a PRP is indistinguishable from a uniform random distribution, each application is a computationally independent random event, like with the die. Mathematically, this is represented as:

For all sets $Y \subseteq X$, applying the transformation to a random permutation $\pi : X \to X$ gives a random permutation $\pi : Y \to Y$.

In randomness of the PRP makes it computationally difficult for an attacker to distinguish between the output and pure randomness.

8.3 Number Theory

Thus far in the course, we have looked at private key encryption, where a secret key is shared between the sender and the receiver. However, private key encryption requires some secure side channel that can safely transfer the key between the two parties. To get around this problem, there is a class of encryption schemes that are able to maintain security even when the key (or part of it) is public. To prepare us for such public key encryption schemes, we look at some mathematical properties that will be useful. Namely, we will look at modular arithmetic.

8.3.1 Notation

The notation going forward in the course is as follows:

- N is a positive integer

- p is a prime
- $\mathbb{Z}_N = \{0, 1, 2, ..., N-1\}$ (a set of integers from 0 to $N-1$)

8.3.2 Modular Arithmetic

Note that when doing addition and multiplication on integers a and b modulo N, the result is always in $\mathbb{Z}_N$. The modulus operator is simply the result of the remainder of a division between two operands. For example $5 \mod 2 = 1$.

For some simple examples, let N = 16:

- $9 + 10 = 3$ in $\mathbb{Z}_{16}$
- $3x8 = 8$ in $\mathbb{Z}_{16}$
- $3 - 8 = 11$ in $\mathbb{Z}_{16}$
  - For negative modulo, keep adding N until you back in the range 0 to $N-1$

Common rules for arithmetic works as expected, for example the distributed property: $x \cdot (y + z) = x \cdot y + x \cdot z$.

8.3.3 Greatest Common Divisor

DEFINITION 8.4. for integers x, y: $\gcd(x, y)$ is the greatest common divisor of x, y

For example, $\gcd(12, 24) = 12$.

FACT 8.5. *For all integers x, y there exist integers a,b such that*

$$a \cdot x + b \cdot y = \gcd(x, y) \mod N$$

and a, b can be found efficiently using the extended Euclid algorithm

DEFINITION 8.6. For integers x, y, if $\gcd(x, y) = 1$, then we say that x and y are relatively prime ($\mod N$)

8.3.4 Module Inversion

DEFINITION 8.7. The inverse of x in $\mathbb{Z}_N$ is an element y in $\mathbb{Z}_N$ such that $x \cdot y = 1$ in $\mathbb{Z}_N$. y is denoted $x^{-1} \mod N$

Inversion is defined this way because there are no fractions in $\mathbb{Z}_N$. As an example, let N be odd. The inverse of $\mathbb{Z}_N$ is $\frac{N+1}{2}$.

$$\begin{aligned} 2 \cdot \frac{N+1}{2} \mod N &= (N+1) \mod N \\ &= 1 \mod N \end{aligned}$$

LEMMA 8.8. *x in $\mathbb{Z}_N$ has an inverse if and only if* $\gcd(x, N) = 1$

Proof.

$$\gcd(x, N) = 1 \Rightarrow \exists a, b : a \cdot x + b \cdot N = 1 \Rightarrow a \cdot X = 1 \text{ in } \mathbb{Z}_N$$

Any multiple of $N \mod N$ is 0. a, b exist due to Euclid's algorithm.

$$\gcd(x, N) > 1 \Rightarrow \forall a : gcd(a \cdot x, N) > 1 \Rightarrow a \cdot x \text{ in } \mathbb{Z}_N$$

The simplification is the same as above. □

8.3.5 More Notation

DEFINITION 8.9. $\mathbb{Z}_N^* = (\text{set of invertible elements in } \mathbb{Z}_N) = \{x \in \mathbb{Z}_N : \gcd(x, N) = 1\}$

Given an x in $\mathbb{Z}_N^*$, you can find x^{-1} using the extended Euclid algorithm.

8.3.6 Solving Modular Linear Equations

To solve a linear equation:

$$a \cdot x + b = 0 \text{ in } \mathbb{Z}_N$$

simply compute

$$x = -b \cdot a^{-1} \text{ in } \mathbb{Z}_N$$

a^{-1} in $\mathbb{Z}_N$ can be found using the extended Euclid algorithm in $O(log^2 N)$ time.

8.4 Conclusion

In this lecture, we covered deterministic encryption, format preserving encryption, and some introductory number theory along with their motivations. As we close on our discussion of private key encryption, it is interesting to see how previous constructions such as the Feistel Network in DES appear to help us add useful features to our encryption schemes. It will be interesting to see how much of the work we've done with private key encryption carries over to public key encryption as well.

Acknowledgement

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

References