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

Instructor: Dakshita Khurana
Scribe: Jason Liu, Nikil Ravi
Date: September 08, 2020

LECTURE
5

Block Ciphers (Continued), Message Authentication

Last lecture, we looked in-depth at AES, a secure block cipher, as well as ways to actually encrypt data with block ciphers. There are notable pitfalls to avoid when using a block cipher (which expects fixed-length input) on variable-length messages; for example, we can't just naïvely use the block cipher on each block in the message, since individual messages could be the same, resulting in frequency analysis attacks. We were able to produce a stream cipher from a block cipher, although we are still limited to one-time keys. In the first half of this lecture, we work on fixing this problem, so that we can potentially reuse the same key to encrypt multiple messages.

As we saw in the course introduction, there is more to cryptography than secretly transmitting messages. Another common problem in security that we will address is integrity, which is in many ways the dual of secrecy. Rather than considering a sender entrusting only some select receivers with their message, we now consider a receiver who is deciding whether they can trust the sender. This problem is orthogonal to secrecy — knowing that nobody else could read a message doesn't guarantee that you can trust its sender. For example, how can you be sure that a message you received is not from someone impersonating the sender? In the second half of the lecture, we will examine Message Authentication Codes (MACs), a scheme that addresses this problem.

5.1 Many-time keys

We have seen how to encrypt arbitrarily-long inputs with smaller keys, but so far, we require a new key for every message. For some applications, it would be nice to be able to use the same key for multiple encryptions. For example, an encrypted filesystem that required a new key for each file would be inefficient and unwieldy to use. Similarly, IP connections typically involve exchanging many small packets; it would be a massive overhead (or even impossible, depending on the packet and key size) to try to constantly negotiate a new key for every encrypted packet under IPsec. In these cases, we really want to be able to reuse the same key.

First, let's formalize exactly what many-time security means. Recall our definition of one-time security block ciphers:

Definition 5.1. E is CPA-secure iff

$$\forall m_0, m_1 \in \{0,1\}^{128}. \underset{k \leftarrow \mathcal{K}}{E}(k, m_0) \approx \underset{k \leftarrow \mathcal{K}}{E}(k, m_1) \tag{5.1}$$

where $\approx$ signifies indistinguishability, i.e., that an adversary cannot (within some error ϵ) tell the difference between the two distributions. Notice that although we assume the block size to be 128 above, this is not a general requirement for one-time security. We are also sampling some key k at random from key space $\mathcal{K}$ in both encryptions. (One consequence of sampling k at random from $\mathcal{K}$ is that there is no requirement for both ks in the equation to be the same.)

To extend this to many-time keys, we should be able to encrypt n pairs of (potentially different) messages with only one key k while ensuring the two distributions are still indistinguishable. Specifically:

Definition 5.2. E is many-time CPA-secure iff

$$\begin{aligned} \forall \vec{m}_0 &= (m_0^1, m_0^2, \ldots, m_0^n) \\ \vec{m}_1 &= (m_1^1, m_1^2, \ldots, m_1^n). \\ \big(\mathrm{E}(k, m_0^1), \mathrm{E}(k, m_0^2), \ldots, \mathrm{E}(k, m_0^n)\big)_{k \leftarrow \mathcal{K}} &\approx \big(\mathrm{E}(k, m_1^1), \mathrm{E}(k, m_1^2), \ldots, \mathrm{E}(k, m_1^n)\big)_{k \leftarrow \mathcal{K}} \end{aligned} \tag{5.2}$$

where each vector $\vec{m}$ contains n messages. The distributions of n ciphertexts (as opposed to just one), each encrypted with only one key k, are indistinguishable. Just as before, both keys k are sampled at random, and are thus not required to be the same. Also note that there are no restrictions on the actual messages — it is possible for a message to repeat within a vector. We just need n arbitrary messages per vector.

This definition already encodes an extra restriction we must put on many-time security: encrypting the same message m multiple times with the same key k must still result in what appears to be a randomly sampled ciphertext c. Specifically, this means that we expect to see a different output c for each encryption of m; otherwise, the frequency analysis attacks we have seen before will be possible. Notice that based on our formalization of many-time keys, this requirement already holds:

Claim 5.3. *Under Definition 5.2, encrypting the same message m multiple times with the same key k will result in distinct ciphertexts.*

Proof. Suppose, for the sake of contradiction, that encrypting m multiple times with the same key k resulted in the same ciphertext c. An adversary could challenge E as follows: in game 1, the adversary encrypts $\vec{m}_1$ containing n copies of the same message m. In game 2, the adversary encrypts $\vec{m}_2$ containing n random distinct messages. Then the ciphertexts for game 1, $\vec{c}_1$, will all be the same due to the assumption, while the ciphertexts for game 2, $\vec{c}_2$, will all be different. The adversary can trivially distinguish game 1 from game 2 by outputting whether the ciphertexts are the same or not, contradicting indistinguishability in Definition 5.2. Therefore, encrypting m multiple times with the same k must result in different ciphertexts. □

Based on this, we already know that E will require at least one more input for many-time encryption, since $\mathrm{E}(k, m)$ given the same k and m will always produce the same output c. Next, we look at actual strategies to achieve many-time security.

5.2 General many-time security strategies

One strategy is to use randomization to achieve many-time security: as long as we are careful enough to ensure the randomization is replicable, decryption is still possible. Specifically:

Claim 5.4. *Let* $\mathrm{F} : \mathcal{K} \times \mathcal{R} \to \mathcal{M}$ *be a PRF. Then*

$$\mathrm{E}(k, m) = (r, \mathrm{F}(k, r) \oplus m) \quad \textit{for some } r \leftarrow \mathcal{R} \tag{5.3}$$

$$\mathrm{D}(r, k, c) = c \oplus \mathrm{F}(k, r) \tag{5.4}$$

is a CPA-secure many-time cipher.

Proof. This is CPA-secure by definition of PRFs. Recall that $\mathrm{F}(k, r)$ is indistinguishable from a uniform random function by definition. Furthermore, we know that the result of $\oplus$ (XOR) with a uniform random variable is also a uniform random variable. Therefore, the output ciphertext c is indistinguishable from a uniform random variable, so a distribution of ciphertexts is indistinguishable from a distribution of uniform random variables. □

By outputting the same r that was used during encryption, decryption can produce the same random result, even though the ciphertext will change for each encryption of the same message. As we will see, outputting the random input will be required for most many-time security strategies.

In practice, this strategy is rarely chosen because it is not parallelizable. Another general strategy is to add a *nonce*: basically, an explicit third input to E, i.e., now we have $\mathrm{E}(k, m, n)$, where n is the nonce. Based on our different-ciphertext requirement, we now see that the nonce n must be different every time the same key k and message m are encrypted; otherwise, we're back to square one. We also need to ensure that when decrypting, the nonce is known; otherwise, decryption can't replicate the same steps encryption took. It suffices to just add the nonce as an output, e.g., $\mathrm{E}(k, m, n) = (c, n)$, but in some modes of operation (like counters), the nonce is already known and doesn't need to be output. In some sense, the random scheme above can be seen as implicitly adding a random nonce.

Why is having a unique nonce better than using new keys? Keys and nonces can both be fairly small, so it seems like if we were purely worried about space usage, then there's no real point in also needing to store a nonce (for filesystems) or transmit a nonce (for IPsec). One possible benefit is that the nonce might not have to be as carefully chosen as a key; for example, we could use a counter instead of a uniformly-random value. (In fact, we might not even need to store or transmit the nonce if it is deterministic.) Another is that we may want to secure the key somehow; for example, encrypted filesystems generally associated some passphrase or key file with the encryption key. Now we find it much more convenient to reuse the same key after it's been authenticated, as opposed to having to authenticate a new key for every file.

So far, beyond being different on each encryption, we have no requirements on the nonce; however, we will see that we may need to further restrict the nonce to maintain CPA security in some upcoming schemes. Next, we consider another commonly-used many-time security strategy: cipher block chaining (CBC).

5.3 Cipher block chaining (CBC)

Cipher block chaining also uses randomization to achieve many-time security. For a PRP (E, D) — recall that block ciphers are also PRPs — CBC allows us to create a many-time stream cipher; i.e., we can encrypt arbitrarily-sized messages and re-use the same key for multiple messages.

We will need to split m into blocks, i.e., $m = m[0] \parallel m[1] \parallel \ldots \parallel m[L]$. For example, using AES, the block size is 128, so each chunk will be 128 bits long. For a message that is shorter than a multiple of 128 bits, the end will have to be padded somehow. If it is important to know exactly where the padding begins, one padding scheme is: add a single 1, then as many 0s as required to reach a length that is a multiple of 128. Deciding where the padding begins is done by scanning for the first 1 backwards from the end of the message. We'll see this again for MACs later.

DEFINITION 5.5. For encryption $\mathrm{E}_{CBC}(k, m) = (IV, c)$, the ciphertext is computed as follows:

$$c[i] = \begin{cases} \mathrm{E}(k, IV \oplus m[0]), & i = 0 \\ \mathrm{E}(k, c[i-1] \oplus m[i]), & L \geq i > 0 \end{cases} \tag{5.5}$$

where IV is the *Initialization Vector*, a fancy way of saying a random nonce, and $L + 1$ is the total number of message blocks. See page 9 of the lecture slides for a good pictoral version of this. Inductively, we know that each $c[i]$ is indistinguishable from uniformly random, and so we can produce each successive $c[i+1]$ using something indistinguishable from random input; then only the first ciphertext, $c[0]$, needs some random input. The benefit we gain is that rather than having to output L 128-bit random values, we only need to output one 128-bit random IV. Much like nonces earlier, we still require that each message use a different IV, as otherwise, $\mathrm{E}_{CBC}(k, m)$ will end up producing the same ciphertext when given the same k and m multiple times.

DEFINITION 5.6. Decryption $\mathrm{D}_{CBC}(IV, k, c) = m$ must undo each step:

$$m[i] = \begin{cases} \mathrm{D}(k, c[0]) \oplus IV, & i = 0 \\ \mathrm{D}(k, c[i]) \oplus c[i-1], & L \geq i > 0 \end{cases} \tag{5.6}$$

Page 10 of the lecture slides has a pictoral version of this; however, notice that decryption is not sequential. In fact, decryption is actually parallelizable, unlike encryption. This is because decryption relies purely on the input c, while encryption requires the output of the previous chunk.

THEOREM 5.7. *For small L, if PRP E is secure over $(\mathcal{K}, \mathcal{X})$, then E_{CBC} is CPA-secure over $(\mathcal{K}, \mathcal{X}^L, \mathcal{X}^{L+1})$, and $\varepsilon_{CBC} = 2\varepsilon_{PRP} + \frac{q^2 L^2}{|\mathcal{X}|}$, where q is the total number of messages encrypted and L is the number of blocks in a message.*

We defer the proof of this to external sources. However, we briefly note that the squares in q^2L^2 originate from the "birthday bound" when considering the probability that two randomly chosen values are equal. For two blocks identical blocks in different indices, i.e., $i, j \in [0, L]$ where $m[i] = m[j]$, the resulting ciphertexts should be different. The probability that $c[i]$ and $c[j]$ are different is at least the probability that the XOR'd uniform random inputs (i.e., $c[i-1]$ and $c[j-1]$) are different, which scales with the squared input size due

to the "birthday bound." In this case, the squares are not problematic to this error bound because the denominator $|\mathcal{X}| = 2^{\text{block length}}$; this exponential is asymptotically larger than any polynomial. Essentially, the block length can be considered a security parameter of CBC.

We discussed before that IV must be random (and thus unpredictable). Interestingly, SSL/TLS 1.0 used the final ciphertext block of the previous message as the IV, which led to a security bug, as this is no longer CPA-secure. To see why, let us consider the indistinguishability game for CPA security.

Claim 5.8. *To be CPA-secure, the IV in CBC must be unpredictable.*

Proof. Suppose, for the sake of contradiction, that given one IV, the adversary could somehow predict the next IV. Consider the following adversary:

Game 1		Game 2	
Challenger	Adversary	Challenger	Adversary
$\leftarrow$	0	$\leftarrow$	0
$\rightarrow$	$(IV_1, c_0 = \mathrm{E}(k, IV_1))$	$\rightarrow$	$(IV_1, c_0 = \mathrm{E}(k, IV_1))$
$\leftarrow$	$m_1 = IV_1 \oplus IV_2$	$\leftarrow$	$m_2 \neq m_1$
$\rightarrow$	$(IV_2, c_1 = \mathrm{E}(k, IV_1))$	$\rightarrow$	$(IV_2, c_2 = \mathrm{E}(k, m_2 \oplus IV_2))$

First, the adversary passes in 0 in both games. Because the adversary can predict IV_2 after seeing IV_1, they can pass in $IV_1 \oplus IV_2$ in game 1, while picking some arbitrary message in game 2. The resulting ciphertext in game 1 is $E(k, m_1 \oplus IV_2) = E(k, IV_1 \oplus IV_2 \oplus IV_2) = \mathrm{E}(k, IV_1)$, while the ciphertext in game 2 is some apparently random c_2. However, because $0 \oplus IV_1 = IV_1$, the first ciphertext in game 1 will be equal to the second. The adversary can then trivially distinguish the two games by outputting whether $c_0 = c_1$ (i.e., game 1) or not (i.e., game 2, where they instead have $c_0 \neq c_2$), contradicting indistinguishability. □

An interesting point is that this only breaks the encryption of IVs. That may or may not lead to breaking the underlying E; it depends on the actual underlying scheme of E.

There is another version of CBC where rather than using a uniform random IV, E_{CBC_2} also accepts a unique nonce. This nonce is encrypted using a separate key k_1 and then used as the IV; the nonce is then also output: $\mathrm{E}_{CBC_2}((k_1, k), m, n) = (n, \mathrm{E}_{CBC}(k, m)$ with $IV = \mathrm{E}(k_1, n))$.

5.4 Random-counter (rand-ctr)

Let $\mathrm{F} : \mathcal{K} \times \{0,1\}^n$ be a PRF. Just as in CBC, break m into L blocks of size n. Sample a random $IV \in \{0,1\}^n$.

Definition 5.9. The random-counter many-time cipher is:

$$\mathrm{E}_{RANDCTR}(k, m) = (IV, F(k, IV + i) \oplus m[i]) \qquad \forall i \in [0, L] \tag{5.7}$$

$$\mathrm{D}_{RANDCTR}(IV, k, m) = F(k, IV + i) \oplus m[i] \qquad \forall i \in [0, L] \tag{5.8}$$

This scheme has the notable benefit of being parallelizable, and also only requiring a PRF (as opposed to the more constrained PRP required by CBC). PRFs may even be more efficient than PRPs, because they don't need to be invertible.

Thus far, we have been concentrating on ciphers, on security and on implementations of some ciphers. Another interesting aspect of cryptography is one of message authenticity, in which we are interested in whether or not the messaged was modified on its way to the receiver, whether or not an adversary managed to read it. In the upcoming half of the lecture, we will look at this problem in more detail, and provide a solution.

5.5 Introduction to MAC

So far, we have been looking at notions of security and secrecy, and different ways of achieving these notions of security. Another important problem in cryptography is that of *authenticity*. To understand this, let's say that as usual, Alice would like to send Bob a message. This time, however, Alice does not care whether an adversary is able to *read* the message as much as whether the adversary is able to modify the message without Bob's knowledge of having done so. In other words, Alice just wants to be sure that her actual message has reached Bob, whether or not the adversary has read it.

Encryption does not necessarily help in this aspect, because the adversary may still be able to *modify* the message. Although it is of interest to have a solution that ensures both security and authenticity, the problem of authenticity is orthogonal to the challenge of maintaining security. We will take a look at an approach to tackle the authenticity aspect of this problem.

In the shared key setting, a solution to the problem described above is intuitively described as follows: the sender uses a key to generate and send a 'tag' along with the actual message. The receiver can then apply an algorithm that uses the same shared key, to the tag and the message, to verify that the message has not been tampered with. This is the underlying intuition behind the use of MACs (Message Authentication Codes).

Definition 5.10. A MAC (Message Authentication Code) is a pair of efficient algorithms (S, V), where:

- The algorithm S, called the signing algorithm, accepts as input two parameters k and m, where k is the shared key, and m is the message required to be sent. The output, t, of the algorithm is called a tag.
- V is an algorithm (called the verification algorithm) that takes in parameters k, m and t as defined above, and outputs either *accept* or *reject*.

such that the following conditions are satisfied:

1. CORRECTNESS: V(k, m, t) = *accept* whenever t = S(k, m).
2. SECURITY: An adversary who does not know (or cannot guess) the shared key, is effectively unable to compute (m', tag'), such that V(k, m', tag') = *accept*.

A more general version of the security condition above is one where the attacker can choose the messages as well. More precisely,

- The attacker can demand tags $(t_1, t_2, ..., t_n)$ for messages $(m_1, m_2, ..., m_n)$ adaptively, i.e, can demand the tag for a certain message after receiving and using knowledge about the tag for another message.

- The attacker wins if it can output (m', tag'), where $m' \notin \{m_1, m_2, ..., m_n\}$, $tag' \notin \{t_1, t_2, ..., t_n\}$, such that V(k, m', tag') = *reject*

The next question that arises is: how do we build a MAC? We will look at this in the following section.

5.6 Construction of MAC using PRF

We now turn to constructing a secure MAC using a secure PRF. The reason we turn to a PRF is that it prevents one from guessing the output of the PRF on any input that is different from the inputs that have already been queried. This is precisely what we need for our security condition in the definition of MAC above.

Here is the proposed construction of MAC using a PRF:

- Say k is a PRF key
- Define the signing algorithm also to be a PRF; that is, *S(k, m)* := *PRF(k, m)*.
- V(k, m, tag) = *accept* if and only if *tag = PRF(k, m)*.

Claim 5.11. *The MAC defined above satisfies the correctness property.*

Proof. This follows directly from the definition of the correctness property for a MAC. V(k, m, tag) returns *accept* if and only if tag = *PRF(k, m)*. Thus, this construction of the MAC satisfies the correctness property. □

Claim 5.12. *The MAC defined above satisfies the security property.*

Proof. By PRF security, $\forall\ m' \notin \{m_1, m_2, ..., m_n\}$,

PRF(k, m') $\approx$ Uniform random α-bit string r (α is the length of the PRF's output)

Let $P_1 = Pr$(Adv generates (m', t'), such that t' = PRF(k, m')). This probability is exactly ϵ_{MAC}, or the error in the MAC.

Let $P_2 = Pr$(Adv generates (m', t'), such that $t' \leftarrow$ r). Note that $[P_2 = 2^{-\alpha}]$

.

Now,we know that $P_1 \leq \epsilon_{PRF} + P_2$ where ϵ_{PRF} is the error in the PRF.

Therefore, $\epsilon_{MAC} \leq \epsilon_{PRF} + 2^{-\alpha}$, as required. □

5.7 Examples of MACs

Now that we have defined and given a construction of an MAC, let us look at some examples of MACs.

- AES (Advanced Encryption Standard) is a MAC for messages of size 128 bits.
- For larger messages, we use:
  - CBC-MAC, or a Cipher Blockchaining MAC.
  - HMAC, which is a hash-function based MAC.

Acknowledgement

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

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