Chapter 5

CCA-Security and Authenticated Encryption

In previous chapters we studied two different notions of security for parties communicating over an open communication channel. In Chapter 3 we focused on the goal of *secrecy* against a *passive* adversary who simply eavesdrops on the parties' communication, and showed CPA-secure encryption schemes realizing this goal. In Chapter 4 we explored *integrity* against an *active* adversary who can inject messages on the channel or otherwise tamper with the parties' communication, and described how message authentication codes can be used to achieve this notion. We consider the missing piece—*secrecy* in the presence of an *active* adversary—in Section 5.1, and introduce the notion of relevant notion of CCA-security there. Beginning in Section 5.2, we then consider the natural question of how to construct encryption schemes that achieve both secrecy and integrity *simultaneously*.

5.1 Chosen-Ciphertext Attacks and CCA-Security

We have so far considered encryption schemes secure only against passive (eavesdropping) adversaries. (Even though chosen-plaintext attacks allow an adversary to control what gets encrypted, the adversary in that setting is still limited to passively observing ciphertexts transmitted by the honest parties.) In the previous chapter, we discussed the importance of also defending against *active* attackers who may interfere with or modify the communication between the honest parties, focusing there on the case of message integrity. What might the effect of active attacks be when it comes to *secrecy*?

Consider a scenario in which a sender encrypts a message m and then transmits the resulting ciphertext c. An attacker who can tamper with the communication can modify c to generate another ciphertext c' that is received by the other party. This receiver will then decrypt c' to obtain a message m'. If $m' \neq m$ (and $m' \neq \perp$), this is a violation of integrity. What is of interest to us here, however, is the potential impact on secrecy. In particular, if the attacker learns partial information about m'—say, from subsequent behavior of the receiver—might that reveal information about the original message m?

This type of attack, in which an adversary causes a receiver to decrypt ciphertexts that the adversary generates, is called a *chosen-ciphertext attack*. Chosen-ciphertext attacks are possible, in principle, any time an attacker has the ability to inject traffic on the channel between the sender and receiver. There are many scenarios in which this can occur. (See also the discussion in Section 12.2.3 regarding chosen-ciphertext attacks in the public-key setting.) In the Midway example from Section 3.4.2, for example, US cryptanalysts could have sent encrypted messages containing the fragment `AF` to the Japanese; by monitoring their subsequent behavior (e.g., movement of troops and the like), the US could have learned information about what `AF` meant.

Alternatively, imagine a client sending encrypted messages to a server. If an adversary can impersonate the client and send ciphertexts to the server that appear to originate from the client, the server will decrypt those ciphertexts and the adversary may learn something about the result; for example, the attacker may be able to deduce when a ciphertext decrypts to an *ill-formed* plaintext (e.g., one that is not formatted correctly) based on the server's reaction (e.g., if the server sends an error message). In Section 5.1.1 we describe in detail an attack of exactly this sort where an attacker is able to leverage the information leaked from these decryptions to learn the entire contents of some other encrypted message! Such attacks have been carried out in practice on web servers to learn the contents of encrypted TLS sessions.

5.1.1 Padding-Oracle Attacks

We motivate the importance of security against chosen-ciphertext attacks by showing a real-world example where such attacks can be devastating. We consider a setting in which a client sends messages encrypted using CBC-mode encryption to a server. We assume the attacker can impersonate the client and send ciphertexts of its choice to the server, which the server will then decrypt. We assume further that the attacker can tell when the resulting decrypted messages are valid (in a sense we will define below) or not. Such information is frequently easy to obtain since, for example, the server might request retransmission or terminate a session if it receives a ciphertext that does not decrypt correctly, and either of those events would be detectable by the attacker. The attack has been shown to work in practice on various deployed protocols.

In our discussion of CBC-mode encryption in Section 3.6.3, we only dealt with the case where the message length was a multiple of the block length of the underlying block cipher F. If a message does not satisfy this property, it must be padded before CBC mode is applied; we refer to the result after padding as the *encoded data*. The padding must allow the receiver to unambiguously recover the original message from the encoded data. One popular padding scheme is defined by the PKCS #7 standard, and works as follows. Assume the original message has an integral number of bytes, and let L denote the block length (in bytes) of the block cipher F. Let $b > 0$ denote the

number of bytes that need to be appended to the message in order to make the total length of the resulting encoded data a multiple of the block length. Then we append to the message the integer b (represented in one byte, i.e., two hexadecimal digits) repeated b times. That is, if one byte of padding is needed then the 1-byte string 0x01 (written in hexadecimal) is appended; if four bytes of padding are needed then 0x04040404 is appended; etc. (Note that b is an integer between 1 and L, inclusive—we cannot have $b = 0$ since this would lead to ambiguous padding. Thus, if the original message length is already a multiple of the block length, then $b = L$.) After padding, the encoded data is encrypted using regular CBC-mode encryption.

When decrypting, the server first uses CBC-mode decryption as usual to recover the encoded data, and then checks whether the encoded data is correctly padded. (This is easily done: simply read the value b of the final byte and then verify that the final b bytes of the result all have value b.) If so, the padding is stripped off and the original message returned. Otherwise, the standard procedure is to return a "bad padding" error. This means the server is serving as a "padding oracle" for the adversary: i.e., the adversary can send an arbitrary ciphertext to the server and learn (based on whether a "bad padding" error is returned) whether the underlying encoded data is padded correctly or not. Although this may seem like meaningless information, we show that it enables an adversary to completely recover the original message corresponding to any ciphertext of its choice.

We describe the attack on a 3-block ciphertext for simplicity. Let IV, c_1, c_2 be a ciphertext observed by the attacker, and let m_1, m_2 be the underlying encoded data (unknown to the attacker) that corresponds to a padded message, as discussed above. (Each block is L bytes long.) Note that

$$m_2 = F_k^{-1}(c_2) \oplus c_1, \tag{5.1}$$

where k is the key (which is, of course, not known to the attacker) being used by the honest parties. The second block m_2 ends in $\underbrace{0\text{x}b \cdots 0\text{x}b}_{b \text{ times}}$, where we let 0x$b$ denote the 1-byte representation of some integer b. The key property used in the attack is that certain changes to the ciphertext yield predictable changes in the underlying encoded data after CBC-mode decryption. Specifically, let c_1' be identical to c_1 except for a modification in the final byte, and consider decryption of the modified ciphertext IV, c_1', c_2. This will result in encoded data m_1', m_2' where $m_2' = F_k^{-1}(c_2) \oplus c_1'$. Comparing to Equation (5.1) we see that m_2' will be identical to m_2 except for a modification in the final byte. (The value of m_1' is unpredictable, but this will not adversely affect the attack.) Similarly, if c_1' is the same as c_1 except for a change in its ith byte, then decryption of IV, c_1', c_2 will result in m_1', m_2' where m_2' is the same as m_2 except for a change in its ith byte. More generally, if $c_1' = c_1 \oplus \Delta$ for any string Δ, then decryption of IV, c_1', c_2 yields m_1', m_2' where $m_2' = m_2 \oplus \Delta$. The upshot is that the attacker can exercise significant control over the final block of the encoded data.

As a warmup, let us see how the adversary can exploit this to learn b, the amount of padding. (This reveals the length of the original message.) Recall that upon decryption, the server looks at the value b of the final byte of the encoded data, and then verifies that the final b bytes all have the same value. The attacker begins by modifying the first byte of c_1 and sending the resulting ciphertext IV, c_1', c_2 to the server. If decryption fails (i.e., the server returns an error) then it must be the case that the server is checking all L bytes of m_2', and therefore $b = L$! Otherwise, the attacker learns that $b < L$, and it can then repeat the process with the second byte, and so on. The left-most modified byte for which decryption fails reveals exactly the left-most byte being checked by the server, and so reveals exactly b.

With b known, the attacker can proceed to learn the bytes of the message one-by-one. We illustrate the idea for the final byte of the message, which we denote by M. The attacker knows that m_2 ends in $0\text{x}M0\text{x}b \cdots 0\text{x}b$ (with $0\text{x}b$ repeated b times) and wishes to learn M. For $0 \le i < 2^8$ define

$$\begin{aligned}\Delta_i \stackrel{\text{def}}{=} 0\text{x}00 \cdots 0\text{x}00\ 0\text{x}i\ \overbrace{0\text{x}(b+1) \cdots 0\text{x}(b+1)}^{b \text{ times}} \\ \oplus\ 0\text{x}00 \cdots 0\text{x}00\ 0\text{x}00\ \overbrace{0\text{x}b \quad \cdots \quad 0\text{x}b}^{b \text{ times}};\end{aligned}$$

i.e., the final $b+1$ bytes of Δ_i contain the integer i (in hexadecimal) followed by the value $(b+1) \oplus b$ (in hexadecimal) repeated b times. If the attacker submits the ciphertext $IV, c_1 \oplus \Delta_i, c_2$ to the server then, after CBC-mode decryption, the final $b+1$ bytes of the resulting encoded data will equal $0\text{x}(M \oplus i)0\text{x}(b+1) \cdots 0\text{x}(b+1)$ (with $0\text{x}(b+1)$ repeated b times), and decryption will fail unless $0\text{x}(M \oplus i) = 0\text{x}(b+1)$. The attacker tries at most 2^8 values $\Delta_0, \ldots, \Delta_{2^8-1}$ until decryption succeeds for some Δ_i, at which point it learns that $M = 0\text{x}(b+1) \oplus 0\text{x}i$. We leave it as an exercise to extend this attack so as to learn the next byte of m_2, as well as all of m_1.

A padding-oracle attack on CAPTCHAs. We have already mentioned that padding-oracle attacks have been carried out on encrypted web traffic. Here we give a second example.

A CAPTCHA is a distorted image of, say, an English word that is easy for humans to read, but hard for a computer to process. CAPTCHAs are used in order to ensure that a human user—and not some automated software—is interacting with a webpage.

CAPTCHAs can be provided as a separate service run on an independent server. To see how this works, we denote a web server by $\mathcal{S}_W$, a CAPTCHA server by $\mathcal{S}_C$, and a user by $\mathcal{U}$. When $\mathcal{U}$ loads a webpage served by $\mathcal{S}_W$, the following events occur: $\mathcal{S}_W$ encrypts a random English word w using a key k that was initially shared between $\mathcal{S}_W$ and $\mathcal{S}_C$, and sends the resulting ciphertext (along with the webpage) to the user. $\mathcal{U}$ forwards the ciphertext to $\mathcal{S}_C$, who decrypts it, obtains w, and renders a distorted image of w (i.e.,

the CAPTCHA) to $\mathcal{U}$. Finally, $\mathcal{U}$ sends w back to $\mathcal{S}_W$ for verification. Note that $\mathcal{S}_C$ decrypts any ciphertext it receives from $\mathcal{U}$ and will issue a "bad padding" error message if decryption fails, as described earlier. This presents $\mathcal{U}$ with an opportunity to carry out a padding-oracle attack, and thus to solve the CAPTCHA (i.e., to determine w) automatically *without any human involvement*, rendering the CAPTCHA ineffective.

5.1.2 Defining CCA-Security

What would it mean for an encryption scheme to be secure against chosen-ciphertext attacks? As usual, to define an appropriate notion of security we need to define two things: the assumed abilities of the attacker, and what constitutes a successful attack. For the latter, we will follow the approach we have taken in several previous definitions of security for encryption (e.g., in Definitions 3.8 and 3.21): namely, we give the attacker a challenge ciphertext c that is generated by encrypting one of two possible messages m_0, m_1 (each chosen with equal probability), and consider the scheme to be broken if the attacker can determine which message was encrypted with probability significantly better than $1/2$.

How should we model the attacker's capabilities in the present setting? Now, the adversary should have the ability not only to obtain the encryption of messages of its choice (as in a chosen-plaintext attack), but also to obtain the *decryption* of ciphertexts of its choice (with one exception discussed later). Formally, we give the adversary access to a *decryption oracle* $\mathsf{Dec}_k(\cdot)$ in addition to an encryption oracle $\mathsf{Enc}_k(\cdot)$. We present the formal definition and defer further discussion.

Consider the following experiment for any private-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, adversary $\mathcal{A}$, and value n for the security parameter.

The CCA indistinguishability experiment $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$**:**

1. *A key k is generated by running* $\mathsf{Gen}(1^n)$.
2. *$\mathcal{A}$ is given input 1^n and oracle access to* $\mathsf{Enc}_k(\cdot)$ *and* $\mathsf{Dec}_k(\cdot)$. *It outputs a pair of equal-length messages m_0, m_1.*
3. *A uniform bit $b \in \{0,1\}$ is chosen, and then a* challenge ciphertext $c \leftarrow \mathsf{Enc}_k(m_b)$ *is computed and given to $\mathcal{A}$.*
4. *The adversary $\mathcal{A}$ continues to have oracle access to* $\mathsf{Enc}_k(\cdot)$ *and* $\mathsf{Dec}_k(\cdot)$*, but is not allowed to query the latter on the challenge ciphertext itself. Eventually, $\mathcal{A}$ outputs a bit b'.*
5. *The output of the experiment is* 1 *if $b' = b$, and* 0 *otherwise. If the output of the experiment is* 1*, we say that $\mathcal{A}$* succeeds.

DEFINITION 5.1 *A private-key encryption scheme Π has* indistinguishable encryptions under a chosen-ciphertext attack*, or is* CCA-secure*, if for all*

probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function negl *such that:*

$$\Pr[\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n),$$

where the probability is taken over all randomness used in the experiment.

For completeness, we remark that the natural analogue of Theorem 3.23 holds for CCA-security as well—namely, if a scheme has indistinguishable encryptions under a chosen-ciphertext attack then it has indistinguishable *multiple* encryptions under a chosen-ciphertext attack, defined appropriately.

Discussion. In the experiment considered above, the adversary is given access to a decryption oracle that returns the *entire result* of decrypting a ciphertext provided by the attacker. In general, this might be much more information than what is available to an attacker in the real world; for example, in the padding-oracle scenario described earlier, the attacker only learns whether decryption results in an error or not. As usual, however, we want to make cryptographic definitions as strong as possible so they are broadly applicable. Since we don't know what information an attacker might be able to learn when a ciphertext it sends is decrypted by a receiver, we make the worst-case assumption that the attacker learns *everything*.

There is one caveat. In the experiment, the adversary is allowed to submit any ciphertexts of its choice to the decryption oracle *except* that it may not request decryption of the challenge ciphertext itself. This restriction is clearly necessary or else there is no hope for any encryption scheme to satisfy the definition. Even with this restriction in place, the definition provides meaningful security. In particular, note that in the context of a padding-oracle attack the attacker does not learn anything by getting the receiver to decrypt the challenge ciphertext (since the attacker knows that it will not cause an error), and so a CCA-secure scheme would not be vulnerable to that attack.

Insecurity of the schemes we have studied. None of the encryption schemes we have seen thus far is CCA-secure. We demonstrate this for Construction 3.28, where encryption of a message m takes the form $\langle r, F_k(r) \oplus m \rangle$. Consider an adversary $\mathcal{A}$ running in the CCA indistinguishability experiment who chooses $m_0 = 0^n$ and $m_1 = 1^n$. Then, upon receiving a ciphertext $c = \langle r, s \rangle$, the adversary flips the first bit of s and asks for a decryption of the resulting ciphertext c'. Since $c' \neq c$, this query is allowed and the decryption oracle answers with either 10^{n-1} (in which case it is clear that $b = 0$) or 01^{n-1} (in which case $b = 1$). This example demonstrates that CCA-security is quite stringent. Any encryption scheme that allows ciphertexts to be "manipulated" in a controlled way cannot be CCA-secure. Thus, CCA-security implies a very important property called *non-malleability*. Loosely speaking, a non-malleable encryption scheme has the property that if the adversary modifies a given ciphertext, the result decrypts to a plaintext that bears no

relation to the original one. This is a very useful property for encryption schemes used in complex cryptographic protocols.

5.2 Authenticated Encryption

CCA-security is extremely important, but is subsumed by an even stronger notion of security we introduce here. Until now, we have considered how to obtain secrecy (using encryption) and integrity (using message authentication codes) separately. The aim of *authenticated encryption*, defined below, is to achieve both goals *simultaneously*. It is best practice to *always ensure secrecy and integrity by default* in the private-key setting. Indeed, in many applications where secrecy is required it turns out that integrity is essential also. Moreover, a lack of integrity can sometimes lead to a breach of secrecy, as illustrated in the previous section.

5.2.1 Defining Authenticated Encryption

We begin, as usual, by defining precisely what we wish to achieve. One way to proceed is to define secrecy and integrity separately. Since we are explicitly concerned with an active adversary here, the natural notion of secrecy is CCA-security. The natural way to define integrity for encryption is via an analogue of the notion of existential unforgeability under an adaptive chosen-message attack that we considered for MACs. (We need a new definition because the syntax of an encryption scheme does not match the syntax of a MAC.) Consider the following experiment defined for a private-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, adversary $\mathcal{A}$, and value n for the security parameter:

The unforgeable encryption experiment $\mathsf{Enc\text{-}Forge}_{\mathcal{A},\Pi}(n)$**:**

1. *A key k is generated by running* $\mathsf{Gen}(1^n)$.
2. *The adversary $\mathcal{A}$ is given input* 1^n *and access to an encryption oracle* $\mathsf{Enc}_k(\cdot)$. *The adversary eventually outputs a ciphertext c. Let* $m := \mathsf{Dec}_k(c)$ *and let $\mathcal{Q}$ denote the set of all queries that $\mathcal{A}$ submitted to its oracle.*
3. $\mathcal{A}$ succeeds *if and only if (1)* $m \neq \perp$ *and (2)* $m \notin \mathcal{Q}$. *In that case the output of the experiment is defined to be* 1.

DEFINITION 5.2 *A private-key encryption scheme* Π *is* unforgeable *if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there is a negligible function* negl *such that:*

$$\Pr[\mathsf{Enc\text{-}Forge}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}(n).$$

We may now define an *authenticated encryption* scheme.

DEFINITION 5.3 *A private-key encryption scheme is an* authenticated encryption (AE) scheme *if it is CCA-secure and unforgeable.*

It is also possible to capture both the above requirements in a definition involving a single experiment. The experiment is somewhat different from previous experiments we have considered, so we provide some motivation before giving the details. The idea is to consider two different scenarios, and require that they be indistinguishable to an attacker. In the first scenario, which can be viewed as corresponding to the real-world context in which the adversary operates, the attacker is given access to both an encryption oracle and a decryption oracle. In the second case, which can be viewed as corresponding to an "ideal" scenario, these two oracles are changed as follows:

- In place of an encryption oracle, the attacker is given access to an oracle that encrypts a 0-string of the correct length. Formally, the attacker is given access to an oracle $\mathsf{Enc}^0_k(\cdot)$ where $\mathsf{Enc}^0_k(m) = \mathsf{Enc}_k(0^{|m|})$. I.e., when requesting an encryption of m, the attacker is instead given an encryption of a 0-string of the same length as m.

- In place of a decryption oracle, the attacker is given access to an oracle $\mathsf{Dec}_\perp(\cdot)$ that always returns the error symbol $\perp$.

If an attacker cannot distinguish the first scenario from the second, then this means (1) any new ciphertexts the attacker generates in the real world will be invalid (i.e., will generate an error upon decryption). This not only implies a strong form of integrity, but also makes chosen-ciphertext attacks useless. Moreover, (2) the attacker cannot distinguish a real encryption oracle from an oracle that always encrypts 0s, which implies secrecy.

Formally, for a private-key encryption scheme Π, adversary $\mathcal{A}$, and value n for the security parameter, define the following experiment:

The authenticated-encryption experiment $\mathsf{PrivK}^{\mathsf{ae}}_{\mathcal{A},\Pi}(n)$**:**

1. *A key k is generated by running* $\mathsf{Gen}(1^n)$.
2. *A uniform bit $b \in \{0,1\}$ is chosen.*
3. *The adversary $\mathcal{A}$ is given input 1^n and access to two oracles:*
   (a) *If $b = 0$, then $\mathcal{A}$ is given access to* $\mathsf{Enc}_k(\cdot)$ *and* $\mathsf{Dec}_k(\cdot)$.
   (b) *If $b = 1$, then $\mathcal{A}$ is given access to* $\mathsf{Enc}^0_k(\cdot)$ *and* $\mathsf{Dec}_\perp(\cdot)$.

   $\mathcal{A}$ is not allowed to query a ciphertext c to its second oracle that it previously received as the response from its first oracle.
4. *The adversary outputs a bit b'.*

5. *The output of the experiment is defined to be* 1 *if* $b' = b$*, and* 0 *otherwise. In the former case, we say that* $\mathcal{A}$ succeeds.

In the experiment, the attacker is not allowed to submit ciphertexts to the decryption oracle that it received from its encryption oracle, since this would lead to a trivial way to distinguish the two scenarios. We remark that in the "real" case (i.e., when $b = 0$) the attacker already knows the decryption of those ciphertexts, so there is not much point in making such queries, anyway.

DEFINITION 5.4 *A private-key encryption scheme is an* authenticated encryption (AE) scheme *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *there is a negligible function* negl *such that*

$$\Pr\left[\mathsf{PrivK}^{\mathsf{ae}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n).$$

We have given two definitions of authenticated encryption. Fortunately, the definitions are equivalent:

THEOREM 5.5 *A private-key encryption scheme satisfies Definition 5.3 if and only if it satisfies Definition 5.4.*

Authenticated encryption with associated data. Often, a message m requires both secrecy and integrity but various *associated data* (e.g., header information) sent along with the message requires integrity only. While it is possible to simply concatenate the message and the associated data (in some way that allows for unambiguous parsing) and then use an AE scheme to encrypt them both, better efficiency can be achieved by providing the associated data with integrity protection only. We omit further details, but note that AE schemes with support for associated data are called *authenticated encryption with associated data (AEAD) schemes* in the literature.

5.2.2 CCA Security vs. Authenticated Encryption

It follows directly from Definition 5.3 that any authenticated encryption scheme is also CCA-secure. The converse, however, is not true, and there are private-key encryption schemes that are CCA-secure but that are *not* authenticated encryption schemes. You are asked to prove this in Exercise 5.9.

One can imagine applications where CCA-security is needed but authenticated encryption is not. One example might be when private-key encryption is used for *key transport*. As a concrete example, say a server gives a tamper-proof hardware token to a user, where the token stores a long-term key k. The server can share a fresh, short-term key k' with the token (that will remain unknown to the user) by giving the user $\mathsf{Enc}_k(k')$; the user is supposed to give this ciphertext to the token, which will decrypt it to obtain k'. CCA-security

is necessary here because chosen-ciphertext attacks can be easily carried out by the user in this context. On the other hand, not much harm is done if the user can generate a valid ciphertext that causes the token to use some arbitrary key k'' that is uncorrelated with k'. (Of course, this depends on what the token does with this short-term key.)

Notwithstanding the above, most applications of private-key encryption in the presence of an active adversary do require integrity. Fortunately, most natural constructions of CCA-secure encryption schemes satisfy the stronger definition of authenticated encryption, anyway. Put differently, there is no reason to ever use a CCA-secure scheme that is *not* also an authenticated encryption scheme, simply because we don't have any constructions of the former that are more efficient than constructions of the latter.

From a *conceptual* point of view, however, the notions of CCA-security and authenticated encryption are distinct. With regard to CCA-security we are not interested in message integrity *per se*; rather, we wish to ensure *secrecy* even against an active adversary who can interfere with the communication from sender to receiver. In contrast, with regard to authenticated encryption we are explicitly interested in the twin goals of *secrecy* and *integrity*.

5.3 Authenticated Encryption Schemes

5.3.1 Generic Constructions

It is tempting to think that any reasonable combination of a CPA-secure encryption scheme and a secure message authentication code should result in an authenticated encryption scheme. In this section we show that this is not the case. This demonstrates that even secure cryptographic tools can be combined in such a way that the result is insecure, and highlights once again the importance of definitions and proofs of security. On the positive side, we show how encryption and message authentication *can* be combined properly to achieve joint secrecy and integrity.

Throughout, let $\Pi_E = (\mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure encryption scheme and let $\Pi_M = (\mathsf{Mac}, \mathsf{Vrfy})$ denote a strongly secure MAC, where key generation in both schemes simply involves choosing a uniform n-bit key. There are three natural approaches to combining encryption and message authentication using independent[1] keys k_E and k_M for Π_E and Π_M, respectively:

1. *Encrypt-and-authenticate:* In this approach, encryption and message authentication are computed independently in parallel. That is, given

[1] *Independent* cryptographic keys should always be used when different schemes are combined. We return to this point at the end of this section.

a message m, the sender transmits the ciphertext $\langle c, t \rangle$ where:

$$c \leftarrow \mathsf{Enc}_{k_E}(m) \quad \text{and} \quad t \leftarrow \mathsf{Mac}_{k_M}(m).$$

The receiver decrypts c to recover m; assuming no error occurred, it then verifies the tag t. If $\mathsf{Vrfy}_{k_M}(m, t) = 1$, the receiver outputs m; otherwise, it outputs an error.

2. *Authenticate-then-encrypt:* Here a tag t is first computed, and then the message and tag are encrypted together. That is, given a message m, the sender transmits the ciphertext c computed as:

$$t \leftarrow \mathsf{Mac}_{k_M}(m) \quad \text{and} \quad c \leftarrow \mathsf{Enc}_{k_E}(m \| t).$$

The receiver decrypts c to obtain $m \| t$; assuming no error occurred, it then verifies the tag t. As before, if $\mathsf{Vrfy}_{k_M}(m, t) = 1$ the receiver outputs m; otherwise, it outputs an error.

3. *Encrypt-then-authenticate:* In this case, the message m is first encrypted and then a tag is computed over the result. That is, the ciphertext is the pair $\langle c, t \rangle$ where:

$$c \leftarrow \mathsf{Enc}_{k_E}(m) \quad \text{and} \quad t \leftarrow \mathsf{Mac}_{k_M}(c).$$

(See also Construction 5.6.) If $\mathsf{Vrfy}_{k_M}(c, t) = 1$, then the receiver decrypts c and outputs the result; otherwise, it outputs an error.

We analyze each of the above approaches when they are instantiated with "generic" secure components, i.e., when instantiated with an *arbitrary* CPA-secure encryption scheme and an *arbitrary* strongly secure MAC (cf. Definition 4.3). We are looking for an approach that provides joint secrecy and integrity when using *any* (secure) components, and so reject as "unsafe" any approach for which this is not the case. This reduces the likelihood of implementation flaws. Specifically, an approach might be implemented by making calls to an "encryption subroutine" and a "message authentication subroutine," and the implementation of those subroutines may be changed at some later point in time. (This commonly occurs when cryptographic libraries are updated, or when standards are modified.) An approach whose security depends on the details of how its underlying components are implemented—rather than on the security they provide—is therefore dangerous.

We stress that if an approach is rejected this does not mean that it is insecure for all possible instantiations of the components; it does, however, mean that any instantiation of the approach must be carefully analyzed and proven secure before it is used.

Encrypt-and-authenticate. Recall that in this approach encryption and message authentication are carried out independently. Given a message m, the ciphertext is $\langle c, t \rangle$ where $c \leftarrow \mathsf{Enc}_{k_E}(m)$ and $t \leftarrow \mathsf{Mac}_{k_M}(m)$. This approach

is problematic since it may not achieve even the most basic level of secrecy. To see this, note that even a strongly secure MAC does not guarantee *any* secrecy and so it is possible for the tag t to leak information about m to an eavesdropper. (As a trivial example, consider a strongly secure MAC where the first bit of the tag is always equal to the first bit of the message.) So the encrypt-and-authenticate approach may yield a scheme that is not even EAV-secure.

The encrypt-and-authenticate approach is insecure against chosen-plaintext attacks even when instantiated with standard components (as opposed to the somewhat contrived example in the previous paragraph). In particular, if a *deterministic* MAC like CBC-MAC is used, then the tag computed on a message (for some fixed key k_M) is the same every time. This allows an eavesdropper to identify when the same message is sent twice, something that is not possible for a CPA-secure scheme. Many MACs used in practice are deterministic, so this represents a real concern.

Authenticate-then-encrypt. Here, a tag $t \leftarrow \mathsf{Mac}_{k_M}(m)$ is first computed; then $m\|t$ is encrypted and the ciphertext $c \leftarrow \mathsf{Enc}_{k_E}(m\|t)$ is transmitted. This combination also does not necessarily yield an authenticated encryption scheme. We have already encountered a CPA-secure encryption scheme for which this approach is insecure: the CBC-mode-with-padding scheme discussed in Section 5.1.1. (We assume in what follows that the reader is familiar with that section.) Recall that this scheme works by first padding the plaintext (which in our case will be $m\|t$) in a specific way so the result is a multiple of the block length, and then encrypting the result using CBC mode. During decryption, if an error in the padding is detected after performing the CBC-mode decryption, then a "bad padding" error is returned. With regard to the authenticate-then-encrypt approach, this means there are now *two* sources of potential decryption failure: the padding may be incorrect, or the tag may not verify. Schematically, the decryption algorithm Dec' in the combined scheme works as follows:

$\mathsf{Dec}'_{k_E,k_M}(c)$:

1. Compute $\tilde{m} := \mathsf{Dec}_{k_E}(c)$. If an error in the padding is detected (i.e., $\tilde{m} = \perp$), return "bad padding" and stop.
2. Otherwise, parse $\tilde{m}$ as $m\|t$. If $\mathsf{Vrfy}_{k_M}(m,t) = 1$ return m; else, output "authentication failure."

Assuming the attacker can distinguish between the two error messages, the attacker can apply the chosen-ciphertext attack described in Section 5.1.1 to recover the entire original plaintext from a given ciphertext. (This is due to the fact that the padding-oracle attack shown in Section 5.1.1 relies only on the ability to learn whether or not there was a padding error, something that is revealed by Dec'.) This type of attack has been carried out successfully

in the real world in various settings, e.g., in configurations of TLS that used authenticate-then-encrypt.

One way to fix the above would be to ensure that only a *single* error message is returned, regardless of the source of decryption failure. This is an unsatisfying solution for several reasons: (1) there may be legitimate reasons (e.g., usability, debugging) to have multiple error messages; (2) forcing the error messages to be the same means that the combination is no longer truly generic, i.e., it requires the implementer of the authenticate-then-encrypt approach to be aware of what error messages are returned by the underlying CPA-secure encryption scheme; (3) most of all, it is extraordinarily hard to ensure that the different errors cannot be distinguished since, e.g., even a difference in the *time* to return each of these errors may allow an adversary to distinguish between them (cf. our earlier discussion of timing attacks at the end of Section 4.2). Some versions of TLS tried using only a single error message with the authenticate-then-encrypt approach, but a padding-oracle attack was still successfully carried out using small differences in timing.

Finally, we note that there are other counterexamples (that do not rely on distinguishing between different errors) showing that authenticate-then-encrypt does not necessarily provide authenticated encryption.

CONSTRUCTION 5.6

Let $\Pi_E = (\mathsf{Enc}, \mathsf{Dec})$ be a private-key encryption scheme and let $\Pi_M = (\mathsf{Mac}, \mathsf{Vrfy})$ be a message authentication code, where in each case key generation is done by simply choosing a uniform n-bit key. Define a private-key encryption scheme $(\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ as follows:

- Gen': on input 1^n, choose independent, uniform $k_E, k_M \in \{0,1\}^n$ and output the key (k_E, k_M).
- Enc': on input a key (k_E, k_M) and a plaintext message m, compute $c \leftarrow \mathsf{Enc}_{k_E}(m)$ and $t \leftarrow \mathsf{Mac}_{k_M}(c)$. Output the ciphertext $\langle c, t \rangle$.
- Dec': on input a key (k_E, k_M) and a ciphertext $\langle c, t \rangle$, first check if $\mathsf{Vrfy}_{k_M}(c, t) \stackrel{?}{=} 1$. If yes, output $\mathsf{Dec}_{k_E}(c)$; if no, output $\perp$.

The encrypt-then-authenticate approach.

Encrypt-then-authenticate. In this approach, the message is first encrypted and then a MAC is computed over the result. That is, the ciphertext is now the pair $\langle c, t \rangle$ where

$$c \leftarrow \mathsf{Enc}_{k_E}(m) \quad \text{and} \quad t \leftarrow \mathsf{Mac}_{k_M}(c).$$

Decryption of $\langle c, t \rangle$ outputs an error if $\mathsf{Vrfy}_{k_M}(c, t) \neq 1$, and otherwise outputs $\mathsf{Dec}_{k_E}(c)$. See Construction 5.6 for a formal description.

This approach *is* sound. As intuition for why, say a ciphertext $\langle c, t\rangle$ is *valid* if t is a valid tag on c. Strong security of the MAC ensures that an adversary will be unable to generate *any* valid ciphertext that it did not receive from its encryption oracle. This immediately implies that Construction 5.6 is unforgeable. Moreover, it effectively renders the decryption oracle useless: for every ciphertext $\langle c, t\rangle$ the adversary submits to its decryption oracle, the adversary either already knows the decryption (if it received $\langle c, t\rangle$ from its encryption oracle) or will receive an error. (Observe also that the tag is verified before decryption takes place; thus, errors during decryption cannot leak anything about the plaintext, in contrast to the padding-oracle attack we saw against the authenticate-then-encrypt approach.) Therefore, CCA-security of the combined scheme reduces to CPA-security of Π_E.

THEOREM 5.7 *Let Π_E be a CPA-secure private-key encryption scheme, and let Π_M be a strongly secure message authentication code. Then Construction 5.6 is an authenticated encryption scheme.*

PROOF We show that the scheme Π resulting from Construction 5.6 is unforgeable, and that it is CCA-secure. (See Definition 5.3.) Toward this end, we first show that strong security of Π_M implies that (except with negligible probability) any "new" ciphertexts an adversary submits to its decryption oracle will result in an error. This immediately implies unforgeability. (In fact, it is stronger than unforgeability.) It also renders the decryption oracle useless, and allows us to reduce CCA-security of Π to CPA-security of Π_E.

In more detail, let $\mathcal{A}$ be a PPT adversary attacking Construction 5.6 in a chosen-ciphertext attack (cf. Definition 5.1). Say a ciphertext that $\mathcal{A}$ submits to its decryption oracle is *new* if $\mathcal{A}$ did not receive it from its encryption oracle or as the challenge ciphertext. A ciphertext $\langle c, t\rangle$ is *valid* (with respect to the secret key (k_E, k_M) chosen as part of the experiment) if $\mathsf{Vrfy}_{k_M}(c, t) = 1$. Let $\mathsf{ValidQuery}$ be the event that $\mathcal{A}$ submits a new, valid ciphertext to its decryption oracle. We prove:

CLAIM 5.8 $\Pr[\mathsf{ValidQuery}]$ *is negligible.*

PROOF Intuitively, this is because if $\mathsf{ValidQuery}$ occurs then the adversary has forged a new, valid pair (c, t) in the $\mathsf{Mac\text{-}sforge}$ experiment. Let $q(\cdot)$ be a polynomial upper bound on the number of decryption-oracle queries made by $\mathcal{A}$, and consider the following adversary $\mathcal{A}_M$ attacking the message authentication code Π_M:

Adversary $\mathcal{A}_M$:
$\mathcal{A}_M$ is given input 1^n and has access to a MAC oracle $\mathsf{Mac}_{k_M}(\cdot)$.

1. Choose uniform $k_E \in \{0,1\}^n$ and $i \in \{1, \ldots, q(n)\}$.

2. Run $\mathcal{A}$ on input 1^n. When $\mathcal{A}$ makes an encryption-oracle query for the message m, answer it as follows:
   (i) Compute $c \leftarrow \mathsf{Enc}_{k_E}(m)$.
   (ii) Query c to the MAC oracle and receive t in response. Return $\langle c, t\rangle$ to $\mathcal{A}$.

   The challenge ciphertext is prepared in the exact same way (with a uniform bit $b \in \{0,1\}$ chosen to select the message m_b that gets encrypted).

   When $\mathcal{A}$ makes a decryption-oracle query for the ciphertext $\langle c, t\rangle$, answer it as follows: If this is the ith decryption-oracle query, output (c, t) and halt. Otherwise:
   (i) If $\langle c, t\rangle$ was a response to a previous encryption-oracle query for a message m, return m.
   (ii) Otherwise, return $\perp$.

In essence, $\mathcal{A}_M$ is "guessing" that the ith decryption-oracle query of $\mathcal{A}$ is the first new, valid query $\mathcal{A}$ makes. In that case, $\mathcal{A}_M$ indeed outputs a new, valid message/tag pair (c, t).

Clearly $\mathcal{A}_M$ runs in polynomial time. We now analyze the probability that $\mathcal{A}_M$ outputs a new, valid message/tag pair. The key point is that the view of $\mathcal{A}$ when run as a subroutine by $\mathcal{A}_M$ is distributed identically to the view of $\mathcal{A}$ in experiment $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$ *until event* $\mathsf{ValidQuery}$ *occurs*. To see this, note that responses to the encryption-oracle queries of $\mathcal{A}$ (as well as the challenge ciphertext) are simulated perfectly by $\mathcal{A}_M$. As for the decryption-oracle queries of $\mathcal{A}$, until $\mathsf{ValidQuery}$ occurs these are all simulated properly. In case (i) this is obvious. As for case (ii), if the ciphertext $\langle c, t\rangle$ submitted to the decryption oracle is new, then as long as $\mathsf{ValidQuery}$ has not yet occurred the correct answer to that query is indeed $\perp$. (Recall also that $\mathcal{A}$ is disallowed from submitting the challenge ciphertext to the decryption oracle.)

Because the view of $\mathcal{A}$ when run as a subroutine by $\mathcal{A}_M$ is distributed identically to the view of $\mathcal{A}$ in experiment $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$ until event $\mathsf{ValidQuery}$ occurs, the probability of event $\mathsf{ValidQuery}$ in experiment $\mathsf{Mac\text{-}sforge}_{\mathcal{A}_M,\Pi_M}(n)$ is the same as the probability of that event in experiment $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$.

If $\mathcal{A}_M$ correctly guesses the first index i for which $\mathsf{ValidQuery}$ occurs, $\mathcal{A}_M$ outputs (c, t) for which $\mathsf{Vrfy}_{k_M}(c, t) = 1$ (since $\langle c, t\rangle$ is valid) and for which it was never given tag t in response to the query $\mathsf{Mac}_{k_M}(c)$ (since $\langle c, t\rangle$ is new). In this case, then, $\mathcal{A}_M$ succeeds in experiment $\mathsf{Mac\text{-}sforge}_{\mathcal{A}_M,\Pi_M}(n)$. The probability that $\mathcal{A}_M$ guesses i correctly is $1/q(n)$. Therefore

$$\Pr[\mathsf{Mac\text{-}sforge}_{\mathcal{A}_M,\Pi_M}(n) = 1] \geq \Pr[\mathsf{ValidQuery}]/q(n).$$

Since Π_M is a strongly secure MAC and q is polynomial, we conclude that $\Pr[\mathsf{ValidQuery}]$ is negligible. ∎

We use Claim 5.8 to prove security of Π. The easier step is to prove that Π is unforgeable. This follows immediately from the claim, and so we just provide informal reasoning. Observe first that the adversary in the unforgeable encryption experiment is a restricted version of the adversary in the chosen-ciphertext experiment. (In the former, the adversary only has access to an encryption oracle.) An attacker succeeds in the unforgeable encryption experiment only if it outputs a ciphertext $\langle c, t\rangle$ that is valid and new. But Claim 5.8 shows precisely that the probability of doing so is negligible.

It is slightly more involved to prove that Π is CCA-secure. Let $\mathcal{A}$ again be a probabilistic polynomial-time adversary attacking Π in a chosen-ciphertext attack. We have

$$\begin{aligned}&\Pr[\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n) = 1]\\ &\quad \leq \Pr[\mathsf{ValidQuery}] + \Pr[\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n) = 1 \wedge \overline{\mathsf{ValidQuery}}]. \qquad (5.2)\end{aligned}$$

We have already shown that $\Pr[\mathsf{ValidQuery}]$ is negligible. We show next that there is a negligible function negl such that

$$\Pr[\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n) = 1 \wedge \overline{\mathsf{ValidQuery}}] \leq \frac{1}{2} + \mathsf{negl}(n).$$

To prove this, we rely on CPA-security of Π_E. Consider the following adversary $\mathcal{A}_E$ attacking Π_E in a chosen-plaintext attack:

Adversary $\mathcal{A}_E$:
$\mathcal{A}_E$ is given input 1^n and has access to $\mathsf{Enc}_{k_E}(\cdot)$.

1. Choose uniform $k_M \in \{0,1\}^n$.
2. Run $\mathcal{A}$ on input 1^n. When $\mathcal{A}$ makes an encryption-oracle query for the message m, answer it as follows:
   (i) Query m to $\mathsf{Enc}_{k_E}(\cdot)$ and receive c in response.
   (ii) Compute $t \leftarrow \mathsf{Mac}_{k_M}(c)$ and return $\langle c, t\rangle$ to $\mathcal{A}$.

   When $\mathcal{A}$ makes a decryption-oracle query for the ciphertext $\langle c, t\rangle$, answer it as follows: If $\langle c, t\rangle$ was a response to a previous encryption-oracle query for a message m, return m. Otherwise, return $\perp$.
3. When $\mathcal{A}$ outputs messages (m_0, m_1), output those same messages and receive a challenge ciphertext c in response. Compute $t \leftarrow \mathsf{Mac}_{k_M}(c)$, and return $\langle c, t\rangle$ to $\mathcal{A}$ as the challenge ciphertext. Continue answering $\mathcal{A}$'s oracle queries as above.
4. Output the same bit b' that is output by $\mathcal{A}$.

Notice that $\mathcal{A}_E$ does not need a decryption oracle because it simply assumes that any decryption query by $\mathcal{A}$ that was not the result of a previous encryption-oracle query is invalid.

Clearly, $\mathcal{A}_E$ runs in probabilistic polynomial time. Furthermore, the view of $\mathcal{A}$ when run as a subroutine by $\mathcal{A}_E$ is distributed identically to the view of $\mathcal{A}$ in experiment $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$ *as long as event* $\mathsf{ValidQuery}$ *never occurs*. Therefore, the probability that $\mathcal{A}_E$ succeeds is at least the probability that $\mathcal{A}$ succeeds and $\mathsf{ValidQuery}$ does not occur; i.e.,

$$\begin{aligned}\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A}_E,\Pi_E}(n)=1] &\geq \Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A}_E,\Pi_E}(n)=1 \wedge \overline{\mathsf{ValidQuery}}]\\ &= \Pr[\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)=1 \wedge \overline{\mathsf{ValidQuery}}].\end{aligned}$$

Since Π_E is CPA-secure, there exists a negligible function negl such that $\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A}_E,\Pi_E}(n)=1] \leq \frac{1}{2}+\mathsf{negl}(n)$. Together with Equation (5.2), this proves that Π is CCA-secure. ∎

The need for independent keys. We conclude this section by stressing a basic principle of cryptography: *different instances of cryptographic primitives should always use independent keys*. To illustrate this, we consider what can happen to the encrypt-then-authenticate methodology if the same key k is used for both encryption and authentication. Let F be a strong pseudorandom permutation. It follows that F^{-1} is a strong pseudorandom permutation also. Define $\mathsf{Enc}_k(m) = F_k(m\|r)$ for $m \in \{0,1\}^{n/2}$ and a uniform $r \in \{0,1\}^{n/2}$; it can be shown that this encryption scheme is CPA-secure. (In fact, it is even CCA-secure; see Exercise 5.9.) Define $\mathsf{Mac}_k(c) = F^{-1}_k(c)$; this is just Construction 4.5, so is strongly secure. However, using these schemes with the same key k to encrypt-then-authenticate a message m yields:

$$\mathsf{Enc}_k(m), \mathsf{Mac}_k(\mathsf{Enc}_k(m)) = F_k(m\|r), F^{-1}_k(F_k(m\|r)) = F_k(m\|r), m\|r,$$

and so the message m is revealed in the clear! This does not in any way contradict Theorem 5.7, since Construction 5.6 expressly requires that k_M, k_E be chosen independently. We encourage the reader to determine where this independence is used in the proof of Theorem 5.7.

Authenticated encryption with associated data. As described at the end of Section 5.2.1, there are settings where a message m is encrypted along with associated data d that requires integrity but not secrecy. It is easy to modify the encrypt-then-authenticate approach to handle this: simply compute $c \leftarrow \mathsf{Enc}_{k_E}(m)$ followed by $t \leftarrow \mathsf{Mac}_{k_M}(d\|c)$.

5.3.2 Standardized Schemes

We close this chapter by briefly describing three AE schemes used in practice that are each inspired by one of the approaches discussed earlier. As usual, our aim here is not to provide an exact description of these schemes, but rather just a high-level understanding of the constructions.

GCM (Galois/counter mode). GCM can be viewed as following the *encrypt-then-authenticate* paradigm, with CTR mode (cf. Section 3.6.3) as

the underlying encryption scheme and GMAC (cf. Section 4.5.2) as the underlying message authentication code. The main differences from the generic combination described in the previous section are that (1) the keys used for encryption and authentication are *not* independent and (2) the same IV is used both for CTR-mode encryption and as the nonce for GMAC. Both these changes can be proven secure for the particular way they are done by GCM.

One important property to be aware of when using GCM is that if the IV ever repeats, then not only does secrecy fail for the two messages encrypted using the same IV, but integrity of the scheme may be completely broken. This is due to a property of GMAC discussed in Exercise 4.21. For this reason, great care must be taken to ensure that IVs do not repeat when using GCM.

When GCM is instantiated with the AES block cipher (see Section 7.2.5) it is extremely fast on most modern processors due to dedicated hardware instructions for both AES and the field operations used in GMAC. The scheme is also highly parallelizable.

CCM (Counter with CBC-MAC). CCM follows the *authenticate-then-encrypt* approach, with CTR mode as the underlying encryption scheme and CBC-MAC (cf. Section 4.4.1) as the underlying message authentication code. Moreover, the same key k is used for both. Although—as discussed in the previous section—the authenticate-then-encrypt approach is not secure in general, and problems can occur when the keys used for encryption and authentication are not independent, CCM itself can be proven secure.

Because CCM relies only on a block cipher using a single key, it is easy to implement. However, CCM is relatively slow (it requires two block-cipher evaluations per plaintext block) and cannot be fully parallelized. In addition, it does not work in an on-line fashion, since it requires the message length to be known before encryption begins. (This is because the length is prepended to the message before CBC-MAC is computed, as discussed in Section 4.4.1.)

ChaCha20–Poly1305. This scheme relies on the *encrypt-then-authenticate* approach, where the underlying encryption is done using the stream cipher ChaCha20 (cf. Section 7.1.5) in unsynchronized mode (cf. Section 3.6.2) and the MAC used is Poly1305 (cf. Section 4.5.2) with ChaCha20 used here to instantiate the pseudorandom function. This scheme is extremely fast in software, and is becoming the method of choice on platforms where the dedicated hardware instructions used by GCM are not available.

5.4 Secure Communication Sessions

We briefly describe the application of authenticated encryption to the setting of two parties who wish to communicate "securely"—namely, with joint secrecy and integrity—over the course of a communication session. (For the

purposes of this section, a *communication session* is simply a period of time during which the communicating parties maintain state.) In our treatment here we are deliberately informal; a formal definition is quite involved, and this topic arguably lies more in the area of network security than cryptography.

Let $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ be an authenticated encryption scheme. Consider two parties A and B who share a key k and wish to use this key to secure their communication over the course of a session. The obvious thing to do here is to use Π: Whenever, say, A wants to transmit a message m to B, she computes $c \leftarrow \mathsf{Enc}_k(m)$ and sends c to B; in turn, B decrypts c to recover the result (ignoring the result if decryption returns $\perp$). Likewise, the same procedure is followed when B wants to send a message to A. This simple approach, however, is vulnerable to various potential attacks:

Re-ordering attack: An attacker can swap the order of messages. For example, if A transmits c_1 (an encryption of m_1) and subsequently transmits c_2 (an encryption of m_2), an attacker who has some control over the network can deliver c_2 before c_1 and thus cause B to output the messages in the wrong order.

Replay attack: An attacker can *replay* a (valid) ciphertext c sent previously by one of the parties. This would cause one party to output a message twice, even though the other party only sent it once.

Message-dropping attack: An attacker may drop some of the messages sent between A and B. Although nothing can prevent the attacker from doing this, we might at least hope that such behavior would be detected by the parties.

Reflection attack: An attacker can take a ciphertext c sent from A to B and send it back to A. This would cause A to output a message m, even though B never sent such a message.

The above list of attacks is not exhaustive, and is just an example of some of the challenges involved in achieving secure communication.

The above attacks can be addressed using *counters* to handle the first three and a *directionality bit* to prevent the fourth.[2] We describe these in tandem. Each party maintains two counters $\mathsf{ctr}_{A,B}$ and $\mathsf{ctr}_{B,A}$ keeping track of the number of messages sent from A to B (resp., B to A) during the session. These counters are initialized to 0 and incremented each time a party sends or receives a (valid) message. The parties also agree on a bit $b_{A,B}$, and define $b_{B,A}$ to be its complement. (One way to do this is to set $b_{A,B} = 0$ iff the identity of A is lexicographically smaller than the identity of B.)

[2] In practice, reflection attacks are often solved by simply having separate keys for each direction (i.e., the parties use a key k_A for messages sent from A to B, and an independent key k_B for messages sent from B to A).

When A wants to transmit a message m to B, she computes the ciphertext $c \leftarrow \mathsf{Enc}_k(b_{A,B} \| \mathsf{ctr}_{A,B} \| m)$ and sends c; she then increments $\mathsf{ctr}_{A,B}$. Upon receiving c, party B decrypts; if the result is $\perp$, he immediately rejects. Otherwise, he parses the decrypted message as $b \| \mathsf{ctr} \| m$. If $b = b_{A,B}$ and $\mathsf{ctr} = \mathsf{ctr}_{A,B}$, then B outputs m and increments $\mathsf{ctr}_{A,B}$; otherwise, B rejects. The above steps, *mutatis mutandis*, are applied when B sends a message to A.

References and Additional Reading

Chosen-ciphertext attacks (in the context of public-key encryption) were first formally defined by Naor and Yung [147] and Rackoff and Simon [168], and have received much subsequent attention as well [17, 68, 112]. The padding-oracle attack originated in the work of Vaudenay [199].

The importance of authenticated encryption was first explicitly highlighted by Katz and Yung [111] and Bellare and Namprempre [21]. Definition 5.4 is due to Shrimpton [184], who also proves Theorem 5.5. Bellare and Namprempre [21] analyze the three generic approaches discussed here, though the idea of using encrypt-then-authenticate for achieving CCA-security goes back at least to the work of Dolev et al. [68]. Krawczyk [122] examines other methods for achieving secrecy and authentication, and also analyzes specific instantiations of the authenticate-then-encrypt approach.

GCM is due to McGrew and Viega [136]. CCM was proposed by Whiting, Housley, and Ferguson [204] and proven secure by Jonsson [104]. ChaCha20–Poly1305 is specified in RFC 8439 [154].

Exercises

5.1 Show that the CBC, OFB, and CTR modes of operation do not give CCA-secure encryption schemes.

5.2 Write pseudocode for obtaining the entire plaintext for a 3-block ciphertext via a padding-oracle attack on CBC-mode encryption using PKCS #7 padding, as sketched in the text.

5.3 Describe a padding-oracle attack on CTR-mode encryption, assuming PKCS #7 padding is used to pad messages to a multiple of the block length before encrypting.

5.4 Show that Construction 5.6 is not necessarily CCA-secure if it is instantiated with a secure MAC that is not *strongly* secure.

5.5 Prove that Construction 5.6 is unforgeable when instantiated with any encryption scheme (even if not CPA-secure) and any secure MAC (even if not strongly secure).

5.6 Consider a strengthened version of unforgeability where $\mathcal{A}$ is additionally given access to a decryption oracle.

(a) Write a formal definition for this version of unforgeability.

(b) Prove that Construction 5.6 satisfies this stronger definition if Π_M is a strongly secure MAC.

(c) Show by counterexample that Construction 5.6 need not satisfy this stronger definition if Π_M is a secure MAC that is not strongly secure. (Compare to the previous exercise.)

5.7 Prove that the authenticate-then-encrypt approach, instantiated with any CPA-secure encryption scheme and any secure MAC, yields a CPA-secure encryption scheme that is unforgeable.

5.8 Let F be a strong pseudorandom permutation, and define a fixed-length encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ as follows: On input $m \in \{0,1\}^{n/2}$ and key $k \in \{0,1\}^n$, algorithm Enc chooses a uniform string $r \in \{0,1\}^{n/2}$ of length $n/2$ and computes $c := F_k(r \| m)$.

Show how to decrypt, and prove that this scheme is CCA-secure for messages of length $n/2$.

5.9 Show that the scheme in the previous exercise is not an authenticated encryption scheme.

5.10 Show a CPA-secure private-key encryption scheme that is unforgeable but is not CCA-secure.