Chapter 4

Message Authentication Codes

4.1 Message Integrity

4.1.1 Secrecy vs. Integrity

One of the most basic goals of cryptography is to enable parties to communicate over an *open communication channel* in a secure way. But what does "secure communication" entail? In Chapter 3 we showed that it is possible to obtain *secret* communication over an open channel. That is, we showed how encryption can be used to prevent an eavesdropper (or possibly a more active adversary) from learning anything about the content of messages sent over an unprotected communication channel. However, not all security concerns are related to secrecy. In many cases, it is of equal or greater importance to guarantee *message integrity* (or *message authentication*) in the sense that each party should be able to identify when a message it receives was sent by the party claiming to send it, and was not modified in transit. We look at two canonical examples.

Consider the case of a user communicating with their bank over the Internet. When the bank receives a request to transfer \$1,000 from the user's account to the account of some other user X, the bank has to consider the following:

1. Is the request authentic? That is, did the user in question really issue this request, or was the request issued by an adversary (perhaps X itself) who is impersonating the legitimate user?
2. Assuming a transfer request was issued by the legitimate user, are the details of the request as received exactly those intended by the legitimate user? Or was, e.g., the transfer amount modified?

Note that standard error-correction techniques do not suffice for the second concern. Error-correcting codes are only intended to detect and recover from "random" errors that affect only a small portion of the transmission, but they do nothing to protect against a malicious adversary who can choose exactly where to introduce an arbitrary number of errors.

A second scenario where the need for message integrity arises in practice is with regard to web cookies. The HTTP protocol used for web traffic is *stateless*, and so when a client and server communicate in some session (e.g., when a user [client] shops at a merchant's [server's] website), any state generated as part of that session (e.g., the contents of the user's shopping cart) is often placed in a "cookie" that is stored by the client and sent from the client to the server as part of each message the client sends. Assume that the cookie stored by some user includes the items in the user's shopping cart along with a price for each item, as might be done if the merchant offers different prices to different users (reflecting, e.g., discounts and promotions, or user-specific pricing). It should be infeasible here for the user to modify the cookie that it stores so as to alter the prices of the items in its cart. The merchant thus needs a technique to ensure the integrity of the cookie that it stores at the user. Note that the contents of the cookie (namely, the items and their prices) are not secret and, in fact, must be known by the user. The problem here is thus purely one of integrity.

In general, one cannot assume the integrity of communication without taking specific measures to ensure it. Indeed, any unprotected online purchase order, online banking operation, email, or SMS message cannot, in general, be trusted to have originated from the claimed source and to have been unmodified in transit. Unfortunately, people are in general trusting and thus information like the caller-ID or an email return address are taken to be "proofs of origin" in many cases, even though they are relatively easy to forge. This leaves the door open to potentially damaging attacks.

In this chapter we will show how to achieve message integrity by using cryptographic techniques to prevent the *undetected* tampering of messages sent over an open communication channel. Note that we cannot hope to prevent adversarial tampering of messages altogether, as that can only be defended against at the physical level. Instead, what we will guarantee is that any such tampering will be detected by the honest parties.

4.1.2 Encryption vs. Message Authentication

Just as the goals of secrecy and message integrity are different, so are the techniques and tools for achieving them. Unfortunately, secrecy and integrity are often confused and unnecessarily intertwined, so let us be clear up front: encryption does *not* (in general) provide any integrity, and encryption should never be used with the intent of achieving message authentication unless it is specifically designed with that purpose in mind (something we will return to in Section 4.5).

One might mistakenly think that encryption solves the problem of message authentication. (In fact, this is a common error.) This is due to the fuzzy, and incorrect, reasoning that since a ciphertext completely hides the contents of the message, an adversary cannot possibly modify an encrypted message in any meaningful way. Despite its intuitive appeal, this reasoning is completely false. We illustrate this point by showing that all the encryption schemes we have seen thus far do not provide message integrity.

Encryption using stream ciphers. Consider the simple encryption scheme in which $\mathsf{Enc}_k(m)$ computes the ciphertext $c := G(k) \oplus m$, where G is a pseudorandom generator. Ciphertexts in this case are very easy to manipulate: flipping any bit in the ciphertext c results in the same bit being flipped in the message that is recovered upon decryption. Thus, given a ciphertext c that encrypts a (possibly unknown) message m, it is possible to generate a modified ciphertext c' such that $m' := \mathsf{Dec}_k(c')$ is the same as m but with one (or more) of the bits flipped. This simple attack can have severe consequences. As an example, consider the case of a user encrypting some dollar amount he wants to transfer from his bank account, where the amount is represented in binary. Flipping the least significant bit has the effect of changing this amount by only \$1, but flipping the 11th least significant bit changes the amount by more than \$1,000! (Interestingly, the adversary in this example does not necessarily learn whether it is increasing or decreasing the initial amount, i.e., whether it is flipping a 0 to a 1 or vice versa. But if the adversary has some partial knowledge about the amount—say, that it is less than \$1,000 to begin with—then the modifications it introduces can have a predictable effect.) We stress that this attack does not contradict the *secrecy* of the encryption scheme (in the sense of Definition 3.8). In fact, the exact same attack applies to the one-time pad encryption scheme, showing that even perfect secrecy is not sufficient to ensure the most basic level of message integrity.

Encryption using block ciphers. The attack described above utilizes the fact that flipping a single bit in a ciphertext keeps the underlying plaintext unchanged except for the corresponding bit (which is also flipped). The same attack applies to the OFB- and CTR-mode encryption schemes, which also encrypt messages by XORing them with a pseudorandom stream (albeit one that changes each time a message is encrypted). We thus see that even using CPA-secure encryption is not enough to prevent message tampering.

One may hope that attacking ECB- or CBC-mode encryption would be more difficult since decryption in these cases involves inverting a (strong) pseudorandom permutation F, and we expect that $F_k^{-1}(x)$ and $F_k^{-1}(x')$ will be completely uncorrelated even if x and x' differ in only a single bit. (Of course, ECB mode does not even guarantee the most basic notion of secrecy, but that does not matter for the present discussion.) Nevertheless, single-bit modifications of a ciphertext still cause partially predictable changes in the plaintext. For example, when using ECB mode, flipping a bit in the ith block of the ciphertext affects *only* the ith block of the plaintext—all other blocks remain unchanged. Although the effect on the ith block of the plaintext may be impossible to predict, changing that one block (while leaving everything else unchanged) may represent a harmful attack. Moreover, the order of plaintext blocks can be changed (without garbling any block) by simply changing the order of the corresponding ciphertext blocks, and the message can be truncated by just dropping ciphertext blocks.

Similarly, when using CBC mode, flipping the jth bit of the IV changes only the jth bit of the first message block m_1 (since $m_1 := F_k^{-1}(c_1) \oplus IV'$, where IV' is the modified IV); all plaintext blocks other than the first remain unchanged (since the ith block of the plaintext is computed as $m_i := F_k^{-1}(c_i) \oplus c_{i-1}$, and blocks c_i and c_{i-1} have not been modified). Therefore, the first block of a CBC-encrypted message can be changed arbitrarily. This a serious concern in practice since the first block often contains important header information.

Finally, note that all encryption schemes we have seen thus far have the property that *every* ciphertext (perhaps satisfying some length constraint) corresponds to *some* valid message. So it is trivial for an adversary to "spoof" a message on behalf of one of the communicating parties—by sending some arbitrary ciphertext—even if the adversary has no idea what the underlying message will be. As we will see when we formally define authenticated encryption in Section 4.5, even an attack of this sort should be ruled out.

4.2 Message Authentication Codes – Definitions

We have seen that, in general, encryption does not solve the problem of message integrity. Rather, an additional mechanism is needed that will enable the communicating parties to know whether or not a message was tampered with. The right tool for this task is a *message authentication code* (MAC).

The aim of a message authentication code is to prevent an adversary from modifying a message sent by one party to another, or from injecting a new message, without the receiver detecting that the message did not originate from the intended party. As in the case of encryption, this is only possible if the communicating parties share some secret that the adversary does not know (otherwise nothing can prevent an adversary from impersonating the party sending the message). Here, we will continue to consider the private-key setting where the parties share the same secret key.[1]

The Syntax of a Message Authentication Code

Before formally defining security of a message authentication code, we first define what a MAC is and how it is used. Two users who wish to communicate in an authenticated manner begin by generating and sharing a secret key k in advance of their communication. When one party wants to send a message m to the other, she computes a MAC tag (or simply a tag) t based on the message and the shared key, and sends the message m and the tag t to the other party. The tag is computed using a *tag-generation algorithm* denoted by Mac; thus, rephrasing what we have already said, the sender of a message m computes $t \leftarrow \mathsf{Mac}_k(m)$ and transmits (m, t) to the receiver. Upon receiving (m, t), the

[1]In the web-cookie example discussed earlier, the merchant is (in effect) communicating "with itself" with the user acting as a communication channel. In that setting, the server alone needs to know the key since it is acting as both sender and receiver.

second party *verifies* whether t is a valid tag on the message m (with respect to the shared key) or not. This is done by running a *verification algorithm* Vrfy that takes as input the shared key as well as a message m and a tag t, and indicates whether the given tag is valid. Formally:

DEFINITION 4.1 *A* message authentication code (*or* MAC) *consists of three probabilistic polynomial-time algorithms* $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *such that:*

1. *The* key-generation algorithm Gen *takes as input the security parameter* 1^n *and outputs a key* k *with* $|k| \geq n$.
2. *The* tag-generation algorithm Mac *takes as input a key* k *and a message* $m \in \{0,1\}^*$*, and outputs a tag* t*. Since this algorithm may be randomized, we write this as* $t \leftarrow \mathsf{Mac}_k(m)$.
3. *The deterministic* verification algorithm Vrfy *takes as input a key* k*, a message* m*, and a tag* t*. It outputs a bit* b*, with* $b = 1$ *meaning* valid *and* $b = 0$ *meaning* invalid*. We write this as* $b := \mathsf{Vrfy}_k(m, t)$.

It is required that for every n*, every key* k *output by* $\mathsf{Gen}(1^n)$*, and every* $m \in \{0,1\}^*$*, it holds that* $\mathsf{Vrfy}_k(m, \mathsf{Mac}_k(m)) = 1$.

If there is a function ℓ *such that for every* k *output by* $\mathsf{Gen}(1^n)$*, algorithm* Mac_k *is only defined for messages* $m \in \{0,1\}^{\ell(n)}$*, then we call the scheme a* fixed-length MAC for messages of length $\ell(n)$.

As with private-key encryption, $\mathsf{Gen}(1^n)$ almost always simply chooses a uniform key $k \in \{0,1\}^n$, and we omit Gen in that case.

Canonical verification. For deterministic message authentication codes (that is, where Mac is a deterministic algorithm), the canonical way to perform verification is to simply re-compute the tag and check for equality. In other words, $\mathsf{Vrfy}_k(m, t)$ first computes $\tilde{t} := \mathsf{Mac}_k(m)$ and then outputs 1 if and only if $\tilde{t} = t$. Even for deterministic MACs, however, it is useful to define a separate Vrfy algorithm in order to explicitly distinguish the semantics of *authenticating* a message vs. *verifying* its authenticity.

Security of Message Authentication Codes

We now define the default notion of security for message authentication codes. The intuitive idea behind the definition is that no efficient adversary should be able to generate a valid tag on any "new" message that was not previously sent (and authenticated) by one of the communicating parties.

As with any security definition, to formalize this notion we have to define both the adversary's power as well as what should be considered a "break." As usual, we consider only probabilistic polynomial-time adversaries[2] and

[2]See Section 4.6 for a discussion of information-theoretic message authentication, where no computational restrictions are placed on the adversary.

so the real question is how we model the adversary's interaction with the communicating parties. In the setting of message authentication, an adversary observing the communication between the honest parties may be able to see all the messages sent by these parties along with their corresponding MAC tags. The adversary may also be able to influence the *content* of these messages, whether directly or indirectly (if, e.g., external actions of the adversary affect the messages sent by the parties). This is true, for example, in the web cookie example from earlier, where the user's own actions influence the contents of the cookie being stored on his computer.

To formally model the above we allow the adversary to request MAC tags for *any* messages of its choice. Formally, we give the adversary access to a *MAC oracle* $\mathsf{Mac}_k(\cdot)$; the adversary can repeatedly submit any message m of its choice to this oracle, and is given in return a tag $t \leftarrow \mathsf{Mac}_k(m)$. (For a fixed-length MAC, only messages of the correct length can be submitted.)

We will consider it a "break" of the scheme if the adversary is able to output any message m along with a tag t such that: (1) t is a valid tag on the message m (i.e., $\mathsf{Vrfy}_k(m,t) = 1$), and (2) the adversary had not previously requested a MAC tag on the message m (i.e., from its oracle). The first condition means that if the adversary were to send (m,t) to one of the honest parties, then this party would be mistakenly fooled into thinking that m originated from the legitimate party since $\mathsf{Vrfy}_k(m,t) = 1$. The second condition is required because it is always possible for the adversary to just copy a message and MAC tag that were previously sent by one of the legitimate parties (and, of course, these would be accepted as valid). Such a *replay attack* is not considered a "break" of the message authentication code. This does *not* mean that replay attacks are not a security concern; they are, and we will have more to say about them below.

A MAC satisfying the level of security specified above is said to be *existentially unforgeable under an adaptive chosen-message attack*. "Existential unforgeability" refers to the fact that the adversary must not be able to forge a valid tag on *any* message, and "adaptive chosen-message attack" refers to the fact that the adversary is able to obtain MAC tags on arbitrary messages chosen adaptively during its attack.

Toward the formal definition, consider the following experiment for a message authentication code $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$, an adversary $\mathcal{A}$, and value n for the security parameter:

The message authentication experiment $\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)$**:**

1. *A key k is generated by running* $\mathsf{Gen}(1^n)$.
2. *The adversary $\mathcal{A}$ is given input 1^n and oracle access to* $\mathsf{Mac}_k(\cdot)$*. The adversary eventually outputs (m,t). Let $\mathcal{Q}$ denote the set of all queries that $\mathcal{A}$ asked its oracle.*
3. $\mathcal{A}$ succeeds *if and only if (1)* $\mathsf{Vrfy}_k(m,t) = 1$ *and (2)* $m \notin \mathcal{Q}$*. In that case the output of the experiment is defined to be* 1.

A MAC is secure if no efficient adversary can succeed in the above experiment with non-negligible probability:

DEFINITION 4.2 *A message authentication code* $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *is* existentially unforgeable under an adaptive chosen-message attack*, or just* secure*, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, there is a negligible function* negl *such that:*

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

Is the definition *too* strong? The above definition is rather strong in two respects. First, the adversary is allowed to request MAC tags for *any* messages of its choice. Second, the adversary is considered to have "broken" the scheme if it can output a valid tag on *any* previously unauthenticated message. One might object that both these components of the definition are unrealistic and overly strong: in "real-world" usage of a MAC, the honest parties would only authenticate "meaningful" messages (over which the adversary might have only limited control), and similarly it should only be considered a breach of security if the adversary can forge a valid tag on a "meaningful" message. Why not tailor the definition to capture this?

The crucial point is that what constitutes a meaningful message is entirely *application dependent.* While some applications of a MAC may only ever authenticate English-text messages, other applications may authenticate spreadsheet files, others database entries, and others raw data. Protocols may also be designed where *anything* will be authenticated—in fact, certain protocols for entity authentication do exactly this. By making the definition of security for MACs as strong as possible, we ensure that secure MACs are broadly applicable for a wide range of purposes, without having to worry about compatibility of the MAC with the semantics of the application.

Replay attacks. We emphasize that the above definition, and message authentication codes on their own, offer no protection against *replay attacks* whereby a previously sent message (and its MAC tag) are replayed to an honest party. Nevertheless, replay attacks are a serious concern! Consider again the scenario where a user (say, Alice) sends a request to her bank to transfer \$1,000 from her account to some other user (say, Bob). In doing so, Alice can compute a MAC tag and append it to the request so the bank knows the request is authentic. If the MAC is secure, Bob will be unable to intercept the request and change the amount to \$10,000 because this would involve forging a valid tag on a previously unauthenticated message. However, nothing prevents Bob from intercepting Alice's message and replaying it *ten times* to the bank. If the bank accepts each of these messages, the net effect is that \$10,000 will be transferred to Bob's account rather than the desired \$1,000.

Despite the real threat that replay attacks represent, a MAC by itself *cannot* protect against such attacks since the definition of a MAC (Definition 4.1) does not incorporate any notion of *state* into the verification algorithm (and so every time a valid pair (m, t) is presented to the verification algorithm, it will always output 1). Rather, protection against replay attacks—if such protection is necessary at all—must be handled by some higher-level application. The reason the definition of a MAC is structured this way is, once again, because we are unwilling to assume any semantics regarding applications that use MACs; in particular, the decision as to whether or not a replayed message should be treated as "valid" may be application dependent.

Two common techniques for preventing replay attacks are to use *sequence numbers* (also known as *counters*) or *time-stamps*. The first approach, described (in a more general context) in Section 4.5.3, requires the communicating users to maintain (synchronized) state, and can be problematic when users communicate over a lossy channel where messages are occasionally dropped (though this problem can be mitigated). In the second approach, using time-stamps, the sender prepends the current time T (say, to the nearest millisecond) to the message before authenticating, and sends T along with the message and the resulting tag t. When the receiver obtains T, m, t, it verifies that t is a valid tag on $T \| m$ and that T is within some acceptable clock skew from the current time T' at the receiver. This method has certain drawbacks as well, including the need for the sender and receiver to maintain closely synchronized clocks, and the possibility that a replay attack can still take place if it is done quickly enough (specifically, within the acceptable time window).

Strong MACs. As defined, a secure MAC ensures that an adversary cannot generate a valid tag on a new message that was never previously authenticated. But it does not rule out the possibility that an attacker might be able to generate a new *tag* on a previously authenticated message. That is, a MAC guarantees that if an attacker learns tags $t_1, \ldots$ on messages $m_1, \ldots$, then it will not be able to forge a valid tag t on any message $m \notin \{m_1, \ldots\}$. However, it may be possible for an adversary to "forge" a *different* valid tag $t_1' \neq t_1$ on the message m_1. In general, this type of adversarial behavior is not a concern. Nevertheless, in some settings it is useful to consider a stronger definition of security for MACs where such behavior is ruled out.

Formally, we consider a modified experiment $\mathsf{Mac\text{-}sforge}$ that is defined in exactly the same way as $\mathsf{Mac\text{-}forge}$, except that now the set $\mathcal{Q}$ contains *pairs* of oracle queries and their associated responses. (That is, $(m, t) \in \mathcal{Q}$ if $\mathcal{A}$ queried $\mathsf{Mac}_k(m)$ and received in response the tag t.) The adversary $\mathcal{A}$ succeeds (and experiment $\mathsf{Mac\text{-}sforge}$ evaluates to 1) if and only if $\mathcal{A}$ outputs (m, t) such that $\mathsf{Vrfy}_k(m, t) = 1$ and $(m, t) \notin \mathcal{Q}$.

DEFINITION 4.3 *A message authentication code* $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *is* strongly secure*, or a* strong MAC*, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, there is a negligible function* negl *such that:*

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

It is not hard to see that if a secure MAC uses canonical verification then it is also strongly secure. This is important since all real-world MACs use canonical verification. We leave the proof of the following as an exercise.

PROPOSITION 4.4 *Let* $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *be a secure MAC that uses canonical verification. Then* Π *is a strong MAC.*

Verification Queries

Definitions 4.2 and 4.3 give the adversary access to a MAC oracle, which corresponds to a real-world adversary who can influence an honest sender to generate a tag for some message m. One could also consider an adversary who interacts with an honest receiver, sending m', t' to the receiver to learn whether $\mathsf{Vrfy}_k(m', t') = 1$. Such an adversary could be captured formally in the natural way by giving the adversary in the above definitions access to a verification oracle as well.

A definition that incorporates a verification oracle in this way is, perhaps, the "right" way to define security for message authentication codes. It turns out, however, that for MACs that use canonical verification it makes no difference: any such MAC that satisfies Definition 4.2 also satisfies the definitional variant in which verification queries are allowed. Similarly, any strong MAC automatically also remains secure even in a setting where verification queries are possible. (This, in fact, serves as one motivation for the definition of strong security for MACs.) For MACs that do not use canonical verification, however, allowing verification queries can make a difference; see Exercises 4.2 and 4.3. Since most MACs covered in this book (as well as MACs used in practice) use canonical verification, we use the traditional definitions that omit access to a verification oracle.

A potential timing attack. One issue not addressed by the above is the possibility of carrying out a *timing attack* on MAC verification. Here, we consider an adversary who can send message/tag pairs to the receiver—thus using the receiver as a verification oracle—and learn not only whether the receiver accepts or rejects, but also the *time* it takes for the receiver to make this decision. We show that if such an attack is possible then a natural implementation of MAC verification leads to an easily exploitable vulnerability.

(Note that in our usual cryptographic definitions of security, the attacker learns only the output of the oracles it has access to, but nothing else. The attack we describe here, which is an example of a *side-channel attack*, shows that certain real-world attacks are not captured by the usual definitions.)

Concretely, assume a MAC using canonical verification. To verify a tag t on a message m, the receiver computes $t' := \mathsf{Mac}_k(m)$ and then compares t' to t, outputting 1 if and only if t' and t are equal. Assume this comparison is implemented using a standard routine (like `strcmp` in C) that compares t and t' one byte at a time, and rejects as soon as the first unequal byte is encountered. The observation is that, when implemented in this way, the *time* to reject differs depending on the *position* of the first unequal byte.

It is possible to use this seemingly inconsequential information to forge a tag on any desired message m. The basic idea is this: say the attacker knows the first i bytes of the correct tag for m. (At the outset, $i = 0$.) The attacker will learn the next byte of the correct tag by sending $(m, t_0), \ldots, (m, t_{255})$ to the receiver, where t_j is the string with the first i bytes set correctly, the $(i+1)$st byte equal to j (in hexadecimal), and the remaining bytes set to `0x00`. All of these tags will likely be rejected (if not, then the attacker succeeds anyway); however, for exactly one of these tags the first $(i + 1)$ bytes will match the correct tag and rejection will take slightly longer than the rest. If t_j is the tag that caused rejection to take the longest, the attacker learns that the $(i+1)$st byte of the correct tag is j. In this way, the attacker learns each byte of the correct tag using at most 256 queries to the verification oracle. For a 16-byte tag, this attack requires only 4096 queries in the worst case.

One might wonder whether this attack is realistic, as it requires access to a verification oracle as well as the ability to measure the difference in time taken to compare i vs. $i + 1$ bytes. In fact, exactly such attacks have been carried out against real systems! As just one example, MACs were used to verify code updates in the Xbox 360, and the implementation of MAC verification used there had a difference of 2.2 milliseconds between rejection times. Attackers were able to exploit this and load pirated games onto the hardware.

Based on the above, we conclude that MAC verification should use *time-independent* string comparison that always compares *all* bytes.

4.3 Constructing Secure Message Authentication Codes

4.3.1 A Fixed-Length MAC

Pseudorandom functions are a natural tool for constructing secure message authentication codes. Intuitively, if the MAC tag t is obtained by applying a pseudorandom function to the message m, then forging a tag on a previously unauthenticated message requires the adversary to correctly guess the value of the pseudorandom function at a "new" input point. The probability of guessing the value of a *random* function on a new point is 2^{-n} (if the output length of the function is n). The probability of guessing such a value for a *pseudorandom* function can be only negligibly greater.

The above idea, shown in Construction 4.5, works for constructing a secure *fixed-length* MAC for messages of length n (since our pseudorandom functions by default have n-bit block length). This is useful, but falls short of our goal. In Section 4.3.2, we show how to extend this to handle messages of arbitrary length. We explore more efficient constructions of MACs for arbitrary-length messages in Sections 4.4 and 5.3.2.

CONSTRUCTION 4.5

Let F be a pseudorandom function. Define a fixed-length MAC for messages of length n as follows:

- Mac: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^n$, output the tag $t := F_k(m)$. (If $|m| \neq |k|$ then output nothing.)
- Vrfy: on input a key $k \in \{0,1\}^n$, a message $m \in \{0,1\}^n$, and a tag $t \in \{0,1\}^n$, output 1 if and only if $t \stackrel{?}{=} F_k(m)$. (If $|m| \neq |k|$, then output 0.)

A fixed-length MAC from any pseudorandom function.

THEOREM 4.6 *If F is a pseudorandom function, then Construction 4.5 is a secure fixed-length MAC for messages of length n.*

PROOF As in previous uses of pseudorandom functions, this proof follows the paradigm of first analyzing the security of the scheme using a truly random function, and then considering the result of replacing the truly random function with a pseudorandom one.

Let $\mathcal{A}$ be a probabilistic polynomial-time adversary. Consider the message authentication code $\widetilde{\Pi} = (\widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Mac}}, \widetilde{\mathsf{Vrfy}})$ which is the same as $\Pi = (\mathsf{Mac}, \mathsf{Vrfy})$ in Construction 4.5 except that a truly random function f is used instead of the pseudorandom function F_k. That is, $\widetilde{\mathsf{Gen}}(1^n)$ works by choosing a uniform function $f \in \mathsf{Func}_n$, and $\widetilde{\mathsf{Mac}}$ computes a tag just as Mac does except that f is used instead of F_k. It is immediate that

$$\Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\widetilde{\Pi}}(n) = 1] \leq 2^{-n} \tag{4.1}$$

because for any message $m \notin \mathcal{Q}$, the value $t = f(m)$ is uniformly distributed in $\{0,1\}^n$ from the point of view of the adversary $\mathcal{A}$.

We next show that there is a negligible function negl such that

$$\left|\Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n) = 1] - \Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\widetilde{\Pi}}(n) = 1]\right| \leq \mathsf{negl}(n); \tag{4.2}$$

combined with Equation (4.1), this shows that

$$\Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n) = 1] \leq 2^{-n} + \mathsf{negl}(n),$$

proving the theorem.

To prove Equation (4.2), we construct a polynomial-time distinguisher D that is given oracle access to some function, and whose goal is to determine whether this function is pseudorandom (i.e., equal to F_k for uniform $k \in \{0,1\}^n$) or random (i.e., equal to f for uniform $f \in \mathsf{Func}_n$). To do this, D emulates the message authentication experiment for $\mathcal{A}$ and observes whether $\mathcal{A}$ succeeds in outputting a valid tag on a "new" message. If so, D guesses that its oracle is a pseudorandom function; otherwise, D guesses that its oracle is a random function. In detail:

Distinguisher D:

D is given input 1^n and access to an oracle $\mathcal{O} : \{0,1\}^n \to \{0,1\}^n$, and works as follows:

1. Run $\mathcal{A}(1^n)$. Whenever $\mathcal{A}$ queries its MAC oracle on a message m (i.e., whenever $\mathcal{A}$ requests a tag on a message m), answer this query in the following way:

   Query $\mathcal{O}$ with m and obtain response t; return t to $\mathcal{A}$.
2. When $\mathcal{A}$ outputs (m, t) at the end of its execution, do:
   (a) Query $\mathcal{O}$ with m and obtain response $\hat{t}$.
   (b) If (1) $\hat{t} = t$ and (2) $\mathcal{A}$ never queried its MAC oracle on m, then output 1; otherwise, output 0.

It is clear that D runs in polynomial time.

Notice that if D's oracle is a pseudorandom function, then the view of $\mathcal{A}$ when run as a sub-routine by D is distributed identically to the view of $\mathcal{A}$ in experiment $\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)$. Furthermore, D outputs 1 exactly when $\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n) = 1$. Therefore

$$\Pr\left[D^{F_k(\cdot)}(1^n) = 1\right] = \Pr\left[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n) = 1\right],$$

where $k \in \{0,1\}^n$ is chosen uniformly in the above. If D's oracle is a random function, then the view of $\mathcal{A}$ when run as a sub-routine by D is distributed identically to the view of $\mathcal{A}$ in experiment $\mathsf{Mac\text{-}forge}_{\mathcal{A},\widetilde{\Pi}}(n)$, and again D outputs 1 exactly when $\mathsf{Mac\text{-}forge}_{\mathcal{A},\widetilde{\Pi}}(n) = 1$. Thus,

$$\Pr\left[D^{f(\cdot)}(1^n) = 1\right] = \Pr\left[\mathsf{Mac\text{-}forge}_{\mathcal{A},\widetilde{\Pi}}(n) = 1\right],$$

where $f \in \mathsf{Func}_n$ is chosen uniformly.

Since F is a pseudorandom function and D runs in polynomial time, there exists a negligible function negl such that

$$\left|\Pr\left[D^{F_k(\cdot)}(1^n) = 1\right] - \Pr\left[D^{f(\cdot)}(1^n) = 1\right]\right| \leq \mathsf{negl}(n).$$

This implies Equation (4.2), completing the proof of the theorem. ■

4.3.2 Domain Extension for MACs

Construction 4.5 is important in that it shows a general paradigm for constructing secure message authentication codes from pseudorandom functions. Unfortunately, the construction is only capable of handling *fixed-length* messages that are furthermore rather short.[3] These limitations are unacceptable

[3]Given a pseudorandom function taking arbitrary-length inputs, Construction 4.5 would yield a secure MAC for messages of arbitrary length. Likewise, a pseudorandom function

in most applications. We show here how a general MAC, handling arbitrary-length messages, can be constructed from any fixed-length MAC for messages of length n. The construction we show is not very efficient and is unlikely to be used in practice. Indeed, far more efficient constructions of secure MACs are known, as we discuss in Sections 4.4 and 5.3.2. We include the present construction for its simplicity and generality.

Let $\Pi' = (\mathsf{Mac}', \mathsf{Vrfy}')$ be a secure fixed-length MAC for messages of length n. Before presenting the construction of a MAC for arbitrary-length messages based on Π', we rule out some simple ideas and describe some canonical attacks that must be prevented. Below, we parse the message m to be authenticated as a sequence of blocks $m_1, \ldots, m_d$; note that, since our aim is to handle messages of arbitrary length, d can vary from message to message.

1. A natural first idea is to simply authenticate each block separately, i.e., compute $t_i := \mathsf{Mac}'_k(m_i)$ for all i, and output $\langle t_1, \ldots, t_d \rangle$ as the tag. This prevents an adversary from sending any previously unauthenticated block without being detected. However, it does not prevent a *block reordering attack* in which the attacker shuffles the order of blocks in an authenticated message. Specifically, if $\langle t_1, t_2 \rangle$ is a valid tag on the message m_1, m_2 (with $m_1 \neq m_2$), then $\langle t_2, t_1 \rangle$ is a valid tag on the (different) message m_2, m_1 (something that is not allowed by Definition 4.2).

2. We can prevent the previous attack by authenticating a block index along with each block. That is, we now compute $t_i = \mathsf{Mac}'_k(i \| m_i)$ for all i, and output $\langle t_1, \ldots, t_d \rangle$ as the tag. (Note that the block length $|m_i|$ will have to change.) This does not prevent a *truncation attack* whereby an attacker simply drops blocks from the end of the message (and drops the corresponding blocks of the tag as well).

3. The truncation attack can be thwarted by additionally authenticating the message length along with each block. (Authenticating the message length as a separate block does not work. Do you see why?) That is, compute $t_i = \mathsf{Mac}'_k(\ell \| i \| m_i)$ for all i, where ℓ denotes the length of the message in bits. (Once again, the block length $|m_i|$ will need to decrease.) This scheme is vulnerable to a *"mix-and-match" attack* where the adversary combines blocks from different messages. For example, if the adversary obtains tags $\langle t_1, \ldots, t_d \rangle$ and $\langle t'_1, \ldots, t'_d \rangle$ on messages $m = m_1, \ldots, m_d$ and $m' = m'_1, \ldots, m'_d$, respectively, it can output the valid tag $\langle t_1, t'_2, t_3, t'_4, \ldots \rangle$ on the message $m_1, m'_2, m_3, m'_4, \ldots$.

We can prevent this last attack by also including a random "message identifier" along with each block that prevents blocks from different messages from being combined. This leads us to Construction 4.7.

with a larger domain would yield a secure MAC for longer messages. However, existing *practical* pseudorandom functions (i.e., block ciphers) take short, fixed-length inputs.

CONSTRUCTION 4.7

Let $\Pi' = (\mathsf{Mac}', \mathsf{Vrfy}')$ be a fixed-length MAC for messages of length n. Define a MAC as follows:

- Mac: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^*$ of (nonzero) length $\ell < 2^{n/4}$, parse m into d blocks $m_1, \ldots, m_d$, each of length $n/4$. (The final block is padded with 0s if necessary.) Choose a uniform identifier $r \in \{0,1\}^{n/4}$.

  For $i = 1, \ldots, d$, compute $t_i \leftarrow \mathsf{Mac}'_k(r\|\ell\|i\|m_i)$, where i, ℓ are encoded as strings of length $n/4$.[†] Output the tag $t := \langle r, t_1, \ldots, t_d \rangle$.
- Vrfy: on input a key $k \in \{0,1\}^n$, a message $m \in \{0,1\}^*$ of length $\ell < 2^{n/4}$, and a tag $t = \langle r, t_1, \ldots, t_{d'} \rangle$, parse m into d blocks $m_1, \ldots, m_d$, each of length $n/4$. (The final block is padded with 0s if necessary.) Output 1 if and only if $d' = d$ and $\mathsf{Vrfy}'_k(r\|\ell\|i\|m_i,\, t_i) = 1$ for $1 \le i \le d$.

[†] Note that i and ℓ can be encoded using $n/4$ bits because $i, \ell < 2^{n/4}$.

A MAC for arbitrary-length messages from any fixed-length MAC.

(Technically, the scheme only handles messages of length less than $2^{n/4}$. Asymptotically, since this is an exponential bound, honest parties will not authenticate messages that long and any polynomial-time adversary could not submit messages that long to its MAC oracle. In practice, when a concrete value of n is fixed, one must ensure that this bound is acceptable.)

THEOREM 4.8 *If Π' is a secure fixed-length MAC for messages of length n, then Construction 4.7 is a secure MAC (for arbitrary-length messages).*

PROOF The intuition is that as long as Π' is secure, an adversary cannot introduce a new block with a valid tag. Furthermore, the extra information included in each block prevents the various attacks (dropping blocks, re-ordering blocks, etc.) sketched earlier. We will prove security by essentially showing that these attacks are the only ones possible.

Let Π be the MAC given by Construction 4.7, and let $\mathcal{A}$ be a probabilistic polynomial-time adversary. We show that $\Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n) = 1]$ is negligible. We first introduce some notation that will be used in the proof. Let Repeat denote the event that the same random identifier appears in two of the tags returned by the MAC oracle in experiment $\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)$. Letting $(m, t = \langle r, t_1, \ldots \rangle)$ denote the final output of $\mathcal{A}$, where $m = m_1, \ldots$ has length ℓ, we let $\mathsf{NewBlock}$ be the event that at least one of the blocks $r\|\ell\|i\|m_i$ was never previously authenticated by Mac' in the course of answering $\mathcal{A}$'s Mac queries. (Note that, by construction of Π, it is easy to tell exactly which blocks are authenticated by Mac'_k when computing $\mathsf{Mac}_k(m)$.) Informally, $\mathsf{NewBlock}$ is the event that $\mathcal{A}$ tries to output a valid tag on a block that was never authenticated by the underlying fixed-length MAC Π'.

We have

$$\begin{aligned}\Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)=1] &= \Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)=1 \wedge \mathsf{Repeat}] \\ &\quad + \Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)=1 \wedge \overline{\mathsf{Repeat}} \wedge \mathsf{NewBlock}] \\ &\quad + \Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)=1 \wedge \overline{\mathsf{Repeat}} \wedge \overline{\mathsf{NewBlock}}] \\ &\le \Pr[\mathsf{Repeat}] \qquad (4.3) \\ &\quad + \Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)=1 \wedge \mathsf{NewBlock}] \\ &\quad + \Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)=1 \wedge \overline{\mathsf{Repeat}} \wedge \overline{\mathsf{NewBlock}}].\end{aligned}$$

We show that the first two terms of Equation (4.3) are negligible, and the final term is 0. This implies $\Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)=1]$ is negligible, as desired.

CLAIM 4.9 $\Pr[\mathsf{Repeat}]$ *is negligible.*

PROOF Let $q(n)$ be the number of MAC oracle queries made by $\mathcal{A}$. To answer the ith oracle query of $\mathcal{A}$, the oracle chooses r_i uniformly from a set of size $2^{n/4}$. The probability of event Repeat is exactly the probability that $r_i = r_j$ for some $i \neq j$. Applying the "birthday bound" (Lemma A.15), we have that $\Pr[\mathsf{Repeat}] \le \frac{q(n)^2}{2^{n/4}}$. Since $\mathcal{A}$ makes only polynomially many queries, this value is negligible. ■

We next consider the final term on the right-hand side of Equation (4.3). We argue that if $\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)=1$, but Repeat did not occur, then it must be the case that $\mathsf{NewBlock}$ occurred. That is, $\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)=1 \wedge \overline{\mathsf{Repeat}}$ implies $\mathsf{NewBlock}$, and so

$$\Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)=1 \wedge \overline{\mathsf{Repeat}} \wedge \overline{\mathsf{NewBlock}}] = 0.$$

This is, in some sense, the heart of the proof.

Again let $q = q(n)$ denote the number of MAC oracle queries made by $\mathcal{A}$, and let r_i denote the random identifier used to answer the ith oracle query of $\mathcal{A}$. If Repeat does not occur then the values $r_1, \ldots, r_q$ are distinct. Let $(m, t = \langle r, t_1, \ldots \rangle)$ be the output of $\mathcal{A}$, with $m = m_1, \ldots$. If $r \notin \{r_1, \ldots, r_q\}$, then $\mathsf{NewBlock}$ clearly occurs. If not, then $r = r_j$ for some unique j, and the blocks $r\|\ell\|1\|m_1, \ldots$ could then not possibly have been authenticated during the course of answering any Mac queries other than the jth such query. Let $m^{(j)}$ be the message that was used by $\mathcal{A}$ for its jth oracle query, and let ℓ_j be its length. There are two cases to consider:

Case 1: $\ell \neq \ell_j$. The blocks authenticated when answering the jth Mac query all have $\ell_j \neq \ell$ in the second position. So $r\|\ell\|1\|m_1$, in particular, was never authenticated in the course of answering the jth Mac query, and $\mathsf{NewBlock}$ occurs.

Case 2: $\ell = \ell_j$**.** If $\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n) = 1$, then we must have $m \neq m^{(j)}$. Let $m^{(j)} = m_1^{(j)}, \ldots$. Since m and $m^{(j)}$ have equal length, there must be at least one index i for which $m_i \neq m_i^{(j)}$. The block $r\|\ell\|i\|m_i$ was then never authenticated in the course of answering the jth Mac query. (Because i is included in the third position of the block, the block $r\|\ell\|i\|m_i$ could only possibly have been authenticated if $r\|\ell\|i\|m_i = r_j\|\ell_j\|i\|m_i^{(j)}$, but this is not true since $m_i \neq m_i^{(j)}$.)

To complete the proof of the theorem, we bound the second term on the right-hand side of Equation (4.3):

CLAIM 4.10 $\Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n) = 1 \wedge \mathsf{NewBlock}]$ *is negligible.*

The claim relies on security of Π'. We construct a PPT adversary $\mathcal{A}'$ who attacks the fixed-length MAC Π' and succeeds in outputting a valid forgery on a previously unauthenticated message with probability

$$\Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A}',\Pi'}(n) = 1] \geq \Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n) = 1 \wedge \mathsf{NewBlock}]. \quad (4.4)$$

Security of Π' means that the left-hand side is negligible, proving the claim.

The construction of $\mathcal{A}'$ is the obvious one and so we describe it briefly. $\mathcal{A}'$ runs $\mathcal{A}$ as a sub-routine, and answers the request by $\mathcal{A}$ for a tag on m by choosing $r \leftarrow \{0,1\}^{n/4}$ itself, parsing m appropriately, and making the necessary queries to its own MAC oracle $\mathsf{Mac}'_k(\cdot)$. When $\mathcal{A}$ outputs $(m, t = \langle r, t_1, \ldots \rangle)$, then $\mathcal{A}'$ checks whether $\mathsf{NewBlock}$ occurs (this is easy to do since $\mathcal{A}'$ can keep track of all the queries it makes to its own oracle). If so, then $\mathcal{A}'$ finds the first block $r\|\ell\|i\|m_i$ that was never previously authenticated by Mac' and outputs $(r\|\ell\|i\|m_i,\, t_i)$. (If not, $\mathcal{A}'$ outputs nothing.)

The view of $\mathcal{A}$ when run as a sub-routine by $\mathcal{A}'$ is distributed identically to the view of $\mathcal{A}$ in experiment $\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)$, and so the probabilities of events $\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n) = 1$ and $\mathsf{NewBlock}$ do not change. If $\mathsf{NewBlock}$ occurs then $\mathcal{A}'$ outputs a block $r\|\ell\|i\|m_i$ that was never previously authenticated by its own MAC oracle; if $\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n) = 1$ then the tag on every block is valid (with respect to Π'), and so in particular this is true for the block output by $\mathcal{A}'$. This means that whenever $\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n) = 1$ and $\mathsf{NewBlock}$ occur we have $\mathsf{Mac\text{-}forge}_{\mathcal{A}',\Pi'}(n) = 1$, proving Equation (4.4). ■

4.4 CBC-MAC

Theorems 4.6 and 4.8 show that it is possible to construct a secure message authentication code for arbitrary-length messages from a pseudorandom function taking inputs of fixed length n. This demonstrates, in principle, that secure MACs can be constructed from block ciphers. Unfortunately, the resulting construction is extremely inefficient: to compute a tag on a message of length dn, the block cipher is evaluated $4d$ times; the tag is more than $4dn$ bits long. Fortunately, far more efficient constructions are available. We explore one such construction here that relies solely on block ciphers, and another in Section 5.3.2 that uses an additional cryptographic primitive.

4.4.1 The Basic Construction

CBC-MAC is a standardized message authentication code used widely in practice. A basic version of CBC-MAC, secure when authenticating messages of any *fixed* length, is given as Construction 4.11. (See also Figure 4.1.) We caution that this basic scheme is *not* secure in the general case when messages of different lengths may be authenticated; see further discussion below.

CONSTRUCTION 4.11

Let F be a pseudorandom function, and fix a length function $\ell > 0$. The basic CBC-MAC construction is as follows:

- Mac: on input a key $k \in \{0,1\}^n$ and a message m of length $\ell(n) \cdot n$, do the following (we set $\ell = \ell(n)$ in what follows):
  1. Parse m as $m = m_1, \ldots, m_\ell$ where each m_i is of length n.
  2. Set $t_0 := 0^n$. Then, for $i = 1$ to ℓ:
     Set $t_i := F_k(t_{i-1} \oplus m_i)$.

  Output t_ℓ as the tag.
- Vrfy: on input a key $k \in \{0,1\}^n$, a message m, and a tag t, do: If m is not of length $\ell(n) \cdot n$ then output 0. Otherwise, output 1 if and only if $t \overset{?}{=} \mathsf{Mac}_k(m)$.

Basic CBC-MAC (for fixed-length messages).

THEOREM 4.12 *Let ℓ be a polynomial. If F is a pseudorandom function, then Construction 4.11 is a secure MAC for messages of length $\ell(n) \cdot n$.*

The proof of Theorem 4.12 is somewhat involved. In the following section we will prove a more general result from which the above theorem follows.

Although Construction 4.11 can be extended in the obvious way to handle messages whose length is an arbitrary multiple of n, the construction is only secure when the length of the messages being authenticated is fixed and agreed upon in advance by the sender and receiver. (See Exercise 4.13.)

The advantage of this construction over Construction 4.5, which also gives a fixed-length MAC, is that the present construction can authenticate longer messages. Compared to Construction 4.7, CBC-MAC is much more efficient, requiring only d block-cipher evaluations for a message of length dn, and with a tag of length n only.

CBC-MAC vs. CBC-mode encryption. CBC-MAC is similar to the CBC mode of operation. There are, however, some important differences:

1. CBC-mode encryption uses a *random IV* and this is crucial for security. In contrast, CBC-MAC uses no IV (alternately, it can be viewed as using the fixed value $IV = 0^n$) and this is also crucial for security. Specifically, CBC-MAC using a random IV is not secure.

2. In CBC-mode encryption all intermediate values t_i (called c_i in the case of CBC-mode encryption) are output by the encryption algorithm as part of the ciphertext, whereas in CBC-MAC only the final block is output as the tag. If CBC-MAC is modified to output all the $\{t_i\}$ obtained during the course of the computation then it is no longer secure.

In Exercise 4.14 you are asked to verify that the modifications of CBC-MAC discussed above are insecure. These examples illustrate the fact that harmless-looking modifications to cryptographic constructions can render them insecure. One should always implement a cryptographic construction exactly as specified and not introduce any variations (unless the variations themselves can be proven secure). Furthermore, it is essential to understand the construction being used. In many cases a cryptographic library provides a programmer with a "CBC function," but does not distinguish between the use of this function for encryption or message authentication.

Secure CBC-MAC for arbitrary-length messages. We briefly describe two ways Construction 4.11 can be modified, in a provably secure fashion, to handle arbitrary-length messages. (Here for simplicity we assume that all messages being authenticated have length a multiple of n, and that Vrfy rejects

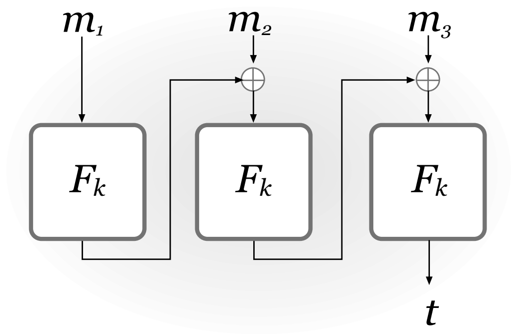

FIGURE 4.1: Basic CBC-MAC (for fixed-length messages).

any message whose length is not a multiple of n. In the following section we treat the more general case where messages can have arbitrary length.)

1. *Prepend* the message m with its length $|m|$ (encoded as an n-bit string), and then compute basic CBC-MAC on the result; see Figure 4.2. Security of this variant follows from the results proved in the next section.

   Note that appending $|m|$ to the *end* of the message and then computing the basic CBC-MAC is *not* secure.

2. Change the scheme so that key generation chooses two independent, uniform keys $k_1 \in \{0,1\}^n$ and $k_2 \in \{0,1\}^n$. Then to authenticate a message m, first compute the basic CBC-MAC of m using k_1 and let t be the result; output the tag $\hat{t} := F_{k_2}(t)$.

The second option has the advantage of not needing to know the message length in advance (i.e., when beginning to compute the tag). However, it has the drawback of using two keys for F. Note that, at the expense of two additional applications of the pseudorandom function, it is possible to store a single key k and then derive keys $k_1 := F_k(1)$ and $k_2 := F_k(2)$ at the beginning of the computation. Despite this, in practice, the operation of initializing a key for a block cipher is considered relatively expensive. Therefore, requiring two different keys—even if they are derived on the fly—is less desirable.

4.4.2 *Proof of Security

In this section we prove security of different variants of CBC-MAC. We begin by summarizing the results, and then give the details of the proof. Before beginning, we remark that the proof in this section is quite involved, and is intended for advanced readers.

Throughout this section, fix a keyed function F that, for security parameter n, maps n-bit keys and n-bit inputs to n-bit outputs. We define a keyed

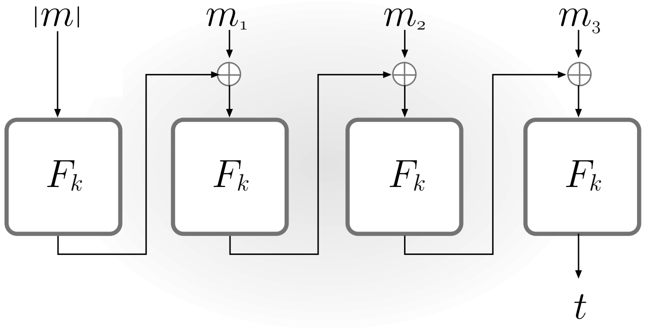

FIGURE 4.2: A variant of CBC-MAC secure for authenticating arbitrary-length messages.

function CBC that, for security parameter n, maps n-bit keys and inputs in $(\{0,1\}^n)^*$ (i.e., strings whose length is a multiple of n) to n-bit outputs. This function is defined as

$$\mathsf{CBC}_k(x_1,\ldots,x_\ell) \stackrel{\text{def}}{=} F_k\left(F_k\left(\cdots F_k(F_k(x_1)\oplus x_2)\oplus\cdots\right)\oplus x_\ell\right),$$

where $|x_1| = \cdots = |x_\ell| = |k| = n$. (We leave CBC_k undefined on the empty string.) Note that CBC is exactly basic CBC-MAC, although here we consider inputs of different lengths.

A set of strings $P \subset (\{0,1\}^n)^*$ is *prefix-free* if it does not contain the empty string, and no string $X \in P$ is a prefix of any other string $X' \in P$. We show:

THEOREM 4.13 *If F is a pseudorandom function, then* CBC *is a pseudorandom function as long as the set of inputs on which it is queried is prefix-free. Formally, for all probabilistic polynomial-time distinguishers D that query their oracle on a prefix-free set of inputs, there is a negligible function* negl *such that*

$$\left|\Pr[D^{\mathsf{CBC}_k(\cdot)}(1^n)=1] - \Pr[D^{f(\cdot)}(1^n)=1]\right| \leq \mathsf{negl}(n),$$

where k is chosen uniformly from $\{0,1\}^n$ and f is chosen uniformly from the set of functions mapping $(\{0,1\}^n)^*$ to $\{0,1\}^n$ (i.e., the value of f at each input is uniform and independent of the values of f at all other inputs).

Thus, we can convert a pseudorandom function F for fixed-length inputs into a pseudorandom function CBC for arbitrary-length inputs (subject to a constraint on which inputs can be queried)! To use this for message authentication, we adapt the idea of Construction 4.5 as follows: to authenticate a message m, first apply some *encoding function* encode to obtain a (nonempty) string $\mathsf{encode}(m) \in (\{0,1\}^n)^*$; then output the tag $\mathsf{CBC}_k(\mathsf{encode}(m))$. For this to be secure (cf. the proof of Theorem 4.6), the encoding needs to be *prefix-free*, namely, to have the property that for any distinct (legal) messages m_1, m_2, the string $\mathsf{encode}(m_1)$ is not a prefix of $\mathsf{encode}(m_2)$. This implies that for any set of (legal) messages $\{m_1,\ldots\}$, the set of encoded messages $\{\mathsf{encode}(m_1),\ldots\}$ is prefix-free.

We now examine two concrete applications of this idea:

- Fix ℓ, and let the set of legal messages be $\{0,1\}^{\ell(n)\cdot n}$. Then we can take the trivial encoding $\mathsf{encode}(m) = m$, which is prefix-free since one string cannot be a prefix of a different string of the same length. This is exactly basic CBC-MAC, and what we have said above implies that basic CBC-MAC is secure for messages of any fixed length (cf. Theorem 4.12).
- One way of handling arbitrary-length (nonempty) messages (technically, messages of length less than 2^n) is to encode a string $m \in \{0,1\}^*$ by prepending its length $|m|$ (encoded as an n-bit string), and then appending as many 0s as needed to make the length of the resulting

string a multiple of n. (This is essentially what is shown in Figure 4.2.) This encoding is prefix-free, and we therefore obtain a secure MAC for arbitrary-length messages.

The rest of this section is devoted to a proof of Theorem 4.13. In proving the theorem, we analyze CBC when it is "keyed" with a *random function g* rather than a random key k for some underlying pseudorandom function F. That is, we consider the keyed function CBC_g defined as

$$\mathsf{CBC}_g(x_1, \ldots, x_\ell) \stackrel{\text{def}}{=} g\left(g\left(\cdots g(g(x_1) \oplus x_2) \oplus \cdots\right) \oplus x_\ell\right)$$

where, for security parameter n, the function g maps n-bit inputs to n-bit outputs, and $|x_1| = \cdots = |x_\ell| = n$. Note that CBC_g as defined here is not efficient (since the representation of g requires space exponential in n); nevertheless, it is still a well-defined, keyed function.

We show that if g is chosen uniformly from Func_n, then CBC_g is indistinguishable from a random function mapping $(\{0,1\}^n)^*$ to n-bit strings, as long as a prefix-free set of inputs is queried. More precisely:

CLAIM 4.14 *Fix any $n \geq 1$. For all distinguishers D that query their oracle on a prefix-free set of q inputs, where the longest such input contains ℓ blocks, it holds that:*

$$\left|\Pr[D^{\mathsf{CBC}_g(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1]\right| \leq \frac{q^2\ell^2}{2^n},$$

where g is chosen uniformly from Func_n, and f is chosen uniformly from the set of functions mapping $(\{0,1\}^n)^*$ to $\{0,1\}^n$.

(The claim is unconditional, and does not impose any constraints on the running time of D. Thus we may take D to be deterministic.) The above implies Theorem 4.13 using standard techniques that we have already seen. In particular, for any D running in polynomial time we must have $q(n), \ell(n) = \mathsf{poly}(n)$ and so $q(n)^2\ell(n)^2 \cdot 2^{-n}$ is negligible.

PROOF (of Claim 4.14) Fix some $n \geq 1$. The proof proceeds in two steps: We first define a notion of *smoothness* and prove that CBC is smooth; we then show that smoothness implies the claim.

Let $P = \{X_1, \ldots, X_q\}$ be a prefix-free set of q inputs, where each X_i is in $(\{0,1\}^n)^*$ and the longest string in P contains ℓ blocks (i.e., each $X_i \in P$ contains at most ℓ blocks of length n). Note that for any $t_1, \ldots, t_q \in \{0,1\}^n$ it holds that $\Pr[\forall i : f(X_i) = t_i] = 2^{-nq}$, where the probability is over uniform choice of the function f from the set of functions mapping $(\{0,1\}^n)^*$ to $\{0,1\}^n$. We say that CBC is (q, ℓ, δ)-*smooth* if for every prefix-free set $P = \{X_1, \ldots, X_q\}$ as above and every $t_1, \ldots, t_q \in \{0,1\}^n$, it holds that

$$\Pr[\forall i : \mathsf{CBC}_g(X_i) = t_i] \geq (1 - \delta) \cdot 2^{-nq},$$

where the probability is over uniform choice of $g \in \mathsf{Func}_n$.

In words, CBC is smooth if for every fixed set of input/output pairs $\{(X_i, t_i)\}$, where the $\{X_i\}$ form a prefix-free set, the probability that $\mathsf{CBC}_g(X_i) = t_i$ for all i is δ-close to the probability that $f(X_i) = t_i$ for all i (where g is a random function from $\{0,1\}^n$ to $\{0,1\}^n$, and f is a random function from $(\{0,1\}^n)^*$ to $\{0,1\}^n$).

CLAIM 4.15 CBC_g *is* (q, ℓ, δ)*-smooth, for* $\delta = q^2\ell^2 \cdot 2^{-n}$.

PROOF For any $X \in (\{0,1\}^n)^*$, with $X = x_1, \ldots$ and $x_i \in \{0,1\}^n$, let $\mathcal{C}_g(X)$ denote the set of inputs on which g is evaluated during the computation of $\mathsf{CBC}_g(X)$; i.e., if $X \in (\{0,1\}^n)^m$ then

$$\mathcal{C}_g(X) \overset{\text{def}}{=} (x_1,\ \mathsf{CBC}_g(x_1) \oplus x_2,\ \ldots,\ \mathsf{CBC}_g(x_1, \ldots, x_{m-1}) \oplus x_m).$$

For $X \in (\{0,1\}^n)^m$ and $X' \in (\{0,1\}^n)^{m'}$, with $\mathcal{C}_g(X) = (I_1, \ldots, I_m)$ and $\mathcal{C}_g(X') = (I'_1, \ldots, I'_{m'})$, say there is a *non-trivial collision in* X if $I_i = I_j$ for some $i \neq j$, and say that there is a *non-trivial collision between* X *and* X' if $I_i = I'_j$ but $(x_1, \ldots, x_i) \neq (x'_1, \ldots, x'_j)$ (in this latter case i may equal j). We say that there is a non-trivial collision in P if there is a non-trivial collision in some $X \in P$ or between some pair of strings $X, X' \in P$. Let Coll be the event that there is a non-trivial collision in P.

We prove the claim in two steps. First, we show that conditioned on there not being a non-trivial collision in P, the probability that $\mathsf{CBC}_g(X_i) = t_i$ for all i is exactly 2^{-nq}. Next, we show that the probability that there is a non-trivial collision in P is less than $\delta = q^2\ell^2 \cdot 2^{-n}$.

Consider choosing a uniform g by choosing, one-by-one, uniform values for the outputs of g on different inputs. Determining whether there is a non-trivial collision between two strings $X, X' \in P$ can be done by first choosing the values of $g(I_1)$ and $g(I'_1)$ (if $I'_1 = I_1$, these values are the same), then choosing values for $g(I_2)$ and $g(I'_2)$ (note that $I_2 = g(I_1) \oplus x_2$ and $I'_2 = g(I'_1) \oplus x'_2$ are defined once $g(I_1), g(I'_1)$ have been fixed), and continuing in this way until we choose values for $g(I_{m-1})$ and $g(I'_{m'-1})$. In particular, the values of $g(I_m), g(I'_{m'})$ need not be chosen in order to determine whether there is a non-trivial collision between X and X'. Continuing this line of reasoning, it is possible to determine whether Coll occurs by choosing the values of g on all but the final entries of each of $\mathcal{C}_g(X_1), \ldots, \mathcal{C}_g(X_q)$.

Assume Coll has not occurred after fixing the values of g on various inputs as described above. Consider the final entries in each of $\mathcal{C}_g(X_1), \ldots, \mathcal{C}_g(X_q)$. These entries are all distinct (this is immediate from the fact that Coll has not occurred), and we claim that the value of g on each of those points has not yet been fixed. Indeed, the only way the value of g could already be fixed on any of those points is if the final entry I_m of some $\mathcal{C}_g(X)$ is equal to a non-final entry I_j of some $\mathcal{C}_g(X')$. But since Coll has not occurred, this can only happen if $X \neq X'$ and $(x'_1, \ldots, x'_j) = (x_1, \ldots, x_m)$. But then X would be a prefix of X', violating the assumption that P is prefix-free.

Since g is a random function, the above means that $\mathsf{CBC}_g(X_1), \ldots, \mathsf{CBC}_g(X_q)$ are *uniform* and *independent* of each other as well as all the other values of g that have already been fixed. (This is because $\mathsf{CBC}_g(X_i)$ is the value of g when evaluated at the final entry of $\mathcal{C}_g(X_i)$, an input value which is different from all the other inputs at which g has already been fixed.) Thus, for any $t_1, \ldots, t_q \in \{0,1\}^n$ we have:

$$\Pr\left[\forall i : \mathsf{CBC}_g(X_i) = t_i \mid \overline{\mathsf{Coll}}\right] = 2^{-nq}. \tag{4.5}$$

We next show that $\overline{\mathsf{Coll}}$ occurs with high probability by upper-bounding $\Pr[\mathsf{Coll}]$. For distinct $X_i, X_j \in P$, let $\mathsf{Coll}_{i,j}$ denote the event that there is a non-trivial collision in X or X', or a non-trivial collision between X and X'. We have $\mathsf{Coll} = \bigvee_{i,j} \mathsf{Coll}_{i,j}$ and so a union bound gives

$$\Pr[\mathsf{Coll}] \leq \sum_{i,j:\, i<j} \Pr[\mathsf{Coll}_{i,j}] = \binom{q}{2} \cdot \Pr[\mathsf{Coll}_{i,j}] \leq \frac{q^2}{2} \cdot \Pr[\mathsf{Coll}_{i,j}]. \tag{4.6}$$

Fixing distinct $X = X_i$ and $X' = X_j$ in P, we now bound $\mathsf{Coll}_{i,j}$. As will be clear from the analysis, the probability is maximized when X and X' are both as long as possible, and thus we assume they are each ℓ blocks long. Let $X = (x_1, \ldots, x_\ell)$ and $X' = (x'_1, \ldots, x'_\ell)$, and let t be the largest integer such that $(x_1, \ldots, x_t) = (x'_1, \ldots, x'_t)$. (Note that $t < \ell$ or else $X = X'$.) We assume $t > 0$, but the analysis below can be easily modified, giving the same result, if $t = 0$. We continue to let $I_1, I_2, \ldots$ (resp., $I'_1, I'_2, \ldots$) denote the inputs to g during the course of computing $\mathsf{CBC}_g(X)$ (resp., $\mathsf{CBC}_g(X')$); note that $(I'_1, \ldots, I'_t) = (I_1, \ldots, I_t)$. Consider choosing g by choosing uniform values for the outputs of g, one-by-one. We do this in $2\ell - 2$ steps as follows:

Steps 1 through $t-1$ (if $t > 1$): In each step i, choose a uniform value for $g(I_i)$, thus defining I_{i+1} and I'_{i+1} (which are equal).

Step t: Choose a uniform value for $g(I_t)$, thus defining I_{t+1} and I'_{t+1}.

Steps $t+1$ to $\ell - 1$ (if $t < \ell - 1$): Choose, in turn, uniform values for each of $g(I_{t+1})$, $g(I_{t+2})$, $\ldots$, $g(I_{\ell-1})$, thus defining I_{t+2}, I_{t+3}, $\ldots$, I_ℓ.

Steps ℓ to $2\ell - 2$ (if $t < \ell - 1$): Choose, in turn, uniform values for each of $g(I'_{t+1})$, $g(I'_{t+2})$, $\ldots$, $g(I'_{\ell-1})$, thus defining I'_{t+2}, I'_{t+3}, $\ldots$, I'_ℓ.

Let $\mathsf{Coll}(k)$ be the event that a non-trivial collision occurs by step k. Then

$$\Pr[\mathsf{Coll}_{i,j}] = \Pr\left[\bigvee_k \mathsf{Coll}(k)\right] \leq \Pr[\mathsf{Coll}(1)] + \sum_{k=2}^{2\ell-2} \Pr[\mathsf{Coll}(k) \mid \overline{\mathsf{Coll}(k-1)}], \tag{4.7}$$

using Proposition A.9. For $k < t$, we claim $\Pr[\mathsf{Coll}(k) \mid \overline{\mathsf{Coll}(k-1)}] = k/2^n$: indeed, if no non-trivial collision has occurred by step $k-1$, the value of

$g(I_k)$ is chosen uniformly in step k; a non-trivial collision occurs only if it happens that $I_{k+1} = g(I_k) \oplus x_{k+1}$ is equal to one of $\{I_1, \ldots, I_k\}$ (which are all distinct, since $\mathsf{Coll}(k-1)$ has not occurred). By similar reasoning, we have $\Pr[\mathsf{Coll}(t) \mid \overline{\mathsf{Coll}(t-1)}] \leq 2t/2^n$ (here there are two values I_{t+1}, I'_{t+1} to consider; note that they cannot be equal to each other). Finally, arguing as before, for $k > t$ we have $\Pr[\mathsf{Coll}(k) \mid \overline{\mathsf{Coll}(k-1)}] = (k+1)/2^n$. Using Equation (4.7), we thus have

$$\begin{aligned}\Pr[\mathsf{Coll}_{i,j}] &\leq 2^{-n} \cdot \left(\sum_{k=1}^{t-1} k + 2t + \sum_{k=t+1}^{2\ell-2} (k+1) \right) \\ &= 2^{-n} \cdot \sum_{k=2}^{2\ell-1} k \;=\; 2^{-n} \cdot (2\ell+1) \cdot (\ell-1) \;<\; 2\ell^2 \cdot 2^{-n}.\end{aligned}$$

From Equation (4.6) we get $\Pr[\mathsf{Coll}] < q^2\ell^2 \cdot 2^{-n} = \delta$. Finally, using Equation (4.5) we see that

$$\begin{aligned}\Pr\left[\forall i : \mathsf{CBC}_g(X_i) = t_i\right] &\geq \Pr\left[\forall i : \mathsf{CBC}_g(X_i) = t_i \mid \overline{\mathsf{Coll}}\right] \cdot \Pr[\overline{\mathsf{Coll}}] \\ &= 2^{-nq} \cdot \Pr[\overline{\mathsf{Coll}}] \;\geq (1-\delta) \cdot 2^{-nq},\end{aligned}$$

as claimed. ■

We now show that smoothness implies the theorem. Assume without loss of generality that D always makes q (distinct) queries, each containing at most ℓ blocks. D may choose its queries adaptively (i.e., depending on the answers to previous queries), but the set of D's queries must be prefix-free.

For distinct $X_1, \ldots, X_q \in (\{0,1\}^n)^*$ and arbitrary $t_1, \ldots, t_q \in \{0,1\}^n$, define $\alpha(X_1, \ldots, X_q; t_1, \ldots, t_q)$ to be 1 if and only if D outputs 1 when making queries $X_1, \ldots, X_q$ and getting responses $t_1, \ldots, t_q$. (If, say, D does not make query X_1 as its first query, then $\alpha(X_1, \ldots; \ldots) = 0$.) Letting $\vec{X} = (X_1, \ldots, X_q)$ and $\vec{t} = (t_1, \ldots, t_q)$, we then have

$$\begin{aligned}\Pr[D^{\mathsf{CBC}_g(\cdot)}(1^n) = 1] &= \sum_{\vec{X} \text{ prefix-free}; \vec{t}} \alpha(\vec{X}, \vec{t}) \cdot \Pr[\forall i : \mathsf{CBC}_g(X_i) = t_i] \\ &\geq \sum_{\vec{X} \text{ prefix-free}; \vec{t}} \alpha(\vec{X}, \vec{t}) \cdot (1-\delta) \cdot \Pr[\forall i : f(X_i) = t_i] \\ &= (1-\delta) \cdot \Pr[D^{f(\cdot)}(1^n) = 1],\end{aligned}$$

where, above, g is chosen uniformly from Func_n, and f is chosen uniformly from the set of functions mapping $(\{0,1\}^n)^*$ to $\{0,1\}^n$. This implies

$$\Pr[D^{f(\cdot)}(1^n) = 1] - \Pr[D^{\mathsf{CBC}_g(\cdot)}(1^n) = 1] \leq \delta \cdot \Pr[D^{f(\cdot)}(1^n) = 1] \leq \delta.$$

A symmetric argument for when D outputs 0 completes the proof. ■

4.5 Authenticated Encryption

In Chapter 3, we studied how it is possible to obtain *secrecy* in the private-key setting using encryption. In this chapter, we have shown how to ensure *integrity* using message authentication codes. One might naturally want to achieve both goals simultaneously, and this is the problem we turn to now.

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.

4.5.1 Definitions

We begin, as usual, by defining precisely what we wish to achieve. At an abstract level, our goal is to realize an "ideally secure" communication channel that provides both secrecy and integrity. Pursuing a definition of this sort is beyond the scope of this book. Instead, we provide a simpler set of definitions that treat secrecy and integrity separately. These definitions and our subsequent analysis suffice for understanding the key issues at hand. (We caution the reader, however, that—in contrast to encryption and message authentication codes—the field has not yet settled on standard terminology and definitions for authenticated encryption.)

Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a private-key encryption scheme. As mentioned already, we define security by separately defining secrecy and integrity. The notion of secrecy we consider is one we have seen before: we require that Π be secure against chosen-ciphertext attacks, i.e., that it be CCA-secure. (Refer to Section 3.7 for a discussion and definition of CCA-security.) We are concerned about chosen-ciphertext attacks here because we are explicitly considering an active adversary who can modify the data sent from one honest party to the other. Our notion of integrity will be essentially that of existential unforgeability under an adaptive chosen-message attack. Since Π does not satisfy the syntax of a message authentication code, however, we introduce a definition specific to this case. 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. *Run* $\mathsf{Gen}(1^n)$ *to obtain a key* k*.*
2. *The adversary* $\mathcal{A}$ *is given input* 1^n *and access to an encryption oracle* $\mathsf{Enc}_k(\cdot)$*. The adversary outputs a ciphertext* c*.*
3. *Let* $m := \mathsf{Dec}_k(c)$*, and let* $\mathcal{Q}$ *denote the set of all queries that* $\mathcal{A}$ *asked its encryption oracle. The output of the experiment is* 1 *if and only if (1)* $m \neq \perp$ *and (2)* $m \notin \mathcal{Q}$*.*

DEFINITION 4.16 *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:*

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

Paralleling our discussion about verification queries following Definition 4.2, here one could also consider a stronger definition in which $\mathcal{A}$ is additionally given access to a decryption oracle. One can verify that the secure construction we present below also satisfies that stronger definition.

We now define a (secure) *authenticated encryption* scheme.

DEFINITION 4.17 *A private-key encryption scheme is an* authenticated encryption scheme *if it is CCA-secure and unforgeable.*

4.5.2 Generic Constructions

It may be tempting to think that any reasonable combination of a 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 message authentication code, 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 keys[4] k_E and k_M for Π_E and Π_M, respectively:

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

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

2. *Authenticate-then-encrypt:* Here a MAC 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:

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

The receiver decrypts c to obtain $m\|t$; assuming no error occurs, 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 MAC tag is computed over the result. That is, the ciphertext is the pair $\langle c, t\rangle$ where:

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

(See also Construction 4.18.) 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., an *arbitrary* CPA-secure encryption scheme and an *arbitrary* (strongly) secure MAC. We want an approach that provides joint secrecy and integrity when using *any* (secure) components, and we will therefore reject as "unsafe" any approach for which there exists even a single counterexample of a secure encryption scheme/MAC for which the combination is insecure. This "all-or-nothing" approach reduces the likelihood of implementation flaws. Specifically, an authenticated encryption scheme 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.) Implementing an approach whose security depends on how its 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 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 transmitted value is $\langle c, t\rangle$ where

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

This approach may not achieve even the most basic level of secrecy. To see this, note that a secure MAC does not guarantee *any* secrecy and so it is possible for the tag $\mathsf{Mac}_{k_M}(m)$ to leak information about m to an eavesdropper.

(As a trivial example, consider a 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 does not even have indistinguishable encryptions in the presence of an eavesdropper.

In fact, the encrypt-and-authenticate approach is likely to be insecure against chosen-plaintext attacks even when instantiated with standard components (unlike the contrived counterexample 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, and so the scheme is not CPA-secure. Most MACs used in practice are deterministic, so this represents a real concern.

Authenticate-then-encrypt. Here, a MAC tag $t \leftarrow \mathsf{Mac}_{k_M}(m)$ is first computed; then $m \| t$ is encrypted and the resulting value $\mathsf{Enc}_{k_E}(m \| t)$ is transmitted. We show that this combination also does not necessarily yield an authenticated encryption scheme.

Actually, we have already encountered a CPA-secure encryption scheme for which this approach is insecure: the CBC-mode-with-padding scheme discussed in Section 3.7.2. (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 authenticate-then-encrypt, this means there are now *two* sources of potential decryption failure: the padding may be incorrect, or the MAC 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, return "bad padding" and stop.
    2. 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 same chosen-ciphertext attack described in Section 3.7.2 to the above scheme to recover the entire original plaintext from a given ciphertext. (This is due to the fact that the padding-oracle attack shown in Section 3.7.2 relies only on the ability to learn whether or not there was a padding error, something that is revealed by this scheme.) This type of attack has been carried out successfully in the real world in various settings, e.g., in configurations of IPsec that use authenticate-then-encrypt.

One way to fix the above scheme 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 be used by an adversary to distinguish between them (cf. our earlier discussion of timing attacks at the end of Section 4.2). Some versions of SSL tried using only a single error message in conjunction with an authenticate-then-encrypt approach, but a padding-oracle attack was still successfully carried out using timing information of this sort. We conclude that authenticate-then-encrypt does not provide authenticated encryption in general, and should not be used.

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

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

Decryption of $\langle c, t\rangle$ is done by outputting $\perp$ if $\mathsf{Vrfy}_{k_M}(c,t) \neq 1$, and otherwise outputting $\mathsf{Dec}_{k_E}(c)$. See Construction 4.18 for a formal description.

CONSTRUCTION 4.18

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 whether $\mathsf{Vrfy}_{k_M}(c,t) \overset{?}{=} 1$. If yes, then output $\mathsf{Dec}_{k_E}(c)$; if no, then output $\perp$.

A generic construction of an authenticated encryption scheme.

This approach *is* sound, as long as the MAC is *strongly secure*, as in Definition 4.3. As intuition for the security of this approach, say a ciphertext $\langle c, t\rangle$ is *valid* if t is a valid MAC 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 4.18 is unforgeable. As for CCA-security, the MAC computed over the ciphertext has the effect of rendering the decryption oracle useless since 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 own encryption oracle) or else can expect the result to be an error (since the adversary cannot generate any new, valid ciphertexts). This means that CCA-security of the combined scheme reduces to the CPA-security of Π_E. Observe also that the MAC is verified before decryption takes place; thus, MAC verification cannot leak anything about the plaintext (in contrast to the padding-oracle attack we saw for the authenticate-then-encrypt approach). We now formalize the above arguments.

THEOREM 4.19 *Let Π_E be a CPA-secure private-key encryption scheme, and let Π_M be a strongly secure message authentication code. Then Construction 4.18 is an authenticated encryption scheme.*

PROOF Let Π' denote the scheme resulting from Construction 4.18. We need to show that Π' is unforgeable, and that it is CCA-secure. Following the intuition given above, say a ciphertext $\langle c, t\rangle$ is *valid* (with respect to some fixed secret key (k_E, k_M)) if $\mathsf{Vrfy}_{k_M}(c, t) = 1$. We show that strong security of Π_M implies that (except with negligible probability) any "new" ciphertexts the adversary submits to the decryption oracle will be invalid. As discussed already, this immediately implies unforgeability. (In fact, it is stronger than unforgeability.) This fact also renders the decryption oracle useless, and means that CCA-security of $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ reduces to the CPA-security of Π_E.

In more detail, let $\mathcal{A}$ be a probabilistic polynomial-time adversary attacking Construction 4.18 in a chosen-ciphertext attack (cf. Definition 3.33). Say a ciphertext $\langle c, t\rangle$ is *new* if $\mathcal{A}$ did not receive $\langle c, t\rangle$ from its encryption oracle or as the challenge ciphertext. Let $\mathsf{ValidQuery}$ be the event that $\mathcal{A}$ submits a new ciphertext $\langle c, t\rangle$ to its decryption oracle which is valid, i.e., for which $\mathsf{Vrfy}_{k_M}(c, t) = 1$. We prove:

CLAIM 4.20 $\Pr[\mathsf{ValidQuery}]$ *is negligible.*

PROOF Intuitively, this is due to the fact that if $\mathsf{ValidQuery}$ occurs then the adversary has forged a new, valid pair (c, t) in the $\mathsf{Mac\text{-}sforge}$ experiment. Formally, 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 (i.e., running in experiment $\mathsf{Mac\text{-}sforge}_{\mathcal{A}_M,\Pi_M}(n)$):

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). 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 $\mathcal{A}$ will be the first new, valid query $\mathcal{A}$ makes. In that case, $\mathcal{A}_M$ outputs a valid forgery on a message c that it had never previously submitted to its own MAC oracle.

Clearly $\mathcal{A}_M$ runs in probabilistic polynomial time. We now analyze the probability that $\mathcal{A}_M$ produces a good forgery. 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 the encryption-oracle queries of $\mathcal{A}$ (as well as computation of 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 the decryption-oracle query is indeed $\perp$. (Note that case (i) is exactly the case that $\langle c, t \rangle$ is not new, and case (ii) is exactly the case that $\langle c, t \rangle$ is new.) Recall 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 $\mathsf{Mac\text{-}forge}_{\mathcal{A}_M,\Pi_M}(n)$ 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 when $\mathsf{ValidQuery}$ occurs, then $\mathcal{A}_M$ 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 4.20 to prove security of Π'. The easier case is to prove that Π' is unforgeable. This follows immediately from the claim, and so we just provide informal reasoning rather than a formal proof. Observe first that the adversary $\mathcal{A}'$ 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). When $\mathcal{A}'$ outputs a ciphertext $\langle c, t\rangle$ at the end of its experiment, it "succeeds" only if $\langle c, t\rangle$ is valid and new. But the previous claim shows precisely that the probability of such an event 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

$$\begin{aligned}&\Pr[\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi'}(n) = 1]\\ &\qquad \leq \Pr[\mathsf{ValidQuery}] + \Pr[\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi'}(n) = 1 \wedge \overline{\mathsf{ValidQuery}}]. \qquad (4.8)\end{aligned}$$

We have already shown that $\Pr[\mathsf{ValidQuery}]$ is negligible. The following claim thus completes the proof of the theorem.

CLAIM 4.21 *There exists 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 claim, 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 these same messages and receive a challenge ciphertext c in response. Compute $t \leftarrow \mathsf{Mac}_{k_M}(c)$, and return $\langle c, t \rangle$ as the challenge ciphertext for $\mathcal{A}$. 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 when $\mathsf{ValidQuery}$ does not occur is the same as the probability that $\mathcal{A}$ succeeds when $\mathsf{ValidQuery}$ does not occur; i.e.,

$$\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A}_E,\Pi_E}(n) = 1 \wedge \overline{\mathsf{ValidQuery}}] = \Pr[\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi'}(n) = 1 \wedge \overline{\mathsf{ValidQuery}}],$$

implying that

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

Since Π_E is CPA-secure, there exists a negligible function negl such that $\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A}_E,\Pi_E}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n)$. This proves the claim. ■

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 here, consider what can happen to the encrypt-then-authenticate methodology when 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 $r \in \{0,1\}^{n/2}$, and define $\mathsf{Mac}_k(c) = F_k^{-1}(c)$. It can be shown that this encryption scheme is CPA-secure (in fact, it is even CCA-secure; see Exercise 4.25), and we know that the given message authentication code is a secure MAC. However, the encrypt-then-authenticate combination using the same key k as applied to the message m yields:

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

and the message m is revealed in the clear! This does not in any way contradict Theorem 4.19, since Construction 4.18 expressly requires that k_M, k_E are chosen (uniformly and) independently. We encourage the reader to examine where this independence is used in the proof of Theorem 4.19.

4.5.3 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, it 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, does not suffice, as there are 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. This causes a mismatch between the two parties' views of their communication session.

Replay attack An attacker can *replay* a (valid) ciphertext c sent previously by one of the parties. Again, this causes a mismatch between what is sent by one party and received by the other.

Reflection attack An attacker can take a ciphertext c sent from A to B and send it back to A. This again can cause a mismatch between the two parties' transcripts of their communication session: A may output a message m, even though B never sent such a message.

Fortunately, the above attacks are easy to prevent using *counters* to address the first two and a *directionality bit* to prevent the third.[5] 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.)

[5]In practice, the issue of directionality is 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 a different 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.

We remark that since the parties are anyway maintaining state (namely, the counters $\mathsf{ctr}_{A,B}$ and $\mathsf{ctr}_{B,A}$), the parties could easily use a *stateful* authenticated encryption scheme Π.

4.5.4 CCA-Secure Encryption

It follows directly from the definition that any authenticated encryption scheme is also secure against chosen-ciphertext attacks. Can there be CCA-secure private-key encryption schemes that are *not* unforgeable? Indeed, there are; see Exercise 4.25.

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 embedded in the token is a long-term key k. The server can upload a fresh, short-term key k' to this token by giving the user $\mathsf{Enc}_k(k')$; the user is supposed to give this ciphertext to the token, which will decrypt it and use k' for the next time period. A chosen-ciphertext attack in this setting could allow the user to learn k', something the user is not supposed to be able to do. (Note that here a padding-oracle attack, which only relies on the user's ability to determine whether a decryption failure occurs, could potentially be carried out rather easily.) On the other hand, not much harm is done if the user can generate a "valid" ciphertext that causes the token to use an arbitrary (unrelated) key k'' for the next time period. (Of course, this depends on what the token does with this short-term key.)

Notwithstanding the above, for private-key encryption most "natural" constructions of CCA-secure schemes that we know anyway satisfy the stronger definition of authenticated encryption. Put differently, there is no real reason to ever use a CCA-secure scheme that is *not* an authenticated encryption scheme, simply because we don't really have any constructions satisfying the former that are more efficient than constructions achieving the latter.

From a *conceptual* point of view, however, it is important to keep the notions of CCA-security and authenticated encryption distinct. With regard to CCA-security we are not interested in message integrity *per se*; rather, we wish to ensure privacy even against an active adversary who can interfere with the communication as it goes from sender to receiver. In contrast, with regard to authenticated encryption we are interested in the twin goals of secrecy and integrity. We stress this here because in the public-key setting that we study later in the book, the difference between authenticated encryption and CCA-security is more pronounced.

4.6 *Information-Theoretic MACs

In previous sections we have explored message authentication codes with *computational* security, i.e., where bounds on the attacker's running time are assumed. Recalling the results of Chapter 2, it is natural to ask whether message authentication in the presence of an *unbounded* adversary is possible. In this section, we show under which conditions *information-theoretic* (as opposed to computational) security is attainable.

A first observation is that it is impossible to achieve "perfect" security in this context: namely, we cannot hope to have a message authentication code for which the probability that an adversary outputs a valid tag on a previously unauthenticated message is 0. The reason is that an adversary can simply guess a valid tag t on any message and the guess will be correct with probability (at least) $1/2^{|t|}$, where $|t|$ denotes the tag length of the scheme.

The above example tells us what we *can* hope to achieve: a MAC with tags of length $|t|$ where the probability of forgery is at most $1/2^{|t|}$, even for unbounded adversaries. We will see that this is achievable, but only under restrictions on how many messages are authenticated by the honest parties.

We first define information-theoretic security for message authentication codes. A starting point is to take experiment $\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)$ that is used to define security for computationally secure MACs (cf. Definition 4.2), but to drop the security parameter n and require that $\Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi} = 1]$ should be "small" for *all* adversaries $\mathcal{A}$ (and not just adversaries running in polynomial time). As mentioned above (and as will be proved formally in Section 4.6.2), however, such a definition is impossible to achieve. Rather, information-theoretic security can be achieved only if we place some bound on the number of messages authenticated by the honest parties. We look here at the most basic setting, where the parties authenticate just a *single* message. We refer to this as *one-time message authentication.* The following experiment modifies $\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(n)$ following the above discussion:

The one-time message authentication experiment $\mathsf{Mac\text{-}forge}^{\mathsf{1\text{-}time}}_{\mathcal{A},\Pi}$**:**

1. *A key k is generated by running* Gen*.*
2. *The adversary $\mathcal{A}$ outputs a message m', and is given in return a tag $t' \leftarrow \mathsf{Mac}_k(m')$.*
3. *$\mathcal{A}$ outputs (m,t).*
4. *The output of the experiment is defined to be* 1 *if and only if (1)* $\mathsf{Vrfy}_k(m,t) = 1$ *and (2) $m \neq m'$.*

DEFINITION 4.22 *A message authentication code* $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *is* one-time ε-secure*, or just* ε-secure*, if for all (even unbounded) adversaries $\mathcal{A}$:*

$$\Pr[\mathsf{Mac\text{-}forge}^{\mathsf{1\text{-}time}}_{\mathcal{A},\Pi} = 1] \leq \varepsilon.$$

4.6.1 Constructing Information-Theoretic MACs

In this section we show how to build an ε-secure MAC based on any *strongly universal function*.[6] We then show a simple construction of the latter.

Let $h : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ be a keyed function whose first input is a key $k \in \mathcal{K}$ and whose second input is taken from some domain $\mathcal{M}$. As usual, we write $h_k(m)$ instead of $h(k, m)$. Then h is *strongly universal* (or *pairwise-independent*) if for any two distinct inputs m, m' the values $h_k(m)$ and $h_k(m')$ are uniformly and independently distributed in $\mathcal{T}$ when k is a uniform key. This is equivalent to saying that the probability that $h_k(m), h_k(m')$ take on any particular values t, t' is exactly $1/|\mathcal{T}|^2$. That is:

DEFINITION 4.23 *A function $h : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ is* strongly universal *if for all distinct $m, m' \in \mathcal{M}$ and all $t, t' \in \mathcal{T}$ it holds that*

$$\Pr[h_k(m) = t \wedge h_k(m') = t'] = \frac{1}{|\mathcal{T}|^2},$$

where the probability is taken over uniform choice of $k \in \mathcal{K}$.

The above should motivate the construction of a one-time message authentication code from any strongly universal function h. The tag t on a message m is obtained by computing $h_k(m)$, where the key k is uniform; see Construction 4.24. Intuitively, even after an adversary observes the tag $t' := h_k(m')$ for any message m', the correct tag $h_k(m)$ for any *other* message m is still uniformly distributed in $\mathcal{T}$ from the adversary's point of view. Thus, the adversary can do nothing more than blindly guess the tag, and this guess will be correct only with probability $1/|\mathcal{T}|$.

CONSTRUCTION 4.24

Let $h : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ be a strongly universal function. Define a MAC for messages in $\mathcal{M}$ as follows:

- Gen: choose uniform $k \in \mathcal{K}$ and output it as the key.
- Mac: on input a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$, output the tag $t := h_k(m)$.
- Vrfy: on input a key $k \in \mathcal{K}$, a message $m \in \mathcal{M}$, and a tag $t \in \mathcal{T}$, output 1 if and only if $t \overset{?}{=} h_k(m)$. (If $m \notin \mathcal{M}$, then output 0.)

A MAC from any strongly universal function.

[6]These are often called strongly universal *hash* functions, but in cryptographic contexts the term "hash" has another meaning that we will see later in the book.

The above construction can be viewed as analogous to Construction 4.5. This is because a strongly universal function h is identical to a random function, as long as it is only evaluated twice.

THEOREM 4.25 *Let $h : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ be a strongly universal function. Then Construction 4.24 is a $1/|\mathcal{T}|$-secure MAC for messages in $\mathcal{M}$.*

PROOF Let $\mathcal{A}$ be an adversary. As usual in the information-theoretic setting, we may assume $\mathcal{A}$ is deterministic without loss of generality. So the message m' that $\mathcal{A}$ outputs at the outset of the experiment is fixed. Furthermore, the pair (m, t) that $\mathcal{A}$ outputs at the end of the experiment is a deterministic function of the tag t' on m' that $\mathcal{A}$ receives. We thus have

$$\begin{aligned}\Pr[\mathsf{Mac\text{-}forge}^{\mathsf{1\text{-}time}}_{\mathcal{A},\Pi} = 1] &= \sum_{t' \in \mathcal{T}} \Pr[\mathsf{Mac\text{-}forge}^{\mathsf{1\text{-}time}}_{\mathcal{A},\Pi} = 1 \ \wedge\ h_k(m') = t'] \\ &= \sum_{\substack{t' \in \mathcal{T} \\ (m,t) := \mathcal{A}(t')}} \Pr[h_k(m) = t \ \wedge\ h_k(m') = t'] \\ &= \sum_{\substack{t' \in \mathcal{T} \\ (m,t) := \mathcal{A}(t')}} \frac{1}{|\mathcal{T}|^2} = \frac{1}{|\mathcal{T}|}.\end{aligned}$$

This proves the theorem. ∎

We now turn to a classical construction of a strongly universal function. We assume some basic knowledge about arithmetic modulo a prime number; readers may refer to Sections 8.1.1 and 8.1.2 for the necessary background. Fix some prime p, and let $\mathbb{Z}_p \stackrel{\text{def}}{=} \{0, \ldots, p-1\}$. We take as our message space $\mathcal{M} = \mathbb{Z}_p$; the space of possible tags will also be $\mathcal{T} = \mathbb{Z}_p$. A key (a, b) consists of a pair of elements from $\mathbb{Z}_p$; thus, $\mathcal{K} = \mathbb{Z}_p \times \mathbb{Z}_p$. Define h as

$$h_{a,b}(m) \stackrel{\text{def}}{=} [a \cdot m + b \bmod p],$$

where the notation $[X \bmod p]$ refers to the reduction of the integer X modulo p (and so $[X \bmod p] \in \mathbb{Z}_p$ always).

THEOREM 4.26 *For any prime p, the function h is strongly universal.*

PROOF Fix any distinct $m, m' \in \mathbb{Z}_p$ and any $t, t' \in \mathbb{Z}_p$. For which keys (a, b) does it hold that both $h_{a,b}(m) = t$ and $h_{a,b}(m') = t'$? This holds only if

$$a \cdot m + b = t \bmod p \quad \text{and} \quad a \cdot m' + b = t' \bmod p.$$

We thus have two linear equations in the two unknowns a, b. These two equations are both satisfied exactly when $a = [(t - t') \cdot (m - m')^{-1} \bmod p]$ and $b = [t - a \cdot m \bmod p]$; note that $[(m - m')^{-1} \bmod p]$ exists because $m \neq m'$ and so $m - m' \neq 0 \bmod p$. Restated, this means that for any m, m', t, t' as above there is a *unique* key (a, b) with $h_{a,b}(m) = t$ and $h_{a,b}(m') = t'$. Since there are $|\mathcal{T}|^2$ keys, we conclude that the probability (over choice of the key) that $h_{a,b}(m) = t$ and $h_{a,b}(m') = t'$ is exactly $1/|\mathcal{K}| = 1/|\mathcal{T}|^2$ as required. ■

Parameters of Construction 4.24. We briefly discuss the parameters of Construction 4.24 when instantiated with the strongly universal function described above, ignoring the fact that p is not a power of 2. The construction is a $1/|\mathcal{T}|$-secure MAC with tags of length $\log|\mathcal{T}|$; the tag length is optimal for the level of security achieved.

Let $\mathcal{M}$ be some fixed message space for which we want to construct a one-time secure MAC. The construction above gives a $1/|\mathcal{M}|$-secure MAC with keys that are twice the length of the messages. The reader may notice two problems here, at opposite ends of the spectrum: First, if $|\mathcal{M}|$ is small then a $1/|\mathcal{M}|$ probability of forgery may be unacceptably large. On the flip side, if $|\mathcal{M}|$ is large then a $1/|\mathcal{M}|$ probability of forgery may be overkill; one might be willing to accept a (somewhat) larger probability of forgery if that level of security can be achieved with shorter keys. The first problem (when $|\mathcal{M}|$ is small) is easy to deal with by simply embedding $\mathcal{M}$ into a larger message space $\mathcal{M}'$ by, for example, padding messages with 0s. The second problem can be addressed as well; see the references at the end of this chapter.

4.6.2 Limitations on Information-Theoretic MACs

In this section we explore limitations on information-theoretic message authentication. We show that any 2^{-n}-secure MAC must have keys of length at least $2n$. An extension of the proof shows that any ℓ-time 2^{-n}-secure MAC (where security is defined via a natural modification of Definition 4.23) requires keys of length at least $(\ell + 1) \cdot n$. A corollary is that *no* MAC with bounded-length keys can provide information-theoretic security when authenticating an unbounded number of messages.

In the following, we assume the message space contains at least two messages; if not, there is no point in communicating, let alone authenticating.

THEOREM 4.27 *Let* $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *be a* 2^{-n}*-secure MAC where all keys output by* Gen *are the same length. Then the keys output by* Gen *must have length at least* $2n$.

PROOF Fix two distinct messages m_0, m_1 in the message space. The intuition for the proof is that there must be at least 2^n possibilities for the tag of m_0 (or else the adversary could guess it with probability better than 2^{-n}); furthermore, even conditioned on the value of the tag for m_0, there must be 2^n possibilities for the tag of m_1 (or else the adversary could forge a tag on m_1 with probability better than 2^{-n}). Since each key defines tags for m_0 and m_1, this means there must be at least $2^n \times 2^n$ keys. We make this formal below.

Let $\mathcal{K}$ denote the key space (i.e., the set of all possible keys that can be output by Gen). For any possible tag t_0, let $\mathcal{K}(t_0)$ denote the set of keys for which t_0 is a valid tag on m_0; i.e.,

$$\mathcal{K}(t_0) \overset{\text{def}}{=} \{k \mid \mathsf{Vrfy}_k(m_0, t_0) = 1\}.$$

For any t_0 we must have $|\mathcal{K}(t_0)| \leq 2^{-n} \cdot |\mathcal{K}|$. Otherwise the adversary could simply output (m_0, t_0) as its forgery; this would be a valid forgery with probability at least $|\mathcal{K}(t_0)|/|\mathcal{K}| > 2^{-n}$, contradicting the claimed security.

Consider now the adversary $\mathcal{A}$ who requests a tag on the message m_0, receives in return a tag t_0, chooses a uniform key $k \in \mathcal{K}(t_0)$, and outputs $(m_1, \mathsf{Mac}_k(m_1))$ as its forgery. The probability that $\mathcal{A}$ outputs a valid forgery is at least

$$\begin{aligned}\sum_{t_0} \Pr[\mathsf{Mac}_k(m_0) = t_0] \cdot \frac{1}{|\mathcal{K}(m_0, t_0)|} &\geq \sum_{t_0} \Pr[\mathsf{Mac}_k(m_0) = t_0] \cdot \frac{2^n}{|\mathcal{K}|} \\ &= \frac{2^n}{|\mathcal{K}|}.\end{aligned}$$

By the claimed security of the scheme, the probability that the adversary can output a valid forgery is at most 2^{-n}. Thus, we must have $|\mathcal{K}| \geq 2^{2n}$. Since all keys have the same length, each key must have length at least $2n$. ■

References and Additional Reading

The definition of security for message authentication codes was adapted by Bellare et al. [18] from the definition of security for digital signatures given by Goldwasser et al. [81] (see Chapter 12). For more on the definitional variant where verification queries are allowed, see [17].

The paradigm of using pseudorandom functions for message authentication (as in Construction 4.5) was introduced by Goldreich et al. [77]. Construction 4.7 is due to Goldreich [76].

CBC-MAC was standardized in the early 1980s [94, 178] and is still used widely today. Basic CBC-MAC was proven secure (for authenticating fixed-length messages) by Bellare et al. [18]. Bernstein [26, 27] gives a more direct (though perhaps less intuitive) proof, and also discusses some generalized versions of CBC-MAC. As noted in this chapter, basic CBC-MAC is insecure when used to authenticate messages of different lengths. One way to fix this is to prepend the length to the message. This has the disadvantage of not being able to cope with *streaming* data, where the length of the message is not known in advance. Petrank and Rackoff [137] suggest an alternate, "on-line" approach addressing this issue. Further improvements were given by Black and Rogaway [32] and Iwata and Kurosawa [95]; these led to a new proposed standard called CMAC.

The importance of authenticated encryption was first explicitly highlighted in [100, 19], who propose definitions similar to what we have given here. Bellare and Namprempre [19] analyze the three generic approaches discussed here, although the idea of using encrypt-then-authenticate for achieving CCA-security goes back at least to the work of Dolev et al. [61]. Krawczyk [108] examines other methods for achieving secrecy and authentication, and also analyzes the authenticate-then-encrypt approach used by SSL. Degabriele and Paterson [54] show an attack on IPsec when configured to authenticate-then-encrypt (the default for authenticated encryption is actually encrypt-then-authenticate; however it is possible to achieve authenticate-then-encrypt in some configurations). Several nongeneric schemes for authenticated encryption have also been proposed; see [110] for a detailed comparison.

Information-theoretic MACs were first studied by Gilbert et al. [73]. Wegman and Carter [177] introduced the notion of strongly universal functions, and noted their application to one-time message authentication. They also show how to reduce the key length for this task by using an *almost* strongly universal function. Specifically, the construction we give here achieves 2^{-n}-security for messages of length n with keys of length $O(n)$; Wegman and Carter show how to construct a 2^{-n}-secure MAC for messages of length ℓ with keys of (essentially) length $\mathcal{O}(n \cdot \log \ell)$. The simple construction of a strongly universal function that we give here is (with minor differences) due to Carter and Wegman [42]. The reader interested in learning more about information-theoretic MACs is referred to the paper by Stinson [166], the survey by Simmons [162], or the first edition of Stinson's textbook [167, Chapter 10].

Exercises

4.1 Say $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ is a secure MAC, and for $k \in \{0,1\}^n$ the tag-generation algorithm Mac_k always outputs tags of length $t(n)$. Prove that t must be super-logarithmic or, equivalently, that if $t(n) = \mathcal{O}(\log n)$ then Π cannot be a secure MAC.

Hint: Consider the probability of randomly guessing a valid tag.

4.2 Consider an extension of the definition of secure message authentication where the adversary is provided with both a Mac and a Vrfy oracle.

(a) Provide a formal definition of security for this case.

(b) Assume Π is a deterministic MAC using canonical verification that satisfies Definition 4.2. Prove that Π also satisfies your definition from part (a).

4.3 Assume secure MACs exist. Give a construction of a MAC that is secure with respect to Definition 4.2 but that is not secure when the adversary is additionally given access to a Vrfy oracle (cf. the previous exercise).

4.4 Prove Proposition 4.4.

4.5 Assume secure MACs exist. Prove that there exists a MAC that is secure (by Definition 4.2) but is *not* strongly secure (by Definition 4.3).

4.6 Consider the following MAC for messages of length $\ell(n) = 2n - 2$ using a pseudorandom function F: On input a message $m_0 \| m_1$ (with $|m_0| = |m_1| = n - 1$) and key $k \in \{0,1\}^n$, algorithm Mac outputs $t = F_k(0\|m_0) \,\|\, F_k(1\|m_1)$. Algorithm Vrfy is defined in the natural way. Is $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ secure? Prove your answer.

4.7 Let F be a pseudorandom function. Show that each of the following MACs is insecure, even if used to authenticate fixed-length messages. (In each case Gen outputs a uniform $k \in \{0,1\}^n$. Let $\langle i \rangle$ denote an $n/2$-bit encoding of the integer i.)

(a) To authenticate a message $m = m_1, \ldots, m_\ell$, where $m_i \in \{0,1\}^n$, compute $t := F_k(m_1) \oplus \cdots \oplus F_k(m_\ell)$.

(b) To authenticate a message $m = m_1, \ldots, m_\ell$, where $m_i \in \{0,1\}^{n/2}$, compute $t := F_k(\langle 1 \rangle \| m_1) \oplus \cdots \oplus F_k(\langle \ell \rangle \| m_\ell)$.

(c) To authenticate a message $m = m_1, \ldots, m_\ell$, where $m_i \in \{0,1\}^{n/2}$, choose uniform $r \leftarrow \{0,1\}^n$, compute

$$t := F_k(r) \oplus F_k(\langle 1 \rangle \| m_1) \oplus \cdots \oplus F_k(\langle \ell \rangle \| m_\ell),$$

and let the tag be $\langle r, t \rangle$.

4.8 Let F be a pseudorandom function. Show that the following MAC for messages of length $2n$ is insecure: Gen outputs a uniform $k \in \{0,1\}^n$. To authenticate a message $m_1 \| m_2$ with $|m_1| = |m_2| = n$, compute the tag $F_k(m_1) \,\|\, F_k(F_k(m_2))$.

4.9 Given any *deterministic* MAC $(\mathsf{Mac}, \mathsf{Vrfy})$, we may view Mac as a keyed function. In both Constructions 4.5 and 4.11, Mac is a pseudorandom function. Give a construction of a secure, deterministic MAC in which Mac is *not* a pseudorandom function.

4.10 Is Construction 4.5 necessarily secure when instantiated using a weak pseudorandom function (cf. Exercise 3.26)? Explain.

4.11 Prove that Construction 4.7 is secure even when the adversary is additionally given access to a Vrfy oracle (cf. Exercise 4.2).

4.12 Prove that Construction 4.7 is secure if it is changed as follows: Set $t_i := F_k(r\|b\|i\|m_i)$ where b is a single bit such that $b = 0$ in all blocks but the last one, and $b = 1$ in the last block. (Assume for simplicity that the length of all messages being authenticated is always an integer multiple of $n/2 - 1$.) What is the advantage of this modification?

4.13 We explore what happens when the basic CBC-MAC construction is used with messages of different lengths.

(a) Say the sender and receiver do not agree on the message length in advance (and so $\mathsf{Vrfy}_k(m,t) = 1$ iff $t \overset{?}{=} \mathsf{Mac}_k(m)$, regardless of the length of m), but the sender is careful to only authenticate messages of length $2n$. Show that an adversary can forge a valid tag on a message of length $4n$.

(b) Say the receiver only accepts 3-block messages (so $\mathsf{Vrfy}_k(m,t) = 1$ only if m has length $3n$ and $t \overset{?}{=} \mathsf{Mac}_k(m)$), but the sender authenticates messages of any length a multiple of n. Show that an adversary can forge a valid tag on a new message.

4.14 Prove that the following modifications of basic CBC-MAC do not yield a secure MAC (even for fixed-length messages):

(a) Mac outputs all blocks $t_1, \ldots, t_\ell$, rather than just t_ℓ. (Verification only checks whether t_ℓ is correct.)

(b) A random initial block is used each time a message is authenticated. That is, choose uniform $t_0 \in \{0,1\}^n$, run basic CBC-MAC over the "message" $t_0, m_1, \ldots, m_\ell$, and output the tag $\langle t_0, t_\ell \rangle$. Verification is done in the natural way.

4.15 Show that appending the message length to the *end* of the message before applying basic CBC-MAC does not result in a secure MAC for arbitrary-length messages.

4.16 Show that the encoding for arbitrary-length messages described in Section 4.4.2 is prefix-free.

4.17 Consider the following encoding that handles messages whose length is less than $n \cdot 2^n$: We encode a string $m \in \{0,1\}^*$ by first appending as many 0s as needed to make the length of the resulting string $\hat{m}$ a nonzero multiple of n. Then we prepend the number of *blocks* in $\hat{m}$ (equivalently, prepend the integer $|\hat{m}|/n$), encoded as an n-bit string. Show that this encoding is *not* prefix-free.

4.18 Prove that the following modification of basic CBC-MAC gives a secure MAC for arbitrary-length messages (for simplicity, assume all messages have length a multiple of the block length). $\mathsf{Mac}_k(m)$ first computes $k_\ell = F_k(\ell)$, where ℓ is the length of m. The tag is then computed using basic CBC-MAC with key k_ℓ. Verification is done in the natural way.

4.19 Let F be a keyed function that is a secure (deterministic) MAC for messages of length n. (Note that F need not be a pseudorandom permutation.) Show that basic CBC-MAC is not necessarily a secure MAC (even for fixed-length messages) when instantiated with F.

4.20 Show that Construction 4.7 is strongly secure.

4.21 Show that Construction 4.18 might not be CCA-secure if it is instantiated with a secure MAC that is *not* strongly secure.

4.22 Prove that Construction 4.18 is unforgeable when instantiated with any encryption scheme (even if not CPA-secure) and any secure MAC (even if the MAC is not strongly secure).

4.23 Consider a strengthened version of unforgeability (Definition 4.16) 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 4.18 satisfies this stronger definition if Π_M is a strongly secure MAC.

(c) Show by counterexample that Construction 4.18 need not satisfy this stronger definition if Π_M is a secure MAC that is not strongly secure. (Compare to the previous exercise.)

4.24 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.

4.25 Let F be a strong pseudorandom permutation, and define the following fixed-length encryption scheme: On input a message $m \in \{0,1\}^{n/2}$ and key $k \in \{0,1\}^n$, algorithm Enc chooses a uniform $r \in \{0,1\}^{n/2}$ and computes $c := F_k(m\|r)$. (See Exercise 3.18.) Prove that this scheme is CCA-secure, but is not an authenticated encryption scheme.

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

4.27 Fix $\ell > 0$ and a prime p. Let $\mathcal{K} = \mathbb{Z}_p^{\ell+1}$, $\mathcal{M} = \mathbb{Z}_p^{\ell}$, and $\mathcal{T} = \mathbb{Z}_p$. Define $h : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ as

$$h_{k_0,k_1,\ldots,k_\ell}(m_1,\ldots,m_\ell) = \left[k_0 + \textstyle\sum_i k_i m_i \bmod p\,\right].$$

Prove that h is strongly universal.

4.28 Fix $\ell, n > 0$. Let $\mathcal{K} = \{0,1\}^{\ell \times n} \times \{0,1\}^{\ell}$ (interpreted as a boolean $\ell \times n$ matrix and an ℓ-dimensional vector), let $\mathcal{M} = \{0,1\}^n$, and let $\mathcal{T} = \{0,1\}^{\ell}$. Define $h : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ as $h_{K,v}(m) = K \cdot m \oplus v$, where all operations are performed modulo 2. Prove that h is strongly universal.

4.29 A *Toeplitz matrix* K is a matrix in which $K_{i,j} = K_{i-1,j-1}$ when $i, j > 1$; i.e., the values along any diagonal are equal. So an $\ell \times n$ Toeplitz matrix (for $\ell > n$) has the form

$$\begin{bmatrix} K_{1,1} & K_{1,2} & K_{1,3} & \cdots & K_{1,n} \\ K_{2,1} & K_{1,1} & K_{1,2} & \cdots & K_{1,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K_{\ell,1} & K_{\ell-1,1} & K_{\ell-2,1} & \cdots & K_{\ell-n+1,1} \end{bmatrix}.$$

Let $\mathcal{K} = T^{\ell \times n} \times \{0,1\}^{\ell}$ (where $T^{\ell \times n}$ denotes the set of $\ell \times n$ Toeplitz matrices), let $\mathcal{M} = \{0,1\}^n$, and let $\mathcal{T} = \{0,1\}^{\ell}$. Define $h : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ as $h_{K,v}(m) = K \cdot m \oplus v$, where all operations are performed modulo 2. Prove that h is strongly universal. What is the advantage here as compared to the construction in the previous exercise?

4.30 Define an appropriate notion of a *two-time* ε-secure MAC, and give a construction that meets your definition.

4.31 Let $\{h_n : \mathcal{K}_n \times \{0,1\}^{10 \cdot n} \to \{0,1\}^n\}_{n \in \mathbb{N}}$ be such that h_n is strongly universal for all n, and let F be a pseudorandom function. (When $K \in \mathcal{K}_n$ we write $h_K(\cdot)$ instead of $h_{n,K}(\cdot)$.) Consider the following MAC: $\mathsf{Gen}(1^n)$ chooses uniform $K \in \mathcal{K}_n$ and $k \in \{0,1\}^n$, and outputs (K, k). To authenticate a message $m \in \{0,1\}^{10 \cdot n}$, choose uniform $r \in \{0,1\}^n$ and output $\langle r,\ h_K(m) \oplus F_k(r) \rangle$. Verification is done in the natural way. Prove that this gives a (computationally) secure MAC for messages of length $10n$.