Chapter 7

Message Authentication

In most people's minds, privacy is the goal most strongly associated to cryptography. But message authentication is arguably even more important. Indeed you may or may not care if some particular message you send out stays private, but you almost certainly do want to be sure of the originator of each message that you act on. Message authentication is what buys you that guarantee.

Message authentication allows one party—the sender—to send a message to another party—the receiver—in such a way that if the message is modified en route, then the receiver will almost certainly detect this. Message authentication is also called *data-origin authentication.* Message authentication is said to protect the *integrity* of a message, ensuring that each message that it is received and deemed acceptable is arriving in the same condition that it was sent out—with no bits inserted, missing, or modified.

Here we'll be looking at the shared-key setting for message authentication (remember that message authentication in the public-key setting is the problem addressed by *digital signatures*). In this case the sender and the receiver share a secret key, K, which they'll use to authenticate their transmissions. We'll define the message authentication goal and we'll describe some different ways to achieve it. As usual, we'll be careful to pin down the problem we're working to solve.

7.1 The setting

It is often crucial for an agent who receives a message to be sure who sent it. If a hacker can call into his bank's central computer and produce deposit transactions that *appears* to be coming from a branch office, easy wealth is just around the corner. If an unprivileged user can interact over the network with his company's mainframe in such a way that the machine *thinks* that the packets it is receiving are coming from the system administrator, then all the machine's access-control mechanisms are for naught. In such cases the risk is that an adversary A, the *forger*, will create messages that look like they come from some other party, S, the (legitimate) *sender*. The attacker will send a message M to R, the *receiver* (or *verifier*), under S's identity. The receiver R will be tricked into believing that M originates with S. Because of this wrong belief, R may inappropriately act on M.

The rightful sender S could be one of many different kinds of entities, like a person, a corporation, a network address, or a particular process running on a particular machine. As the receiver R, you might know that it is S that supposedly sent you the message M for a variety of reasons. For

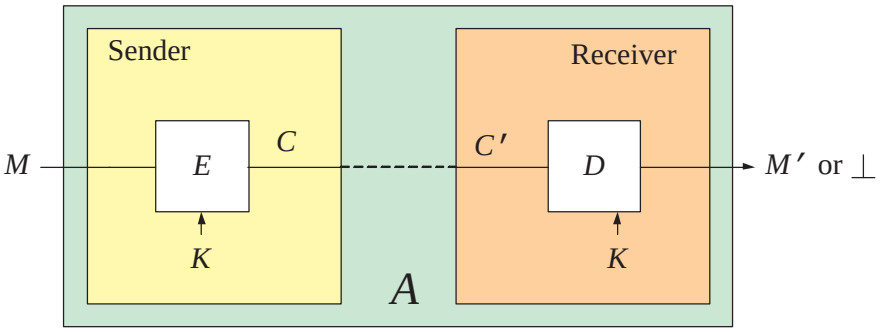

Figure 7.1: An authenticated-encryption scheme. Here we are authenticating messages with what is, syntactically, just an encryption scheme. The sender transmits a transformed version C of M and the receiver is able to recover $M' = M$ or else obtain indication of failure. Adversary A controls the communication channel and may even influence messages sent by the sender.

example, the message M might be tagged by an identifier which somehow names S. Or it might be that the manner in which M arrives is a route dedicated to servicing traffic from S.

Here we're going to be looking at the case when S and R already share some secret key, K. How S and R came to get this shared secret key is a separate question, one that we deal with later.

There are several high-level approaches for authenticating transmissions.

1. The most general approach works like this. To authenticate a message M using the key K, the sender will apply some encryption algorithm $\mathcal{E}$ to K, giving rise to a ciphertext C. When we speak of encrypting M in this context, we are using the word in the broadest possible sense, as any sort of keyed transformation on the message that obeys are earlier definition for the syntax of an encryption scheme; in particular, we are *not* suggesting that C conceals M. The sender S will transmit C to the receiver R. Maybe the receiver will receive C, or maybe it will not. The problem is that an adversary A may control the channel on which messages are being sent. Let C' be the message that the receiver actually gets. The receiver R, on receipt of C', will apply some decryption algorithm $\mathcal{D}$ to K and C'. We want that this should yield one of two things: (1) a message M' that is the original message M; or (2) an indication $\perp$ that C' be regarded as inauthentic. Viewed in this way, message authentication is accomplished by an encryption scheme. We are no longer interested in the privacy of the encryption scheme but, functionally, it is still an encryption scheme. See Fig. 7.1. We sometimes use the term *authenticated encryption* to indicate that we are using an encryption scheme to achieve authenticity.

2. Since our authenticity goal is not about privacy, most often the ciphertext C that the sender transmits is simply the original message M together with a tag T; that is, $C = \langle M, T \rangle$. When the ciphertext is of this form, we call the mechanism a *message-authentication scheme*. A message-authentication scheme will be specified by a tag-generation algorithm TG and a tag-verification algorithm VF. The former may be probabilistic or stateful; the latter is neither. The tag-generation algorithm TG produces a tag $T \xleftarrow{\$} \mathrm{TG}_K(M)$ from a key K and the message. The tag-verification algorithm VF $\leftarrow \mathrm{VF}_K(M', T')$ produces a bit from a key K, a message M', and a tag T'. The intent is that the bit 1 tells the receiver to accept M', while the bit 0 tells the receiver to reject M'. See Fig. 7.5

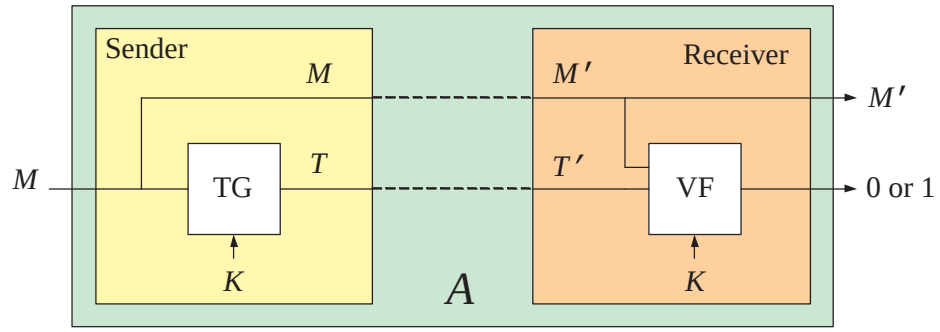

Figure 7.2: A message authentication scheme. This is a special case of the more general framework from the prior diagram. The authenticated message C is now understood to be the original message M together with a tag T. Separate algorithms generate the tag and verify the pair.

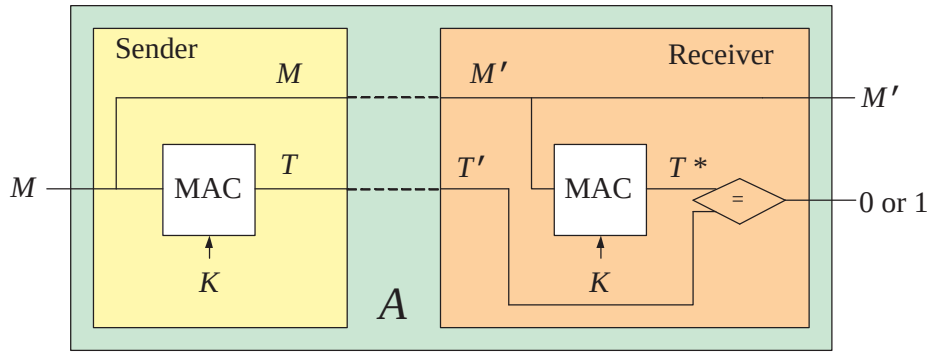

Figure 7.3: A message authentication code. This is a special case of a message authentication scheme. The authenticated message C is now understood to be the original message M together with a tag T that is computed as a deterministic and stateless function of M and K. The receiver verifies the authenticity of messages using the same MACing algorithm.

3. The most common possibility of all occurs when the tag-generation algorithm TG is deterministic and stateless. In this case we call the tag-generation algorithm, and the scheme itself, a *message authentication code*, or MAC. When authentication is accomplished using a MAC, we do not need to specify a separate tag-verification algorithm, for tag-verification always works he same way: the receiver, having received $\langle M', T' \rangle$, computes $T^* = \mathrm{MAC}_K(M')$. If this computed-tag T^* is identical to the received tag T' then the receiver regards the message M' as authentic; otherwise, the receiver regards M' as inauthentic. We write $T = \mathrm{MAC}_K(M)$ for the tag generated by the specified MAC. See Fig. 7.5

When the receiver decides that a message he has received is inauthentic what should he do? The receiver might want to just ignore the bogus message. Perhaps it was just noise on the channel; or perhaps taking action will do more harm than good, opening up new possibilities for denial-of-service attacks. Alternatively, the receiver may want to take more decisive actions, like tearing down the channel on which the message was received and informing some human being of apparent mischief. The proper course of action is dictated by the circumstances and the security policy of the receiver.

We point out that adversarial success in violating authenticity demands an active attack: to succeed, the adversary has to do more than listen—it has to get some bogus message to the receiver. In some communication scenarios it may be difficult for the adversary to get its messages to the receiver. For example, it may be tricky for an adversary to drop its own messages onto a physically secure phone line or fiber-optic channel. In other environments it may be trivial for the adversary to put messages onto the channel. Since we don't know what are the characteristics of the sender—receiver channel it is best to assume the worst and think that the adversary has plenty of power over this channel. We will actually assume even more than that, giving the adversary the power of creating legitimately authenticated messages.

We wish to emphasize that the message-authentication problem is very different from the privacy problem. We are not worried about secrecy of the message M; our concern is in whether the adversary can profit by injecting new messages into the communications stream. Not only is the problem conceptually different but, as we shall now see, privacy-providing encryption does nothing to ensure message authenticity.

7.2 Privacy does not imply authenticity

We know how to encrypt data so as to provide privacy, and something often suggested—and even done—is to encrypt as a way to provide authenticity. Fix a symmetric encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, and let parties S and R share a key K for this scheme. When S wants to send a message M to R, she encrypts it, transferring a ciphertext $M' = C$ generated via $C \xleftarrow{\$} \mathcal{E}_K(M)$. The receiver B decrypts it and, if it "makes sense", he regards the recovered message $M = \mathcal{D}_K(C)$ as authentic.

The argument that this works is as follows. Suppose, for example, that S transmits an ASCII message M_{100} which indicates that R should please transfer \$100 from the checking account of S to the checking account of some other party A. The adversary A wants to change the amount from the \$100 to \$900. Now if M_{100} had been sent in the clear, A can easily modify it. But if M_{100} is encrypted so that ciphertext C_{100} is sent, how is A to modify C_{100} so as to make S recover the different message M_{900}? The adversary A does not know the key K, so she cannot just encrypt M_{900} on her own. The privacy of C_{100} already rules out that C_{100} can be profitably tampered with.

The above argument is completely wrong. To see the flaws let's first look at a counter-example. If we encrypt M_{100} using a one time pad, then all the adversary has to do is to xor the byte of the ciphertext C_{100} that encodes the character "1" with the xor of the bytes for 1 and 9. That is, when we one-time pad encrypt, the privacy of the transmission does *not* make it difficult for the adversary to tamper with ciphertext so as to produce related ciphertexts.

How should one react to this counter-example? What you should *not* conclude is that one-time pad encryption is unsound. Our goal for the one-time pad was to provide privacy, and nothing we have said suggests that one-time pad encryption does not. Faulting the one-time pad encryption scheme for not providing authenticity is like faulting a car for not being able to fly; there is no reason to expect a tool designed to solve one problem to be effective at solving another.

You should *not* conclude that the example is contrived, and that you'd fare far better with some other encryption method. One-time-pad encryption is not at all contrived. And other methods of encryption, like CBC encryption, are only marginally better at protecting message integrity. This will be explored in the exercises.

You should *not* conclude that the failure stemmed from a failure to add "redundancy" before the message was encrypted. Adding redundancy is something like this: before the sender S encrypts his data he pads it with some known, fixed string, like 128 bits of zeros. When the receiver decrypts the

ciphertext he checks whether the decrypted string ends in 128 zeros. He rejects the transmission if it does not. Such an approach can, and almost always will, fail. For example, the added redundancy does absolutely nothing for our one-time-pad example.

What you *should* conclude is that privacy-providing encryption was never an appropriate approach for protecting its authenticity. With hindsight, this is pretty clear. The fact that data is encrypted need not prevent an adversary from being able to make the receiver recover data different from that which the sender had intended. Indeed with most encryption schemes *any* ciphertext will decrypt to *something*, so even a random transmission will cause the receiver to receive something different from what the sender intended, which was not to send any message at all. Now perhaps the random ciphertext will look like garbage to the receiver, or perhaps not. Since we do not know what the receiver intends to do with his data it is impossible to say.

Since the encryption schemes we have discussed were not designed for authenticating messages, they don't. We emphasize this because the belief that good encryption, perhaps after adding redundancy, already provides authenticity, is not only voiced, but even printed in books or embedded into security systems.

Good cryptographic design is goal-oriented. One must understand and formalize our goal. Only then do we have the basis on which to design and evaluate potential solutions. Accordingly, our next step is to come up with a definition for a message-authentication scheme and its security.

7.3 Syntax for message authentication

In Section 7.1 we sketched three approaches, each more narrow than then the next, for providing authenticity. The first, which we called authenticated encryption, one provides authenticity by using what is a symmetric encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The imagined purpose shifts from providing privacy to providing authenticity, but the syntax of does not change. Recall that we already built into our definition of a symmetric encryption scheme the possibility that decryption would output a distinguished value $\perp$. We didn't use that capability in defining privacy—but we will need it for authenticity. Intuitively, the decryption mechanism outputting $\perp$ is interpreted as meaning that the ciphertext received (that is, the authenticated message) should be regarded as invalid.

We also singled out two more specific ways to provide authenticity. special cases of the above encryption schemes designed The first was a *message-authentication scheme*. Formally, this is a pair of algorithms $(\mathrm{TG}, \mathrm{VF})$. The first of these may be probabilistic or stateful, while the second is deterministic. Algorithm TG (for "tag generation") takes as input a string $K \in \mathcal{K}$, for some associated set $\mathcal{K}$, and a string $M \in \{0,1\}^*$. The set $\mathcal{K}$ is either finite or otherwise has an associated probability distribution (we must be able to choose a random point K from $\mathcal{K}$). The tag-generation algorithm TG produces a tag $T \xleftarrow{\$} \mathrm{TG}_K(M) \in \{0,1\}^* \cup \{\perp\}$. Algorithm VF (for "verification") takes as input strings $K \in$, $M \in \{0,1\}^*$, and $T \in \{0,1\}^*$. It outputs a bit $\mathrm{VF}_K(M,T) \in \{0,1\}$. The intended semantics is 1 for *accept* and 0 for *reject*. We insist that if $T \xleftarrow{\$} \mathrm{TG}_K(M)$ and $T \neq \perp$ then $\mathrm{VF}_K(M,T) = 1$. Every message-authentication scheme gives rise to an encryption scheme where $\mathcal{E}_K(M)$ computes $T \xleftarrow{\$} \mathrm{TG}_K(M)$ and returns $\langle M, T\rangle$, and $\mathcal{D}_K(\langle M,T\rangle) = M$ if $\mathrm{VF}_K(M,T) = 1$ while $\mathcal{D}_K(\langle M,T\rangle) = \perp$ otherwise. Of course this encryption scheme does nothing to provide privacy.

A *message authentication code* (MAC) corresponds to the special case of a message-authentication scheme in which tag-generation is deterministic and stateful. Formally, a message authentication code is a deterministic algorithm MAC: $\mathcal{K} \times \{0,1\}^* \to \{0,1\}^* \cup \{\perp\}$ where $\mathcal{K}$ is a finite set, or is otherwise endowed with a probability distribution. The tag for a message M is $T = \mathrm{MAC}_K(M)$.

To verify $\langle M, T\rangle$ the receiver checks if $T = \mathrm{MAC}_K(M)$. If so, message M is viewed as authentic; otherwise, the message is viewed as being a forgery.

Note that our definitions don't permit stateful message-recovery / verification. Stateful functions for the receiver can be problematic because of the possibility of messages not reaching their destination—it is too easy for the receiver to be in a state different from the one that we'd like. All the same, stateful MAC verification functions are essential for detecting "replay attacks."

Recall that it was essential for the IND-CPA security of an encryption scheme that the encryption algorithm be probabilistic or stateful—you couldn't achieve IND-CPA security with a deterministic encryption algorithm. But we will see that probabilism and state are not necessary for achieving secure message authentication. This realization is built into the fact that we deal with MACs.

7.4 Definitions of security

Let us concentrate first on message authentication codes. We begin with a discussion of the issues and then state a formal definition.

The goal that we seek to achieve with a MAC is to be able to detect any attempt by the adversary to modify the transmitted data. We don't want the adversary to be able to produce messages that the receiver will deem authentic—only the sender should be able to do this. That is, we don't want that the adversary A to be able to create a pair (M, Tag) such that $\mathrm{VF}_K(M, \mathit{Tag}) = 1$, but M did not originate with the sender S. Such a pair (M, Tag) is called a *forgery*. If the adversary can make such a pair, she is said to have forged.

In some discussions of security people assume that the adversary's goal is to recover the secret key K. Certainly if it could do this, it would be a disaster, since it could then forge anything. It is important to understand, however, that an adversary might be able to forge without being able to recover the key, and if all we asked was for the adversary to be unable to recover the key, we'd be asking too little. Forgery is what counts, not key recovery.

Now it should be admitted right away that some forgeries might be useless to the adversary. For example, maybe the adversary can forge, but it can only forge strings that look random; meanwhile, suppose that all "good" messages are supposed to have a certain format. Should this really be viewed as a forgery? The answer is *yes.* If checking that the message is of a certain format was really a part of validating the message, then that should have been considered as part of the message-authentication code. In the absence of this, it is not for us to make assumptions about how the messages are formatted or interpreted; we really have no idea. Good protocol design means the security is guaranteed no matter what is the application.

In our adversary's attempt to forge a message we could consider various attacks. The simplest setting is that the adversary wants to forge a message even though it has never seen any transmission sent by the sender. In this case the adversary must concoct a pair (M, T) that is valid, even though it hasn't obtained any information to help. This is called a *no-message attack.* It often falls short of capturing the capabilities of realistic adversaries, since an adversary who can inject bogus messages onto the communications media can probably see valid messages as well. We should let the adversary use this information.

Suppose the sender sends the transmission (M, T) consisting of some message M and its legitimate tag T. The receiver will certainly accept this—that is built into our definition. Now at once a simple attack comes to mind: the adversary can just repeat this transmission, (M, T), and get the receiver to accept it once again. This attack is unavoidable, for our MAC is a deterministic function that the receiver recomputes. If the receiver accepted (M, T) once, he's bound to do it

again.

What we have just described is called a *replay attack*. The adversary sees a valid (M, T) from the sender, and at some later point in time it re-transmits it. Since the receiver accepted it the first time, he'll do so again.

Should a replay attack count as a valid forgery? In real life it usually should. Say the first message was "Transfer $1000 from my account to the account of party A." Then party A may have a simple way to enriching herself: it just keeps replaying this same authenticated message, happily watching her bank balance grow.

It is important to protect against replay attacks. But for the moment we will not try to do this. We will say that a replay is *not* a valid forgery; to be valid a forgery must be of a message M which was *not* already produced by the sender. We will see later that we can always achieve security against replay attacks by simple means; that is, we can take any message authentication mechanism which is not secure against replay attacks and modify it—after making the receiver stateful—so that it will be secure against replay attacks. At this point, not worrying about replay attacks results in a cleaner problem definition. And it leads us to a more modular protocol-design approach—that is, we cut up the problem into sensible parts ("basic security" and then "replay security") solving them one by one.

Of course there is no reason to think that the adversary will be limited to seeing only one example message. Realistic adversaries may see millions of authenticated messages, and still it should be hard for them to forge.

For some message authentication schemes the adversary's ability to forge will grow with the number q_s of legitimate message-tag pairs it sees. Likewise, in some security systems the number of valid (M, T) pairs that the adversary can obtain may be architecturally limited. (For example, a stateful Signer may be unwilling to MAC more than a certain number of messages.) So when we give our quantitative treatment of security we will treat q_s as an important adversarial resource.

How exactly do all these tagged messages arise? We could think of there being some distribution on messages that the sender will authenticate, but in some settings it is even possible for the adversary to influence which messages are tagged. In the worst case, imagine that the adversary *itself* chooses which messages get authenticated. That is, the adversary chooses a message, gets its tag, chooses another message, gets its tag, and so forth. Then it tries to forge. This is called an *adaptive chosen-message attack*. It wins if it succeeds in forging the MAC of a message which it has not queried to the sender.

At first glance it may seem like an adaptive chosen-message attack is unrealistically generous to our adversary; after all, if an adversary could really obtain a valid tag for *any* message it wanted, wouldn't that make moot the whole point of authenticating messages? In fact, there are several good arguments for allowing the adversary such a strong capability. First, we will see examples—higher-level protocols that use MACs—where adaptive chosen-message attacks are quite realistic. Second, recall our general principles. We want to design schemes which are secure in *any* usage. This requires that we make worst-case notions of security, so that when we err in realistically modeling adversarial capabilities, we err on the side of caution, allowing the adversary more power than it might really have. Since eventually we will design schemes that meet our stringent notions of security, we only gain when we assume our adversary to be strong.

As an example of a simple scenario in which an adaptive chosen-message attack is realistic, imagine that the sender S is forwarding messages to a receiver R. The sender receives messages from any number of third parties, $A_1, \ldots, A_n$. The sender gets a piece of data M from party A_i along a secure channel, and then the sender transmits to the receiver $\langle i \rangle \,\|\, M \,\|\, MAC_K(\langle i \rangle \,\|\, M)$. This is the sender's way of attesting to the fact that he has received message M from party A_i. Now if one of these third parties, say A_1, wants to play an adversarial role, it will ask the sender

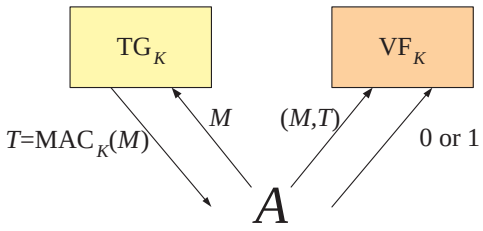

Figure 7.4: *The model for a message authentication code. Adversary A has access to a tag-generation oracle and a tag-verification oracle. The adversary wants to get the verification oracle to answer 1 to some (M, T) for which it didn't earlier ask the signing oracle M. The verification oracle returns 1 if $T = \mathrm{MAC}_K(M)$ and 0 if $T \neq \mathrm{MAC}_K(M)$.*

to forward its adaptively-chosen messages $M_1, M_2, \ldots$ to the receiver. If, based on what it sees, it can learn the key K, or even if it can learn to forge message of the form $\langle 2 \rangle \| M$, so as to produce a valid $\langle 2 \rangle \| M \| MAC_K(\langle 2 \rangle \| M)$, then the intent of the protocol will have been defeated.

So far we have said that we want to give our adversary the ability to obtain MACs for messages of its choosing, and then we want to look at whether or not it can forge: produce a valid (M, T) pair where it never asked the sender to MAC M. But we should recognize that a realistic adversary might be able to produce lots of candidate forgeries, and it may be content if any of these turn out to be valid. We can model this possibility by giving the adversary the capability to tell if a prospective (M, T) pair is valid, and saying that the adversary forges if it ever finds an (M, T) pair that is but M was not MACed by the sender.

Whether or not a real adversary can try lots of possible forgeries depends on the context. Suppose the receiver is going to tear down a connection the moment he detects an invalid tag. Then it is unrealistic to try to use this receiver to help you determine if a candidate pair (M, T) is valid—one mistake, and you're done for. In this case, thinking of there being a single attempt to forge a message is quite adequate.

On the other hand, suppose that a receiver just ignores any improperly tagged message, while it responds in some noticeably different way if it receives a properly authenticated message. In this case a quite reasonable adversarial strategy may be ask the verifier about the validity of a large number of candidate (M, T) pairs. The adversary hopes to find at least one that is valid. When the adversary finds such an (M, T) pair, we'll say that it has won.

Let us summarize. To be fully general, we will give our adversary two different capabilities. The first adversarial capability is to obtain a MAC M for any message that it chooses. We will call this a signing query. The adversary will make some number of them, q_{s}. The second adversarial capability is to find out if a particular pair (M, T) is valid. We will call this a verification query. The adversary will make some number of them, q_{v}. Our adversary is said to succeed—to forge—if it ever makes a verification query (M, T) and gets a return value of 1 (ACCEPT) even though the message M is not a message that the adversary already knew a tag for by virtue of an earlier signing query. Let us now proceed more formally.

Let MAC: $\mathcal{K} \times \{0,1\}^* \to \{0,1\}^*$ be an arbitrary message authentication code. We will formalize a quantitative notion of security against adaptive chosen-message attack. We begin by describing the model.

We distill the model from the intuition we have described above. There is no need, in the model, to think of the sender and the verifier as animate entities. The purpose of the sender, from the adversary's point of view, is to authenticate messages. So we will embody the sender as an oracle

that the adversary can use to authenticate any message M. This *tag-generation oracle*, as we will call it, is our way to provide the adversary black-box access to the function $\mathrm{MAC}_K(\cdot)$. Likewise, the purpose of the verifier, from the adversary's point of view, is to have that will test attempted forgeries. So we will embody the verifier as an oracle that the adversary can use to see if a candidate pair (M, T) is valid. This *verification oracle*, as we will call it, is our way to provide the adversary black-box access to the function $\mathrm{VF}_K(\cdot)$ which is 1 if $T = \mathrm{MAC}_K(M)$ and 0 otherwise. Thus, when we become formal, the cast of characters—the sender, receiver, and the adversary—gets reduced to just the adversary, running with its oracles.

Definition 7.4.1 [MAC security] Let $\mathrm{MAC}\colon \mathcal{K} \times \{0,1\}^* \to \{0,1\}^*$ be a message authentication code and let A be an adversary. We consider the following experiment:

Experiment $\mathbf{Exp}^{\text{uf-cma}}_{\mathrm{MAC}}(A)$
$K \xleftarrow{\$} \mathcal{K}$
Run $A^{\mathrm{MAC}_K(\cdot),\mathrm{VF}_K(\cdot,\cdot)}$ where $\mathrm{VF}_K(M,T)$ is 1 if $\mathrm{MAC}_K(M) = T$ and 0 otherwise
if A made a VF_K query (M,T) such that
– The oracle returned 1, and
– A did not, prior to making verification query (M,T), make tag-generation query M
then return 1 else return 0

The *uf-cma advantage* of A is defined as

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathrm{MAC}}(A) \quad = \quad \Pr\left[\mathbf{Exp}^{\text{uf-cma}}_{\mathrm{MAC}}(A) \Rightarrow 1\right] \ . \blacksquare$$

Let us discuss the above definition. Fix a message authentication code MAC. Then we associate to any adversary A its "advantage," or "success probability." We denote this value as $\mathbf{Adv}^{\text{uf-cma}}_{\mathrm{MAC}}(A)$. It's just the chance that A manages to forge. The probability is over the choice of key K and the probabilistic choices, if any, that the adversary A makes.

As usual, the advantage that can be achieved depends both on the adversary strategy and the resources it uses. Informally, Π is secure if the advantage of a practical adversary is low.

As usual, there is a certain amount of arbitrariness as to which resources we measure. Certainly it is important to separate the oracle queries (q_{s} and q_{v}) from the time. In practice, signing queries correspond to messages sent by the legitimate sender, and obtaining these is probably more difficult than just computing on one's own. Verification queries correspond to messages the adversary hopes the verifier will accept, so finding out if it does accept these queries again requires interaction. Some system architectures may effectively limit q_{s} and q_{v}. No system architecture can limit t; that is limited primarily by the adversary's budget.

We emphasize that there are contexts in which you are happy with a MAC that makes forgery impractical when $q_{\mathrm{v}} = 1$ and $q_{\mathrm{s}} = 0$ (an "impersonation attack") and there are contexts in which you are happy when forgery is impractical when $q_{\mathrm{v}} = 1$ and $q_{\mathrm{s}} = 1$ (a "substitution attack"). But it is perhaps more common that you'd like for forgery to be impractical even when q_{s} is large, like 2^{50}, and when q_{v} is large, too.

Naturally the key K is not directly given to the adversary, and neither are any random choices or counter used by the MAC-generation algorithm. The adversary sees these things only to the extent that they are reflected in the answers to her oracle queries.

With a definition for MAC security in hand, it is not hard for us to similarly define authenticity for encryption schemes and message-authentication schemes. Let us do the former; we will explore the latter in exercises. We have an encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ and we want to measure how effective an adversary is at attacking its authenticity.

Definition 7.4.2 [Authenticity of an encryption scheme] Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme and let A be an adversary. We consider the following experiment:

Experiment $\mathbf{Exp}^{\text{auth}}_{\Pi}(A)$
  $K \xleftarrow{\$} \mathcal{K}$
  Run $A^{\mathcal{E}_K(\cdot),\text{VF}_K(\cdot)}$ where $\text{VF}_K(C)$ is 1 if $\mathcal{D}_K(C) \in \{0,1\}^*$ and 0 if $\mathcal{D}_K(C) = \perp$
  if A made a VF_K query C such that
  – The oracle returned 1, and
  – A did not, prior to making verification query C,
    make an encryption query that returned C
  then return 1 else return 0

The *authenticity advantage* of A is defined as

$$\mathbf{Adv}^{\text{auth}}_{\Pi}(A) \quad = \quad \Pr[\mathbf{Exp}^{\text{auth}}_{\Pi}(A) \Rightarrow 1] \; . \blacksquare$$

We note that we could just as well have provided A with a decryption oracle $\mathcal{D}_K(\cdot)$ instead of a verification oracle $\text{VF}_K(\cdot)$, giving the adversary credit if it ever manages to ask a this oracle a query C that decrypts to something other than $\perp$ and where C was not already returned by the encryption oracle.

7.5 Examples

Let us examine some example message authentication codes and use the definition to assess their strengths and weaknesses. We fix a PRF F: $\mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. Our first scheme MAC1: $\mathcal{K} \times \{0,1\}^* \to \{0,1\}^*$ works as follows:

algorithm $\text{MAC1}_K(M)$
  if ($|M| \bmod n \neq 0$ or $|M| = 0$) then return $\perp$
  Break M into n-bit blocks $M = M_1 \ldots M_m$
  for $i \leftarrow 1$ to m do $Y_i \leftarrow F_K(M_i)$
  $T \leftarrow Y_1 \oplus \cdots \oplus Y_n$
  return T

Now let us try to assess the security of this message authentication code.

Suppose the adversary wants to forge the tag of a certain given message M. A priori it is unclear this can be done. The adversary is not in possession of the secret key K, so cannot compute F_K and use it to compute T. But remember that the notion of security we have defined says that the adversary is successful as long as it can produce a correct tag for *some* message, not necessarily a given one. We now note that even without a chosen-message attack (in fact without seeing any examples of correctly tagged data) the adversary can do this. It can choose a message M consisting of two equal blocks, say $M = X \| X$ where X is some n-bit string, set $T \leftarrow 0^n$, and make verification query (M, T). Notice that $\text{VF}_K(M, Tag) = 1$ because $F_K(x) \oplus F_K(x) = 0^n = T$. In more detail, the adversary is as follows.

algorithm $A_1^{\text{MAC}_K(\cdot),\text{VF}_K(\cdot,\cdot)}$
  Let X be any n-bit string
  $M \leftarrow X \| X$
  $T \leftarrow 0^n$
  $d \leftarrow \text{VF}_K(M, T)$

Then $\mathbf{Adv}^{\text{uf-cma}}_{\text{MAC}}(A_1) = 1$. Furthermore A_1 makes no signing oracle queries, uses $t = O(n)$ time, and its verification query has length $2n$-bits, so it is very practical.

There are many other attacks. For example we note that

$$T \;=\; F_K(M_1) \oplus F_K(M_2)$$

is not only the tag of M_1M_2 but also the tag of M_2M_1. So it is possible, given the tag of a message, to forge the tag of a new message formed by permuting the blocks of the old message. We leave it to the reader to specify the corresponding adversary and compute its advantage.

Let us now try to strengthen the scheme to avoid these attacks. Instead of applying F_K to a data block, we will first prefix the data block with its index. To do this, first pick some parameter ι with $1 \le \iota \le n-1$. We will write each block's index as an ι-bit string. The MAC-generation algorithm is the following:

algorithm $\text{MAC2}_K(M)$
  $\eta \leftarrow n - \iota$
  if ($|M| \bmod \eta \neq 0$ or $|M| = 0$ or $|M|/\eta \geq 2^\iota$) then return $\perp$
  Break M into η-bit blocks $M = M_1 \ldots M_m$
  for $i \leftarrow 1$ to m do $Y_i \leftarrow F_K([i]_\iota \,\|\, M_i)$
  $T \leftarrow Y_1 \oplus \cdots \oplus Y_m$
  return Tag

As the code indicates, we divide M into blocks, but the size of each block is smaller than in our previous scheme: it is now only $\eta = n - \iota$ bits. Then we prefix the i-th message block with the value i itself, the block index, written in binary as a string of length exactly m bits. It is to this padded block that we apply F_K before taking the xor.

Note that encoding of the block index i as an $iota$-bit string is only possible if $i < 2^\iota$. This means that we cannot authenticate a message M having more 2^ι blocks. This explains the conditions under which the MAC returns $\perp$. However this is a feasible restriction in practice, since a reasonable value of ι, like $\iota = 32$, is large enough that very long messages will be in the message space.

Anyway, the question we are really concerned with is the security. Has this improved from scheme MAC1? Begin by noticing that the attacks we found on MAC1 no longer work. For example if X is an η-bit string and we let $M = X \,\|\, X$ then its tag is *not* likely to be 0^n. Similarly, the second attack discussed above, namely that based on permuting of message blocks, also has low chance of success against the new scheme. Why? In the new scheme, if M_1, M_2 are strings of length η, then

$$\begin{aligned} \text{MAC2}_K(M_1M_2) &= F_K([1]_\iota \,\|\, M_1) \oplus F_K([2]_\iota \,\|\, M_2) \\ \text{MAC2}_K(M_2M_1) &= F_K([1]_m \,\|\, M_2) \oplus F_K([2]_\iota \,\|\, M_1)\,. \end{aligned}$$

These are unlikely to be equal. As an exercise, a reader might upper bound the probability that these values are equal in terms of the value of the advantage of F at appropriate parameter values.

All the same, MAC2 is still insecure. The attack however require a more non-trivial usage of the chosen-message attacking ability. The adversary will query the tagging oracle at several related points and combine the responses into the tag of a new message. We call it A_2–

algorithm $A_2^{\text{MAC}_K(\cdot),\text{VF}_K(\cdot)}$
  Let A_1, B_1 be distinct, η-bit strings
  Let A_2, B_2 be distinct η-bit strings
  $T_1 \leftarrow \text{MAC}_K(A_1A_2)$; $T_2 \leftarrow \text{MAC}_K(A_1B_2)$; $T_3 \leftarrow \text{MAC}_K(B_1A_2)$

$T \leftarrow T_1 \oplus T_2 \oplus T_3$
$d \leftarrow \mathrm{VF}_K(B_1B_2, T)$

We claim that $\mathbf{Adv}^{\text{uf-cma}}_{\text{MAC2}}(A_2) = 1$. Why? This requires two things. First that $\mathrm{VF}_K(B_1B_2, T) = 1$, and second that B_1B_2 was never a query to $\mathrm{MAC}_K(\cdot)$ in the above code. The latter is true because we insisted above that $a_1 \neq b_1$ and $a_2 \neq b_2$, which together mean that $B_1B_2 \notin \{A_1A_2, A_1B_2, B_1A_2\}$. So now let us check the first claim. We use the definition of the tagging algorithm to see that

$$\begin{aligned} T_1 &= F_K([1]_\iota \,\|\, A_1) \oplus F_K([2]_\iota \,\|\, A_2) \\ T_2 &= F_K([1]_\iota \,\|\, A_1) \oplus F_K([2]_\iota \,\|\, B_2) \\ T_3 &= F_K([1]_\iota \,\|\, B_1) \oplus F_K([2]_\iota \,\|\, A_2)\ . \end{aligned}$$

Now look how A_2 defined T and do the computation; due to cancellations we get

$$\begin{aligned} T &= T_1 \oplus T_2 \oplus T_3 \\ &= F_K([1]_\iota \,\|\, B_1) \oplus F_K([2]_\iota \,\|\, B_2)\ . \end{aligned}$$

This is indeed the correct tag of B_1B_2, meaning the value T' that $\mathrm{VF}_K(B_1B_2, T)$ would compute, so the latter algorithm returns 1, as claimed. In summary we have shown that this scheme is insecure.

It turns out that a slight modification of the above, based on use of a counter or random number chosen by the MAC algorithm, actually yields a secure scheme. For the moment however we want to stress a feature of the above attacks. Namely that these attacks did not cryptanalyze the PRF F. The attacks did not care anything about the structure of F; whether it was DES, AES, or anything else. They found weaknesses in the message authentication schemes themselves. In particular, the attacks work just as well when F_K is a random function, or a "perfect" cipher. This illustrates again the point we have been making, about the distinction between a tool (here the PRF) and its usage. We need to make better usage of the tool, and in fact to tie the security of the scheme to that of the underlying tool in such a way that attacks like those illustrated here are provably impossible under the assumption that the tool is secure.

7.6 The PRF-as-a-MAC paradigm

Pseudorandom functions make good MACs, and constructing a MAC in this way is an excellent approach. Here we show why PRFs are good MACs, and determine the concrete security of the underlying reduction. The following shows that the reduction is almost tight—security hardly degrades at all.

Let $F\colon \mathcal{K} \times D \to \{0,1\}^\tau$ be a family of functions. We associate to F a message authentication code $\mathrm{MAC}\colon \mathcal{K} \times D \to \{0,1\}^\tau$ via

algorithm $\mathrm{MAC}_K(M)$
  if $(M \notin D)$ then return $\perp$
  $T \leftarrow F_K(M)$
  return T

Note that when we think of a PRF as a MAC it is important that the domain of the PRF be whatever one wants as the domain of the MAC. So such a PRF probably won't be realized as a blockcipher. It may have to be realized by a PRF that allows for inputs of many different lengths,

since you might want to MAC messages of many different lengths. As yet we haven't demonstrated that we can make such PRFs. But we will. Let us first relate the security of the above MAC to that of the PRF.

Proposition 7.6.1 Let $F\colon\ \mathcal{K}\times D \to \{0,1\}^\tau$ be a family of functions and let MAC be the associated message authentication code as defined above. Let A by any adversary attacking Π, making $q_{\rm s}$ MAC-generation queries of total length μ_s, $q_{\rm v}$ MAC-verification queries of total length μ_v, and having running time t. Then there exists an adversary B attacking F such that

$$\mathbf{Adv}^{\text{uf-cma}}_{\Pi}(A) \;\;\leq\;\; \mathbf{Adv}^{\rm prf}_{F}(B) + \frac{q_{\rm v}}{2^\tau}\ . \tag{7.1}$$

Furthermore B makes $q_{\rm s}+q_{\rm v}$ oracle queries of total length $\mu_s+\mu_v$ and has running time t.

Proof: Remember that B is given an oracle for a function $f\colon\ D\to\{0,1\}^\tau$. It will run A, providing it an environment in which A's oracle queries are answered by B.

algorithm B^f
  $d \leftarrow 0$; $S \leftarrow \emptyset$
  Run A
    When A asks its signing oracle some query M:
      Answer $f(M)$ to A ; $S \leftarrow S \cup \{M\}$
    When A asks its verification oracle some query (M, Tag):
      if $f(M) = Tag$ then
        answer 1 to A ; if $M \notin S$ then $d \leftarrow 1$
        else answer 0 to A
  Until A halts
  return d

We now proceed to the analysis. We claim that

$$\Pr\left[\mathbf{Exp}^{\text{prf-1}}_{F}(B)\Rightarrow 1\right] \;\;=\;\; \mathbf{Adv}^{\text{uf-cma}}_{\Pi}(A) \tag{7.2}$$

$$\Pr\left[\mathbf{Exp}^{\text{prf-0}}_{F}(B)\Rightarrow 1\right] \;\;\leq\;\; \frac{q_{\rm v}}{2^\tau}\ . \tag{7.3}$$

Subtracting, we get Equation (7.1). Let us now justify the two equations above.

In the first case f is an instance of F, so that the simulated environment that B is providing for A is exactly that of experiment $\mathbf{Exp}^{\text{uf-cma}}_{\Pi}(A)$. Since B returns 1 exactly when A makes a successful verification query, we have Equation (7.2).

In the second case, A is running in an environment that is alien to it, namely one where a random function is being used to compute MACs. We have no idea what A will do in this environment, but no matter what, we know that the probability that any particular verification query (M, Tag) with $M \notin S$ will be answered by 1 is at most $2^{-\tau}$, because that is the probability that $Tag = f(M)$. Since there are at most $q_{\rm v}$ verification queries, Equation (7.3) follows. ∎

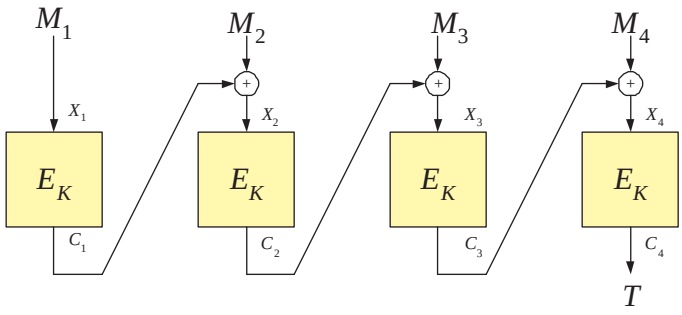

Figure 7.5: *The CBC MAC, here illustrated with a message M of four blocks, $M = M_1M_2M_3M_4$.*

7.7 The CBC MAC

A very popular class of MACs is obtained via cipher-block chaining of a given blockcipher. The method is as follows:

Scheme 7.7.1 CBC MAC] Let $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher. The CBC MAC over blockcipher E has key space $\mathcal{K}$ and is given by the following algorithm:

algorithm $\mathrm{MAC}_K(M)$
  if $M \notin (\{0,1\}^n)^+$ then return $\perp$
  Break M into n-bit blocks $M_1 \cdots M_m$
  $C_0 \leftarrow 0^n$
  for $i = 1$ to m do $C_i \leftarrow E_K(C_{i-1} \oplus M_i)$
  return C_m

See Fig. 7.5 for an illustration with $m = 4$. ▮

As we will see below, the CBC MAC is secure only if you restrict attention to strings of some one particular length: the domain is restricted to $\{0,1\}^{mn}$ for some constant m. If we apply the CBC MAC across messages of varying lengths, it will be easy to distinguish this object from a random function.

Theorem 7.7.2 [1] Fix $n \geq 1$, $m \geq 1$, and $q \geq 2$. Let A be an adversary that asks at most q queries, each of mn bits. Then that

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{CBC}[\mathsf{Func}(mn,n)]}(A) \;\leq\; \frac{m^2q^2}{2^n}\,. \blacksquare$$

Proof: Let A be an adversary that asks exactly q queries and assume without loss of generality that it never repeats a query. Refer to games C0–C9 in Fig. 7.6. Let us begin by explaining the notation used there. Each query M^s in the games is required to be a string of blocks, and we silently parse M^s to $M^s = M^s_1 M^s_2 \cdots M^s_m$ where each M_i is a block. Recall that $M^s_{1\to i} = M^s_1 \cdots M^s_i$. The function $\pi\colon \{0,1\}^n \to \{0,1\}^n$ is initially undefined at each point. The set $\mathrm{Domain}(\pi)$ grows as we define points $\pi(X)$, while $\overline{\mathrm{Range}}(\pi)$, initially $\{0,1\}^n$, correspondingly shrinks. The table $\mathbb{Y}$ stores blocks and is indexed by strings of blocks P having at most m blocks. A random block will come to occupy selected entries $\mathbb{Y}[X]$ except for $\mathbb{Y}[\varepsilon]$, which is initialized to the constant block 0^n and

On the s^{th} query $F(M^s)$ Game C1
100 $P \leftarrow \mathsf{Prefix}(M^1, \ldots, M^s)$
101 $C \leftarrow \mathbb{Y}[P]$
102 for $j \leftarrow \|P\|_n + 1$ to m do
103 $X \leftarrow C \oplus M^s_j$
104 $C \xleftarrow{\$} \{0,1\}^n$
105 if $C \in \mathrm{Range}(\pi)$ then $bad \leftarrow \mathsf{true}$, $C \xleftarrow{\$} \overline{\mathrm{Range}}(\pi)$
106 if $X \in \mathrm{Domain}(\pi)$ then $bad \leftarrow \mathsf{true}$, $C \leftarrow \pi(X)$
107 $\pi(X) \leftarrow C$
108 $\mathbb{Y}[M^s_{1\to j}] \leftarrow C$ omit for Game C0
109 return C

On the s^{th} query $F(M^s)$ Game C2
200 $P \leftarrow \mathsf{Prefix}(M^1, \ldots, M^s)$
201 $C \leftarrow \mathbb{Y}[P]$
202 for $j \leftarrow \|P\|_n + 1$ to m do
203 $X \leftarrow C \oplus M^s_j$
204 $C \xleftarrow{\$} \{0,1\}^n$
205 if $X \in \mathrm{Domain}(\pi)$ then $bad \leftarrow \mathsf{true}$
206 $\pi(X) \leftarrow C$
207 $\mathbb{Y}[M^s_{1\to j}] \leftarrow C$
208 return C

On the s^{th} query $F(M^s)$ Game C3
300 $P \leftarrow \mathsf{Prefix}(M^1, \ldots, M^s)$
301 $C \leftarrow \mathbb{Y}[P]$
302 for $j \leftarrow \|P\|_n + 1$ to m do
303 $X \leftarrow C \oplus M^s_j$
304 $C \xleftarrow{\$} \{0,1\}^n$
305 if $X \in \mathrm{Domain}(\pi)$ then $bad \leftarrow \mathsf{true}$
306 $\pi(X) \leftarrow \mathsf{defined}$
307 $\mathbb{Y}[M^s_{1\to j}] \leftarrow C$
308 return C

On the s^{th} query $F(M^s)$ Game C4
400 $P \leftarrow \mathsf{Prefix}(M^1, \ldots, M^s)$
401 $C \leftarrow \mathbb{Y}[P]$
402 for $j \leftarrow \|P\|_n + 1$ to m do
403 $X \leftarrow C \oplus M^s_j$
404 if $X \in \mathrm{Domain}(\pi)$ then $bad \leftarrow \mathsf{true}$
405 $\pi(X) \leftarrow \mathsf{defined}$
406 $C \leftarrow \mathbb{Y}[M^s_{1\to j}] \xleftarrow{\$} \{0,1\}^n$
407 $Z^s \xleftarrow{\$} \{0,1\}^n$
408 return Z^s

500 for $s \leftarrow 1$ to q do Game C5
501 $\mathtt{P}^s \leftarrow \mathsf{Prefix}(\mathtt{M}^1, \ldots, \mathtt{M}^s)$
502 $C \leftarrow \mathbb{Y}[\mathtt{P}^s]$
503 for $j \leftarrow \|\mathtt{P}^s\|_n + 1$ to m do
504 $X \leftarrow C \oplus \mathtt{M}^s_j$
505 if $X \in \mathrm{Domain}(\pi)$ then $bad \leftarrow \mathsf{true}$
506 $\pi(X) \leftarrow \mathsf{defined}$
507 $C \leftarrow \mathbb{Y}[\mathtt{M}^s_{1\to j}] \xleftarrow{\$} \{0,1\}^n$

600 for $s \leftarrow 1$ to q do Game C6
601 $\mathtt{P}^s \leftarrow \mathsf{Prefix}(\mathtt{M}^1, \ldots, \mathtt{M}^s)$
602 $C \leftarrow \mathbb{Y}[\mathtt{P}^s]$
603 $X \leftarrow C \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|_n+1}$
604 if $X \in \mathrm{Domain}(\pi)$ then $bad \leftarrow \mathsf{true}$
605 $\pi(X) \leftarrow \mathsf{defined}$
606 $C \leftarrow \mathbb{Y}[\mathtt{M}^s_{1\to\|\mathtt{P}^s\|_n+1}] \xleftarrow{\$} \{0,1\}^n$
607 for $j \leftarrow \|\mathtt{P}^s\|_n + 2$ to m do
608 $X \leftarrow C \oplus \mathtt{M}^s_j$
609 if $X \in \mathrm{Domain}(\pi)$ then $bad \leftarrow \mathsf{true}$
610 $\pi(X) \leftarrow \mathsf{defined}$
611 $C \leftarrow \mathbb{Y}[\mathtt{M}^s_{1\to j}] \xleftarrow{\$} \{0,1\}^n$

700 for $X \in \{0,1\}^+$ do $\mathbb{Y}[X] \xleftarrow{\$} \{0,1\}^n$ Game C7
701 for $s \leftarrow 1$ to q do
702 $\mathtt{P}^s \leftarrow \mathsf{Prefix}(\mathtt{M}^1, \ldots, \mathtt{M}^s)$
703 if $\mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|_n+1} \in \mathrm{Domain}(\pi)$ then $bad \leftarrow \mathsf{true}$
704 $\pi(\mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|_n+1}) \leftarrow \mathsf{defined}$
705 for $j \leftarrow \|\mathtt{P}^s\|_n + 2$ to m do
706 if $\mathbb{Y}[\mathtt{M}^s_{1\to j-1}] \oplus \mathtt{M}^s_j \in \mathrm{Domain}(\pi)$ then $\mathsf{bad} \leftarrow \mathsf{true}$
707 $\pi(\mathbb{Y}[\mathtt{M}^s_{1\to j-1}] \oplus \mathtt{M}^s_j) \leftarrow \mathsf{defined}$

800 for $X \in \{0,1\}^+$ do $\mathbb{Y}[X] \xleftarrow{\$} \{0,1\}^n$ Game C8
801 for $s \leftarrow 1$ to q do
802 $\mathtt{P}^s \leftarrow \mathsf{Prefix}(\mathtt{M}^1, \ldots, \mathtt{M}^s)$
803 if $\mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|_n+1} \in \mathrm{Domain}(\pi)$ then$\mathsf{bad} \leftarrow \mathsf{true}$
804 $\pi(\mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|_n+1}) \leftarrow \mathsf{defined}$
805 for $j \leftarrow \|\mathtt{P}^s\|_n + 1$ to $m - 1$ do
806 if $\mathbb{Y}[\mathtt{M}^s_{1\to j}] \oplus \mathtt{M}^s_{j+1} \in \mathrm{Domain}(\pi)$ then $\mathsf{bad} \leftarrow \mathsf{true}$
807 $\pi(\mathbb{Y}[\mathtt{M}^s_{1\to j}] \oplus \mathtt{M}^s_{j+1}) \leftarrow \mathsf{defined}$

900 for $X \in \{0,1\}^+$ do $\mathbb{Y}[X] \xleftarrow{\$} \{0,1\}^n$ Game C9
901 for $s \leftarrow 1$ to q do $\mathtt{P}^s \leftarrow \mathsf{Prefix}(\mathtt{M}^1, \ldots, \mathtt{M}^s)$
902 $\mathsf{bad} \leftarrow \exists (r,i) \neq (s,j)(r \leq s)(i \geq \|\mathtt{P}^r\|_n + 1)(j \geq \|\mathtt{P}^s\|_n + 1)$
903 $\mathbb{Y}[\mathtt{P}^r] \oplus \mathtt{M}^r_{\|\mathtt{P}^r\|_n+1} = \mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|_n+1}$ and $r < s$ or
904 $\mathbb{Y}[\mathtt{M}^r_{1\to i}] \oplus \mathtt{M}^r_{i+1} = \mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|_n+1}$ or
905 $\mathbb{Y}[\mathtt{M}^r_{1\to i}] \oplus \mathtt{M}^r_{i+1} = \mathbb{Y}[\mathtt{M}^s_{1\to j}] \oplus \mathtt{M}^s_{j+1}$ or
906 $\mathbb{Y}[\mathtt{P}^r] \oplus \mathtt{M}^r_{\|\mathtt{P}^r\|_n+1} = \mathbb{Y}[\mathtt{M}^s_{1\to j}] \oplus \mathtt{M}^s_{j+1}$

Figure 7.6: Games used in the CBC MAC analysis. Let $\mathsf{Prefix}(M^1, \ldots, M^s)$ be ε if $s = 1$, else the longest string $P \in (\{0,1\}^n)^*$ s.t. P is a prefix of M^s and M^r for some $r < s$. In each game, **Initialize** sets $\mathbb{Y}[\varepsilon] \leftarrow 0^n$.

is never changed. The value defined (introduced at line 306) is an arbitrary point of $\{0,1\}^n$, say 0^n. Finally, $\mathsf{Prefix}(M^1,\ldots,M^s)$ is the longest string of blocks $P = P_1 \cdots P_p$ that is a prefix of M^s and is also a prefix of M^r for some $r < s$. If Prefix is applied to a single string the result is the empty string, $\mathsf{Prefix}(P^1) = \varepsilon$. As an example, letting A, B, and C be distinct blocks, $\mathsf{Prefix}(\mathtt{ABC}) = \varepsilon$, $\mathsf{Prefix}(\mathtt{ACC}, \mathtt{ACB}, \mathtt{ABB}, \mathtt{ABA}) = \mathtt{AB}$, and $\mathsf{Prefix}(\mathtt{ACC}, \mathtt{ACB}, \mathtt{BBB}) = \varepsilon$.

We briefly explain the game chain up until the terminal game. Game C0 is obtained from game C1 by dropping the assignment statements that immediately follow the setting of bad. Game **C1** is a realization of $\mathrm{CBC}^m[\mathsf{Perm}(n)]$ and game **C0** is a realization of $\mathsf{Func}(mn,n)$. Games C1 and C0 are designed so that the fundamental lemma applies, so the advantage of A in attacking the CBC construction is at most $\Pr[A^{\mathrm{C0}} \text{ sets } \mathsf{bad}]$. **C0→C2**: The C0 → C2 transition is a lossy transition that takes care of bad getting set at line 105, which clearly happens with probability at most $(0 + 1 + \cdots + (mq-1))/2^n \le 0.5\, m^2q^2/2^n$, so $\Pr[A^{\mathrm{C0}} \text{ sets } \mathsf{bad}] \le \Pr[A^{\mathrm{C2}} \text{ sets } \mathsf{bad}] + 0.5\, m^2q^2/2^n$. **C2→C3**: Next notice that in game C2 we never actually use the values assigned to π, all that matters is that we *record* that a value had been placed in the domain of π, and so game C3 does just that, dropping a fixed value $\mathsf{defined} = 0^n$ into $\pi(X)$ when we want X to join the domain of π. **C3→C4**: Now notice that in game C3 the value returned to the adversary, although dropped into $\mathbb{Y}[M_1^s \cdots M_m^s]$, is never subsequently used in the game so we could as well choose a random value Z^s and return it to the adversary, doing nothing else with Z^s. This is the change made for game C4. The transition is conservative. **C4→C5**: Changing game C4 to C5 is by the "coin-fixing" technique. Coin-fixing in this case amounts to letting the adversary choose the sequence of queries $\mathtt{M}^1,\ldots,\mathtt{M}^m$ it asks and the sequence of answers returned to it. The queries still have to be valid: each M^s is an mn-bit string different from all prior ones: that is the query/response set. For the worst $\mathtt{M}^1,\ldots,\mathtt{M}^m$, which the coin-fixing technique fixes, $\Pr[A^{\mathrm{C4}} \text{ sets } \mathsf{bad}] \le \Pr[\mathrm{C5} \text{ sets } \mathsf{bad}]$. Remember that, when applicable, coin-fixing is safe. **C5→C6**: Game C6 unrolls the first iteration of the loop at lines 503–507. This transformation is conservative. **C6→C7**: Game C7 is a rewriting of game C6 that omits mention of the variables C and X, directly using values from the $\mathbb{Y}$-table instead, whose values are now chosen at the beginning of the game. The change is conservative. **C7→C8**: Game C8 simply re-indexes the for loop at line 705. The change is conservative. **C8→C9**: Game C9 restructures the setting of bad inside the loop at 802–807 to set bad in a single statement. Points were into the domain of π at lines 804 and 807 and we checked if any of these points coincide with specified other points at lines 803 and 806. The change is conservative.

At this point, we have only to bound $\Pr[A^{\mathrm{C9}} \text{ sets } \mathsf{bad}]$. We do this using the sum bound and a case analysis. Fix any $r, i,\ s, j$ as specified in line 902. Consider the following ways that bad can get set to true.

Line 903. We first bound $\Pr[\mathbb{Y}[\mathtt{P}^r] \oplus \mathtt{M}^r_{\|\mathtt{P}^r\|_n+1} = \mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|_n+1}]$. If $\mathtt{P}^r = \mathtt{P}^s = \varepsilon$ then $\Pr[\mathbb{Y}[\mathtt{P}^r] \oplus \mathtt{M}^r_{\|\mathtt{P}^r\|_n+1} = \mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|_n+1}] = \Pr[\mathtt{M}^r_1 = \mathtt{M}^s_1] = 0$ because $\mathtt{M}^r$ and $\mathtt{M}^s$, having only ε as a common block prefix, must differ in their first block. If $\mathtt{P}^r = \varepsilon$ but $\mathtt{P}^s \ne \varepsilon$ then $\Pr[\mathbb{Y}[\mathtt{P}^r] \oplus \mathtt{M}^r_{\|\mathtt{P}^r\|_n+1} = \mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|_n+1}] = \Pr[\mathtt{M}^r_1 = \mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|_n+1}] = 2^{-n}$ since the probability expression involves the single random variable $\mathbb{Y}[\mathtt{P}^s]$ that is uniformly distributed in $\{0,1\}^n$. If $\mathtt{P}^r \ne \varepsilon$ and $\mathtt{P}^s = \varepsilon$ the same reasoning applies. If $\mathtt{P}^r \ne \varepsilon$ and $\mathtt{P}^s \ne \varepsilon$ then $\Pr[\mathbb{Y}[\mathtt{P}^r] \oplus \mathtt{M}^r_{\|\mathtt{P}^r\|_n+1} = \mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|_n+1}] = 2^{-n}$ unless $\mathtt{P}^r = \mathtt{P}^s$, so assume that to be the case. Then $\Pr[\mathbb{Y}[\mathtt{P}^r] \oplus \mathtt{M}^r_{\|\mathtt{P}^r\|_n+1} = \mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|_n+1}] = \Pr[\mathtt{M}^r_{\|\mathtt{P}^r\|_n+1} = \mathtt{M}^s_{\|\mathtt{P}^s\|_n+1}] = 0$ because $\mathtt{P}^r = \mathtt{P}^s$ is the *longest* block prefix that coincides in $\mathtt{M}^r$ and $\mathtt{M}^s$.

Line 904. We want to bound $\Pr[\mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|_n+1} = \mathbb{Y}[\mathtt{M}^r_{1\to i}] \oplus \mathtt{M}^r_{i+1}]$. If $\mathtt{P}^s = \varepsilon$ then $\Pr[\mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|_n+1} = \mathbb{Y}[\mathtt{M}^r_{1\to i}] \oplus \mathtt{M}^r_{i+1}] = \Pr[\mathtt{M}^s_{\|\mathtt{P}^s\|_n+1} = \mathbb{Y}[\mathtt{M}^r_{1\to i}] \oplus \mathtt{M}^r_{i+1}] = 2^{-n}$ because it involves a single random value $\mathbb{Y}[\mathtt{M}^r_{1\to i}]$. So assume that $\mathtt{P}^s \ne \varepsilon$. Then $\Pr[\mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|_n+1} = \mathbb{Y}[\mathtt{M}^r_{1\to i}] \oplus \mathtt{M}^r_{i+1}] = 2^{-n}$ unless

$\mathtt{P}^s = \mathtt{M}^r_{1\to i}$ in which case we are looking at $\Pr[\mathtt{M}^s_{\|\mathtt{P}^s\|_n+1} = \mathtt{M}^r_{\|\mathtt{P}^s\|_n+1}]$. But this is 0 because $\mathtt{P}^s = \mathtt{M}^r_{1\to i}$ means that the longest prefix that $\mathtt{M}^s$ shares with $\mathtt{M}^r$ is $\mathtt{P}^s$ and so $\mathtt{M}^s_{\|\mathtt{P}^s\|_n+1} \neq \mathtt{M}^r_{\|\mathtt{P}^s\|_n+1}$.

Line 905. What is $\mathbb{Y}\,[\mathtt{M}^s_{1\to j}] \oplus \mathtt{M}^s_{j+1} = \mathbb{Y}\,[\mathtt{M}^r_{1\to i}] \oplus \mathtt{M}^r_{i+1}$. It is 2^{-n} unless $i = j$ and $\mathtt{M}^s_{1\to j} = \mathtt{M}^r_{1\to i}$. In that case $\|\mathtt{P}^s\|_n \geq j$ and $\|\mathtt{P}^r\|_n \geq i$, contradicting our choice of allowed values for i and j at line 902.

Line 906. We must bound $\Pr[\mathbb{Y}\,[\mathtt{P}^r] \oplus \mathtt{M}^r_{\|\mathtt{P}^r\|_n+1} = \mathbb{Y}\,[\mathtt{M}^s_{1\to j}] \oplus \mathtt{M}^s_{j+1}]$. As before, this is 2^{-n} unless $\mathtt{P}^r = \mathtt{M}^s_{1\to j}$ but we can not have that $\mathtt{P}^r = \mathtt{M}^s_{1\to j}$ because $j \geq \|\mathtt{P}^s\|_n + 1$.

There are at most $0.5m^2q^2$ tuples (r, i, s, j) considered at line 902 and we now know that for each of them bad gets set with probability at most 2^{-n}. So $\Pr[\text{Game C9 sets } \mathsf{bad}] \leq 0.5m^2q^2/2^n$. Combining with the loss from the C0→C2 transition we have that $\Pr[\text{Game C0 sets}\mathsf{bad}] \leq m^2q^2/2^n$, completing the proof. ∎

7.8 The universal-hashing approach

We have shown that one paradigm for making a good MAC is to make something stronger: a good PRF. Unfortunately, out-of-the-box PRFs usually operate on strings of some fixed length, like 128 bits. That's almost certainly not the domain that we want for our MAC's message space. In this section we describe a simple paradigm for extending the domain of a PRF by using a universal hash-function family. Several MACs can be seen as instances of this approach.

Definition 7.8.1 Let $H\colon \mathcal{K} \times \mathcal{M} \to \{0,1\}^n$ and let δ be a real number. We say that H is δ-AU (read this as δ almost-universal) if for all distinct $M, M' \in \mathcal{M}$, $\Pr[K \xleftarrow{\$} \mathcal{K} :\ H_K(M) = H_K(M')] \leq \delta$.

Definition 7.8.2 Let $H\colon \mathcal{K} \times \mathcal{M} \to \{0,1\}^n$ and $F\colon \mathcal{K}' \times \{0,1\}^n \to \{0,1\}^\tau$ be function families. Then $F \circ H$ is the function family $F \circ H\colon (\mathcal{K} \times \mathcal{K}') \times \mathcal{M} \to \{0,1\}^\tau$ defined by $F \circ H_{(K,K')}(M) = F_{K'}(H_K(M))$.

Theorem 7.8.3 Let $H\colon \mathcal{K} \times \mathcal{M} \to \{0,1\}^n$ be a δ-AU function family and let $F = \mathsf{Func}(n,\tau)$ be the family of all functions from n bits to τ bits. Let A be an adversary that asks at most q queries. Then $\mathbf{Adv}^{\mathrm{prf}}_{F\circ H}(A) \leq \binom{q}{2}\delta^2$.

To be continued. Give a CW mod-p arithmetic MAC. Then describe EMAC and CMAC, and HMAC, probably in different sections.

7.9 Problems

Problem 1 Consider the following variant of the CBC MAC, intended to allow one to MAC messages of arbitrary length. The construction uses a blockcipher $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, which you should assume to be secure. The domain for the MAC is $(\{0,1\}^n)^+$. To MAC M under key K compute $\mathrm{CBC}_K(M \,\|\, |M|)$, where $|M|$ is the length of M, written in n bits. Of course K has k bits. Show that this MAC is completely insecure: break it with a constant number of queries.

Problem 2 Consider the following variant of the CBC MAC, intended to allow one to MAC messages of arbitrary length. The construction uses a blockcipher $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, which you should assume to be secure. The domain for the MAC is $(\{0,1\}^n)^+$. To MAC M under key (K, K') compute $\mathrm{CBC}_K(M) \oplus K'$. Of course K has k bits and K' has n bits. Show that this MAC is completely insecure: break it with a constant number of queries.

Problem 3 Let $\mathrm{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme and let $\mathrm{MA} = (\mathcal{K}', \mathrm{MAC}, \mathrm{VF})$ be a message authentication code. Alice (A) and Bob (B) share a secret key $K = (K1, K2)$ where $K1 \leftarrow \mathcal{K}$ and $K2 \leftarrow \mathcal{K}'$. Alice wants to send messages to Bob in a private and authenticated way. Consider her sending each of the following as a means to this end. For each, say whether it is a secure way or not, and briefly justify your answer. (In the cases where the method is good, you don't have to give a proof, just the intuition.)

(a) $M, \mathrm{MAC}_{K2}(\mathcal{E}_{K1}(M))$

(b) $\mathcal{E}_{K1}(M, \mathrm{MAC}_{K2}(M))$

(c) $\mathrm{MAC}_{K2}(\mathcal{E}_{K1}(M))$

(d) $\mathcal{E}_{K1}(M), \mathrm{MAC}_{K2}(M)$

(e) $\mathcal{E}_{K1}(M), \mathcal{E}_{K1}(\mathrm{MAC}_{K2}(M))$

(f) $C, \mathrm{MAC}_{K2}(C)$ where $C = \mathcal{E}_{K1}(M)$

(g) $\mathcal{E}_{K1}(M, A)$ where A encodes the identity of Alice; B decrypts the received ciphertext C and checks that the second half of the plaintext is "A".

In analyzing these schemes, you should assume that the primitives have the properties guaranteed by their definitions, but no more; for an option to be good it must work for *any* choice of a secure encryption scheme and a secure message authentication scheme.

Now, out of all the ways you deemed secure, suppose you had to choose one to implement for a network security application. Taking performance issues into account, do all the schemes look pretty much the same, or is there one you would prefer?

Problem 4 Refer to problem 4.3. Given a blockcipher E: $\mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ construct a cipher E': $\mathcal{K}' \times \{0,1\}^{2n} \to \{0,1\}^{2n}$. Formalize and prove a theorem that shows that E' is a secure PRP if E is.

Problem 5 Let H: $\{0,1\}^k \times D \to \{0,1\}^L$ be a hash function, and let $\Pi = (\mathcal{K}, \mathrm{MAC}, \mathrm{VF})$ be the message authentication code defined as follows. The key-generation algorithm $\mathcal{K}$ takes no inputs and returns a random k-bit key K, and the tagging and verifying algorithms are:

algorithm $\mathrm{MAC}_K(M)$	algorithm $\mathrm{VF}_K(M, \mathit{Tag}')$
$\mathit{Tag} \leftarrow H(K, M)$	$\mathit{Tag} \leftarrow H(K, M)$
return Tag	if $\mathit{Tag} = \mathit{Tag}'$ then return 1
	else return 0

Show that

$$\mathbf{Adv}^{\text{cr2-hk}}_{H}(t, q, \mu) \;\leq\; (q-1) \cdot \mathbf{Adv}^{\text{uf-cma}}_{\Pi}(t', q-1, \mu, q-1, \mu)$$

for any t, q, μ with $q \geq 2$, where t' is $t + O(\log(q))$. (This says that if Π is a secure message authentication code then H was a CR2-HK secure hash function.)

Bibliography

[1] M. Bellare, J. Kilian and P. Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences* , Vol. 61, No. 3, Dec 2000, pp. 362–399.

[2] M. Bellare, R. Canetti and H. Krawczyk. Keying hash functions for message authentication. *Advances in Cryptology – CRYPTO '96*, Lecture Notes in Computer Science Vol. 1109, N. Koblitz ed., Springer-Verlag, 1996.

[3] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: Fast and secure message authentication. *Advances in Cryptology – CRYPTO '99*, Lecture Notes in Computer Science Vol. 1666, M. Wiener ed., Springer-Verlag, 1999.

[4] J. Black and P. Rogaway. CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions. *Advances in Cryptology – CRYPTO '00*, Lecture Notes in Computer Science Vol. 1880, M. Bellare ed., Springer-Verlag, 2000.