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," since it authenticates the point-of-origin for each message. Message authentication is said to protect the "integrity" of messages, ensuring that each that 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 out. If a hacker can call into his bank's central computer and produce deposit transactions that *appear* to be coming from a branch office, easy wealth is just around the corner. If an unprivilaged 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. An Internet interlouper who can provide bogus financial data to on-line investors by making the data *seem* to have come from a reputable source when it does not might induce an enemy to make a disasterous investment.

In all of these 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—under S's identity. The Receiver R will be tricked into believing that M origiates with S. Because of this wrong belief, R may act on M in a way that is somehow inappropriate.

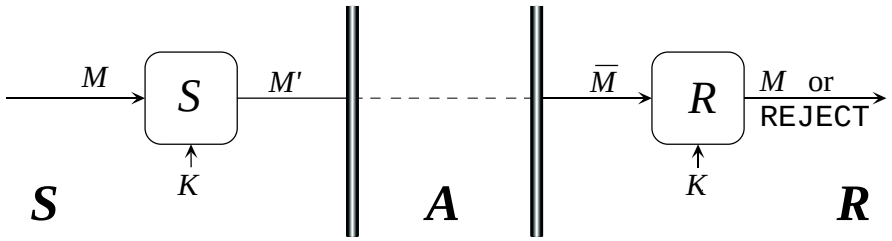

Figure 7.1: A message-authentication scheme. Sender S wants to send a message M to receiver R in such a way that R will be sure that M came from S. They share key K. Adversary A controls the communication channel. Sender S sends an authenticated version of M, M', which adversary A may or may not pass on. On receipt of a message $\overline{M}$, receiver R either recovers a message that S really sent, or else R gets an indication that $\overline{M}$ is inauthentic.

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 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 currently 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.

Authenticating messages may be something done for the benefit of the Receiver R, but the Sender S will certainly need to help out—he'll have to *authenticate* each of his messages. See Fig. 7.1. To authenticate a message M using the key K the legitimate Sender will apply some "message-authenticating algorithm" $\mathcal{S}$ to K and M, giving rise to an "authenticated message" M'. The sender S will transmit the authenticated message M' to the receiver R. Maybe the Receiver will get R—and then again, maybe not. The problem is that an adversary A controls the channel on which messages are being sent. Let's let $\overline{M}$ be the message that the Receiver actually gets. The receiver R, on receipt of $\overline{M}$, will apply some "message-recovery algorithm" to K and $\overline{M}$. We want that this should yield one of two things: (1) the original message M, or else (2) an indication that $\overline{M}$ should not be regarded as authentic.

Often the authenticated message M' is just the original message M together with a fixed-length "tag." The tag serves to validate the authenticity of the message M. In this case we call the message-authentication scheme a *message authentication code*, or *MAC*. See Fig. 7.2

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 possiblities for denial-of-service attacks. Or 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.

Adversarial success in violating the authenticity of messages demands an active attack: to succeed, the adversary has to get some bogus data to the receiver R. If the attacker just watches S and R commuicate she hasn't won this game. In some communication scenerios it may be difficult for the adversary to get her own messages to the receiver R—it might not *really* control the communication channel. For example, it may be difficult for an adversary to drop its own messages

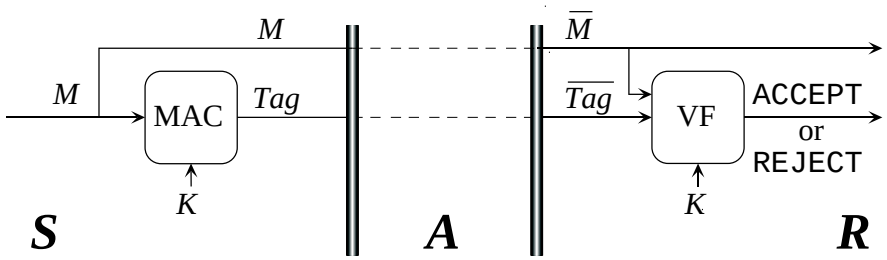

Figure 7.2: A message authentication code (MAC). A MAC is a special-case of a message-authentication scheme, where the authenticated message is the original message M together with a tag Tag. The adversary controls the channel, so we can not be sure that M and Tag reach their intended destination. Instead, the Receiver gets $\overline{M}, \overline{T}$. The Receiver will apply a verification function to K, $\overline{M}$ and $\overline{T}$ to decide if $\overline{M}$ should be regarded as the transmitted message, M, or as the adversary's creation.

onto a dedicated phone line or network link. In other environments it may be trivial, no harder than dropping a packet onto the Internet. 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 the communications media (and even some power over influencing what messages are legitimately sent out).

We wish to emphasize that the authentication problem is very different from the encryption 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 whether she undersands the contents of the communication. Indeed, as we shall see, encryption provides no ready solution for message authentication.

7.2 Privacy does not imply authenticity

We know how to encrypt data so as to provide privacy, and something often suggested (and done) is to encrypt as a way to provide data authenticity, too. 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} which encodes the character "1" with the XOR of the bytes which encode "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.

There are many possible reactions to this counter-example. Let's look at some.

What you should *not* conclude is that one-time pad encryption is unsound. The goal of encryption was to provide privacy, and nothing we have said has suggested that one-time pad encryption does not. Faulting an 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 need an airplane, not a car, if you want to fly.

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 encypts 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 in our one-time pad example.

What you *should* conclude is that encrypting a message 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 encryption was not designed for authenticating messages, it very rarely does. We emphasize this because the belief that good encryption, perahaps 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 of message-authentication schemes

A message authentication scheme $\mathcal{MA} = (\mathcal{K}, \mathcal{S}, \mathcal{R})$ is simply a symmetric encryption scheme, consisting of a triple of algorithms. What is changed is the security goal, which is no longer privacy but authenticity. For this reason we denote and name the scheme and some of the algorithms differently. What used to be called the encryption algorithm is now called the message authenticating algorithm; what used to be called the decryption algorithm is now called the message recovery algorithm.

As we indicated already, a message-authentication code (MAC) is the special case of a message-authentication scheme in which the authenticated message M' consists of M together with a fixed-length string, *Tag*. Usually the length of the tag is between 32 and 128 bits. MACs of 32 bits, 64 bits, 96 bits, and 128 bits are common.

It could be confusing, but it is very common practice to call the tag itself a MAC. That is, the scheme itself is called MAC, but so too is the computed tag.

Definition 7.1 [MAC] A *message-authentication code* Π consists of three algorithms, $\Pi = (\mathcal{K}, \mathrm{MAC}, \mathrm{VF})$, as follows:

- The randomized *key generation* algorithm $\mathcal{K}$ returns a string K. We let $\mathsf{Keys}(\Pi)$ denote the set of all strings that have non-zero probability of being output by $\mathcal{K}$. The members of this set are called *keys*. We write $K \xleftarrow{\$} \mathcal{K}$ for the operation of executing $\mathcal{K}$ and letting K denote the key returned.
- The *MAC-generation* algorithm MAC, which might be randomized or stateful, takes a key $K \in \mathsf{Keys}(\Pi)$ and a *plaintext* $M \in \{0,1\}^*$ to return a *tag* $\mathit{Tag} \in \{0,1\}^* \cup \{\perp\}$. We write $\mathit{Tag} \xleftarrow{\$} \mathrm{MAC}_K(M)$ to denote the operation of executing MAC on K and M and letting Tag denote the tag returned.
- The deterministic *MAC-verification* algorithm VF takes a key $K \in \mathsf{Keys}(\mathcal{SE})$, a message $M \in \{0,1\}^*$ and a candidate tag $\mathit{Tag} \in \{0,1\}^*$ to return either 1 (ACCEPT) or 0 (REJECT). We write $d \leftarrow \mathrm{VF}_K(M, \mathit{Tag})$ to denote the operation of executing VF on K, M and Tag and letting d denote the decision bit returned.

We require that for any key $K \in \mathsf{Keys}(\Pi)$ and any message $M \in \{0,1\}^*$

$$\Pr\left[\ \mathit{Tag} = \perp \text{ OR } \mathrm{VF}_K(M, \mathit{Tag}) = 1 \ :\ \mathit{Tag} \xleftarrow{\$} \mathrm{MAC}_K(M)\ \right] \ =\ 1\ .$$

A number $\tau \geq 1$ is called the *tag-length* associated to the scheme if for any key $K \in \mathsf{Keys}(\Pi)$ and any message $M \in \{0,1\}^*$

$$\Pr\left[\ \mathit{Tag} = \perp \text{ OR } |\mathit{Tag}| = \tau \ :\ \mathit{Tag} \xleftarrow{\$} \mathrm{MAC}_K(M)\ \right] \ =\ 1\ . \blacksquare$$

Any message authentication code gives rise to an associated message authentication scheme in which the authenticated message consists of the message together with the tag. In more detail, if $\Pi = (\mathcal{K}, \mathrm{MAC}, \mathrm{VF})$ is a message authentication code, then its associated message authentication scheme is $\mathcal{MA} = (\mathcal{K}, \mathcal{S}, \mathcal{R})$ where the key-generation algorithm remains unchanged and

Algorithm $\mathcal{S}_K(M)$	Algorithm $\mathcal{R}_K(M')$
$\quad \mathit{Tag} \xleftarrow{\$} \mathrm{MAC}_K(M)$	$\quad$ Parse M' as (M, Tag)
$\quad M' \leftarrow (M, \mathit{Tag})$	$\quad$ If $\mathrm{VF}_K(M, \mathit{Tag}) = 1$ then return 1 else return 0
$\quad$ Return M'	

Let us make a few comments about Definition 7.1. First, we emphasize that, so far, we have only defined MAC and message-authentication scheme "syntax"—we haven't yet said anything formal about security. Of course any viable message-authentication scheme will require some security properties. We'll get there in a moment. But first we needed to pin down exactly what type of objects we're talking about.

Note that our definitions don't permit stateful message-recovery or stateful MAC-verfication. Stateful functions for the Receiver can be problematic because of the possiblity of messages not reaching their destiation—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 essiential for detecting "replay attacks," and are therefore important tools.

Recall that it was essential for security of an encryption scheme that the encryption algorithm be probabilistic or stateful—you couldn't do well at achieving our strong notions of privacy with a determinisitic encryption algorithm. But this isn't true for message authentication. It is possible (and even common) to have secure message authentication schemes in which the message-authenticating algorithm is deterministic or stateless, and to have secure message-authentication codes in which the MAC-generation algorithm is deterministic or stateless.

When the MAC-generation algorithm is deterministic and stateless, MAC verification is invariably accomplished by having the Verifier compute the correct tag for the received message M (using the MAC-generation function) and checking that it matches the received tag. That is, the MAC-verification function is simply the following:

algorithm $\mathrm{VF}_K(M, \mathit{Tag})$
  $\mathit{Tag}' \leftarrow \mathrm{MAC}_K(M)$
  if $(\mathit{Tag} = \mathit{Tag}'$ and $\mathit{Tag}' \neq \perp)$ **then return** 1 **else return** 0.

For a deterministic MAC we need only specify the key-generation function and the MAC-generation function: the MAC-verification function is then understood to be the one just described. That is, a deterministic MAC may be specified with a pair of functions, $\Pi = (\mathcal{K}, \mathrm{MAC})$, and not a triple of functions, $\Pi = (\mathcal{K}, \mathrm{MAC}, \mathrm{VF})$, with the understanding that one can later refer to VF and it is the canonical algorithm depicted above.

7.4 A definition of security for MACs

Let's concentrate on MACs. We begin with a discussion of the issues and then state a formal definition.

7.4.1 Towards a definition of security

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 validitating the message, then that should have been considered as part of the message-authentication scheme. 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. Asking that the adversary be unable to forge "meaningful" messages, whatever that might mean, would again be asking too little.

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, Tag) which passes the verification test, 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, Tag) consisting of some message M and its legitimate tag Tag. The Receiver will certainly accept this—we demanded that. Now at once a simple attack comes to mind: the adversary can just repeat this transmission, (M, Tag), and get the Receiver to accept it once again. This attack is unavoidable, so far, in that we required in the syntax of a MAC for the MAC-verification functions to be stateless. If the Verifier accepted (M, Tag) once, he's bound to do it again.

What we have just described is called a *replay attack*. The adversary sees a valid (M, Tag) 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 MAC'ed 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 MAC which is not secure against replay attacks and modify it—after making the Verifier 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 MACs the adversary's ability to forge will grow with the number q_s of legitimate message-MAC pairs it sees. Likewise, in some sucurity systems the number of valid (M, Tag) 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 MAC, chooses another message, gets its MAC, and so forth. Then it tries to forge. This is called an *adaptive chosen-message attack*. It wins if it succeds 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 unrealisticly generous to our adversary; after all, if an adversary could really obtain a valid MAC for *any* message it wanted, wouldn't that make moot the whole point of authenticting 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 modelling 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 scenerio 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 to forward its adaptively-chosen messages $M_1, M_2, \ldots$ to the Reciever. 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, even though most it has correctly used a MAC.

So far we have said that we want to give our adversary the ability to obtain MACs for messages of her choosing, and then we want to look at whether or not it can forge: produce a valid (M, Tag) 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 possiblity by giving the adversary the capability to tell if a prospective (M, Tag) pair is valid, and saying that the adversary forges if it ever finds an (M, Tag) 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 Verifier is going to tear down a connection the moment he detects an invalid tag. Then it is unrealistic to try to use this Verifier to help you determine if a candidate pair (M, Tag) 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 adequtate.

On the other hand, suppose that a Verifier just ignores any improperly tagged message, while it responds in some noticably 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, Tag) pairs. The adversary hopes to find at least one that is valid. When the adversary finds such an (M, Tag) pair, we'll say that it has won.

Let us summarize. To be fully general, we will give our adversary two different capabities. The first adversarial capaiblity 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, Tag) 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, Tag) 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 viture of an earlier signing query. Let us now proceed more formally.

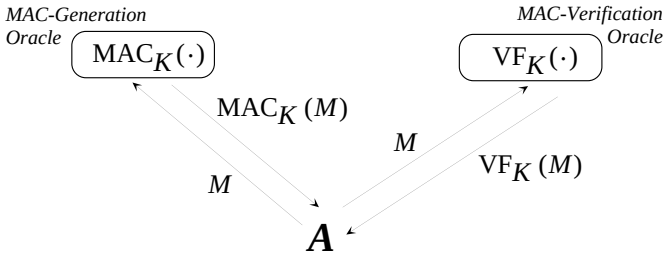

Figure 7.3: *The model for a message authentication code. Adversary A has access to a MAC-generation oracle and a MAC-verification oracle. The adversary wants to get the MAC-verification oracle to accept some (M, Tag) for which it didn't earlier ask the MAC-generation oracle for M.*

7.4.2 Definition of security

Let $\mathcal{MA} = (\mathcal{K}, \mathrm{MAC}, \mathrm{VF})$ be an arbitrary message authentication scheme. We will formalize a quantitative notion of security against adpative 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 adversray can use to authenticate any message M. This "signing 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 something to whom to send attempted forgeries. So we will embody the Verifier as an oracle that the adversray can use to see if a candidate pair (M, Tag) is valid. This "verification oracle," as we will call it, is our way to provide the adversary black-box access to the function $\mathrm{VF}_K(\cdot)$. Thus, when we become formal, the cast of characters—the Sender, Verifier, and Adversary—gets reduced to just the adversry, running with her oracles. The Sender and Verifier have vanished.

Definition 7.2 [**MAC Security**] Let $\Pi = (\mathcal{K}, \mathrm{MAC}, \mathrm{VF})$ be a message authentication code, and let A be an adversary. We consider the following experiment:

Experiment $\mathbf{Exp}^{\text{uf-cma}}_{\Pi}(A)$
$K \xleftarrow{\$} \mathcal{K}$
Run $A^{\mathrm{MAC}_K(\cdot),\mathrm{VF}_K(\cdot,\cdot)}$
If A made a verification query (M, Tag) such that the following are true
– The verification oracle returned 1
– A did not, prior to making verification query (M, Tag), make signing query M
Then return 1 else return 0

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

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

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

We emphasize that there are contexts in which you are happy with a MAC that makes forgery impractical when $q_{\text{v}} = 1$ and $q_{\text{s}} = 0$ (an "impersonation attack") and there are contexts in which you are happy when forgery is imporactical when $q_{\text{v}} = 1$ and $q_{\text{s}} = 1$ (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.

We might talk of the total length of an adversary's MAC-generation oracle queries, which is the sum of the lengths of all messages it queries to this oracle. When we say this value is at most μ_s we mean it is so across all possible coins of the adversary and all possible answers returned by the oracle. We might talk of the total length of an adversary's MAC-verification oracle queries, which is the sum of the lengths of all messages in the queries its makes to its MAC-verification oracle. (Each such query is a pair, but we count only the length of the message). The same conventions apply.

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.

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: $\{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^L$. Our first scheme $\Pi_1 = (\mathcal{K}, \text{MAC})$ is a deterministic, stateless MAC, so that we specify only two algorithms, the third being the canonical associated verification algorithm discussed above. The key-generation algorithm simply picks at random a k-bit key K and returns it, while the MAC-generation algorithm works as follows:

algorithm $\text{MAC}_K(M)$
  if $(|M| \bmod \ell \neq 0$ or $|M| = 0)$ **then return** $\perp$
  Break M into ℓ bit blocks $M = M[1] \dots M[n]$
  for $i = 1, \dots, n$ **do** $y_i \leftarrow F_K(M[i])$
  $\mathit{Tag} \leftarrow y_1 \oplus \cdots \oplus y_n$
  return Tag

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 hence will have a hard time computing Tag. However, 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 ℓ-bit string, set $Tag \leftarrow 0^L$, and make verification query (M, Tag). Notice that $\mathrm{VF}_K(M, Tag) = 1$ because $F_K(x) \oplus F_K(x) = 0^L = Tag$. So the adversary is successful. In more detail, the adversary is:

Adversary $A_1^{\mathrm{MAC}_K(\cdot),\mathrm{VF}_K(\cdot,\cdot)}$
  Let x be some ℓ-bit string
  $M \leftarrow x \,\|\, x$
  $Tag \leftarrow 0^L$
  $d \leftarrow \mathrm{VF}_K(M, Tag)$

Then $\mathbf{Adv}^{\text{uf-cma}}_{\Pi_1}(A_1) = 1$. Furthermore A_1 makes no signing oracle queries, uses $t = O(\ell + L)$ time, and its verification query has length 2ℓ-bits, so it is very practical.

There are many other attacks. For example we note that

$$Tag \;=\; F_K(M[1]) \oplus F_K(M[2])$$

is not only the tag of $M[1]M[2]$ but also the tag of $M[2]M[1]$. 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 we pick some parameter m with $1 \le m \le \ell - 1$, and write the index as an m-bit string. The MAC-generation algorithm of the deterministic, stateless MAC $\Pi_1 = (\mathcal{K}, \mathrm{MAC})$ is as follows:

algorithm $\mathrm{MAC}_K(M)$
  $l \leftarrow \ell - m$
  if $(|M| \bmod l \neq 0$ or $|M| = 0$ or $|M|/l \geq 2^m)$ **then return** $\perp$
  Break M into l bit blocks $M = M[1] \ldots M[n]$
  for $i = 1, \ldots, n$ **do** $y_i \leftarrow F_K([i]_m \,\|\, M[i])$
  $Tag \leftarrow y_1 \oplus \cdots \oplus y_n$
  return Tag

As before, the verification algorithm is the canonical one that simply recomputes the tag using MAC and checks whether it is correct.

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 $l = \ell - m$ 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 m-bit string is only possible if $i < 2^m$. This means that we cannot authenticate a message M having more 2^m blocks. This explains the conditions under which the MAC-generation algorithm returns $\perp$. However this is hardly a restriction in practice since a reasonable value of m, like $m = 32$, is large enough that typical messages fall in the message space.

Anyway, the question we are really concerned with is the security. Has this improved with respect to Π_1? Begin by noticing that the attacks we found on Π_1 no longer work. For example if x is an $\ell - m$ bit string and we let $M = x \,\|\, x$ then its tag is *not* likely to be 0^L. (This would happen only if $F_K([1]_m \,\|\, x) = F_K([2]_m \,\|\, x)$ which is unlikely if F is a good PRF and impossible if F is a block cipher, since every instance of a block cipher is a permutation.) Similar arguments show that the second attack discussed above, namely that based on permuting of message blocks, also has low success against the new scheme. Why? In the new scheme, if $M[1], M[2]$ are strings of length $\ell - m$, then

$$\begin{aligned} \mathrm{MAC}_K(M[1]M[2]) &= F_K([1]_m \,\|\, M[1]) \oplus F_K([2]_m \,\|\, M[2]) \\ \mathrm{MAC}_K(M[2]M[1]) &= F_K([1]_m \,\|\, M[2]) \oplus F_K([2]_m \,\|\, M[1]) \,. \end{aligned}$$

These are unlikely to be equal for the same reasons discussed above. 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.

However, Π_2 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–

Adversary $A_2^{\mathrm{MAC}_K(\cdot)}$
  Let a_1, b_1 be distinct, $\ell - m$ bit strings
  Let a_2, b_2 be distinct $\ell - m$ bit strings
  $Tag_1 \leftarrow \mathrm{MAC}_K(a_1 a_2)$; $Tag_2 \leftarrow \mathrm{MAC}_K(a_1 b_2)$; $Tag_3 \leftarrow \mathrm{MAC}_K(b_1 a_2)$
  $Tag \leftarrow Tag_1 \oplus Tag_2 \oplus Tag_3$
  $d \leftarrow \mathrm{VF}_K(b_1 b_2, Tag)$

We claim that $\mathbf{Adv}^{\text{uf-cma}}_{\Pi_2}(A_2) = 1$. Why? This requires two things. First that $\mathrm{VF}_K(b_1 b_2, Tag) = 1$, and second that $b_1 b_2$ 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 $b_1 b_2 \notin \{a_1 a_2, a_1 b_2, b_1 a_2\}$. So now let us check the first claim. We use the definition of the tagging algorithm to see that

$$\begin{aligned} Tag_1 &= F_K([1]_m \,\|\, a_1) \oplus F_K([2]_m \,\|\, a_2) \\ Tag_2 &= F_K([1]_m \,\|\, a_1) \oplus F_K([2]_m \,\|\, b_2) \\ Tag_3 &= F_K([1]_m \,\|\, b_1) \oplus F_K([2]_m \,\|\, a_2) \,. \end{aligned}$$

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

$$\begin{aligned} Tag &= Tag_1 \oplus Tag_2 \oplus Tag_3 \\ &= F_K([1]_m \,\|\, b_1) \oplus F_K([2]_m \,\|\, b_2) \,. \end{aligned}$$

This is indeed the correct tag of $b_1 b_2$, meaning the value Tag' that $\mathrm{VF}_K(b_1 b_2, Tag)$ 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 cryptanalysis of the message authentication schemes 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 $F\colon \mathsf{Keys} \times D \to \{0,1\}^\tau$ be a family of functions. We define the *associated message authentication code* $\Pi = (\mathcal{K}, \mathrm{MAC})$ via:

algorithm $\mathcal{K}$	**algorithm** $\mathrm{MAC}_K(M)$
$K \xleftarrow{\$} \mathsf{Keys}$	**if** $(M \notin D)$ **then return** $\perp$
return K	$\mathit{Tag} \leftarrow F_K(M)$
	Return Tag

Since this is a deterministic stateless MAC, we have not specified a verification algorithm. It is understood to be the canonical one discussed above.

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 block cipher. 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 lenghts. 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 7.3 Let $F\colon \mathsf{Keys} \times D \to \{0,1\}^\tau$ be a family of functions and let $\Pi = (\mathcal{K}, \mathrm{MAC})$ be the associated message authentication code as defined above. Let A by any adversary attacking Π, making q_{s} MAC-generation queries of total length μ_s, q_{v} MAC-verification queries of total length μ_v, and having running time t. Then there exists an adversary B attacking F such that

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

Furthermore B makes $q_{\mathrm{s}} + q_{\mathrm{v}}$ 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.

Adversary 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 $f(M) = \mathit{Tag}$ **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

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

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

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 $\mathbf{Exp}_{\Pi}^{\text{uf-cma}}(A)$. 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 $\mathit{Tag} = f(M)$. Since there are at most q_{v} verification queries, Equation (7.3) follows. ▌

7.7 The CBC MACs

A very popular class of MACs is obtained via cipher-block chaining of a given block cipher. Here is the most basic scheme in this class.

Scheme 7.4 [Basic CBC MAC] Let E: $\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher. The basic CBC MAC $\Pi = (\mathcal{K}, \text{MAC})$ is a deterministic, stateless MAC that has as a parameter an associated message space Messages. The key-generation algorithm $\mathcal{K}$ simply picks K via $K \xleftarrow{\$} \{0,1\}^k$ and returns K. The MAC generation algorithm is as follows:

Algorithm $\text{MAC}_K(M)$
  If $M \notin \mathsf{Messages}$ then return $\perp$
  Break M into n-bit blocks $M[1] \cdots M[m]$
  $C[0] \leftarrow 0^n$
  For $i = 1, \ldots, m$ do $C[i] \leftarrow E_K(C[i-1] \oplus M[i])$
  Return $C[m]$

See Fig. 7.4 for an illustration with $m = 4$. The verification algorithm VF is the canonical one since this MAC is deterministic: It just checks, on input (K, M, Tag), if $\mathit{Tag} = \text{MAC}_K(M)$. ▌

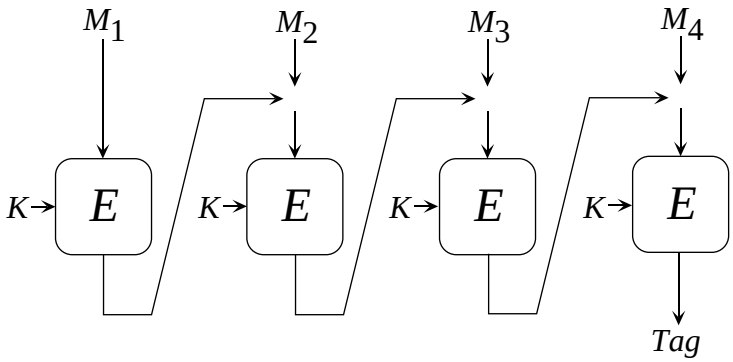

Figure 7.4: *The CBC MAC, here illustrated with a message M of four blocks, $M = M_1M_2M_3M_4$.*

As we will see below, the choice of message space Messages is very important for the security of the CBC MAC. If we take it to be $\{0,1\}^{mn}$ for some fixed value m, meaning the length of all authenticated messages is the same, then the MAC is secure. If however we allow the generation of CBC-MACs for messages of different lengths, by letting the message space be the set of all strings having length a positive multiple of n, then the scheme is insecure.

Theorem 7.5 [1] Let E: $\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher, let $m \geq 1$ be an integer, and let Π be the CBC-MAC of Scheme 7.4 over message space $\{0,1\}^{mn}$. Then for any adversary A making at most q MAC-generation queries, one MAC-verification query and having running time t there exists an adversary B, making $q+1$ oracle queries and having running time t, such that

$$\mathbf{Adv}^{\text{uf-cma}}_{\Pi}(A) \;\leq\; \mathbf{Adv}^{\text{prp-cpa}}_{E}(B) + \frac{m^2q^2}{2^{n-1}} \;.\blacksquare$$

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.