Last Revised: February 24, 2000

Chapter 6

Message Authentication

In this chapter we address message authentication, the second major goal in cryptography. 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 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 the public-key setting is the problem addressed by *digital signature*). 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.

6.1 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. If an Internet interlouper can provide bogus financial data to on-line investors, he can make data *seem* to have come from a reputable source when it does not, perhaps inducing 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 them come from some other party, S, the (legidimate) 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.

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 it in Chapter **??**.

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 Figure 6.1. To authenticate a message M using the key K the legidimate Sender will apply some "message-authenticating function" $\mathcal{S}$ to K and M, giving rise 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 function" 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

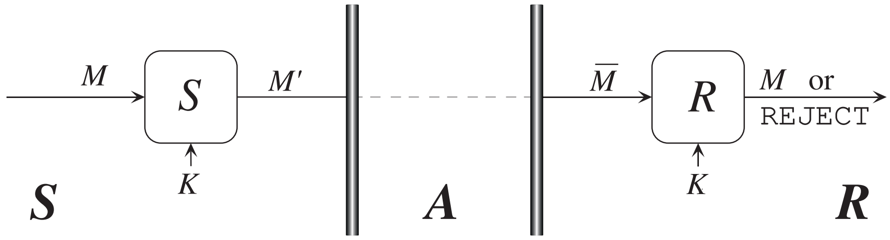

Figure 6.1: A message-authentication scheme. Sender S wants to send a message M to receiver R in such a way that R will be sure that M came from S. They share key K. Adversary A controls the communication channel. Sender S sends an authenticated version of M, M', which adversary A may or may not pass 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.

as authentic.

Usually 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 Figure 6.2

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 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.

Unlike encryption, 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—she might not *really* control the communication channel. For example, it may be difficult for an adversary to drop her own messages 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

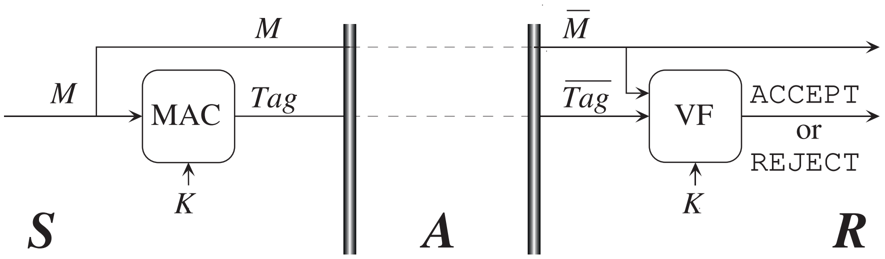

Figure 6.2: A message authentication code (MAC). A MAC is a special-case of a message-authentication scheme, where the authenticated message is the original message M together with a tag Tag. The adversary controls the channel, so we can not be sure that M and Tag reach their intended destination. Instead, the Receiver gets $\overline{M}, \overline{Tag}$. The Receiver will apply a verification function to K, $\overline{M}$ and $\overline{Tag}$ to decide if $\overline{M}$ should be regarded as the transmitted message, M, or as the adversary's creation.

has plenty of power over the communications media (and even some power over influencing what messages are legidimately 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.

6.2 Encryption does not provide 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 data authenticity, too. Fix a symmetric encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, 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 $C \stackrel{R}{\leftarrow} \mathcal{E}_K(M)$. 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 screwdriver because you can not use it to cut vegetables. 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 any 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 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. These authors or programmers do not understand how the cryptographic community has separated out our various goals, with encrpytion being the tool for achieving privacy, and for achieving that goal alone. Hapilly, we have other tools for achieving message authenticity.

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.

6.3 Syntax of message-authentication schemes

A message authentication scheme Π has three components. The first is the function $\mathcal{K}$ which generates the shared keys for the Sender and the Receiver. The second is the function $\mathcal{S}$ that takes a key K and a message M and produces an authenticated message M'. The third is the function $\mathcal{R}$ that takes a key K and what is supposed to be an authenticated message M', and returns either an underlying message M or else an indication that the message M' should not be rejected.

There are some details to take care of. We pin these down in the following definition.

Definition 6.3.1 A ***message-authentication scheme*** Π consists of three functions, $\Pi = (\mathcal{K}, \mathcal{S}, \mathcal{R})$, and associated sets $\mathsf{Key}(\Pi)$, $\mathit{Msgs}(\Pi) \subseteq \{0,1\}^*$, and $\mathit{Auth}(\Pi) \subseteq \{0,1\}^*$, as follows:

- The ***key-generation function*** $\mathcal{K}$ is a probablistic function which takes no inputs and returns a value $K \in \mathsf{Key}(\Pi)$, called a *key*. We write $K \stackrel{R}{\leftarrow} \mathcal{K}$ to denote the chosing of K by computing $\mathcal{K}$.
- The ***message-authenticating function*** $\mathcal{S}$ is a deterministic, probablistic, or stateful function which takes a key $K \in \mathsf{Key}(\Pi)$ and a string $M \in \{0,1\}^*$ and ruturns a value M', where $M' \in \mathit{Auth}(\Pi)$ if $M \in \mathit{Msgs}(\Pi)$ and $M' = \text{ERROR}$ if $M \notin \mathit{Msgs}(\Pi)$. We write

$M' \stackrel{R}{\leftarrow} \mathcal{S}_K(M)$ to denote the choosing of M' by computing $\mathcal{S}$ on K and M.

- The ***message-recovery function*** $\mathcal{R}$ is a deterministic function which takes a key $K \in \mathsf{Key}(\Pi)$ and a string $M' \in \{0,1\}^*$ and returns a value M, where $M \in \mathit{Msgs}(\Pi)$ or $M = \text{REJECT}$. We write $M \leftarrow \mathcal{R}_K(M')$ to denote the choosing of M by computing $\mathcal{S}$ on K and M'.

We require that if $K \in \mathsf{Key}(\Pi)$ and $M' \stackrel{R}{\leftarrow} \mathcal{R}_K(M)$ and $M' \neq \text{ERROR}$, then $\mathcal{S}_K(M') = M$. ▌

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 it the computed tag. It's not really a problem, you'll be able to keep them straight.

Since MACs are so important let us take the time to specialize Definition 6.3.1 for MACs.

Definition 6.3.2 [MAC] A ***message-authentication code*** Π consists of three functions, $\Pi = (\mathcal{K}, \text{MAC}, \text{VF})$, and associated sets $\mathsf{Key}(\Pi)$, $\mathit{Msgs}(\Pi) \subseteq \{0,1\}^*$, and $\mathit{Tags}(\Pi) = \{0,1\}^{\mathit{tagLen}(\Pi)}$, for some number $\mathit{tagLen}(\Pi) \geq 1$, as follows:

- The ***key-generation function*** $\mathcal{K}$ is a probablistic function which takes no inputs and returns a value $K \in \mathsf{Key}(\Pi)$, called a *key.* We write $K \stackrel{R}{\leftarrow} \mathcal{K}$ to denote the chosing of K by computing $\mathcal{K}$.
- The ***MAC-generation function*** MAC is a deterministic, probablistic, or stateful function which takes a key $K \in \mathsf{Key}(\Pi)$ and a string $M \in \{0,1\}^*$ and returns a value Tag, called the "MAC" or "tag," where $\mathit{Tag} \in \mathit{Tags}(\Pi)$ if $M \in \mathit{Msgs}(\Pi)$ and $\mathit{Tag} = \text{ERROR}$ otherwise. We write $\mathit{Tag} \stackrel{R}{\leftarrow} \text{MAC}_K(M)$ to denote the choosing of Tag by computing MAC on K and M.
- The ***MAC-verification function*** VF is a deterministic function which takes a key $K \in \mathsf{Key}(\Pi)$, a string $M \in \{0,1\}^*$, and a string $\mathit{Tag} \in \{0,1\}^*$, and returns a value of ACCEPT or REJECT. We write $b \leftarrow \text{VF}_K(M, \mathit{Tag})$ to denote the choosing of b by computing VF on K, M, and Tag.

We require that if $K \in \mathsf{Key}(\Pi)$ and $M \in \mathit{Msgs}$ and $\mathit{Tag} \xleftarrow{R} \mathrm{MAC}_K(M)$ then $\mathrm{VF}_K(M, \mathit{Tag}) = \textsc{accept}$. If $M \notin \mathit{Msgs}$ or $\mathit{Tag} \notin \mathit{Tags}$ then $\mathrm{VF}_K(M, \mathit{Tag}) = \textsc{reject}$. ▮

Let us pause and make a few comments about Definitions 6.3.1 and 6.3.2. 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 exactly is the type of objects we're talking about.

Next the reader should notice that we said that, in both definitions, Key was a set—we didn't say it was a set of strings. The added generality lets us say things like "the key is a pair of functions, h and f," or "the key contains an infinite sequence of numbers, each between 0 and $2^{32} - 1$." Now since K might not be a string we can't very well speak of $(\mathcal{K}, \mathcal{S}, \mathcal{R})$, or $(\mathcal{K}, \mathrm{MAC}, \mathrm{VF})$, as algorithms: algorithms map strings to strings. So we called these things functions. Recall that we made the same definitional choice when we defined encryption schemes, and for the same reason—it's just too convenient to sometimes speak of message-authentication schemes which depend on infinite objects. Now in any "practical" message-authentication scheme the set of keys $\mathsf{Key}(\Pi)$ will need to be a finite set, and the functions $\mathcal{K}$, MAC and VF will all be given by algorithms—-hopefully, quite efficient ones! But allowing greater generality is a useful intermediate step. Not to worrry, we'll have only strings and algorithms before we're through.

Note that our definitions we didn't permit stateful message recovery or MAC-verfication. Stateful functions for he 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. We will eventually allow stateful verification. We take up this issue in Section **??**.

When we defined encryption, it was essential for security that the encryption functionj be probabilistic or stateful—you couldn't do well at achieving our strong notion of security with a determinisitic encryption function. But this isn't true for message authentication: it is no problem for the the message-authenticating function (or MAC-generation function) to be deterministic. In fact, most MACs do use deterministic MAC-generation functions. In this case, 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:

Function $\mathrm{VF}_K(M, \mathit{Tag})$
  $\mathit{Tag}' \leftarrow \mathrm{MAC}_K(M)$
  If $\mathit{Tag} = \mathit{Tag}'$ then return ACCEPT else return REJECT.

For a deterministic MAC we'll only specify the key-generation function and the MAC-generatio function: the MAC-verification function is then understood to be the one just described. That is, a deterministic MAC is specified with a pair of functions, $\Pi = (\mathcal{K}, \mathrm{MAC})$, and not a triple of functions, $\Pi = (\mathcal{K}, \mathrm{MAC}, \mathrm{VF})$.

6.4 Example message-authentication schemes

Before getting to the business of defining security for message-authentication schemes let us look at some important examples of message authentication schemes. We will be concrete, to give you a feel for the type of message-authentication codes which people use.

Example 6.4.1 [CBC MAC] Let E: $\{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a block cipher, such RC6, and let $m \geq 1$ be a number. The CBC MAC uses the block cipher E to generate an n-bit MAC for messages whose lengths are some multiple of n. More eplicitly, the m-fold CBC MAC over E is the deterministic message authentication code $\mathrm{CBC}^m(E)$ in which the tag of an m-block message is the last block of ciphertext obtained by processing the message in CBC mode with zero IV. In more detail, the scheme $\mathrm{CBC}^m(E) = (\mathcal{K}, \mathrm{MAC})$ is defined as follows. The key space is $\mathsf{Key} = \{0,1\}^k$. The message space is $\mathit{Msgs} = \{0,1\}^{nm}$. The tag space is $\mathit{Tags} = \{0,1\}^n$. The key-generation algorithm is the algorithm which picks a random k-bit string K and outputs it. The MAC-generation algorithm takes a message M and does the following:

Algorithm $\mathrm{MAC}_K(M)$
  Divide M into n-bit blocks, $M = M_1 \cdots M_m$
  $C_0 \leftarrow 0^n$
  For $i = 1, \ldots, m$ do $C_i \leftarrow E_K(C_{i-1} \oplus M_i)$
  Return C_m

See Figure 6.2.

Since the MAC is deterministic, the MAC-verification algorithm is understood. It just checks, on input (K, M, Tag), if $\mathit{Tag} = \mathrm{MAC}_K(M)$.

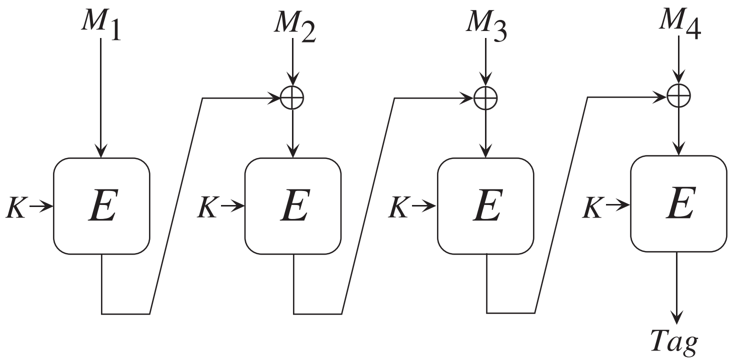

Figure 6.3: *The CBC MAC, here illustrated with a message M of four blocks, $M = M_1M_2M_3M_4$.*

As we will see later, the choice of message space is important for the security of the CBC MAC. If we had taken the message space to be $Msgs = \cup_{m\geq 1}\{0,1\}^{mn}$, as may seem natural, this MAC will be insecure. But if the length of the messages is restricted to some single fixed value, as above, the scheme is secure. We will address security in Section **??**. ▮

Example 6.4.2 [HMAC] *Describe HMAC.* ▮

Example 6.4.3 [Encrypting a MAC'ed Message] Both of the examples given above were MACs. Here is an example of a message-authentication scheme which is not a MAC. ...

6.5 Towards a Definition of Security

Rather than put down a definition for security as though it fell from the sky, let us spend some time to build up our intuition about what properties a message-authentication code should have to deserve to be called "secure". Let's concentrate on MACs, as it will be easy to lift our definition to general message-authentication schemes.

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 $\mathrm{VF}_K(M, Tag) = 1$, 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 she could do this, it would be a disaster, since she 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 she 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 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 she 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 she 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 she 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: she just keeps replaying this same MAC'ed message, hapilly 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.

If the adversary wants to be successful she can take a valid pair (M, Tag) and use it to concoct pair valid pair (M', Tag') such that $M \neq M'$. If she can do this, she has won. This is sometimes called a *subsitution attack*. In a substitution attack the adversary takes a single message and tag, (M, Tag), and uses them in her attempt to forge.

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 q_s—that is, her forgery probablity can be expected to grow with the number of examples of legitimately authenticated messages. 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 *herself* 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 she tries to forge. This is called an *adaptive chosen-message* *attack*. It is the same type of attack that we concentrated on in defining secure encryption.

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 she wanted, wouldn't that make moot the whole point of authenticting messages be useless? 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 she 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 $\langle i \rangle \parallel M \parallel MAC_K(\langle i \rangle \parallel M)$. 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, she will ask the Sender to forward her adaptively-chosen messages $M_1, M_2, \ldots$ to the Reciever. If, based on what she sees, she can learn the key K, or even if she can learn to forge message of the form $\langle 2 \rangle \parallel M$, so as to produce a valid $\langle 2 \rangle \parallel M \parallel MAC_K(\langle 2 \rangle \parallel M)$, 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 she can forge: produce a valid (M, Tag) where she never asked the adversary to MAC M. But we should recognize that a realistic adversary might be able to produce lots of candidate forgeries, and she 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 she ever finds an (M, Tag) pair that is. We'll rule out messages that the adversry already knows a tag for.

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 she responds in some noticably different way if she 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 she 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 she 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 she ever makes a verification query (M, Tag) and gets a return value of 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.

6.6 Definition of security

Let $\Pi = (\mathcal{K}, \mathrm{MAC}, \mathrm{VF})$ 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 begin by distilling out the model from the intuition we have described. 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,

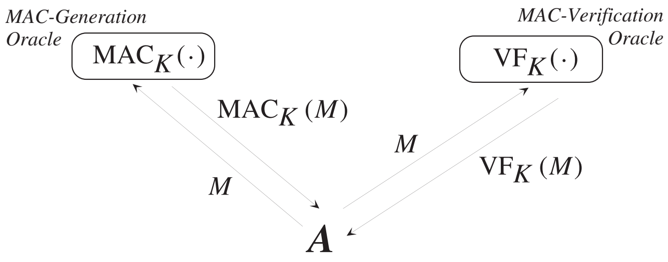

Figure 6.4: *The model for a message authentication code. Adversary A has access to a MAC-generation oracle and a MAC-verification oracle. The adversary wants to get the MAC-verification oracle to accept some (M, Tag) for which she didn't earlier ask the MAC-generation oracle for M.*

the cast of characters—the Sender, Verifier, and Adversary—gets reduced to just he adversry, running with her oracles. The Sender and Verifier have vanished—reduced to oracles, poor things.

Here, in detail, is how the game is run—the the experiment which defines whether or not the adversary wins when she attacks the message-authentication code $\Pi = (\mathcal{K}, \mathrm{MAC}, \mathrm{VF})$.

Definition 6.6.1 [MAC Security] Let $\Pi = (\mathcal{K}, \mathrm{MAC}, \mathrm{VF})$ be a message authentication code, and let A be an adversary. Let $\mathbf{Adv}_{\Pi}^{\mathrm{mac}}(A)$ denote the probability that A succeeds in the following experiment:

> Let $K \stackrel{R}{\leftarrow} \mathcal{K}$
> Run $A^{\mathrm{MAC}_K(\cdot), \mathrm{VF}_K(\cdot,\cdot)}$
> If A ever asksed a MAC-verification query $\mathrm{VF}_K(M, Tag)$,
> getting a return value of ACCEPT, and A did not earlier ask
> MAC-generation query $\mathrm{MAC}_K(M)$, then A *succeeds*;
> else A *fails*.

Let $q_{\mathrm{s}}, q_{\mathrm{v}}, m, t \geq 0$ be numbers. Then we let

$$\mathbf{Adv}_{\Pi}^{\mathrm{mac}}(q_{\mathrm{s}}, q_{\mathrm{v}}, m, t) = \max_{A} \{ \mathbf{Adv}_{\Pi}^{\mathrm{mac}}(A) \}$$

where the maximum is over all adversaries A that make at most q_{s} signing queries, at most q_{v} verification queries, each signing and verification query is of length at most m, and the adversary's total running time is t. ∎

Let us discuss the above definition. Fix a MAC scheme Π. Then we associate to any adversary A is its "advantage," or "success probability." We denote this value as $\mathbf{Adv}^{\mathrm{mac}}_{\Pi}(A)$. 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. The insecurity of the MAC itself, which we also denote with $\mathbf{Adv}$, is the success probability of the "cleverest" possible adversary, amongst all adversaries restricted to specified computational resources.

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 she 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 emphaisize that there are contexts in which you are happy with a MAC that makes forgery impracitical 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 imporactical when $q_{\mathrm{v}} = 1$ and $q_{\mathrm{s}} = 1$ (a "subsitution 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 maybewhen q_{v} is large, too.

The maximal length of each message provided to an oracle, m, is a resource that could well be parameterized in different ways. For example, we could, alternatively, have looked at the total length of all queries, m. Or we could look at the $(q_{\mathrm{s}} + q_{\mathrm{v}})$-vector which specifies the length of each of the oracle queries. The point is simply that getting lots of bits of MACed message may be more difficult that getting a few bits. Perhaps the Signer won't MAC messages that a gigabytes long—and no doubt sending such long messages takes longer than sending short ones. We choose to parameterize by maximal message length simply because it is convenient for the results we will show.

In some MAC schemes security does not depend on the adversary's running time, t, being bounded. In this case the scheme is said to be "information-theoretically secure." Likewise, in some MAC schemes security does not degrade with m or q_{s}.

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.

6.7 Example schemes

Mihir, I'm unconvinced that this is really needed, but maybe examples never hurt....

Let us examine some example message authentication schemes and use the definition to assess their strengths and weaknesses. We fix a PRF $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$. Our first scheme $\Pi_1 = (\mathcal{K}, \mathrm{MAC}, \mathrm{VF})$ works like this–

Algorithm $\mathrm{MAC}_K(M)$	Algorithm $\mathrm{VF}_K(M, \sigma)$
Divide M into l bit blocks, $M = x_1 \ldots x_n$	Divide M into l bit blocks, $M = x_1 \ldots x_n$
For $i = 1, \ldots, n$ do $y_i \leftarrow F_K(x_i)$	For $i = 1, \ldots, n$ do $y_i \leftarrow F_K(x_i)$
$\sigma \leftarrow y_1 \oplus \cdots \oplus y_n$	$\sigma' \leftarrow y_1 \oplus \cdots \oplus y_n$
Return σ	If $\sigma = \sigma'$ then return 1 else return 0

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

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 σ. 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 $\sigma \leftarrow 0^l$, and output M, σ. Notice that $\mathrm{VF}_K(M, \sigma) = 1$ because $F_K(x) \oplus F_K(x) = 0^l = \sigma$. So the adversary is successful. In more detail, the adversary is:

Adversary $A_1^{\mathrm{MAC}_K(\cdot)}$
  Let x be some l-bit string
  Let $M \leftarrow x \,\|\, x$
  Let $\sigma \leftarrow 0^l$
  Return (M, σ)

Then $\mathbf{Succ}^{\mathrm{ma}}(\Pi_1, A_1) = 1$. Furthermore A_1 makes no oracle queries, uses $t = O(l)$ time, and outputs an l-bit message in its forgery, so we have shown that

$$\mathbf{Adv}^{\mathrm{ma}}(\Pi_1; t, 0, l) \;=\; 1\,.$$

That is, the scheme Π_1 is totally insecure.

There are many other attacks. For example we note that if $\sigma = F_K(M_1) \oplus F_K(M_2)$ is the tag of $M_1 M_2$ then σ is also the correct 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 $1 \le m \le l-1$, and write the index as an m-bit string. The message authentication scheme $\Pi_1 = (\mathcal{K}, \mathrm{MAC}, \mathrm{VF})$ looks like this:

Algorithm $\mathrm{MAC}_K(M)$	Algorithm $\mathrm{VF}_K(M, \sigma)$
Divide M into $l-m$ bit blocks, $M = x_1 \ldots x_n$	Divide M into $l-m$ bit blocks, $M = x$
For $i = 1, \ldots, n$ do $y_i \leftarrow F_K(\langle i \rangle \,\|\, x_i)$	For $i = 1, \ldots, n$ do $y_i \leftarrow F_K(\langle i \rangle \,\|\, x_i)$
$\sigma \leftarrow y_1 \oplus \cdots \oplus y_n$	$\sigma' \leftarrow y_1 \oplus \cdots \oplus y_n$
Return σ	If $\sigma = \sigma'$ then return 1 else return 0

As the code indicates, we divide M into smaller blocks: not of size l, but of size $l - m$. Then we prefix the i-th message block with the value i itself, the block index, written in binary. Above $\langle i \rangle$ denotes the integer i written as a binary string of 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 than 2^m blocks. That is, the message space is confined to strings of length at most $(l-m)(2^m - 1)$, and, for simplicity, of length a multiple of $l - m$ bits. 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, and since l is typically at least 64, we have at least 32 bits left for the data itself.

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 take the adversary A_1 above. (It needs a minor modification to make sense in the new setting, namely the chosen block x should not be of length l but of length $l - m$. Consider this modification made.) What is its success probability when viewed as an adversary attacking Π_2? The question amounts to asking what is the chance that $\mathrm{VF}_K(M, \sigma) = 1$ where VF is the verification algorithm of our amended scheme and M, σ is the output of A_1. The verification algorithm will compute $\sigma' = F_K(\langle 1 \rangle \parallel x) \oplus F_K(\langle 2 \rangle \parallel x)$ and test whether this equals 0^l, the value of σ output by A. This happens only when

$$F_K(\langle 1 \rangle \parallel x) = F_K(\langle 2 \rangle \parallel x) \ ,$$

and this is rather unlikely. For example if we are using a block cipher it never happens because F_K is a permutation. Even when F is not a block cipher, this event has very low probability as long as F is a good PRF; specifically, $\mathbf{Succ}^{\mathrm{ma}}(\Pi_2, A_1)$ is at most $\mathbf{Adv}^{\mathrm{prf}}_F(t, 2)$ where $t = O(l)$. (A reader might make sure they see why this bound is true.) So the attack has very low success probability.

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

$$\begin{aligned}
\mathrm{MAC}_K(M_1 M_2) &= F_K(\langle 1 \rangle \parallel M_1) \oplus F_K(\langle 2 \rangle \parallel M_2) \\
\mathrm{MAC}_K(M_2 M_1) &= F_K(\langle 1 \rangle \parallel M_2) \oplus F_K(\langle 2 \rangle \parallel M_1) \ .
\end{aligned}$$

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 insecurity of F at appropriate parameter values.

However, Π_2 is still insecure. The attacks 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 x_1, x_1' be distinct, $l - m$ bit strings, and let x_2, x_2' be distinct $l - m$ bit strings
  $\sigma_1 \leftarrow \mathrm{MAC}_K(x_1 x_2)$; $\sigma_2 \leftarrow \mathrm{MAC}_K(x_1 x_2')$; $\sigma_3 \leftarrow \mathrm{MAC}_K(x_1' x_2)$
  $\sigma \leftarrow \sigma_1 \oplus \sigma_2 \oplus \sigma_3$
  Return $(x_1' x_2', \sigma)$

We claim that $\mathbf{Succ}^{\mathrm{ma}}(\Pi_2, A_2) = 1$. Why? This requires two things. First that $\mathrm{VF}_K(x_1' x_2', \sigma) = 1$, and second that $x_1' x_2'$ was never a query to $\mathrm{MAC}_K(\cdot)$ in the above code. The latter is true because we insisted above that $x_1 \neq x_1'$ and $x_2 \neq x_2'$, which together mean that $x_1' x_2' \notin \{x_1 x_2, x_1 x_2, x_1' x_2\}$.

So now let us check the first claim. We use the definition of the tagging algorithm to see that

$$\begin{aligned}\sigma_1 &= F_K(\langle 1\rangle \parallel x_1) \oplus F_K(\langle 2\rangle \parallel x_2)\\ \sigma_2 &= F_K(\langle 1\rangle \parallel x_1) \oplus F_K(\langle 2\rangle \parallel x_2')\\ \sigma_3 &= F_K(\langle 1\rangle \parallel x_1') \oplus F_K(\langle 2\rangle \parallel x_2)\ .\end{aligned}$$

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

$$\begin{aligned}\sigma &= \sigma_1 \oplus \sigma_2 \oplus \sigma_3\\ &= F_K(\langle 1\rangle \parallel x_1') \oplus F_K(\langle 2\rangle \parallel x_2')\ .\end{aligned}$$

This is indeed the correct tag of $x_1'x_2'$, meaning the value σ' that $\mathrm{VF}_K(x_1'x_2', \sigma)$ would compute, so the latter algorithm returns 1, as claimed. In summary we have shown that

$$\mathbf{Adv}^{\mathrm{ma}}(\Pi_2; t, 3, 4(l-m)) \;=\; 1\ ,$$

where $t = O(l)$. So the scheme Π_2 is also totally insecure.

Later we will see how a slight modification of the above 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, RC6, 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.

6.8 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.

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 can'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 restate the definition of a PRF, to make sure that the variable domain is clear.

Definition 6.8.1 A **pseudorandom function** (PRF) is a function $F : \mathsf{Key} \times \mathsf{Message} \to \{0,1\}^n$ where Key is a finite set (or else it comes endowed with a probablity measure), and $\mathsf{Message}$ is a nonempty set of strings.

To make a MAC from a PRF F we are simply setting the key generator to be the algorithm that samples from Key, and we set $\mathrm{MAC}_a(M) = F_a(M)$. The MAC is deterministic, so we don't have to separately specify a MAC-verification function. For notational convenience, we will not distinguish between F as a PRF and the F-induced MAC scheme.

Proposition 6.8.2 Let $F : \mathsf{Key} \times \mathsf{Message} \to \{0,1\}^n$ be a PRF. Suppose that there exists an adversry A_{mac} that, running in time t_{mac}, asking q_{mac} queries, these totalling μ_{mac} bits, forges with probablity $\epsilon_{\mathrm{mac}} = \mathbf{Adv}_F^{\mathrm{mac}}(A_{\mathrm{mac}})$. Then there exists an adversary A_{prf} that, running in time t_{prf}, asking q_{prf} queries, these totalling μ_{prf} bits, distinguishes a random instance of F from a random function with advantage $\epsilon_{\mathrm{prf}} = \mathbf{Adv}_F^{\mathrm{prf}}(A_{\mathrm{prf}})$ where

$$t_{\mathrm{prf}} = \tilde{O}(t_{\mathrm{mac}}), \quad q_{\mathrm{prf}} = q_{\mathrm{mac}} + 1, \quad \mu_{\mathrm{prf}} = \mu_{\mathrm{mac}}, \text{ and } \quad \epsilon_{\mathrm{prf}} = \epsilon_{\mathrm{mac}} - 2^{-n}.$$

Proof: Adversary A_{prf} has an oracle f. Let A_{prf} work as follows

> Run A_{mac}.
> When A_{mac} makes a query, x, to its (MAC) oracle g, return $f(x)$.
> Finally A_{mac} halts, outputting a pair (x^*, σ^*).
> If x^* was not already asked of f, and $f(x^*) = \sigma^*$,
> then output 1, otherwise output 0.

With the obvious shorthand,

$$\begin{aligned}
\mathbf{Adv}_F^{\mathrm{prf}}(A_{\mathrm{prf}}) &= \Pr[A_{\mathrm{prf}}^{F_a(\cdot)} = 1] - \Pr[A^{\rho}\cdot = 1] \\
&\leq \Pr[A_{\mathrm{mac}}^{F_a(\cdot)} \text{ forges}] - 2^{-n} \\
&= \epsilon_{\mathrm{mac}} - 2^{-n}
\end{aligned}$$

The running time and query complexity of $A_{\rm prf}$ are clearly as claimed: the $\tilde{O}$ overhead is for checkig if x^* is a new query, while the $+1$ in the query complexity accounts for asking $f(x^*)$. Recall that, by our convention, $\mu_{\rm mac}$ already includes the length of x^*. ▮

6.9 Making a PRF from a PRF and a Universal Hash Function

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.

Theorem 6.9.1 Let $H : \mathsf{Key}(H) \times \mathsf{Message} \to \{0,1\}^n$ be a δ-AU hash-function family. Let $F : \mathsf{Key}(F) \times \{0,1\}^n \to \{0,1\}^s$ be $\epsilon_F(t,q)$-secure, as a PRF; that is, $\epsilon_F(t,q) = \mathbf{Adv}_F^{\rm prf}(t,q)$. Define the PRF $FH : (\mathsf{Key}(F) \times \mathsf{Key}(H)) \times \mathsf{Message} \to \{0,1\}^s$ by $FH_{(a,k)}(x) = F_a(H_k(x))$. Then FH is (t', q', μ, ϵ')-secure, as a PRF, where $\cdots$.

To be completed.

Let us given a concrete example of this approach. We saw in Chapter 5 that one could hash a sequence of words $M_{m-1}\ldots M_0$ using a key $k \in \{0, 2^{89}-1\}$ by computing $H_k(M) = (k^m + M_{m-1}k^{m-1} + \ldots + M_1 k + M_0) \bmod (2^{89}-1)$. If the message is limited to $2^{32}-1$ words, say, then this hash function family has collision probability bounded by $\frac{2^{32}-1}{2^{89}-1} > 2^{-57}$. So to MAC a message M, compute $H_k(M)$, encode this into a 128-bit string, and apply DES_a, yielding the desired authentication tag.

6.10 An XOR Scheme

Mihir, I'd like to completely revise this, directly proving the Bernstein Bernstein-variant of our XOR MAC (that's the version where you encipher the XOR of the PRF outputs), which has a trivial proof in the above framework.

We now consider a family of message authentication schemes called XOR MACs and show that they are secure.

6.10.1 The schemes

The schemes use a similar paradigm to the second example scheme discussed above. A certain block size l is given. We choose a parameter m, this time $1 \leq m \leq l-2$. We view the message M as being divided into blocks of size $l-m-1$, and denote the i-th block by x_i. We denote by $\langle i \rangle$ the encoding of integer i as an m-bit binary string. We assume that $|M| \leq (l-m-1)(2^m-1)$ so that the index of any block can be written as an m-bit string.

Both to define the schemes and to analyze their security it is helpful to first introduce an auxiliary function which we call XOR-Tag$^?(\cdot,\cdot)$. It takes an oracle for a function $f\colon \{0,1\}^l \to \{0,1\}^L$. It also takes two inputs. The first is an $l-1$ bit string which we call s, and whose role will emerge later. The second is the message M discussed above. It processes these inputs using f as indicated below and returns a value we call τ.

Algorithm XOR-Tag$^f(s, M)$
  Divide M into $l-m-1$ bit blocks, $M = x_1 \ldots x_n$
  $y_0 \leftarrow f(0 \,\|\, s)$
  For $i = 1, \ldots, n$ do $y_i \leftarrow f(1 \,\|\, \langle i \rangle \,\|\, x_i)$
  $\tau \leftarrow y_0 \oplus y_1 \oplus \cdots \oplus y_n$
  Return τ

Our auxiliary function applies f at $n+1$ points, each of these points being an l-bit string. The first point is $0 \,\|\, s$. Namely we prefix the $(l-1)$-bit string s with a zero bit, which brings the total to l-bits, and then apply f to get y_0. The other n points on which f is applied are all prefixed with the bit 1. That is followed by an encoding of the block index and then the data block itself, for a total length of $1 + m + (l-m-1) = l$ bits.

We are now ready to describe the scheme. We fix a family of functions $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$, and the key for the message authentication scheme is simply a k-bit key K for the family F, which specifies a specific function F_K. The parties will use F_K in the role of f above. There are actually two versions of the scheme. One is deterministic and stateful, making use of a global counter; the other is stateless and randomized. The difference is only in how s is chosen. We begin with the counter version. Here the sender maintains an integer counter c, initially 0. We denote by $\langle c \rangle$ its encoding as an $l-1$ bit integer. (The counter thus ranges from 0 to $2^{l-1}-1$. Note that when i is a block index, $\langle i \rangle$ also denotes its binary encoding, but as an m bit string, so that the notation $\langle \cdot \rangle$ is a bit overloaded in that the length of the string returned depends on the context of its argument, but hopefully this will not cause too much confusion.) The *counter-based*

XOR MAC scheme using F, denoted $\mathcal{C}\text{-}\mathcal{XOR}^F = (\mathcal{K}, \text{MAC}, \text{VF})$, works as follows–

Algorithm $\text{MAC}_K(M)$	Algorithm $\text{VF}_K(M, \sigma)$
$\quad \tau \leftarrow \text{XOR-Tag}^{F_K}(\langle c \rangle, M)$	$\quad$ Parse σ as (s, τ)
$\quad \sigma \leftarrow (\langle c \rangle, \tau)$	$\quad \tau' \leftarrow \text{XOR-Tag}^{F_K}(s, M)$
$\quad c \leftarrow c + 1$	$\quad$ If $\tau = \tau'$ then return 1 else return 0
$\quad$ Return σ	

In other words, the tag for message $M = x_1 \ldots x_n$ is a pair consisting of the current counter value c encoded in binary, and the subtag τ, where

$$\tau \quad = \quad F_K(0 \,\|\, \langle c \rangle) \oplus F_K(1 \,\|\, \langle 1 \rangle \,\|\, x_1) \oplus \cdots \oplus F_K(1 \,\|\, \langle n \rangle \,\|\, x_n) \, .$$

To verify the received tag $\sigma = (\langle c \rangle, \tau)$ the verification algorithm recomputes the correct subtag, calling it τ', as a function of the given counter value, and then checks that this subtag matches the one provided in σ.

The randomized version of the scheme, namely the *randomized XOR MAC* scheme using F, is denoted $\mathcal{R}\text{-}\mathcal{XOR}^F = (\mathcal{K}, \text{MAC}, \text{VF})$. It simply substitutes the counter with a random $(l-1)$-bit value chosen anew at each application of the tagging algorithm. In more detail, the algorithms work as follows–

Algorithm $\text{MAC}_K(M)$	Algorithm $\text{VF}_K(M, \sigma)$
$\quad r \stackrel{R}{\leftarrow} \{0,1\}^{l-1}$	$\quad$ Parse σ as (r, τ)
$\quad \tau \leftarrow \text{XOR-Tag}^{F_K}(r, M)$	$\quad \tau' \leftarrow \text{XOR-Tag}^{F_K}(r, M)$
$\quad \sigma \leftarrow (r, \tau)$	$\quad$ If $\tau = \tau'$ then return 1 else return 0
$\quad$ Return σ	

In other words, the tag for message $M = x_1 \ldots x_n$ is a pair consisting of a random value r and the subtag τ where

$$\tau \quad = \quad F_K(0 \,\|\, r) \oplus F_K(1 \,\|\, \langle 1 \rangle \,\|\, x_1) \oplus \cdots \oplus F_K(1 \,\|\, \langle n \rangle \,\|\, x_n) \, .$$

To verify the received tag $\sigma = (r, \tau)$ the verification algorithm recomputes the correct subtag, calling it τ', as a function of the given value r, and then checks that this subtag matches the one provided in σ.

6.10.2 Security considerations

Before we consider security it is important to clarify one thing about possible forgeries. Recall that a forgery is a pair M, σ consisting of a message M and a value σ which purports to be a valid tag for M. In the XOR schemes, a tag σ

is a pair (s, τ) where s is an $(l-1)$-bit string and τ is an L-bit string. Now, we know that the tagging algorithm itself generates the first component in a very specific way. For concreteness, take the counter-based XOR scheme; here s is the value of a counter, and thus for legitimately tagged messages, a value that never repeats from one message to the next. (We assume no more than 2^{l-1} messages are authenticated so that the counter does not wrap around.) This does *not* mean that the adversary is forced to use a counter value in the role of s in its attempted forgery $M, (s, \tau)$. The adversary is free to try any value of s, and in particular to re-use an already used counter value. Remember that the adversary's goal is to get the verification algorithm to accept the pair $M, (s, \tau)$, subject only to the constraint that M was not a query to the tagging oracle. Look at the code of the verification algorithm: it does not in any way reflect knowledge of s as a counter, or try to check any counter-related property of s. In fact the verification algorithm does not maintain a counter at all; it is stateless. So there is nothing to constrain an adversary to use the value of the counter in its attempted forgery.

A similar situation holds with respect to the randomized version of the XOR scheme. Although the legitimate party chooses r at random, so that legitimate tags have random values of r as the first component of their tags, the adversary can attempt a forgery $M, (r, \tau)$ in which r is quite non-random; the adversary gets to choose r and can set it to whatever it wants. This freedom on the part of the adversary must be remembered in analyzing the schemes.

To get some intuition about the security of these schemes, it is helpful to return to the attack that we used to break the example scheme Π_2 in the previous section, and see that it fails here. We will look at the attack in the context of the counter-based XOR scheme. Remember that the attack, specified by adversary A_2 above, requested the tags of three related messages and XORed the returned values to get the tag of the third message, exploiting commonality between the values to get cancellations. For the same three messages under the new scheme, let us look at the subtags returned by the tagging oracle. They are:

$$\begin{aligned}
\text{XOR-Tag}^{F_K}(\langle 0\rangle, x_1x_2) &= F_K(0 \,\|\, \langle 0\rangle) \oplus F_K(1 \,\|\, \langle 1\rangle \,\|\, x_1) \oplus F_K(1 \,\|\, \langle 2\rangle \,\|\, x_2)\\
\text{XOR-Tag}^{F_K}(\langle 1\rangle, x_1x_2') &= F_K(0 \,\|\, \langle 1\rangle) \oplus F_K(1 \,\|\, \langle 1\rangle \,\|\, x_1) \oplus F_K(1 \,\|\, \langle 2\rangle \,\|\, x_2')\\
\text{XOR-Tag}^{F_K}(\langle 2\rangle, x_1'x_2) &= F_K(0 \,\|\, \langle 2\rangle) \oplus F_K(1 \,\|\, \langle 1\rangle \,\|\, x_1') \oplus F_K(1 \,\|\, \langle 2\rangle \,\|\, x_2)\ .
\end{aligned}$$

Summing these three values yields a mess, something that does not look like the subtag of any message, because the values corresponding to the counter don't cancel. So this attack does not work.

Is there another attack? It seems hard to come up with one, but that does not mean much; maybe the attack is quite clever. This is the point where the kind of approach we have been developing, namely provable security, can be instrumental. We will see that the XOR schemes can be proven secure under the assumption that the family F is a PRF. This means that simple attacks like the above do not exist. And our confidence in this stems from much more than an inability to find the attacks; it stems from our confidence that the underlying family F is itself secure.

6.10.3 Results on the security of the XOR schemes

We state the theorems that summarize the security of the schemes, beginning with the counter-based scheme. We call the (nteger) parameter m in the scheme the block-indexing parameter in the following. We also let $\mathrm{MsgSp}(l,m)$ denote the set of all strings M such that the length of M is $n \cdot (l-m-1)$ for some integer n in the range $1 \leq n \leq 2^m - 1$; this is the message space for the XOR message authentication schemes.

The theorem below has (what should by now be) a familiar format. It upper bounds the insecurity of the counter-based XOR message authentication scheme in terms of the insecurity of the underlying PRF F. In other words, it upper bounds the maximum (over all strategies an adversary might try) of the probability that the adversary can break the XOR scheme (namely, successfully forge a correct tag for an as yet un-authenticated message), and the upper bound is in terms of the (assumed known) maximum ability to break the PRF F that underlies the scheme. It is another example of the kind of "punch line" we strive towards: a guarantee that there is simply no attack against a scheme, no matter how clever, as long as we know that the underlying tool is good. In particular we are assured that attacks like those we have seen above on our example schemes will not work against this scheme.

Also as usual, the bounds are quantitative, so that we can use them to assess the amount of security we will get when using some specific PRF (say a block cipher) in the role of F. The bounds are rather good: we see that the chance of breaking the message authentication scheme is hardly more than that of breaking the PRF.

Theorem 6.10.1 Suppose $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ is a PRF, and let $\mathcal{C}\text{-}\mathcal{XOR}^F = (\mathcal{K}, \mathrm{MAC}, \mathrm{VF})$ be the corresponding counter-based XOR message authentication scheme as described above, with block-indexing parameter $m \leq l-2$ and message space $\mathrm{MsgSp}(l,m)$. Then for any t, q, μ with

$q < 2^{l-1}$ we have

$$\mathbf{Adv}^{\mathrm{ma}}(\mathcal{C}\text{-}\mathcal{XOR}^F; t, q, \mu) \quad \leq \quad \mathbf{Adv}_F^{\mathrm{prf}}(t', q') + 2^{-L} ,$$

where $t' = t + O(\mu)$ and $q' = q + 1 + \mu/(l - m - 1)$. ▮

The result for the randomized version of the scheme is similar except for accruing an extra term. This time, there is a "collision probability" type term of $q^2/2^l$ in the upper bound, indicating that we are unable to rule out a breaking probability of this magnitude regardless of the quality of the PRF. We will see later that this is an inherent feature of the scheme, which is subject to a sort of birthday attack.

Theorem 6.10.2 Suppose $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ is a PRF, and let $\mathcal{R}\text{-}\mathcal{XOR}^F = (\mathcal{K}, \mathrm{MAC}, \mathrm{VF})$ be the corresponding randomized XOR message authentication scheme as described above, with block-indexing parameter $m \leq l - 2$ and message space $\mathrm{MsgSp}(l, m)$. Then for any t, q, μ

$$\mathbf{Adv}^{\mathrm{ma}}(\mathcal{R}\text{-}\mathcal{XOR}^F; t, q, \mu) \quad \leq \quad \mathbf{Adv}_F^{\mathrm{prf}}(t', q') + \frac{q^2}{2^l} + 2^{-L} ,$$

where $t' = t + O(\mu)$ and $q' = q + 1 + \mu/(l - m - 1)$. ▮

We will not prove these results at this time. We will be able to do this later when we have developed some more technology.

NOTE: there a section Mihir wrote, mac-x.tex (formerly mac-extra.tex) that gives a proof for xor-counter MAC. But I am not sure we want to cover XOR schemes, anyway

6.11 The EMAC Construction

We wish to show that if $M, M' \in (\{0,1\}^n)^+$ are distinct strings then $\Pr_\pi[\mathrm{CBC}_\pi(M) = \mathrm{CBC}_\pi(M')]$ is small. By "small" we mean a slowly growing function of $m = |M|/n$ and $m' = |M'|/n$. Formally, for $n, m, m' \geq 1$, define the *collision probability* of the CBC MAC to be

$$V_n(m, m') \;\stackrel{\mathrm{def}}{=}\; \max_{M \in \{0,1\}^{nm},\, M' \in \{0,1\}^{nm'},\, M \neq M'} \{\, \Pr[\pi \stackrel{R}{\leftarrow} \mathrm{Perm}(n) :\ \mathrm{CBC}_\pi(M) = \mathrm{CBC}_\pi(M')] \,\} .$$

(The character "V" is meant to suggest collisions.)

Lemma 6.11.1 [CBC MAC Collision Bound] Let $n, m, m' \geq 1$. Then

$$V_n(m, m') \;\leq\; \frac{2.5\,(m + m')^2}{2^n} .$$

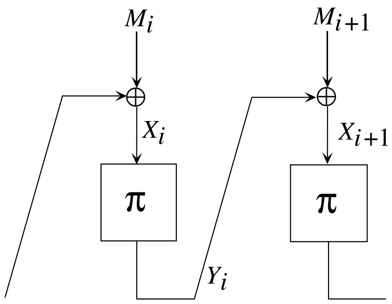

Figure 6.5: *A fragment of the CBC construction showing the labeling convention used in the proof of Lemma* **??**.

Proof: Although M and M' are distinct, they may share some common prefix. Let k be the index of the last block in which M and M' agree. (If M and M' have unequal first blocks then $k = 0$.)
Each particular permutation π is equally likely among all permutations from $\{0,1\}^n$ to $\{0,1\}^n$. In our analysis, we will view the selection of π as an incremental procedure. This will be equivalent to selecting π uniformly at random. In particular, we view the computation of $\mathrm{CBC}_\pi(M)$ and $\mathrm{CBC}_\pi(M')$ as playing the game given in Figure 6.6. Here the notation M_i indicates the ith block of M. We initially set each range point of π as `undefined`; the notation $\mathrm{Domain}(\pi)$ represents the set of points x where $\pi(x)$ is no longer `undefined`. We use $\mathrm{Range}(\pi)$ to denote the set of points $\pi(x)$ which are no longer `undefined`; we use $\overline{\mathrm{Range}}(\pi)$ to denote $\{0,1\}^n - \mathrm{Range}(\pi)$.
During the game, the X_i are those values produced after XORing with the current message block, M_i, and the Y_i values are $\pi(X_i)$. See Figure 6.5.
We are concerned with the probability that π will cause $\mathrm{CBC}_\pi(M) = \mathrm{CBC}_\pi(M')$, which will occur in our game iff $Y_m = Y'_{m'}$. Since π is invertible, this occurs iff $X_m = X'_{m'}$. As we shall see, this condition will cause *bad* = `true` in our game. However, we actually set *bad* to `true` in many other cases in order to simplify the analysis.
The idea behind the variable *bad* is as follows: throughout the program (lines 4, 11, and 16) we randomly choose a range value for π at some `undefined` domain point. Since π has not yet been determined at this point, the selection of our range value will be an independent uniform selection: there is no dependence on any prior choice. If the range value for π were already determined by some earlier choice, the analysis would become more involved. We avoid the latter condition by setting *bad* to `true` whenever such interdependencies are detected. The detection mechanism works as follows:

$bad \leftarrow \mathsf{false}$; **for all** $x \in \{0,1\}^n$ **do** $\pi(x) \leftarrow \texttt{undefined}$; $X_1 \leftarrow M_1$; $X'_1 \leftarrow M'_1$; $BAD \leftarrow \{X_1, X'_1\}$

for $i \leftarrow 1$ **to** k **do**
  if $X_i \in \mathrm{Domain}(\pi)$ **then** $Y_i \leftarrow Y'_i \leftarrow \pi(X_i)$
  else $Y_i \leftarrow Y'_i \stackrel{R}{\leftarrow} \overline{\mathrm{Range}}(\pi)$; $\pi(X_i) \leftarrow Y_i$
    if $i < m$ **then** $X_{i+1} \leftarrow Y_i \oplus M_{i+1}$
      if $X_{i+1} \in BAD$ **then** $bad \leftarrow \mathsf{true}$ **else** $BAD \leftarrow BAD \cup \{X_{i+1}\}$
    if $i < m'$ **then** $X'_{i+1} \leftarrow Y'_i \oplus M'_{i+1}$
      if $X'_{i+1} \in BAD$ **then** $bad \leftarrow \mathsf{true}$ **else** $BAD \leftarrow BAD \cup \{X'_{i+1}\}$

for $i \leftarrow k+1$ **to** m **do**
  if $X_i \in \mathrm{Domain}(\pi)$ **then** $Y_i \leftarrow \pi(X_i)$
  else $Y_i \stackrel{R}{\leftarrow} \overline{\mathrm{Range}}(\pi)$; $\pi(X_i) \leftarrow Y_i$
    if $i < m$ **then** $X_{i+1} \leftarrow Y_i \oplus M_{i+1}$
      if $X_{i+1} \in BAD$ **then** $bad \leftarrow \mathsf{true}$ **else** $BAD \leftarrow BAD \cup \{X_{i+1}\}$

for $i \leftarrow k+1$ **to** m' **do**
  if $X'_i \in \mathrm{Domain}(\pi)$ **then** $Y'_i \leftarrow \pi(X'_i)$
  else $Y'_i \stackrel{R}{\leftarrow} \overline{\mathrm{Range}}(\pi)$; $\pi(X'_i) \leftarrow Y'_i$
    if $i < m$ **then** $X'_{i+1} \leftarrow Y'_i \oplus M'_{i+1}$
      if $X'_{i+1} \in BAD$ **then** $bad \leftarrow \mathsf{true}$ **else** $BAD \leftarrow BAD \cup \{X'_{i+1}\}$

Figure 6.6: *Game used in the proof of Lemma* **??**. *The algorithm gives one way to compute the CBC MAC of distinct messages $M = M_1 \cdots M_m$ and $M' = M'_1 \cdots M'_{m'}$. These messages are identical up to block k, but different afterwards. The computed MACs are Y_m and $Y_{m'}$, respectively.*

throughout the processing of M and M' we will require π be evaluated at $m+m'$ domain points $X_1,\cdots,X_m$ and $X'_1,\cdots,X'_{m'}$. If all of these domain points are distinct (ignoring duplications due to any common prefix of M and M'), we can rest assured that we are free to assign their corresponding range points without constraint. We maintain a set BAD to track which domain points have already been determined; initially X_1 and X'_1 are the only such points, since future values will depend on random choices not yet made. Of course if $k>0$ then $X_1 = X'_1$ and BAD contains only one value. Next we begin randomly choosing range points; if ever any such choice leads to a value already contained in the BAD set, we set the flag *bad* to true.

We now bound the probability of the event that $bad = \mathsf{true}$ by analyzing our game. The variable *bad* can be set true in lines 6, 8, 13, and 18. In each case it is required that some Y_i was selected such that $Y_i \oplus M_{i+1} \in BAD$ (or possibly that some Y'_i was selected such that $Y'_i \oplus M'_{i+1} \in BAD$). The set BAD begins with at most 2 elements and then grows by 1 with each random choice of Y_i or Y'_i. We know that on the ith random choice in the game the BAD set will contain at most $i+1$ elements. And so each random choice of Y_i (resp. Y'_i) from the co-range of π will cause $Y_i \oplus M_{i+1}$ (resp. $Y'_i \oplus M'_{i+1}$) to be in BAD with probability at most $(i+1)/(N-i+1)$. We have already argued that in the absence of $bad = \mathsf{true}$ each of the random choices we make are independent. We make $m-1$ choices of Y_i to produce X_2 through X_m and $m'-1$ choices of Y'_i to determine X'_2 through $X'_{m'}$ and so we can compute

$$\Pr[bad = \mathsf{true}] \;\le\; \sum_{i=1}^{m-1+m'-1} \frac{i+1}{N-i+1}.$$

Using the fact that $m, m' \le N/4$, we can bound the above by

$$\sum_{i=1}^{m+m'-2} \frac{i+1}{N-i} \;\le\; \frac{2}{N}\sum_{i=1}^{m+m'-2} i+1 \;\le\; \frac{(m+m')^2}{N}\ .$$

This completes the proof. ▌

6.12 The HMAC Construction

6.13 The UMAC Construction

Sketch UMAC, particularly improving security with the use of counters (the "standard" Wegman-Carter method), and the hash function NH). Old material follows.

Today the most effective paradigm for fast message authentication is based on the use of "almost xor universal hash functions". The design of these hash functions receives much attention and has resulted in some very fast ones, so that universal hash based MACs are the fastest MACs around. Let us begin by describing the tool, and then seeing how it can be used for message authentication.

6.13.1 Almost xor universal hash functions

Let H: $Keys(H) \times Dom(H) \to \{0,1\}^L$ be a family of functions. We think of them as hash functions because the domain $Dom(H)$ of any individual function H_K is typically large, being the message space of the desired message authentication scheme.

Fix any two points a_1, a_2 in the domain $Dom(H)$ of the family, the only restriction on them being that they are not allowed to be equal. Also fix a point b in the range $\{0,1\}^L$ of the family. With H fixed, we can associate to these three points a probability

$$\begin{aligned} \mathrm{UHColPr}_H(a_1, a_2, b) &= \Pr\left[K \stackrel{R}{\leftarrow} Keys(H) \,:\, H_K(a_1) \oplus H_K(a_2) = b \right] \\ &= \Pr\left[h \stackrel{R}{\leftarrow} H \,:\, h(a_1) \oplus h(a_2) = b \right] , \end{aligned}$$

the two expressions above being equal by definition. We are interested in keeping this probability low for all choices of a_1, a_2, b. The quality of H as an almost xor universal family, which we call the insecurity of H, is accordingly measured by the maximum value of this probability, the maximum being over the choices of a_1, a_2, b.

Definition 6.13.1 Let H: $Keys(H) \times Dom(H) \to \{0,1\}^L$ be a family of functions. Let

$$\mathbf{Adv}^{\mathrm{uh}}(H) = \max_{a_1,a_2,b} \left\{ \Pr\left[K \stackrel{R}{\leftarrow} Keys(H) \,:\, H_K(a_1) \oplus H_K(a_2) = b \right] \right\} ,$$

the maximum being over all *distinct* points $a_1, a_2 \in Dom(H)$ and all strings $b \in \{0,1\}^L$. ▮

The smaller the value of $\mathbf{Adv}^{\mathrm{uh}}(H)$, the better the quality of H as an almost xor-universal function. We say that H is a *xor-universal hash function* if $\mathbf{Adv}(H) = 2^{-L}$. (We will see later that this is the lowest possible value of the insecurity.)

The simplest example is the family of all functions.

Proposition 6.13.2 The family $R^{l,L}$ of all functions of l-bits to L-bits is xor-universal, meaning $\mathbf{Adv}^{\mathrm{uh}}(R^{l,L}) = 2^{-L}$.
Proof: With distinct $a_1, a_2 \in \{0,1\}^l$, and $b \in \{0,1\}^L$ fixed, we clearly have

$$\Pr\left[h \stackrel{R}{\leftarrow} R^{l,L} \ : \ h(a_1) \oplus h(a_2) = b \right] \quad = \quad 2^{-L}$$

because h is a random function. ▌

Another source of examples is polynomials over finite fields.

Example 6.13.3 Identify $\{0,1\}^l$ with $\mathrm{GF}(2^l)$, the finite field of 2^l elements. We fix an irreducible, degree l polynomial over GF(2) so as to be able to do arithmatic over the field. The hash function H we define takes as key a pair α, β of points in $\{0,1\}^l$ such that $\alpha \neq 0$. The domain is $\{0,1\}^l$ and the range is $\{0,1\}^L$ where $L \leq l$. We define the function by

$$H_{\alpha,\beta}(x) \quad = \quad [\alpha x + \beta]_{1\ldots L} \ .$$

That is, with key α, β and input $x \in \{0,1\}^l$, first compute, in the finite field, the value $\alpha x + \beta$. View this as an l-bit string, and output the first L bits of it.

Proposition 6.13.4 The family H: $\mathit{Keys}(H) \times \{0,1\}^l \to \{0,1\}^L$ defined above, where $L \leq l$ and $\mathit{Keys}(H)$ is the set of all pairs (a, b) of l-bit strings such that $a \neq 0$, is a xor-universal hash function.
Proof: We need to show that $\mathbf{Adv}(H) = 2^{-L}$. Accordingly fix $a_1, a_2 \in \{0,1\}^l$ such that $a_1 \neq a_2$, and fix $b \in \{0,1\}^L$. Fix any key for the function, meaning any $\alpha \neq 0$ and any β. Notice that $y = \alpha x + \beta$ iff $x = \alpha^{-1}(y - \beta)$. (The arithmatic here is over the finite field, and we are using the assumption that $\alpha \neq 0$.) This means that the map of $\mathrm{GF}(2^l)$ to $\mathrm{GF}(2^l)$ given by $x \mapsto \alpha x + \beta$ is a permutation. The proposition follows from this. ▌

It is useful to interpret the almost xor-universal measure in another, more dynamic way. Imagine that the choice of the points a_1, a_2, b is made by an adversary. This adversary C knows that H is the target family. It clunks along for a while and then outputs some distinct values $a_1, a_2 \in \mathit{Dom}(H)$, and a value $b \in \{0,1\}^L$. Now a key K is chosen at random, defining the function H_K: $\mathit{Dom}(H) \to \{0,1\}^L$, and we test whether or not $H_K(a_1) \oplus H_K(a_2) = b$. If so, the adversary C wins. We denote the probability that the adversary wins by $\mathbf{Succ}^{\mathrm{uh}}(H, C)$. We then claim that this probability is at most $\mathbf{Adv}^{\mathrm{uh}}(H)$.

The reason is that there is a single best strategy for the adversary, namely to choose points a_1, a_2, b which maximize the probability $\mathrm{UHColPr}_H(a_1, a_2, b)$ defined above. This should be relatively clear, at least for the case when the adversary is deterministic. But the claim is true even when the adversary is probabilistic, meaning that the triple of points it outputs can be different depending on its own coin tosses. (In such a case, the probability defining $\mathbf{Succ}^{\mathrm{uh}}(C)$ is taken over the choice of K and also the coin tosses of C.) We justify this claim in Proposition 6.13.5 below. We thus have two, equivalent ways of thinking about $\mathbf{Adv}^{\mathrm{uh}}(H)$, one more "static" and the other more "dynamic". Depending on the setting, we may benefit more from one view than another.

Before stating and proving Proposition 6.13.5, however, let us emphasize some features of this notion. A key feature of the game is that the steps must follow a particular order: *first* the adversary chooses points a_1, a_2, b, *then* K is chosen at random and the function H_K is defined. The adversary is *not* allowed to choose a_1, a_2, b as a function of K; it must first commit to them, and then there is some probability of its winning the game.

This notion differs from others we have considered in that there is no computational restriction on the adversary. Namely, it can run for as long as it likes in deciding how to choose a_1, a_2, b, and the security condition is true nonetheless. Thus, it is a purely information theoretic notion.

Here now is the promised bound.

Proposition 6.13.5 Let $H Keys(H) \times Dom(H) \to \{0,1\}^L$ be a family of functions and C a (possibly probabilistic) algorithm that outputs a triple a_1, a_2, b such that a_1, a_2 are distinct points in $Dom(H)$ and $b \in \{0,1\}^L$. Then

$$\mathbf{Succ}^{\mathrm{uh}}(H, C) \quad \leq \quad \mathbf{Adv}^{\mathrm{uh}}(H) \; .$$

Proof: Remember that to say C is probabilistic means that it has as an auxiliary input a sequence ρ of random bits of some length r, and uses them in its computation. Depending on the value of r, the output triple of C will change. We can denote by $a_1(\rho), a_2(\rho), b(\rho)$ the triple that C outputs when its coins are ρ. For any particular value of ρ it is clear from Definition 6.13.1 that

$$\begin{aligned}
&\Pr\left[\, K \stackrel{R}{\leftarrow} Keys(H) \;:\; H_K(a_1(\rho)) \oplus H_K(a_2(\rho)) = b(\rho) \,\right] \\
&\leq \max_{a_1, a_2, b} \{ \Pr\left[\, K \stackrel{R}{\leftarrow} Keys(H) \;:\; H_K(a_1) \oplus H_K(a_2) = b \,\right] \} \\
&= \mathbf{Adv}^{\mathrm{uh}}(H) \; .
\end{aligned}$$

Using this we get

$$\begin{aligned}
\mathbf{Succ}^{\mathrm{uh}}(H,C) &= \Pr\left[\,\rho \stackrel{R}{\leftarrow} \{0,1\}^r \,;\, K \stackrel{R}{\leftarrow} \mathit{Keys}(H) \;:\; H_K(a_1(\rho)) \oplus H_K(a_2(\rho)) = b(\rho)\,\right] \\
&= \sum_{\rho\in\{0,1\}^r} \Pr\left[\,K \stackrel{R}{\leftarrow} \mathit{Keys}(H) \;:\; H_K(a_1(\rho)) \oplus H_K(a_2(\rho)) = b(\rho)\,\right] \cdot 2^{-r} \\
&\leq \sum_{\rho\in\{0,1\}^r} \mathbf{Adv}^{\mathrm{uh}}(H) \cdot 2^{-r} \\
&= \mathbf{Adv}^{\mathrm{uh}}(H)\ .
\end{aligned}$$

The first equality is by definition of $\mathbf{Succ}^{\mathrm{uh}}(H,C)$. In the second line we used the fact that the coins of C are chosen at random from the set of all strings of length r. In the third line, we used the above observation. ▮

How low can $\mathbf{Adv}^{\mathrm{uh}}(H)$ go? We claim that the lowest possible value is 2^{-L}, the value achieved by a xor-universal family. The following justifies this claim.

Proposition 6.13.6 Let $H\mathit{Keys}(H) \times \mathit{Dom}(H) \to \{0,1\}^L$ be a family of functions. Then

$$\mathbf{Adv}^{\mathrm{uh}}(H) \;\geq\; 2^{-L}\ .$$

Proof: Fix two distinct points $a_1, a_2 \in \mathit{Dom}(H)$, and for any fixed key $K \in \mathit{Keys}(H)$ let

$$c(K) \;=\; \Pr\left[\,b \stackrel{R}{\leftarrow} \{0,1\}^L \;:\; H_K(a_1) \oplus H_K(a_2) = b\,\right]\ .$$

Then $c(K) = 2^{-L}$. Why? With K, a_1, a_2 all fixed, $H_K(a_1) \oplus H_K(a_2)$ is some fixed value, call it b'. The above is then just asking what is the probability that $b = b'$ if we pick b at random, and this of course is 2^{-L}.
Now consider the adversary C that picks b at random from $\{0,1\}^L$ and outputs the triple a_1, a_2, b. (Note this adversary is probabilistic, because of its random choice of b.) Then

$$\begin{aligned}
\mathbf{Succ}^{\mathrm{uh}}(H,C) &= \Pr\left[\,b \stackrel{R}{\leftarrow} \{0,1\}^L \,;\, K \stackrel{R}{\leftarrow} \mathit{Keys}(H) \;:\; H_K(a_1) \oplus H_K(a_2) = b\,\right] \\
&= \sum_{K\in \mathit{Keys}(H)} c(K) \cdot \Pr\left[\,K' \leftarrow \mathit{Keys}(H) \;:\; K' = K\,\right] \\
&= \sum_{K\in \mathit{Keys}(H)} 2^{-L} \cdot \Pr\left[\,K' \leftarrow \mathit{Keys}(H) \;:\; K' = K\,\right] \\
&= 2^{-L} \cdot 1\ .
\end{aligned}$$

Thus we have been able to present an adversary C such that $\mathbf{Succ}^{\mathrm{uh}}(H, C) = 2^{-L}$. From Proposition 6.13.5 it follows that $\mathbf{Adv}^{\mathrm{uh}}(H) \geq 2^{-L}$. ▮

6.13.2 The corresponding MACs

Let H: $Keys(H) \times \mathrm{MsgSp} \to \{0,1\}^L$ be a family of hash functions, and let F: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a PRF. We associate to them the *xor-universal hash based MACs*. There are two such MACs; one stateful (using counters) and deterministic, the other stateless and randomized. The key will be a pair of strings, K_1, K_2, where the first subkey is for H and the second is for F. (We call them the *hashing* and *masking* keys respectively.) In both cases, the basic paradigm is the same. The message is first hashed to a string x using H_{K_1}, and this value is then "encrypted" by XORing with $F_{K_2}(s)$ to yield a value τ, where s is some point chosen by the sender. The tag contains τ, but also s so as to permit verification. The difference in the two version is in how s is selected. In the counter version it is a counter, and in the randomized version a random number chosen anew with each application of the tagging algorithm.

Here now is the full description of the counter-based version of the scheme, $\mathcal{C}\text{-}\mathcal{UHM}^{H,F} = (\mathcal{K}, \mathrm{MAC}, \mathrm{VF})$–

Algorithm $\mathrm{MAC}_{K_1,K_2}(M)$	Algorithm $\mathrm{VF}_{K_1,K_2}(M, \sigma)$
$x \leftarrow H_{K_1}(M)$	Parse σ as (s, τ)
$\tau \leftarrow F_{K_2}(c) \oplus x$	$x' \leftarrow F_{K_2}(s) \oplus \tau$
$\sigma \leftarrow (c, \tau)$	$x \leftarrow H_{K_1}(M)$
$c \leftarrow c + 1$	If $x = x'$ then return 1 else return 0
Return σ	

The randomized version $\mathcal{R}\text{-}\mathcal{UHM}^{H,F} = (\mathcal{K}, \mathrm{MAC}, \mathrm{VF})$ is like this–

Algorithm $\mathrm{MAC}_{K_1,K_2}(M)$	Algorithm $\mathrm{VF}_{K_1,K_2}(M, \sigma)$
$x \leftarrow H_{K_1}(M)$	Parse σ as (s, τ)
$r \xleftarrow{R} \{0,1\}^l$	$x' \leftarrow F_{K_2}(s) \oplus \tau$
$\tau \leftarrow F_{K_2}(r) \oplus x$	$x \leftarrow H_{K_1}(M)$
$\sigma \leftarrow (r, \tau)$	If $x = x'$ then return 1 else return 0
Return σ	

Lemma 6.13.7 Let H: $Keys(H) \times \mathrm{MsgSp} \to \{0,1\}^L$ be a family of functions, and A an adversary attacking the message authentication scheme $\mathcal{C}\text{-}\mathcal{UHM}^{H,R^{l,L}}$. Then for any q, μ with $q < 2^l$ we have

$$\mathbf{Succ}^{\mathrm{ma}}(\mathcal{C}\text{-}\mathcal{UHM}^{H,R^{l,L}}, A) \quad \leq \quad \mathbf{Adv}^{\mathrm{uh}}(H) \; .$$

Proof of Lemma 6.13.7: The adversary A makes a sequence $M_1, \ldots, M_q$ of queries to its $\mathrm{MAC}_{K_1,K_2}(\cdot)$ oracle, and these are answered according to the above scheme. Pictorially:

$$\begin{array}{ccccc} M_1 & \Longrightarrow & \sigma_1 & = & (s_1, \tau_1) \\ M_2 & \Longrightarrow & \sigma_2 & = & (s_2, \tau_2) \\ \vdots & & \vdots & & \vdots \\ M_q & \Longrightarrow & \sigma_q & = & (s_q, \tau_q) \end{array}$$

Here $s_i = \langle i-1 \rangle$ is simply the (binary representation of the) counter value, and $\tau_i = f(s_i) \oplus h(M_i)$, where $h = H_{K_1}$ is the hash function instance in use, and $f = R_{K_2}^{l,L}$ is the random function specified by the second key. Following this chosen-message attack, A outputs a pair M, σ where $\sigma = (s, \tau)$. We may assume wlog that $M \notin \{M_1, \ldots, M_q\}$. We know that A will be considered successful if $\mathrm{VF}_{K_1,K_2}(M, \sigma) = 1$. We wish to upper bound the probability of this event.

Let New be the event that $s \notin \{s_1, \ldots, s_q\}$, and Old the complement event, namely that $s = s_i$ for some value of $i \in \{1, \ldots, q\}$. Let $\mathbf{P}[\cdot]$ denote the probability of event "$\cdot$" in the experiment $\mathrm{ForgeExp}(\mathcal{C}\text{-}\mathcal{UHM}^{H,R^{l,L}}, A)$. We consider

$$\begin{aligned} p_1 &= \Pr\left[\mathsf{Old} \mid \mathrm{VF}_{K_1,K_2}(M,\sigma) = 1\right] \\ p_2 &= \Pr\left[\mathsf{New} \mid \mathrm{VF}_{K_1,K_2}(M,\sigma) = 1\right] \\ q &= \mathbf{P}\left[\mathsf{New}\right] . \end{aligned}$$

We will use the following two claims.

Claim 1: $p_1 \leq \mathbf{Adv}^{\mathrm{uh}}(H)$.

Claim 2: $p_2 \leq 2^{-L}$.

We will prove these claims later. Let us first check that they yield the desired result:

$$\begin{aligned} \mathbf{Succ}^{\mathrm{ma}}(\mathcal{C}\text{-}\mathcal{UHM}^{H,R^{l,L}}, A) &= \mathbf{P}\left[\mathrm{VF}_{K_1,K_2}(M,\sigma) = 1\right] \\ &= p_1 q + p_2(1-q) \\ &\leq \mathbf{Adv}^{\mathrm{uh}}(H) \cdot q + 2^{-L} \cdot (1-q) \\ &\leq \mathbf{Adv}^{\mathrm{uh}}(H) \cdot q + \mathbf{Adv}^{\mathrm{uh}}(H) \cdot (1-q) \\ &\leq \mathbf{Adv}^{\mathrm{uh}}(H) . \end{aligned}$$

The first line is simply by definition of the success probability. The second line is obtained by conditioning. In the third line we used the claims. In the fourth line we used Proposition 6.13.6.

It remains to prove the claims. We begin with the second.

Proof of Claim 2: Since the queries of the adversary did not result in the function f being evaluted on the point s, the value $f(s)$ is uniformly distributed from the point of view of A. Or, remember the dynamic view of random functions; we can imagine that f gets specified only as it is queried. Since the tagging oracle (as invoked by A) has not applied f at s, we can imagine that the coins to determine $f(s)$ are tossed after the forgery is created. With that view it is clear that

$$p_2 = \mathbf{P}\left[f(s) \oplus h(M) = \tau\right] = 2^{-L} .$$

Note that here we did not use anything about the hash function; the claim is true due only to the randomness of f. □

Proof of Claim 2:

Adversary C
    Initialize counter c to 0
    For $i = 1, \ldots, q$ do
        $A \to M_i$
        $\tau_i \stackrel{R}{\leftarrow} \{0,1\}^L$; $s_i \leftarrow \langle c \rangle$; $\sigma_i \leftarrow (s_i, \tau_i)$
        $A \leftarrow \sigma_i$; $c \leftarrow c+1$
    $A \to M, \sigma$
    Parse σ as (s, τ)
    If $s \notin \{s_1, \ldots, s_q\}$ then FAIL
    Else let i be such that $s = s_i$
    Let $b \leftarrow \tau_i \oplus \tau$ and return M, M_i, b

We claim that $\mathbf{Succ}^{\mathrm{uh}}(H, C) = p_1$.

▮

Theorem 6.13.8 Let H: $\mathit{Keys}(H) \times \mathrm{MsgSp} \to \{0,1\}^L$ be a family of functions, and let F: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a PRF. Then for any t, q, μ we have

$$\mathbf{Adv}^{\mathrm{ma}}(\mathcal{C}\text{-}\mathcal{UHM}^{H,F}; t, q, \mu) \quad \leq \quad \mathbf{Adv}^{\mathrm{uh}}(H) + \mathbf{Adv}_F^{\mathrm{prf}}(t', q+1)$$

where $t' = t + O(\mu)$.

6.14 Problems

Problem 6.1 Consider the following variant of the CBC MAC, intended to allow one to MAC messages of arbitrary length. The construction uses a block cipher $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, 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 6.2 Consider the following variant of the CBC MAC, intended to allow one to MAC messages of arbitrary length. The construction uses a block cipher $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, which you should assume to be secure. The domain for the MAC is $(\{0,1\}^n)^+$. To MAC M under key K compute $\mathrm{CBC}_K(M) \oplus K'$. 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 6.3 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?

6.15 References and Related Work