G22.3033-003 Introduction to Cryptography November 15^{th}, 2001

Lecture 11

Lecturer: Yevgeniy Dodis *Scribe: Yevgeniy Tovshteyn*

This lecture we will be dedicated to the problem of *message authentication*. First, we define the problem of message authentication and show how it is different from the problem of encryption. Then we define the notion of message authentication schemes, and the subcase of those — message authentication codes (MAC). We will examine goals and capabilities of the intruder and will define the strongest security notion for MACs. Then we will see how MAC can be implemented using PRF (and the other way around). Then we will examine the problem with long messages and two approaches solving it. The second approach will lead us to the definition of a special family of *universal* hash functions, and the usage of such family to enhance the efficiency of PRF and MAC for long inputs. At the end we will see examples of universal hashing functions.

1 Introduction to Authentication

While Encryption is a topic of cryptography that deals with privacy, Authentication is a topic of cryptography that deals with trust. When a sender sends a message to a receiver, how does a receiver know if it comes from appropriate sender? What if the intruder tries to imitate a sender, or tampers with the real messages being sent, etc.? Is there a way for the recipient to be "sure" that the message indeed came from the supposed sender, and was not modified in the transit?

Similar to the encryption scenario, there are two approaches to solving this problem: the *secret-key* and the *public-key*. In this lecture we start with the secret-key setting. In this scenario, the sender and the recipient share a secret key s (not known to the attacker). This key helps the sender to "tag" the message, so that the recipient (only) can verify the validity of the "tag", but nobody can "forge" a valid tag. Thus, the goal is to establish authenticated communication between a dedicated sender/receiver pair.

To summarize, we are trying to design the following "s-functionality". Let's say sender is to send message m (in the clear, we are not solving encryption problem at the moment). So, it calls up his s-functionality that put the message m into the "envelope" T. Then it goes through open communication, accessible to intruders, so by the time it gets to the receiver it could change from T to some T'. Receiver receives this envelope T'. He calls up his s-functionality that validates whether envelope T' was stuffed by s-functionality on the sender's side. If it does, we can be "sure" that $T' = T$, receiver can extract m' from T' as a valid authenticated message, and in fact $m' = m$. Otherwise receiver rejects T' as invalid. Thus, the security of Authentication would imply inability by the intruder to produce a valid T' (in other words to send to the receiver something it would accept), not produced by the sender. The above is very close to a formal definition of a *message authentication scheme*.

1.1 Message Authentication Schemes

The above discussion leads to the following general definition (below we only define the syntax, and not the *security*). Below $\mathcal{M}$ is the corresponding message space, e.g. $\mathcal{M} = \{0,1\}^\ell$.

Definition 1 (Message Authentication Scheme) *Message Authentication Scheme is a triple (*Gen,Auth,Rec*) of* PPT *algorithms:*

a) The key generating algorithm Gen *outputs the shared secret key:* $s \leftarrow \mathsf{Gen}(1^k)$.

b) The message authentication algorithm Auth *is used to produce a value ("envelop")* $T \leftarrow \mathsf{Auth}_s(m)$*, for any* $m \in \mathcal{M}$*. It could be deterministic, as we will see.*

c) The deterministic message recovery algorithm Rec *recovers the message from the envelope* T*:* $\mathsf{Rec}_s(T) = \tilde{m} \in \mathcal{M} \cup \{\perp\}$*, where* $\perp$ *means that the message was improperly authenticated.*

The correctness property states that $\tilde{m} = m$*, i.e.* $\forall s, m, \quad \mathsf{Rec}_s(\mathsf{Auth}_s(m)) = m$.

Lets for now assume our message authentication schemes are stateless. Later we can examine stateful message authentication schemes as well.

Before moving any further (in particular, define the security of message authentication schemes), let us compare (secret-key) encryption and authentication.

1.2 Authentication vs. Encryption

Just to see how different the problem of Authentication is from the problem of Encryption, let's see that Encryption is not solving (and is *not intended to solve*) authenticity at all. Even the best encryption schema, one time pad for one message, was not addressing the issue. Lets say intruder E knows format of the message where the sender says "please credit 100 dollars to E", and say the one-time pad is used over ASCII alphabet. He can easily "flip" the 1 for a 9 for example, by simply flipping several bits of the encrypted message (in fact, the encryption will reveal the one-time pad, and E can encrypt anything he wants with it now). Similar attacks will apply to other encryption schemes we studied so far (e.g., counter scheme will have the same attack).

To summarize, encryption schemes are designed so that E cannot *understand* the contents of the encryption, but might allow E to tamper with it, or insert bogus messages. For example, in many encryption schemes *every* string is a legal encryption is some valid message. In this case, E can send any "garbage" string, and the recipient will decrypt it into a valid (albeit probably "useless") message. This should not be possible in a "secure" message authentication scheme.

Conversely, authentication be itself does not solve privacy. In fact, nothing prevents the envelop to have the message m *in the clear*.[1] To emphasize this point even further, for the rest of this lecture we will talk about a special class of message authentication schemes,

[1] Later we will study the problem of *authenticated encryption*, which simultaneously provides privacy and authenticity.

called *message authentication codes* (MACs), which *always* contain the message in the clear. As we will see, dealing with this special case is more intuitive, since it clearly illustrates our goal of message authentication.

2 Message Authentication Codes (MAC)

2.1 Definition

As we said, MAC is a special case of message authentication schemes. It implies sending message m in the clear along with a *tag* t. Namely, envelope $T = (m, t)$, and the message recovery only checks if the tag t is "valid". Thus, the receiver's goal is to validate message m by verifying whether she/he can trust that tag t is a valid tag for m. In case of successful validation receiver outputs "accept", otherwise "reject".

Definition 2 (Message Authentication Code) *Message Authentication Code,* MAC*, is a triple (*Gen,Tag,Ver*) of* PPT *algorithms:*

a) The key generating algorithm Gen *outputs the shared secret key:* $s \leftarrow \mathsf{Gen}(1^k)$.

b) The tagging algorithm Tag *is produces a tag* $t \leftarrow \mathsf{Tag}_s(m)$*, for any* $m \in \mathcal{M}$*. It could be deterministic, as we will see.*

c) The deterministic verification algorithm Ver *produces a value* $\mathsf{Ver}_s(m, t) \in \{accept, reject\}$ *indicating if* t *is a valid tag for* m.

The correctness property states that $\tilde{m} = m$*, i.e.* $\forall s, m, \quad \mathsf{Ver}_s(m, \mathsf{Tag}_s(m)) = accept$.

2.2 Security

The security as usual is measured by the probability of unauthorized PPT adversary $\mathcal{A}$ to succeed. But what is the goal of $\mathcal{A}$? The most ambitious goal is to recover secret key s. Similar to the encryption scenario, this is too ambitious (e.g., part of s can be never used, so it is not a big deal to prevent $\mathcal{A}$ from recovering s). A more reasonable goal is to come up with the message-tag pair (m, t) such that $\mathsf{Ver}_s(m, t) =$ "*accept*". Having that pair, $\mathcal{A}$ can masquerade as a valid sender of m by simply sending (m, t) to the receiver. This attack is called a *forgery*. There are two kinds of forgery: universal and existential. Universal forgery implies that $\mathcal{A}$ can come up with trusted tag t for any message m that $\mathcal{A}$ wants. Existential forgery implies that $\mathcal{A}$ can produce at least a single pair (m, t) that triggers Ver to "accept", even if the message m is "useless". Existential forgery is the least ambitious of these goals, so the strongest security definition would imply that no $\mathcal{A}$ can achieve even this, easiest of the goals. Moreover, in many applications such (strong) security is indeed needed.

Next, what can $\mathcal{A}$ do to try to achieve its goal? There are several options.

- The weakest attack mode to to give $\mathcal{A}$ no special capabilities. I.e., $\mathcal{A}$ cannot intrude into communication between sender and receiver and has no oracle access to any of MAC algorithms.

- A more reasonable assumption is that $\mathcal{A}$ has some access to communication between sender and receiver, and can try to analyze valid message-tag pairs (m, t) that it observes. Of course, once we allow this, we have to restrict $\mathcal{A}$ from outputting a "forgery" (m, t) for the message m that it already observed.[2] Taken to the extreme, we can allow $\mathcal{A}$ observe valid tags for *any* message m that $\mathcal{A}$ wants. Namely, we can give $\mathcal{A}$ *oracle access* to the tagging function $\mathsf{Tag}_s(\cdot)$. This is called (adaptive) *chosen message attack* (CMA).

- Finally, we can assume $\mathcal{A}$ has oracle access to the receiver, so that it can learn whether any message-tag pair (m, t) is valid. Alternatively, we may think that $\mathcal{A}$ keeps sending "fake" message/tag pairs, and wins the moment the recipient accepts such a pair.

Looking at the above, we now make the strongest definition of security, such that prevent $\mathcal{A}$ from its easiest goal, existential forgery, but lets $\mathcal{A}$ use both the tagging and the verification oracle.

Definition 3 (MAC's Security) *A* $\mathsf{MAC} = (\mathsf{Gen}, \mathsf{Tag}, \mathsf{Ver})$ *is "secure", i.e. existentially unforgeable against* CMA*, if* $\forall$ PPT $\mathcal{A}$*, who outputs a "forgery"* (m, t) *such that* m *has never been queried to oracle* $\mathsf{Tag}_s(\cdot)$*,*

$$\Pr(\mathsf{Ver}_s(m,t) = \mathit{accept} \mid s \leftarrow \mathsf{Gen}(1^k),\ (m,t) \leftarrow \mathcal{A}^{\mathsf{Tag}_s(\cdot),\mathsf{Ver}_s(\cdot)}(1^k)) \leq \mathsf{negl}(k)$$

For completeness, we also give the corresponding (more general definition) for any message authentication scheme (even though we will mainly work with MACs).

Definition 4 (Security of Message Authentication Scheme) *A message authentication scheme* $(\mathsf{Gen}, \mathsf{Auth}, \mathsf{Rec})$ *is "secure", i.e. existentially unforgeable against* CMA*, if* $\forall$ PPT $\mathcal{A}$*, who outputs a "forgery"* T *such that* $\mathsf{Rec}_s(T)$ *has never been queried to oracle* $\mathsf{Auth}_s(\cdot)$*,*

$$\Pr(\mathsf{Rec}_s(T) \neq \perp \mid s \leftarrow \mathsf{Gen}(1^k),\ T \leftarrow \mathcal{A}^{\mathsf{Auth}_s(\cdot),\mathsf{Rec}_s(\cdot)}(1^k)) \leq \mathsf{negl}(k)$$

Remark 5

a) Usually, Gen *just outputs a random string as a key. We will see examples of this later.*

b) When Tag *is deterministic, oracle access to* $\mathsf{Ver}_s(\cdot)$ *is not needed. Instead of calling* $\mathsf{Ver}_s(m', t')$ $\mathcal{A}$ *can call* $\mathsf{Tag}_s(m')$ *and compare its output with* t'*. In fact, from the asymptotic perspective, similar conclusion can be made about a general message authentication scheme: if* Auth *is deterministic, oracle access to* $\mathsf{Rec}(\cdot)$ *is not needed (why?), even though we loose a large polynomial factor in the security.*

[2] A marginally stronger definition forbids $\mathcal{A}$ to output a *pair* (m, t) that it already observed. In other words, $\mathcal{A}$ is allowed to forge a "new" tag t' for the message m whose tag $t \neq t'$ it already observed. However, this strengthening is not that crucial for several reasons. First, most MAC's are deterministic, in which case there is no difference between the two notions. Second, if m was legally sent and the receiver is "allowed" to accept m again, it will do it with the "old" tag t as well, so there is no value to even change t to $t' \neq t$. Finally, in many applications such "duplication" is anyway forbidden and is enforced by other means like time-stamps, etc.

2.3 MACs, Unpredictable Functions and PRFs

In this section we will build a secure MAC. In fact, we restrict our attention to *deterministic* (stateless) MAC's, whose keys ("seeds") are random strings of some length. Hence and without loss of generality, the verification algorithm $\mathsf{Ver}_s(m,t)$ simply checks if $t = \mathsf{Tag}_s(m)$, so we do not need to explicitly specify it. Moreover, $\mathsf{Tag}_s(m)$ is now a deterministic function indexed by the seed s. Thus, we can view such a MAC as a *family of functions* $\mathcal{F} = \{f_s\{0,1\}^\ell \to \{0,1\}^{|tag|}, s \leftarrow \mathsf{Gen}(1^{|s|})\}$, where $f_s = \mathsf{Tag}_s$. For simplicity, let us assume $|s| = k$. The chosen message attack (with no unnecessary oracle to Ver_s) corresponds to oracle access to $f_s(\cdot)$, for a random (unknown) s. The success corresponds to outputting a correct pair $(x, f_s(x))$, where x was not submitted to the $f_s(\cdot)$ oracle, i.e. *predicting* the value $f_s(x)$ for a "new" x. Not surprisingly, such, in some sense "canonical", MAC is called an *unpredictable function*. More formally, the family $\mathcal{F}$ is called an *unpredictable family* of for any PPT $\mathcal{A}$,

$$\Pr(f_s(x) = y \mid s \leftarrow \{0,1\}^k,\ (x,y) \leftarrow A^{f_s(\cdot)}(1^k)) \leq \mathsf{negl}(k)$$

where x is not allowed to be queried to the oracle $f_s(\cdot)$. Hence, to construct our first secure MAC it suffices to construct an unpredictable fumction family. Not suprisingly, a PRF family is an unpredictable family, provided its output length b is "non-trivial" (i.e., $\omega(\log k)$, so that $2^{-b} = \mathsf{negl}(k)$).

Theorem 6 (PRF $\Rightarrow$ MAC) *If $\mathcal{F}$ is a* PRF *family with non-trivial output length, then $\mathcal{F}$ is unpredictable, and thus defines a secure deterministic* MAC.

Proof: By the random function model, we can replace f_s with a truly random function f with output length b. Since b is "non-trivial" and x has to be "new", any adversary has a negligible probability to predict $f(x)$, completing the proof. □

Hence, "pseudorandomness implies unpredictability". But does the converse hold? It turns out *the answer is "yes"*, but the construction is quite tricky. Given unpredictable $\mathcal{F}$, we construct the following family $\mathcal{G}$ with output length 1 bit (from the properties of PRF's, such "short" output is not a problem and can be easily stretched), very similar to the Goldrecih-Levin construction.

Theorem 7 (Naor-Reingold, Goldreich-Levin) *Let $\mathcal{F} = \{f_s : \{0,1\}^\ell \to \{0,1\}^b \mid s \in \{0,1\}^k\}$ be an unpredictable family. Define $\mathcal{G} = \{g_{s,r} : \{0,1\}^\ell \to \{0,1\} \mid s \in \{0,1\}^k, r \in \{0,1\}^b\}$ as follows: $g_{s,r}(x) = f_s(x) \cdot r \bmod 2$, where $\alpha \cdot \beta$ denotes the inner product modulo 2, for $\alpha, \beta \in \{0,1\}^b$. Then $\mathcal{G}$ is a* PRF *family.*

Notice, we now know that OWF $\Rightarrow$ PRF $\Rightarrow$ MAC $\Rightarrow$ OWF, i.e. all these primitives are equivalent!

2.4 Dealing with Long Messages

In practice, concrete PRF have short fixed input length ℓ (for example, when implemented using a block cipher). One the other hand, in practice we want to be able to sign messages of length $L \gg \ell$ (e.g., one wants to sign a book using 128-bit block cipher modelled as a PRF). There are two approaches to deal with this problem.

1. Split m into n blocks $m_1 \dots m_n$ of length ℓ each (let's not worry about messages whose lenth is not a multiple of ℓ). Somehow separately tag each block using f_s. The simplest approach would be to just output $f_s(m_1) \circ \dots \circ f_s(m_n)$ as a tag. However, one can easily see that this is insecure (why?). Using more advanced suggestion, one can try $f_s(1 \circ m_1) \circ \dots \circ f_s(n \circ m_n)$. Again, there is a problem (which?). However, a few more "fixes" can make this approach work. However, the length of the tag now is quite large (proportional to $nb = Lb/\ell$, which could be large). Can we make it smaller? This is what the second approach below does.

2. The second approach involves a general efficient construction of a PRF family on L bit inputs from the one on $l \ll L$ bit inputs. Moreover, the output of the resulting PRF is as short as that of the original PRF (implying that tags as still very short!). Notice, this is more general than building a "long-input" MAC out of a "short-input" PRF: we actually build a "long-input" PRF!

   The idea is to design a special shrinking (hash) function $h : \{0,1\}^L \to \{0,1\}^\ell$ and use $f_s(h(m))$ as the tag. Notice, however, since there are $2^{L-\ell}$ times more possible messages than there are possible hash value, there are many pairs of messages (m_1, m_2) which "collide" under h: $h(m_1) = h(m_2)$. Unfortunately, the knowledge of any two such messages m_1 and m_2 allow the adversary to produce break the resulting "pseudo-PRF"; in fact, to produce a forgery of the resulting "pseudo-MAC". Indeed, since $f_s(h(m_1)) = f_s(h(m_2))$, $\mathcal{A}$ can ask for the tag $\alpha = f_s(h(m_1))$ of m_1 and output (m_2, α) as the forgery of a "new" message m_2. Luckily, this negative news only mean that *one public* h is not enough, similar to the fact that one public function cannot be a PRF: we need a *family of hash functions* h:

$$\mathcal{H} = \{h_t : \{0,1\}^L \to \{0,1\}^\ell, t \in \mathcal{K}\}$$

   where $\mathcal{K}$ is the correspinding key space for t. Now, we pick two independent keys for our resulting family $\mathcal{F}(\mathcal{H})$: the key s for f_s, and the key t for h_t, i.e. $\mathcal{F}(\mathcal{H}) = \{f_s(h_t(\cdot))\}$. The main question is:

   What properties of $\mathcal{H}$ make $\mathcal{F}(\mathcal{H})$ is a PRF *family when $\mathcal{F}$ is such?*

   We answer this question in the next section.

3 PRF and Universal Hashing

As the first observation, notice that for no two messages $m_1 \neq m_2$ can we have $\Pr_t(h_t(m_1) = h_t(m_2)) \geq \varepsilon$, where ε is non-negligible. Namely, no two elements are likely to collide. Indeed, otherwise the adversary who learns $\alpha = f_s(h_t(m_1))$ has probability ε that $f_s(h_t(m_2)) = \alpha$, which breaks the security of the PRF. It turns out that this necessary condition is also *sufficient*!

Definition 8 (ε-universal family of hash functions) *$\mathcal{H}$ is called* ε-universal *if*

$$\forall\ x, x' \in \{0,1\}^L,\ \ s.t.\ x \neq x' \implies \Pr_t(h_t(x) = h_t(x')) \leq \varepsilon$$

$\mathcal{H}$ is called (weakly) universal *if* $\varepsilon = \mathsf{negl}(|t|)$.

Theorem 9 *$\mathcal{F}(\mathcal{H}) = \{f_s(h_t(\cdot))\}$ is a* PRF *(and thus defines a* MAC*) if $\mathcal{F}$ is a* PRF *family and $\mathcal{H}$ is (weakly) universal.*

Proof: We have to show that $\forall$ PPT $\mathcal{A}$, $\mathcal{A}^{f_s(h_t(\cdot))} \approx \mathcal{A}^{Z(\cdot)}$, where Z is random function from $\{0,1\}^L$ to $\{0,1\}^b$. We will use two-step hybrid argument:

$$\mathcal{A}^{f_s(h_t(\cdot))} \approx \mathcal{A}^{R(h_t(\cdot))} \approx \mathcal{A}^{Z(\cdot)}$$

where R is random function from $\{0,1\}^\ell$ to $\{0,1\}^b$. The first step immediately follows from the definition of PRF. Hence, it suffices to show that $\mathcal{A}^{R(h_t(\cdot))} \approx \mathcal{A}^{Z(\cdot)}$.

Let $m_1 \ldots m_q$ be (wlog, distinct) queries $\mathcal{A}$ makes to its oracle (whatever it is). When given oracle access to $Z(\cdot)$, $\mathcal{A}$ gets q totally random and independent values. Intuitively, since R is a radnom function, as long as no two values $h_t(m_i)$ collide, $\mathcal{A}$ also gets q totally random and independent value. Thus, it suffices to show that the probability $h_t(m_) = h_t(m_j)$ for some i and j is $\mathsf{negl}(k)$. Since $\mathcal{H}$ is ε-unversal, where $\varepsilon = \mathsf{negl}(k)$, and there are at most $q^2 \leq \mathsf{poly}(k)$ pairs of i and j, the intuitive claim above follows, since $q^2\varepsilon = \mathsf{negl}(k)$.

The above intuition is almost formal, even though making it formal is slightly tricky. Let X be the event that during $\mathcal{A}$'s run with $R(h_t)$-oracle, a collision happened among $h_t(m_1) \ldots h_t(m_q)$. First, if X does not happen, the values $R(h_t(m_1)) \ldots R(h_t(m_q))$ are all random and independent from each other, exactly as the Z-oracle would return. Hence, the probability that $\mathcal{A}$ can tell apart the "Z-world" and the "$R(h_t)$-world" is at most the probability of X (defined in the "$R(h_t)$-world"). However, and this is the tricky point, once a collision happens in "$R(h_t)$-world", it does not matter how we answer the oracle queries of $\mathcal{A}$, since X has already happened!

Specifically, we can imagine the modified "$R(h_t)$-world", where *all* the queries of $\mathcal{A}$ are answered completely at random and independently from each other, irrespective of whether of not X happened. We claim that this does not alter the probability of X. Indeed, up to the point a collision happened (if it ever happens), all the queries are supposed to be answered at random, since R is a random function. What happens after X happens is irrelevant. But now we run $\mathcal{A}$ in a manner *independent from* t. Indeed, all the queries are simply answered at random. Thus, we can imagine that we *first* ran $\mathcal{A}$ to completion, and *only then* selected t and checked if X (i.e., a collision among $h_t(m_1) \ldots h_t(m_q)$) happened! But this means that $m_1 \ldots m_q$ are defined *before* (and independently from) t. By the ε-universality of $\mathcal{H}$, and since there are at most q^2 pairs of indices $i < j$, we get that $\Pr(X) \leq q^2\varepsilon = \mathsf{negl}(k)$, since q is polynomial and ε is negligible.

This argument completes the proof. □

We now give a variety of (weakly) universal families $\mathcal{H}$. As will will see, this primitive is quite easy to construct, both information-theoretically and computationally.

3.1 Information-Theoretic Examples

Matrix-Construction. Let A be a random $L \times \ell$ zero-one matrix. A can serve as a key to the family of hash functions as defined below:

$$h_A(x) = Ax$$

where $x \in \{0,1\}^L$ is viewed as a zero-one vector, and Ax denotes the matrix-vector product over $\mathbb{Z}_2$.

Let's examine the probability of collision. $\forall x_1 \neq x_2$, acollision means that $Ax_1 = Ax_2 \iff A(x_1 - x_2) = 0$ or $Az = 0$, where $z = x_1 - x_2 \neq \vec{0}$. Since A is random, and $z \neq \vec{0}$, each of ℓ entries of Az is a random bit, and hence the probability of collision is:

$$\Pr_A(Az = 0) = \frac{1}{2^\ell}$$

This is negligible, so this family of hashing functions is (weakly) universal. The biggest problem of this family is that key is quite enormously long (albeit still polynomially), even though the probability of collision is the best we can hope for: $1/2^\ell$ (in particular, it is independent of L). We would like the key to be $O(\ell)$, independent of the size of the message.

Polynomial Construction. Let F be a finite field of size roughly 2^l. $\mathbb{Z}_p$, where p is ℓ-bit long, is an example of such a finite field, but it is more convenient to use $GF[2^\ell]$ — a finite field of size 2^ℓ, since it takes exactly ℓ bits to represent an element in this field. Now, view $m = m_1, \ldots, m_n$ (i.e., $|m| = L \approx n\ell$), where each $m_i \in F$, as n coefficients of a degree $(n-1)$ polynomial over F (see below). Specifically, select a random point $x \in F$ as the key to a function h_x in the hash family, defined as

$$h_x(m_1, \ldots, m_n) = q_m(x) = \sum_{i=1}^{n} m_i \cdot x^{i-1}$$

where all the operations are done in F. Let's examine the probability of a collision between two distinct "polynomials" m and u. A collision here means

$$h_x(m) = h_x(v) \iff q_m(x) = q_u(x) \iff q_{m-u}(x) = 0 \iff \sum_{i=1}^{n} (m_i - u_i) \cdot x^{i-1} = 0$$

where at least one $m_i - u_i \neq 0$, i.e. $q_{m-u}(\cdot)$ is a *non-zero* polynomial of degree at most $(n-1)$. It is a well known fact that any polynomial of degree d can have at most d roots in F. Since the point (our key) $x \in F$ was chosen at random, the probability that x is one of these at most $(n-1)$ roots of $q_{m-u}(\cdot)$ is at most $\frac{n-1}{|F|}$. This is negligible since $|F| \approx 2^\ell$.

Notice, the key size is indeed only ℓ bits, independent of the message length $L = n\ell$ (instead, the error depends on L).

3.2 Computational Examples and the CBC-MAC

The next several examples use a PRF family $\mathcal{F} = \{f_s : \{0,1\}^\ell \to \{0,1\}^\ell\}$. Notice, we are not "cheating" here. We are using "short-input" PRF f_t to build "long-input" (weakly) universal $\mathcal{H} = \{h_t\}$. In principle, to build of our "long-input" PRF, we will have to combine it with another independently selected PRF f_s, via $f_s(h_t(\cdot))$. As we will see, however, a simple general trick allows us to avoid making s and t independent. Namely, sacrifize 1 bit in ℓ, and always apply $f_s(1, \cdot)$ when constructing the hash function $h_s(\cdot)$, and use $f_s(0, h_s(\cdot))$ on the outer layer. Using the "random function paradigm", $f_s(0, \cdot)$ and $f_s(1, \cdot)$ indeed look

like two independent random function. In fact, in specific case will not even have to do that (see below), even though it is a very inexpensive "loss" anyway. Below, we describe the hash function without the "trick" above.

To summarize, the advantage of using a PRF in bulding $\mathcal{H}$ is saving on the key size + making the construction possibly very efficient (since "practical" PRF's are very cheap). As a downside, the error probabilities will be worse, and will depend on the "computational closeness" of our PRF to a truly random function. Namely, to prove the universality of the hash function, we first assume that f_t is a truly random function (by the "random function paradigm"), and then prove the information-theoretic security as before.[3] In all the example below, we assume that: $m = m_1 \dots m_n$, where all $|m_i| = \ell'$, $L = \ell' n$, $\ell' \approx \ell$ (see below for details), *the number of blocks n is fixed*,[4], and t is a random key for our "base" PRF.

Using XOR Mode. Define

$$h_t(m_1 \dots m_n) = f_t(m_1, 1) \oplus f_t(m_2, 2) \oplus \dots \oplus f_t(m_n, n)$$

(so that the input to the PRF is slightly longer — $\ell = \ell' + \log n$ bits long). Assuming f is now a truly random function from ℓ to ℓ bits, and if $(u_1, \dots, u_n) \neq (m_1, \dots, m_n)$, say $m_i \neq u_i$ we get that

$$\begin{aligned} \Pr_f[f(m_1, 1) \oplus \dots \oplus f(m_n, n) = f(u_1, 1) \oplus \dots \oplus f(u_n, n)] &= \Pr_f[f(m_i, i) \oplus f(u_i, i) = \alpha] \\ &= \frac{1}{2^\ell} \end{aligned}$$

where α is some string independent[5] of $f(m_i, i) \oplus f(u_i, i)$, which in turn is random since $u_i \neq m_i$. As we indicated, to build a PRF out of it, we actually use

$$f_s(0,\ f_s(1, m_1, 1) \oplus \dots \oplus f_s(1, m_n, n))$$

Using CBC Mode (CBC-MAC). We can view this construction as simply applying the CBC mode of operation with IV being 0^ℓ (a string of ℓ zeros) and outputting the *last block only* (remember, we do not need to "decrypt", only to "tag"):

$$h_t(m_1 \dots m_n) = m_n \oplus f_t(m_{n-1} \oplus f_t(\dots f_t(m_2 \oplus f_t(m_1)) \dots))$$

The proof of universality of this $\mathcal{H}$ is a bit tricky, so we omit it. The main ideas are similar to what we have done earlier with CBC-encryption: intuitively, if $m \neq u$, say $m_i \neq u_i$, and f is a truly random function, the values $h_t(m)$ and $h_t(u)$ "diverge once and for all" w.h.p., starting at the i-th application of the f.

Theoretically, it seems like we need to apply the trick with prepending 0 and 1, when making it into a PRF, but a careful direct analysis shows that *this is not needed* (we omit the proof). Therefore, here is the final PRF, also known as the CBC-MAC (with $s = t$):

$$f_s(h_s(m_1 \dots m_n)) = f_s(m_n \oplus f_s(m_{n-1} \oplus f_s(\dots f_s(m_2 \oplus f_s(m_1)) \dots)))$$

[3] Of course, the construction will be inefficient with a truly random function, but this does not concern us: the efficient PRF construction is what we are using, only the *proof* uses a random function.

[4] It is possible to extend the construction for variable number of blocks, but we will not do it for simplicity.

[5] That is why we used the block number inside f.

CBC-MAC is extremely popular, and is used a lot in practice. Notice, we do not actually need $\mathcal{F}$ be a PRP family here (unlike for the encryption where we need to recover the message), but it should be a length-preserving PRF family.

4 XOR-MAC

We conclude the treatment of MAC's by mentioning another popular MAC construction using PRF's and the XOR operation. Namely, let $\mathcal{F}$ be the PRF family and $\mathcal{H}$ be a hash family from L to ℓ bits, whose properties will be given in a second. Rather than making the MAC output $f_s(h_t(m))$, we now let it output $(nonce, f_s(nonce) \oplus h_t(m))$. The verification of $(nonce, v)$ checks that $v = f_s(nonce) \oplus h_t(m)$. Here *nonce* is the value that w.h.p. never repeats again, like a random string, or a counter (notice the similarity with encryption). In particular, this method is typically either randomized (nonce is random), or stateful (nonce is a counter), unlike our previous method. Also, one has to either know or transmit the nonce. Finally, it is used only to make a MAC, and not a (more general) "long-input" PRF.

Still, what are the properties of $\mathcal{H}$ that make this method go through? As a simple attack, given a valid tag $(nonce, v)$ of m and a value a, the adversary can try to output a "forgery" $(nonce, v \oplus a)$ for some $m' \neq m$. It is easy to see that this will be successful if and only if $h_t(m) \oplus h_t(m') = a$. Since a, m, m' are arbitrary, at the very least we must have that for any $m \neq m'$, and any $a \in \{0,1\}^\ell$, we have

$$\Pr_t(h_t(m) \oplus h_t(m') = a) \leq \varepsilon$$

(where ε is negligible). Such families are called (weakly) *ε-xor-universal*. Notice, regular universality corresponds to $a = 0$ since $h_t(m) = h_t(m')$ iff $h_t(m) \oplus h_t(m') = 0$. Thus, a further disadvantage of this method is that it uses more restrictive classes of hash functions! However, the latter criticism is typically not a big deal, since many universal families are actually xor-universal. It turns out that xor-universality is sufficient:

Theorem 10 *$f_s(nonce) \oplus h_t(\cdot)$ defines a secure* MAC *whenever all the nonces are unique w.h.p., $\mathcal{F}$ is a* PRF *family and $\mathcal{H}$ is (weakly) xor-universal.*

The most used xor-universal family comes from the XOR mode of the previous section (and uses PRF to build h_t):

$$h_t(m_1 \dots m_n) = f_t(m_1, 1) \oplus f_t(m_2, 2) \oplus \cdots \oplus f_t(m_n, n)$$

It is easy to see that our proof from the previous section in fact showed that $\mathcal{H}$ is (weakly) xor-universal (check it). As in the previous section, we have to use the trick with prepending 0 and 1 to make the final MAC construction and use the same key:

$$\mathsf{Tag}_s(m) = (nonce,\ f_s(0, nonce) \oplus f_s(1, m_1, 1) \oplus \cdots \oplus f_s(1, m_n, n))$$

This is called the XOR-MAC. Naturally, it has a randomized or counter flavor depending on whether the nonce is random, or is a counter (in the later case the nonce need not be explicitly sent over).