CSCI-GA.3210-001 MATH-GA.2170-001	**Introduction to Cryptography**	March28, 2012

Lecture 10

Lecturer: Yevgeniy Dodis *Spring 2012*

Last time we defined several modes of operation for encryption. Today we prove their security, and then spend the rest of the lecture studying the problem of message authentication. We will see that the problem is very different from encryption, although some tool, such as PRFs, come up pretty handy!

1 Security of CBC, CFB and OFB Modes

The CPA-security of the CFB, OFB and CBC modes is shown by very similar technique. In fact, one "universal proof" suffices. We give a proof sketch here.

First, as usual, the block cipher (i.e., our PRP) g_s is replaced by a truly random function F (this was justified last time) by first replacing it by a random permutation, and then replacing the latter by a random function). Now, take any adversary Adv. Rather than choosing the whole random function F right away for the experiment with Adv, we select it "as needed" as we move along. Namely, we keep the table of $(input, output)$ pairs that were used so far. When a new input x to F arrives, we use the table if the input was used before, and otherwise add $(x, random)$ to the table, and use $random$ as the output of F. It is easy to see from the description of the CFB, OFB and CBC modes, that as long as we never perform the table look-up, i.e. no input is repeated twice during the run, the advantage of Adv is 0. Namely, all Adv sees is a bunch of random strings, independent from anything else.

Thus, it suffices to show that the probability an "input collision" happens is negligible. Let B_j denote the ("bad") event that the input collision happened within first j evaluations of F. We want to show that $\Pr(B_T) = \mathsf{negl}(k)$, where T is the total number of calls to F (roughly $T = nt$, where t is the number of encryption queries made by Adv and n is the number of blocks per message). We will show by induction on j that

$$\Pr(B_j) \le \frac{(j-1)T}{2^\ell}$$

Indeed, $\Pr(B_1) = 0$. Now, in order for B_j to happen, either there should be a collision among the first $(j-1)$ calls to F (i.e., B_{j-1} should happen), or the j-th input to F collided with one of the previous $(j-1)$ inputs. Formally,

$$\Pr(B_j) \le \Pr(B_{j-1}) + \Pr(B_j \mid \overline{B_{j-1}})$$

Notice, by inductive assumption the first probability is at most $(j-2)T/2^\ell$. On the other hand, from the facts that (1) $\overline{B_{j-1}}$ means that the previous $(j-1)$-st input was distinct from all the earlier inputs; (2) F is the random function, which combined with (1) gives that $(j-1)$-st output is truly random; and (3) the description of OFB/CFB/CBC modes

together with (1) and (2) implies that the j-th input is totally random, we conclude that the probability that *random* j-th input to F collides with the previous $(j-1)$ inputs is at most $\frac{j-1}{2^\ell} \leq \frac{T}{2^\ell}$. Combining these,

$$\Pr(B_j) \leq \frac{(j-2)T}{2^\ell} + \frac{T}{2^\ell} = \frac{(j-1)T}{2^\ell}$$

But now we see that $\Pr(B_T) \leq T^2/2^\ell$, which is indeed negligible since T is polynomial. Namely, w.h.p. no input collisions happen, and all the adversary simply sees is a bunch of truly random strings, giving him no way to predict the challenge b.

Remark 1 *Notice, how more subtle the proof becomes for these modes. In some sense, it is surprising that much simpler C-CTR and R-CTR modes are not used more often: unlike OFB,CFB and CBC modes, they are easily parallelizable for both encryption and decryption; unlike the CBC mode they work with a* PRF *and not only a* PRP*; finally, they give comparable (or even better in case of C-CTR) security guarantees.*

2 Secret-Key Encryption in Practice

In practice, specific efficient block and stream ciphers are used. These practical schemes usually follow some high level guidelines we saw in our formal study. However, the design of block and stream ciphers is usually viewed as "black art". We name a few such ciphers used in practice, and refer you to outside sources to learn more about those (i.e., see "Applied Cryptography" by Bruce Schneier).

- Stream ciphers: RC4, SEAL, WAKE, A5, Hughes XPD/KPD, and many-many others.
- Block Ciphers: DES, AES (Rijndael), Lucifer, FEAL, IDEA, RC2, RC6, Blowfish, Serpent, and many-many-many-many others.

We notice that block ciphers are used much more commonly. Most of them (with a notable exception of AES) use Feistel Network as part of their design, so Feistel transform is very useful. The current standard is AES — advanced encryption standard, which recently replaced DES — old data encryption standard. Also, using any stream cipher, or a block cipher with a good mode of operation, the size of the ciphertext is (almost) equal to the size of the plaintext, which is very efficient.

3 Public-Key Encryption in Practice

In general, secret-key (or symmetric) encryption is very efficient (even the theoretical ones we constructed using PRGs). On the contrary, public-key (or asymmetric) encryption is much less efficient, especially for large messages (aside from efficiency issues, the size of the ciphertext is usually much larger than the plaintext as well). On the other hand, a major drawback to symmetric key schemes is that they require that you share a secret with the intended recipient of your communication. On the contrary, public key encryption does *not* require you to share any secret knowledge with the recipient.

So can we have a relatively efficient public-key encryption, especially for long messages? The answer is "yes". And the technique is to combine the "slow" ordinary public-key encryption E' with the fast secret-key encryption Enc. This folklore technique is known as integration of public- and secret-key schemes. If auxiliary E' has public key PK and secret key SK, we define new public-key encryption E, which has the same public/secret keys, as follows:

$$E_{PK}(m) = (E'_{PK}(s), \mathsf{Enc}_s(m)), \qquad \text{where } s \leftarrow_R \{0,1\}^k$$
$$D_{SK}(c_1, c_2): \quad \text{get } s = D'_{SK}(c_1); \quad \text{output } m = \mathsf{Dec}_s(c_2)$$

That is, we encrypt a short, randomly chosen secret using s a public key scheme, and then transmit a stream or block cipher encryption of the message using the random secret. The recipient first decrypts the random secret, then uses that to decrypt the message. Provided that E'_{PK} and Enc are CPA-secure, the new scheme is also CPA-secure (in fact, one-message security suffices for the symmetric scheme Enc). The proof is left as an exercise.

In fact, this method can be generalized to any key encapsulation scheme capable of encapsulating a key s long enough to perform symmetric encryption of long messages. If $\langle \psi, s \rangle$ is the output of the key encapsulation algortihm, simply set the ciphertext to $c = \langle \psi, \mathsf{Enc}_s(m) \rangle$. When decrypting c, use key decapsulation to recover s from ψ, and then use s to recover m.

4 Introduction to Authentication

The rest of the 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 sub-case 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.

4.1 Motivation

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*.

4.2 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. $\forall s, m, \quad \mathsf{Rec}_s(\mathsf{Auth}_s(m)) = m$. $\diamondsuit$

Let us 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.

4.3 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, 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.

5 Message Authentication Codes (MAC)

5.1 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. $\forall s, m, \quad \mathsf{Ver}_s(m, \mathsf{Tag}_s(m)) = accept$. $\diamondsuit$

5.2 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, $\mathcal{A}$

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

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 is 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 $\mathsf{Tag}_s(\cdot)$,

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

$\diamondsuit$

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

[2] A slightly 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. This notion is called *strong unforgeability*. Luckily, most of our constructions will be strongly unforgeable. Moreover, in most (but not all; see below) settings this strengthening is not that crucial. First, most MAC's are deterministic and have "canonical" verification $t \stackrel{?}{=} \mathsf{Tag}_s(m)$, in which case there is no difference between the standard and strong unforgeability. 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.

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 $\mathsf{Auth}_s(\cdot)$,

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

$\diamondsuit$

Remark 2

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

b) When Tag *is deterministic and verification* Ver *is "canonical" (i.e. it simply checks* $\mathsf{Tag}_s(m) \stackrel{?}{=} t$*), — which include the deterministic PRF-based MACs studied in the next section, — oracle access to* $\mathsf{Ver}_s(\cdot)$ *is not needed.*[3] *Instead of calling* $\mathsf{Ver}_s(m', t')$*, $\mathcal{A}$ can call* $\mathsf{Tag}_s(m')$ *and compare its output with* t'*. And if $\mathcal{A}$ would like to check his candidate forgery* (m', t') *before outputting it, and at most q such checks are made by $\mathcal{A}$, we can simply let $\mathcal{A}$ "pretend" that all the checks are false, and output at random one of these q questions to* Tag *as its forgery. True, the success probability goes down by a factor of q, but since q is polynomial in k, it still remains non-negligible if it was non-negligible. More generally, that last argument extends to general message authentication scheme: if* Auth *is deterministic and* Rec *is "canonical" (i.e., it also checks* $T \stackrel{?}{=} \mathsf{Auth}_s(\mathsf{Rec}_s(T))$*), oracle access to* $\mathsf{Rec}(\cdot)$ *is not needed (why?), even though again we loose a large polynomial factor in the security.*

5.3 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* $\mathcal{F} = \{f_s\{0,1\}^\ell \rightarrow \{0,1\}^{|tag|}, s \leftarrow \mathsf{Gen}(1^{|s|})\}$, 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 function 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 $2^{-b} = \mathsf{negl}(k)$).

[3]The canonical verification is important. Otherwise, one can have artificial examples where carefully crafted verification queries on a message whose "canonical" tag is known to the attacker could reveal the entire secret key s.

Theorem 1 (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 (exactly 2^{-b} in the random function model) 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 2 (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.*

We remark that it is very important that the value r be kept secret (as implied by the notation above). Notice, this is different from the usual Goldreich-Levin setting, where the anverter for the one-way function learns r. Also notice that we now know that OWF $\Rightarrow$ PRF $\Rightarrow$ MAC $\Rightarrow$ OWF, i.e. all these primitives are equivalent!

5.4 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?). Turns out that a few more "fixes" can make this approach work. Unfortunately, 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 $\ell \ll L$ 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. There are two ways out of this problem:

- First way is to notice that it suffices to ensure that *it is computationally hard* to find such a collision (m_1, m_2) for h, even if the attacker knows the description of h. Such functions are called *collision-resistant* hash functions (CRHFs). It is easy to see that using CRHFs is easily enough to solve our problem. However, we will see that it is quite non-trivial to construct them.[4] We will come back to this approach soon.
- A simpler and more effective way to solve the problem is to use a *secret function* h not known to the attacker. Formally, 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 last question in the next section.

6 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 5 [ε-universal family of hash functions] $\mathcal{H}$ is called *ε-universal* if

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

If $\varepsilon = 2^{-\ell}$ (which turns out the smallest it can get when $\ell < L$), then $\mathcal{H}$ is simply called (perfectly) *universal*. In assymptotic terms, $\mathcal{H}$ is called *almost universal (AU)* if $\varepsilon = \mathsf{negl}(|t|)$. ♢

Theorem 3 *$\mathcal{F}(\mathcal{H}) = \{f_s(h_t(\cdot))\}$ is a* PRF *(and thus defines a* MAC*) if $\mathcal{F}$ is a* PRF *family and $\mathcal{H}$ is almost universal.*

[4] In particular, it turns out one is unlikely to construct them even from the strongest general assumption that we studied so far — existence of TDPs. However, we will later construct them from specific number-theoretic assumptions, such as discrete log.

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 $m_1 \dots m_q$ 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 values. 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 $h_t(m_1) \dots h_t(m_q)$. First, if X does not happen, the values $R(h_t(m_1)) \dots R(h_t(m_q))$ 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 $h_t(m_1) \dots h_t(m_q)$) happened! But this means that $m_1 \dots m_q$ 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. $\square$