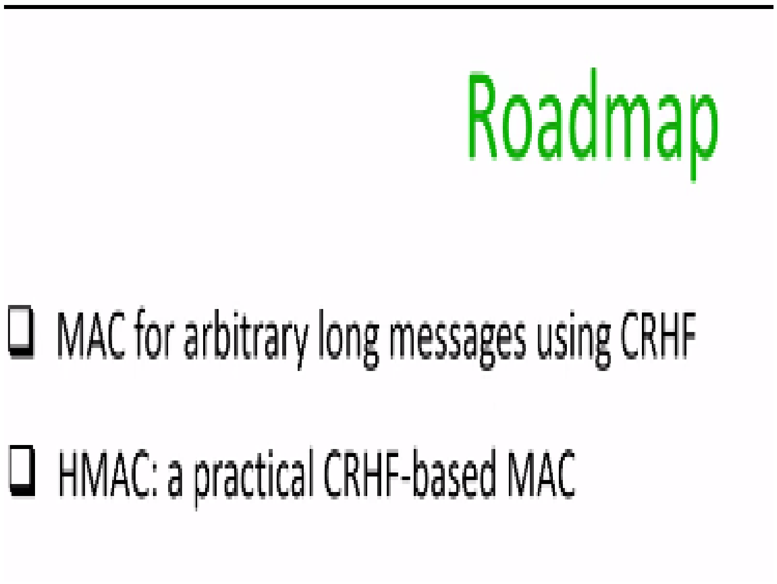
Foundations of Cryptography

Prof. Dr. Ashish Choudhury
(Former) Infosys Foundation Career Development Chair Professor
International Institute of Information Technology – Bangalore

Lecture – 29
Message Authentication Using Hash Functions

(Refer Slide Time: 00:33)



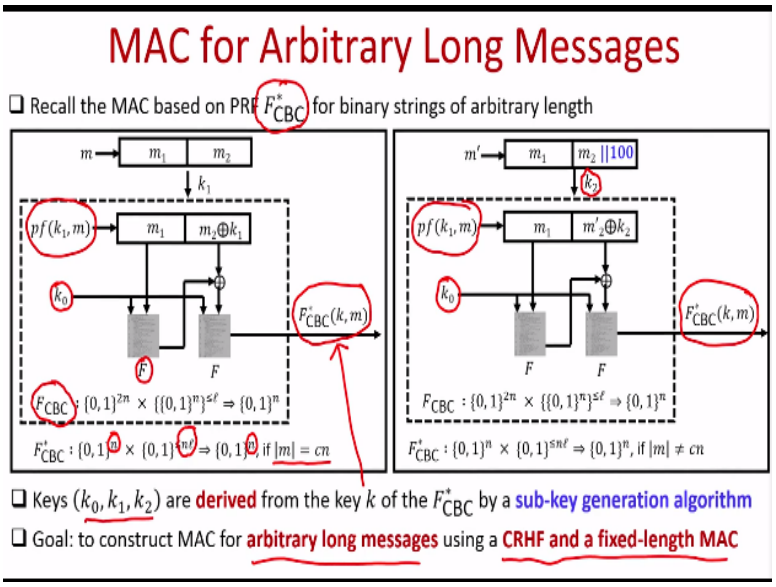
Roadmap

- MAC for arbitrary long messages using CRHF
- HMAC: a practical CRHF-based MAC

Hello everyone, welcome to this lecture. Just to recall in the last lecture, we have introduced a new cryptographic primitive, namely cryptographic hash functions and we also discussed rigorously the one of the important security properties that we require from cryptographic hash functions namely that of collision resistance. We also saw how to construct cryptographic hash functions using Merkle-Damgard paradigm. So the plan for this lecture is as follows.

We will see how to use collision-resistant hash function to design message authentication codes for arbitrary long messages and we will see an instantiation of this paradigm, practical instantiation namely HMAC, which is a widely used message authentication code used in practice.

(Refer Slide Time: 01:10)


m
m_1
m_2
k_1
$pf(k_1, m)$
m_1
$m_2 \oplus k_1$
k_0
F
F
$F^*_{CBC}(k, m)$
$F_{CBC} : \{0,1\}^{2n} \times \{\{0,1\}^n\}^{\leq \ell} \Rightarrow \{0,1\}^n$
$F^*_{CBC} : \{0,1\}^n \times \{0,1\}^{n\ell} \Rightarrow \{0,1\}^n$, if $|m| = cn$
m'
m_1
$m_2 || 100$
k_2
$pf(k_1, m)$
m_1
$m'_2 \oplus k_2$
k_0
F
F
$F^*_{CBC}(k, m)$
$F_{CBC} : \{0,1\}^{2n} \times \{\{0,1\}^n\}^{\leq \ell} \Rightarrow \{0,1\}^n$
$F^*_{CBC} : \{0,1\}^n \times \{0,1\}^{\leq n\ell} \Rightarrow \{0,1\}^n$, if $|m| \neq cn$

MAC for Arbitrary Long Messages

- Recall the MAC based on PRF F^*_{CBC} for binary strings of arbitrary length
- Keys (k_0, k_1, k_2) are **derived** from the key k of the F^*_{CBC} by a **sub-key generation algorithm**
- Goal: to construct MAC for **arbitrary long messages** using a **CRHF and a fixed-length MAC**

So just to recall, we have already seen how to construct message authentication codes for long messages. So recall the construction of our PRF (F^*_{CBC}) F* using the CBC mode of PRF for fixed-length inputs and this PRF F* based on CBC mode takes inputs as binary strings of arbitrary length and gives you a tag or an output of fixed size. So just to recall how exactly this construction F* looks like. It takes an input which is a key of size n bits and the actual input on which the PRF needs to be evaluated which can be a bit string of length up to n times *l* bits and it gives you a fixed output of size n bits.

Basically, depending upon whether the underlying message on which you want to evaluate this PRF F* is a multiple of n or not, we are actually having one of the 2 possible cases. So the first case is when the number of blocks in your message on which you want to evaluate your PRF F*, it is already a multiple of n, in that case what we do is, we first apply a prefix-free randomized encoding operated by a key k_1.

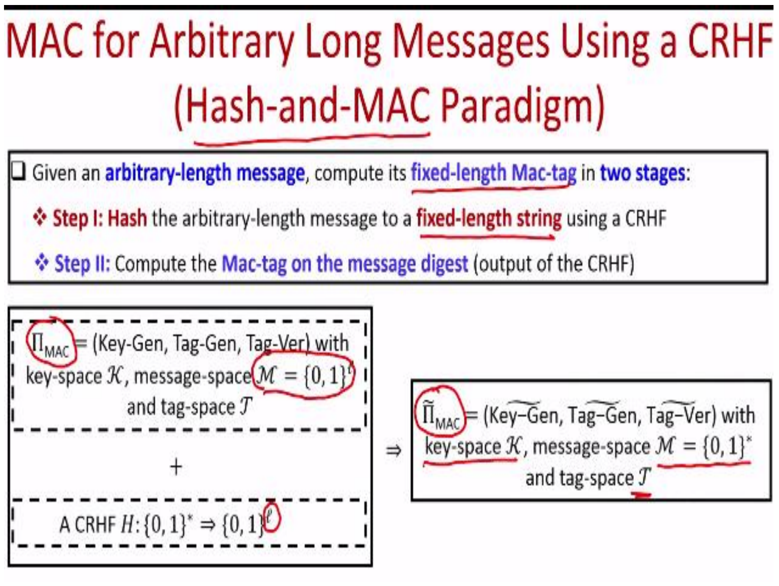
Once we have the prefix-free encoding of your message on which you want to evaluate your PRF, what we do is basically we evaluate the CBC mode of PRF which is block wise secure and where the number of blocks in the encoded input m is already a multiple of the block size of your fixed size PRF F and this we operate using another key k_0 and the overall output is taken as the output of your PRF F* for the input m. This is for the case when the size of your message is c times some n for a constant c.

Whereas if the size of the message is not a multiple of n, then what we do is we do a padding, before computing the encoding of your message. So, the padding is a deterministic padding, where we divide your message into blocks of n bits, n bit, n bits and the last block basically consists of the padded bits which is 1 followed by the required number of zeros and then we compute a prefix-free encoding of this padded message m' under the key k_2. Once we have the prefix-free encoding of the message, we now operate the block wise secure PRF F_{CBC} under the key k_0.

That is taken as the overall outcome of your PRF F* for the input m. So, here the key is k_0, k_1, k_2 and they are derived from the master key k with which you are going to operate your PRF F* by a sub key generation algorithm which could be publicly known and depending upon whether your message size is some constant times n or not, the prefix-free encoding is operated with either key k_1 or with the key k_2. So that is an indication to the receiving side whether the message which has been evaluated by this PRF F* its size is already some constant times n or not.

Now the goal here is we want to construct a message authentication codes again for arbitrarily long messages, but now using a collision-resistant hash function and a fixed-length MAC and we hope that indeed if we have one such construction, then we can completely get rid of the large number of PRF invocations which are used in this construction F*, right.

(Refer Slide Time: 04:49)



MAC for Arbitrary Long Messages Using a CRHF (Hash-and-MAC Paradigm)

- Given an **arbitrary-length message**, compute its **fixed-length Mac-tag** in **two stages**:
  - **Step I: Hash** the arbitrary-length message to a **fixed-length string** using a CRHF
  - **Step II:** Compute the **Mac-tag on the message digest** (output of the CRHF)

Π_{MAC} = (Key-Gen, Tag-Gen, Tag-Ver) with key-space $\mathcal{K}$, message-space $\mathcal{M} = \{0,1\}^{\ell}$ and tag-space $\mathcal{T}$

+

A CRHF $H: \{0,1\}^* \Rightarrow \{0,1\}^{\ell}$

$\Rightarrow$

$\widetilde{\Pi}_{MAC}$ = ($\widetilde{\text{Key-Gen}}$, $\widetilde{\text{Tag-Gen}}$, $\widetilde{\text{Tag-Ver}}$) with key-space $\mathcal{K}$, message-space $\mathcal{M} = \{0,1\}^*$ and tag-space $\mathcal{T}$

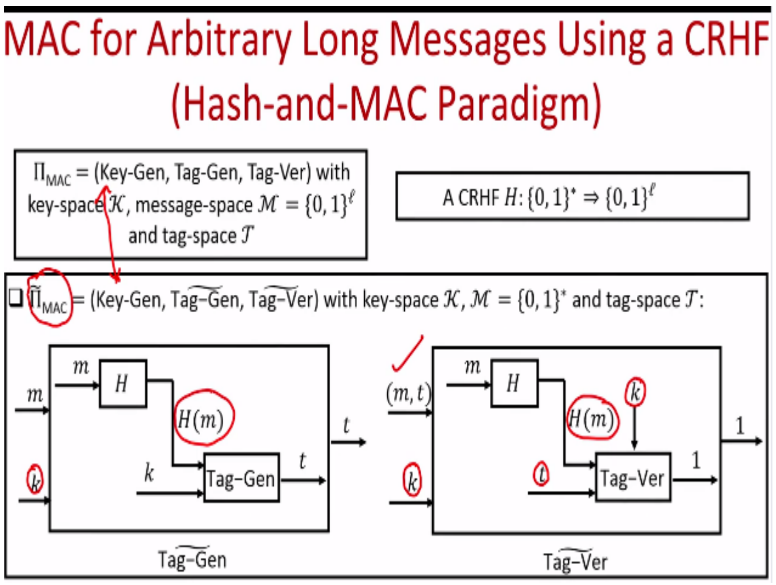
Let us see how we can design a message authentication code for arbitrary long messages using a collision-resistant hash function and this paradigm is popularly called as Hash-and-MAC paradigm. So as the name suggests what we do is if you are given an arbitrary length message on which you want to compute the authentication tag, then the tag is computed in 2 stages. Remember that our tag should be of fixed length, its length should not depend upon the message which you want to authenticate.

So this fixed-length tag is computed in 2 stages. In stage 1, we first hash the arbitrary message on which you want to compute the tag to a fixed-length string and this is done using a collision-resistant hash function and now once you have the hash of the message or the digest of the message which you have obtained in step 1, what we do is we compute the tag on the output which we have obtained on the previous step by using some fixed-length message authentication code and that is why the name Hash-and-MAC paradigm.

We hash first the input and then we compute tag on the output of the hash of the message. So block wise what you are given here is you are given a message authentication code which I denote by say Π_{MAC} and which is a secured MAC, which has a key generation algorithm, tag generation algorithm, and tag verification algorithm and it can authenticate fixed-size messages, namely messages of size say l bits and it has its own message-space, key-space and tag-space.

We are also given say a secure or collision-resistant hash function say H taking arbitrary length inputs and giving you fixed set size outputs, the outputs of size l bits. Then what we are going to do in this Hash-and-MAC paradigm is, we are going to combine these 2 primitives and obtain a secure MAC, which I denote as $\widetilde{\Pi}$ and it will have its key generation algorithm, tag generation algorithm, and tag verification algorithm.

The key space of the MAC that we obtained by composing the fixed length MAC and the collision-resistant hash function will be the same as the key space of the fixed-length MAC, whereas the message space will be strings or binary strings of arbitrary length and the tag space will be the same as the tag space of your fixed-length MAC Π_{MAC}.
(Refer Slide Time: 07:02)


MAC for Arbitrary Long Messages Using a CRHF (Hash-and-MAC Paradigm)

Π_{MAC} = (Key-Gen, Tag-Gen, Tag-Ver) with key-space $\mathcal{K}$, message-space $\mathcal{M} = \{0,1\}^{\ell}$ and tag-space $\mathcal{T}$

A CRHF $H: \{0,1\}^* \Rightarrow \{0,1\}^{\ell}$

❑ $\widetilde{\Pi}_{MAC}$ = (Key-Gen, $\widetilde{\text{Tag-Gen}}$, $\widetilde{\text{Tag-Ver}}$) with key-space $\mathcal{K}$, $\mathcal{M} = \{0,1\}^*$ and tag-space $\mathcal{T}$:

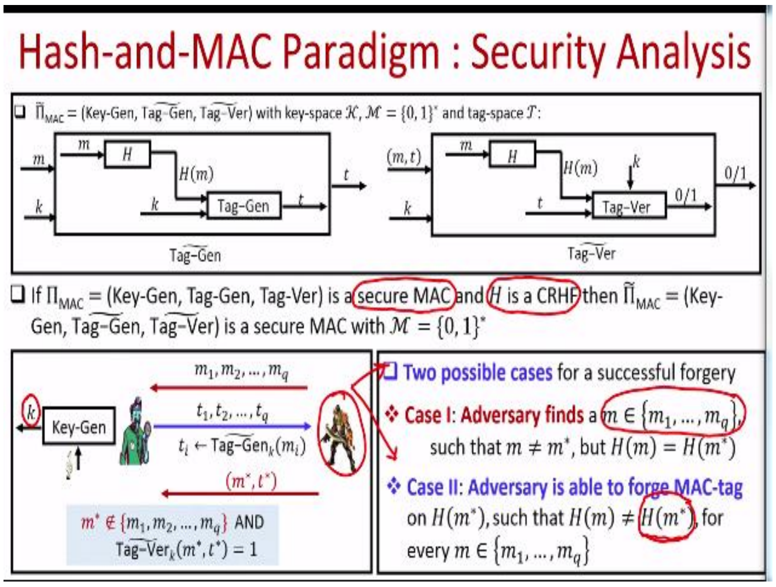
So, here is how we compose the fixed-length MAC along with the collision-resistant hash function to obtain the message authentication code $\widetilde{\Pi}$. So, the tag generation algorithm is as follows. It takes a message which could be of any length, it is an arbitrary bit string and a key k randomly generated by the key generation algorithm. So just to stress here the key generation algorithm of the composed MAC is the same as the key generation algorithm of your base MAC. So, the base MAC outputs a uniformly random key of some fixed size, then so is the key generation algorithm of this composed MAC.

So key k is one of those keys and m is the message on which we want to compute the tag. So, what we do internally inside this tag generation algorithm is we apply the collision-resistant hash function on your input m and once we have the hash of the message which is say of size l bits, we invoke the tag generation algorithm of our base MAC under the key k of the composed MAC and the resultant output is considered as the tag generated by the composed MAC.

The tag verification algorithm is similarly done here. So imagine you are given an arbitrary length input m along with the corresponding tag and you want to verify it with respect to a key k. So what we do is we re-compute the tag on the message part of the input. So that we do by performing the hash or computing the hash of the message and then once we have the hash of the message, we again perform or we compute the tag verification algorithm with respect to the input H(m) and the tag component of the input that you have received for this tag verification algorithm.

What we do is we invoke the tag verification algorithm of our base MAC with respect to the key k, and if the tag verification fails, that means this message,tag pair should be rejected whereas if the tag verification of the base MAC is successful, then that means we should accept the message, tag that we have obtained for the composed MAC. So that is how the Hash-and-MAC paradigm works.

(Refer Slide Time: 09:19)



Hash-and-MAC Paradigm : Security Analysis

- $\tilde{\Pi}_{MAC}$ = (Key-Gen, $\widetilde{\text{Tag-Gen}}$, $\widetilde{\text{Tag-Ver}}$) with key-space $\mathcal{K}$, $\mathcal{M} = \{0,1\}^*$ and tag-space $\mathcal{T}$:

m
k
H
$H(m)$
Tag-Gen
t
$\widetilde{\text{Tag-Gen}}$
(m,t)
Tag-Ver
0/1
$\widetilde{\text{Tag-Ver}}$

- If Π_{MAC} = (Key-Gen, Tag-Gen, Tag-Ver) is a secure MAC and H is a CRHF then $\tilde{\Pi}_{MAC}$ = (Key-Gen, $\widetilde{\text{Tag-Gen}}$, $\widetilde{\text{Tag-Ver}}$) is a secure MAC with $\mathcal{M} = \{0,1\}^*$

k
Key-Gen
$m_1, m_2, \ldots, m_q$
$t_1, t_2, \ldots, t_q$
$t_i \leftarrow \widetilde{\text{Tag-Gen}}_k(m_i)$
(m^*, t^*)

$m^* \notin \{m_1, m_2, \ldots, m_q\}$ AND $\widetilde{\text{Tag-Ver}}_k(m^*, t^*) = 1$

- Two possible cases for a successful forgery
  - Case I: Adversary finds a $m \in \{m_1, \ldots, m_q\}$, such that $m \neq m^*$, but $H(m) = H(m^*)$
  - Case II: Adversary is able to forge MAC-tag on $H(m^*)$, such that $H(m) \neq H(m^*)$, for every $m \in \{m_1, \ldots, m_q\}$

So, now we want to analyze whether indeed this Hash-and-MAC paradigm is going to give us a secure message authentication code for authenticating arbitrary length inputs. So what we want to prove here is that if the component-wise, the fixed-length MAC is a secure MAC, say CMA secure or strong CMA secure and if the H component that is given to us is also a collision-resistant hash function, then the composed MAC that we have obtained is indeed a secure MAC which can authenticate arbitrary length inputs.

So for simplicity what we are going to prove is we are going to prove the CMA security of the composed MAC assuming that the base MAC is also a deterministic, but that need not be the case if your base MAC is a randomized MAC, then the overall MAC that we are obtaining is also a randomized MAC in which case we should go for strong CMA security, but just to keep our argument simple, we assume that the base MAC is a deterministic MAC and as hence the composed MAC is also a deterministic MAC and hence we are going to prove the CMA security.

So just to recall how exactly the CMA game will be played in against this composed MAC. So as per the rules of the CMA game, adversary is going to ask for tags on several messages of its choice adaptively, namely polynomial number of messages and to respond to adversary’s queries, the challenger of the experiment runs the key generation algorithm obtains a uniformly random key and it computes the tag on all the messages for which the adversary has asked for the tag.

Those tags are returned back to the adversary as per the tag generation algorithm of the composed MAC scheme, and finally, the adversary outputs a forgery, namely a message, tag pair and we say that adversary has won the game or the output of the experiment is 1, if this message m* which has been submitted by the adversary is different from all the messages for which the adversary has asked for the tags and the tag verification of the composed MAC on the m*, t* gives you the output 1.

So our goal is to show that if the component-wise the base MAC is secure and the underlying hash function is collision resistant, the probability that any poly-time adversary could win this experiment is upper bounded by some negligible probability. So the idea behind the proof is as follows. So if at all there exists an adversary who can come up with a forgery m*, t* in polynomial time, then there could be 2 possible cases: the first case could be that the adversary who has come up with the forgery m*, t* for him it so happens that there exists at least one of the messages for which he has asked for the tag such that, which we denote say by m, such that the forged message m* even though that is different from that particular special message m, it so happened that the message m, m* constitutes a collision for your underlying hash function.

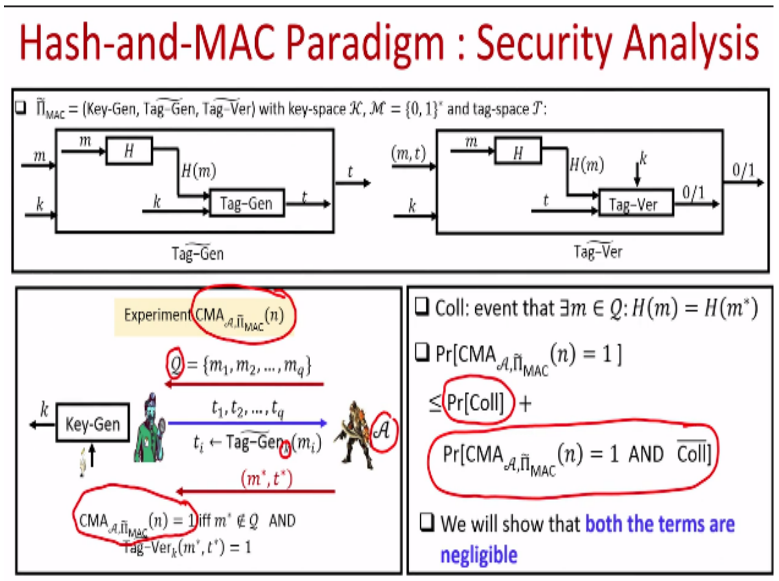
If that happens, then in that case it is easy for the adversary to come up with a tag on the message m* because adversary has already queried for the tag on the message m and say it has obtained a tag t and if the hash of the message m and the hash of the message m* are same that means the tag for the message m* as per the composed message authentication code will also be t. And hence by knowing m, t, it will be easy for an adversary to come up with the tag on the message m* and that will be the forgery for the adversary.

But this internally means that adversary needs to find out a collision for the underlying hash function in polynomial amount of time, which contradicts our assumption that the underlying hash function is a collision-resistant hash function, right. So this is one of the cases by which one of the possibilities under which the adversary could come up with the forgery m*, t*. The second case could be that it may so happen that the message m*, its hash is different from hash of all the messages for which the adversary has asked for or queried for the tag.

But even if the hash of the message m and the hash of the message m* are different, it may so happen that the adversary is able to actually forge a tag on the fixed-length input, namely hash of the message m*. If that is the case, then again whatever forgery m*, t* adversary has submitted, it will be considered as a valid forgery, but for this case 2 to be true, what adversary basically has to do, it has to basically forge a tag on the message H(m*).

Where H(m*) is different from all the H(m)'s for which the adversary has asked for the tag as for the composed scheme, but this will contradict our assumption that the base message authentication code that we are assuming is CMA secure. So these are the two possible cases under which an arbitrary adversary against a composed scheme could come up with a forgery. Now what we are going to informally establish is that both these case 1 and case 2 are going to be successful for any poly-time adversary only with some negligible probability, right.

(Refer Slide Time: 14:32)



Hash-and-MAC Paradigm : Security Analysis

- $\tilde{\Pi}_{MAC}$ = (Key-Gen, $\widetilde{\text{Tag-Gen}}$, $\widetilde{\text{Tag-Ver}}$) with key-space $\mathcal{K}$, $\mathcal{M} = \{0,1\}^*$ and tag-space $\mathcal{T}$:

m
k
H
$H(m)$
Tag-Gen
t
$\widetilde{\text{Tag-Gen}}$
(m, t)
Tag-Ver
0/1
$\widetilde{\text{Tag-Ver}}$
Key-Gen
$\mathcal{A}$

Experiment $\text{CMA}_{\mathcal{A},\tilde{\Pi}_{MAC}}(n)$

$Q = \{m_1, m_2, \ldots, m_q\}$

$t_1, t_2, \ldots, t_q$

$t_i \leftarrow \widetilde{\text{Tag-Gen}}_k(m_i)$

(m^*, t^*)

$\text{CMA}_{\mathcal{A},\tilde{\Pi}_{MAC}}(n) = 1$ iff $m^* \notin Q$ AND $\widetilde{\text{Tag-Ver}}_k(m^*, t^*) = 1$

- Coll: event that $\exists m \in Q: H(m) = H(m^*)$
- $\Pr[\text{CMA}_{\mathcal{A},\tilde{\Pi}_{MAC}}(n) = 1]$

  $\leq \Pr[\text{Coll}] +$

  $\Pr[\text{CMA}_{\mathcal{A},\tilde{\Pi}_{MAC}}(n) = 1 \text{ AND } \overline{\text{Coll}}]$
- We will show that both the terms are negligible

So formally, here is your CMA game against the composed message authentication code for arbitrary length messages against an arbitrary poly-time adversary. So it asks for the tags for certain number of messages and the set of those messages I am denoting by this *Q* set and in response, it obtains the tag on those messages as per the unknown uniformly random key picked by the key generation algorithm and as for the syntax of this composed scheme, each of this tag t_i is basically obtained by first hashing the message m_i and then computing a fixed length tag under the unknown key k on the hash of m_i's.

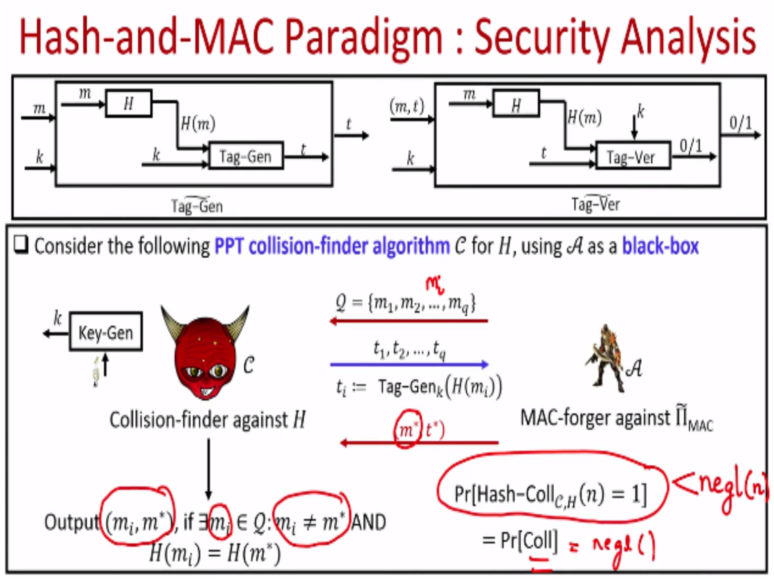
After obtaining the tags for this Q messages, adversary submits a forgery and our goal is to analyze what is the success probability of adversary winning this experiment which we denote that CMA experiment outputting 1 and just as per the rules of the game the CMA experiment outputs 1 if and only if the forged message m* is different from all the messages in the Q set and a tag verification with respect to the unknown key for this message m* with respect to the tag t* is successful.

So to analyze the success probability of the CMA experiment, let me call an events Coll which basically denotes the event that there exist at least one message in the set of messages for which the adversary has asked for the tag such that the hash of that message m is the same as the hash of the forged message m*, and now it is easy to see that the success probability of the adversary against the CMA experiment or the probability that the output of the CMA experiment is 1 can be splitted into 2 disjoint event, namely conditioned on the event whether the event collision happens or not, right.

So the overall probability that the output of the CMA experiment is 1, can be written as the probability that the output of the CMA experiment is 1 and the event collision occurs plus the probability that the output of the CMA experiment is 1 and the event collision does not occur, right. So, this follows from the basic rules of probability and now what I can do is this first term here $\Pr[\mathrm{CMA}_{\mathcal{A},\widetilde{\Pi}_{\mathrm{MAC}}}(n) = 1 \text{ AND Coll}]$ can always be upper bounded by the probability of the event collision to happen and the remaining probability I am retaining as it is, right.

So just I am doing some substitutions here. So, I can always upper bound the probability that output of the experiment CMA is 1 by this entity and now what we are going to show is that each of this expressions on your right hand side, namely the probability of the event collision to happen and the probability that the output of the CMA experiment is 1 and the event collision does not happen both are negligible functions of your underlying security parameter.

(Refer Slide Time: 17:35)


Hash-and-MAC Paradigm : Security Analysis
m
H
$H(m)$
k
Tag-Gen
t
$\widetilde{\text{Tag-Gen}}$
(m,t)
Tag-Ver
0/1
$\widetilde{\text{Tag-Ver}}$
❑ Consider the following PPT collision-finder algorithm $\mathcal{C}$ for H, using $\mathcal{A}$ as a black-box
Key-Gen
$\mathcal{C}$
Collision-finder against H
$Q = \{m_1, m_2, \ldots, m_q\}$
$t_1, t_2, \ldots, t_q$
$t_i :=$ Tag-Gen$_k(H(m_i))$
(m^*, t^*)
$\mathcal{A}$
MAC-forger against $\widetilde{\Pi}_{\text{MAC}}$
Output (m_i, m^*), if $\exists m_i \in Q$: $m_i \neq m^*$ AND $H(m_i) = H(m^*)$
Pr[Hash-Coll$_{\mathcal{C},H}(n) = 1$] < negl(n)
= Pr[Coll] = negl()

So let us establish these 2 facts one by one. So both of these facts we are going to establish through a reduction proof. So what we are going to do is assume we have an arbitrary adversary who can forge your MAC or who can submit a forgery against the combined or the composed MAC and using that our goal is to find or create another poly-time adversary this $\mathcal{C}$ whose goal is to basically find a collision in the underlying collision-resistant hash function.

So what basically do is we do here in the reduction is the adversary or the MAC forger asks for tag on several messages of its choice, say Q number of messages, and to respond to those queries what this collision finder algorithm does is it runs the key generation algorithm of the composed MAC itself, generates a uniformly random key, and it computes the tag on the messages for which our MAC forger has asked for the tag and the responses are given back to the MAC forger and as per the syntax of the composed MAC, each of this t_i is basically computed by first hashing the message as per the underlying hash function H and then computing a fixed-length tag on the hash of each m_i under the unknown key k. So if you see what is happening in this reduction is from the viewpoint of this MAC forger, right, if we consider this MAC forger, the probability distribution of the information that it is receiving from this collision finder is exactly the same as this MAC forger would have expected by participating in a real instance of the CMA game against the composed MAC.

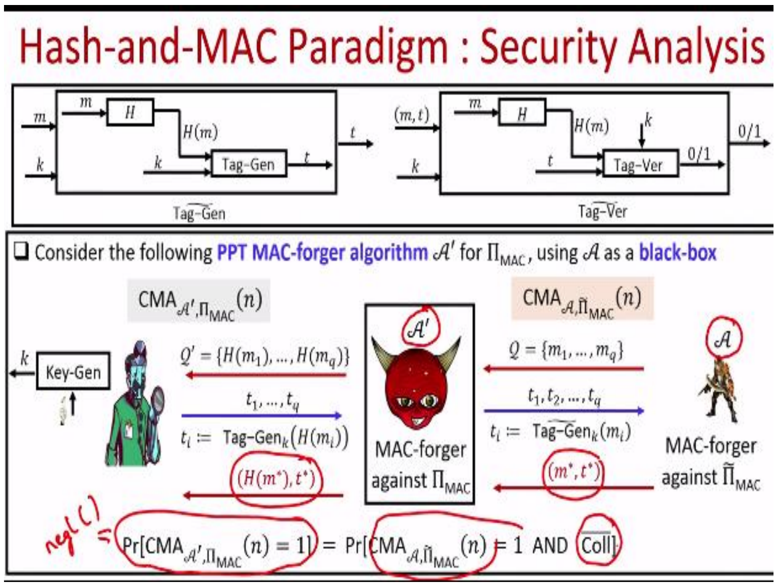
Because in a real instance of the MAC CMA game, what basically the adversary would have done is it would have submitted several messages of its choice and the tags that it would have seen in response would have exactly the same distribution as provided by the collision finder algorithm to this MAC forger, right. So view wise, the probability distribution of the information that the MAC forger is seeing in this reduction is exactly the same as its adversary would have seen in a genuine instance of the CMA experiment.

After getting the tags on the messages of adversary's choice, basically this adversary or the MAC forger outputs of a forgery and what is the goal of the collision finder? The goal of the collision finder is to spot a collision in the underlying hash function. So what it does is it basically parses the set of messages in the query set, so it has the set of messages for which the adversary has asked for the tag and it parses those messages and sees if there exist at least one such message m_i in that set such that the message m_i is different from the forged message m* and the hash of that message m_i is same as the hash of the message m*. If that is the case, then this collision finder algorithm outputs a collision for the underlying hash function H and it is easy to see that the running time of this collision finder algorithm is polynomial time, if the running time of the MAC forger algorithm is of polynomial time, right. So now it is easy to see that the probability that the collision finder successfully outputs a collision is exactly the same with which the event collision happens when this adversary MAC forger participates in an instance of the CMA game.

Because if indeed the event collision happens, that means there exists at least one message m_i which is different from m*, but the hash of m* is same as H of m_i, then indeed that case the collision finder algorithm will be able to successfully find the collision and event that m_i different from m*, but their hash values are same is nothing but the event collision. So since I am assuming that my underlying hash function is a collision-resistant hash function, then as per the definition of collision-resistant hash function, this probability should be some negligible function of the security parameter.

Since that probability is the same as the probability of the event collision happening that means the event collision also happens with some negligible probability. So that established our first fact that we wanted to establish.

(Refer Slide Time: 21:57)



Hash-and-MAC Paradigm : Security Analysis

m
H
$H(m)$
k
Tag-Gen
t
$\widetilde{\text{Tag-Gen}}$
(m, t)
k
Tag-Ver
0/1
$\widetilde{\text{Tag-Ver}}$
$\mathsf{CMA}_{\mathcal{A}', \Pi_{\mathsf{MAC}}}(n)$
$\mathsf{CMA}_{\mathcal{A}, \tilde{\Pi}_{\mathsf{MAC}}}(n)$
Key-Gen
$Q' = \{H(m_1), \ldots, H(m_q)\}$
$t_1, \ldots, t_q$
$t_i :=$ Tag-Gen$_k(H(m_i))$
$(H(m^*), t^*)$
$\mathcal{A}'$
MAC-forger against Π_{MAC}
$Q = \{m_1, \ldots, m_q\}$
$t_1, t_2, \ldots, t_q$
$t_i := \widetilde{\text{Tag-Gen}}_k(m_i)$
(m^*, t^*)
$\mathcal{A}$
MAC-forger against $\tilde{\Pi}_{\mathsf{MAC}}$
negl( )

❑ Consider the following PPT MAC-forger algorithm $\mathcal{A}'$ for Π_{MAC}, using $\mathcal{A}$ as a black-box

$$\Pr[\mathsf{CMA}_{\mathcal{A}', \Pi_{\mathsf{MAC}}}(n) = 1] = \Pr[\mathsf{CMA}_{\mathcal{A}, \tilde{\Pi}_{\mathsf{MAC}}}(n) = 1 \text{ AND } \overline{\mathsf{Coll}}]$$

Now let us establish the same fact, namely we want prove that assuming that the event collision does not happen, the probability that a MAC forger algorithm against the composed MAC can win the CMA game is upper bounded by some negligible function and that we establish by giving a reduction. So, we assume that we have a poly-time adversary, MAC adversary who can forge against, who can break the security or the CMA security of the composed MAC.

Now using that adversary, we design another adversary which we denote as say A**'** and its goal is to actually create a forgery against the fixed length base MAC. So now, what this adversary for the composed MAC does is it participates in an instance of the CMA game against the composed MAC and as per the rules of the CMA game, it asks for the tags for several messages of his choice.

What this adversary A**'** now does is it invokes an instance of CMA game against the fixed-length MAC, where the messages which are authenticated are of fixed length, namely *l*-bit messages, right. So the difference in the CMA part, CMA experiment here and the CMA experiment here is the length of the messages. In the CMA experiment against the composed scheme, each of this messages m_1, m_2, m_q, they could be of any length, but in the CMA game played against a fixed-length MAC, the query set can consist of only messages whose lengths are of *l* bits.

So what this adversary A**'** does is it plays a dual role on your left-hand side he is actually acting as an adversary and its goal is to win an instance of CMA game against a fixed-length

MAC, but on the right-hand side he is acting as a verifier right and he is interacting with the MAC forger against the arbitrary length inputs. So what this adversary A’ is now doing is it got a set of queries on which it is supposed to create tags as for the composed scheme.

To do that what it does is it hashes all those messages and though the hash of those messages are supplied as a query set to the verifier in the CMA game against the fixed-length MAC, basically adversary A’ is asking now for the tags on the hash of this messages m_1, m_2, m_q and as per the rules of the CMA game, the verifier of the CMA game against the fixed-length MAC generates a key and it responds back by computing a tag on hash of each of the messages which has been submitted by the adversary A’, right.

Now what this adversary A’ does is it has to respond back to the queries which the adversary A has raised to the adversary A’. So what it does is it supplies the same response to the MAC forger as it has retrieved from the verifier in the CMA game against the fixed-length MAC. So again, before proceeding further, let us understand what is happening here? If we consider the view of the adversary A against the composed scheme, the view is exactly the same as this adversary would have seen by participating in a genuine instance of CMA game against the composed MAC.

Basically, he would have submitted any messages of his choice and in response he would have seen tags where the distribution of the tag will be exactly the same. Namely, the tags will be computed by first hashing the messages, so that is why the hash of the messages are supplied by the adversary A’ here and once the hash of the messages have been computed, a fixed-length tag would have been obtained on the hash of those messages and that is what will be the distribution of the tag values that the adversary A would have seen in the composed MAC and that is what exactly is happening in the reduction.

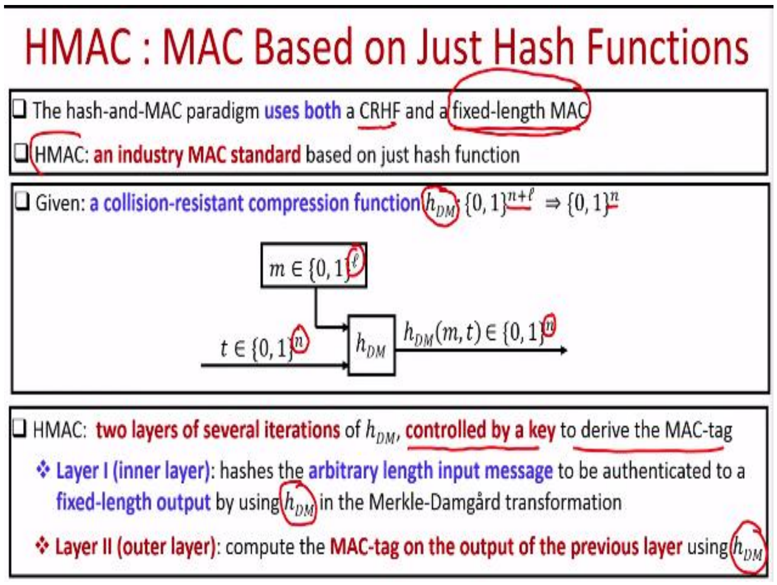
So as per the view of the adversary A is considered, its view is exactly the same as it would have expected by participating in a genuine instance of the CMA game against the composed MAC. So what this adversary A does now does it submits a forgery against a composed MAC say m*, t* and what the adversary A’ has to now do is it has to create or it has to output fixed-length forgery for the fixed-length base MAC, right. So what it does is basically whatever forgery m*, t* has been submitted by our adversary A, the adversary A’ computes a hash on the message m* and that is the message on which it is creating a forgery at the corresponding forged tag is t*. So now it is easy to see that what is the probability that our adversary A' which we have created here is able to win the CMA game against the base MAC or the fixed-length MAC. Well, the probability that it can win the CMA game against a fixed-length MAC is exactly the same as the adversary A wins the CMA game against the composed MAC and the event collision does not happen in the instance of the experiment.

Namely the event collision does not happen that means the hash of all the messages m_1, m_2, m_q, they are different from the hash of the message m* that is precisely the event collision not happening and adversary A still winning the game, right. If adversary A still wins the game that means the message m*, t* is indeed a forgery even in the presence of the event negation of collision, then indeed the H(m*, t*) is a valid forgery for your fixed-length MAC, which will imply that adversary A' has won the CMA game against the fixed-length MAC.

Now since we are assuming that our base MAC or the fixed-length MAC Π_{MAC} is secure, then this $\Pr[\mathrm{CMA}_{\mathcal{A}',\Pi_{\mathrm{MAC}}}(n) = 1]$ quantity is a negligible quantity, namely there exist no adversary A' who can win the CMA game against a fixed-length MAC with probability better than negligible probability in some security parameter. That means, the other event, the event on your right hand side, namely the adversary or the MAC forger winning the CMA game against your composed MAC in the presence of the complementary event collision also occurs with some negligible function in the security parameter.

So this proves the two facts that we wanted to state and this overall shows that the Hash-and-MAC paradigm is indeed going to give you a secure MAC for authenticating arbitrary long messages.

(Refer Slide Time: 28:30)



HMAC : MAC Based on Just Hash Functions

- The hash-and-MAC paradigm uses both a CRHF and a fixed-length MAC
- HMAC: an industry MAC standard based on just hash function

- Given: a collision-resistant compression function h_{DM}: $\{0,1\}^{n+\ell} \Rightarrow \{0,1\}^n$

$$\begin{tikzcd} & m \in \{0,1\}^{\ell} \arrow[d] & \\ t \in \{0,1\}^{n} \arrow[r] & \boxed{h_{DM}} \arrow[r, "h_{DM}(m,t) \in \{0,1\}^{n}"] & {} \end{tikzcd}$$

- HMAC: two layers of several iterations of h_{DM}, controlled by a key to derive the MAC-tag
  - Layer I (inner layer): hashes the arbitrary length input message to be authenticated to a fixed-length output by using h_{DM} in the Merkle-Damgård transformation
  - Layer II (outer layer): compute the MAC-tag on the output of the previous layer using h_{DM}

So now we have a general paradigm or generic paradigm, namely Hash-and-MAC paradigm. So what we will do is that we will see an instantiation of this paradigm which we call as HMAC and basically, the motivation of this HMAC construction is that we want to create a MAC which is just based on hash functions, right. So, if you see the Hash-and-MAC paradigm, it uses 2 cryptographic primitives, namely it uses a cryptographic collision-resistant hash function as well as it uses a fixed-length MAC.

Whereas the HMAC instantiation that we are going to see for the Hash-and-MAC paradigm, it is an industry MAC standard and the advantage of this HMAC construction is that it just involves hash functions, everything just based on hash function, you do not require another primitive namely a fixed-length MAC and let us see how exactly we construct this HMAC. So what we are given here is we are given a collision-resistant compression function which takes an input of size n+*l* bits and gives you an output of n bits and it is provably secure and it is collision resistant.

So, for concrete instantiation of such a compression function, you can take the Davies-Meyer construction which we had discussed in the last lecture, whose security we have proved in the ideal cipher model. So pictorially, this is how your Davies-Meyer function will operate. It takes an input of size n+*l* bits, but that input of size n+*l* bits you can imagine as if it consists of 2 chunks of inputs or 2 parts of input, one part of *l* bits, another part of n bits and overall the output is the fixed-size output of n bits.

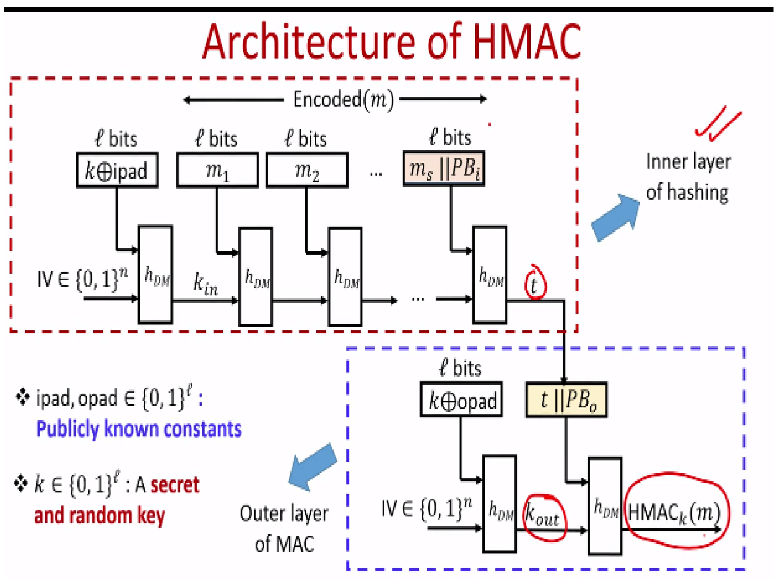
We had seen in the last lecture that this Davies-Meyer construction, you can instantiate using any secure block cipher, say AES, DES and so on. So what this HMAC does is basically it has 2 layers of several iterations of this Davies-Meyer construction and overall it is controlled by a key right, because remember we want to design a MAC, so we have a key for the MAC. So these 2 layers of the several iteration of the Davies-Meyer construction is controlled by a key and finally we obtain the tag.

So what exactly these 2 layers are, right? So the layer 1 is an inner layer and what it does, it takes the arbitrary input on which you want to compute the tag and it creates a fixed-length output of that input on which we want to compute the tag as per the Merkle-Damgard transformation by iteratively applying this Davies-Meyer function several times and remember the output of the Merkle-Damgard transformation gives you a fixed-length output.

Once we have the layer 1**,** what we do in the outer layer is we actually compute the tag on the output that we have obtained from the inner layer, and this again is done interestingly by using one instance of the Davies-Meyer function. So that is the difference here, right. So even though this can be viewed as an instantiation of your Hash-and-MAC paradigm because we are first hashing the message and then we are creating the tag on the message.

The interesting part here is that the tag for the message is also computed using an instantiation in one instance of the Davies-Meyer function or the collision-resistant compression function. We do not need a separate MAC for this second layer.

(Refer Slide Time: 31:55)

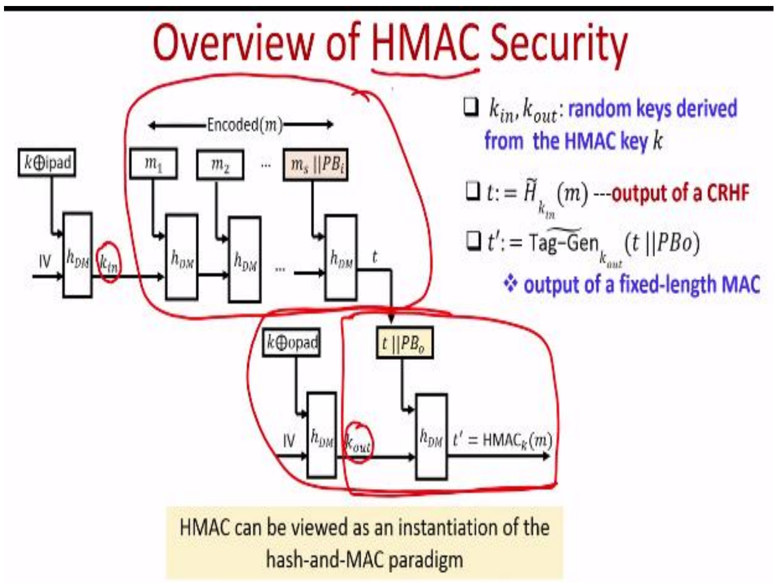


So let us see the architecture of the HMAC. So we have some publicly known constants here which we denote as ipad and opad, ipad basically stands for input pad and opad stands for output pad, each of length *l* bits and we have a secret key of *l* bits which is the master key for the overall MAC that we want to construct, right. So what we do is we invoke one instance of the Davies-Meyer construction where the overall input was of size n+*l* bits.

The message part is basically the XOR of the ipad with the key and we have a fixed publicly known IV and then what we do is we actually apply the Merkle-Damgard transformation. So what we have done basically here is you have the message m on which you want to compute the tag for the message under the key k. So, what we do is this part is basically inner layer of hashing where this inner layer of hashing this part is nothing but a Merkle-Damgard transformation where we would have encoded the message and then the last block we would have added some padding bits, which basically denotes the binary representation of the number of *l*-bit blocks which are present in your message and then we would have done a sequence of chaining, where in each iteration we will have one invocation of your Davies-Meyer function on the current block of the message and the output of the previous invocation of h_{DM}. The difference here is that the IV now here is an IV or a key value which is obtained by running one invocation of the Davies-Meyer function on this input k XOR ipad with a fixed known IV.

So this is your inner layer of hashing and it will give you a fixed-size output and what we now do in the MAC part or the outer layer is the following. We take the output t which is of size say n bits and n might be less than *l*. So what we do is we do the padding here and we do 2 invocations of the Davies-Meyer function here where the first invocation is now with the XOR of k with opad and whatever output comes it is used as a key, right, or it is used as one of the inputs along with the tag concatenated with the output pad being input and whatever output comes out that is taken as the overall output of the HMAC construction for your message and this second layer actually is the outer layer of the message authentication code. So you have the inner layer of hashing where we take the arbitrary length message, compute a fixed-length tag, and then we take that fixed-length tag and then again do 2 invocations of the Davies-Meyer function to compute the final MAC.

(Refer Slide Time: 34:44)



Overview of HMAC Security

- k_{in}, k_{out}: random keys derived from the HMAC key k
- $t := \widetilde{H}_{k_{in}}(m)$ ---output of a CRHF
- $t' :=$ $\widetilde{\text{Tag-Gen}}_{k_{out}}(t \| PBo)$
  - output of a fixed-length MAC

HMAC can be viewed as an instantiation of the hash-and-MAC paradigm

So, now we will not go into the full-fledged security of the HMAC construction, but let us intuitively try to understand what is happening here. So this is the overall architecture of the HMAC construction. So what we are doing here is we are actually deriving 2 keys by running the Davies-Meyer function here, you can call those keys as k_{in} and k_{out} right and k_{in} is basically serving as the IV for your Merkle-Damgard transformation, whereas the k_{out} is serving as the one part of the input for the Davies-Meyer function when we are actually computing the tag.

So imagine that we define a function G which takes a key k for the overall MAC and it invokes Davies-Meyer function twice like this, sorry for this typo this DM should come in the subscript. So we have 2 invocations of the Davies Meyer function. The first invocation is on input IV concatenated with the XOR of k and ipad and the second invocation is on the input IV concatenated with k XOR opad and say the resultant output is k_{in} and k_{out}, right. Interestingly, what we can prove is the construction G(k) that we have defined like this.

It is like a pseudorandom generator if we go in the ideal cipher model. So in the security definition of the pseudorandom generator, the game was that the adversary is given a challenge sample and it has to distinguish whether the sample is generated by the pseudorandom generator or a true random generator, but if we take that game in the ideal cipher model, then apart from the challenge sample which is given to the adversary. Adversary have also got oracle access to the underlying block ciphers which are used in the instantiation of the Davies-Meyer function, and based on oracle access to the underlying block ciphers which are used in the Davies-Meyer function, the goal of the adversary is to distinguish the challenge sample whether it is truly random or whether its pseudorandom. So, we can formally prove that a construction G(k) that we have defined like this indeed is a pseudorandom generator in the ideal cipher model.

That means what that the distribution of the 2 keys that we have obtained, namely k_{in} and k_{out} which we are using in the inner layer and the outer layer, their distribution is computationally indistinguishable from a distribution of truly random keys k_{in} and k_{out} that would have been obtained by running a true random generator. So, that means you can safely consider the derived key k_{in} and the derived key k_{out} to be as good as true random keys. Now, let us focus on the inner layer of the hashing, right.

So again sorry for the typo here, this should H_{MD}, so basically this denotes the Merkle-Damgard transformation. So what we are doing in the inner layer of hashing is basically a Merkle-Damgard transformation on the input k_{in} concatenated with the encoded m. The only difference now is instead of a fixed IV which is all zeros in the case of the Merkle-Damgard transformation, the IV with which we are triggering the Merkle-Damgard transformation is a derived key k_{in}, right.

So, you can imagine in some way that whatever function that we have computed by applying the Merkle-Damgard transformation on the encoded input and concatenated with k_{in} that is kind of define a keyed hash function which I denote as $\widetilde{H}$ right, and we can prove that this function $\widetilde{H}_{kin}$ constitutes a collision-resistant hash function even if adversary has seen polynomial number of outputs of this keyed hash function for several messages of its choice. So, we can formally establish this that is very simple, but due to lack of time, I am not going into the formal details.

So, what it means is even though we are doing a Merkle-Damgard transformation during the inner layer of hashing controlled by a key k_{in} that constitutes a keyed hash function, which can be formally proved to be collision resistant even if adversary has oracle access to that keyed hash function. That means, whatever output t that we are obtaining here that you cannot consider as an outcome of a collision-resistant hash function, right. So that is what the first part of the overview of the security proof.

The second part is with respect to the MAC part or the fixed-length MAC part that we are computing here. S, imagine that I define a tag generation algorithm, again sorry for the typo here it should be H_{DM} which basically is one instantiation of the Davies-Meyer construction with a key as input concatenated with the message also as the input, right. So that will be the overall tag generation algorithm for the message m under the key k.

So basically, the part that I have highlighted here under the box that is nothing but the outcome of this tag generation algorithm, that means you can imagine that this t' that is coming is an output of a tag generation algorithm and we can correspondingly define a corresponding tag verification algorithm and interestingly we can prove that the way this Davies-Meyer function is operating here in the ideal cipher model, we can consider this instantiation of Davies-Meyer function to be a secure MAC for fixed-lengths inputs.

So if that is the case, then the output t' that we are obtaining here it can be considered as the tag for the message t concatenated with padded bits under the key k_{out} and as per our assumption the key k_{out} is computationally indistinguishable from a uniformly random key. That means we can imagine that the t' is an outcome of a fixed-length MAC. So, overall what we can imagine is that even though everything involves instantiation of the Davies-Meyer construction, overall this HMAC can be considered as an instantiation of your Hash-and-MAC paradigm.

Because the first part, namely the inner part is nothing but doing a keyed hashing for the message as per the Merkle-Damgard transformation, where the key is computationally indistinguishable because it is derived from one invocation of the Davies-Meyer function which in the ideal cipher model we can prove to be equivalent to a pseudorandom generator and actual message authentication code right; this part is basically a keyed invocation of Davies-Meyer function on the output that we are obtaining from the hashing stage and again in the ideal cipher model, we can prove that the instantiation of this Davies-Meyer function as per the key where the key k_{out} is computationally indistinguishable from a uniformly random key gives you a message authentication code which cannot be forged. So as a result, we can view this overall construction of HMAC as an instantiation of the Hash-and-MAC paradigm and as we have stated earlier, the Hash-and-MAC paradigm is indeed secure if your underlying hash function is collision resistant and if your MAC is a secure MAC for fixed-length messages.

The only difference in the proof is that underlying components are secure in the ideal cipher model, namely the keyed hash function is collision resistant in the ideal cipher model and the underlying fixed-length MAC can be proved to be a secure MAC in the ideal cipher model and that implies overall that HMAC is secure, and this is really one of the highly popular instantiation of message authentication code which is used in practice.

So that brings me to the end of this lecture. Just to summarize in this lecture, we have seen the Hash-and-MAC paradigm which is a generic construction and which gives you a message authentication codes for arbitrary long messages by combining or composing a MAC for fixed-length messages and a collision-resistant hash function and we had also seen an instantiation of this paradigm, namely the construction of the HMAC message authentication code. Thank you.