Message Authentication Code

Ali El Kaafarani

[1]Mathematical Institute
[2] PQShield Ltd.

UNIVERSITY OF OXFORD

Outline

1. Message Integrity

2. Message Authentication Code (MAC)

Message Integrity

- We want parties to *securely* communicate over insecure channels.
- Is is enough to encrypt the messages?
- What if the messages were modified in transit?
- What about authenticity?
- There is clearly a difference between secrecy and Integrity, therefore different cryptographic tools should be used to achieve both of them.

What about perfect secrecy

- Recall that OTP is a perfectly secure encryption scheme.
- Does it ensure any level of message integrity?
- From a given ciphertext, you can produce a new *valid* ciphertext, by just flipping a single bit!
- This could change the amount of money that you want to transfer from your account.
- Perfect secrecy is not violated here!
- But, perfect secrecy *simply* doesn’t imply message integrity!

Outline

1. Message Integrity
2. Message Authentication Code (MAC)

Message Authentication Code (MAC)

- Message authentication code is the tool to be used to ensure message integrity.
- Informally speaking, the MAC's goal is to prevent an adversary from tampering with the messages.
- To prevent the adversary from impersonating, parties need to share a secret key as in the encryption!

MAC: Formal Definition

Definition

A MAC *consists of the following three probabilistic polynomial-time algorithms* (KeyGen, Mac, Verify)*:*

- KeyGen(1^n)*: takes the security parameter* n *and outputs a key* k *s.t.* $|k| \geq n$
- Mac$_k(m \in \{0,1\}^*)$*: is a tagging algorithm, takes a key* k *and a message* m *and outputs a tag* t*.*
- Verify$_k(m, t)$*: a deterministic algorithm that outputs a bit* b*, 0 for invalid and 1 for valid.*

MAC

- **Correctness of MAC:** $\forall n, \forall k \leftarrow \text{KeyGen}(1^n)$ and $\forall m \in \{0,1\}^*$, $\text{Verify}_k(m, \text{Mac}_k(m)) = 1$ holds.
- Fixed-length MAC: if it is just defined for messages $m \in \{0,1\}^{\ell(n)}$, we call the scheme a *fixed-length MAC for messages of length* $\ell(n)$.

Security of MAC-Intuition

- Intuitively speaking, an adversary should not be able to efficiently produce a valid tag on a new message that wasn't authenticated before.
- Taking into consideration that the adversary can see all the messages/tags pairs, in our formal definition, we need to give the adversary access to a tagging Oracle.

Security of MAC- Formal Definition

Given $S = (\text{KeyGen}, \text{Mac}, \text{Verify})$, an adversary $\mathcal{A}$, and a security parameter n, we define the following experiment:

Experiment

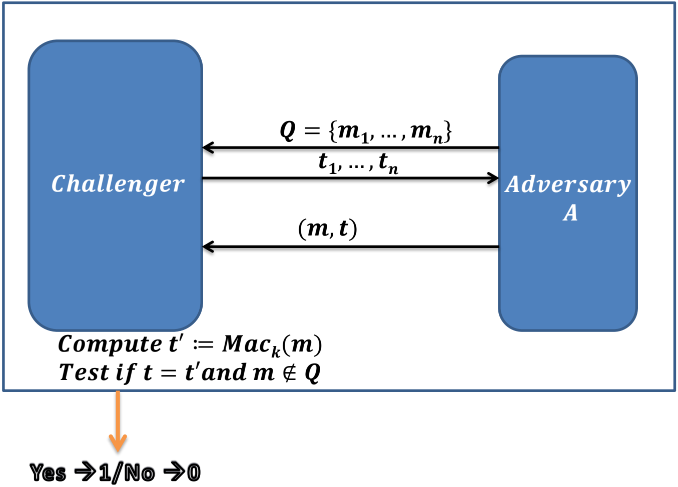
$\text{Mac}_{\mathcal{A},S}^{\text{Unforg}}$

- *Key generation:* $k \leftarrow \text{KeyGen}(1^n)$*.*
- *Tag queries: the adversary* $\mathcal{A}$ *is given oracle access to* $\text{Mac}_k()$*. The set of all his queries is* Q*.*
- *Adversary's output: the adversary* $\mathcal{A}$ *eventually outputs* (m, t)
- *Experiment's output: if*

$$\text{Verify}_k(m, t) = 1 \wedge m \notin Q$$

output 1, otherwise output 0.

MAC^{unforg} Game



Security of MAC

A MAC scheme is said to be *Existentially unforgeable under an adaptive chosen-message attack* if no efficient adversary can win the previous game with non-negligible probability. Formally speaking,

Definition

A message authentication code $S = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ *is secure if for all probabilistic polynomial-time adversary* $\mathcal{A}$*, the following holds*

$$\Pr[\mathsf{Mac}^{\mathsf{Unforg}}_{\mathcal{A},S}(n) = 1] \leq \mathsf{negl}(n)\,.$$

MAC and Replay attacks

- An adversary cannot change the message without being detected by the receiver if it has a valid tag.
- However, the adversary can replay and send the same message again.
- The receiver cannot really detect this malicious behaviour.
- Therefore MAC doesn't prevent replay attacks from happening.
- Common techniques to prevent replay attacks:
  - Counters: users maintain synchronized state.
  - Time-stamps: add the current time to the beginning of the messages before authenticating them.

Security of MAC- what is the difference here?

Given $S = (\text{KeyGen}, \text{Mac}, \text{Verify})$, an adversary $\mathcal{A}$, and a security parameter n, we define the following experiment:

Experiment

$\text{Mac}_{\mathcal{A},S}^{\text{Unforg}}$

- *Key generation:* $k \leftarrow \text{KeyGen}(1^n)$.
- *Tag queries: the adversary* $\mathcal{A}$ *is given oracle access to* $\text{Mac}_k()$. *The set of all his queries is* Q.
- *Adversary's output: the adversary* $\mathcal{A}$ *eventually outputs* (m, t)
- *Experiment's output: if*

$$\text{Verify}_k(m, t) = 1 \land (m, t) \notin Q$$

output 1, otherwise output 0.

Strongly Secure MAC

Informally speaking, if a MAC scheme is strongly secure, then adversaries can't produce tags on any message (including already authenticated ones!).

Definition

A message authentication code $S = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ *is strongly secure if for all probabilistic polynomial-time adversary* $\mathcal{A}$*, the following holds*

$$\Pr[\mathsf{Mac}^{St-\mathsf{Unforg}}_{\mathcal{A},S}(n) = 1] \leq \mathsf{negl}(n)\,.$$

If the Mac algorithm in S is deterministic, and the verification is done by computing $t' = \mathsf{Mac}_k(m)$ and testing whether or not $t' = t$, then Secure MACs are Strongly secure as well.

MAC- Side Channel Attacks

- When giving the adversary access to a MAC oracle, he just learns the output, not the time taken by the Oracle to perform the task.
- This is not what happens in the real systems!
- If the MAC verification doesn't use time independent string comparison (in the case of deterministic MAC), then the adversary can measure the difference in time taken to compare j or $j+1$ bytes!
- This is a realistic attack, Xbox 360 had this difference, i.e. between rejection times, equal to 2.2 milliseconds.
- Attackers managed to exploit this!
- Conclusion: MAC verification should always compare all the bytes.

MAC- Side Channel Attacks

- When giving the adversary access to a MAC oracle, he just learns the output, not the time taken by the Oracle to perform the task.
- This is not what happens in the real systems!
- If the MAC verification doesn't use time independent string comparison (in the case of deterministic MAC), then the adversary can measure the difference in time taken to compare j or $j + 1$ bytes!
- This is a realistic attack, Xbox 360 had this difference, i.e. between rejection times, equal to 2.2 milliseconds.
- Attackers managed to exploit this!
- Conclusion: MAC verification should always compare all the bytes.

MAC- Side Channel Attacks

- When giving the adversary access to a MAC oracle, he just learns the output, not the time taken by the Oracle to perform the task.
- This is not what happens in the real systems!
- If the MAC verification doesn't use time independent string comparison (in the case of deterministic MAC), then the adversary can measure the difference in time taken to compare j or $j + 1$ bytes!
- This is a realistic attack, Xbox 360 had this difference, i.e. between rejection times, equal to 2.2 milliseconds.
- Attackers managed to exploit this!
- Conclusion: MAC verification should always compare all the bytes.

MAC- Side Channel Attacks

- When giving the adversary access to a MAC oracle, he just learns the output, not the time taken by the Oracle to perform the task.
- This is not what happens in the real systems!
- If the MAC verification doesn't use time independent string comparison (in the case of deterministic MAC), then the adversary can measure the difference in time taken to compare j or $j+1$ bytes!
- This is a realistic attack, Xbox 360 had this difference, i.e. between rejection times, equal to 2.2 milliseconds.
- Attackers managed to exploit this!
- Conclusion: MAC verification should always compare all the bytes.

MAC- Side Channel Attacks

- When giving the adversary access to a MAC oracle, he just learns the output, not the time taken by the Oracle to perform the task.
- This is not what happens in the real systems!
- If the MAC verification doesn't use time independent string comparison (in the case of deterministic MAC), then the adversary can measure the difference in time taken to compare j or $j+1$ bytes!
- This is a realistic attack, Xbox 360 had this difference, i.e. between rejection times, equal to 2.2 milliseconds.
- Attackers managed to exploit this!
- Conclusion: MAC verification should always compare all the bytes.

MAC- Side Channel Attacks

- When giving the adversary access to a MAC oracle, he just learns the output, not the time taken by the Oracle to perform the task.
- This is not what happens in the real systems!
- If the MAC verification doesn't use time independent string comparison (in the case of deterministic MAC), then the adversary can measure the difference in time taken to compare j or $j+1$ bytes!
- This is a realistic attack, Xbox 360 had this difference, i.e. between rejection times, equal to 2.2 milliseconds.
- Attackers managed to exploit this!
- Conclusion: MAC verification should always compare all the bytes.

A fixed-Length MAC from a PRF

Definition

Given a pseudorandom function F, a fixed-length MAC for messages of length n consists of the two following algorithms:

- Mac$(k \in \{0,1\}^n, m \in \{0,1\}^n)$*: it outputs the tag* $t \leftarrow F_k(m)$.
- Verify$(k \in \{0,1\}^n, m \in \{0,1\}^n, t \in \{0,1\}^n)$*: it output 1 iff* $t = F_k(m)$

If $|m| \neq |k|$, then Mac outputs $\perp$ and Verify outputs 0.

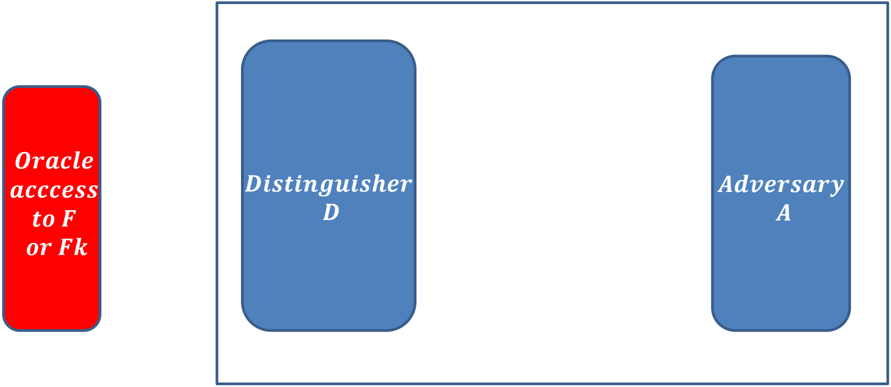
A fixed-Length MAC from a PRF

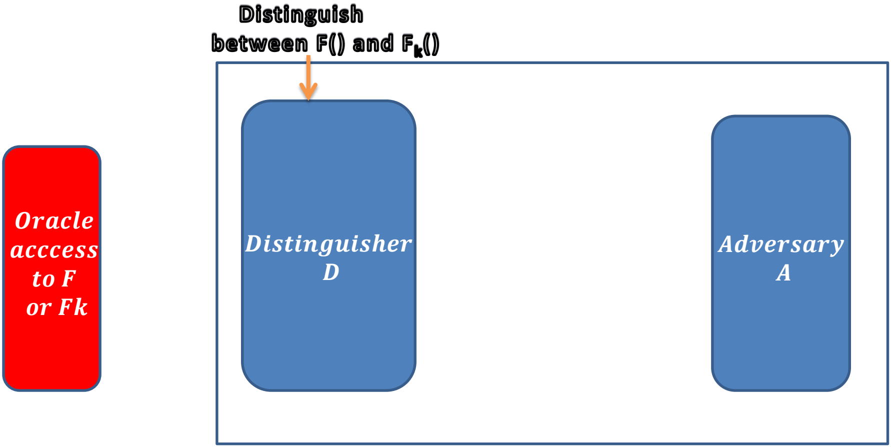
Theorem

If F is a pseudorandom function, then the fixed-length MAC for messages of length n is secure.

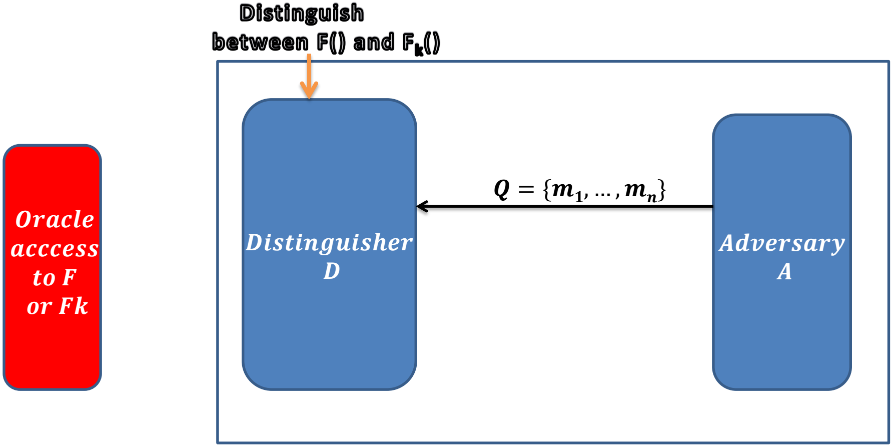
Intuition of the proof:

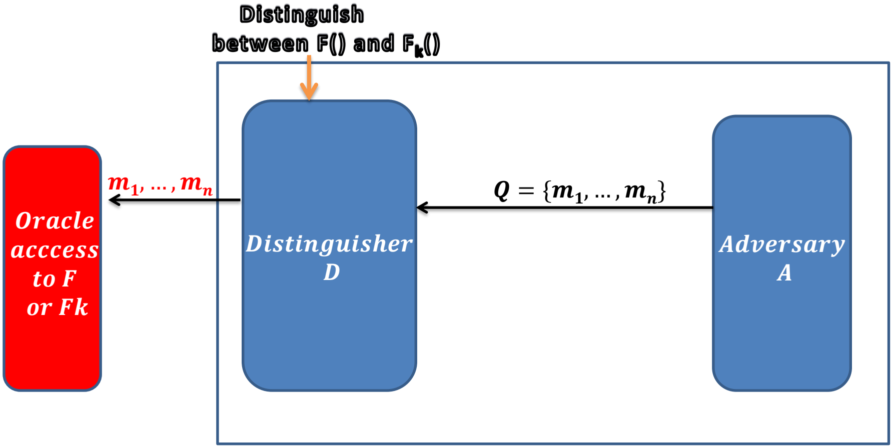
- Define D as a distinguisher that is given access to some function and needs to tell whether this function is pseudorandom or truly random.
- Let $\mathcal{A}$ be the adversary trying to attack MAC.
- D will emulate the MAC experiment for $\mathcal{A}$ and check if it succeeds in producing a valid tag on a new message m.
- if $\mathcal{A}$ manages to produce a valid tag, D will guess that its oracle is "pseudo-random", otherwise it outputs "truly random"

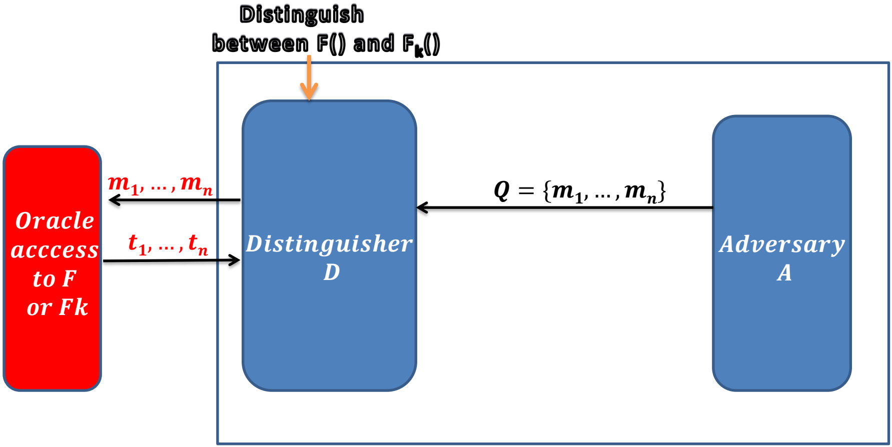

Oracle acccess to F or Fk
Distinguisher D
Adversary A


Distinguish
between F() and F_k()
Oracle acccess to F or Fk
Distinguisher D
Adversary A

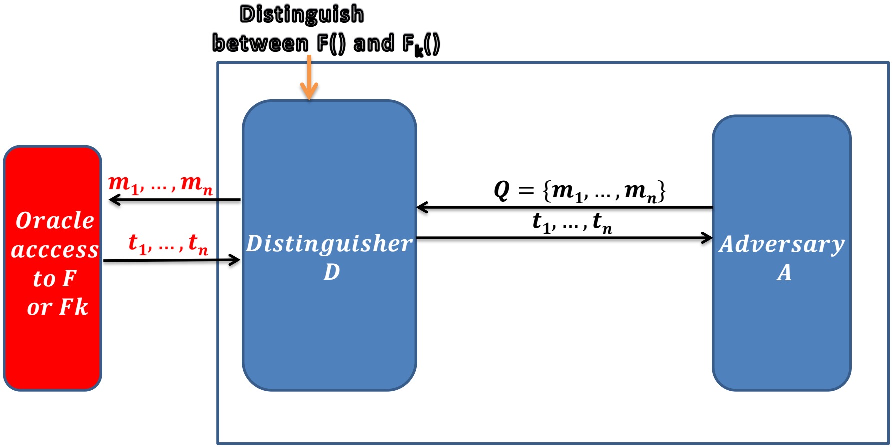
Note that in the "adaptive" setting, the messages $m_1, \ldots, m_n$ will be sent separately.

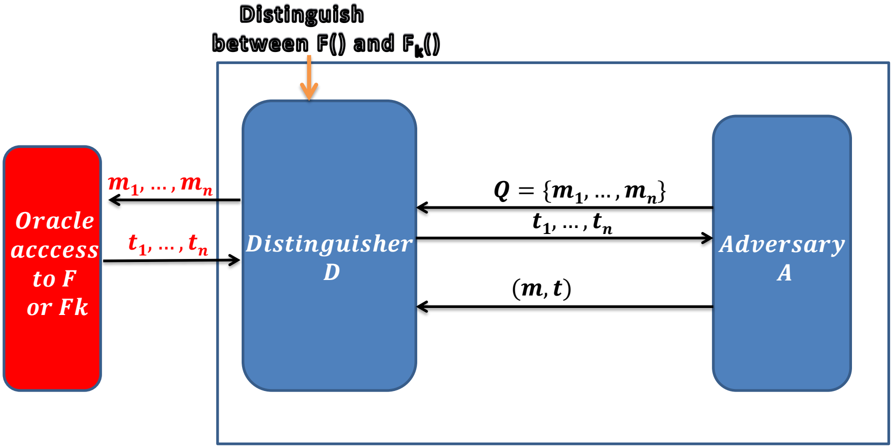


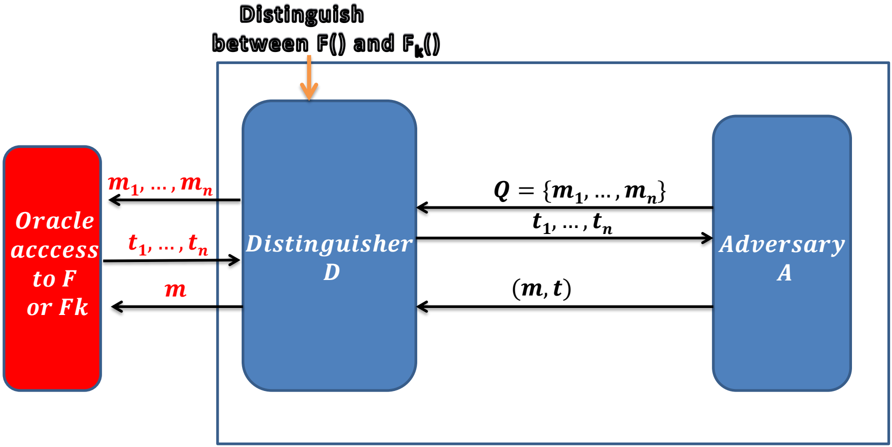

Distinguish
between F() and F_k()
$m_1, \dots, m_n$
Oracle acccess to F or Fk
$Q = \{m_1, \dots, m_n\}$
Distinguisher D
Adversary A

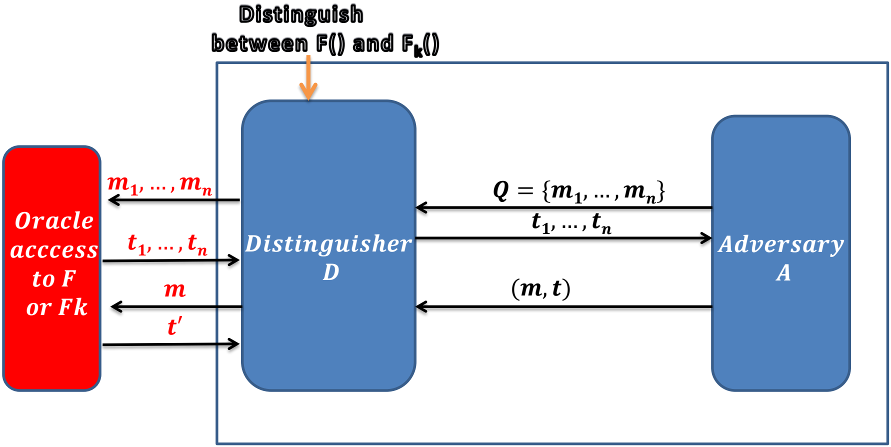


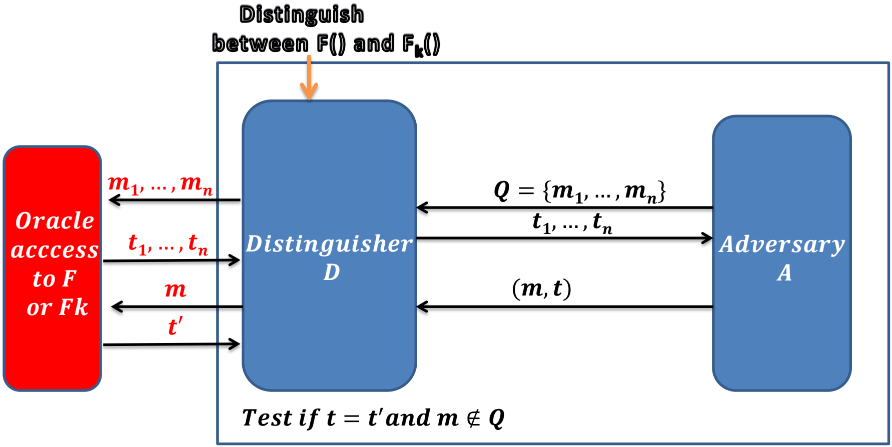
Distinguish
between F() and F_k()
Oracle acccess to F or Fk
m_1, … , m_n
t_1, … , t_n
Distinguisher D
Q = {m_1, … , m_n}
Adversary A

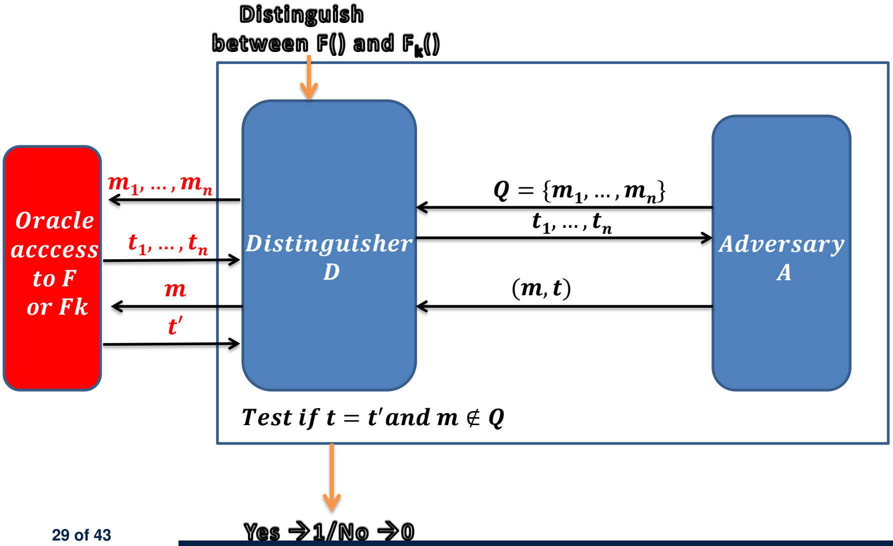

Distinguish
between F() and $F_k()$
Oracle acccess to F or Fk
$m_1, \dots, m_n$
$t_1, \dots, t_n$
Distinguisher D
$Q = \{m_1, \dots, m_n\}$
$t_1, \dots, t_n$
Adversary A


Distinguish
between F() and F_k()
Oracle acccess to F or Fk
m_1, …, m_n
t_1, …, t_n
Distinguisher D
Q = {m_1, …, m_n}
t_1, …, t_n
(m, t)
Adversary A


Distinguish
between F() and $F_k()$
Oracle acccess to F or Fk
$m_1, \dots, m_n$
$t_1, \dots, t_n$
m
Distinguisher D
$Q = \{m_1, \dots, m_n\}$
$t_1, \dots, t_n$
(m, t)
Adversary A


Distinguish
between F() and F_k()
$m_1, \dots, m_n$
Oracle acccess to F or Fk
$t_1, \dots, t_n$
m
t'
Distinguisher D
$Q = \{m_1, \dots, m_n\}$
$t_1, \dots, t_n$
(m, t)
Adversary A


Distinguish
between F() and $F_k()$
Oracle acccess to F or Fk
$m_1, \dots, m_n$
$t_1, \dots, t_n$
m
t'
Distinguisher D
$Q = \{m_1, \dots, m_n\}$
$t_1, \dots, t_n$
(m, t)
Adversary A
Test if $t = t'$ and $m \notin Q$


Distinguish
between F() and $F_k()$
Oracle acccess to F or Fk
$m_1, \dots, m_n$
$t_1, \dots, t_n$
m
t'
Distinguisher D
$Q = \{m_1, \dots, m_n\}$
$t_1, \dots, t_n$
(m, t)
Adversary A
Test if $t = t'$ and $m \notin Q$
Yes →1/No →0

A fixed-Length MAC from a PRF

Sketch Proof.

We first analyse the security of the MAC if we use a truly random function f, and then we replace f by a psendorandom function F_k. Let the first MAC system be $S' = (\mathsf{KeyGen}', \mathsf{Mac}', \mathsf{Verify}')$ and the second MAC be $S = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$. Since for any message $m \notin Q$, the value $t = f(m)$ is uniformly distributed in $\{0,1\}^n$ from the point of view of the adversary $\mathcal{A}$ (remember, KeyGen' samples f uniformly at random from Func_n), it is then straight forward to deduce that

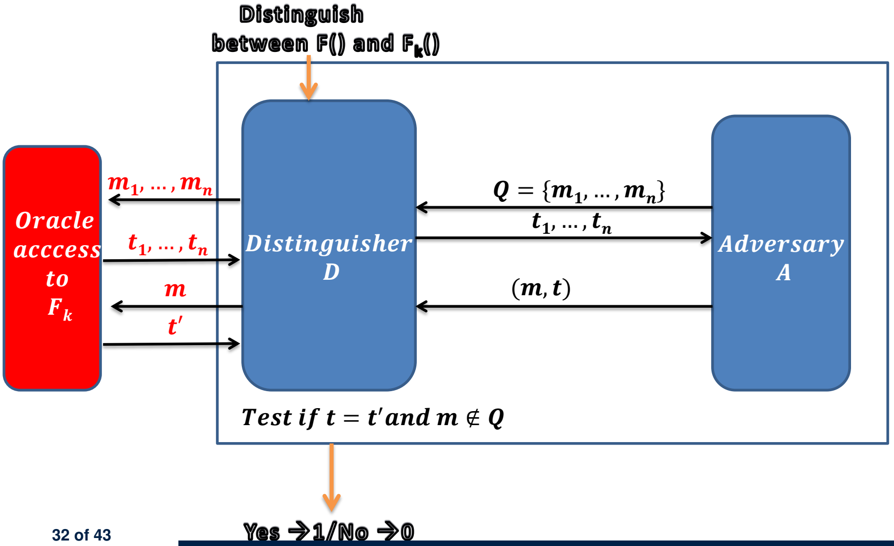
$$\Pr[\mathsf{Mac}^{\mathsf{Unforg}}_{\mathcal{A},S'}(n) = 1] \leq 2^{-n}.$$

A fixed-Length MAC from a PRF

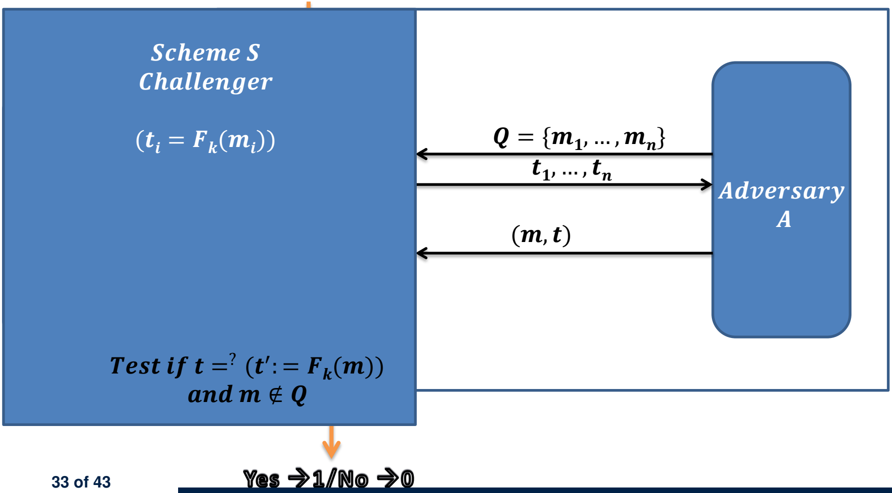
Sketch Proof.

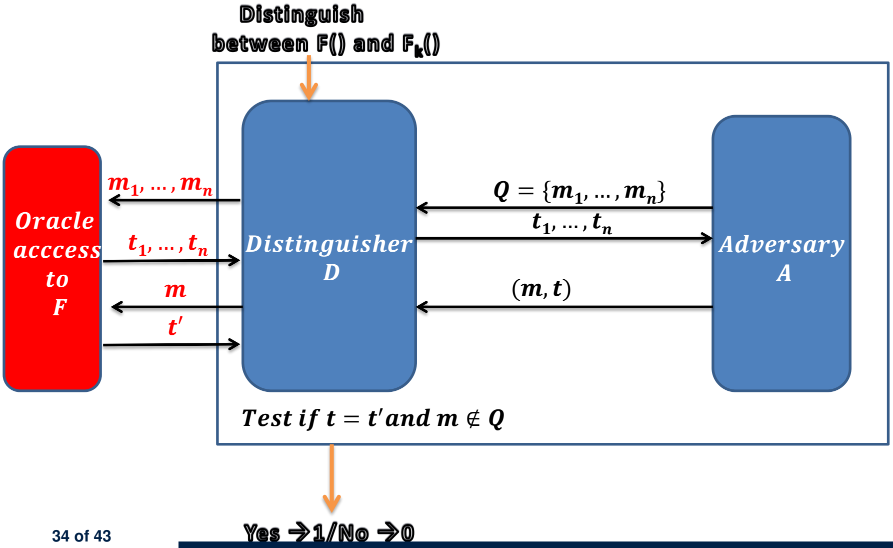
We can distinguish between two cases:

- D*'s oracle is a pseudo-random function: in this case, the view of* $\mathcal{A}$ *that is run as a subroutine by* D *and its view in the experiment* $\text{Mac}^{\text{Unforg}}_{\mathcal{A},S}(n)$ *are distributed identically. Moreover,* D *outputs 1 exactly when* $\text{Mac}^{\text{Unforg}}_{\mathcal{A},S'}(n)$ *outputs 1.*
- D*'s oracle is a truly-random function: in this case, the view of* $\mathcal{A}$ *that is run as a subroutine by* D *and its view in the experiment* $\text{Mac}^{\text{Unforg}}_{\mathcal{A},S'}(n)$ *are distributed identically. Moreover,* D *outputs 1 exactly when* $\text{Mac}^{\text{Unforg}}_{\mathcal{A},S'}(n)$ *outputs 1.*

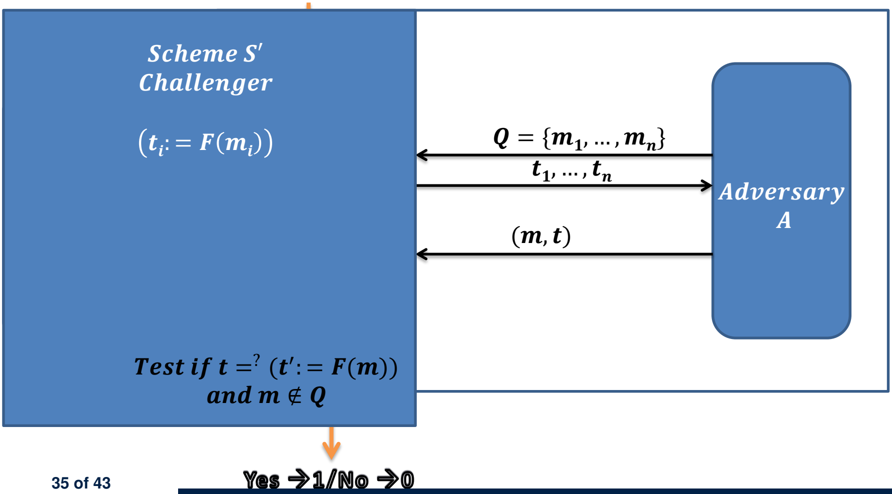

Distinguish
between F() and $F_k()$
Oracle acccess to F_k
$m_1, \dots, m_n$
$t_1, \dots, t_n$
m
t'
Distinguisher D
$Q = \{m_1, \dots, m_n\}$
$t_1, \dots, t_n$
(m, t)
Adversary A
Test if $t = t'$ and $m \notin Q$
Yes →1/No →0

MAC^{unforg}


Scheme S
Challenger
$(t_i = F_k(m_i))$
$Q = \{m_1, \dots, m_n\}$
$t_1, \dots, t_n$
Adversary
A
(m, t)
Test if $t =^? (t' := F_k(m))$
and $m \notin Q$
Yes →1/No →0


Distinguish
between F() and F_k()
Oracle acccess to F
m_1, … , m_n
t_1, … , t_n
m
t'
Distinguisher D
Q = {m_1, … , m_n}
t_1, … , t_n
(m, t)
Adversary A
Test if t = t' and m ∉ Q
Yes →1/No →0

MAC^{unforg}


Scheme S'
Challenger
$(t_i := F(m_i))$
$Q = \{m_1, \dots, m_n\}$
$t_1, \dots, t_n$
(m, t)
Adversary
A
Test if $t =^? (t' := F(m))$
and $m \notin Q$
Yes → 1/No → 0

Sketch Proof.

As a result, we have that

$$\Pr[\mathsf{Mac}^{\mathsf{Unforg}}_{\mathcal{A},S'}(n) = 1] = \Pr[D^{f()}(n) = 1] \tag{1}$$

and

$$\Pr[\mathsf{Mac}^{\mathsf{Unforg}}_{\mathcal{A},S}(n) = 1] = \Pr[D^{F_k()}(n) = 1] \tag{2}$$

Sketch Proof.

If F_k is a pseudo-random function, using (1) and (2) we can deduce

$$|\Pr[\mathsf{Mac}^{\mathsf{Unforg}}_{\mathcal{A},S'}(n) = 1] - \Pr[\mathsf{Mac}^{\mathsf{Unforg}}_{\mathcal{A},S}(n) = 1]| \leq \mathsf{negl}(n) \quad (3)$$

together with (1), we have

$$\Pr[\mathsf{Mac}^{\mathsf{Unforg}}_{\mathcal{A},S}(n) = 1] \leq 2^{-n} + \mathsf{negl}(n)\,.$$

From fixed length MAC to general MAC for arbitrary-length messages.

- If the PRF has a larger domain, MAC is secure for longer messages.
- Furthermore, if the PRF can take arbitrary-length input, then the previous MAC is secure for arbitrary-length messages.
- Our problem is with existing pseudo-random functions used in practice.
- They are block ciphers that can just take short fixed-length inputs!
- Question: How to build a MAC for arbitrary-length messages?

A general MAC from a fixed-length one

Potential attacks:

- Block re-ordering attack: the attacker changes the order of blocks, if (t_1, t_2) is a valid tag on (m_1, m_2) where $m_1 \neq m_2$, then (t_2, t_1) is a valid tag on (m_2, m_1) as m_2, m_1 is a different message!
  Solution: authenticate a block index with each block.
- Truncation attack: the attacker removes blocks from the end of the message and their corresponding blocks from the tag.
  Solution: authenticate the message length with each block
- Mix-and-match attack: the attacker has valid tags (t_1, t_2, t_3) and (t'_1, t'_2, t'_3) on the messages (m_1, m_2, m_3) and (m'_1, m'_2, m'_3). He outputs (t_1, t'_2, t_3) on the message (m_1, m'_2, m_3).
  Solution: authenticate a *random message identifier* along with each block.

A general MAC from a fixed-length one

Potential attacks:

- Block re-ordering attack: the attacker changes the order of blocks, if (t_1, t_2) is a valid tag on (m_1, m_2) where $m_1 \neq m_2$, then (t_2, t_1) is a valid tag on (m_2, m_1) as m_2, m_1 is a different message!
Solution: authenticate a block index with each block.
- Truncation attack: the attacker removes blocks from the end of the message and their corresponding blocks from the tag.
Solution: authenticate the message length with each block
- Mix-and-match attack: the attacker has valid tags (t_1, t_2, t_3) and (t'_1, t'_2, t'_3) on the messages (m_1, m_2, m_3) and (m'_1, m'_2, m'_3). He outputs (t_1, t'_2, t_3) on the message (m_1, m'_2, m_3).
Solution: authenticate a *random message identifier* along with each block.

A general MAC from a fixed-length one

Potential attacks:

- Block re-ordering attack: the attacker changes the order of blocks, if (t_1, t_2) is a valid tag on (m_1, m_2) where $m_1 \neq m_2$, then (t_2, t_1) is a valid tag on (m_2, m_1) as m_2, m_1 is a different message! Solution: authenticate a block index with each block.
- Truncation attack: the attacker removes blocks from the end of the message and their corresponding blocks from the tag. Solution: authenticate the message length with each block
- Mix-and-match attack: the attacker has valid tags (t_1, t_2, t_3) and (t'_1, t'_2, t'_3) on the messages (m_1, m_2, m_3) and (m'_1, m'_2, m'_3). He outputs (t_1, t'_2, t_3) on the message (m_1, m'_2, m_3). Solution: authenticate a *random message identifier* along with each block.

A general MAC from a fixed-length one

Potential attacks:

- Block re-ordering attack: the attacker changes the order of blocks, if (t_1, t_2) is a valid tag on (m_1, m_2) where $m_1 \neq m_2$, then (t_2, t_1) is a valid tag on (m_2, m_1) as m_2, m_1 is a different message!
  Solution: authenticate a block index with each block.
- Truncation attack: the attacker removes blocks from the end of the message and their corresponding blocks from the tag.
  Solution: authenticate the message length with each block
- Mix-and-match attack: the attacker has valid tags (t_1, t_2, t_3) and (t'_1, t'_2, t'_3) on the messages (m_1, m_2, m_3) and (m'_1, m'_2, m'_3). He outputs (t_1, t'_2, t_3) on the message (m_1, m'_2, m_3).
  Solution: authenticate a *random message identifier* along with each block.

A general MAC from a fixed-length one

Potential attacks:

- Block re-ordering attack: the attacker changes the order of blocks, if (t_1, t_2) is a valid tag on (m_1, m_2) where $m_1 \neq m_2$, then (t_2, t_1) is a valid tag on (m_2, m_1) as m_2, m_1 is a different message!
Solution: authenticate a block index with each block.
- Truncation attack: the attacker removes blocks from the end of the message and their corresponding blocks from the tag.
Solution: authenticate the message length with each block
- Mix-and-match attack: the attacker has valid tags (t_1, t_2, t_3) and (t'_1, t'_2, t'_3) on the messages (m_1, m_2, m_3) and (m'_1, m'_2, m'_3). He outputs (t_1, t'_2, t_3) on the message (m_1, m'_2, m_3).
Solution: authenticate a *random message identifier* along with each block.

A general MAC from a fixed-length one

Potential attacks:

- Block re-ordering attack: the attacker changes the order of blocks, if (t_1, t_2) is a valid tag on (m_1, m_2) where $m_1 \neq m_2$, then (t_2, t_1) is a valid tag on (m_2, m_1) as m_2, m_1 is a different message!
  Solution: authenticate a block index with each block.
- Truncation attack: the attacker removes blocks from the end of the message and their corresponding blocks from the tag.
  Solution: authenticate the message length with each block
- Mix-and-match attack: the attacker has valid tags (t_1, t_2, t_3) and (t'_1, t'_2, t'_3) on the messages (m_1, m_2, m_3) and (m'_1, m'_2, m'_3). He outputs (t_1, t'_2, t_3) on the message (m_1, m'_2, m_3).
  Solution: authenticate a *random message identifier* along with each block.

A general MAC from a fixed-length one

Definition

Let $S_1 = (\mathsf{KeyGen}_1, \mathsf{Mac}_1, \mathsf{Verify}_1)$ *be a fixed-length MAC for messages of length* n*, we define a MAC* S *for arbitrary-length messages as follows:*

- $\mathsf{Mac}(k \in \{0,1\}^n, m \in \{0,1\}^*)$*:*
  - *it takes a key* k *and a messge* m*, where* $|m| = \ell < 2^{n/4}$*.*
  - *it then parses* m *into* d *blocks of length* $n/4$*, i.e.* $m_1, \cdots, m_d$*.*
  - *if the last block is not of size* $n/4$*, we pad it with* $0s$
  - *it uniformly chooses* $r \in \{0,1\}^{n/4}$
  - *For* $i = 1, \cdots, d$*, compute* $t_i \leftarrow \mathsf{Mac}_1(k, r||\ell||i||m_i)$*, where* i, ℓ *are encoded as strings of length* $n/4$*.*
  - *Output* $t = (r, t_1, \cdots, t_d)$*.*
- $\mathsf{Verify}(k, m, (r, t_1, \cdots, t_{d'}))$*: parse* m *into* d *blocks, then output 1 iff* $\mathsf{Verify}_1(k, r||\ell||i||m_i, t_i) = 1$ *for* $1 \le i \le d$ *and* $d' = d$*.*

A general MAC from a fixed-length one

Theorem

If S_1 is a secure fixed-length MAC for messages of length n, then S as defined above is a secure MAC for arbitrary-length messages.

Proof.

Exercise. hint: show that the aforementioned attacks are the only possible ones! □

Another way to build a secure MAC for arbitrary-length messages is to use hash functions, which will be covered soon!

Further Reading (1)

- N.J. Al Fardan and K.G. Paterson.
  Lucky thirteen: Breaking the TLS and DTLS record protocols.
  In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 526–540, May 2013.
- J Lawrence Carter and Mark N Wegman.
  Universal classes of hash functions.
  In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112. ACM, 1977.
- Jean Paul Degabriele and Kenneth G Paterson.
  On the (in) security of IPsec in MAC-then-Encrypt configurations.
  In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 493–504. ACM, 2010.

Further Reading (2)

- Ted Krovetz and Phillip Rogaway.
  The software performance of authenticated-encryption modes.
  In *Fast Software Encryption*, pages 306–327. Springer, 2011.
- Douglas R. Stinson.
  Universal hashing and authentication codes.
  Designs, Codes and Cryptography, 4(3):369–380, 1994.