CS579: Foundations of Cryptography

Spring 2023

Message Authentication Codes

Instructor: **Nikos Triandopoulos**

STEVENS INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

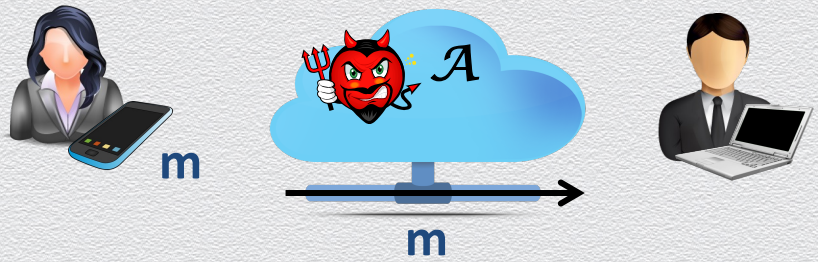
Security problems studied by modern cryptography

- Classical cryptography: **message encryption**
  - early cryptographic schemes were developed to provide **secrecy / confidentiality**
  - while in transit (or at rest), no message should be **leaked** to an outsider
- Modern cryptography: **wide variety** of security problems
  - secrecy is **<u>not the only</u>** **security concern** when using an unprotected channel
  - today we need / study a large set of **security properties** beyond secrecy
- The sibling of message encryption: **message authentication**
  - **<u>another cornerstone</u>** of any secure system aiming to provide **integrity**
  - while in transit (or at rest), no message should be **modified** by an outsider

Two cornerstone security properties

Common setting

- communication (storage) over an "**open**," i.e., **unprotected**, channel (medium)



Fundamental security problems

- while in transit (at rest)
  - no message (file) should be **leaked** to an outsider or unauthorized party
  - no message (file) should be **modified** by an outsider or unauthorized party

Core cryptographic protections

- **encryption schemes** provide **secrecy / confidentiality**
- **MAC schemes** provide **integrity / unforgeability**

Message authentication: Motivation

- Information has **value**
  - critical information must be protected not to leak to unauthorized parties
  - **message encryption** takes care of this
- What is, indeed, valuable is **correct** information
  - **incorrect** or **falsified** information may be of **little value**
  - **random** information may be **useless**
  - **maliciously crafted** information can be **harmful**
  - **message authentication** takes care of this

Example 1

- Secure electronic banking
  - a bank receives an electronic request to transfer $1,000 from user A to user B
  - concerns
    - did user A, indeed, order this money transfer, or **an attacker** (even user B)?
    - even if it was ordered by A, is the amount $1,000 authentic, i.e., the original amount intended to be transferred by A, or was it **modified** while in transit?
- Adversarial Vs. random message-transmission errors
  - standard error-correction are **not sufficient** to address these concerns

Example 2

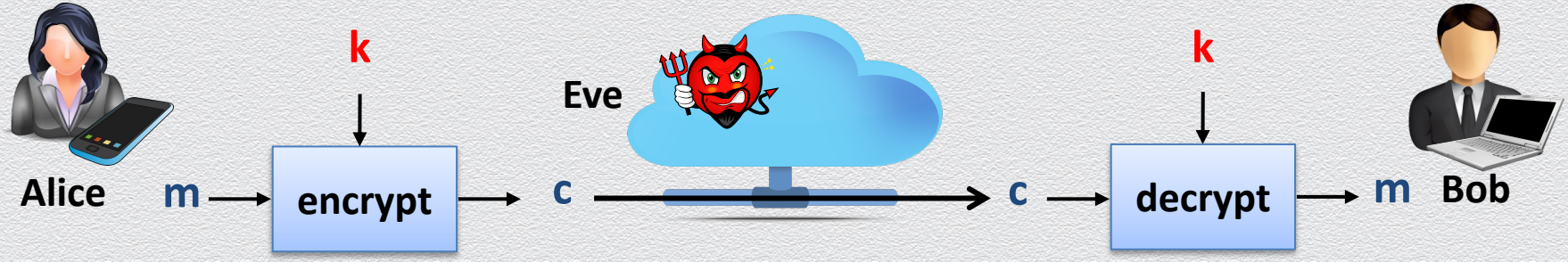
- Web browser cookies
  - a user is performing an online purchase (e.g., via Amazon)
  - HTTP traffic between client and server is **stateless**; so session-related info is stored at the client in the form of a "**cookie**", included in messages sent to server
  - cookies typically include client-specific information that affect the transaction (e.g., the user's shopping cart along with a discount due to a coupon)
  - concerns
    - was such state maliciously altered by the client (possibly harming the server)?
  - here, secrecy is irrelevant

Integrity of communications / computations

- Fundamental security property for any computing system
  - no unprotected system can be assumed to be trustworthy w.r.t.
    - origin of information (source) – e.g., impersonation, phishing, etc.
    - contents of information – e.g., man-in-the-middle attacks, email spam, etc.
  - specific protection measures must be taken to guarantee a system's integrity
  - prevention Vs. detection
    - adversarial tampering with information cannot be avoided altogether!
    - such tampering of messages / computations should be at least detectable
  - goal: prevent undetected tampering (thus, avoid "consuming" falsified information)

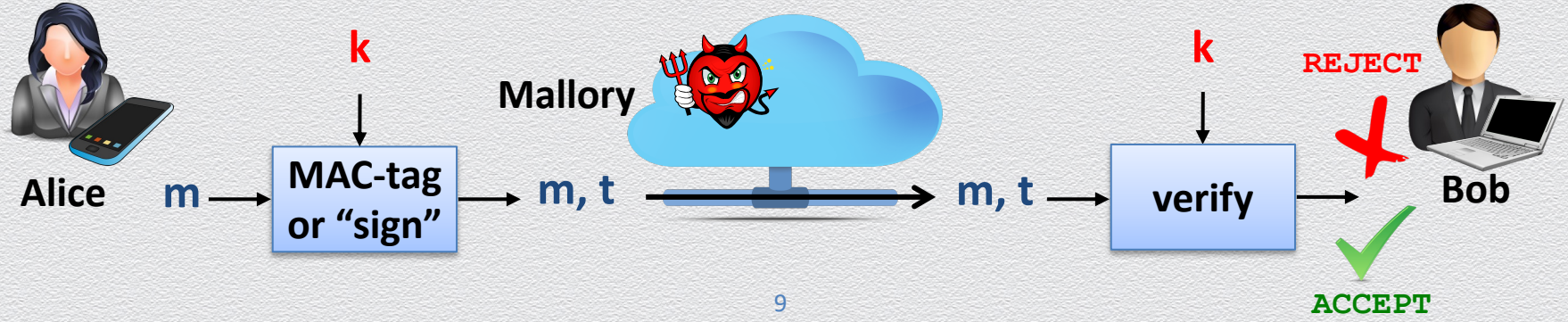
Recall: Symmetric-key message encryption

- Motivation: **Secret communication**
  - Alice wants to send a message m to Bob; Eve can eavesdrop sent messages
- Solution concept: **Symmetric-key encryption scheme**
  - Alice and Bob share a **secret key k** (selected uniformly at random)
  - Alice **encrypts** plaintext m to **ciphertext** c and sends c instead of m
  - Bob **decrypts** received c to original m; Eve "**cannot learn**" m from c



Symmetric-key message authentication

- Motivation: **Authenticated communication**
  - Alice wants to send a message m to Bob; Mallory can manipulate sent messages
- Solution concept: **Symmetric-key message authentication codes (MACs)**
  - Alice and Bob share a **secret key k** (selected uniformly at random)
  - Alice **annotates / "signs"** message m with **tag** t and sends (m, t)
  - Bob **verifies** authenticity of received m using t; Mallory "**cannot forge**" a message



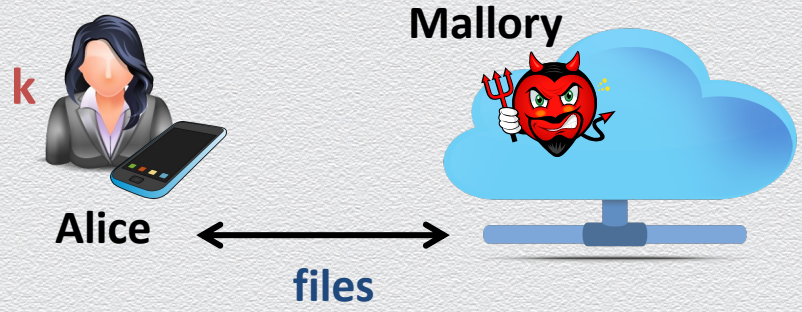
Main application areas

Secure communication

- **verify authenticity of messages** sent among parties
- assumption
  - Alice and Bob **securely generate, distribute and store shared key k**
  - attacker does not learn key k

Secure storage

- **verify authenticity of files** outsourced to the cloud
- assumption
  - Alice **securely generates and stores key k**
  - attacker does not learn key k



Encrypting does not mean authenticating

- Contrary to common misconception
  - encrypted traffic generally does not provide any integrity guarantees
  - incorrect reasoning / false intuition

"since ciphertext c hides message m contents, Mallory cannot meaningfully modify c or m"

- Why this is incorrect?
  - all encryption schemes (seen so far) are based on one-time pad, i.e., masking via XOR
  - consider flipping a single bit of ciphertext c; what happens to plaintext m?
  - such property of one-time pad does not contradict the secrecy definitions
  - generally, secrecy and integrity are distinct properties

Two distinct properties

Secrecy

- **sensitive** information has value
  - if **leaked**, it can be **risky**
- **specific** scope / **general** semantics
- **prevention**
- does **not** imply integrity
  - e.g., bit-flipping "attack"

Integrity

- **correct** information has value
  - if **manipulated**, it can **harmful**
  - **random Vs. adversarial** manipulation
- **wider** scope / **context-specific** semantics
  - source Vs. content authentication
  - replay attacks
- **detection**
- does **not** imply secrecy
  - e.g., user knows cookies' "contents"

Symmetric-key message authentication code (MAC)

Abstract crypto primitive

- defined by triplet of PPT algorithms **(Gen, Mac, Vrfy)**, security parameter 1^n
  - Gen: prob. alg., on input 1^n, outputs a key k from key space $\mathcal{K}$
  - Mac: prob. alg., on input message $m \in \{0,1\}^*$ and key k, outputs tag t, $\mathbf{Mac_k(m) \rightarrow t}$
  - Vrfy: det. alg., on input a pair (m, t) and key k, outputs a bit b, $\mathbf{b := Vrfy_k(m,t)}$
- satisfying desired properties
  - **efficiency**: key generation & tag computation/verification "are fast"
  - **correctness**: for all m, k it holds that $Vrfy_k(m, Mac_k(m))) = 1$
  - **security**: one "cannot forge" a verifiable pair (m, t)

Conventions

Similarly to encryption schemes

- fixed-length MAC for messages of length l(n)
  - a MAC defined over message space $\{0,1\}^{l(n)}$ for a given function $l()$
- key selection
  - typically, Gen(1^n) selects key k **uniformly at random** from the key space $\mathcal{K}$
  - then, we can omit Gen, so that a MAC is described as a pair (Mac,Vrfy)

Moreover

- canonical verification
  - when Mac is deterministic, Vrfy typically amounts to re-computing the tag t
  - $\text{Vrfy}_k(m,t)$: 1. $t' := \text{Mac}_k(m)$ 2. if $t = t'$, output 1 else output 0
- still, one should be able to conceptually distinguish between
  - authenticating m (i.e., computing t) Vs. verifying authenticity of m (i.e., running Vrfy)

Security of MACs

Setting: Alice wants to send a message m to Bob; Mallory can manipulate sent messages

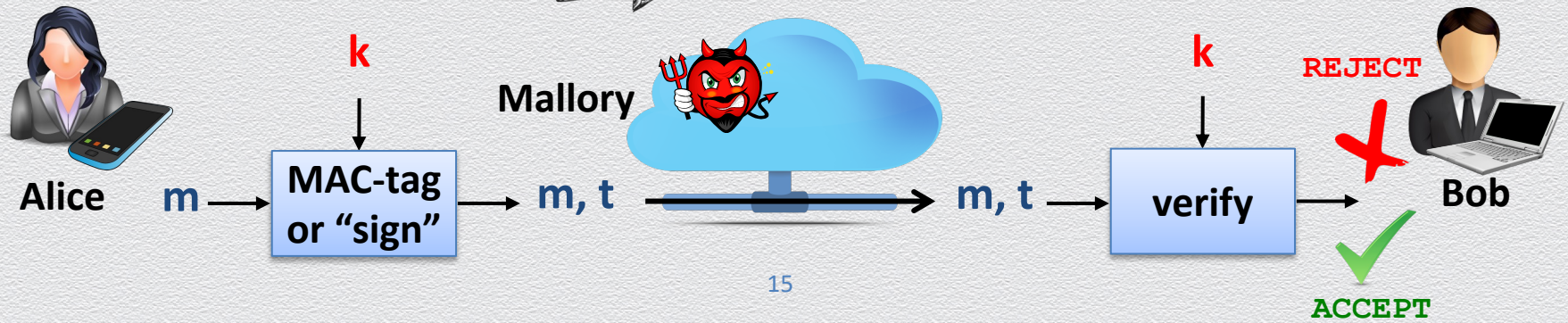
Secret-key: Alice and Bob share a **secret key k** (selected uniformly at random)

MAC functionality & security:

- Alice **annotates / "signs"** message m with **tag** t and sends (m, t)
- Bob **verifies** authenticity of received m using t
- Mallory "**cannot forge**" a verifiable message-tag pair

what it means to "break" a MAC

who is Mallory?



Security model – intuition

In practice, the real-life adversary

- **observes** a traffic of **authenticated** (and successfully verified) **messages**
- is often **able to partially influence** this traffic
  - e.g., with its purchasing behavior in the example of cookies representing shopping carts
- has as goal to **forge a verifiable message**
  - typically by inserting a "**new**" message m^*, t^* into the traffic so that pair (m^*, t^*) is **verifiable**
  - if m* equals any **previously observed** message, its goal is **trivial** to achieve
    - typically known as a **replay attack**; it can be harmful in real life but will be excluded now
- is capable of launching a **brute-force attack** (given that $Mac_k(m) \rightarrow t$ is **publicly known**)
  - given any observed pair (m, t), **exhaustively search key space $\mathcal{K}$** to find the used key k

Threat model – capturing real-world attackers

Mallory is an adversary $\mathcal{A}$ that is

- "active" (on the wire)
  - allow $\mathcal{A}$ to **observe** and/or **manipulate** sent messages
- "well-informed" (w.r.t. its intelligence) ⇒ via access to oracle $\mathrm{Mac}_k()$
  - allow $\mathcal{A}$ to **request MAC tags** of messages of **its choice** (i.e., to fully control the traffic)
- "replay-attack safe" (w.r.t. its goal)
  - restrict A to target to **forge only new** messages (i.e., not present in the traffic)
- "PPT" (w.r.t. its power)
  - restrict $\mathcal{A}$ to be **computationally bounded** (thus, capturing computational security)
  - **new messages may be forged undetectably** but they can be found
    - only with **negligible** probability or after an **exponentially** large computation

Game-based definition of computational MAC security

Game Mac-forge$_{\mathcal{A},\Pi}$(n) = 1 iff 1. $\mathrm{Vrfy}_k(m^*,t^*) = 1$ &
2. m^* not in $\mathcal{Q}$

MAC scheme
Π = (Gen, Mac, Vrfy)

security parameter 1^n

Π $\mathcal{T}$

$\mathrm{Gen}(1^n) \rightarrow k$

$\mathrm{Mac}_k(m_i) \rightarrow t_i$

m_1

t_1

m_2

t_2

…

m^*,t^*

$\mathcal{A}^{\mathrm{Mac}(k,\ )}$

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

or just **secure**

Security definition: We say that Π is **existentially unforgeable under an adaptive chosen-message attack** if for all PPT A, there exists a negligible function negl so that

$$\Pr[\ \text{Mac-forge}_{\mathcal{A},\Pi}(n) = 1\ ] \leq \mathrm{negl}(n)$$

Notes on security definition

- Is it a rather strong security definition?
  - allowing A to **query MAC tags for any message**
    - real-world senders will authenticate only “meaningful” messages
  - allowing A to break the scheme by **forging any new message**
    - similarly, a breach of security should only consider “meaningful” forged messages
- It is the right approach…
  - message “**meaningfulness” depends on higher-level application**
    - text messaging apps require authentication of English-text messages
    - but other apps may require authentication of binary tranfered files...
    - security definition should better be **agnostic** of the specific higher application

Notes on security definition (II)

- Are reply attacks important in practice?
  - absolutely yes! they constitute a very realistic and serious threat in practice
  - e.g., integrity of e-banking transactions: what if a money transfer is “replayed”?
  - whether replayed messages are valid or safe depends on higher app
- Still, it’s preferable to consider a “replay-attack unsafe” security definition
  - better not to assume any semantics regarding the high-level app, but instead delegate the validity or safety check to this app that “consumes” the messages
- Eliminating replay attacks
  - use of counters (i.e., common shared state) between sender & receiver
  - use of timestamps along with a (relaxed) authentication window for validation

Notes on security definition (III)

Secure MACs

- adversary $\mathcal{A}$ cannot forge tags on new messages (i.e., never previously authenticated)
- doesn't rule out that $\mathcal{A}$ may be able to forge a new tag on a previously auth. message

Strong MACs

- adversary $\mathcal{A}$ is allowed to win even if it forges a "new" tag on an "old" message
- game Mac-forge$_{\mathcal{A},\Pi}$(n) can be modified to Mac-sforge$_{\mathcal{A},\Pi}$(n)
  - list $\mathcal{Q}$ contains pairs of messages and their corresponding tag
  - the attack pair (m^*,t^*) is then required not to be included in $\mathcal{Q}$

Proposition

- any secure MAC with canonical verification is also a strong MAC

Notes on security definition (IV)

Verification queries

- adversary $\mathcal{A}$ has only access to a message-authentication oracle
- what if $\mathcal{A}$ is also given access to a message-verification oracle?
- this would, indeed, be the right way to define security (since the extension is realistic!)

Canonical verification

- tagging-only queries w/ canonical Vrfy() is equivalent to this extended threat model

Strong MACs

- also this stronger property is preserved in the extended threat model above

Overall: Symmetric-key MAC

Triple of PPT algorithms **(Gen, Mac, Vrfy)**, security parameter 1^n

- Gen: prob. alg., on input 1^n, **uniformly at random** selects $k \in \mathcal{K}$
- Mac: prob. alg., on input $m \in \{0,1\}^*$ and k, outputs **tag** $\mathbf{Mac_k(m) \rightarrow t}$
  - fixed-length MAC for messages of length $l(n)$
- Vrfy: det. alg., on input a pair (m, t) and k, outputs $\mathbf{b := Vrfy_k(m,t) \in \{acc, rej\}}$
  - **canonical**: accept only if can recompute claimed tag – "**re-compute & compare**"
- **efficiency, correctness** (for all m, k: $Vrfy_k(m, Mac_k(m))) = 1$)
- **security**: one "cannot forge" a verifiable pair (m, t)
  - PPT $\mathcal{A}$ has **oracle access** to $Mac_k()$ but can't have any **new** message accepted
  - replay attacks are **not** modeled (to be treated by higher-level app)

Recall: Computational MAC security

Game Mac-forge$_{\mathcal{A},\Pi}$(n) = 1 iff 1. $\mathrm{Vrfy}_k(m^*,t^*) = 1$ &
2. m^* not in $\mathcal{Q}$

MAC scheme
Π = (Gen, Mac, Vrfy)

security parameter 1^n

Π $\mathcal{T}$

$\mathrm{Gen}(1^n) \rightarrow k$

$\mathrm{Mac}_k(m_i) \rightarrow t_i$

m_1

t_1

m_2

t_2

…

m^*,t^*

$\mathcal{A}^{\mathrm{Mac}(k,\ )}$

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

We say that Π is **secure** if for all PPT $\mathcal{A}$, there exists a negligible function negl so that

$$\Pr[\ \text{Mac-forge}_{\mathcal{A},\Pi}(n) = 1\ ] \leq \mathrm{negl}(n)$$

Recall: Computational MAC security

Game Mac-forge$_{\mathcal{A},\Pi}$(n) = 1 iff 1. $Vrfy_k(m^*,t^*) = 1$ &
2. $(m^*,t^*) \notin \mathcal{Q}$

MAC scheme
Π = (Gen, Mac, Vrfy)

security parameter 1^n

Π $\mathcal{T}$

$Gen(1^n) \rightarrow k$

$Mac_k(m_i) \rightarrow t_i$

m_1

t_1

m_2

t_2

…

m^*,t^*

$\mathcal{A}^{Mac(k,\ )}$

$\mathcal{Q} = (m_1,t_1), (m_2,t_2)$ …

We say that Π is **strongly secure** if for all PPT $\mathcal{A}$, there exists a negligible function negl such that

$Pr[Mac\text{-}sforge_{\mathcal{A},\Pi}(n) = 1] \leq negl(n)$

Strong MAC

Game Mac-sforge$_{\mathcal{A},\Pi}$(n) = 1 iff 1. $\mathrm{Vrfy}_k(m^*,t^*) = 1$ &
2. (m^*,t^*) not in $\mathcal{Q}$

MAC scheme
Π = (Gen, Mac, Vrfy)

security parameter 1^n

Π $\mathcal{T}$

$\mathrm{Gen}(1^n) \rightarrow k$

$\mathrm{Mac}_k(m_i) \rightarrow t_i$

m_1

t_1

m_2

t_2

…

m^*,t^*

$\mathcal{A}^{\mathrm{Mac}(k,\ )}$

$\mathcal{Q} = (m_1,t_1), (m_2,t_2)$ …

We say that Π is **strongly secure** if for all PPT $\mathcal{A}$, there exists a negligible function negl so that

$$\Pr[\ \text{Mac-sforge}_{\mathcal{A},\Pi}(n) = 1\ ] \leq \mathrm{negl}(n)$$

(Strong) MAC w/ verification queries

Game Mac-s<u>V</u>forge$_{\mathcal{A},\Pi}$(n) = 1 iff 1. $\mathrm{Vrfy}_k(m^*,t^*) = 1$ &
2. (m^*,t^*) not in $\mathcal{Q}$

MAC scheme
Π = (Gen, Mac, Vrfy)

security parameter 1^n

Π $\mathcal{T}$

$\mathrm{Gen}(1^n) \rightarrow k$

$\mathrm{Mac}_k(m_i) \rightarrow t_i$

m_1 or (m_1, t_1)

t_1 or acc/rej

m_2 or (m_2, t_2)

t_2 or acc/rej

…

m^*,t^*

$\mathcal{A}^{\mathrm{Mac}(k,\ ),\mathrm{Vrfy}(k,)}$

$\mathcal{Q} = (m_1,t_1), (m_2,t_2) \ldots$

We say that Π is **strongly V-secure** if for all PPT $\mathcal{A}$, there exists a negligible function negl so that

$$\Pr[\ \text{Mac-s}\underline{\mathbf{V}}\text{forge}_{\mathcal{A},\Pi}(n) = 1\ ] \leq \mathrm{negl}(n)$$

Verification queries Vs. timing attacks on MAC verification

In game Mac-s<u>V</u>forge$_{\mathcal{A},\Pi}$(n)

- queries to oracle Verf$_k$() return `acc/rej` (i.e., a **single bit**)

In real life

- implicit tag verification is **<u>feasible</u>** (e.g., by detecting a difference in verifier's behavior)
- but also an attacker may receive **<u>more than this 1-bit info</u>** via other "**side channels**"
  - Vrfy(m,t) of a **canonical** MAC returns `acc` only if t = Mac$_k$(m)
  - if implemented using `strcmp`, then comparison occurs **<u>byte by byte until first mismatch</u>**
  - thus, the time to return `rej` **<u>depends on position</u>** of the first unequal byte
  - i.e., that attacker may also receive **timing-related information**

Side-channel attack via tag verification w/ timing

- attacker $\mathcal{A}$ wishes to forge a **verifiable s-byte tag t** for target message m
  - assume that $\mathcal{A}$ knows t_i, i.e., the first i bytes of tag t (for some i = 0, 1, …, s-1)
  - for j = 0, …, 255
    - send verification query (m, t_j) where $\mathbf{t_j = t_i \,||\, j \,||\, (00)^{s-i-1}}$
    - get response res_j (probably `rej`) and **measure time**$_j$ spent for computation of res_j
  - if $time_{j*}$ is the **maximum measured response time**, then set $\mathbf{t_{i+1} = t_i \,||\, j^*}$
- **realistic** attack
  - forged code updates in Xbox 360 to load pirated games into the hardware
  - exploited differences of 2.2msec between rejection times!
  - 4096 queries are needed to recover a 16-byte tag!

Generic MAC constructions

Recall: Symmetric-key MAC

PPT algorithms **(Gen, Mac, Vrfy)**, security parameter 1^n

- Gen(1^n): **uniformly at random** selects $k \in \mathcal{K}$
- Mac_k: on input $m \in \{0,1\}^*$, outputs **tag** $\mathbf{Mac_k(m) \rightarrow t}$
  - fixed-length MAC for messages of length $l(n)$
- Vrfy_k: on input a pair (m, t), outputs $\mathbf{b := Vrfy_k(m,t) \in \{acc, rej\}}$
  - **canonical:** accept iff claimed tag can be re-computed

Properties

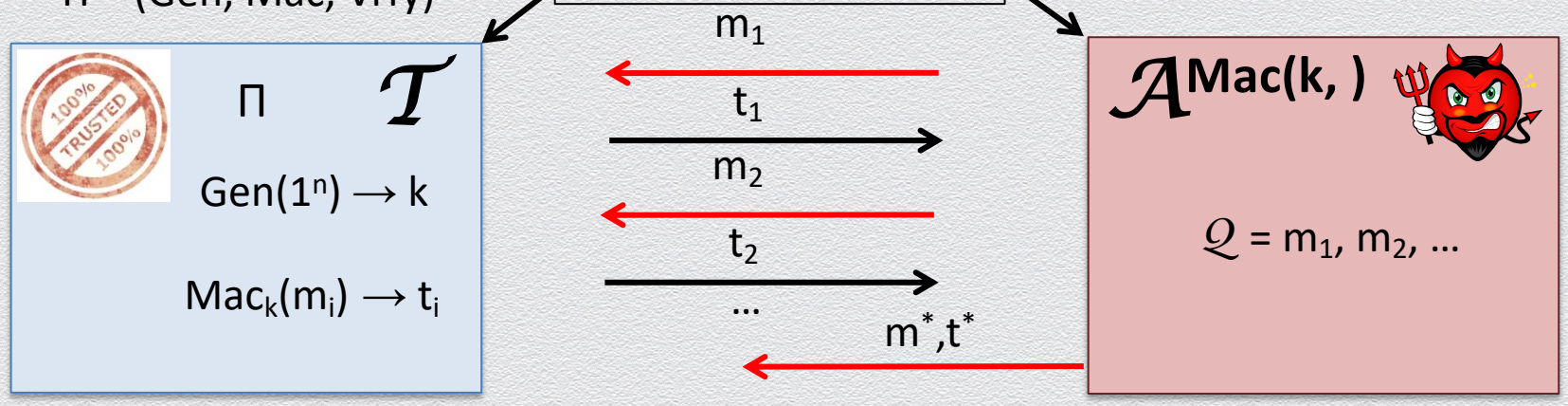
- **efficiency, correctness** (for all m, k: $\text{Vrfy}_k(m, \text{Mac}_k(m))) = 1$)
- **security**: no PPT $\mathcal{A}$ with oracle access to $\text{Mac}_k()$ can have any new message accepted

Recall: MAC security

Game Mac-forge$_{\mathcal{A},\Pi}$(n) = 1 iff 1. $\mathrm{Vrfy}_k(m^*,t^*) = 1$ &
2. m^* not in $\mathcal{Q}$

MAC scheme
$\Pi = (\mathrm{Gen}, \mathrm{Mac}, \mathrm{Vrfy})$

security parameter 1^n



We say that Π is **secure** if for all PPT $\mathcal{A}$, there exists a negligible function negl so that

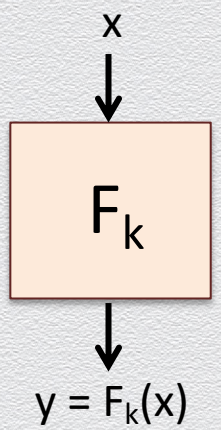
$$\Pr[\ \text{Mac-forge}_{\mathcal{A},\Pi}(n) = 1\ ] \leq \mathrm{negl}(n)$$

Three generic MAC constructions

- fixed-length MAC
  - direct application of a PRF for tagging
  - limited applicability
- domain extension for MACs
  - straightforward secure extension of fix-length MAC
  - inefficient
- CBC-MAC
  - resembles CBC-mode encryption
  - efficient

1. Fixed-length MAC

- based on use of a PRF
  - employ a PRF F_k in the obvious way to compute and canonically verify tags
  - set tag t to be the pseudorandom string derived by evaluating F_k on message m
- secure, provided that F_k is a secure PRF
  - security proof by reduction
    - "MAC security is reduced to PRF security"
    - in class



MAC scheme Π

$Gen(1^n)$: $\{0,1\}^n \rightarrow k$

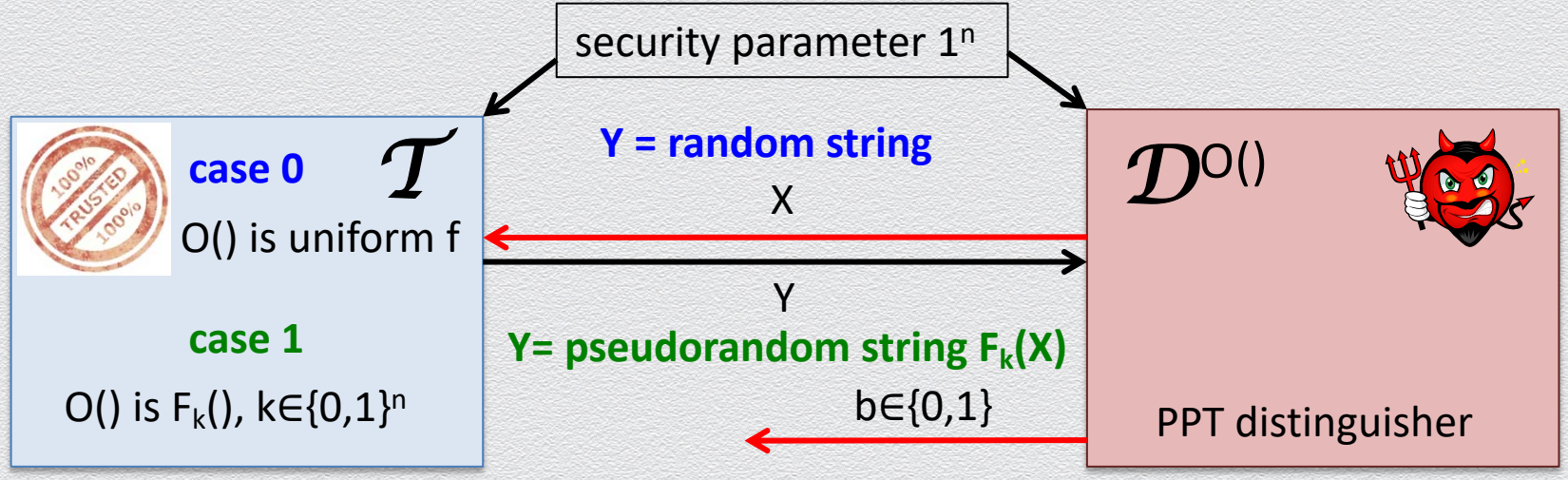
$Mac_k(m)$: set $t = F_k(m)$

$Vrfy_k(m,t)$: return 1 iff $t = F_k(m)$

Recall: PRF – security

b = 0 when $\mathcal{D}$ thinks that its oracle is f()

b = 1 when $\mathcal{D}$ thinks that its oracle is $F_k()$

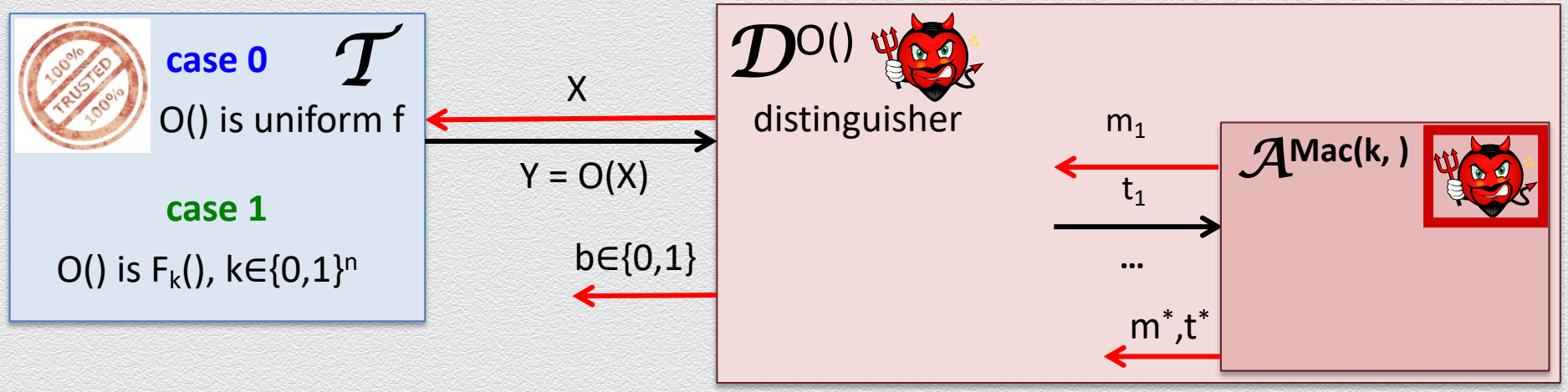


$$| \Pr[\mathcal{D}^{F(k,)}(1^n) = 1] - \Pr[\mathcal{D}^{f()}(1^n) = 1] | \leq negl(n)$$

$\mathcal{D}$ behaves the same no matter what its oracle is!

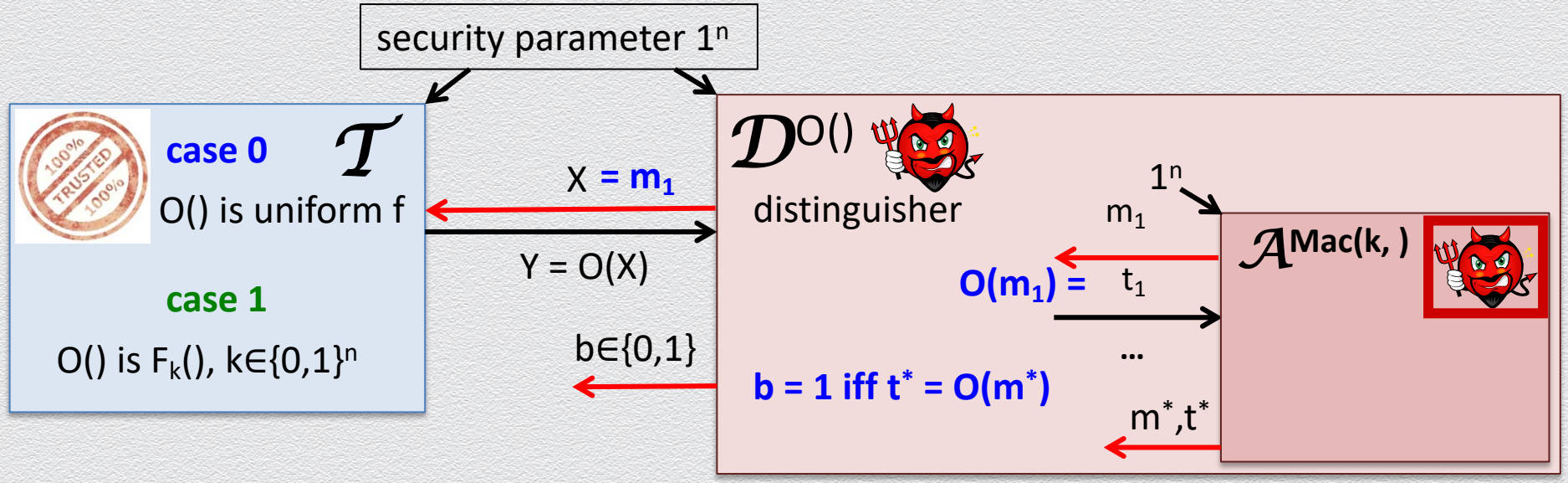
Proof (1)

- suppose that MAC scheme is not secure; then ∃ winning $\mathcal{A}$ for Mac-forge$_{\mathcal{A},\Pi}$(n)
- Let $\mathcal{A}$ be any PPT Mac-forger
- $\mathcal{A}$ can be used to construct a winning distinguisher $\mathcal{D}$ against PRF F_k



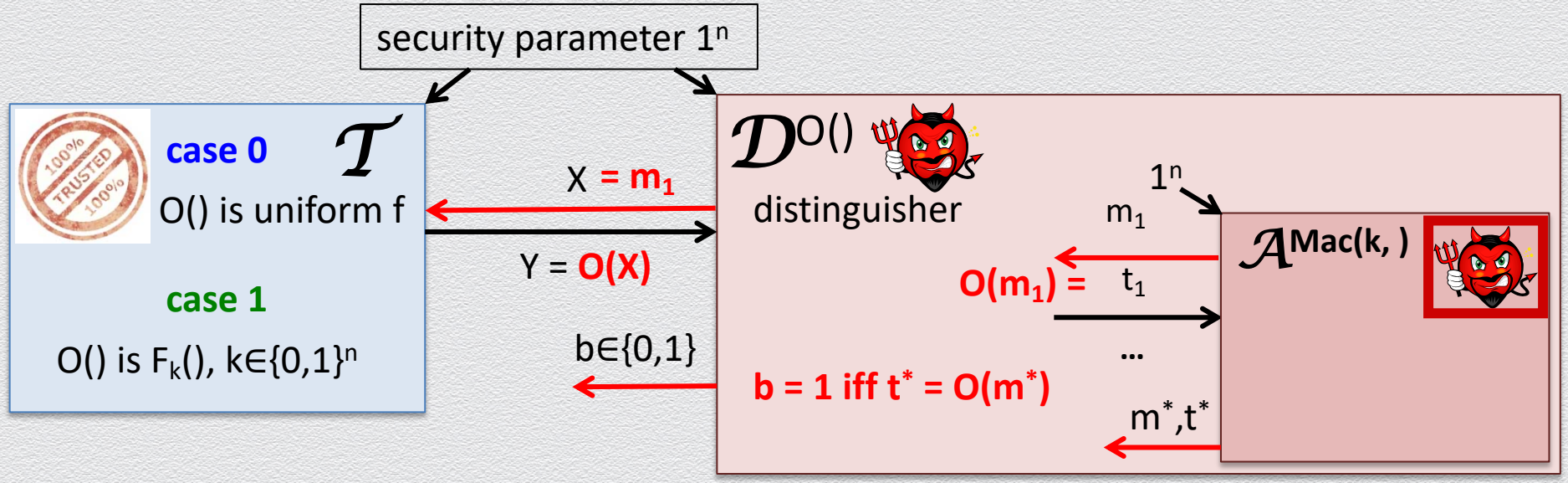
Proof (2)

- reduction $\mathcal{D}$ operates in the standard way: it answers $\mathcal{A}$'s Mac queries using oracle O()
- oracle O() initiates to case 0 or 1; $Mac_k()$ query on m_i is merely answered as $O(m_i)$
- $Vrfy_k()$ canonically checks whether t^* is consistent with oracle O()



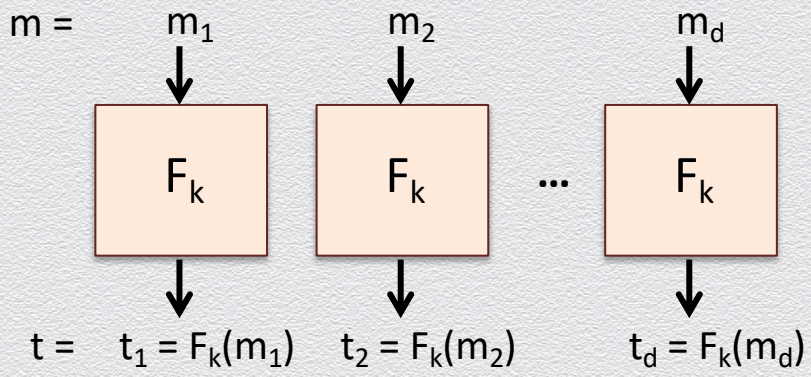
Proof (3)

- consider MAC scheme Π', setting tags as $t = f(m)$ instead of $t = F_k(m)$
- by reduction: $a = \Pr[\mathcal{D}^{f()} = 1] = \Pr[\mathcal{A}$ breaks $\Pi'] = 2^{-n}$; $b = \Pr[\mathcal{D}^{F(k,)} = 1] = \Pr[\mathcal{A}$ breaks $\Pi]$
- since F_k is a PRF, $|a - b| \leq negl(n)$, thus $b = \Pr[\mathcal{A}$ breaks $\Pi] \leq negl'(n)$



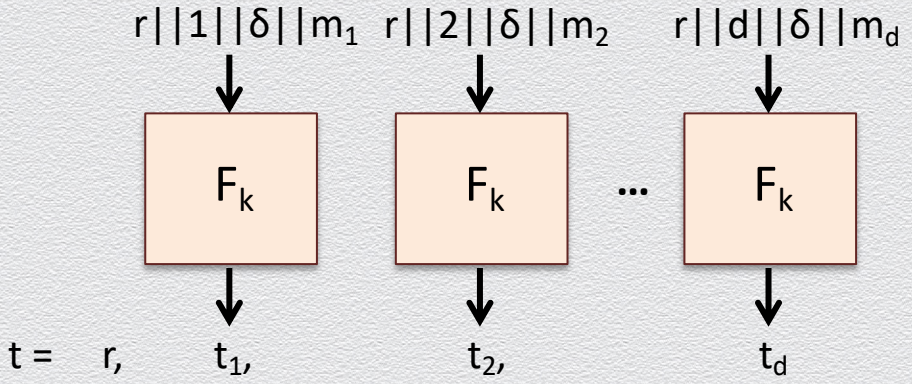
2. Domain extension for MACs

- suppose we have the previous fix-length MAC scheme Π
- how can we use it to authenticate a message m of arbitrary length?
- naïve approach
  - pad m and view it as d blocks $m_1, m_2, \ldots, m_d$
  - separately authenticate block m_i via Π
- security issues
  - reordering attack; verify block index, $t = F_k(m_i||i)$
  - truncation attack; verify message length $\delta = |m|$, $t = F_k(m_i||i||\delta)$
  - mix-and-match attack; randomize tags (using message-specific fresh nonce)

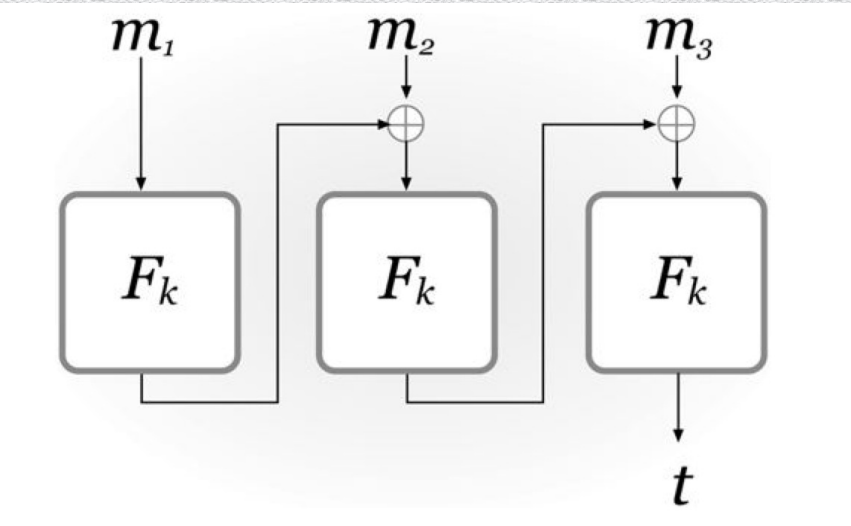


2. Domain extension for MACs – final scheme

- assumes a MAC scheme Π for messages of size n
- set tag of message m of size $\delta = |m|$ at most $2^{n/4}$ as follows
  - choose fresh random r of size n/4; view m as d blocks of size n/4 each
  - separately apply Π on each block, authenticating also its index and m's length δ
- security
  - extension is secure, provided that F_k is a secure PRF



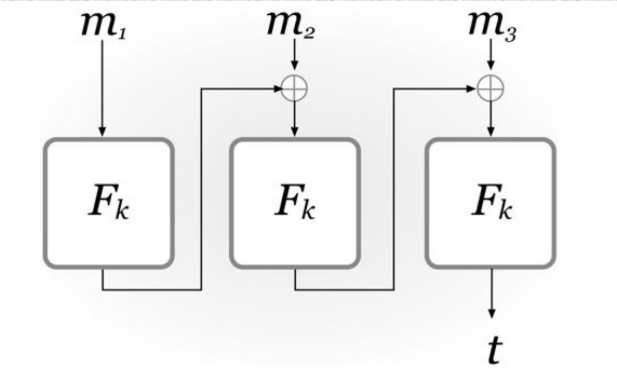
3. CBC-MAC



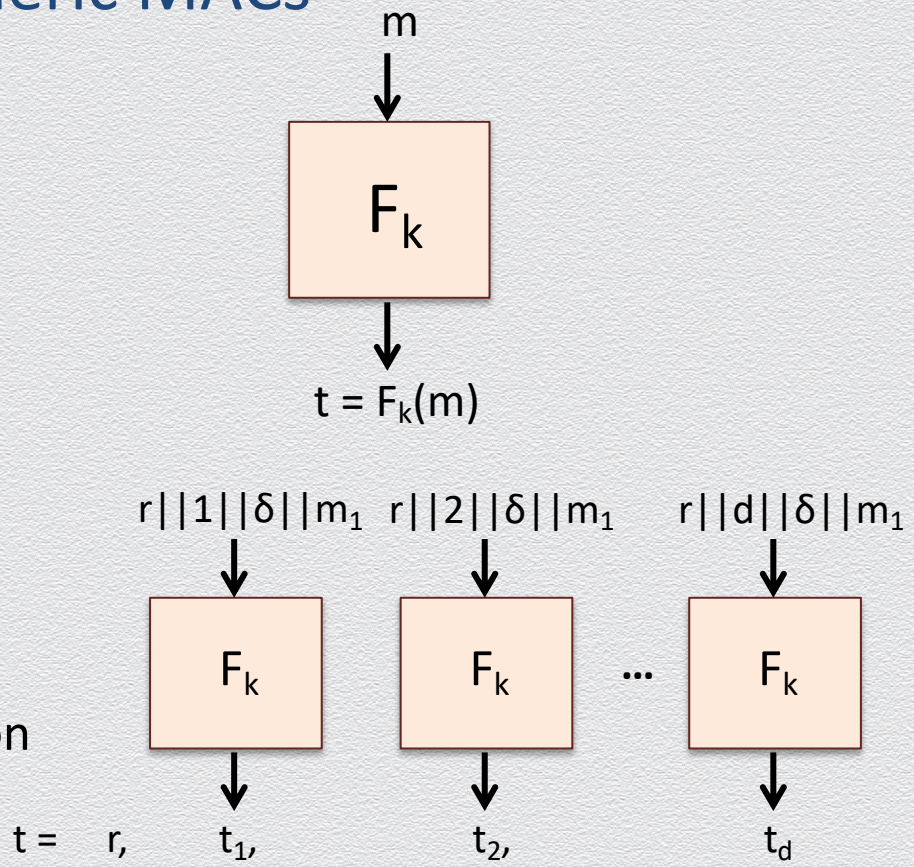
- employs a PRF in a similar manner as in CBC-mode encryption
- security
  - secure only when **fixed-length** messages are authenticated, provided that F_k is a secure PRF
  - messages of length equal to any multiple of n can be authenticated
    - but this length need be fixed in advance
  - insecure, otherwise

3. CBC-MAC Vs. previous generic MACs

- can authenticate longer messages than basic PRF-based scheme (1)



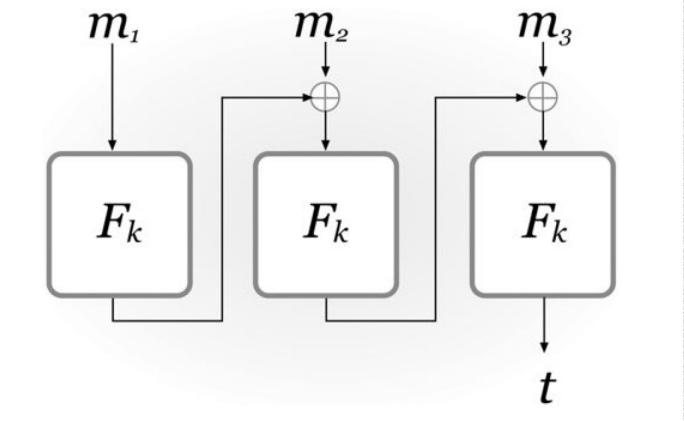
- more efficient than domain-extension MAC scheme (2)



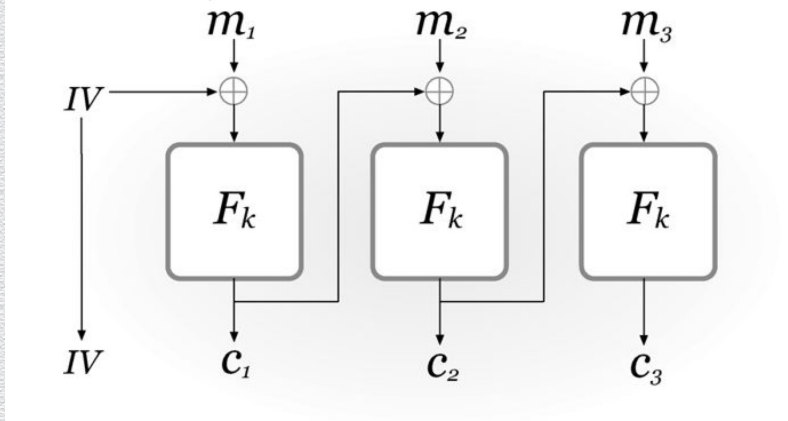
3. CBC-MAC Vs. CBC-mode encryption

- crucially for their security
  - CBC-MAC uses no IV (or uses an IV set to 0) and only the last PRF output
  - CBC-mode encryption uses a random IV and all PRF outputs
  - “simple”, innocent modification can be catastrophic…

CBC-MAC

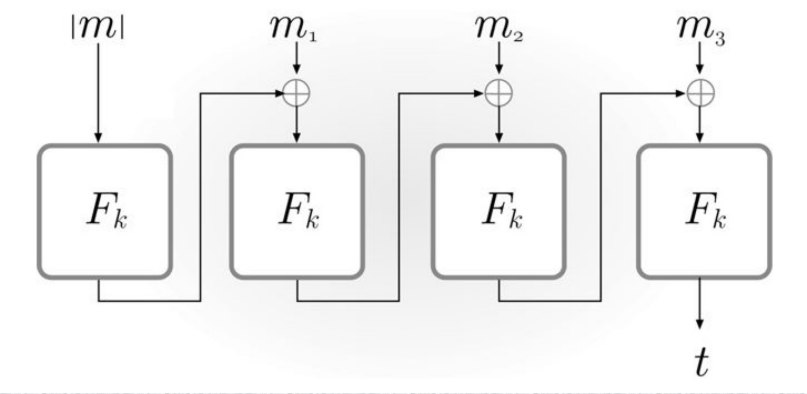


CBC-mode encryption



3. Authenticating arbitrary message lengths

- CBC-MAC can be extended in two ways to support arbitrary-length messages
- apply CBC-MAC on m, **prepended with its length |m|** (encoded as n-bit string)
  - appending |m| at the end is not secure
  - needs to know |m| in advance
- apply CBC-MAC **twice using independent keys**
  - one on the message m itself, using the **first key**, to compute an **initial** tag t'
  - one on the initial tag t', using the **second key**, to compute the **final tag** t for m
  - needs to manage two keys (undesirable, even if derived on the fly, e.g., as $F_k(1)$, $F_k(2)$)



Authenticated encryption

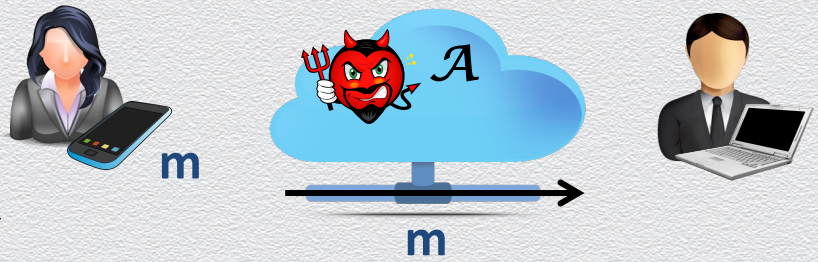
Authenticated encryption: Catching two birds at once

Common setting

- communication (storage) over an "**open**," i.e., **unprotected**, channel (medium)

Fundamental security problems

- while in transit (at rest)
  - no message (file) should be **leaked** to $\mathcal{A}$
  - no message (file) should be **modified** by $\mathcal{A}$



Core cryptographic protections

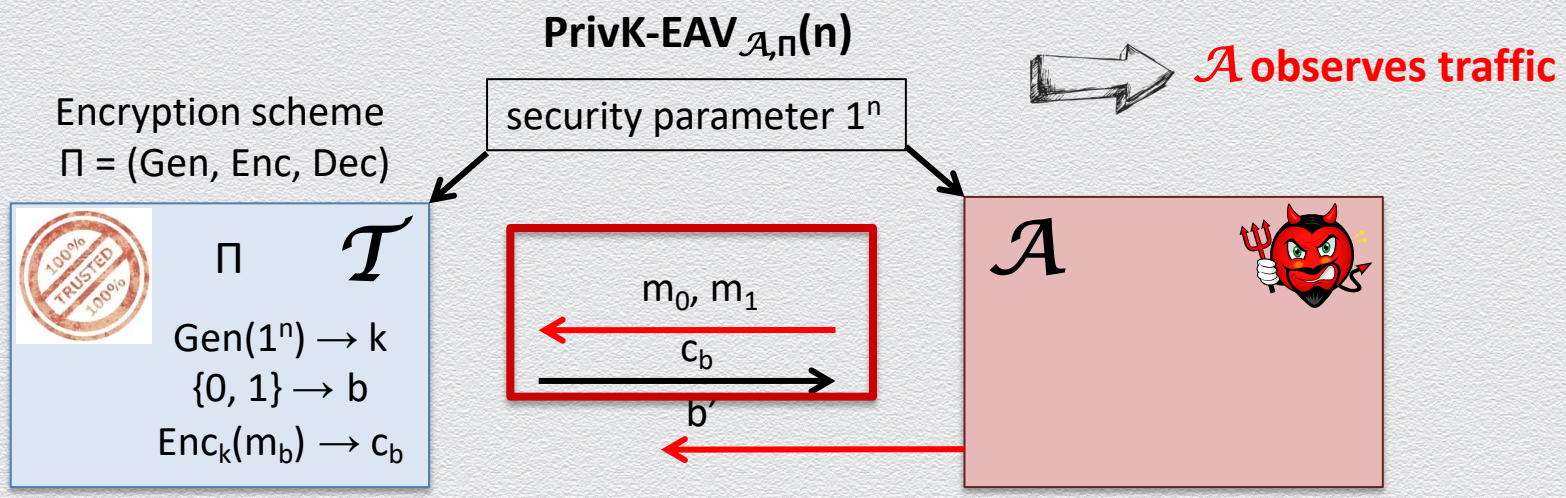
- **encryption schemes** provide **secrecy / confidentiality**
- **MAC schemes** provide **integrity / unforgeability**

Can we achieve both at once in the symmetric-key setting? Yes…

Authenticated encryption: goal, justification, challenges

- goal: crypto scheme that realizes an "**ideally secure**" communication channel
- motivation: important in practice
  - real apps often **need both**
  - also a **good security hygiene**: even if a given app "asks" only/more for secrecy or integrity, it's always better **to achieve both**!
- threat model: active adversary
  - attacker will try to **learn** new information about any of the sent messages – **passive**
  - attacker will try to **undetectably alter** any of the sent messages – **active**
  - this conceptual **contradiction** creates the need for a **new** definitional model
- subtleties: just "putting together" an encryption scheme and a MAC won't do the job…

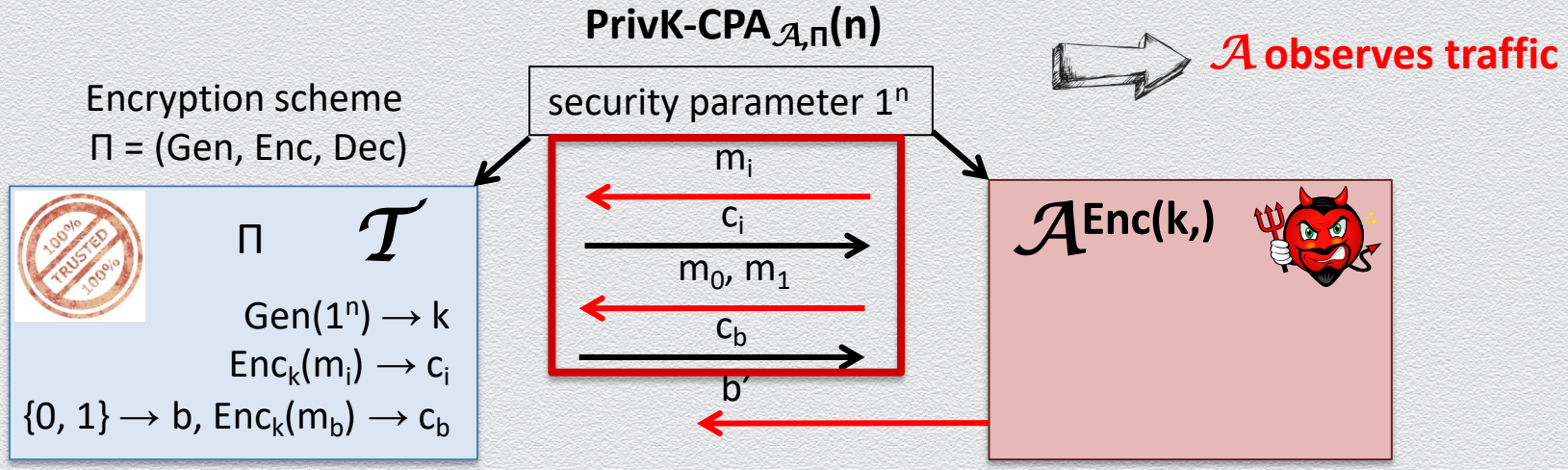
Recall: EAV-security for secrecy



Π = (Enc,Dec) is **EAV-secure**, if ∀PPT adversaries $\mathcal{A}$, ∃a negl. function ε(n) s. t.

$$\Pr[\text{PrivK-EAV}_{\mathcal{A},\Pi}(n)=1] \leq \tfrac{1}{2} + \varepsilon(n)$$

Recall: CPA-security for secrecy



Π = (Enc,Dec) is **CPA-secure**, if ∀PPT adversaries $\mathcal{A}$, ∃a negl. function ε(n) s. t.

$$\Pr[\text{PrivK-CPA}_{\mathcal{A},\Pi}(n)=1] \leq \tfrac{1}{2} + \varepsilon(n)$$

Recall: MAC security

Mac-forge$_{\mathcal{A},\Pi}$(n) = 1 iff 1. $\mathrm{Vrfy}_k(m^*,t^*) = 1$ &
2. m^* not in $\mathcal{Q}$

MAC scheme
$\Pi = (\mathrm{Gen}, \mathrm{Mac}, \mathrm{Vrfy})$

security parameter 1^n

Π $\mathcal{T}$

$\mathrm{Gen}(1^n) \rightarrow k$

$\mathrm{Mac}_k(m_i) \rightarrow t_i$

m_1

t_1

m_2

t_2

…

m^*,t^*

$\mathcal{A}$ observes traffic

$\mathcal{A}^{\mathrm{Mac}(k,\ )}$

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

$\mathcal{A}$ manipulates traffic

Π is **secure** if for all PPT $\mathcal{A}$, there exists a negligible function $\varepsilon(n)$ s.t.

$$\Pr[\ \text{Mac-forge}_{\mathcal{A},\Pi}(n) = 1\ ] \leq \varepsilon(n)$$

Authenticated encryption: Definitional issues

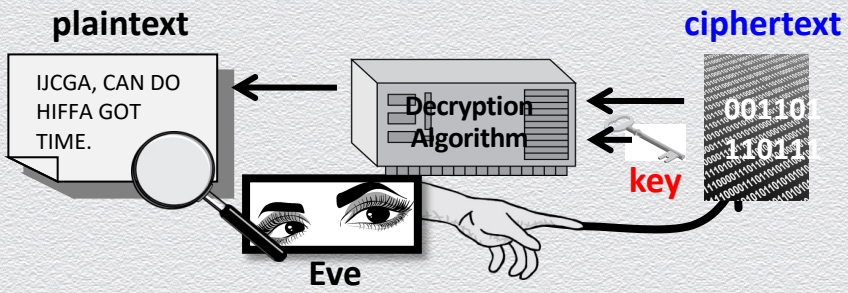
- need to capture two security properties: secrecy & integrity
  - simply, define each property "separately"
- **secrecy**, Π_E = (Gen, Enc, Dec)
  - Π needs to satisfy the strongest security notion of **CPA-security**
  - attacker capable of launching **chosen-ciphertext attacks (CCA)**
    - also has access to **oracle $\mathbf{Dec_k()}$**, receiving **decrypted messages** or **decryption failures**
  - imposes stringent requirements: CCA-security implies **non-malleability** of encryption
    - manipulating a valid ciphertext should either **invalidate** its decryption or produce a "random" plaintext that is **unrelated to the original plaintext**
- **integrity**, Π_M = (Gen, Mac, Vrfy)
  - standard security: existential unforgeability under an adaptive chosen-message attack
- but Π_{Enc} and Π_{MAC} do not have the same syntax

Recall: Possible eavesdropping attacks

An attacker may possess a

- (a) collection of ciphertexts
  - ciphertext only attack
- (b) collection of plaintext/ciphertext pairs
  - known plaintext attack
- (c) collection of plaintext/ciphertext pairs for plaintexts selected by the attacker
  - chosen plaintext attack
- (d) collection of plaintext/ciphertext pairs for (plaintexts and) ciphertexts selected by the attacker
  - chosen ciphertext attack

(d)

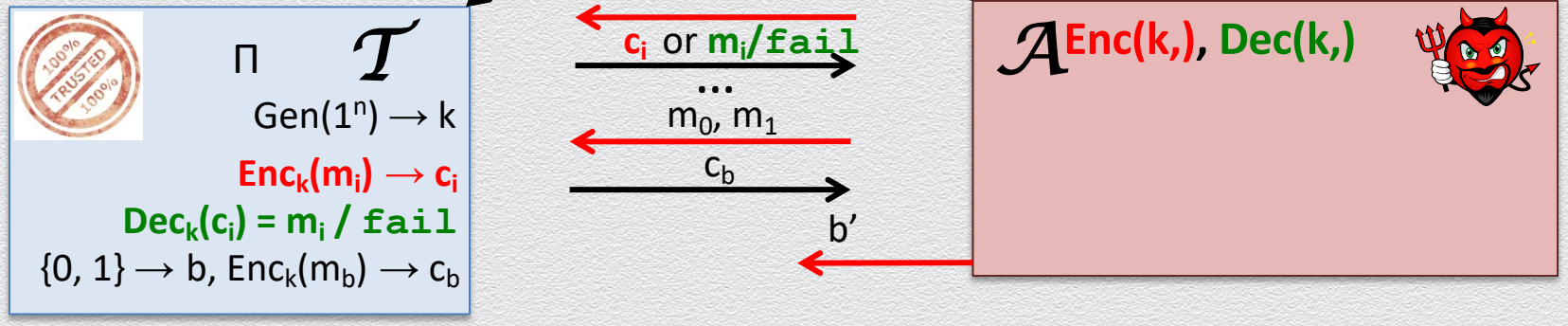


CCA-security for secrecy

PrivK-CCA$_{\mathcal{A},\Pi}$(n) = 1 iff b' = b

Encryption scheme
Π = (Gen, Enc, Dec)

security parameter 1^n



Π = (Enc,Dec) is **CCA-secure**, if ∀PPT adversary $\mathcal{A}$, ∃ a negl. function ε(n) s. t.

$$\Pr[\text{PrivK-CCA}_{\mathcal{A},\Pi}(n)=1] \le \tfrac{1}{2} + \varepsilon(n)$$

Unforgeable secrecy

Enc-Forge$_{\mathcal{A},\Pi}$(n) = 1 iff 1. $\mathrm{Dec}_k(c^*) = m^* \neq$ `fail` &
2. m^* not in $\mathcal{Q}$

Encryption scheme
Π = (Gen, Enc, Dec)

security parameter 1^n

Π $\mathcal{T}$

$\mathrm{Gen}(1^n) \rightarrow k$

$\mathrm{Enc}_k(m_i) \rightarrow c_i$

$\mathrm{Dec}_k(c_i) = m_i$ / `fail`

m_i [or c_i]

c_i [or m_i/`fail`]

...

c^*

$\mathcal{A}^{\mathrm{Enc}(k,)\ [,\ \mathrm{Dec}(k,)]}$

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

Π = (Enc,Dec) is **unforgeable**, if $\forall$ PPT adversary $\mathcal{A}$, $\exists$ a negl. function $\varepsilon(n)$ s. t.

$$\Pr[\text{Enc-Forge}_{\mathcal{A},\Pi}(n)=1] \leq \varepsilon(n)$$

Authenticated encryption (AE): Security definition

- A private-key encryption scheme Π = (Gen, Enc, Dec) is a (secure) **authenticated encryption** scheme if it is **CCA-secure** and **unforgeable**.

Generic AE constructions – First attempt

3 generic ways to construct a **secure authenticated encryption** scheme Π_{AE}

- Π_E = (Enc, Dec) is **CPA-secure** encryption scheme; Π_M = (Mac, Vrfy) is **secure MAC**
  - instantiate Π_E, Π_M using **independent** secret keys ke, km; then
- 1. **encrypt-and-authenticate**
  - $\mathrm{Enc}_{ke}(m) \longrightarrow c$; $\mathrm{Mac}_{km}(m) \longrightarrow t$; send ciphertext (c, t)
  - if $\mathrm{Dec}_{ke}(c) = m \neq$ `fail` and $\mathrm{Vrfy}_{km}(m,t)$ accepts, output m; else output `fail`
  - **insecure scheme, generally**
    - e.g., MAC tag t may leak information about m
    - e.g., if MAC is deterministic (e.g., CBC-MAC) then Π_{AE} is not even CPA-secure

Generic AE constructions – Second attempt

3 generic ways to construct a **secure authenticated encryption** scheme Π_{AE}

- Π_E = (Enc, Dec) is **CPA-secure** encryption scheme; Π_M = (Mac, Vrfy) is **secure MAC**
  - instantiate Π_E, Π_M using **independent** secret keys ke, km; then
- 2. **authenticate-then-encrypt**
  - $\mathrm{Mac}_{km}(m) \rightarrow t$; $\mathrm{Enc}_{ke}(m||t) \rightarrow c$; send ciphertext c
  - if $\mathrm{Dec}_{ke}(c) = m||t \neq$ `fail` and $\mathrm{Vrfy}_{km}(m,t)$ accepts, output m; else output `fail`
  - **insecure scheme, generally**
    - e.g., CPA-secure CBC-mode encryption with padding is not CCA-secure, if $\mathrm{Dec}_{ke}()$ leaks an error bit due to bad padding of the original message

Generic AE constructions – Third attempt

3 generic ways to construct a **secure authenticated encryption** scheme Π_{AE}

- Π_E = (Enc, Dec) is **CPA-secure** encryption scheme; Π_M = (Mac, Vrfy) is **secure MAC**
  - instantiate Π_E, Π_M using **independent** secret keys ke, km; then
- 3. **encrypt-then-authenticate** (cf. "authenticated encryption")
  - $\mathrm{Enc}_{ke}(m) \rightarrow c$; $\mathrm{Mac}_{km}(c) \rightarrow t$; send ciphertext (c, t)
  - if $\mathrm{Vrfy}_{km}(c,t)$ accepts then output $\mathrm{Dec}_{ke}(c) = m$, else output `fail`
  - **secure scheme, generally**: as long as Π_M is a **strong MAC**
    - $\mathcal{A}$ cannot produce valid ciphertexts not explicitly queried via oracle $\mathrm{Enc}_{ke}()$
    - $\mathcal{A}$ cannot meanigfully exploit oracle $\mathrm{Dec}_{ke}()$
    - CCA-security reduced to CPA-security + secure MAC

Application: Secure communication sessions

An AE scheme Π_{AE} = (Enc, Dec) enables 2 parties to **communicate securely**

- session: period of time during which sender and receiver maintain state
- idea: send any message m as $c = Enc_k(m)$ & ignore received c's that don't verify
- security
  - by applying Π_{AE} messages' **<u>secrecy and integrity is protected</u>**
- remaining possible attacks
  - **re-ordering** attack — counters can be used to eliminate reordering/replays
  - **replay** attack — directional bit can be used to eliminate reflections
  - **reflection** attack — $c = Enc_k(b_{a\rightarrow b}||ctr_{A,B}||m)$; $ctr_{A,B}$++