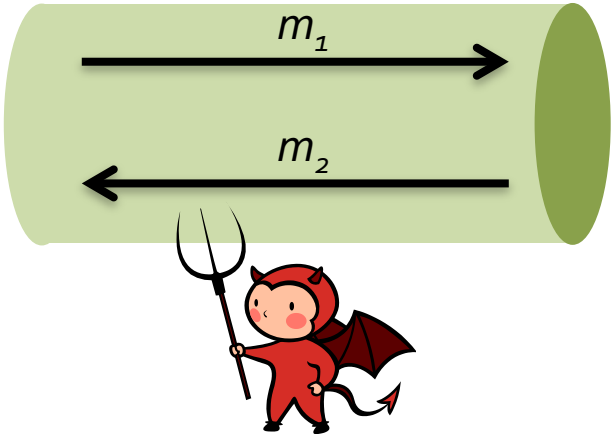
# Motivation for Authenticated Encryption

# Authenticated Encryption (AE)




Security goals:

Confidentiality and integrity of messages exchanged between Alice and Bob.

Adversarial capabilities:

Adversary can arbitrarily delete, reorder, modify, etc, bits on the wire.

Adversary can mount chosen plaintext and chosen ciphertexts attacks – formalised via encryption and decryption *oracles.*

Tools we have:

Encryption (e.g. block cipher in CBC mode, CTR mode, stream cipher) and MAC algorithms (e.g. HMAC, CBC-MAC).

# Formalising Symmetric Encryption

A symmetric encryption scheme consists of a triple of algorithms: (KGen,Enc,Dec).

**KGen**: key generation, selects a key $K$ uniformly at random from $\{0,1\}^k$.

**Enc**: encryption, takes as input key $K$, plaintext $m \in \{0, 1\}^*$ and produces output $c \in \{0, 1\}^*$.

**Dec**: decryption, takes as input key $K$, ciphertext $C \in \{0, 1\}^*$ and produces output $m \in \{0, 1\}^*$ or an error message, denoted $\perp$.

**Correctness**: we require that for all keys $K$, and for all plaintexts $m$,

$$\mathrm{Dec}_K(\mathrm{Enc}_K(m)) = m.$$

**Notes**:

- Enc may be randomised (cf. CBC mode, CTR mode).
- In reality, there will be a maximum plaintext length that can be encrypted by a given scheme.
- Nonce-based and stateful formalisms to follow later.

# Authenticated Encryption – Informal Definition

A symmetric encryption scheme is said to offer Authenticated Encryption security if:

A chosen plaintext attacker (i.e. an attacker with access to an encryption oracle) can learn nothing about plaintexts from ciphertexts except their lengths.

AND

An attacker with access to an encryption oracle cannot *forge* any new ciphertexts.

- What does it mean "to learn nothing about plaintexts from ciphertexts"?
- How do we formalise "cannot forge any new ciphertexts"?
- Why is that property important anyway?

We use security *games*, like the one introduced previously for MAC unforgeability.

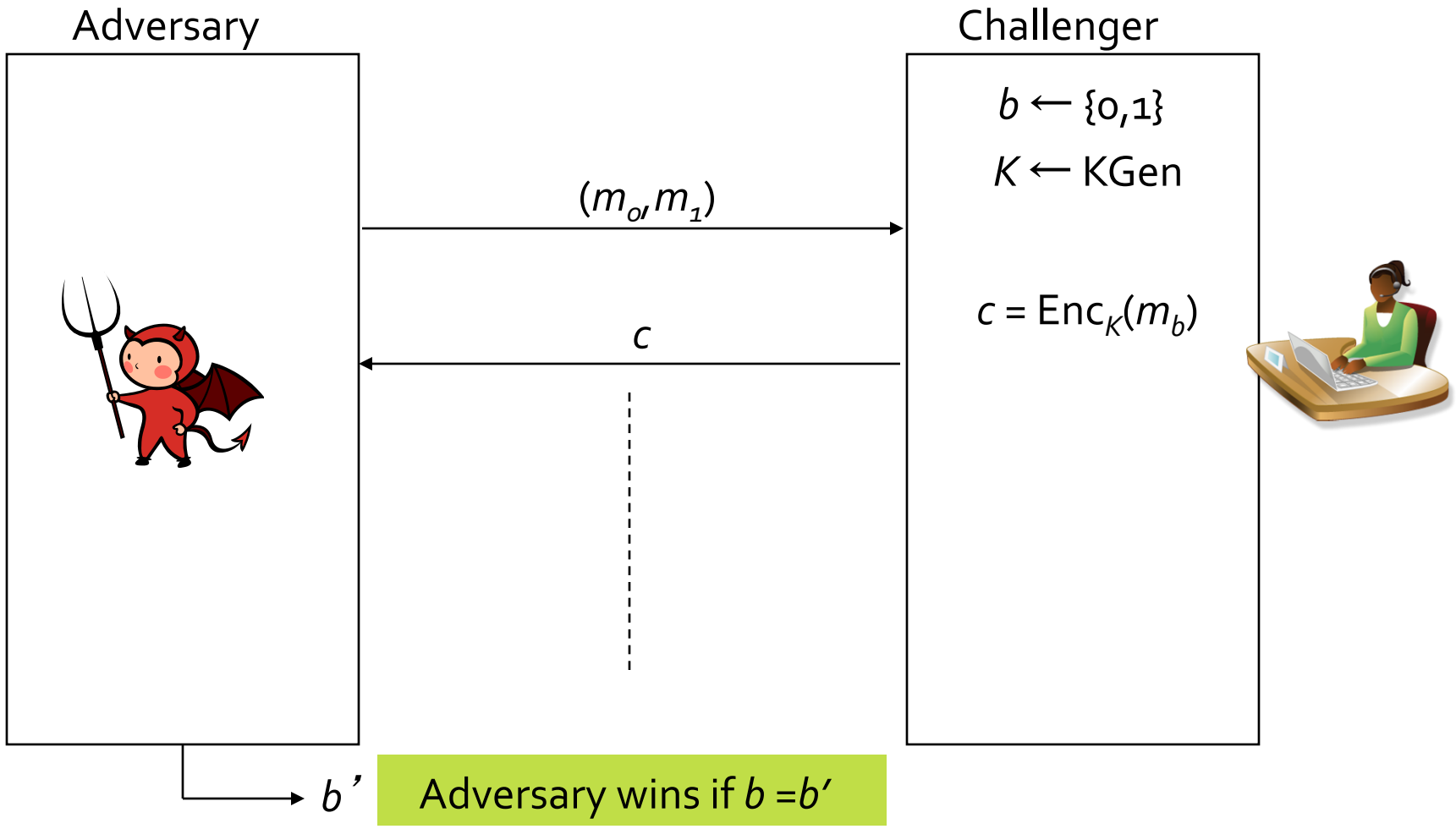
# IND-CPA security

- The adversary has repeated access to *Left-or-Right (LoR)* encryption oracle.
- In each query, the adversary submits pairs of equal length plaintexts ($m_0, m_1$) to the oracle.
  - We can have $m_0 = m_1$, so we get an encryption oracle "for free".
- The adversary gets back $c$, an encryption of $m_b$, where $b$ is a fixed but random bit.
- After all queries are made, the adversary outputs its estimate $b'$ for bit $b$.
- The adversary wins if it decides correctly.

**IND = Indistinguishable**

**CPA = Chosen Plaintext Attack**

# IND-CPA security in a picture



# IND-CPA security

The adversary's *advantage* in the IND-CPA security game is defined to be:

$$|\Pr(b=b') - 1/2|.$$

- We have "-1/2" here because a dumb adversary can always guess.
- A scheme SE is said to be IND-CPA secure if the advantage is "small" for *any* adversary using "reasonable" resources.
- Concepts of "small" and "reasonable" can be formalised, but are beyond the scope of this talk.
- It can be proved that schemes like CBC-mode and CTR-mode meet this security definition *if used properly and if they are built using a good block cipher.*

# Motivating stronger security

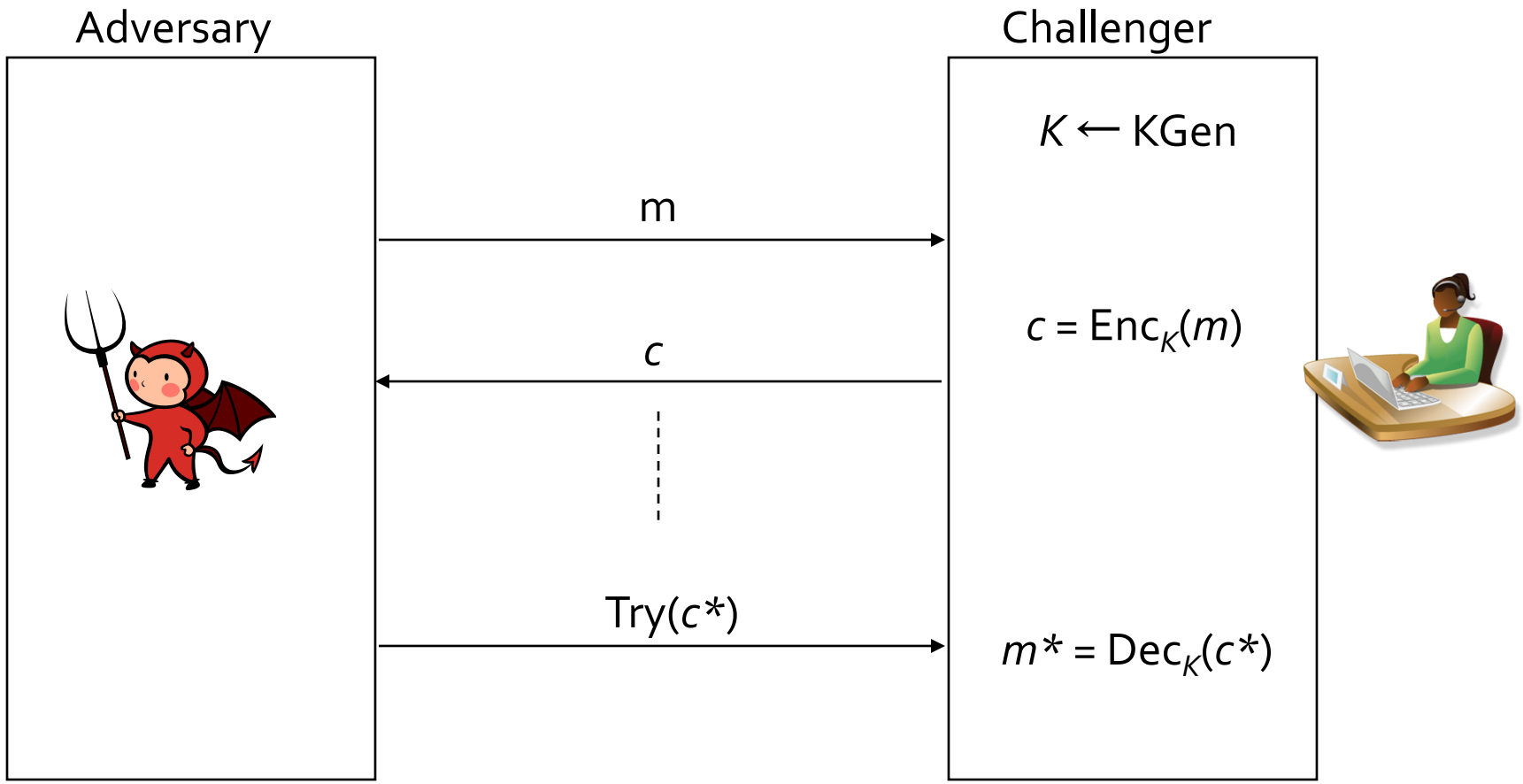
In CBC and CTR modes, an active adversary can *manipulate* ciphertexts and learn information from how these are decrypted.

- For CTR mode, bit flipping in plaintext is trivial by performing bit flipping in the ciphertext.
    - Modify $c$ to $c$ XOR $\Delta$ to change the underlying plaintext from $p$ to $p$ XOR $\Delta$ .
- CBC mode: cut and paste attacks, padding oracle attacks.
- Or create completely new ciphertexts from scratch?
    - A random string of bits of the right length is a valid ciphertext for *some* plaintext for both CBC and CTR modes!

# Motivating stronger security

- These kinds of attack do not break IND-CPA security, but are clearly undesirable if we want to build secure channels.
- A modified plaintext may result in wrong message being delivered to an application, or unpredictable behaviour at the receiving application.
- We really want some kind of *non-malleable* encryption, guaranteeing integrity as well as confidentiality.
- Two basic security notions: *integrity of plaintexts* and *integrity of ciphertexts*.

# INT-CTXT security in a picture




Adversary wins if $c^*$ is "new" and $m^* \neq \perp$

# Integrity of Ciphertexts – INT-CTXT

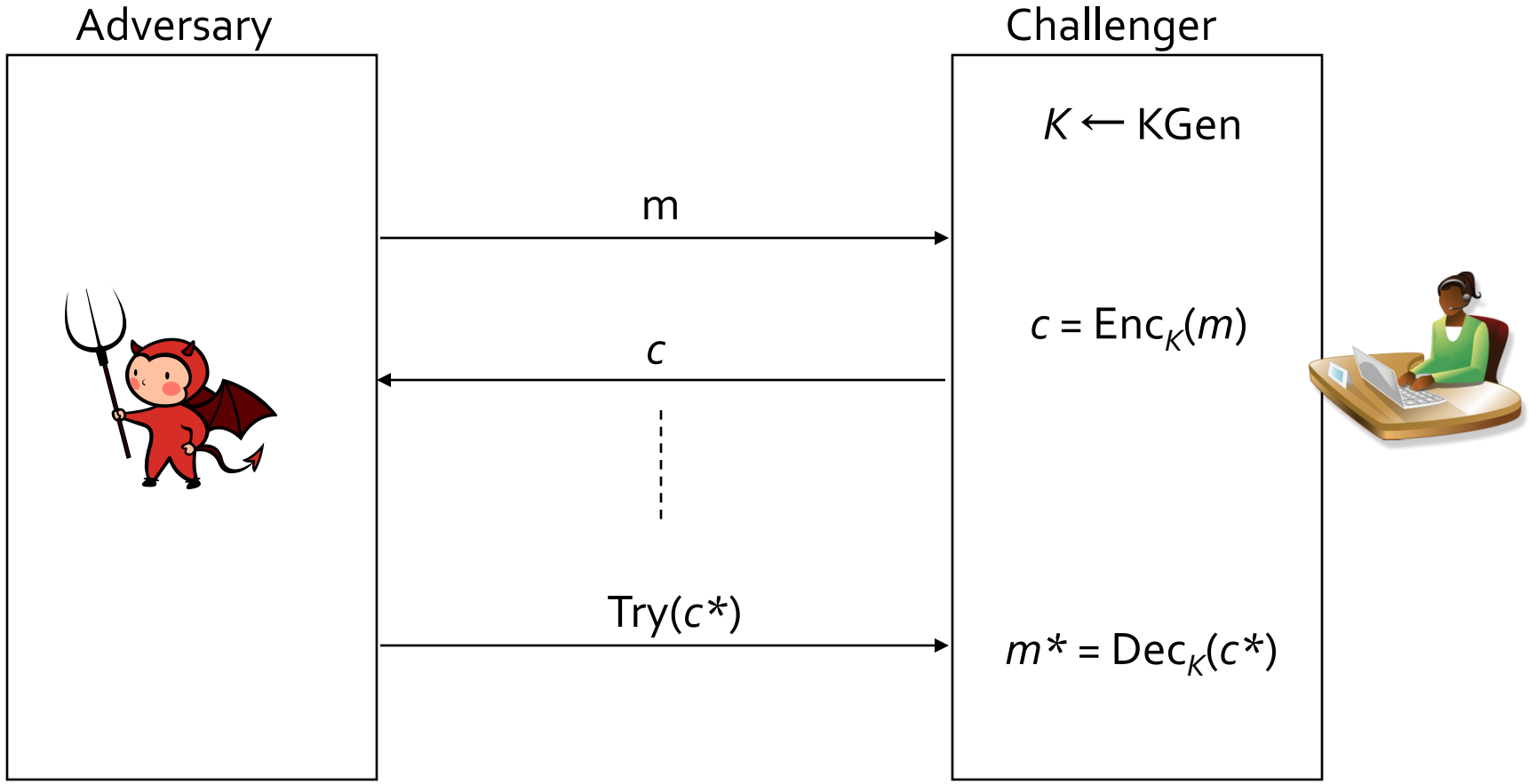
- Attacker has repeated access to an encryption oracle and a "Try" oracle.
  - Encryption oracle takes any $m$ as input, and outputs $\mathrm{Enc}_K(m)$.
  - Try oracle takes any c* as input (and has no output).
- Adversary's task is to submit $c^*$ to its Try oracle such that:
  1. $c^*$ is distinct from all the ciphertexts $c$ output by the encryption oracle; and
  2. $\mathrm{Dec}_K(c^*)$ decrypts to message $m^* \neq \perp$.

- Hence adversary wins if it can create a "ciphertext forgery" – a new ciphertext that it did not get from its encryption oracle.
- NB: we do not insist that $m^*$ be different from all the $m$ queried to the encryption oracle, only that $c^*$ be different from all the outputs of that oracle.

# INT-CTXT security

- A symmetric encryption scheme is said to provide INT-CTXT security if the success probability of any adversary using reasonable resources is small.
- Again, this can be made precise (but not today!).

# INT-PTXT security in a picture




Adversary wins if $m^*$ is "new" and $m^* \neq \perp$

# INT-PTXT security

- INT-PTXT: same as INT-CTXT, but now adversary needs to come up with a ciphertext $c^*$ that encrypts a message $m^*$ such that $m^*$ was never queried to the encryption oracle.

- Informally, INT-PTXT security means that the adversary can't force a new *plaintext* to be accepted by the receiver.

- If a scheme is INT-CTXT secure, then it is also INT-PTXT secure.

- For a secure channel, we actually want INT-PTXT security, not INT-CTXT security. (Why?)

# Definitions for AE Security

# AE Security

Recall that a symmetric encryption scheme is said to offer Authenticated Encryption security if:

A chosen plaintext attacker can learn nothing about plaintexts from ciphertexts except their lengths.
AND
An attacker with access to an encryption oracle cannot *forge* any new ciphertexts.

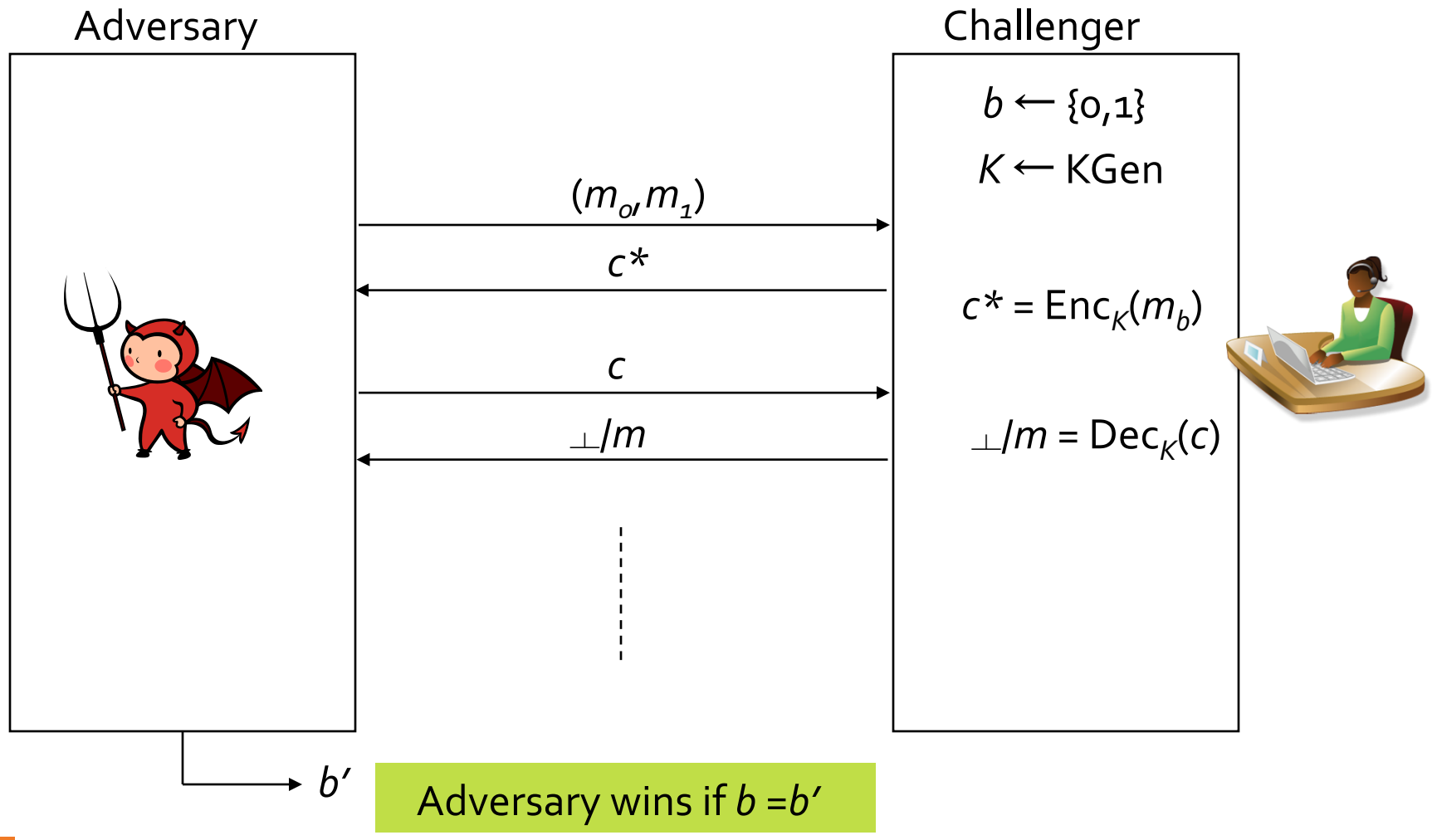
More formally, we can now say that:

$$\text{AE} = \text{IND-CPA} + \text{INT-CTXT}$$

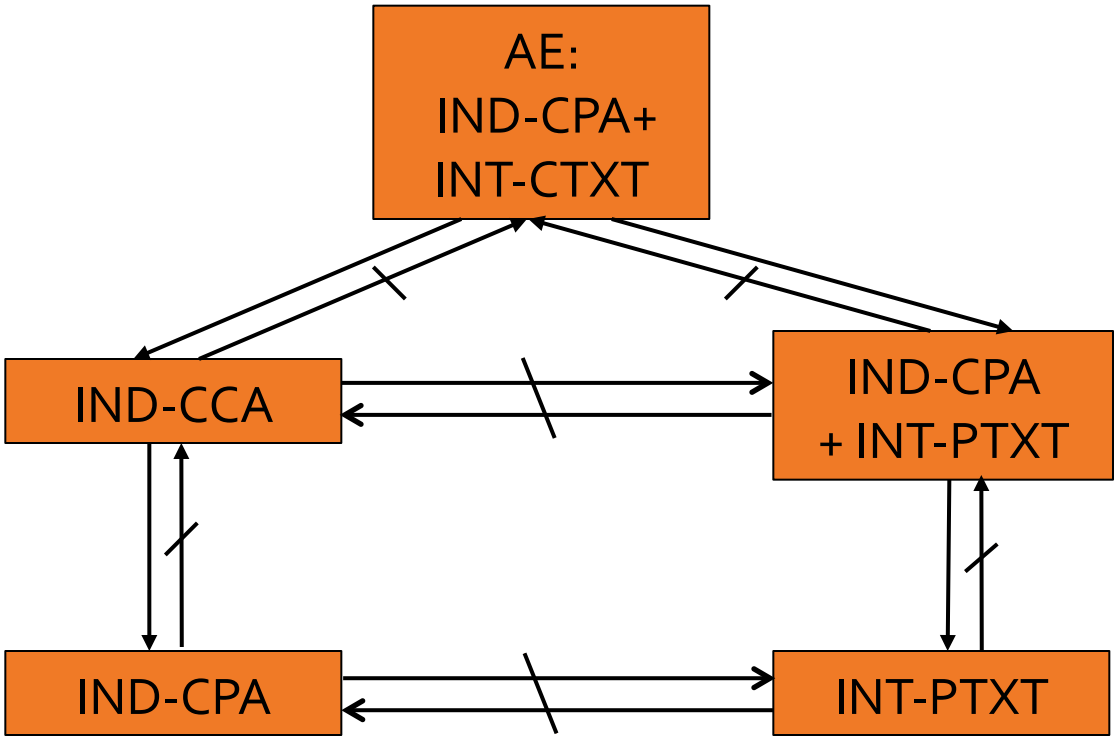
# What about chosen ciphertext attacks?

- We are also interested in security against chosen ciphertext attacks.
  - Here the adversary has access to both an encryption oracle *and* a decryption oracle.
  - Leading to the IND-CCA security notion, stronger than IND-CPA.
- This attack model may arise in practice, or the attacker may have an *approximation* to a decryption oracle.
  - An attacker might not be able to learn the full plaintext, but could get partial information about the decryption process, for example, error messages, timing information, etc.
  - cf. padding oracle attacks, ICMP attack on IPsec, etc.

# IND-CCA security in a picture



# Relations between security notions



# AE security and beyond

- AE security has emerged as the natural target security notion for symmetric encryption.
- In part because AE security implies IND-CCA security and INT-PTXT security.

- However it's not the end of the story:
  - In many applications we want to integrity protect some data and provide confidentiality for the remainder – AE with Associated Data, AEAD.
  - AE security does not protect against attacks on secure channels based on reordering or deletion of ciphertexts.
  - For this, we need stateful or nonce-based security definitions.

# Generic composition

# Generic composition for AE

- We have IND-CPA secure encryption schemes (e.g. CBC mode, CTR mode) and we have SUF-CMA secure MAC schemes.
- Can we combine these to obtain AE security for symmetric encryption?
- Problem first addressed by Bellare-Namprempre (2000) and Krawczyk (2001).
- Generic options: E&M, MtE, EtM.
- (In what follows, *KM* denotes a MAC key, and *KE* an encryption key.)

# Generic composition for AE

**Encrypt-and-MAC (E&M)**

- compute $c' \leftarrow \mathrm{Enc}_{KE}(m)$ and $\tau \leftarrow \mathrm{Tag}_{KM}(m)$ and output $c = (c', \tau)$.
- used in SSH

**MAC-then-Encrypt (MtE)**

- compute $\tau \leftarrow \mathrm{Tag}_{KM}(m)$ and output $c = \mathrm{Enc}_{KE}(m \,||\, \tau)$.
- used in TLS

**Encrypt-then-MAC (EtM)**

- compute $c' \leftarrow \mathrm{Enc}_{KE}(m)$ and $\tau \leftarrow \mathrm{Tag}_{KM}(c')$ and output $c = (c', \tau)$.
- used in IPsec ESP "enc + auth"

# Security of generic composition for AE

- Generic options: E&M, MtE, EtM.
- Of these, only EtM gives AE security in general.
  - Assuming encryption is IND-CPA secure and MAC is SUF-CMA secure.
  - Intuition: MACing the ciphertext *c'* provides ciphertext integrity; IND-CPA security of encryption carries over to the composition.
  - Plus point: check MAC on ciphertext, don't even decrypt if it fails; no temptation for programmer to "use the plaintext anyway" if MAC fails.

# Security of generic composition for AE

- To see why E&M fails to be secure in general:
  - Suppose we have a SUF-CMA secure MAC scheme, with tagging algorithm $\mathrm{Tag}_{KM}(m)$.
  - Think about the MAC scheme which outputs $\mathrm{Tag}_{KM}(m) \,||\, m$.
  - Is it SUF-CMA secure?
  - What about the security of the resulting E&M scheme?

# Security of generic composition for AE

To see why MtE can fail to be secure is more subtle.

**Example**

Consider the MtE encryption scheme in which MAC is provided by HMAC and the encryption scheme is provided by CBC-mode using simplified TLS padding.

Good MAC (SUF-CMA) and good encryption scheme (IND-CPA)!

- KGen: select at random two keys, $KM$, $KE$.
- Encryption: $c = \text{CBC-Enc}_{KE}(\text{TLS-PAD}(m \,||\, \text{Tag}_{KM}(m)))$.
- Decryption: ???

# Security of MtE generic composition for AE

- Encryption: $c = \text{CBC-Enc}_{KE}(\text{TLS-PAD}(m \,||\, \text{Tag}_{KM}(m)))$.
- Decryption:
    1. Perform CBC-mode decryption.
    2. Perform depadding – possibility of padding error.
    3. Perform MAC verification – possibility of MAC verification error.

If the errors at steps 2 and 3 are *distinguishable*, then we can carry out a padding oracle attack!

- Padding error -> padding bad.
- MAC verification error -> padding good!

This attack is a special case of a chosen-ciphertext attack, which should be prevented by AE security (recall AE security implies IND-CCA security).

# Security of MtE generic composition for AE

- We've just seen an example of a scheme constructed from components that are both good (IND-CPA secure encryption scheme, SUF-CMA secure MAC) but for which the MtE composition fails to be secure.

- The example is closely related to the construction that is used in TLS.

- Specific ways of instantiating MtE can be made secure, but it's unsafe in general and must be avoided wherever possible.

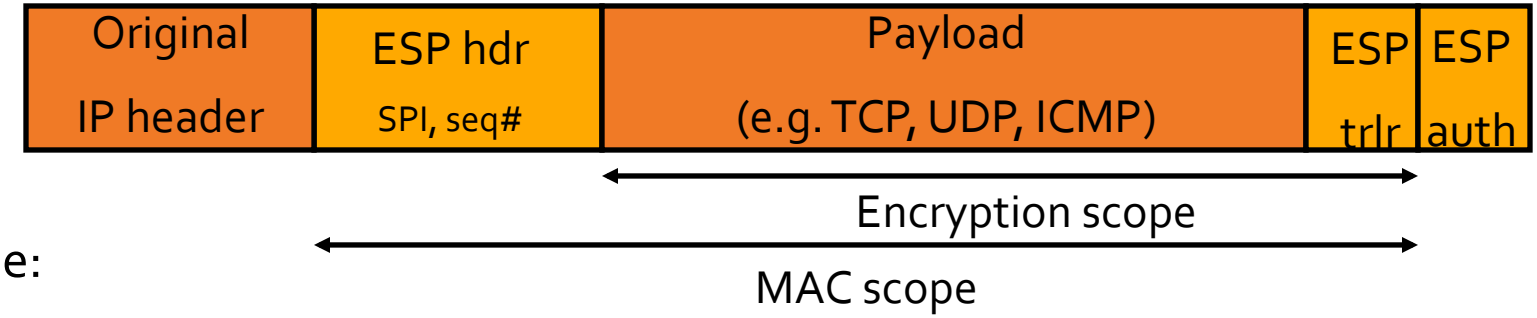
# AEAD

# Authenticated Encryption with Associated Data (AEAD)

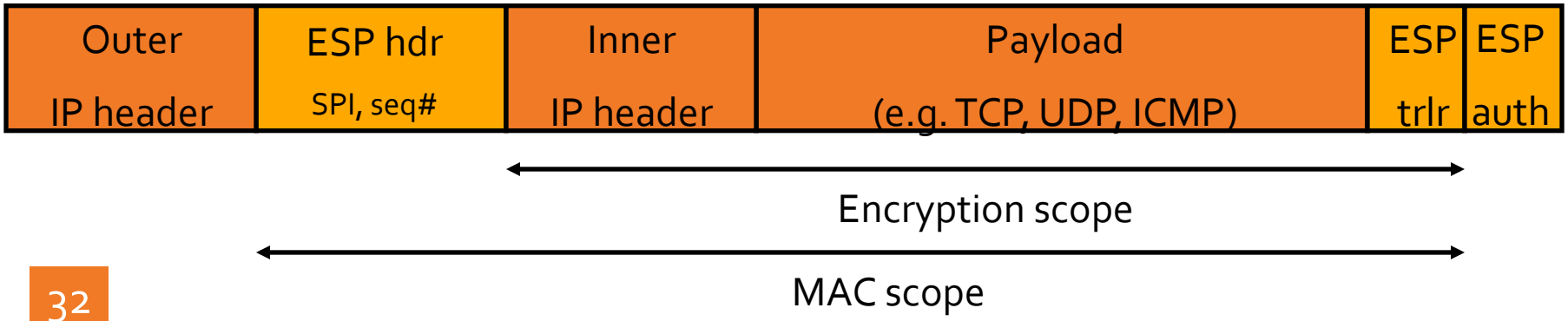
In practical applications, we often require confidentiality and integrity for some data fields and only integrity for others.

**Example: ESP in transport and tunnel modes in IPsec**

Transport mode:




Tunnel mode:



# Authenticated Encryption with Associated Data (AEAD)

An AEAD scheme consists of a triple of algorithms: (KGen,Enc,Dec).

**KGen**: key generation, selects a key $K$ uniformly at random from $\{0,1\}^k$.

**Enc**: encryption, takes as input key $K$, **associated data $AD \in \{0,1\}^*$**, plaintext $m \in \{0,1\}^*$, and produces output $c \in \{0,1\}^*$.

**Dec**: decryption, takes as input key $K$, **associated data $AD \in \{0,1\}^*$**, ciphertext $c \in \{0,1\}^*$, and produces output $m \in \{0,1\}^*$ or an error message, denoted $\perp$.

**Correctness**: we require that for all keys $K$, for all associated data strings $AD$, and for all plaintexts $m$:

$$\mathrm{Dec}_K(\mathrm{AD},\mathrm{Enc}_K(\mathrm{AD},m)) = m.$$

**AEAD security (informal):**

IND-CPA security for messages $m$, integrity for combination of associated data $AD$ and ciphertext $c$.

# Nonce-based AEAD

# Nonce-based AEAD

Nonce-based AEAD = AEAD with nonces!

**Motivation**:

- AEAD schemes as we have described them so far must consume randomness in Enc algorithm to achieve AE security
    - (IND-CPA security requires randomised encyption – why?)
- Guaranteeing good sources of randomness is hard.
- It's dangerous to hand this responsibility to the programmer, by asking him to supply the required randomness (e.g. IV for CBC mode).
- It is arguably easier to ensure that the programmer always passes a new nonce value as one of the inputs to the Enc algorithm (along with message *m* and associated data *AD*).

# Nonce-based AEAD

A nonce-based AEAD scheme consists of a triple of algorithms: (KGen,Enc,Dec).

**KGen**: key generation, selects a key $K$ uniformly at random from $\{0,1\}^k$.

**Enc**: encryption, takes as input key $K$, **nonce N** $\in \{0,1\}^n$, associated data $AD \in \{0,1\}^*$, plaintext $m \in \{0,1\}^*$, and produces output $c \in \{0,1\}^*$.

**Dec**: decryption, takes as input key $K$, **nonce N** $\in \{0,1\}^n$, associated data $AD \in \{0,1\}^*$, ciphertext $c \in \{0,1\}^*$, and produces output $m \in \{0,1\}^*$ or an error message, denoted $\perp$.

**Correctness**: we require that for all keys $K$, for all nonces N, for all associated data strings $AD$, and for all plaintexts $m$:

$$\mathrm{Dec}_K(\mathrm{N},\mathrm{AD},\mathrm{Enc}_K(\mathrm{N},\mathrm{AD},m)) = m.$$

# Security for nonce-based AEAD

**Nonce-based AEAD security (informal):**

IND-CPA security for messages *m*, integrity for combination of associated data *AD* and ciphertext *c*, **for adversaries that never repeat the nonce in encryption queries.**

- In the IND-CPA security game, the adversary now gets to specify a pair ($m_0$,$m_1$), along with *AD* and *N* in encryption queries.
  - Adversary never repeats *N.*
- In the INT-CTXT game, adversary now gets to specify *m*, *AD* and *N* in encryption queries.
  - Adversary never repeats *N.*
- Idea of nonce restriction: application will ensure an adversary can never access encryption/decryption functionalities with a repeated nonce.

# Using nonce-based AEAD

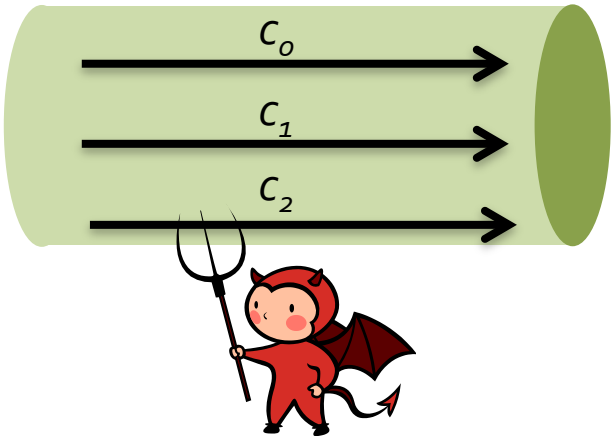
**Enc**: encryption, takes as input key $K$, **nonce N** $\in \{0,1\}^n$, associated data $AD \in \{0,1\}^*$, plaintext $m \in \{0,1\}^*$, and produces output $c \in \{0,1\}^*$.

**Dec**: decryption, takes as input key $K$, **nonce N** $\in \{0,1\}^n$, associated data $AD \in \{0,1\}^*$, ciphertext $c \in \{0,1\}^*$, and produces output $m \in \{0,1\}^*$ or an error message, denoted $\perp$.

**Notes:**

- For decryption to "undo" encryption, the same value of the associated data *AD* needs to be used.
- But the ciphertext *c* does not "contain" *AD.*
- In applications, *AD* may need to be sent along with *c*, or be reconstructed at the receiver.
- For decryption to "undo" encryption, the same nonce value *N* needs to be used.
- Again, *N* is not included in the ciphertext *c.*
- In applications, then, sender and receiver typically maintain a synchronized counter to ensure they both use the same *N* when encrypting and decrypting.

# Using nonce-based AEAD




$c_0 = \mathrm{Enc}_K(N{=}0, AD_0, m_0)$

$c_1 = \mathrm{Enc}_K(N{=}1, AD_1, m_1)$

$c_2 = \mathrm{Enc}_K(N{=}2, AD_2, m_2)$

$m_0 = \mathrm{Dec}_K(N{=}0, AD_0, c_0)$

$m_1 = \mathrm{Dec}_K(N{=}1, AD_1, c_1)$

$m_2 = \mathrm{Dec}_K(N{=}2, AD_2, c_2)$

- A sends B a sequence of messages $m_0, m_1, m_2, \ldots$ using nonce-based AEAD.
- A uses an incrementing counter for the nonces; B uses the same counter values when decrypting.
- What happens if the adversary deletes a ciphertext?
- What happens if the adversary reorders the ciphertexts, delivering $c_2$ before $c_1$, say?
- In both cases, receiver will use the wrong counter during decryption, so decryption will fail, producing an error message; adversary learns nothing, and so can't arrange undetectable deletion or force a message to be delivered "out of order".

# Further Constructions

# AEAD constructions

So far we have only seen *generic constructions* for AE schemes.

- EtM is the only one that is safe to use.
- EtM extends to the AEAD setting:

$$c' \leftarrow \mathrm{Enc}_{KE}(m);\ \tau' \leftarrow \mathrm{Tag}_{KM}(AD \,||\, c') \text{ and } c = (c', \tau').$$

- NB this is only secure if the length of *AD* is fixed or otherwise known to both Enc and Dec algorithms.
- EtM also extends to the nonce-based setting if "E" is a nonce-based encryption scheme.
  - Example: CBC-mode with $IV = \mathrm{E}_K(N)$ - use key to derive "random" IV block from nonce.
- Many other AEAD schemes are available; we will look at just two, CCM and GCM.

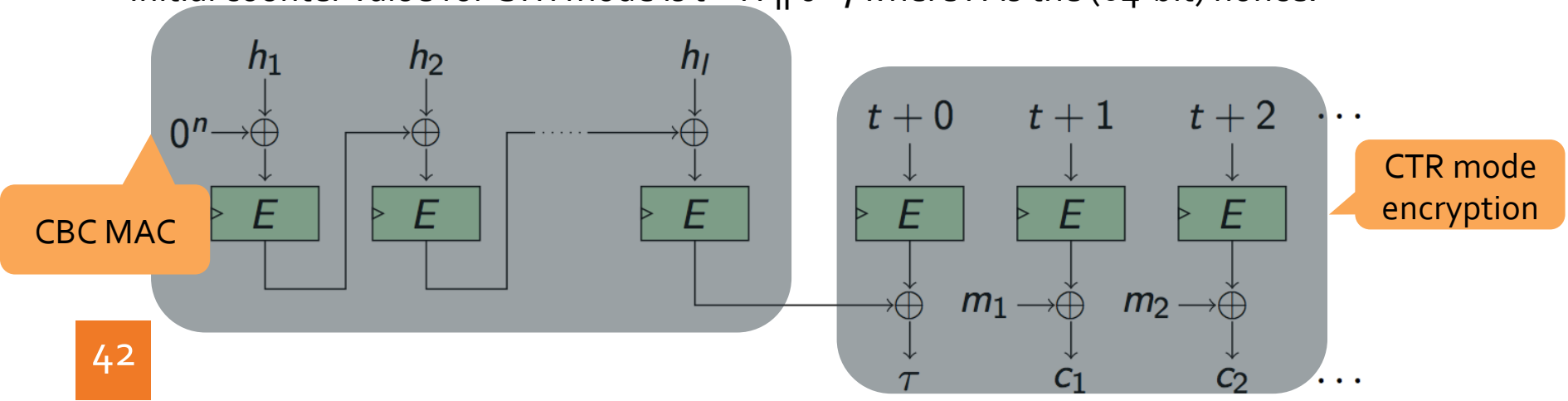
# CCM

**CCM = Counter with CBC MAC.**

- Basically, an instantiation of **MtE** with M = CBC-MAC and E = CTR mode, using a 128-bit block cipher, e.g. AES.

**Modifications**:

- Use same key for "M" and "E" components. (Bad idea in general, OK here.)
- Apply CBC-MAC to the string: h = $N$ || len($m$)$_{64}$ || $m$ || len($AD$)$_{64}$ || $AD$ || padding.
- Here, "||" means concatenate, len($X$)$_{64}$ means the 64-bit encoding of the length of string X.
- Initial counter value for CTR mode is $t = N \| 0^{64}$, where $N$ is the (64-bit) nonce.



# CCM

**CCM = Counter with CBC MAC.**

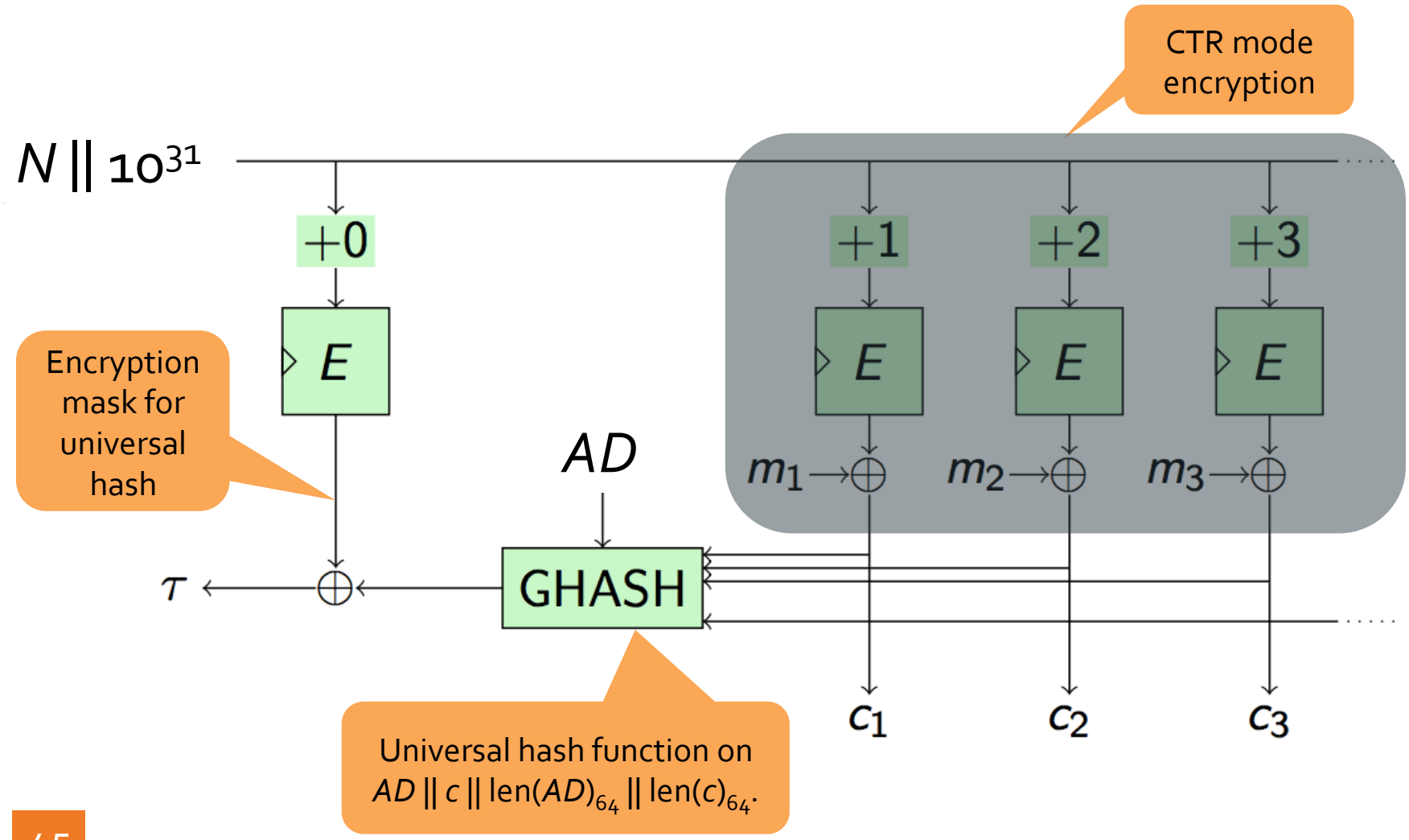
- CCM is quite slow: it needs one pass over associated data *AD* in CBC-MAC and two passes over the message *m*, one in CBC-MAC and one in CTR-mode encryption.
- CCM only uses block cipher in "forward direction", i.e. only "E" and no "D".
- CCM is patent-free.
- CCM is used in WPA2, the successor to WEP and WPA/TKIP.
- CCM is standardised for use in IPsec and TLS 1.2.
- CCM is specified in full in RFC 3610 (https://tools.ietf.org/html/rfc3610).
- CCM has as a security proof based on block cipher being a pseudo-random permutation.
- No known attacks (when implemented properly!)

# GCM

**GCM = Galois Counter Mode.**

- Basically, an instantiation of **EtM** with E = CTR mode using a 128-bit block cipher, e.g. AES, and M = a Wegman-Carter MAC.
- Nonces *N* can be of arbitrary length, special processing for 96-bit case for speed.
- Faster than CCM: speed up comes from use of fast MAC algorithm built from universal hash function family called GHASH.
- GCM only uses block cipher in "forward direction", i.e. only "E" and no "D".
- *AD* and *m* can be processed in block-wise fashion, no buffering required.
- GCM is patent-free.
- GCM is standardised for use in IPsec and TLS 1.2, now widely used in TLS.
- GCM is specified in full in NIST Special Publication SP800-38D (2007).
- GCM has a security proof based on block cipher being a pseudo-random permutation.
- No known attacks of significance (when implemented properly!)

# GCM (for 96-bit nonces)



# Other things you should probably know about AE

- Other modes are seeing growing adoption, e.g. OCB.
- Recent SHA-3 winner KECCAK can be adapted to produce an AE scheme!
- The whole area was mired in patents on early algorithm designs but the situation is gradually improving.

- Don't rely on wikipedia for discussion of the security of generic composition (it says MtE is OK; it's not in general)!

- CAESAR competition on-going (http://competitions.cr.yp.to/caesar.html), generating lots of new research activity and some controversy.
- See also the AE zoo https://aezoo.compute.dtu.dk/doku.php