Authenticated Encryption and Secure Channels

Kenny Paterson

Information Security Group

@kennyog; www.isg.rhul.ac.uk/~kp

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Overview

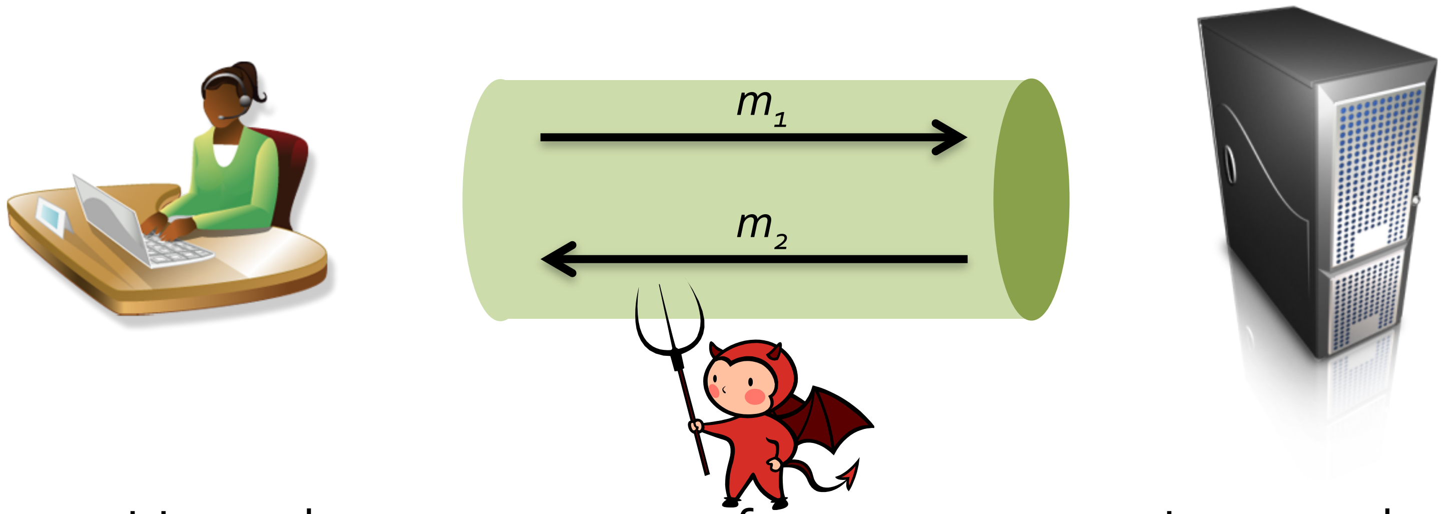
- Secure channels and their properties
- Security for symmetric encryption – from block ciphers to AEAD
- Going beyond AEAD – the ciphertext fragmentation and streaming settings.

Secure channels and their properties

Why do we need secure channels?

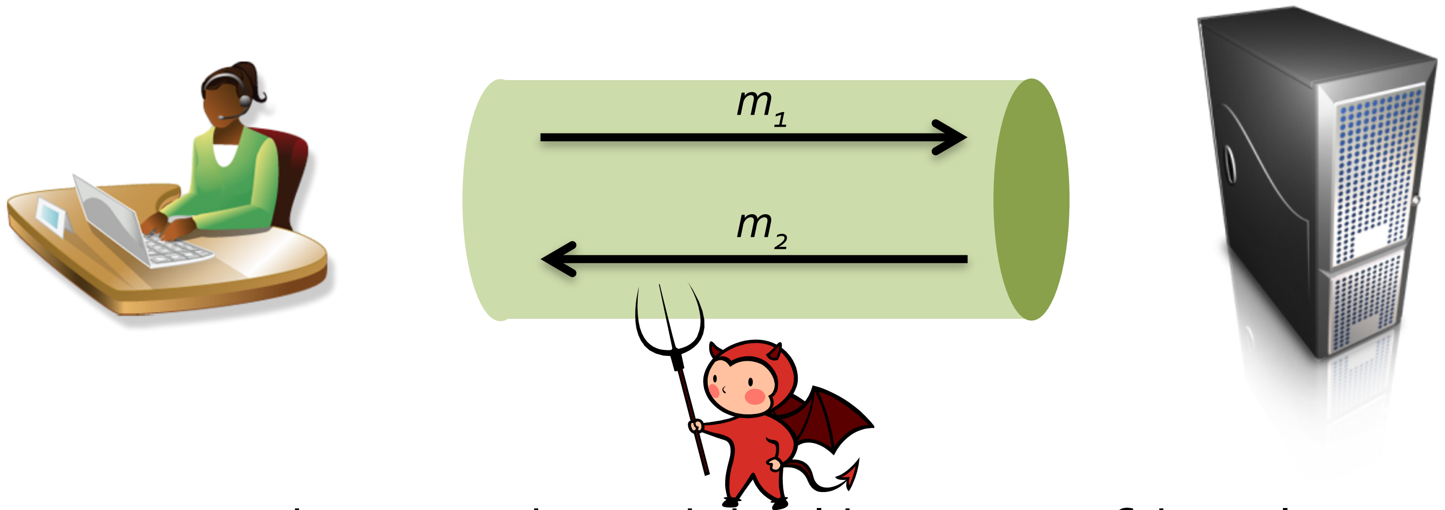
- Secure communications is the most common real-world application of cryptography today (**discuss**!)
- Consequently, secure channels are extremely widely-deployed in practice:
    - SSL/TLS, DTLS, IPsec, SSH, OpenVPN,…
    - WEP/WPA/WPA2
    - GSM/UMTS/LTE
    - Cryptocat, OTR, SilentCircle
    - Telegram, Signal, iMessage, and a thousand other messaging apps
    - QUIC, MinimalT, TCPcrypt

Secure communications



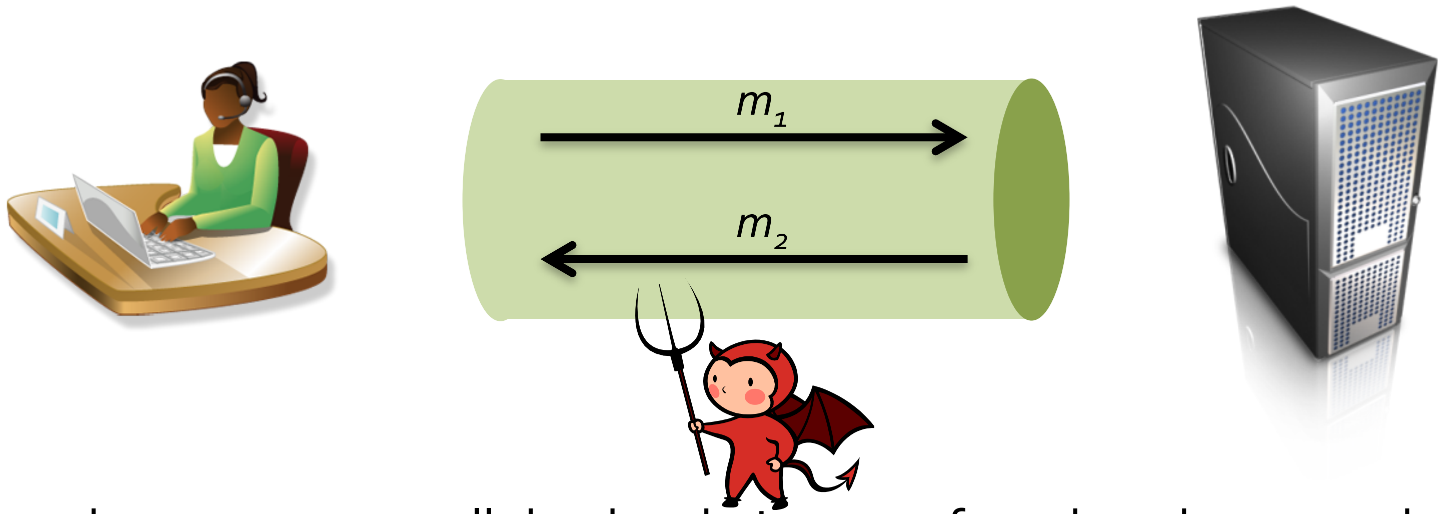
- Two entities exchange a sequence of messages over an insecure channel.
- Entities are often called *Alice* and *Bob*, but they need not be people.
- The messages will be sent over a communications network, e.g. a wireless LAN, "the Internet".
- Alice and Bob use cryptography to make build a secure communications channel on top of the insecure channel.

Secure communications: Security goals



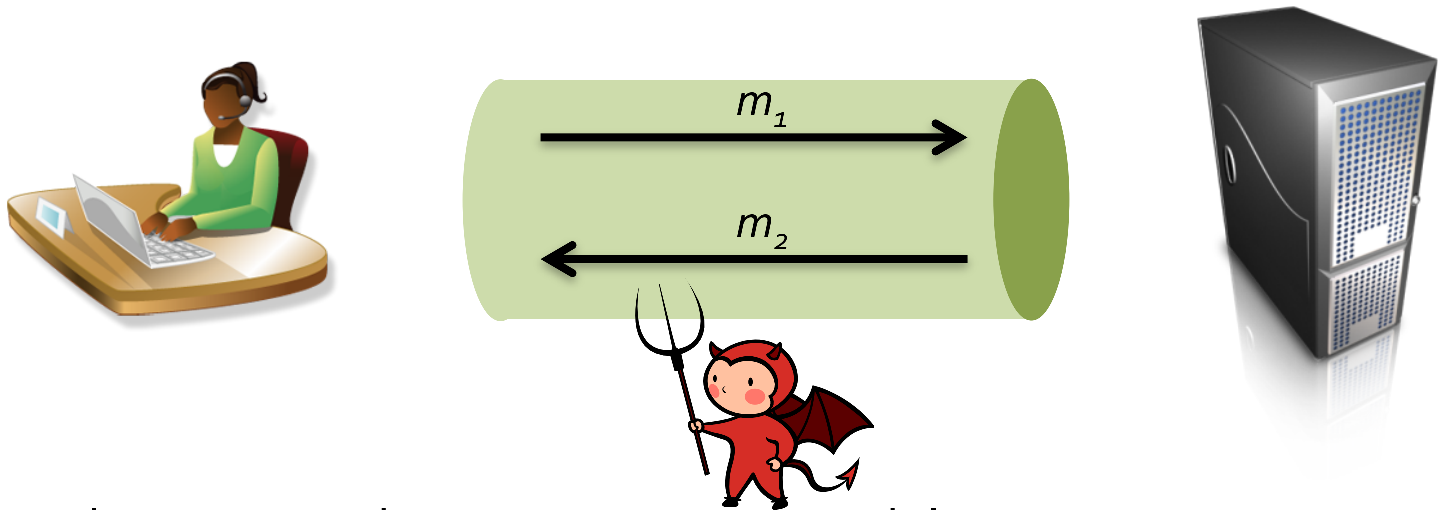
- The messages that are exchanged should remain confidential.
- It should be hard for the adversary to inject messages of his own into the sequence – so Alice and Bob should be able to check the *authenticity of origin* and *integrity* of the messages.
- Alice and Bob should be able to detect when messages are deleted.
- Alice and Bob should be able to detect when messages are reordered (possibly by the adversary, possibly by the network).

Secure communications: Adversarial capabilities



- The adversary can see all the data being transferred on the network (passive adversary).
- The adversary has sufficient control over the network that he can delete, delay, modify and reorder network packets at will (active adversary).
- The adversary can inject entirely new network packets (active adversary).
- **To what extent are these capabilities realistic?**

Secure communications: Adversarial capabilities



- The adversary may have even greater capabilities:
  - He can ask for specific messages *m* of his choice to be passed over the secure channel (chosen plaintext attack).
  - He can observe the effects on network messages when he injects network packets of his own (chosen ciphertext attack).
  - For example, he may be able to observe *error messages* exchanged between the entities in response to the messages he injects, and this may leak useful information.

Basic security goals

- **Confidentiality** – privacy for data
- **Integrity** – detection of data modification
- **(Data Origin) Authenticity** – assurance concerning the source of data

- (Often integrity and authenticity are equated/ confused – for example “Authenticated Encryption”.)

Some less obvious security goals

- **Anti-replay**
  - Detection that messages have been repeated
- **Drop-detection**
  - Detection that messages have been deleted by the adversary or dropped by the network.
- **Prevention of re-ordering**
  - Preserving the relative order of messages in *each* direction.
  - Preserving the relative order of messages sent and received in *both* directions.
- **Prevention of traffic-analysis.**
  - Using traffic padding and length-hiding techniques.

Possible functionality requirements

- **Speedy**
- **Low-memory**
- **On-line/parallelisable crypto-operations**
    - Performance is heavily hardware-dependent.
    - May have different algorithms for different platforms.
- **IPR-friendly**
    - This issue has slowed down adoption of many otherwise good algorithms, e.g. OCB.
- **Easy to implement**
    - Without introducing any side-channels.

API requirements

- We need a clean and well-defined Application Programming Interface (API).
- Because the reality is that our secure channel protocol will probably be used blindly by a security-naïve developer.
- Developers want to “open” and “close” secure channels, and issue “send” and “recv” commands.
- They’d like to simply replace TCP with a “secure TCP” having the same API.
- Or to just have a black-box functionality for delivering messages securely.

Additional API-driven requirements

- Does the channel provide a stream-based functionality or a message-oriented functionality?
- TCP-like or UDP-like?
- Does the channel accept messages of arbitrary length and perform its own fragmentation and reassembly, or is there a maximum message length?
- How is error handling performed? Is a single error fatal, leading to tear-down of channel, or is the channel tolerant of errors?
- How are these errors signalled to the calling application? How should the programmer handle them?

Additional API-driven requirements

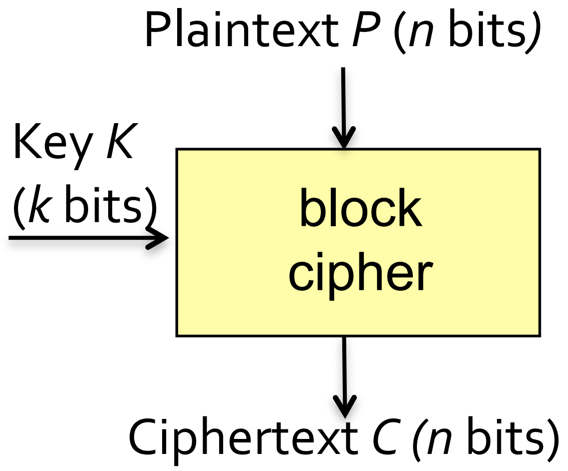
- Does the secure channel itself handle retransmissions? Or is this left to the application? Or is it guaranteed by the underlying network transport?
- Does the channel offer data compression?

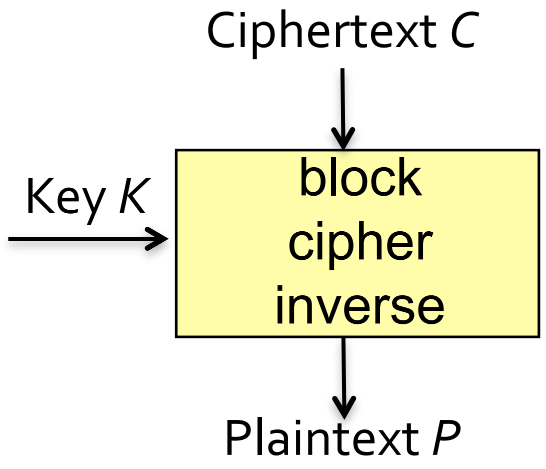
- **These are design choices that all impact on security**
- **They are not well-reflected in security definitions for symmetric encryption**
- **Main question: how can we use symmetric encryption to build secure channels?**

Block ciphers

Defining block ciphers

- Block ciphers encrypt and decrypt *blocks* of bits: *n* bits at a time.
  - Typically, *n* = 64 (eg DES) or *n* = 128 (AES).
- Encryption is under the control of a key *K* of size *k* bits.
  - Typically, *k* = 64 or *k* = 128.





Defining block ciphers

Definition:

A block cipher with key length k and block size n consists of two sets of efficiently computable bijections:

$$E_K: \{0,1\}^n \text{ --> } \{0,1\}^n \qquad \text{and} \qquad D_K: \{0,1\}^n \text{ --> } \{0,1\}^n$$

such that D_K is the inverse of E_K for each K in $\{0,1\}^k$.

Notes:

- E stands for encipher, D for decipher (not encrypt/decrypt).
- Elements K in $\{0,1\}^k$ are called keys; there are 2^k possible keys.

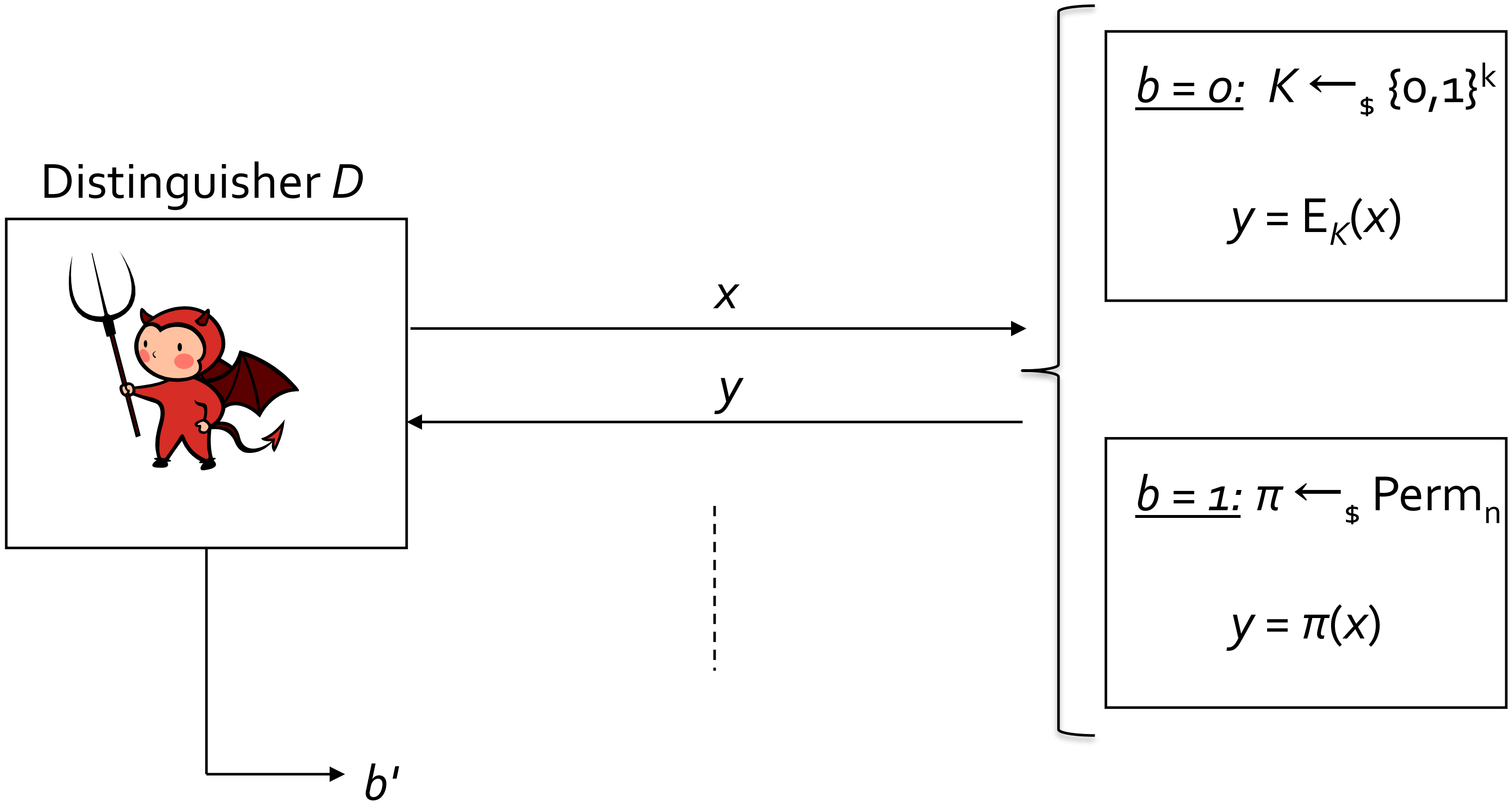
Security for block ciphers: Pseudorandomness

- Each key K gives a different pair of bijections (E_K, D_K).
  - So a block cipher gives us 2^k bijections from n bits to n bits.
  - But here are a total of $(2^n)!$ bijections from n bits to n bits.
  - And $(2^n)! >> 2^k$ for typical block cipher parameters.
- We would like each bijection from the block cipher to “look like” a random bijection, even though it comes from a tiny set.
- Another word for a bijection is a *permutation.*

Security for block ciphers: Pseudorandom Permutations (PRPs)

- This can be formalised as a pseudorandomness property:
  - A distinguisher *D* interacts either with (E_K, D_K) for a random choice of *K*, or with a random bijection and its inverse (π, π^{-1}).
  - The block cipher is said to be a strong-PRP if there is no *efficient D* that can tell the difference between (E_K, D_K) and (π, π^{-1}).
  - The block cipher is said to be a PRP if there is no *efficient D* that can tell the difference between E_K and π.
  - Efficiency of *D* can be *quantified* by the resources consumed by *D*: number of queries to (E_K, D_K) / (π, π^{-1}) and its running time.

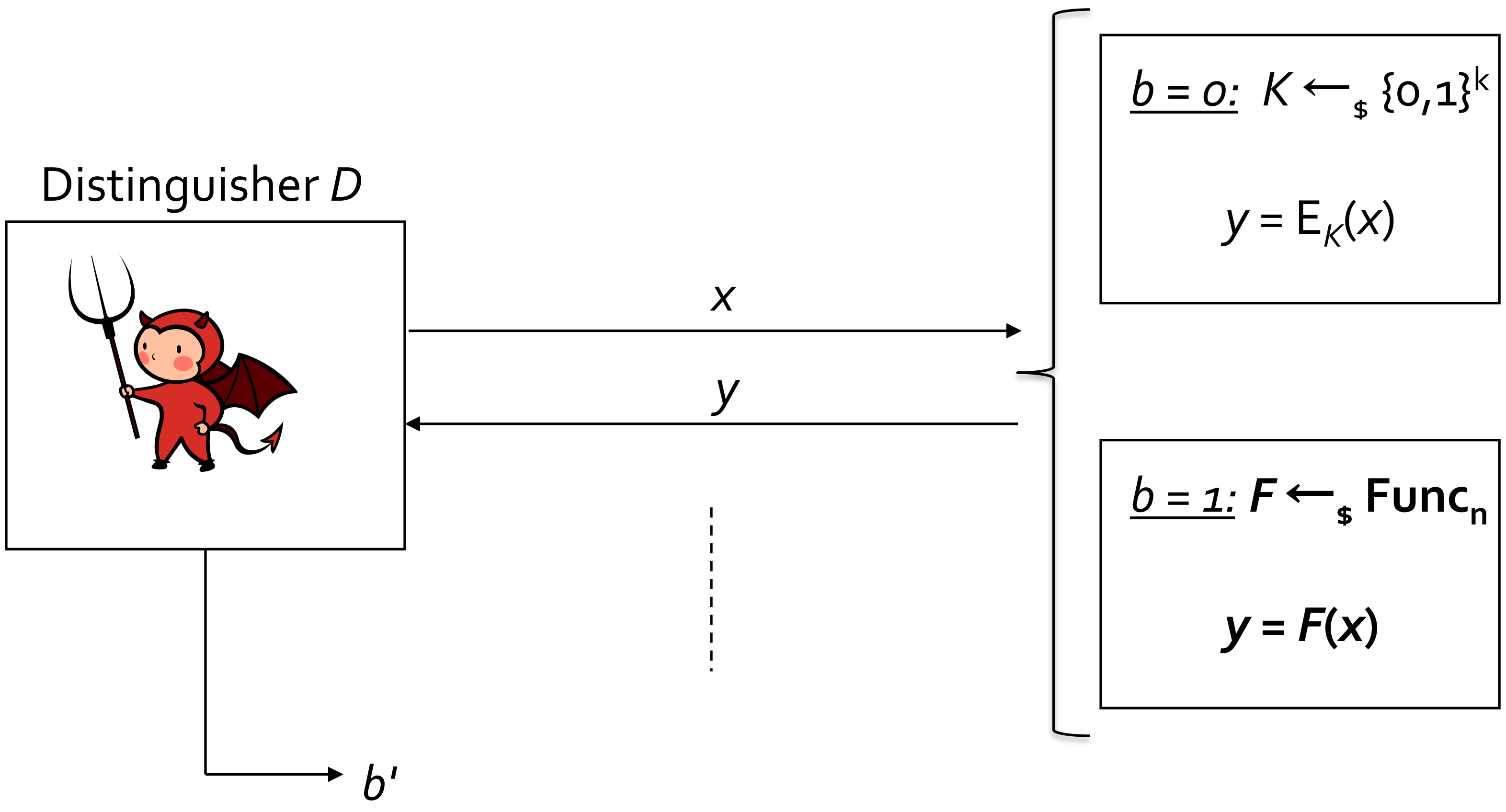
Formal definition of PRP security



Distinguisher D
x
y
$\underline{b = 0:}$ $K \leftarrow_\$ \{0,1\}^k$
$y = E_K(x)$
$\underline{b = 1:}$ $\pi \leftarrow_\$ \text{Perm}_n$
$y = \pi(x)$
b'

$\text{Adv}_E^{PRP}(D) := \Pr[b'=1 \mid b=1] - \Pr[b'=1 \mid b=0]$

Formal definition of **PRF** security


Distinguisher D
x
y
b'
$\underline{b = 0:}$ $K \leftarrow_{\$} \{0,1\}^k$
$y = E_K(x)$
$\underline{b = 1:}$ $\boldsymbol{F} \leftarrow_{\$}$ **Func**$_n$
$\boldsymbol{y = F(x)}$

$\mathrm{Adv}_E^{PRF}(D) := \Pr[b'=1 \mid b=1] - \Pr[b'=1 \mid b=0]$

The PRP/PRF switching lemma

A PRP for large n is already a good PRF! (and vice-versa).

Theorem (informal)

For any distinguisher D making at most q oracle queries, the difference in PRP and PRF advantages of D is at most $q^2/2^n$.

The proof uses relatively straightforward birthday bounds.

Loosely speaking, this means we can think of good block ciphers as being either random functions or random permutations (when secretly keyed).

Security for symmetric encryption

Modes of operation

- A block cipher encrypts a message of exactly *n* bits.
    - What if the message is not a multiple of *n* bits?
    - What if the message is not of a fixed length but actually a TCP-like stream?

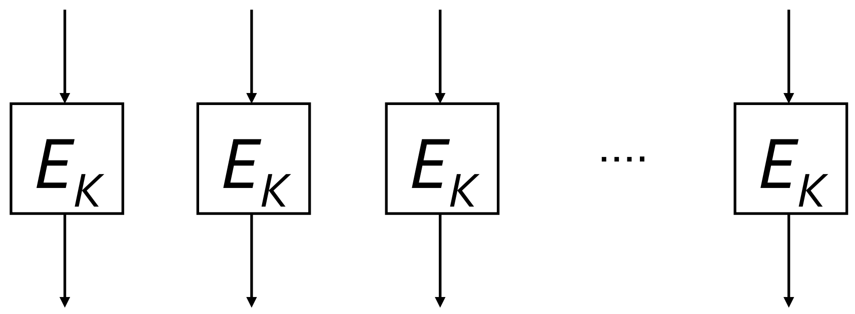
- Modes of operation provide different ways of using a block cipher to encrypt flexible amounts of data.
    - Different performance characteristics.
    - Different error-propagation properties.
    - Different suitability for different applications .

Main modes of operation

- FIPS 81 specifies four modes for DES.
    - ECB - Electronic Code Book.
    - CBC - Cipher Block Chaining.
    - CFB - Cipher Feedback.
    - OFB - Output Feedback.
- ANSI X9.52 specifies 7 modes for triple-DES
    - The four modes above and variants.
- Most common modes now in use: CBC mode and Counter Mode (CTR).

Electronic Code Book (ECB)

- ECB is the simplest way to use a block cipher to encrypt longer messages.

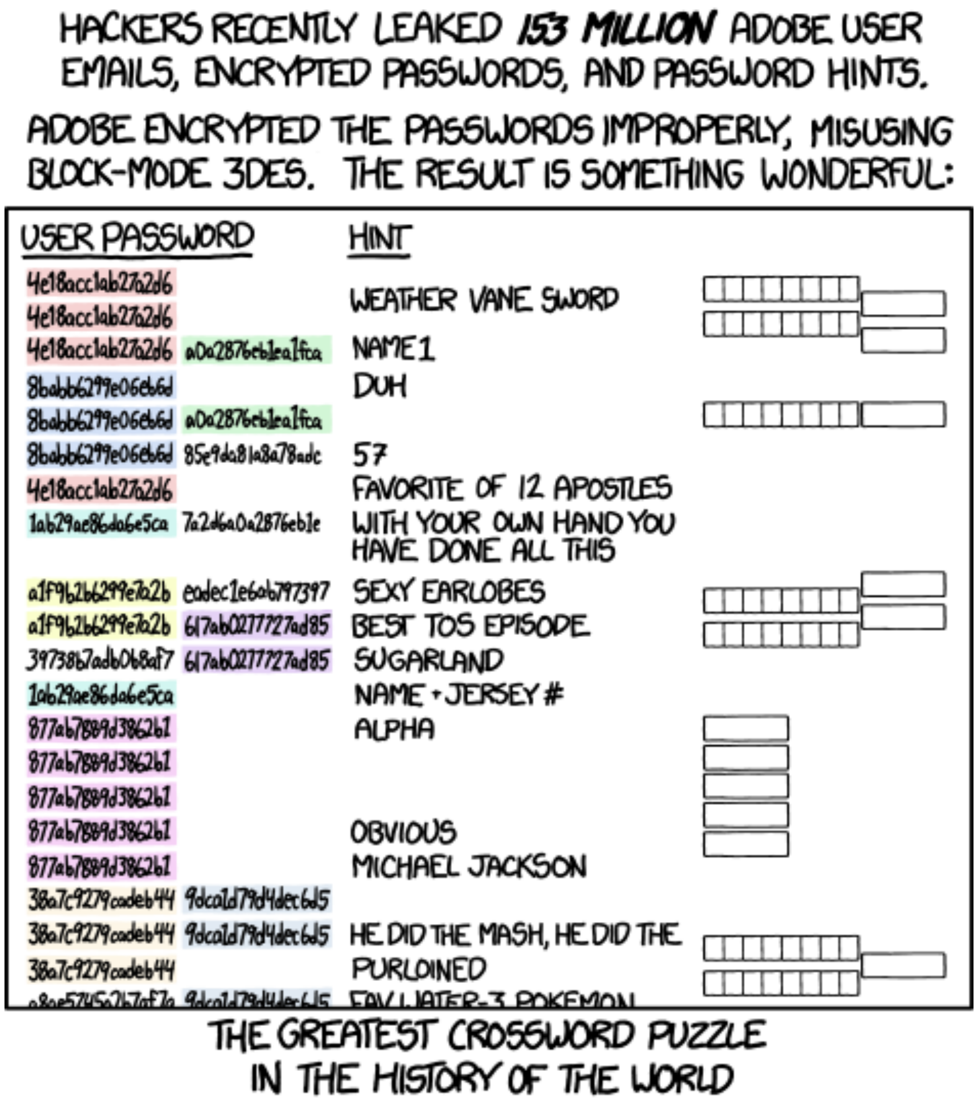


- Split message into blocks.
- Encryption can be parallelized.
- Any error in a ciphertext block affects the decryption of a single block.

ECB information leakage

- For a fixed key *K*, a given block of plaintext is always encrypted in the same way to produce the same ciphertext block.
- Encryption is *deterministic.*
- Leads to serious information leakage in many applications.
- ECB mode is *very rarely* the correct mode to use…

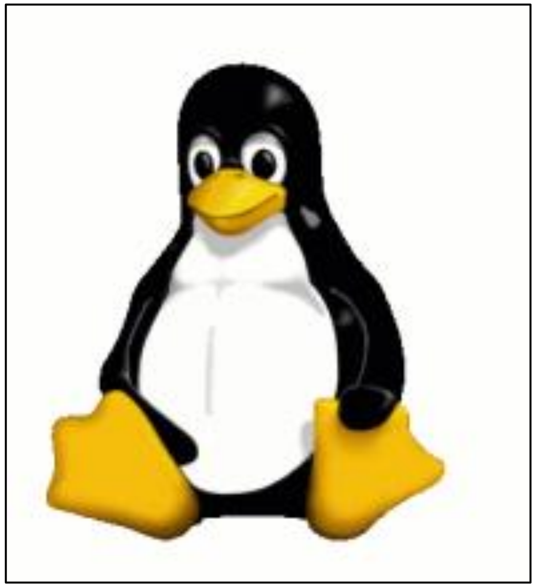
ECB information leakage

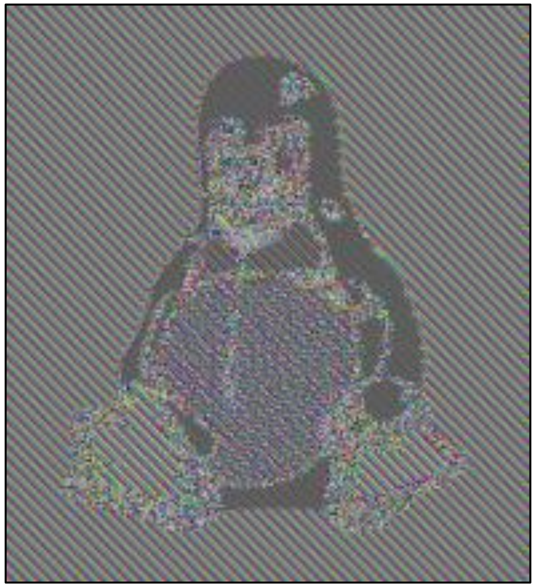


HACKERS RECENTLY LEAKED *153 MILLION* ADOBE USER EMAILS, ENCRYPTED PASSWORDS, AND PASSWORD HINTS.
ADOBE ENCRYPTED THE PASSWORDS IMPROPERLY, MISUSING BLOCK-MODE 3DES. THE RESULT IS SOMETHING WONDERFUL:
USER PASSWORD
HINT
4e18acc1ab27a2d6
4e18acc1ab27a2d6 WEATHER VANE SWORD
4e18acc1ab27a2d6 a0a2876eb1ea1fca NAME1
8babb6299e06eb6d DUH
8babb6299e06eb6d a0a2876eb1ea1fca
8babb6299e06eb6d 85e9da81a8a78adc 57
4e18acc1ab27a2d6 FAVORITE OF 12 APOSTLES
1ab29ae86da6e5ca 7a2d6a0a2876eb1e WITH YOUR OWN HAND YOU HAVE DONE ALL THIS
a1f96266299e7a2b eadec1e6ab797397 SEXY EARLOBES
a1f96266299e7a2b 617ab0277727ad85 BEST TOS EPISODE
3973867adb068af7 617ab0277727ad85 SUGARLAND
1ab29ae86da6e5ca NAME + JERSEY #
877ab7889d3862b1 ALPHA
877ab7889d3862b1
877ab7889d3862b1
877ab7889d3862b1 OBVIOUS
877ab7889d3862b1 MICHAEL JACKSON
38a7c9279cadeb44 9dca1d79d4dec6d5
38a7c9279cadeb44 9dca1d79d4dec6d5 HE DID THE MASH, HE DID THE
38a7c9279cadeb44 PURLOINED
a8ae5745a2b7af7a 9dca1d79d4dec6d5 FAV WATER-3 POKEMON
THE GREATEST CROSSWORD PUZZLE IN THE HISTORY OF THE WORLD

https://filippo.io/analyzing-the-adobe-leaked-passwords/

ECB information leakage

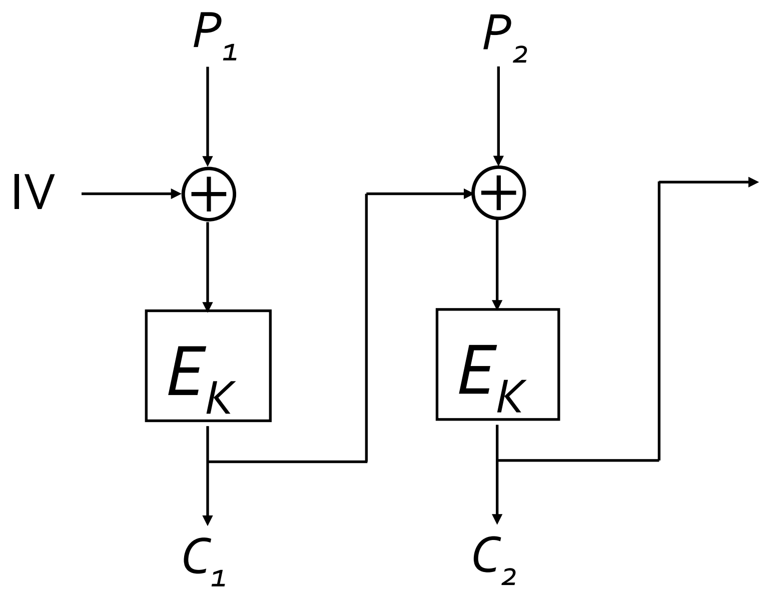

Tux the Penguin, the Linux mascot. Created in 1996 by Larry Ewing with The GIMP. lewing@isc.tamu.edu


ECB-Tux

CBC mode

Cipher Block Chaining (CBC) mode

- Aims to hinder information leakage of ECB mode.
- Uses previous ciphertext block (or IV) to randomise the input to the block cipher at each application.
- Not parallelisable.



Encryption equation:

$C_0 = \text{IV}$

$C_i = E_K(P_i \oplus C_{i-1})$

Decryption equation:

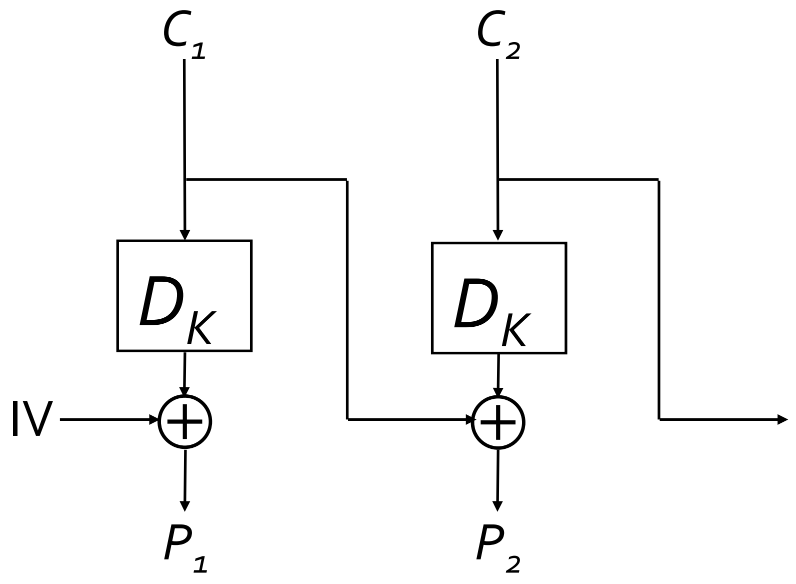
So:

$$D_K(C_i) = P_i \oplus C_{i-1}$$

and hence:

$$D_K(C_i) \oplus C_{i-1} = P_i$$

CBC mode decryption



Encryption equation:

$C_0 = \text{IV}$

$C_i = E_K(P_i \oplus C_{i-1})$

Decryption equation:

So:

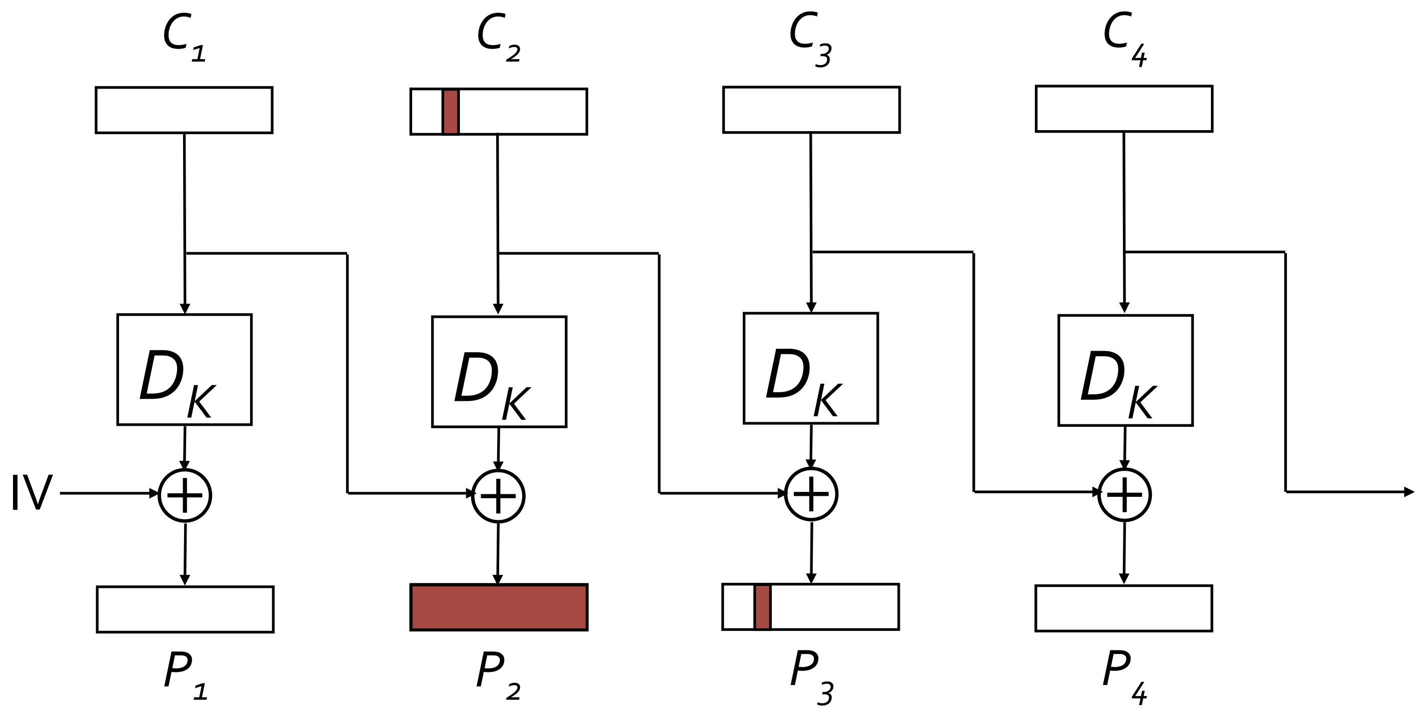
$$D_K(C_i) = P_i \oplus C_{i-1}$$

and hence:

$$D_K(C_i) \oplus C_{i-1} = P_i$$

Error propagation in CBC

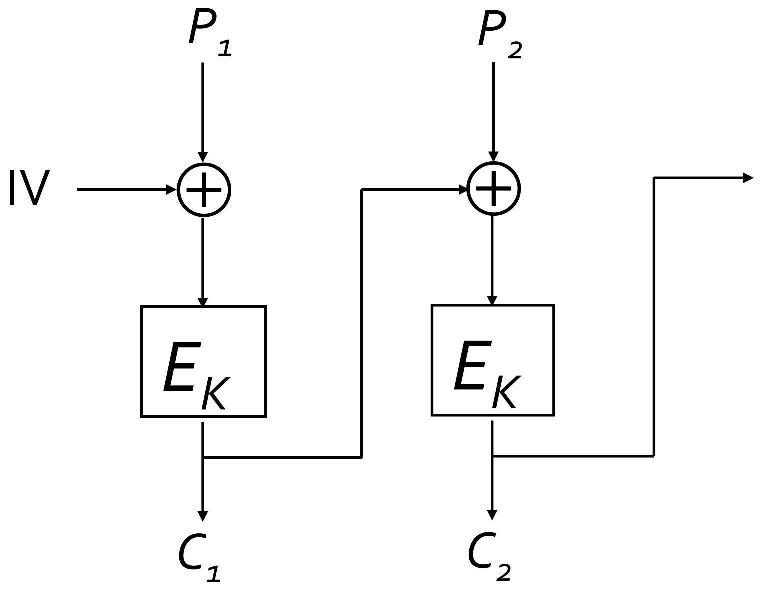
- Suppose an error arises in ciphertext block C_2.



- Then the error propagates to P_3, and P_2 gets randomised.

Further remarks on CBC mode

- IV is needed for decryption; often written as C_0 to emphasize this (ciphertext expansion: ciphertext at least one block larger than plaintext).
- IV needs to be random and unpredictable for each message encrypted (requires a good source of randomness).
- Padding may be needed to make plaintext into a whole number of blocks.



Encryption equation:

$C_0 = \text{IV}$

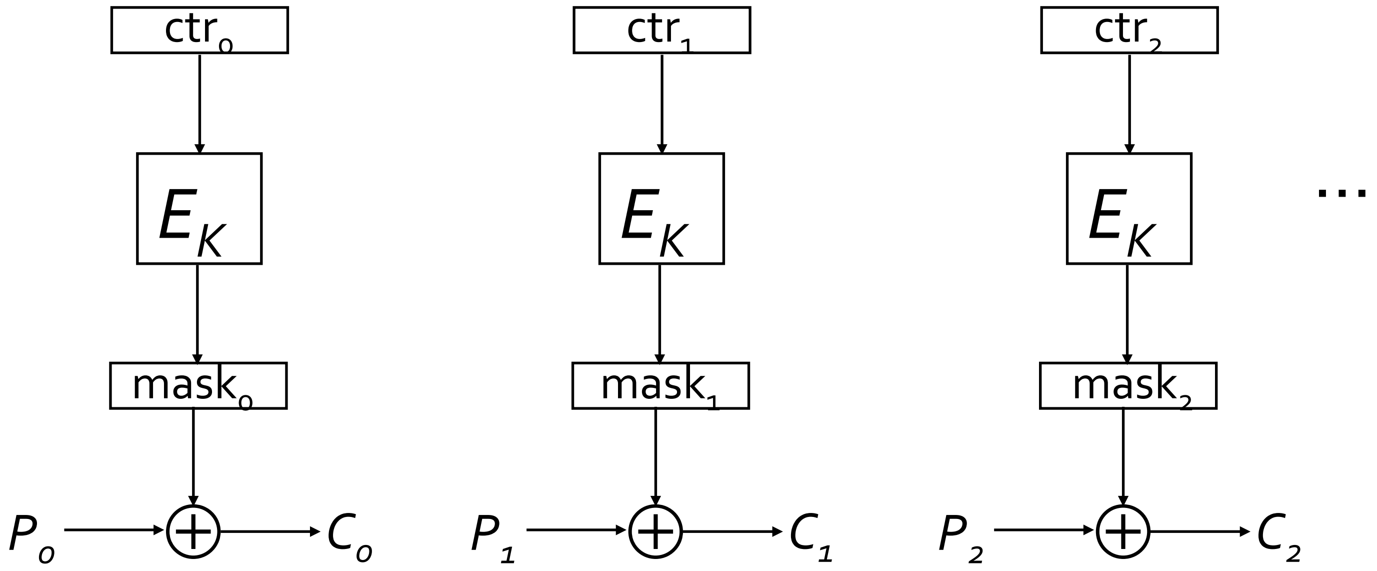
$C_i = E_K(P_i \oplus C_{i-1})$

Decryption equation:

$P_i = D_K(C_i) \oplus C_{i-1}$

Counter Mode (CTR) encryption

- ctr_i is set to be an incrementing counter.
- No padding is needed since last block of output can be truncated to required length.
- Parallelisable, can also pre-compute encryption masks before plaintext known.



Counter Mode (CTR) encryption

- CTR is a stream cipher mode.
  - Turns a block cipher into a stream cipher.
  - Block ciphers usually slower than dedicated stream cipher designs in general, so still a place for stream ciphers in applications.

- Error propagation: a bit-flip in the ciphertext leads to a bit-flip in the plaintext.
  - More generally, XOR of a mask Δ with the ciphertext leads to the same mask Δ being XORed onto the plaintext.
  - So CTR mode does not provide any integrity.
  - Did CBC mode?

Counter Mode (CTR) encryption

- Key security requirement: for a fixed key K, counter values must **not** repeat.
- Similar to keystream repeat issue for a stream cipher: XOR of ciphertexts yields XOR of plaintexts.
- Achieved by:
  - Starting with $ctr_0 = 0$ and changing key for every plaintext (often impractical);
  - **OR** starting with a random value for ctr_0 for each plaintext (requires a good source of randomness, need to avoid collisions);
  - **OR** maintaining a record of the last value of ctr_i used (requires state in encryption algorithm, needs to survive across power cycles).
  - **OR** constructing ctr_i by concatenating a *per plaintext nonce* supplied by the calling application and an internal counter (starting from zero for each new plaintext) (requires per plaintext nonce to not repeat, hence some kind of state needed in application).

Towards security for symmetric encryption

- These modes are instances of *symmetric encryption schemes*
- Before defining security, we first need a general definition for symmetric encryption.
- Then we need to ask: what is our security target? Possibly confidentiality, but not integrity for CTR and CBC modes.
- What resources does the attacker have? Ciphertext only? Chosen plaintext? Chosen ciphertext + decryption capability?
- (Security will turn out to rely on pseudorandomness of block cipher.)

Formalising Symmetric Encryption

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

KGen: key generation, usually 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 and may be stateful (cf. CBC mode, CTR mode).
- Dec may be stateful, usually not randomised.
- In reality, there will be a maximum plaintext length that can be encrypted by a given scheme.

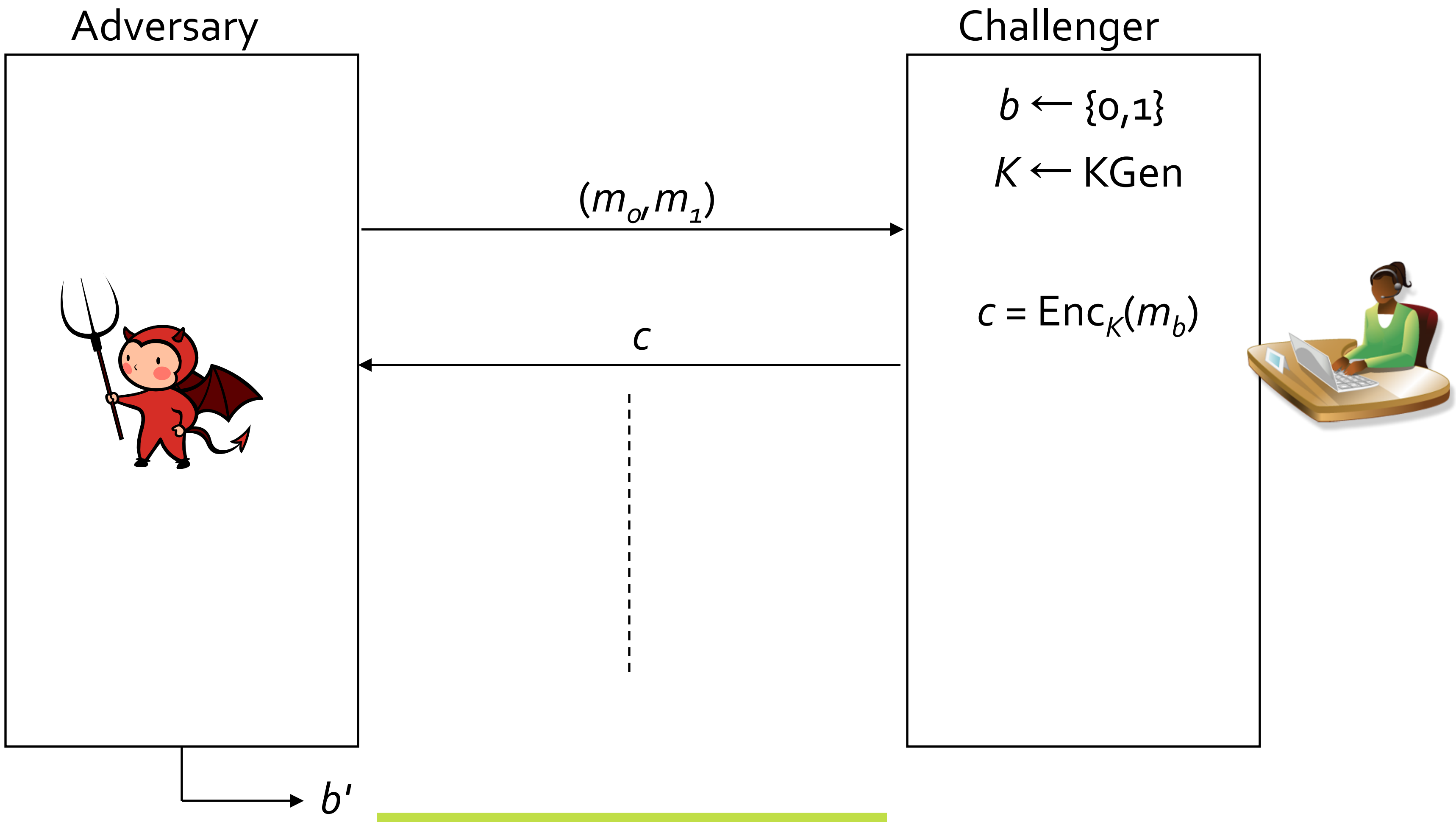
IND-CPA security for SE

- 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 capability "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



Adversary wins if $b = b'$

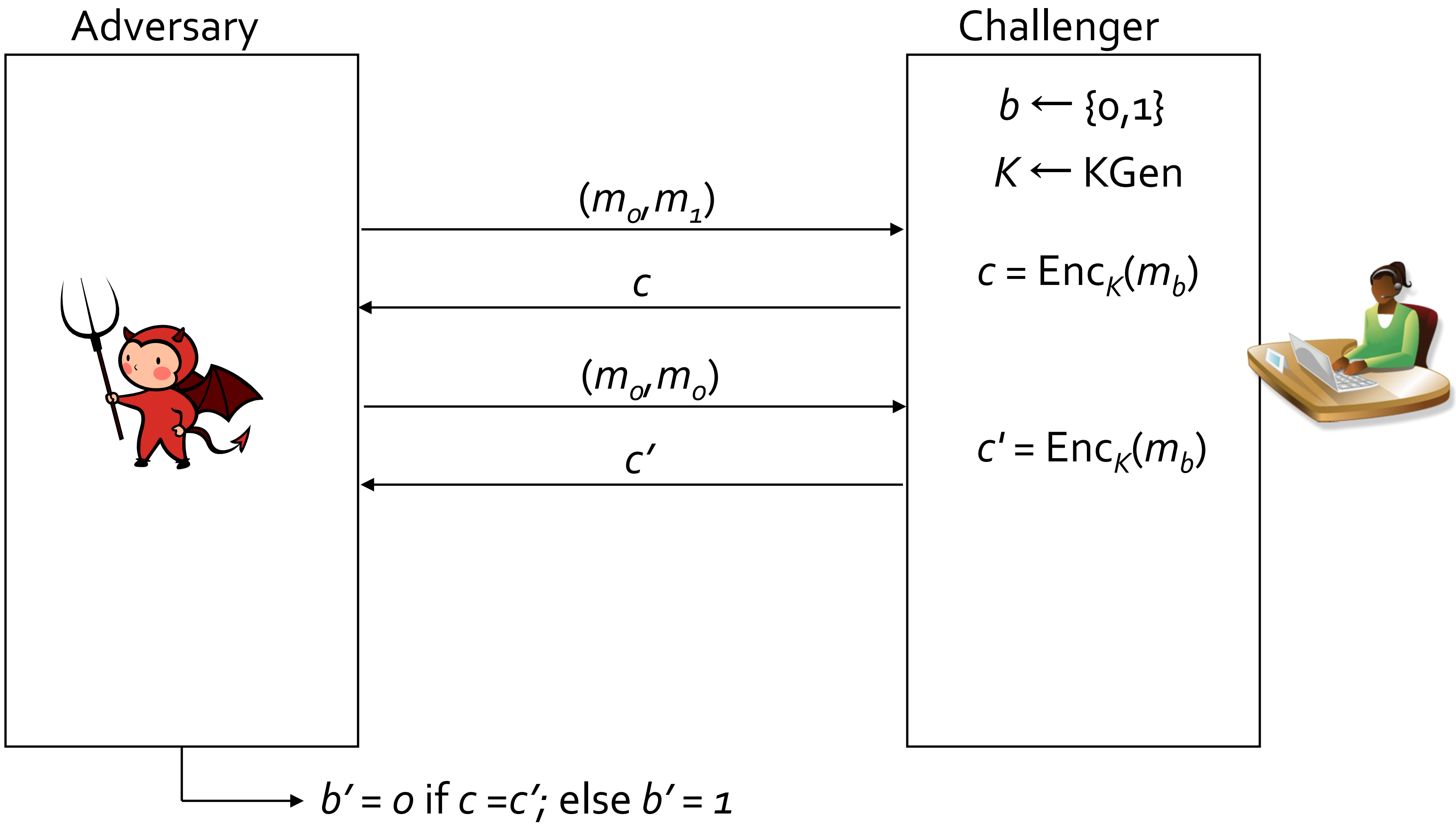
IND-CPA security

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.
- More useful: quantify security of any scheme in terms of resources consumed by adversary and any assumptions on underlying components, e.g. a block cipher.
    - Number of queries to encryption oracle, q, number of bits queried across all queries, running time.

IND-CPA security + stateless, deterministic encryption



$b' = 0$ if $c = c'$; else $b' = 1$

Conclusion: Enc cannot be both stateless and deterministic

IND-CPA security

Informally, IND-CPA security is a *computational* version of perfect security.

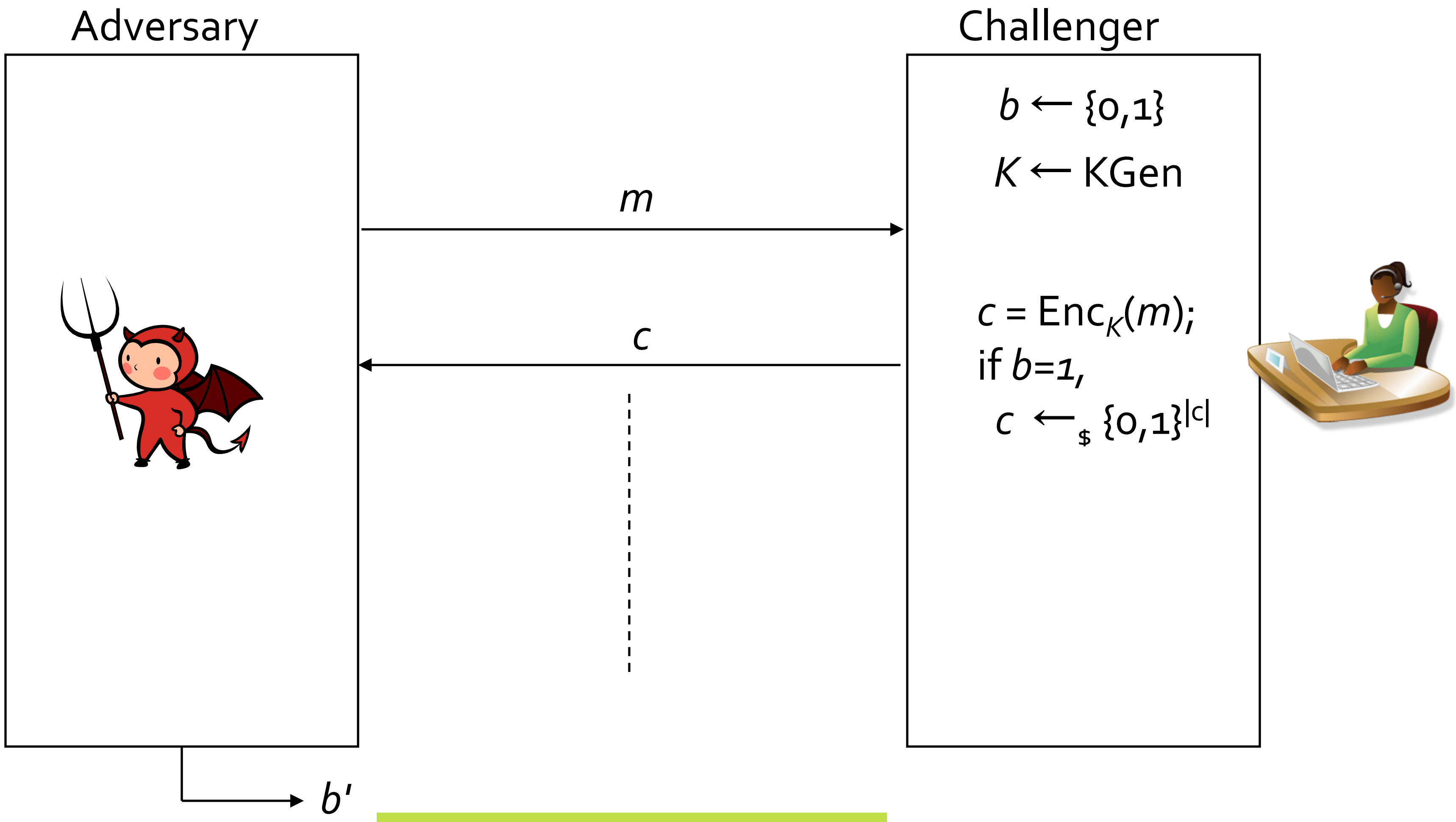
- Ciphertext leaks nothing about the plaintext to a computationally-bounded adversary because adversary can't tell if left or right messages were encrypted.
- Stronger notion than requiring the adversary to recover plaintext.

[BDJR97] developed a variety of equivalent notions.

- RoR-CPA, FtG-CPA and SEM-CPA.

IND$-CPA: even stronger notion than IND-CPA: adversary has encryption oracle and gets back either real encryption or random bits.

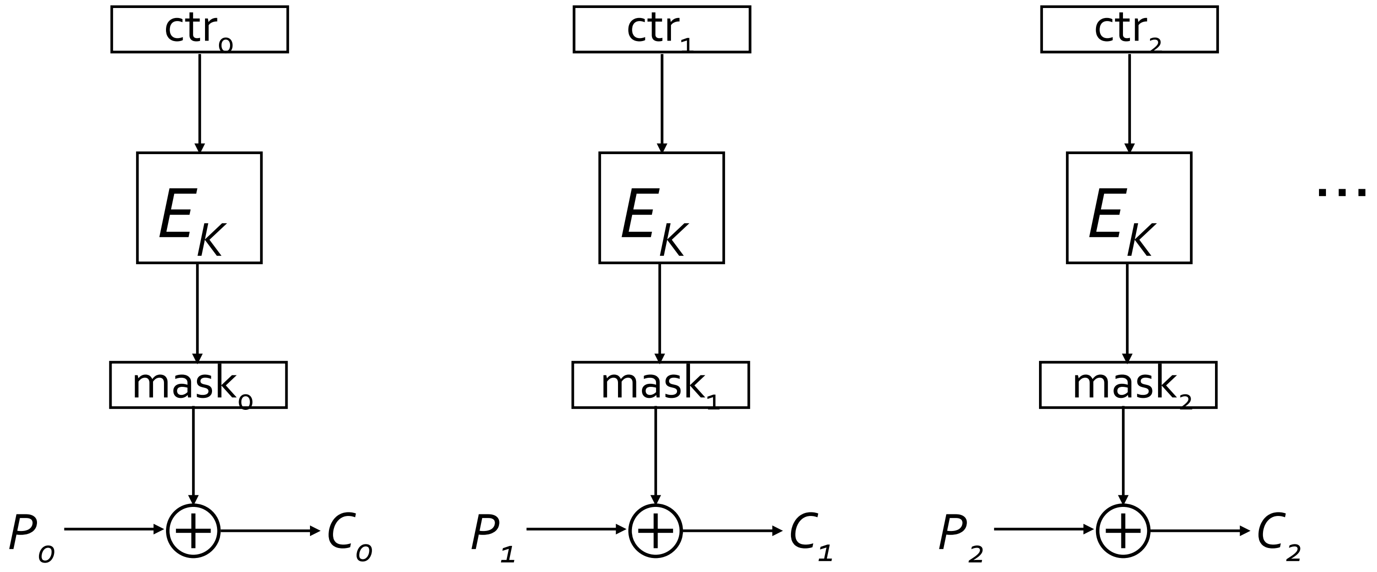
IND$-CPA security in a picture



Adversary wins if $b = b'$

IND$-CPA security of CTR mode (informal)

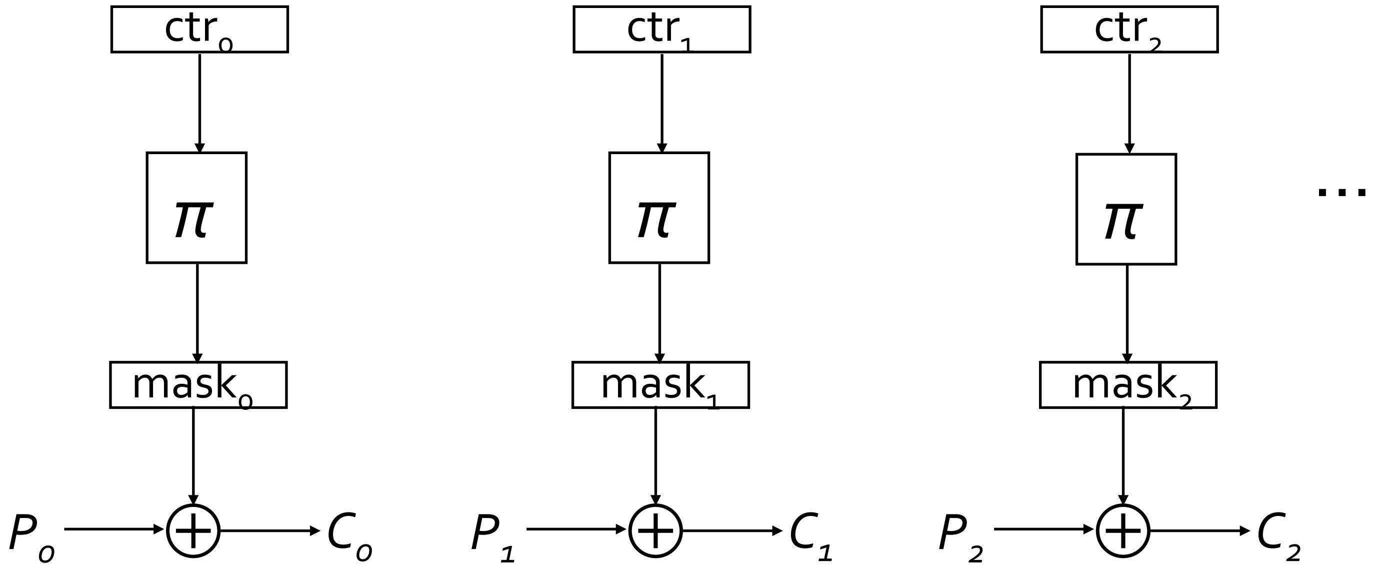
***b* = 0**



- Picture shows part of encryption of just one message $m = P_0 P_1 \ldots P_{t-1}$

IND$-CPA security of CTR mode (informal)

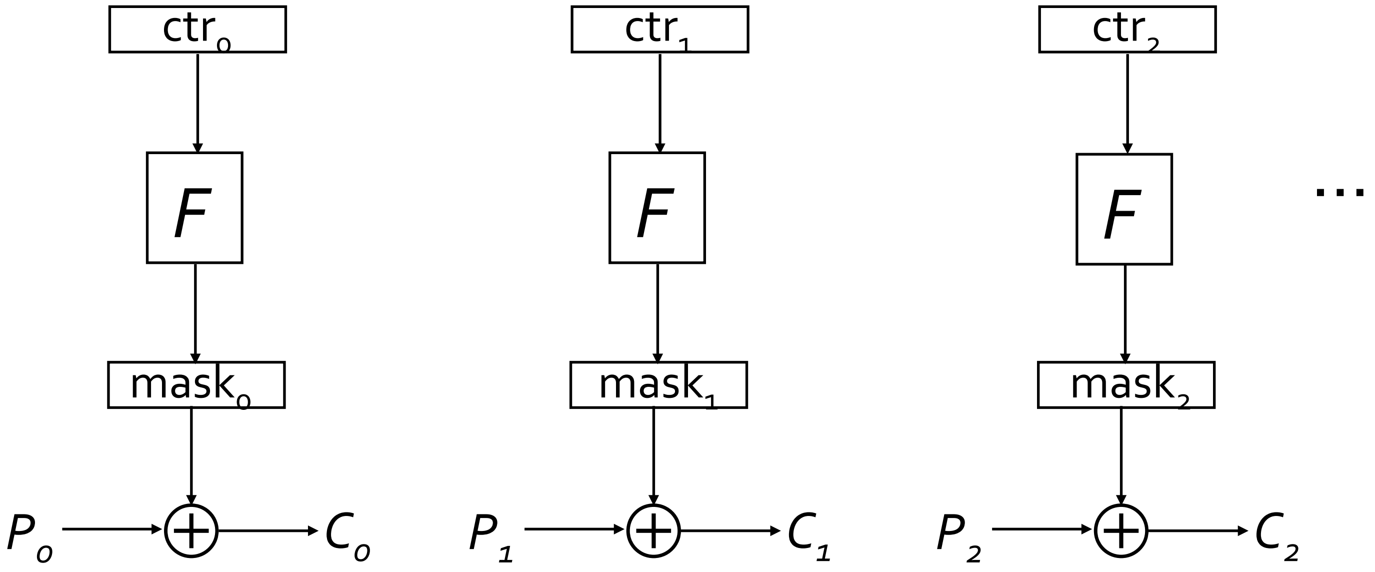
Switch to random perm



- Indistinguishable to adversary by PRP security of *E.*

IND$-CPA security of CTR mode (informal)

Switch to random function

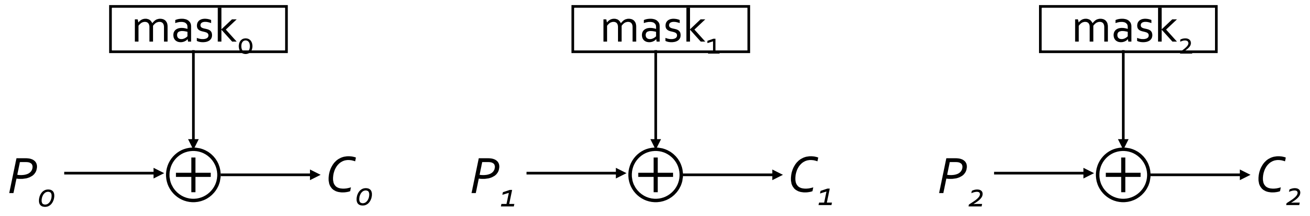


- Indistinguishable to adversary by PRP/PRF switching lemma.

IND\$-CPA security of CTR mode (informal)

Switch to random outputs

$\text{mask}_i \leftarrow_\$ \{0,1\}^n$ for each i



- Indistinguishable to adversary assuming counter values are all distinct.
- This is now just one-time pad encryption!

IND\$-CPA security of CTR mode (informal)

Switch to random ciphertext blocks

$C_i \leftarrow_\$ \{0,1\}^n$ for each i

C_0 C_1 C_2

- Same distribution of ciphertexts by information theoretic security of OTP.
- This is now the $b = 1$ version of the IND\$-CPA game.

IND\$-CPA security of CTR mode (informal)

- We started with the $b = 0$ game and moved one step at a time to the $b = 1$ game.
- At each step, we informally argued why an adversary cannot tell the difference when a small change was introduced.
- The steps rely on the PRP security of E and the PRP/PRF switching lemma.
- More careful accounting will give a concrete bound on any IND-CPA adversary's advantage:
    - One term for advantage of the best adversary against PRP security of E.
    - A second term of the form $q^2/2^n$ where q is the total number of blocks involved in encryption queries.

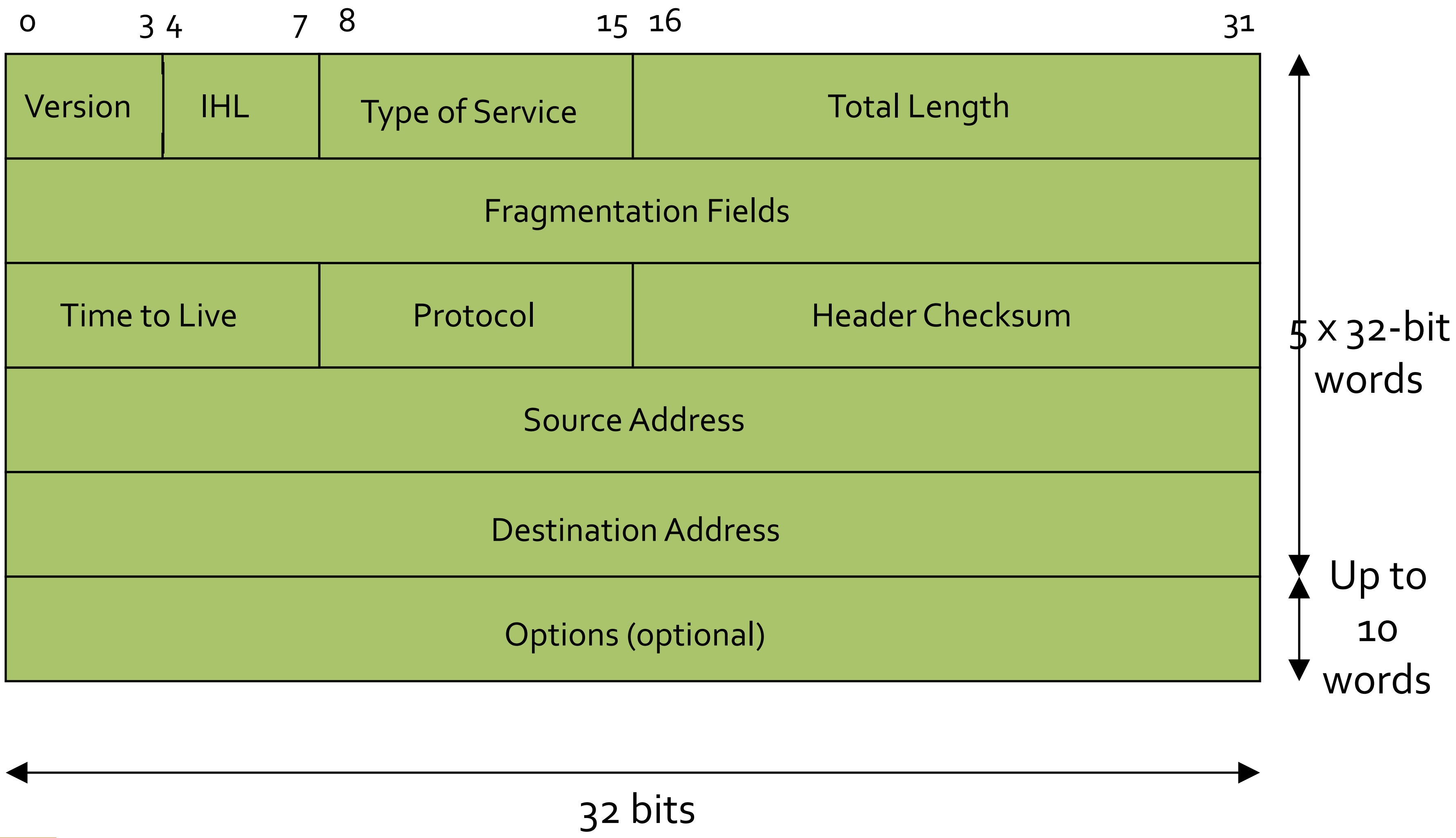
An attack on CBC mode

Attacking Linux implementation of ESP mode IPsec

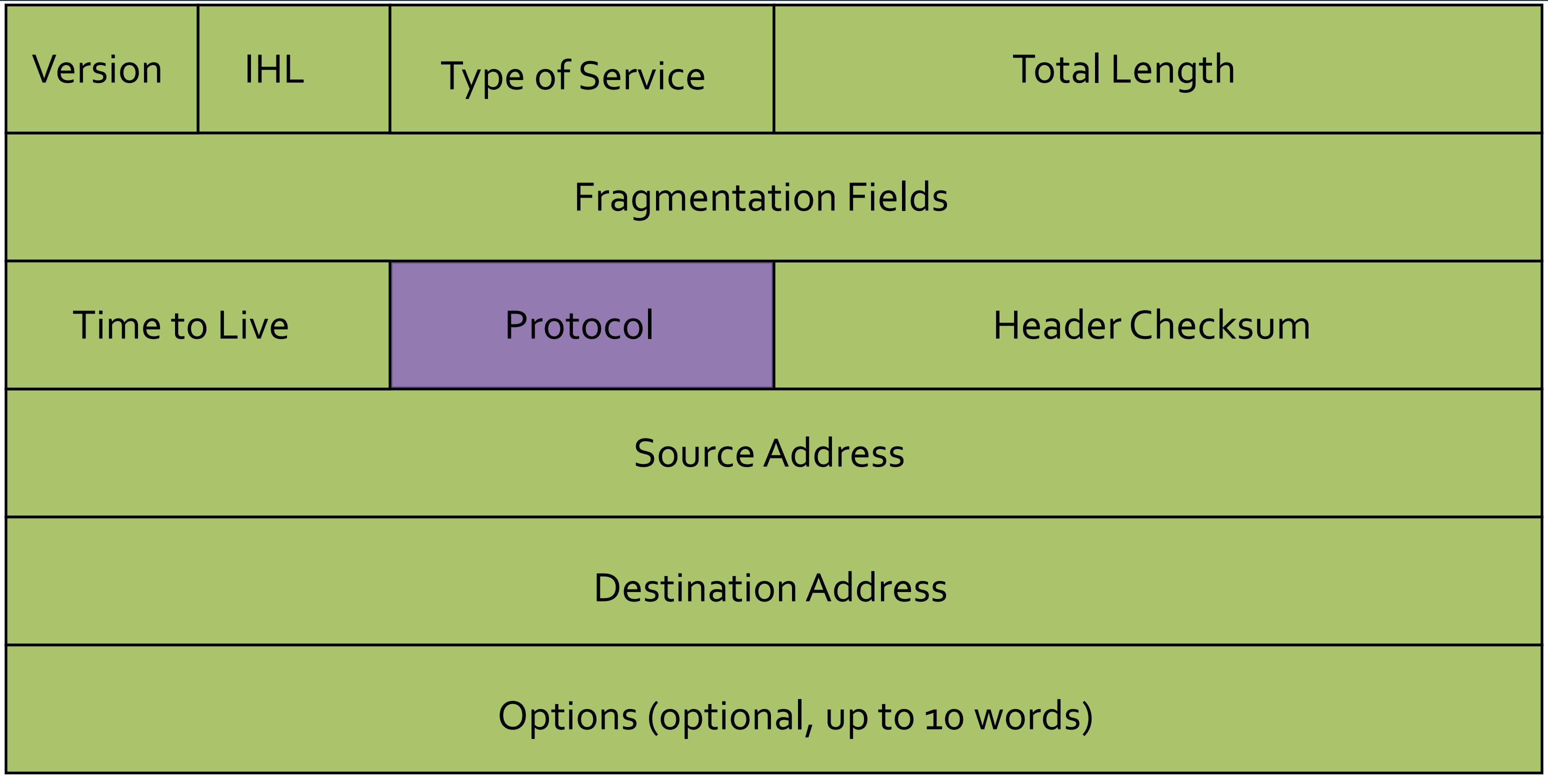
[Paterson-Yau, 2006]:

- Three different (but related) attacks on Linux kernel implementation of encryption-only ESP in tunnel mode.
  - Here, an entire IP packet is encrypted using CBC mode and ciphertext forms payload of a new IP packet.
- Attacks exploit bit flipping in CBC mode.
- Bit flipping results in error messages and packet re-direction.
  - Error messages are carried by ICMP protocol and reveal (some) plaintext data.
  - Packet redirection can send inner packet to attacker's machine.

IP header format



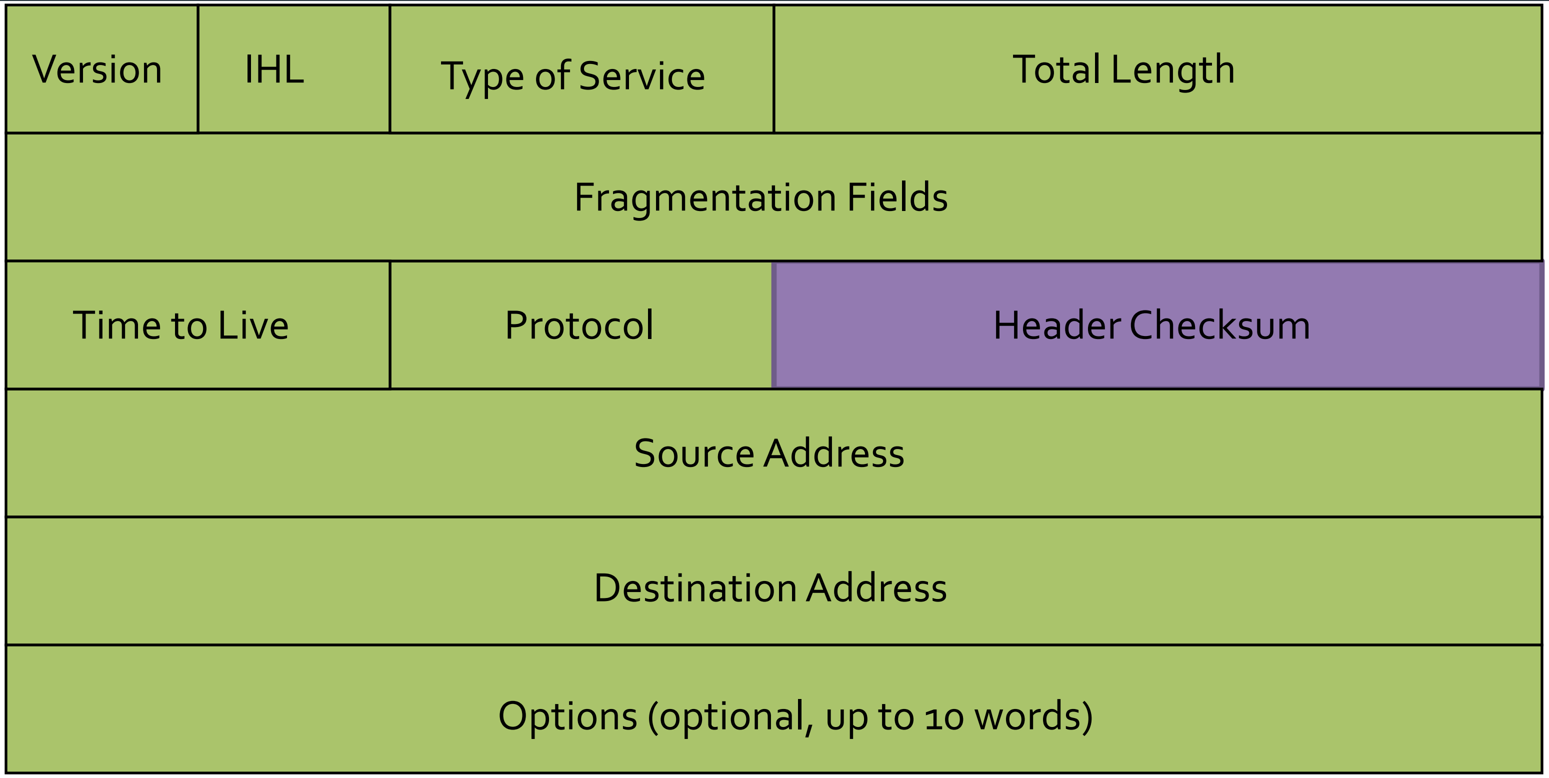
IP header format



Protocol field (8 bits):

- Indicates upper layer protocol in IP payload.
- Possible values are dependent on IP implementation and protocols it supports.
- Typical values: 0x01 for ICMP, 0x06 for TCP, 0x17 for UDP.

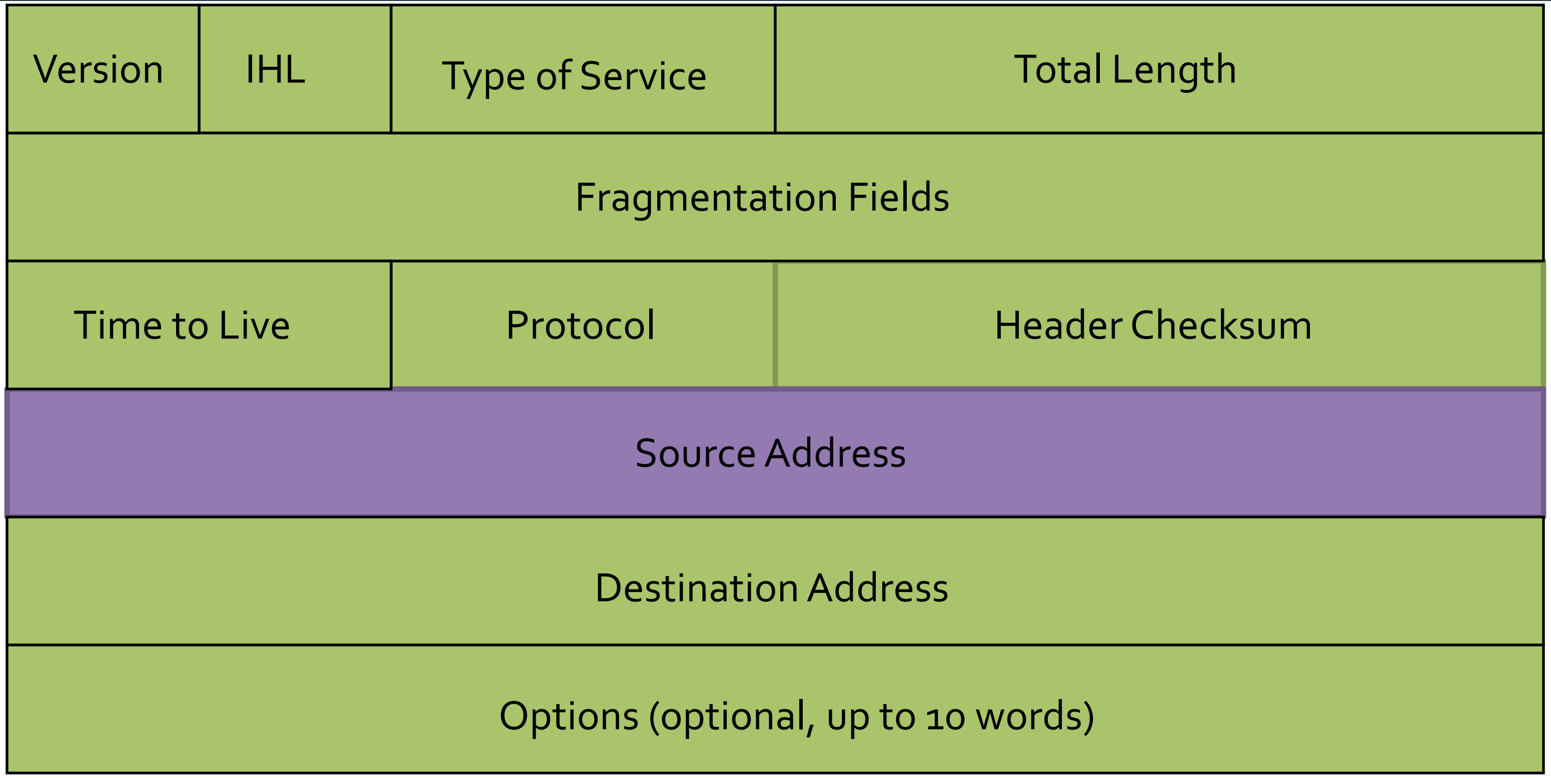
IP header format



Header checksum (16 bits):

- 1's complement sum of 16 bit words in header (inc. any options).
- Incorrect checksum leads to datagram being silently dropped.
- Provides error detection for IP headers.

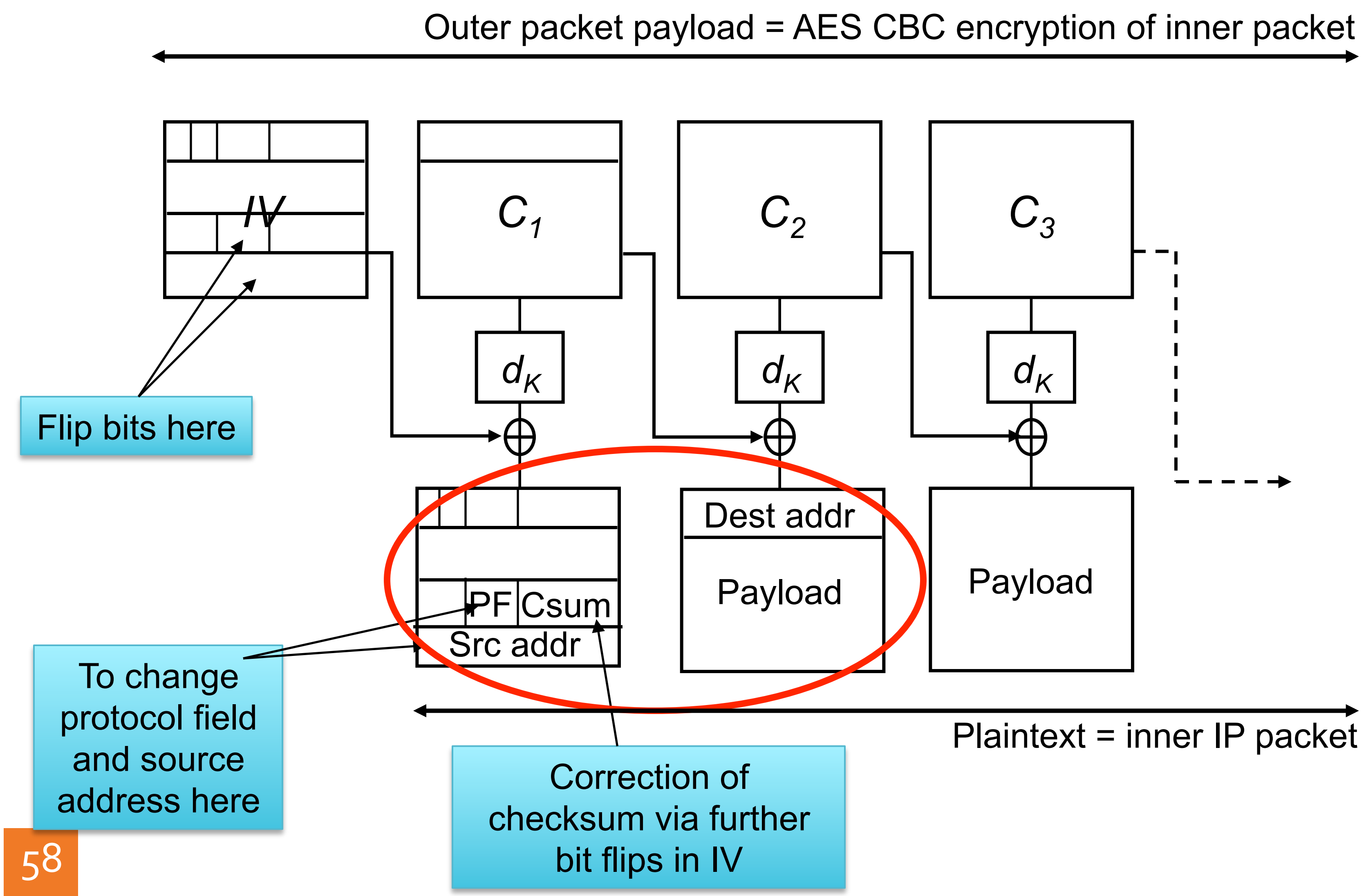
IP header format



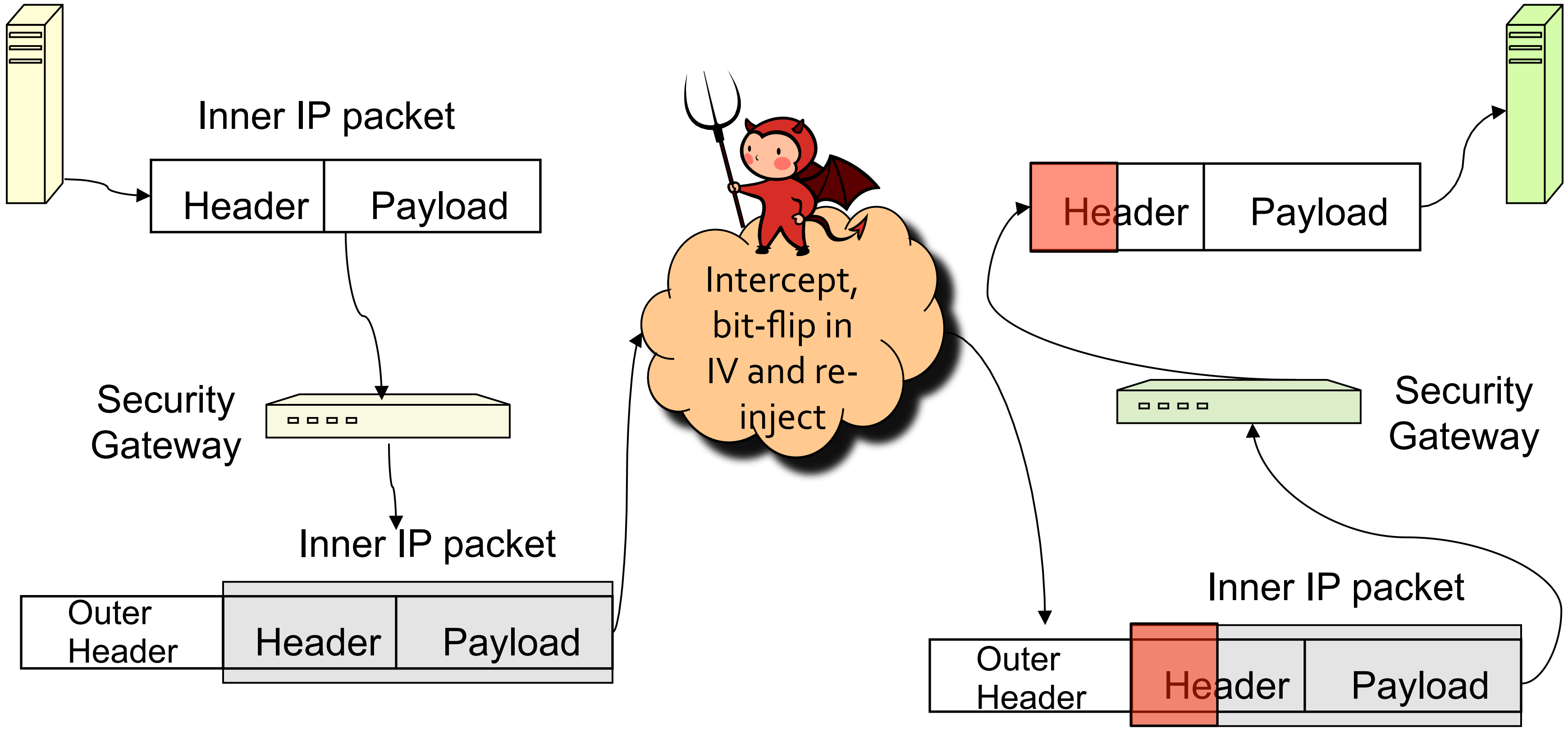
Source Address (32 bits):

- Contains the IP address of the host originating the datagram.
- Needed so any replies or error messages can be delivered back to source.

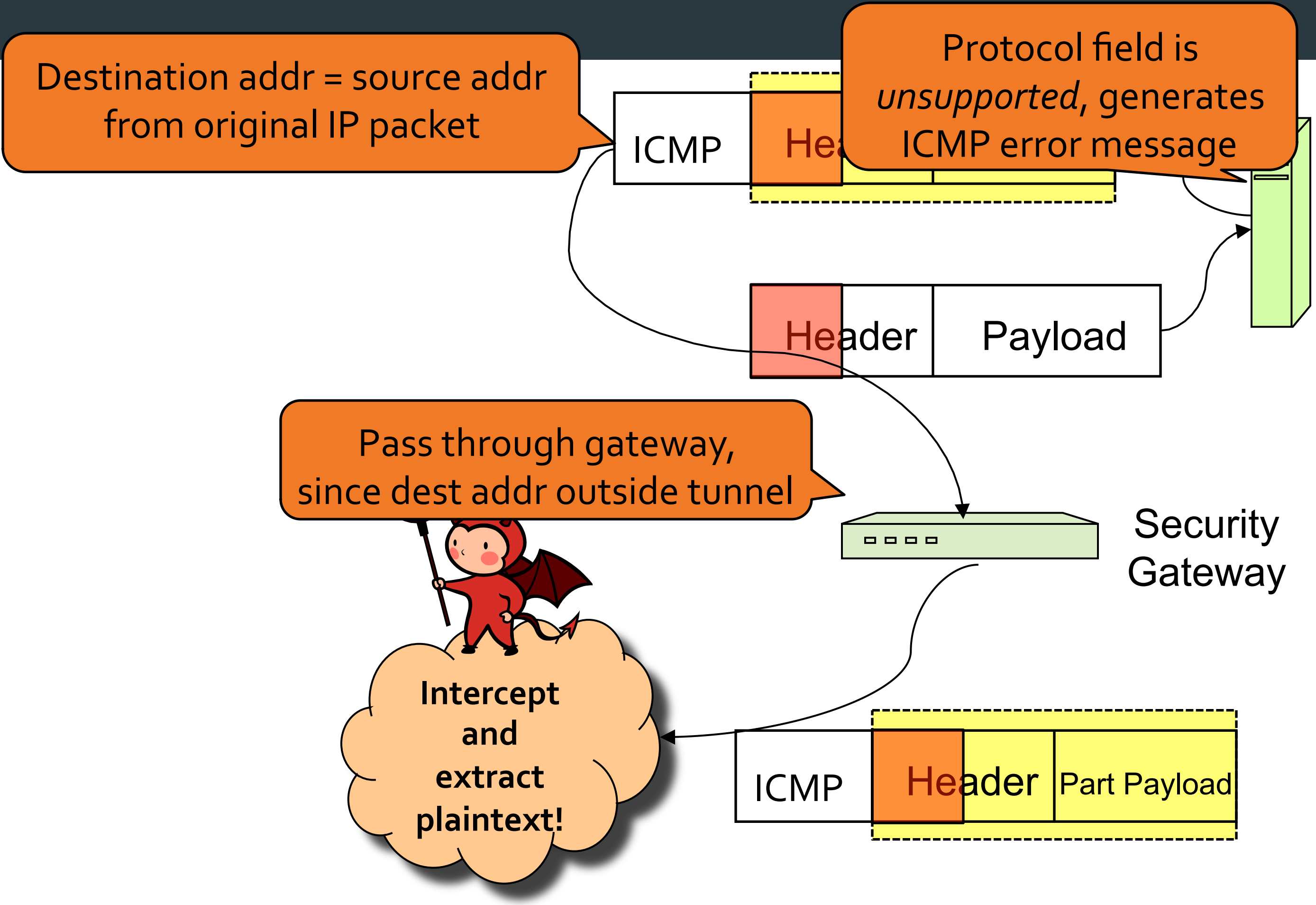
Example attack: protocol field manipulation


Outer packet payload = AES CBC encryption of inner packet
IV
C_1
C_2
C_3
d_K
d_K
d_K
Flip bits here
PF
Csum
Src addr
Dest addr
Payload
Payload
Plaintext = inner IP packet
To change protocol field and source address here
Correction of checksum via further bit flips in IV

Attack visualisation



Attack visualisation


Destination addr = source addr from original IP packet
Protocol field is *unsupported*, generates ICMP error message
ICMP
Hea
Header
Payload
Pass through gateway, since dest addr outside tunnel
Security Gateway
Intercept and extract plaintext!
ICMP
Header
Part Payload

The attack in words

- Attacker intercepts packet, does bit flipping needed to manipulate protocol field and source address, and to correct checksum.
- Attacker than injects modified datagram into network.
- Inner packet decrypted by gateway and forwarded to end-host.
- End-host generates ICMP “protocol unreachable” message in response to modified protocol field in header.

The attack in words

- ICMP payload carries inner packet header and 528 bytes of inner packet's payload.
  - Payload now in plaintext form!
  - ICMP message is sent to host indicated in source address
  - And we have modified this address so that ICMP message does not pass through IPsec tunnel.
- Attacker intercepts ICMP message to get plaintext bytes.
- These ideas were used in [PY06] to build an attack client that can efficiently extract *all* plaintext from an IPsec encryption-only tunnel.

Integrity notions for symmetric encryption

Motivating stronger security

In CBC and CTR modes, we've seen how 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 Δ to change the underlying plaintext from *p* to *p* XOR Δ .
- CBC mode: see IPsec attack.
- 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

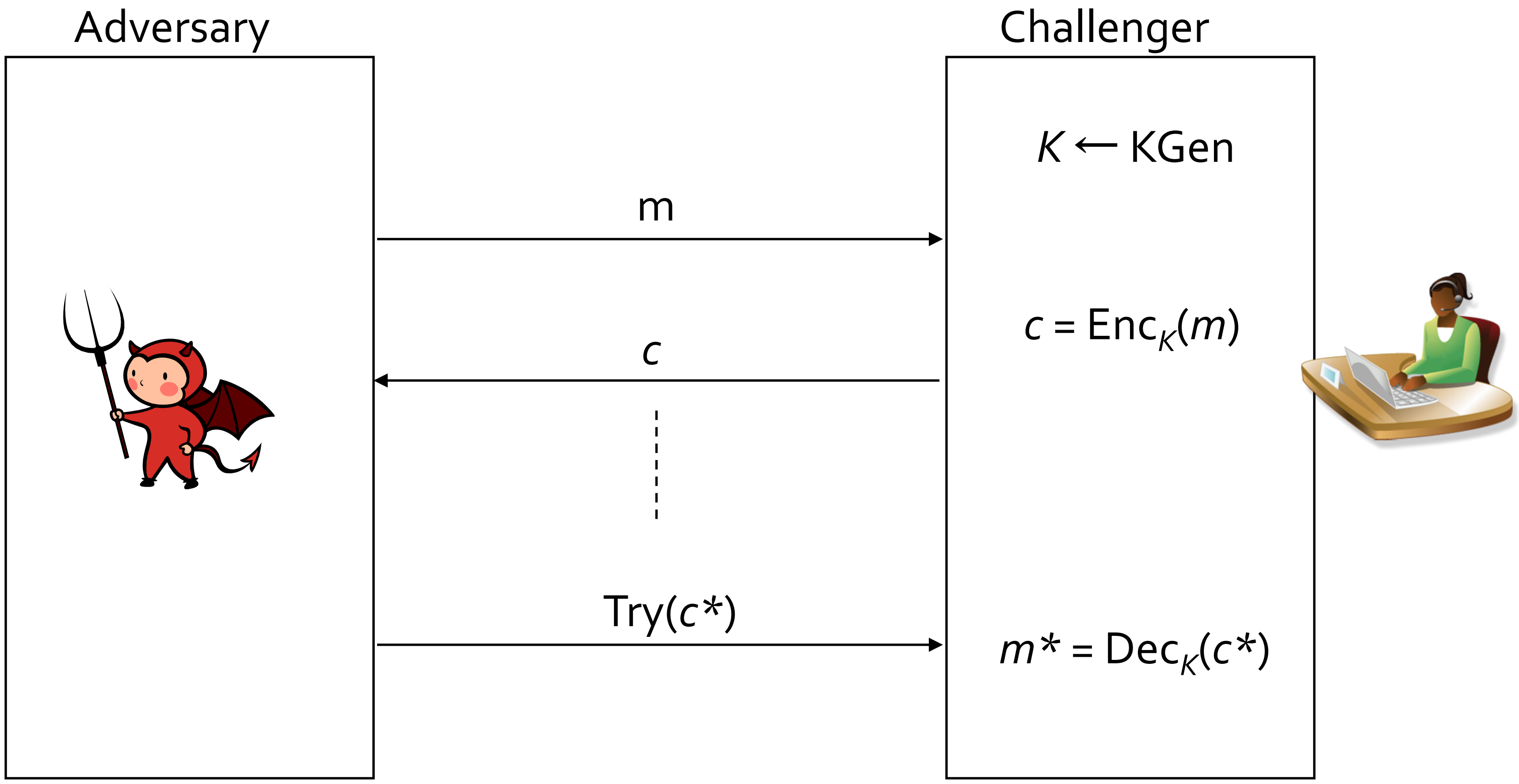
- These kinds of attack do not break IND-CPA security, but are clearly undesirable if we want to build secure channels.
    - A modified plaintext may result in wrong message being delivered to an application, or unpredictable behaviour at the receiving application.
- We really want some kind of *non-malleable*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

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 in a picture



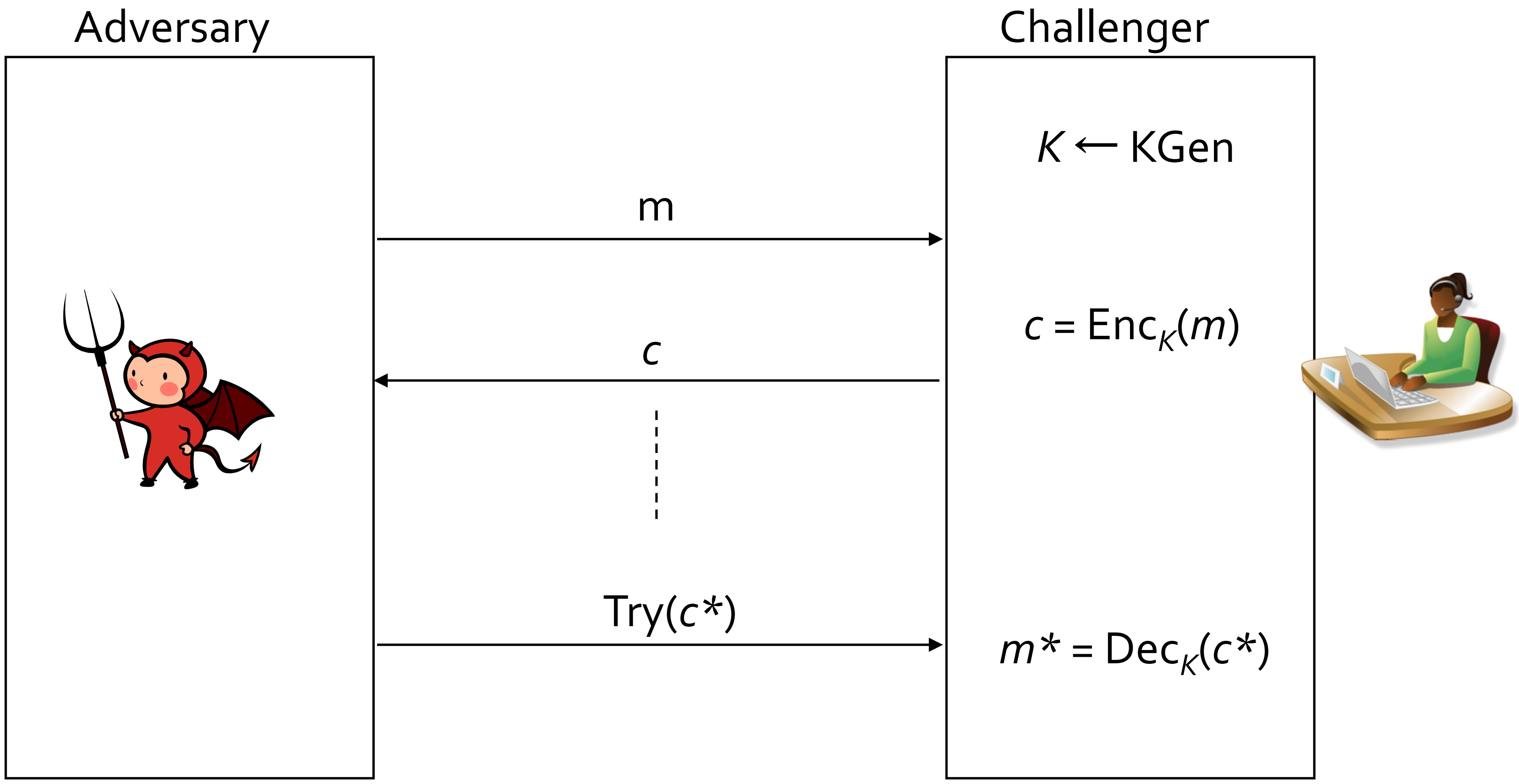
Adversary wins if c* is "new" and $m^* \neq \perp$

INT-CTXT security

- A symmetric encryption scheme is said to provide INT-CTXT security if the success probability of any *efficient* adversary using *reasonable* resources is small.
- Again, this can be made concrete.

- 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.**

INT-PTXT security in a picture



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

Achieving INT-CTXT and INT-PTXT security

- INT-CTXT and INT-PTXT security can be achieved by carefully combining modes (e.g. CTR, CBC) with Message Authentication Codes (MACs).
- More on this soon!

Authenticated Encryption Security

AE Security

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:

AE = IND-CPA + INT-CTXT

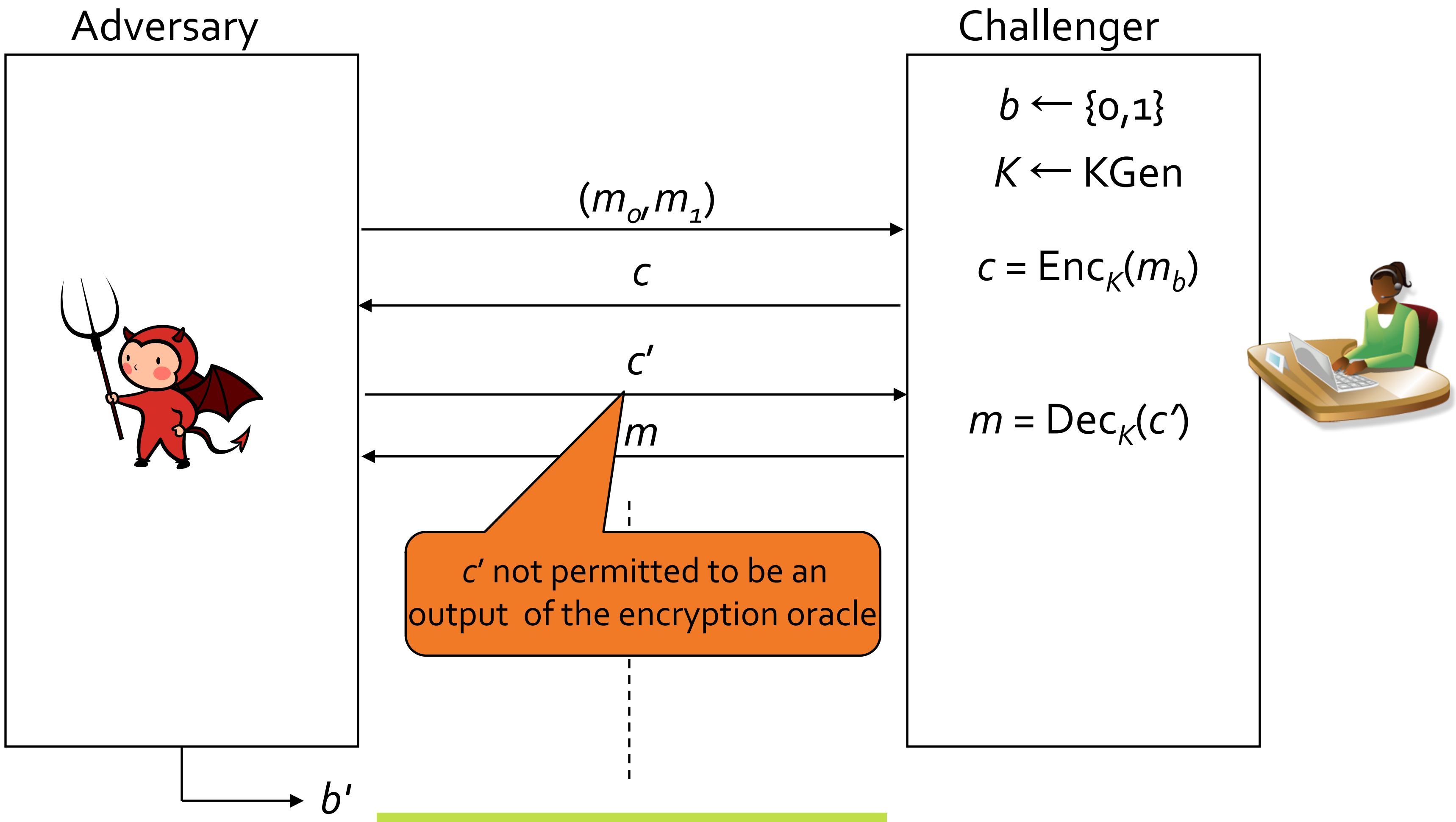
But what about chosen ciphertext attacks?

- We are also interested in security against chosen ciphertext attacks.
- This attack model may actually 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, the ICMP attack on IPsec, etc.

Chosen ciphertext attacks

- IND-CCA security:
  - Attacker now has repeated access to LoR encryption oracle *and to a decryption oracle*.
  - LoR encryption oracle as before.
  - Decryption oracle takes any *c* as input, and outputs $\mathrm{Dec}_K(c)$, which is either a message *m* or a failure symbol $\perp$.
  - Adversary not permitted to submit output of LoR encryption oracle to its decryption oracle.
  - (To prevent trivial win).
- All basic modes of operation are insecure in this model!
  - Why?

IND-CCA security in a picture


Adversary
Challenger
$b \leftarrow \{0,1\}$
$K \leftarrow \mathrm{KGen}$
(m_0, m_1)
c
$c = \mathrm{Enc}_K(m_b)$
c'
m
$m = \mathrm{Dec}_K(c')$
c' not permitted to be an output of the encryption oracle
b'

Adversary wins if $b = b'$

AE security implies IND-CCA security

Informal reasoning:

- Suppose we have a successful IND-CCA adversary against an AE-secure scheme.
- Its decryption oracle is only any use to it if it can come up with a new and valid ciphertext *c** not output by the encryption oracle.
- But if it can come up with a new ciphertext *c**, then it has broken INT-CTXT security!
- So we can assume the adversary never comes up with a valid *c*.*
- This means we can always reply with "⊥" to any decryption query.
- This means the IND-CCA adversary is effectively reduced to being an IND-CPA one.
- But this contradicts AE security, since AE security implies IND-CPA security.

Relations between security notions for symmetric encryption

$$\begin{tikzcd}
 & \boxed{\begin{array}{c}\text{AE:}\\ \text{IND-CPA}\\ \text{+ INT-CTXT}\end{array}} \arrow[dl, shift right] \arrow[dr, shift left] & \\
\boxed{\text{IND-CCA}} \arrow[ur, shift right, "/" marking] \arrow[rr, shift left, "/" marking] \arrow[d, shift right] & & \boxed{\begin{array}{c}\text{IND-CPA}\\ \text{+ INT-PTXT}\end{array}} \arrow[ul, shift left, "/" marking] \arrow[ll, shift left] \arrow[d, shift right] \\
\boxed{\text{IND-CPA}} \arrow[u, shift right, "/" marking] \arrow[rr, shift left, "/" marking] & & \boxed{\text{INT-PTXT}} \arrow[u, shift right, "/" marking] \arrow[ll, shift left]
\end{tikzcd}$$

AE security and beyond

- Because AE security implies IND-CCA security and INT-PTXT security, AE security has emerged as the natural target security notion for symmetric encryption.
- 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.
- The "AE implies IND-CCA" proof only works if there is a single possible error message.
    - See [BDPS13] for development of models and relations in this setting (which is important for practice).

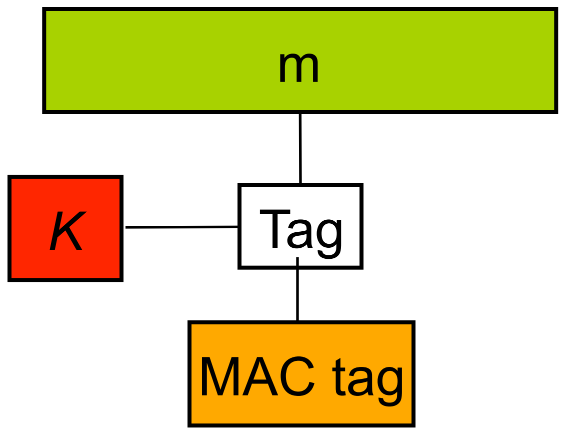
Towards AE security: MACs

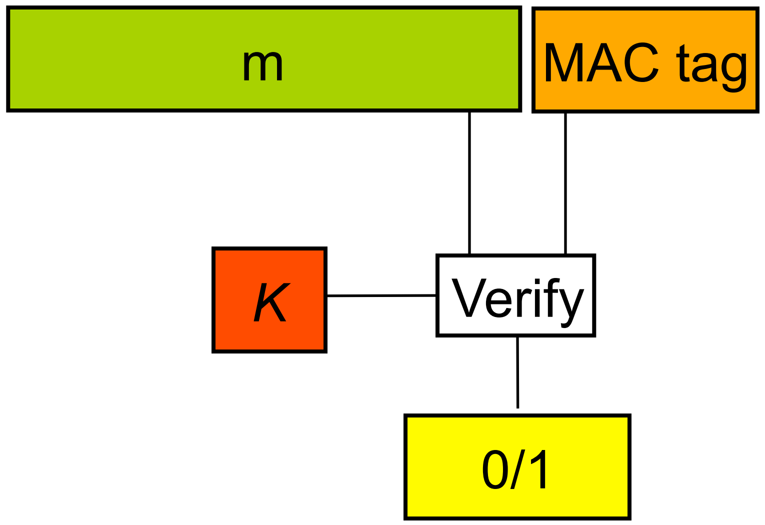
- We will use MACs to achieve AE security.

- MACs provide authenticity/integrity protection for messages.
  - Symmetric analogue of a digital signature.

- Syntax: MAC = (KGen,Tag,Verify).
  - KGen outputs key *K.*
  - Tag has as input a key *K*, a message *m* of arbitrary length, and outputs a short MAC tag τ.
  - Verify has as input a key *K*, a message *m*, a MAC tag τ and outputs 0 or 1, indicating correctness of tag τ for *m* under *K.*

MACs

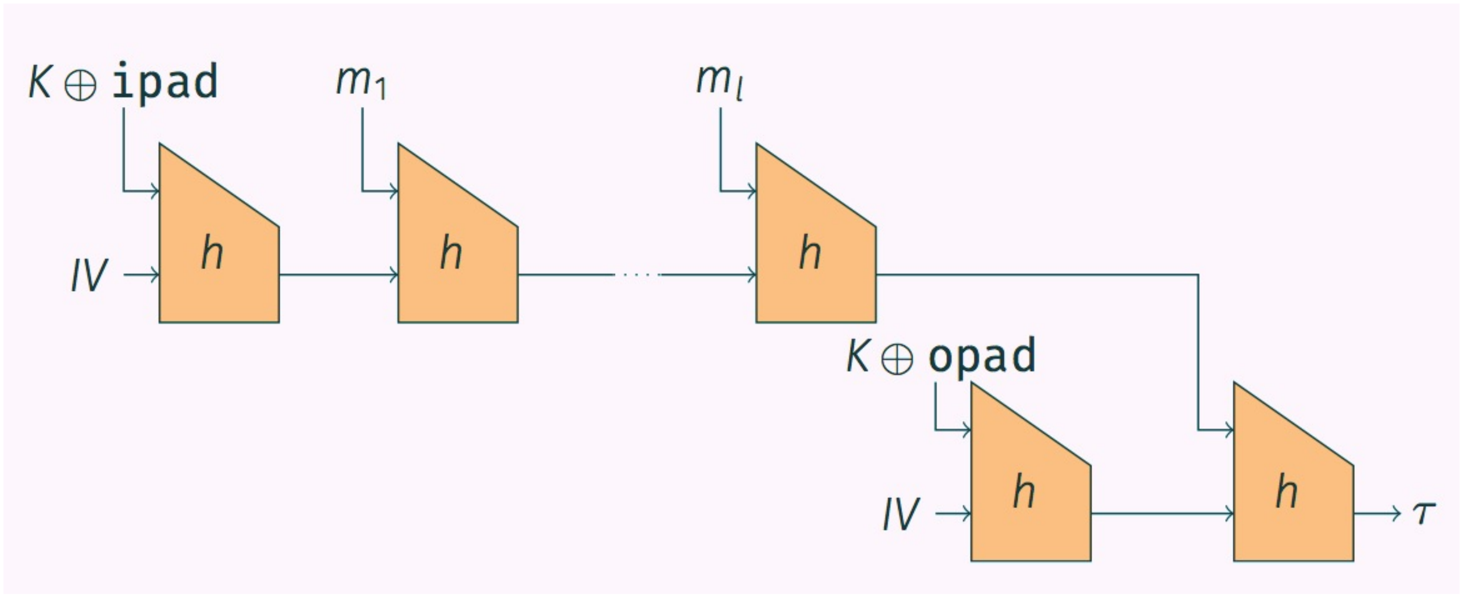




- Key security requirement is *unforgeability*.
- Having seen MAC tags for many chosen messages, an adversary cannot create the correct MAC tag for another chosen message.
- Strong and weak forms of unforgeability:
  - New MAC tag on (possibly) queried message versus MAC tag on unqueried message.
  - SUF-CMA and (W)UF-CMA security.

Example: HMAC

HMAC is a general purpose method for building a MAC from a compression function h: $\{0,1\}^k \times \{0,1\}^n \dashrightarrow \{0,1\}^n$.



Tag(K,m) = H((K⊕ opad) || H((K⊕ ipad) || m))

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 (e.g. HMAC).
- Can we combine these to obtain AE security for symmetric encryption?
- 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 $c = (c' \| \tau)$.
- variant 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: EtM

- Generic options: E&M, MtE, EtM.

- EtM gives AE security.
  - 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: E&M

To see why E&M fails to be secure in general:

- Suppose we have a SUF-CMA secure MAC scheme, with tagging algorithm $\text{Tag}_{KM}(.)$.
- Think about the MAC scheme which outputs $\text{Tag}_{KM}(m) \,||\, m_0$ where m_0 is the first bit of *m.*
- Is it SUF-CMA secure?
- What about the security of the resulting E&M scheme?

- Artificial MAC, but consider also natural case when MAC is a PRF.
  - Consider pair of Enc queries (m_0,m_0) and (m_0,m_1).
  - If MAC tag remains the same, then m_0 was encrypted, so guess $b=0$.

Security of generic composition for AE: MtE

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 = CBC-Enc$_{KE}$ (TLS-PAD(m || Tag$_{KM}$(m))).
    - TLS-PAD: add “00”, or “01 01”, or “02 02 02”, etc.
- 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 and recover the plaintext!

- 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 (and 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 should be avoided wherever possible.**

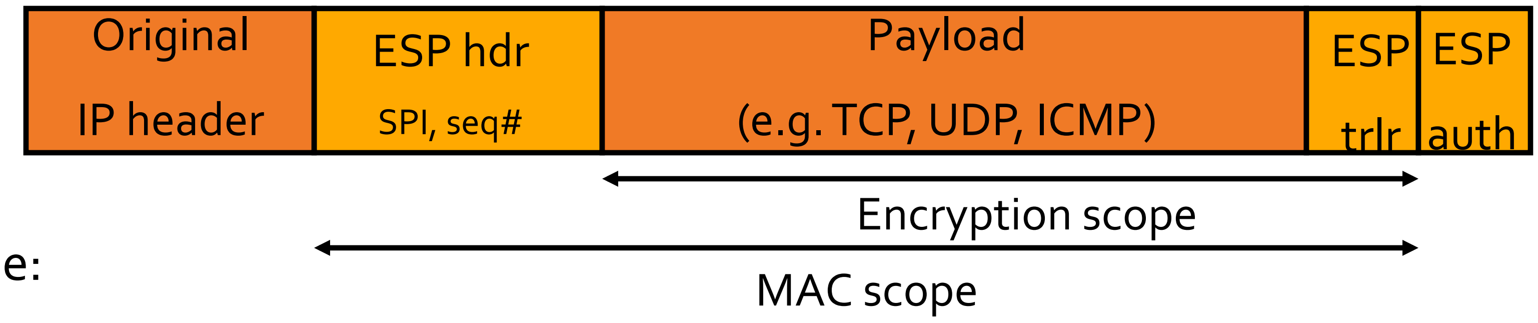
AEAD

Authenticated Encryption with Associated Data (AEAD)

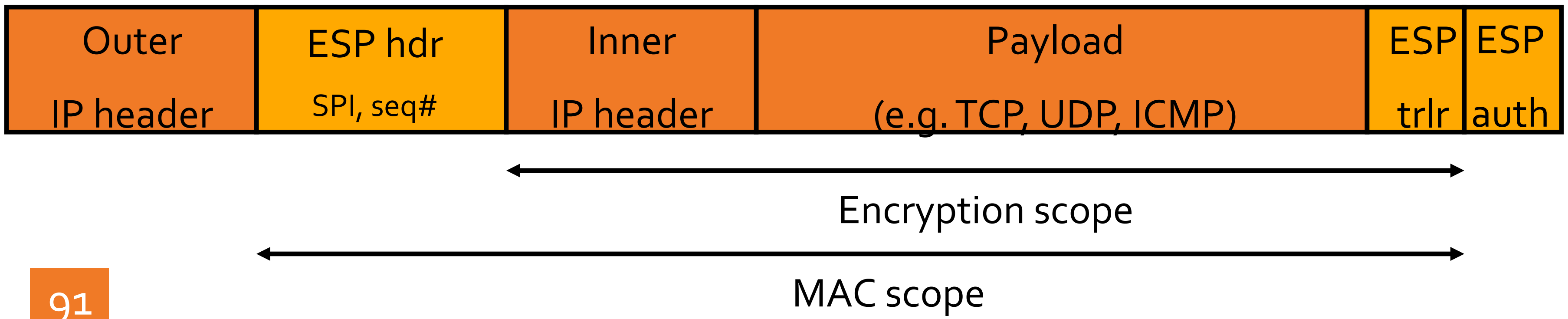
In practical applications, we often require confidentiality and integrity for some data fields and only integrity for others.

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!

Nonce = Number used once.

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).
- Guaranteeing good sources of randomness is hard.
- It may be dangerous to hand this responsibility to the programmer, by asking him/her 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 their 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.*

Using nonce-based AEAD

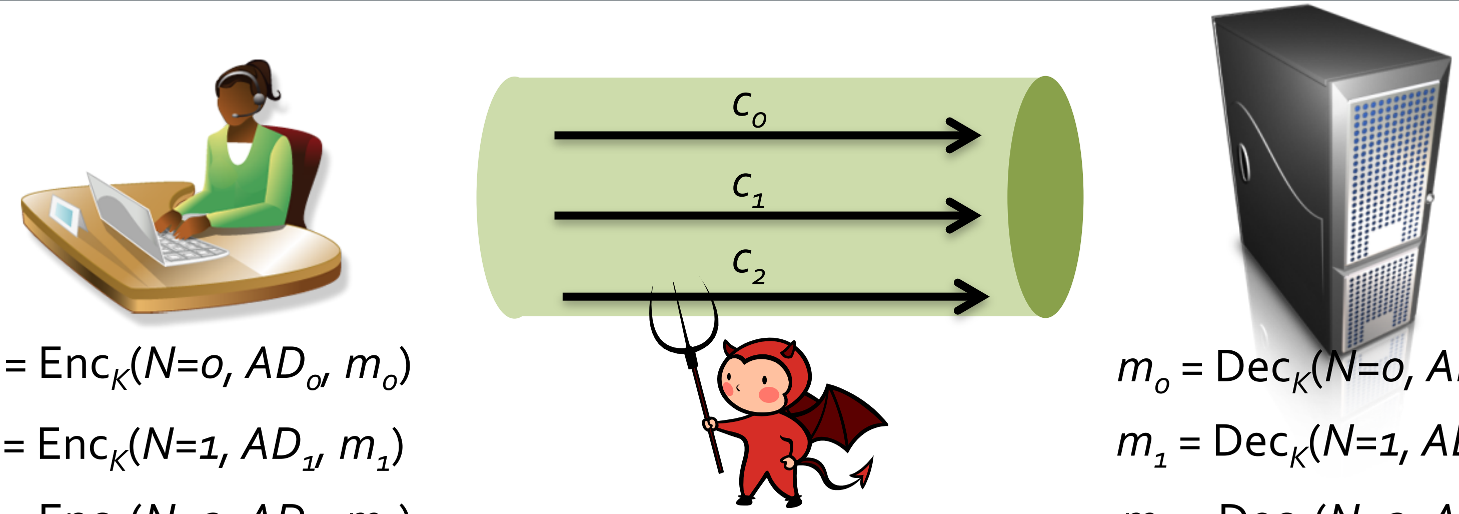
Enc: encryption, takes as input key *K*, **nonce N** $\in \{0, 1\}^n$, associated data *AD* $\in \{0, 1\}^*$, plaintext *m* $\in \{0, 1\}^*$, and produces output *c* $\in \{0, 1\}^*$.

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 = Enc$_K$(N=0, AD_0, m_0)

c_1 = Enc$_K$(N=1, AD_1, m_1)

c_2 = Enc$_K$(N=2, AD_2, m_2)

m_0 = Dec$_K$(N=0, AD_0, c_0)

m_1 = Dec$_K$(N=1, AD_1, c_1)

m_2 = Dec$_K$(N=2, AD_2, c_2)

- A sends B a sequence of messages m_0, m_1, m_2,… using nonce-based AEAD.
- A uses an incrementing counter for the nonces; B uses the same nonce values when decrypting.
- Nonces/counters can be *implicit* or *explicit* on secure channel.

Secure channels from nonce-based AEAD

- 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".
- This gives us a basic secure channel functionality for *atomic* messages from nonce-based AEAD.
- Security captured by **stateful** security notions for SE, introduced by Bellare-Kohno-Namprempre.

Example nonce-based AEAD schemes

Further constructions

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

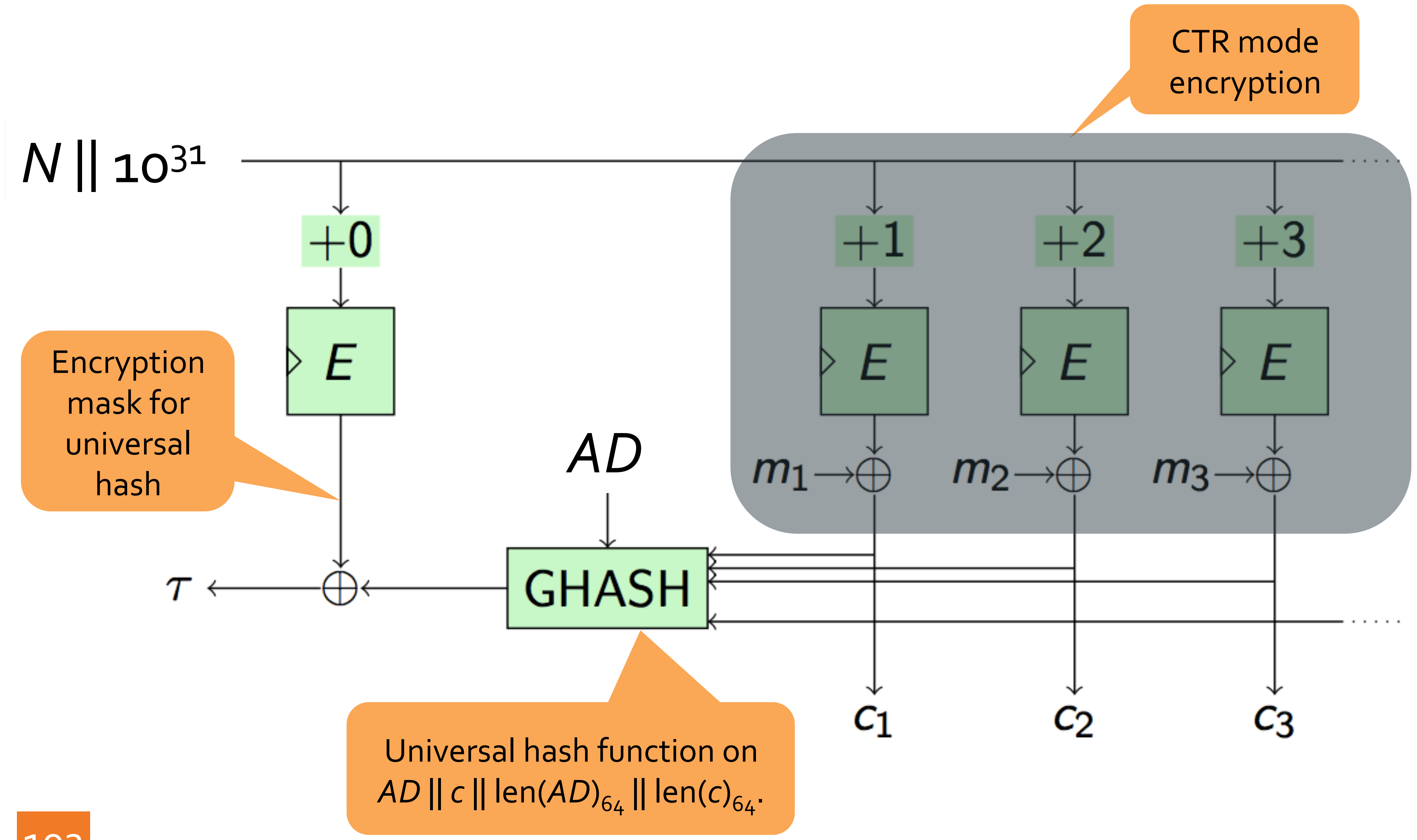
- EtM is the only one that is safe to use.
- EtM extends to the AEAD setting:

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

- But 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 $IV = E_K(N)$ - use key to derive "random" IV block from nonce.

- Many other AEAD schemes are available; we will look at just one, GCM.

GCM (for 96-bit nonces)



Universal hash function on $AD \,||\, c \,||\, \text{len}(AD)_{64} \,||\, \text{len}(c)_{64}$.

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.
- 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.
- GCM is fragile: can fail spectacularly if a nonce is ever repeated.

AEAD ≠ secure channel

AEAD ≠ secure channel

- Recall our application developer:
  - He wants a drop-in replacement for TCP that's secure
  - Actually, he might *just* want to send and receive some atomic messages and not a TCP-like stream
- To what extent does AEAD meet this requirement?
- It doesn't…

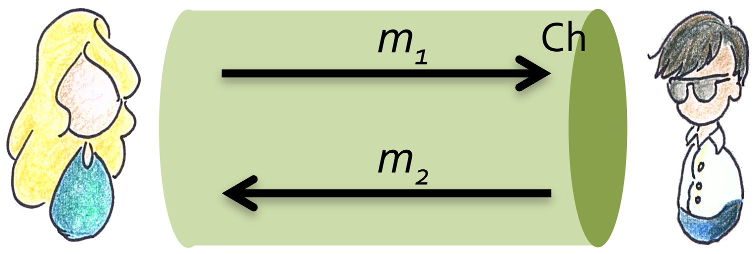
AEAD ≠ secure channel

Enc(.,.,.)

+

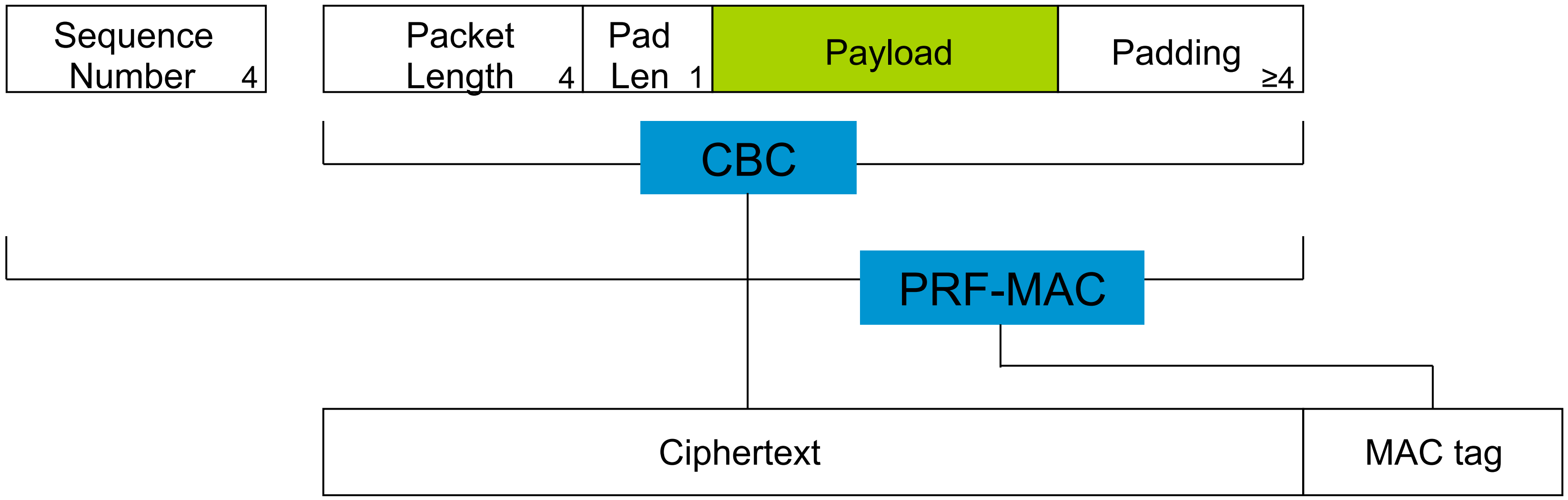
Dec(.,.,.)

MIND THE GAP



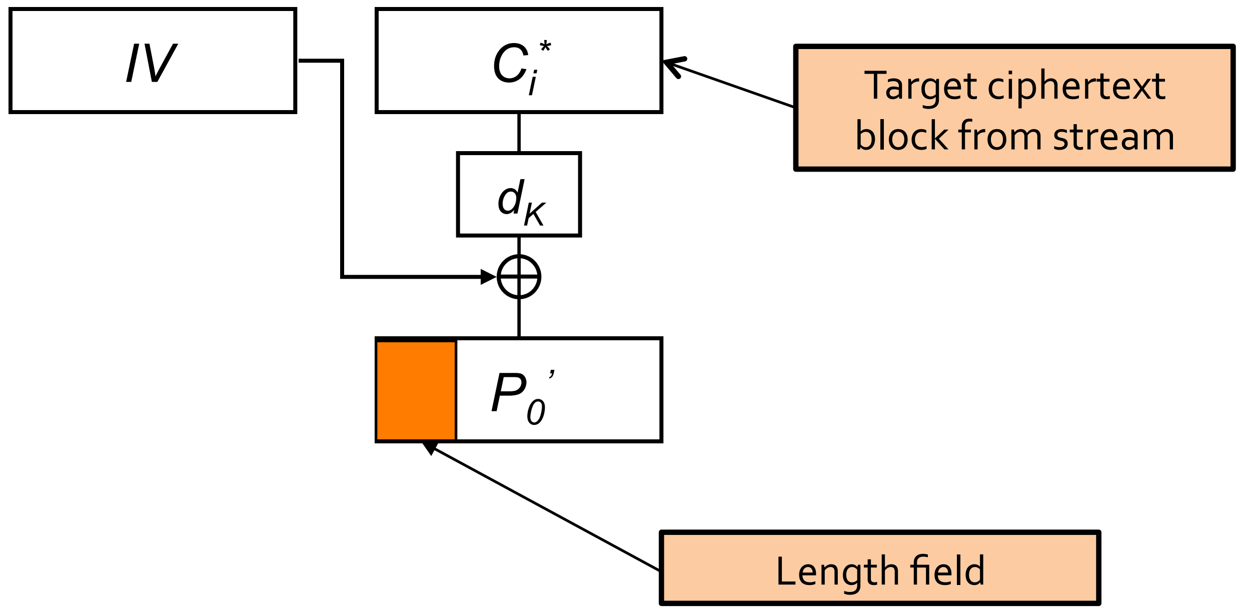
There's a significant semantic gap between AEAD's functionality and raw security guarantees, and all the things a developer might expect a secure channel to provide.

SSH Binary Packet Protocol



- Packet length field measures the size of the packet on the wire
  - Encrypted to hide the true length of SSH packets
- Needs random IV for CBC-mode to prevent chaining attack
- Construction with random IVs was proven to be IND-sfCPA and INT-sfCTXT secure (Bellare-Kohno-Namprempre, 2002)

Breaking SSH (Albrecht-Paterson-Watson, 2009)

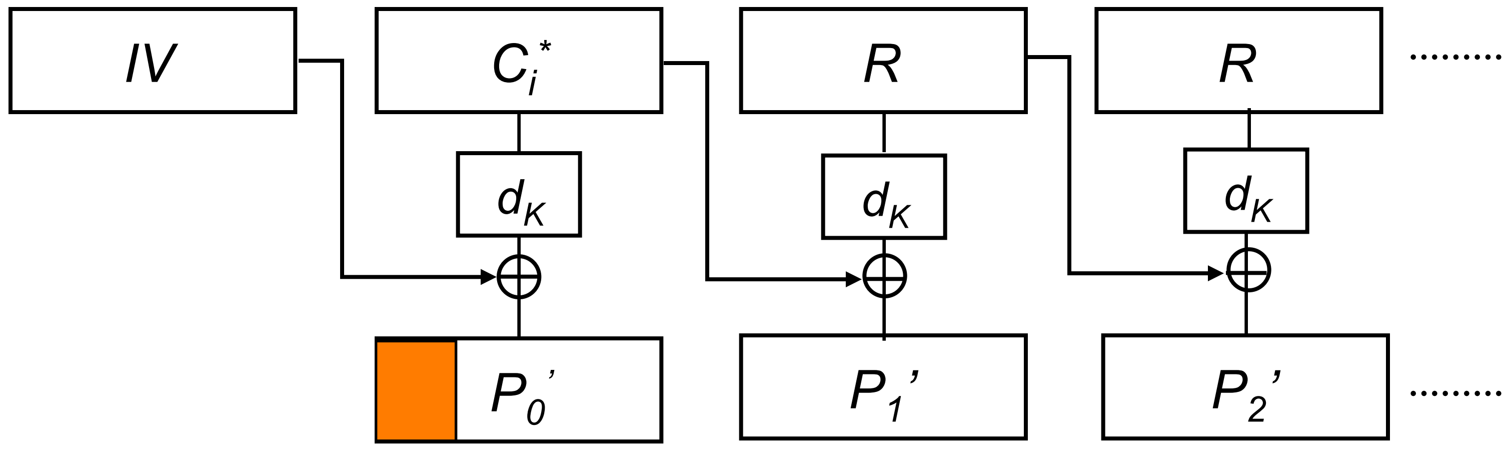


- The receiver will treat the first 32 bits of the calculated plaintext block as the packet length field for the new packet
- Here:

$$P_o' = IV \oplus d_K(C_i^*)$$

where *IV* is known

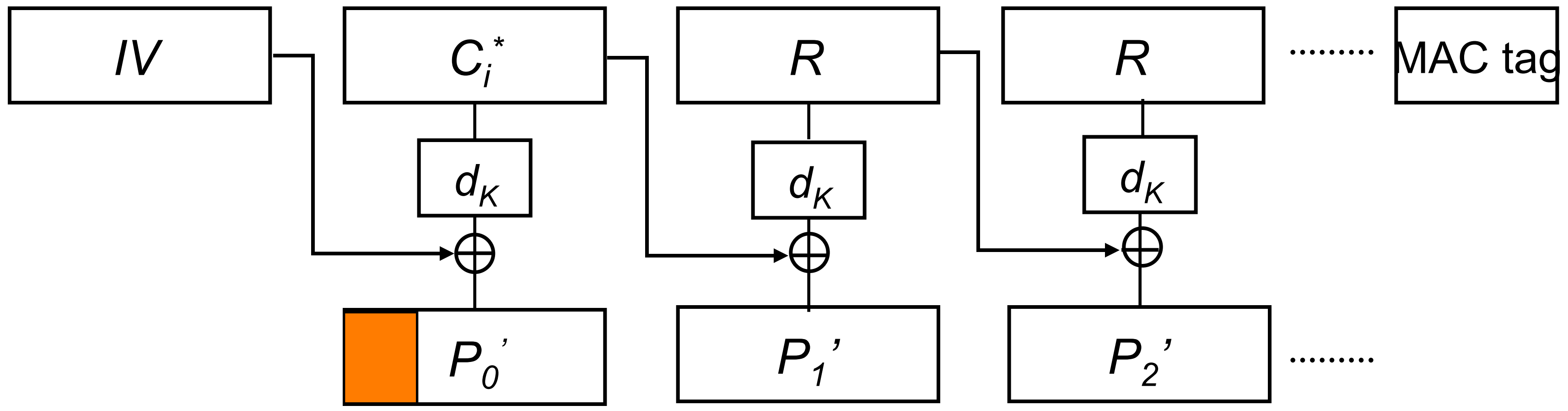
Breaking SSH (Albrecht-Paterson-Watson, 2009)



The attacker then feeds random blocks to the receiver

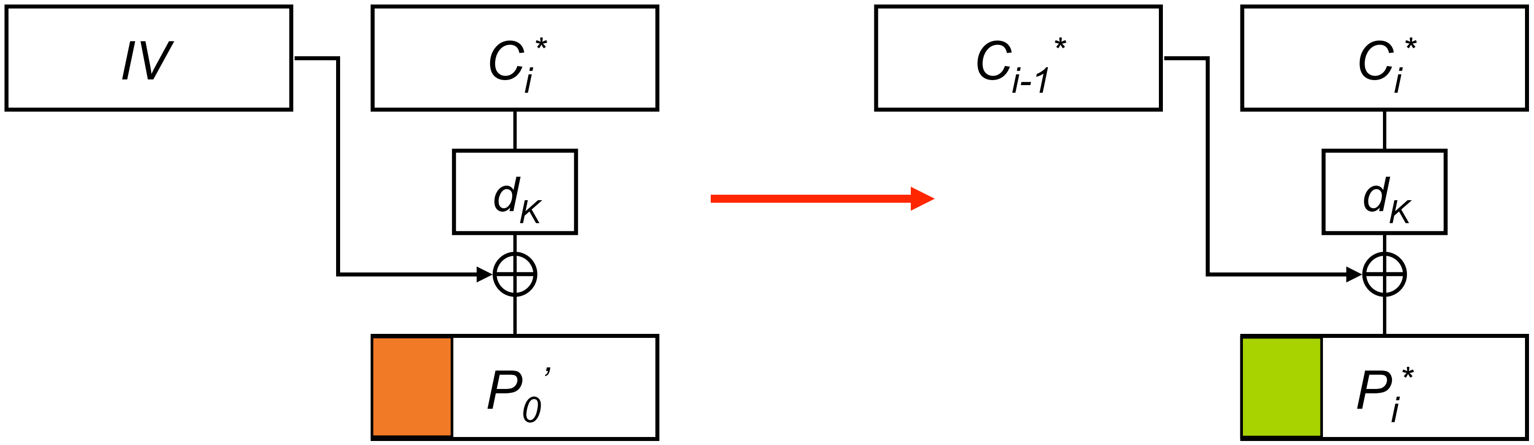
- One block at a time, waiting to see what happens at the server when each new block is processed
- This is possible because SSH runs over TCP and tries to do online processing of incoming blocks

Breaking SSH (Albrecht-Paterson-Watson, 2009)



- Once enough data has arrived, the receiver will receive what it thinks is the MAC tag
  - The MAC check will fail with overwhelming probability
  - Consequently the connection is terminated (with an error message)
- How much data is "enough" so that the receiver decides to check the MAC?
- Answer: whatever is specified in the length field 🟧

Breaking SSH (Albrecht-Paterson-Watson, 2009)



- Knowing IV and 32 bits of P_0', the attacker can now recover 32 bits of the target plaintext block:

$$P_i^* = C_{i-1}^* \oplus d_K(C_i^*) = C_{i-1}^* \oplus IV \oplus P_0'$$

(Real attack is a bit more complicated, but follows this idea.)

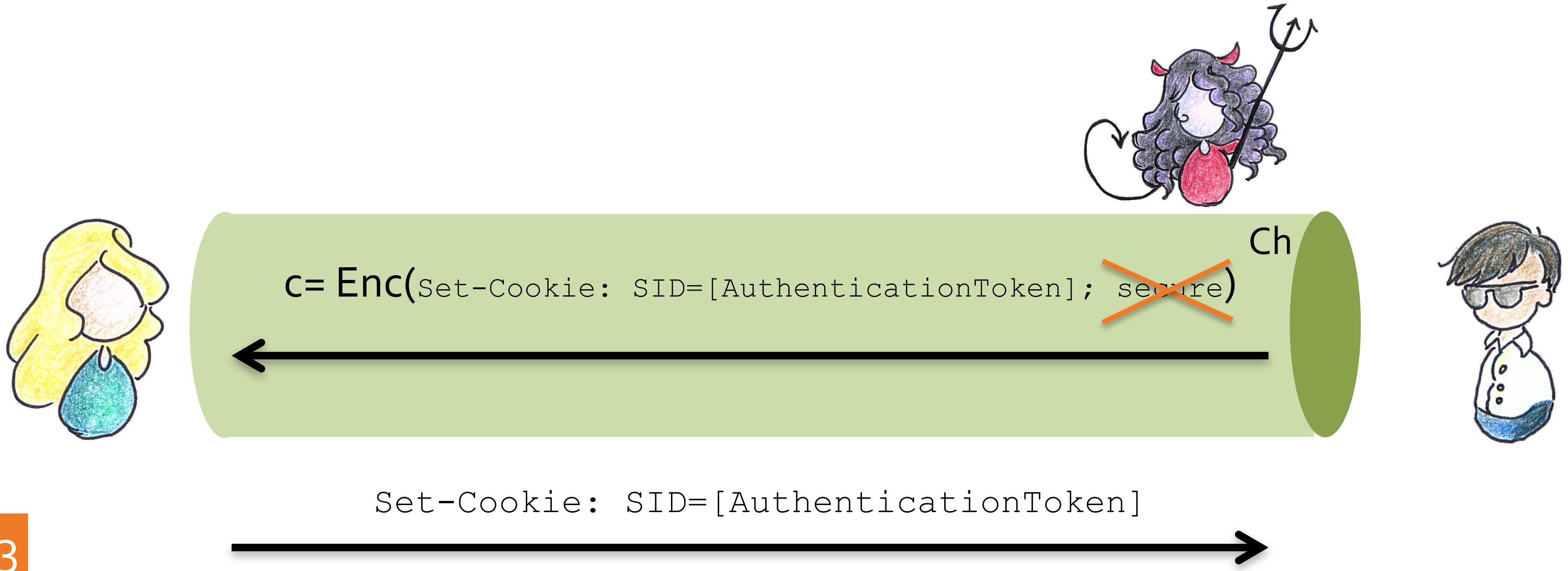
SSH lessons

- Model used for security proof was inadequate.
    - It assumed length known and atomic processing of ciphertexts
    - But fragmented adversarial delivery over TCP is possible
- Implementation can't know if complete ciphertext has arrived because of encrypted length field, unless it decrypts first block.
- That's not in any of the AE/AEAD security models!
- This modeling gap addressed in (Paterson-Watson, 2010) and (Boldyreva-Degabriele-Paterson-Stam, 2012).

Second example: cookie cutters

Bhargavan, Delignat-Lavaud, Fournet, Pironti, Strub 2014: cookie cutter attack on "HTTP over SSL/TLS"

- Attacker forces part of the HTTP header (e.g., cookie) to be cut off
- Partial message/header arrives and might be misinterpreted


Ch
c= Enc(Set-Cookie: SID=[AuthenticationToken]; ~~secure~~)
Set-Cookie: SID=[AuthenticationToken]

Cookie cutters

Why doesn't this violate the proven integrity of SSL/TLS encryption?

```
6.2.1. Fragmentation

The record layer fragments information blocks
into TLSPlaintext records [...].  Client
message boundaries are not preserved in the
record layer (i.e., multiple client messages
of the same ContentType MAY be coalesced into
a single TLSPlaintext record, or a single
message MAY be fragmented across several
records).
```

RFC 5246 TLS v1.2

Cookie cutters

Why doesn’t this violate the proven integrity of SSL/TLS encryption?

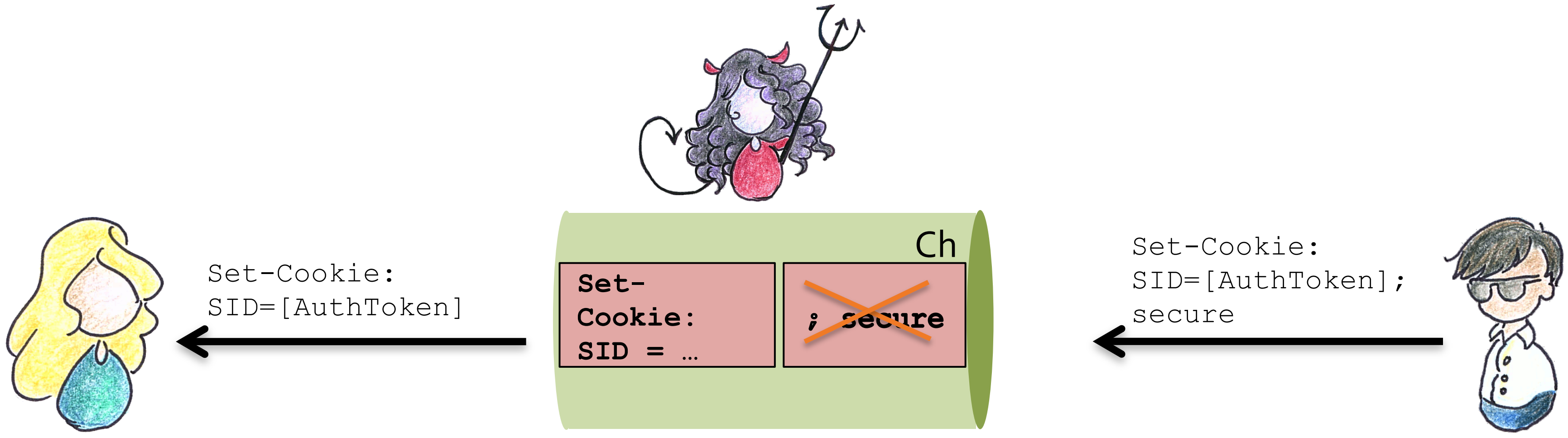
`6.2.1. Fragmentation`

`The record layer fragments information blocks into TLSPlaintext records [...].  Client message boundaries are not preserved in the record layer (i.e., multiple client messages of the same ContentType MAY be coalesced into a single TLSPlaintext record,` **`or a single message MAY be fragmented across several records`**`).`

RFC 5246 TLS v1.2

Cookie cutters

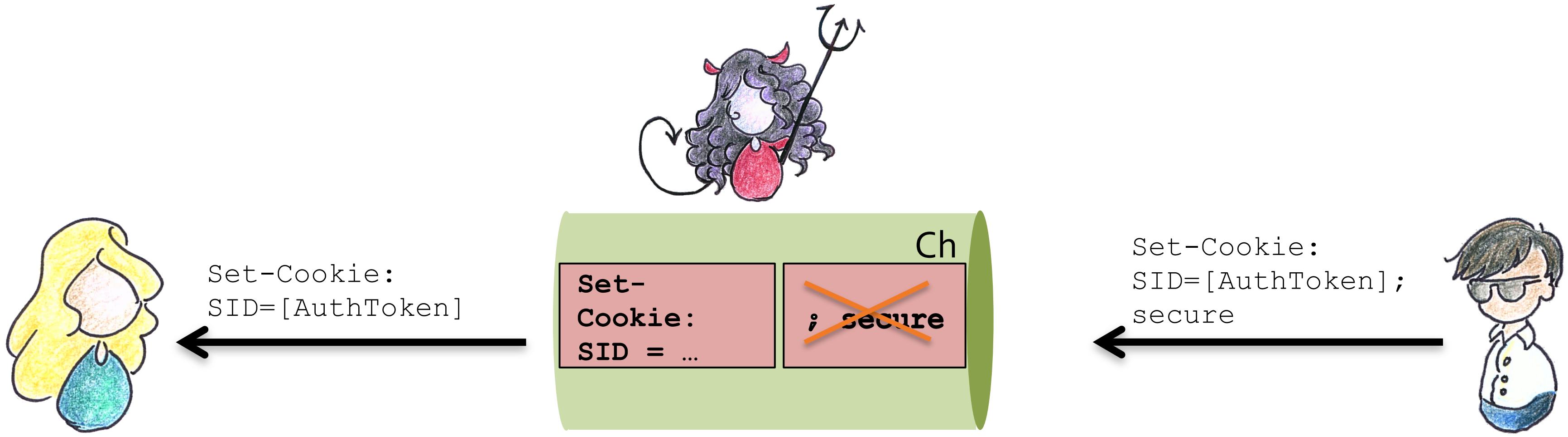
- So SSL/TLS can (and will) fragment when *sending.*
- Compare to SSH that only has to deal with fragments when *receiving.*
- Both protocols provide a TCP-like *streaming* interface to applications, not a UDP-like message-oriented one.



2 TLS records

Cookie cutters

- It's up to the calling application to deal with message boundaries if it wants to use SSL/TLS for atomic message delivery
- Cookie cutter attack relies on a buggy browser that does not check for correct HTTP message termination
- This happens in practice, presumably because developers do not understand the interface provided by SSL/TLS



From AEAD to secure channels

From AEAD to secure channels

- SSL/TLS is not alone in presenting a streaming interface to applications.
- Also SSH “tunnel mode”, QUIC.
- What security can we hope for from such a channel?
- Boldyreva-Degabriele-Paterson- Stam (2012) already treated the case where the receiver handles fragmented ciphertexts.
- In Fischlin-Günther-Marson-Paterson (2015), we provide a systematic study of the case where *both* sender and receiver may fragment, as in TLS

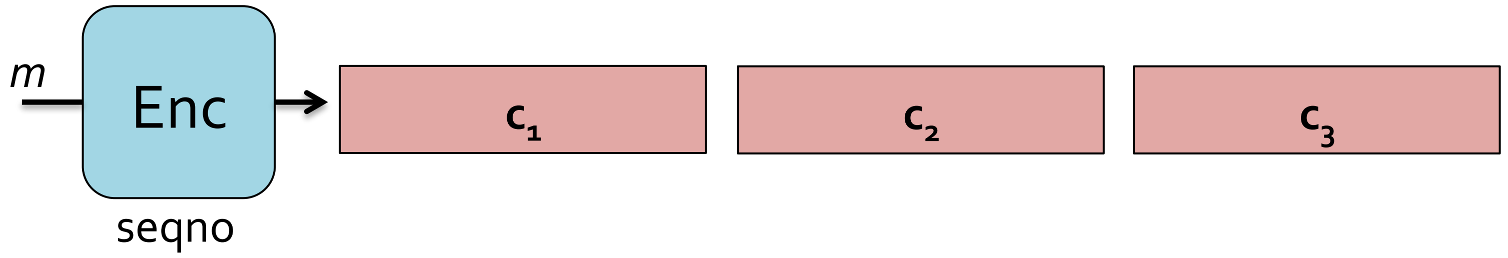
Streaming secure channels (FGMP15)

- Defining CCA and integrity notions in the full streaming setting is non-trivial!
  - Hard part is to define when adversary's decryption queries deviate from sent stream, and from which point on to suppress decryption oracle outputs.
- We develop streaming analogues of IND-CPA, IND-CCA, INT-PTXT and INT-CTXT.
- We recover an analogue of the classic relation:

$$\text{IND-CPA} + \text{INT-CTXT} \rightarrow \text{IND-CCA}$$

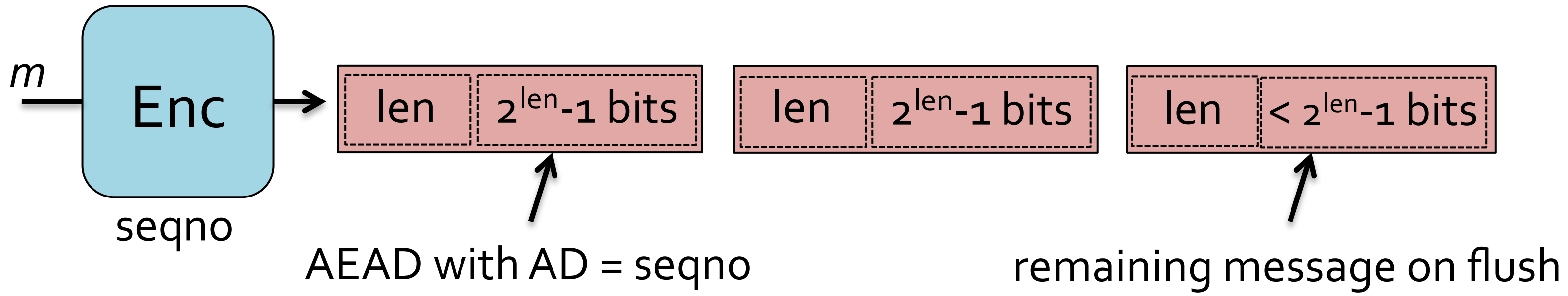
Streaming secure channels (FGMP15)

- We give a generic construction for a secure streaming channel that validates the SSL/TLS design
- The construction uses AEAD as a component
- Security as streaming channel follows from standard AEAD security properties



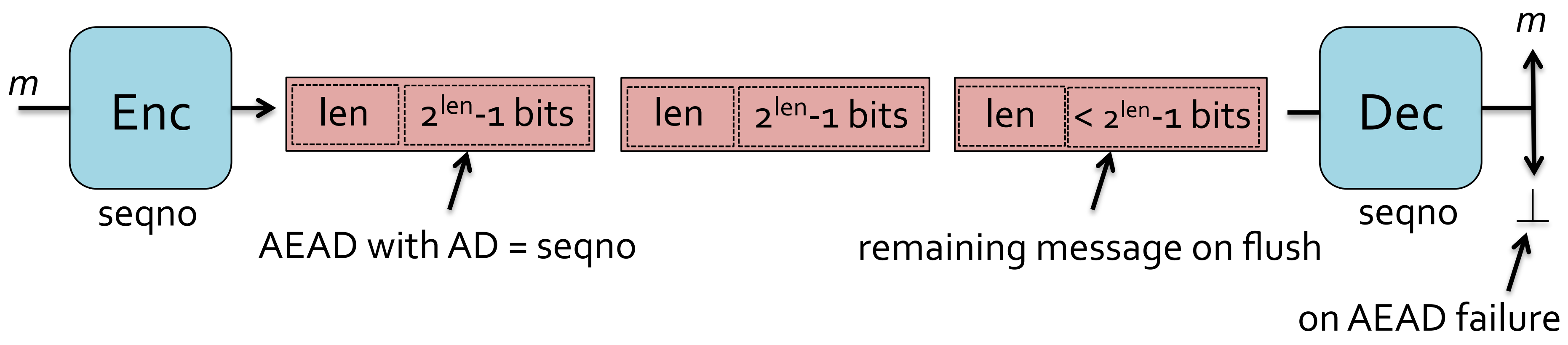
Streaming secure channels (FGMP15)

- We give a generic construction for a secure streaming channel that validates the SSL/TLS design
- The construction uses AEAD as a component
- Security as streaming channel follows from standard AEAD security properties



Streaming secure channels (FGMP15)

- We give a generic construction for a secure streaming channel that validates the SSL/TLS design
- The construction uses AEAD as a component
- Security as streaming channel follows from standard AEAD security properties



Closing remarks

Closing remarks

- We've seen the evolution from simple security models for symmetric encryption to more sophisticated security notions for secure channels.
- The cryptography community is focussed on AEAD as an end goal.
- AEAD is an important tool but not the same thing as a secure channel.
- Key take-away: it can be profitable to think top-down too (from API to crypto).