Applied Cryptography
Spring Semester 2023
Lectures 8 and 9

Kenny Paterson (@kennyog)

Applied Cryptography Group

https://appliedcrypto.ethz.ch/

Overview of this lecture

Attacks on modes, focussing on CBC mode:

- Padding and padding oracle attacks on CBC mode
- Attacks on CBC mode with predictable IVs
- (Yet another attack on CBC mode based on lack of integrity – see additional slides.)

Padding and padding oracle attacks

Padding in CBC mode

- So far, we've assumed all plaintexts have bit-lengths that are multiples of *n*, the block size.
- In practice, we want to be able to encrypt arbitrary length messages.
- Common solution: use message padding.
- Message padding: applies an injective map pad(·) to messages prior to CBC mode encryption, remove padding after decryption.
  - pad(·) takes inputs in $\{0,1\}^*$.
  - pad(·) has outputs in $\{\{0,1\}^n\}^*$, i.e. bit-strings that are multiples of *n* in length.
  - pad(·) is necessarily expanding.
  - pad(·) and its inverse both need to be efficiently computable.
  - pad(·) may be randomised or deterministic.

Padding in CBC mode

Concrete example: simplified TLS padding

- Message is always a whole number of bytes (multiple of 8 bits).
- pad() adds (t+1) copies of byte value t, where $0 \le t < n/8$, to bring message length up to multiple of n bits (n/8 bytes).
- So, for AES, where n=128, we have $n/8 = 16$ and the possible padding strings added to messages are (in byte format):

0x00

0x01 0x01

...

0x0F 0x0F ... 0x0F (16 copies of 0x0F)

Padding in CBC mode

Concrete example (ctd): simplified TLS padding

- Note that at least one byte of padding is *always* added.
- If message length is already an exact multiple of *n* bits, then a complete new block containing only padding is added (an oxoF … oxoF block for *n*=128).
- Padding is expanding.
- Many other padding functions are possible: see exercises.

- Decryption operation now does CBC mode decryption, followed by removal of padding.
- Security consequences?

Padding oracle attacks

- After CBC mode decryption, message handling code needs to:
  1. parse plaintext as: `message||padding`
  2. remove `padding`
  3. pass `message` to application
- The `padding` string may be incorrectly formatted – due to errors or **malicious inputs**.
- For example, with simplified TLS padding, `padding` should only be one of 16 possibilities:

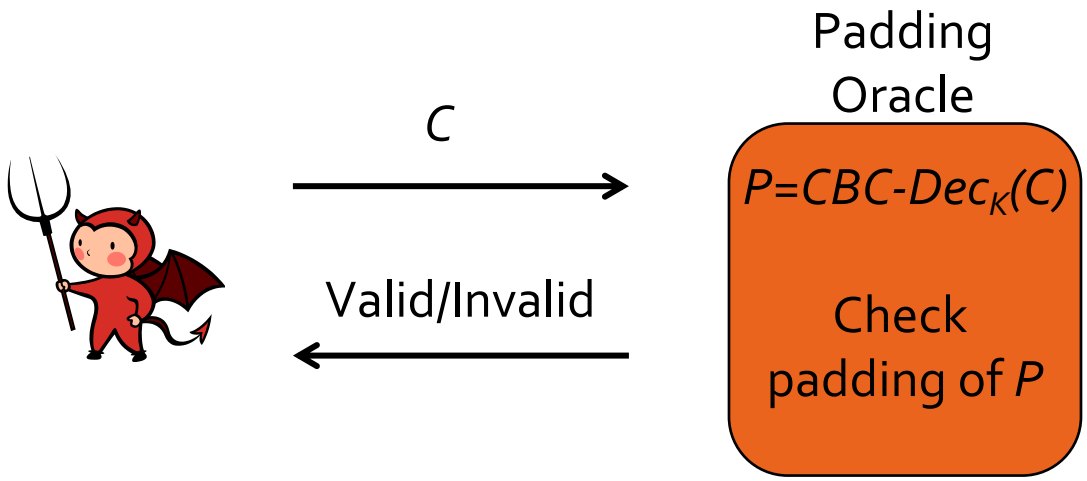
  "0x00" or "0x01 0x01" or … or "0x0F 0x0F… 0x0F".
- What should the code do if the padding is not one of these 16 strings?

Padding oracle attacks

- Typical implementation behaviour: throw an exception with a helpful error message if padding is not one of the expected values.
  - Because crypto code should not return garbage data to the calling application.
- An adversary may be able to detect when this happens:
  - The error message may be directly visible in a response message sent on the network.
  - Error messages may be recorded in logs.
  - Timing behaviour of application code: padding error may arise earlier in processing than other errors, e.g. sanity checking of plaintext message.

Padding oracle attacks

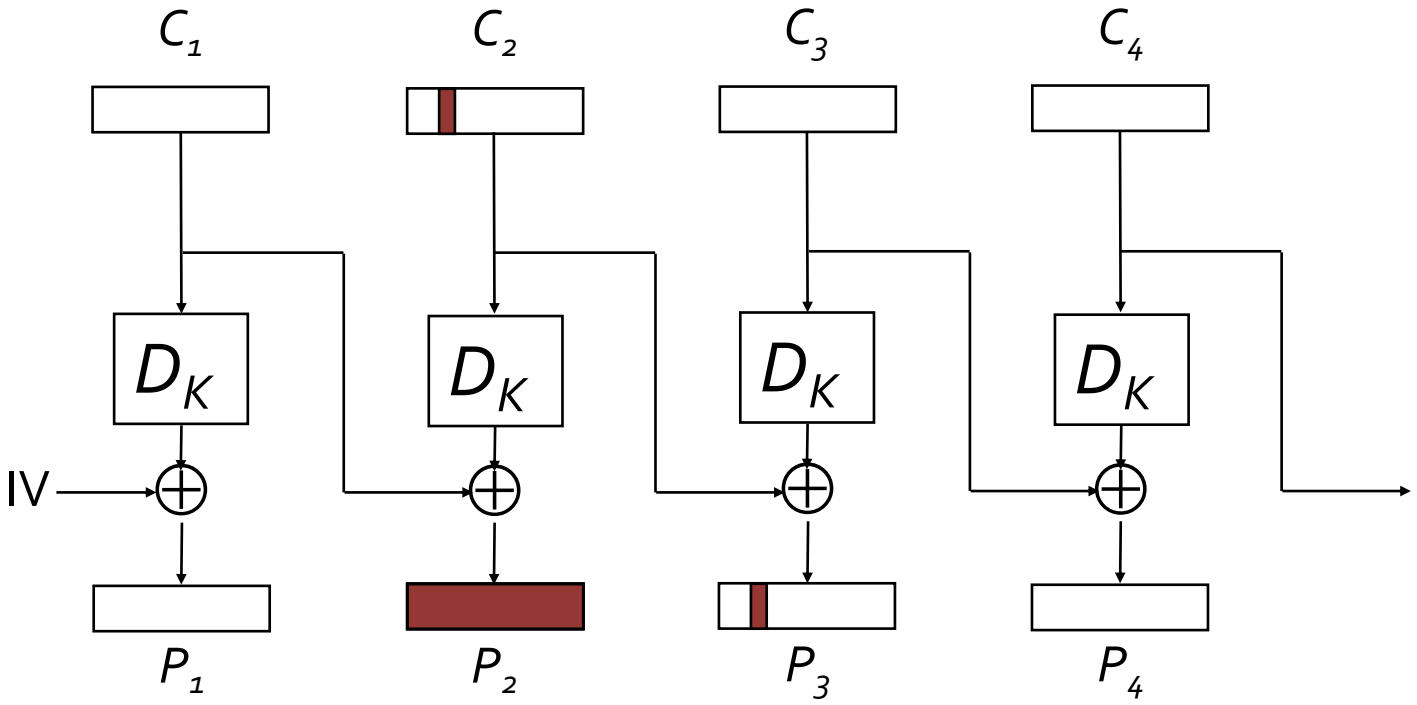
- An adversary can exploit distinguishable padding errors to do a full plaintext recovery attack!
- To simplify the presentation, we suppose the adversary has access to a *padding oracle*:



- Adversary can submit arbitrary ciphertexts *C* and learn whether the underlying plaintexts have valid or invalid padding (1-bit of leakage per query).

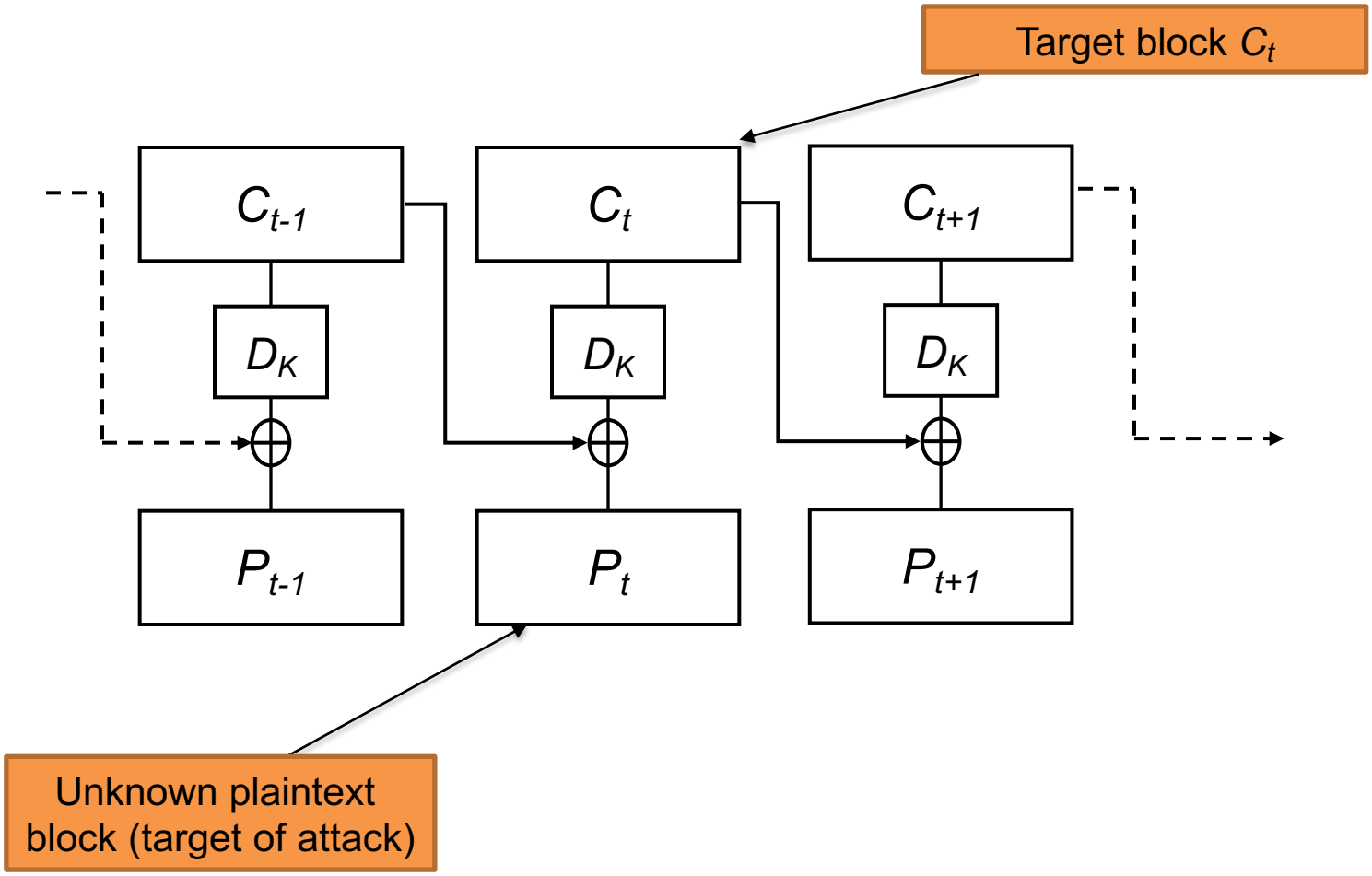
Reminder: Error propagation in CBC mode

- Suppose C_2 is replaced by $C_2 \oplus \Delta$.

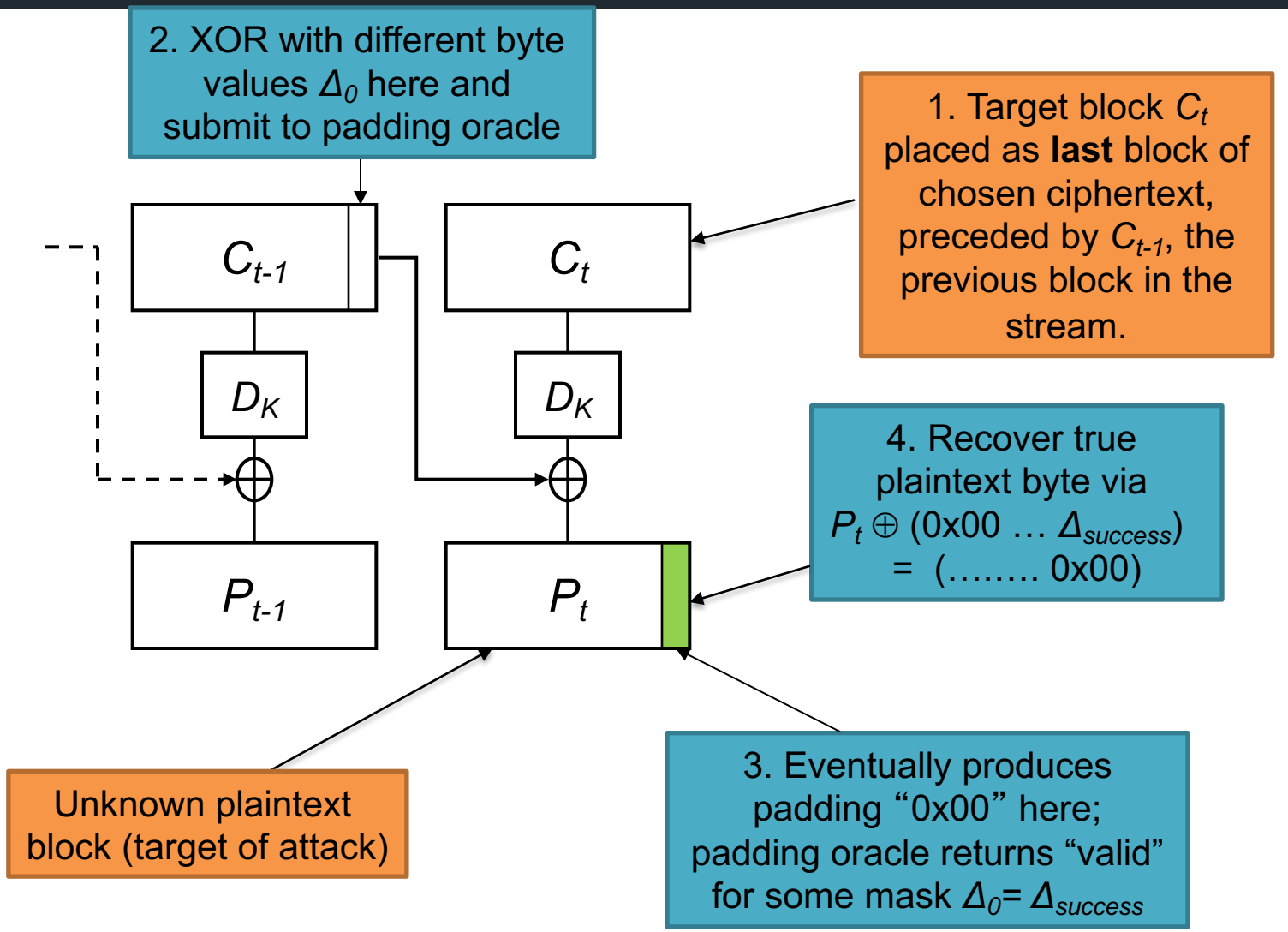


- Then after decryption the error propagates to P_3, and P_2 gets randomised.
- If we replace C_i by $C_i \oplus \Delta$ then the effect is to replace P_{i+1} by $P_{i+1} \oplus \Delta$.

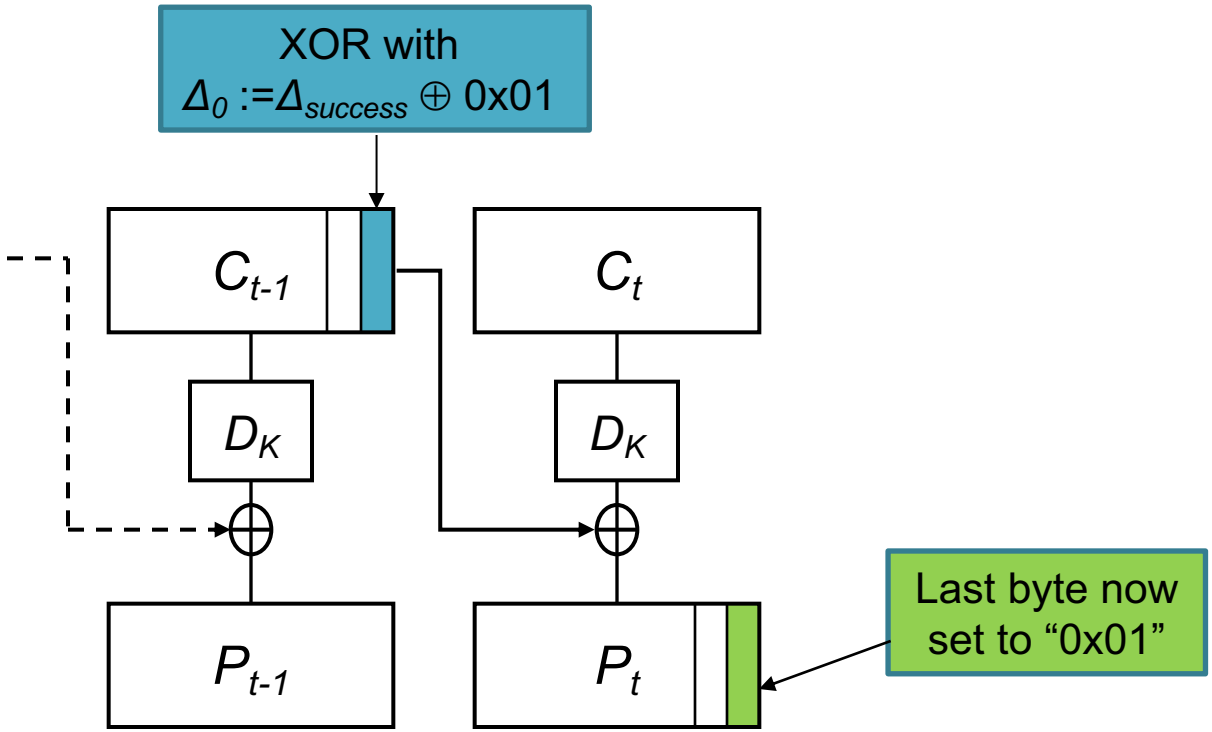
Padding oracle attack for simplified TLS padding



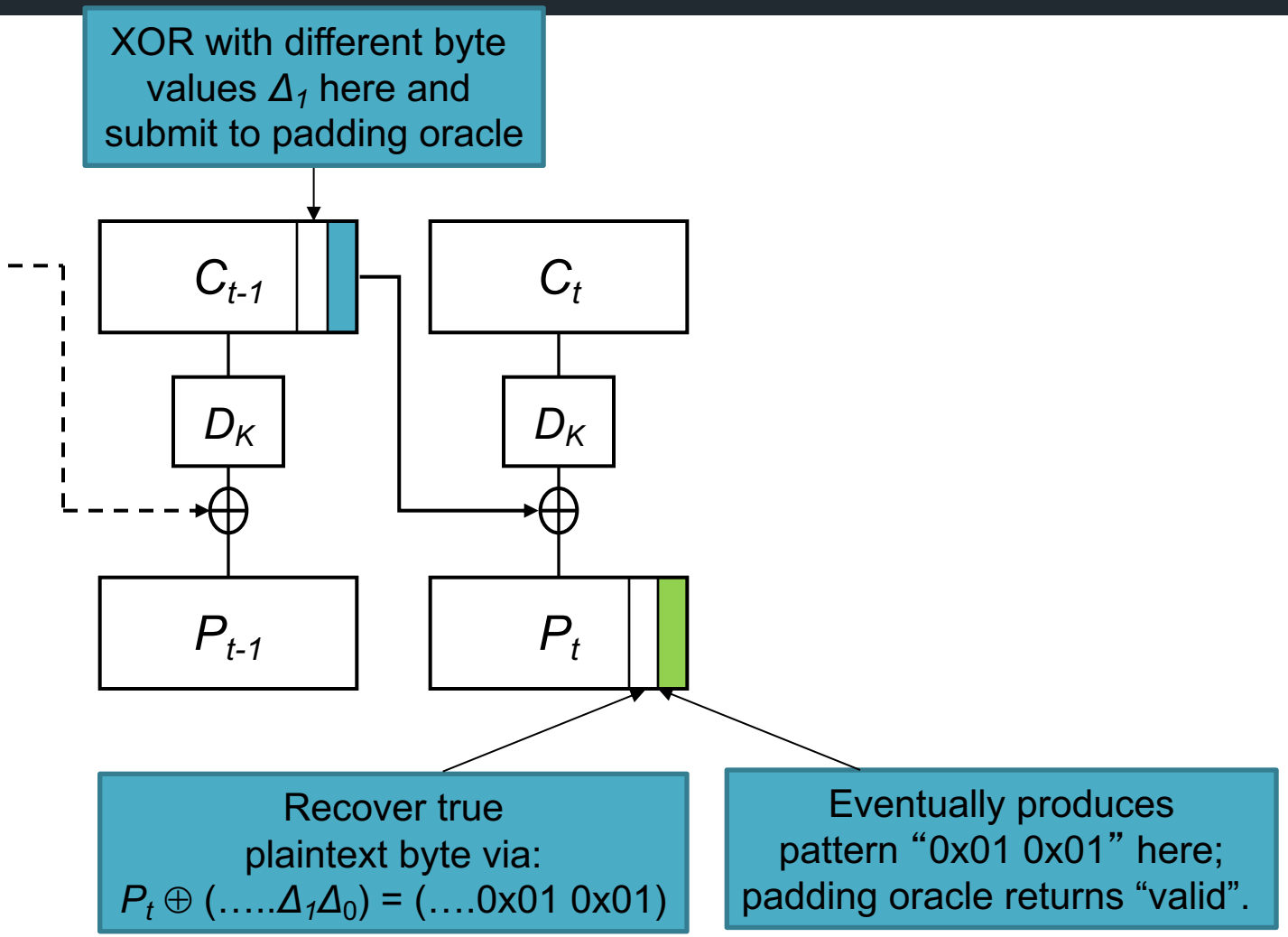
Padding oracle attack for simplified TLS padding



Padding oracle attack for simplified TLS padding



Padding oracle attack for simplified TLS padding



Padding oracle attack for simplified TLS padding

- An average of 128 calls to the padding oracle are needed to extract the last byte of each plaintext block.
  - May need to disambiguate "unlucky" case where initial success is for "0x01 0x01" instead of "0x00".
  - This requires one more trial – details omitted here.
- Can extend to the entire block, with an average of 128 trials per byte.
  - Target bytes from right to left in block, gradually increasing length of valid padding pattern.
- Can extend to an entire ciphertext with many blocks.
  - Because attacker can place *any* target block C_t as last block of ciphertext, preceded by C_{t-1}.

Remarks

- Original concept of the padding oracle attack is due to Vaudenay (2002):
  - https://link.springer.com/chapter/10.1007%2F3-540-46035-7_35
- It's a kind of *chosen ciphertext attack*, and as such is not covered by the IND-CPA security model.
- In practice, there are many details to take care of:
  - The padding oracle may be unreliable, e.g. due to timing noise.
  - The padding oracle may be "single shot" because errors cause session termination.
  - There may be additional constraints on ciphertexts, e.g. minimum length, maximum length.
- Yet exploitable padding oracles are endemic in implementations and continue to be discovered to this day.
  - Many different padding schemes can be attacked, not only TLS simplified padding.
  - Practical examples: XML encryption, ASP.NET, SSL/TLS, DTLS.

Further remarks

- The SSL/TLS case is particularly interesting:
  - Originally proposed as a target by Vaudenay in 2002.
  - Shown to be feasible by Canvel et al. (CRYTPO 2003) for SSL/TLS via a timing side channel, patched, and then largely forgotten about.
  - Rebooted by the Lucky 13 attack in 2013 (for all versions of SSL and TLS), exploiting an even smaller timing channel left after the patch. See: https://www.ieee-security.org/TC/SP2013/papers/4977a526.pdf
  - **Re**rebooted by the POODLE attack in 2014 (for SSLv3) based on analysis of SSL error messages – no timing channel needed! See: https://www.openssl.org/~bodo/ssl-poodle.pdf
- Main take-away: handling of padding is security critical, and CBC mode with padding is generally vulnerable.
- Additional protection for CBC mode is needed – integrity via MACs so that modified ciphertexts are detected.
- However the MAC and the encryption scheme need to be combined in the right way (and SSL/TLS got it wrong!).

Attack on CBC mode with predictable IVs

Attack on CBC mode with predictable IVs

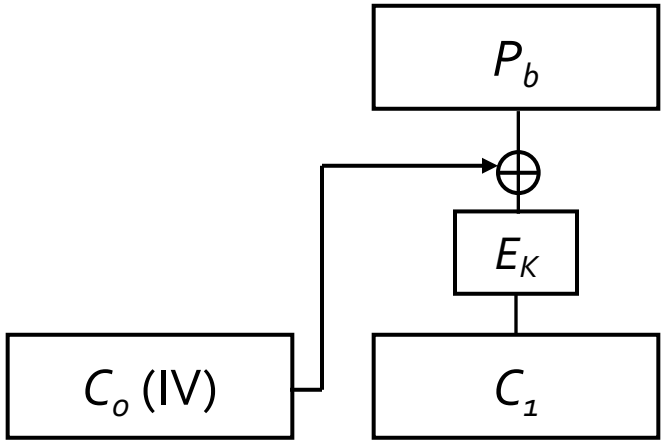
- We said earlier that the CBC mode IV (block C_0) needs to be random.
- This is required by theory: to get an IND-CPA security proof for CBC mode to work.
- But it's also important in practice.
- We show an attack in the IND-CPA model on CBC mode with predictable IVs.
  - First observed for CBC mode in general by Rogaway in 1995.
  - See Section 9 of: http://web.cs.ucdavis.edu/~rogaway/papers/draft-rogaway-ipsec-comments-00.txt
- In some circumstances, this attack can be amplified to yield a *full plaintext recovery* attack.

Attack on CBC mode with predictable IVs

- Suppose, for simplicity, that only single-block plaintexts P will be encrypted using CBC-mode and that there is no padding in use.
- Consider the IND-CPA attack setting.
- Adversary submits a pair of single-block messages (P_0,P_1) and receives a CBC-mode encryption of P_b.
- This will be a two-block ciphertext C_0, C_1 in which C_0 is the IV and:

$$C_1 = E_K(P_b \oplus C_0)$$

Attack on CBC mode with predictable IVs



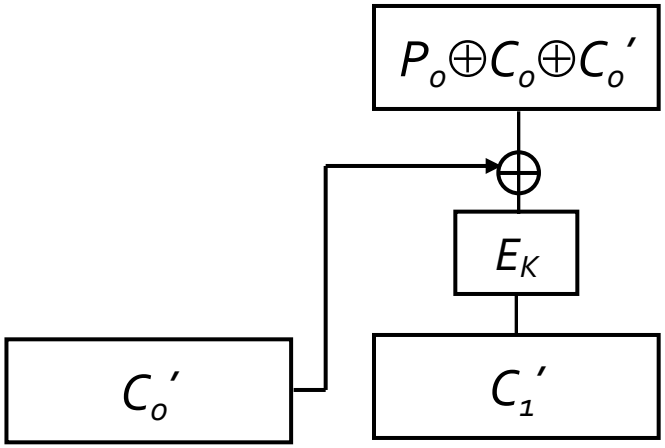
Attack on CBC mode with predictable IVs

- Attacker predicts that some block C_0' will be used as the IV for the next encryption.
- Attacker now requests encryption of the single block message:

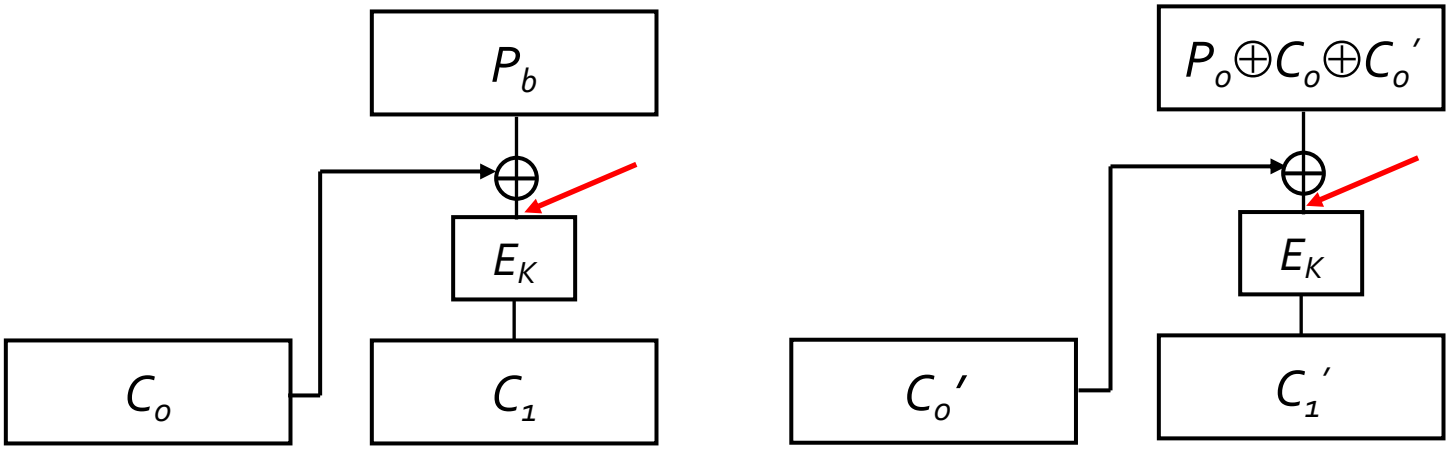
$$P_0 \oplus C_0 \oplus C_0'$$

(in the IND-CPA attack setting: set both messages to this value).

- Attacker receives as ciphertext C_0', C_1'.



Attack on CBC mode with predictable IVs



Old ciphertext

New ciphertext

- If $P_b = P_o$, then the inputs to block cipher are the same in both encryptions (as indicated by red arrows), otherwise, they are different.
- But E_K is a permutation – so distinct inputs give distinct outputs.
- So we have: $\boldsymbol{P_b = P_o}$ **if and only if** $\boldsymbol{C_1 = C_1'}$**.**
- Hence $\boldsymbol{b}$**=o if and only if** $\boldsymbol{C_1 = C_1'}$.
- Formally, this breaks IND-CPA security of CBC mode with predictable IVs.

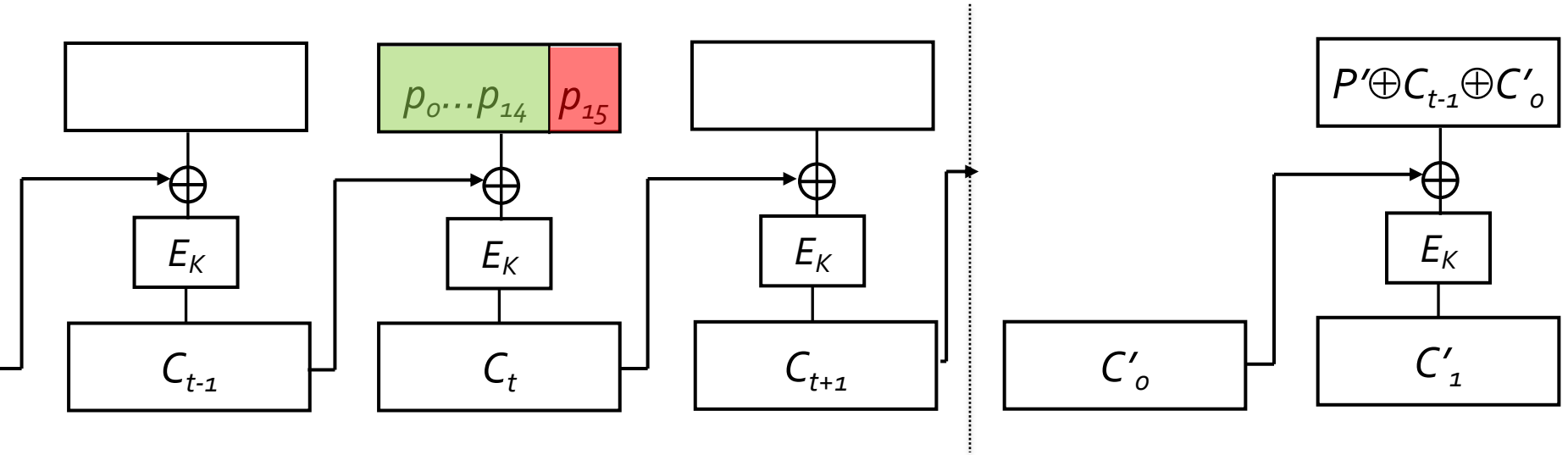
Attack on CBC mode with predictable IVs

- The attack extends easily to multiple-block plaintexts, with the guessed block P_0 / P_1 being in an arbitrary but known position in the sequence of plaintext blocks.
- The key requirement is that the attacker needs to be able to place $P_0 \oplus C_0 \oplus C_0'$ as the first block in a chosen plaintext.
- The attack applies in the case of IV chaining, where the IV for current encryption is set to be last block from previous ciphertext.
  - As used in SSL 3.0, TLS 1.0, SSH in CBC mode.
  - The weakness for SSL/TLS was noted by Dai and Moller in 2004.

Attack on CBC mode with predictable IVs

- In special circumstances the attack can be extended to *full plaintext recovery.*
  - The BEAST attack on SSL/TLS, Duong and Rizzo, 2011.
  - **Incomplete** research paper: https://nerdoholic.org/uploads/dergln/beast_part2/ssl_jun21.pdf
  - The attack was the first major attack on the SSL/TLS record protocol in many years, followed by CRIME, Lucky 13, RC4 attacks, POODLE…
  - Introduced "malicious Javascript in the browser" as a method for mounting IND-CPA attacks on SSL/TLS.
  - Some details follow.

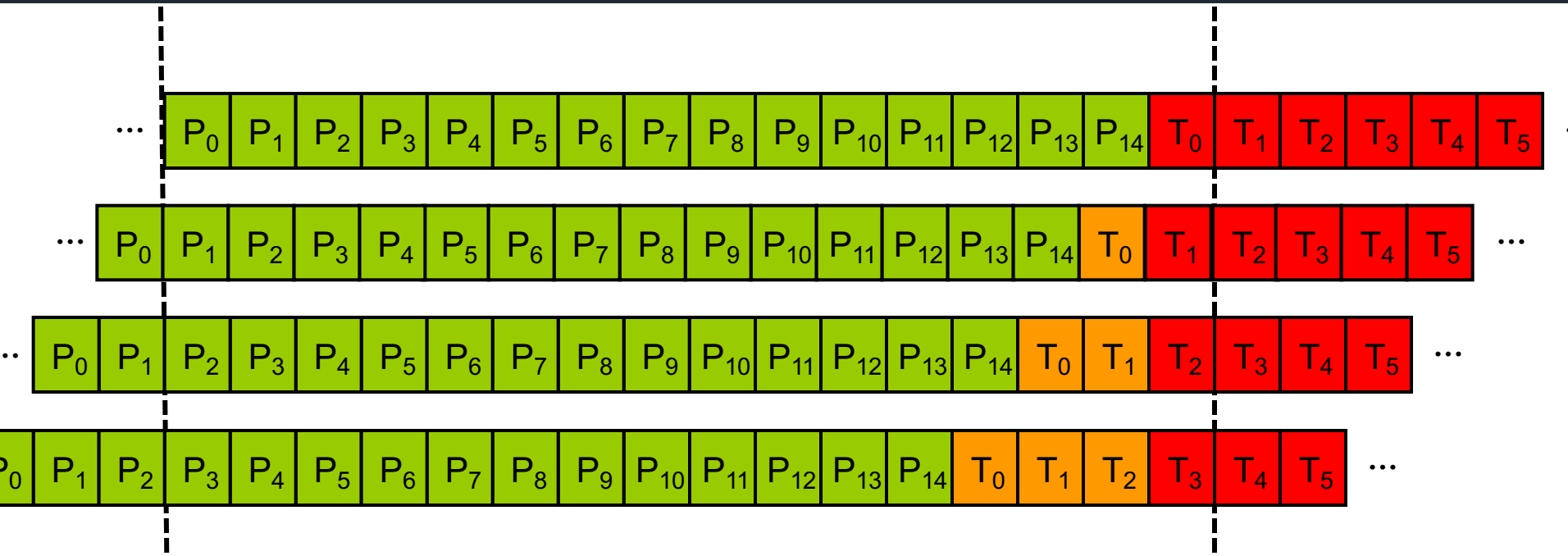
The BEAST – Part 1 – Basic mechanics



- Attacker is interested in byte p_{15} underlying some block C_t. Assume it already knows all the other bytes $p_0 \ldots p_{14}$ in this block.
- We **assume** that chosen plaintext blocks can be repeatedly placed as the first block of chosen plaintexts, and that the IV is predictable.
- Attacker sets $P' = p_0 p_1 \ldots p_{14} g$ where g is a guess for unknown byte p_{15}.
- Attacker now asks for encryption of plaintext block $P' \oplus C_{t-1} \oplus C'_0$ where C'_0 is the predicted IV.
- Attacker iterates over 256 possible values for g; we have:

$$g = p_{15} \text{ if and only if } C'_1 = C_t .$$

The BEAST – Part 2– Byte sliding



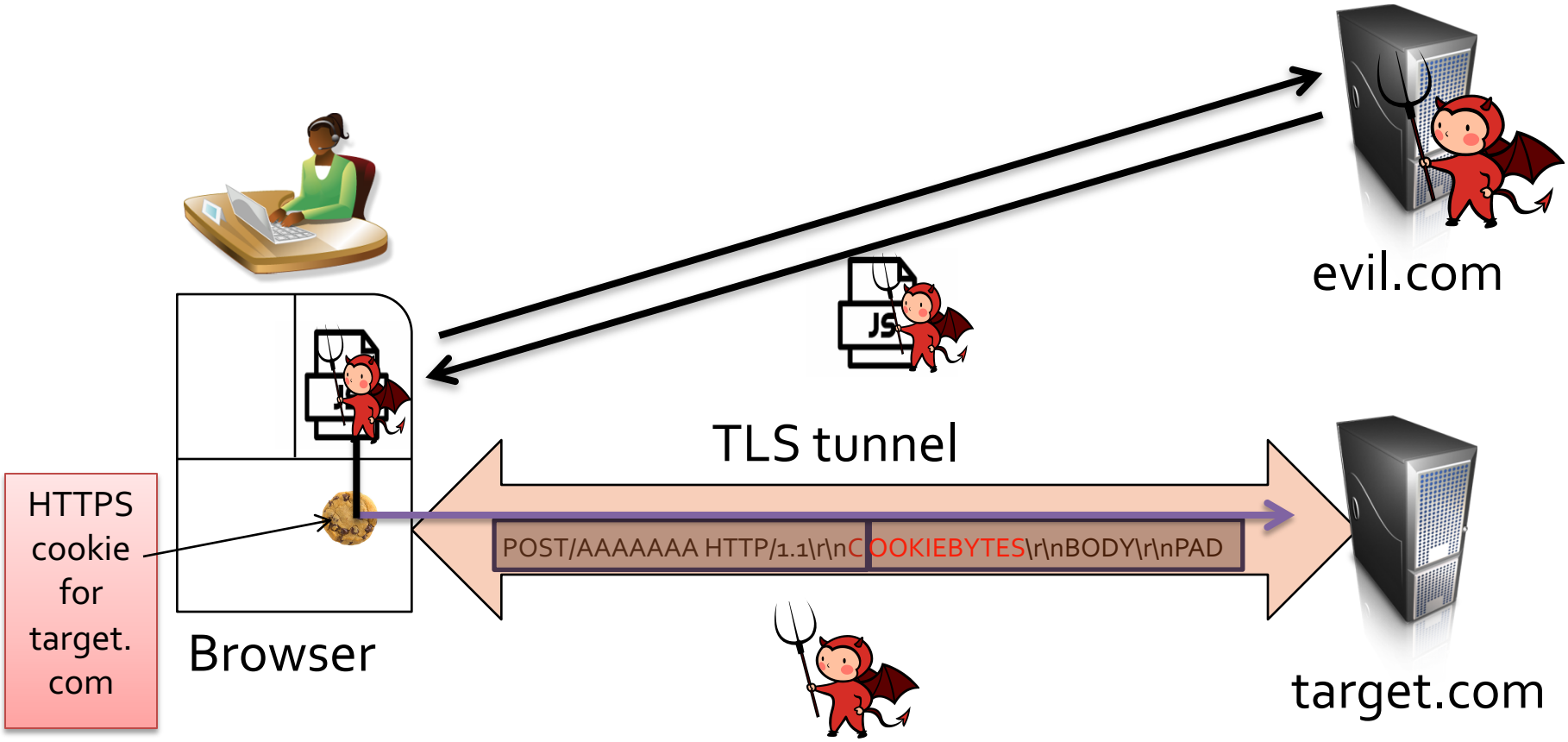
- Now assume attacker can control position of unknown bytes in stream with respect to CBC block boundaries (**chosen boundary privilege**).
- We repeat previous single-byte recovery attack with sliding bytes.

  Green: initially known bytes.

  Red: unknown (target) bytes.

  Orange: recovered bytes.

The BEAST – Part 3 – Realisation in the browser



BEAST – Part 4 – Key Features

- BEAST JavaScript loaded ahead of time into client browser from compromised or malicious wesbite.
  - Provides chosen-plaintext capability.
- Attack target is HTTP secure cookie, which is attached automatically to out-going HTTP requests.
- BEAST JavaScript uses HTTP padding to realise the chosen boundary privilege.
  - JavaScript needs to be able to inject its chosen plaintext blocks at the very start of SSL/TLS Record Protocol messages.
  - JavaScript also needs to communicate with MITM attacker.
  - Issues around Same Origin Policy.
- Summary: it's complicated, but it can be made to work, at least in theory.

BEAST – Impact

- BEAST was a major headache for TLS.
  - Perceived to be a realistic attack by vendors.
  - Most client implementations were "stuck" at TLS 1.0.
- Best solution: switch to using TLS 1.1 or 1.2.
  - Uses random IVs, so attack prevented.
  - But needs server-side support too.
- For TLS 1.0, various hacks were done:
  - Use 1/n-1 record splitting in client.
  - Send zero-length dummy record ahead of each real record.
  - Or switch to using RC4?
    - As recommended by some expert commentators!

BEAST – Lessons

- A theoretical vulnerability pointed out in 1995 became a practical attack in 2011.
  - Attacks really do get better with time.
  - Practitioners really should listen to theoreticians.
  - And, in this case, they did: TLS 1.1 and 1.2 use random IVs.
  - Problem was that no-one was using these versions in 2011.

- Tools from the wider security field were needed to make the attacks "practical" and headline news.
  - Man-in-the-browser via Javascript.
  - Fair game given the huge range of ways in which TLS get used.

Homework

- **Action 1:** start working on exercise sheet 3.
- **Action 2:** check out lab 3 in preparation for Friday.
- **Action 3:** read Boneh-Shoup – Chapter 5 on modes.
- **Action 4**: check out some of the cool research papers referenced in this lecture. ☺

- The road ahead:
  - Hash functions
  - MAC algorithms
  - Fixing basic modes using Authenticated Encryption

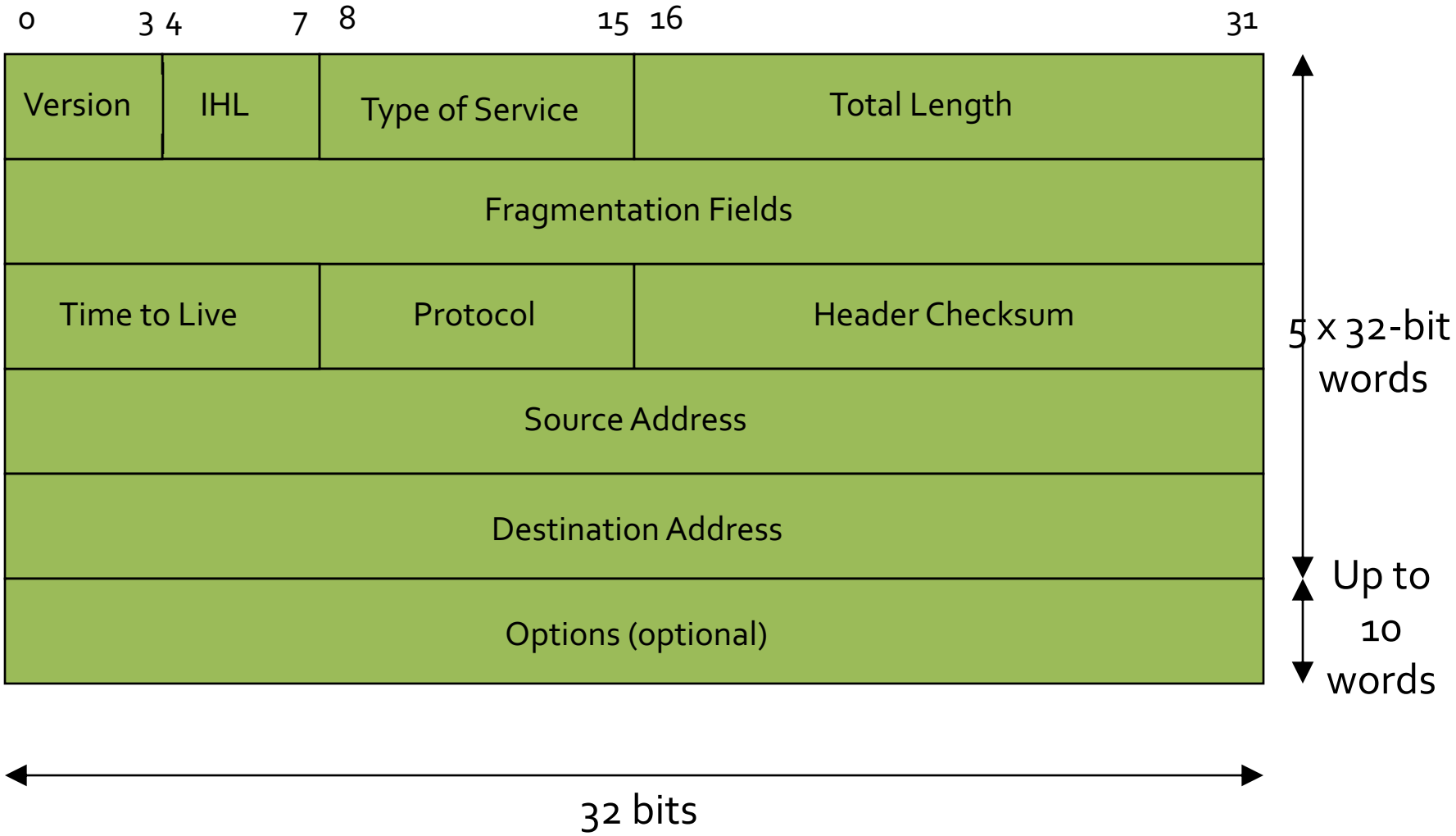
Yet another attack on CBC mode…

Attacking Linux implementation of ESP mode IPsec

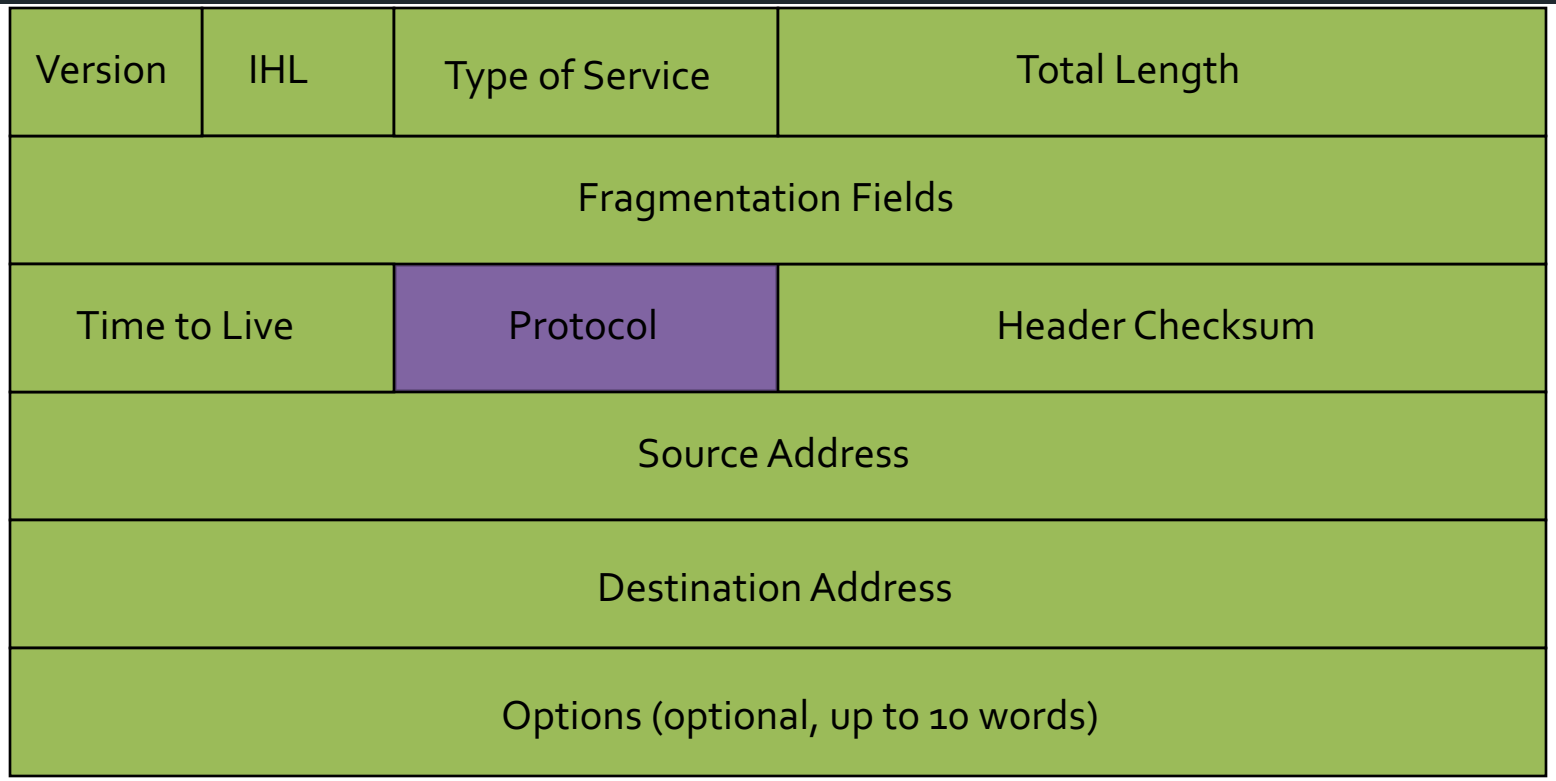
[Paterson-Yau, https://eprint.iacr.org/2005/416]:

- Three different (but related) attacks on Linux kernel implementation of encryption-only ESP in tunnel mode.
- Again 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.

IPv4 header format



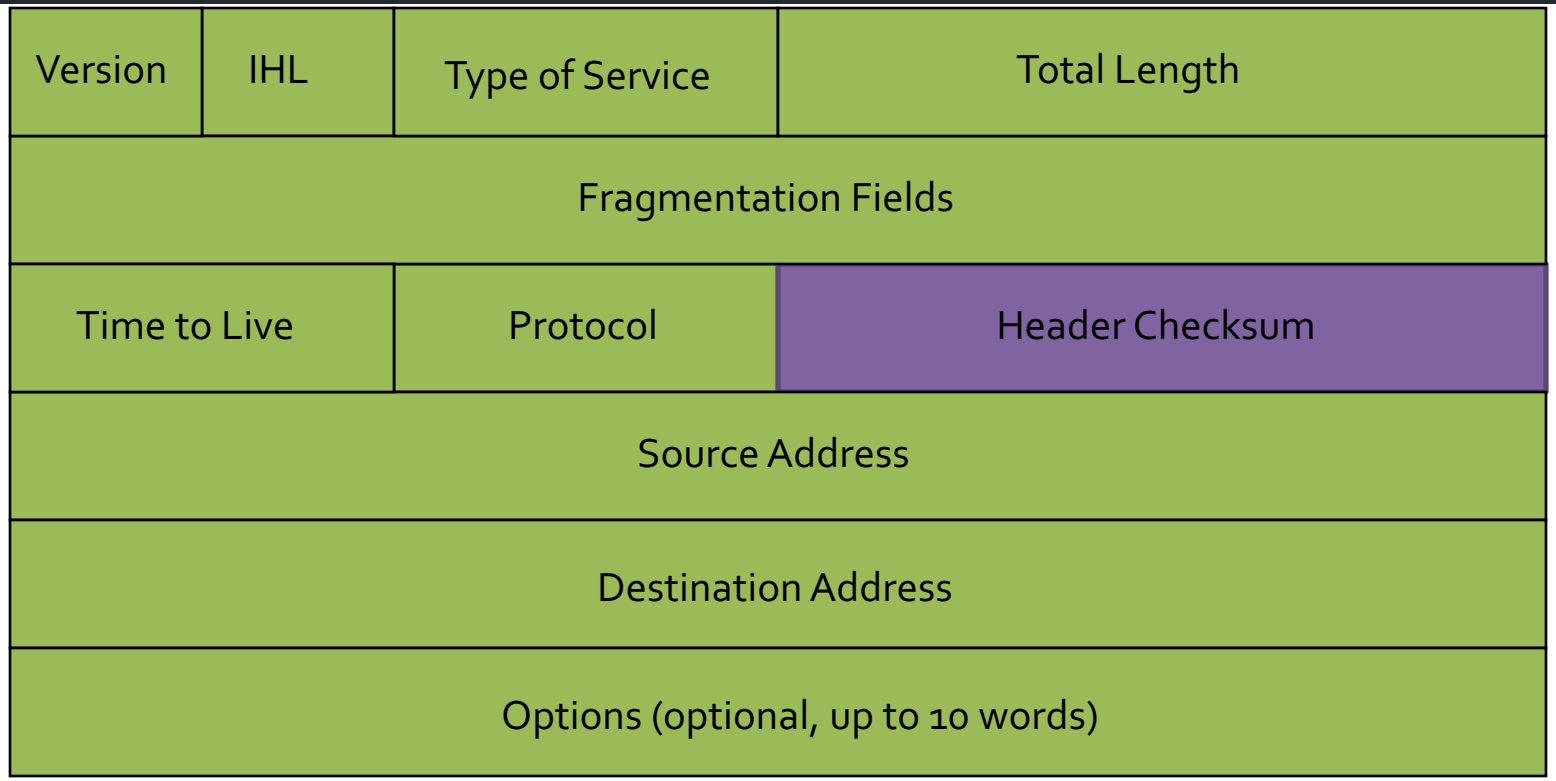
IPv4 header format



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.

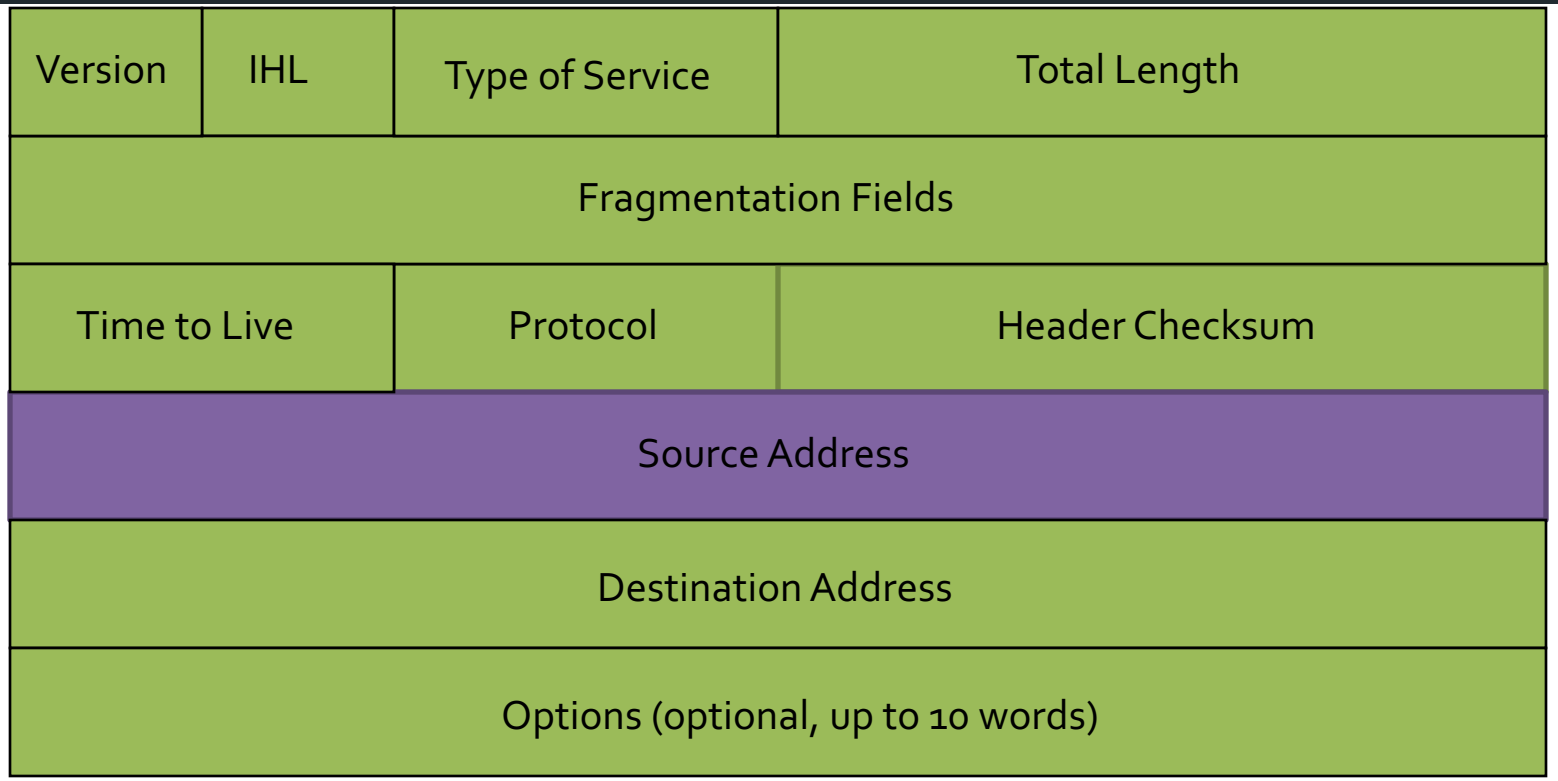
IPv4 header format



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.

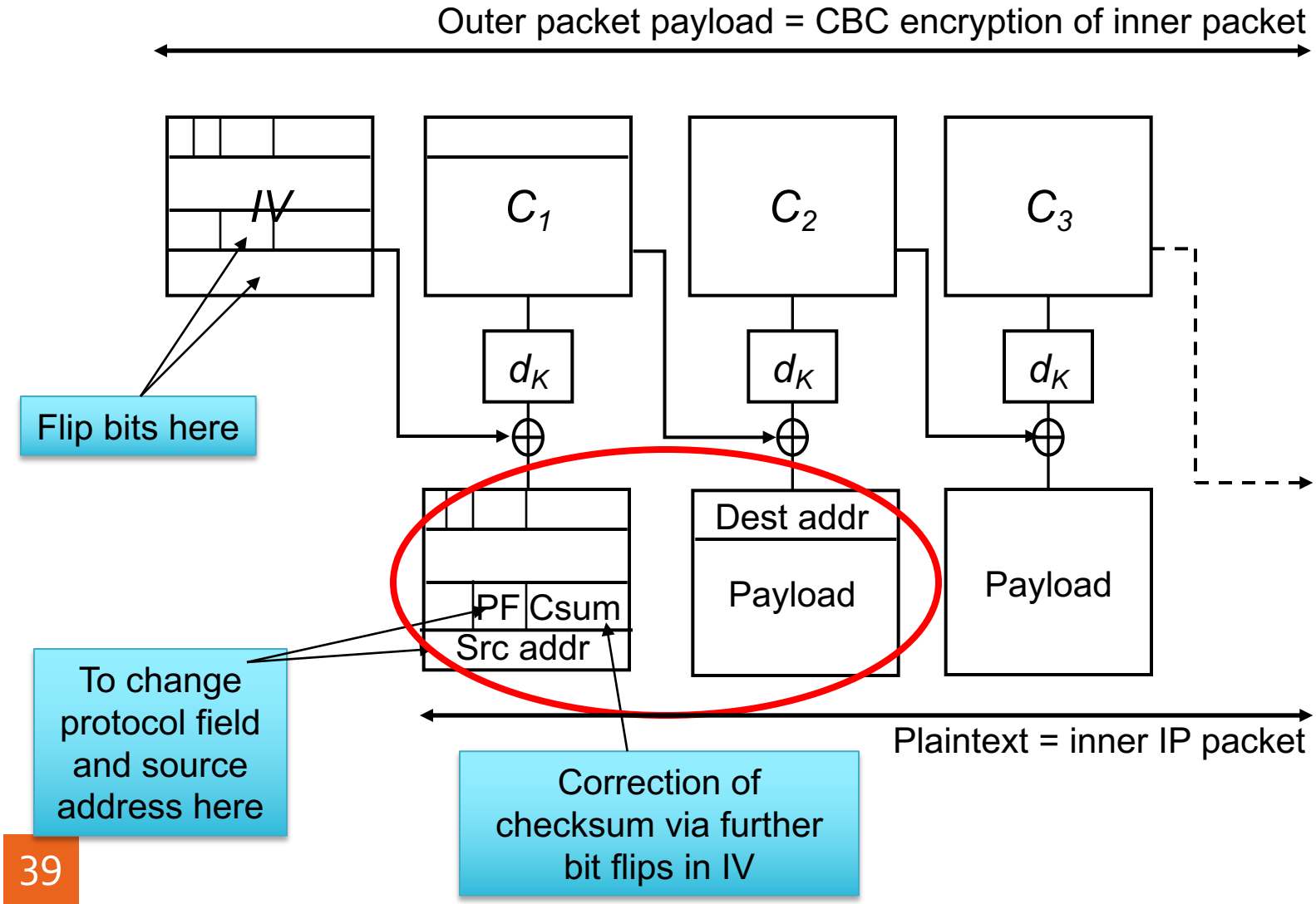
IPv4 header format



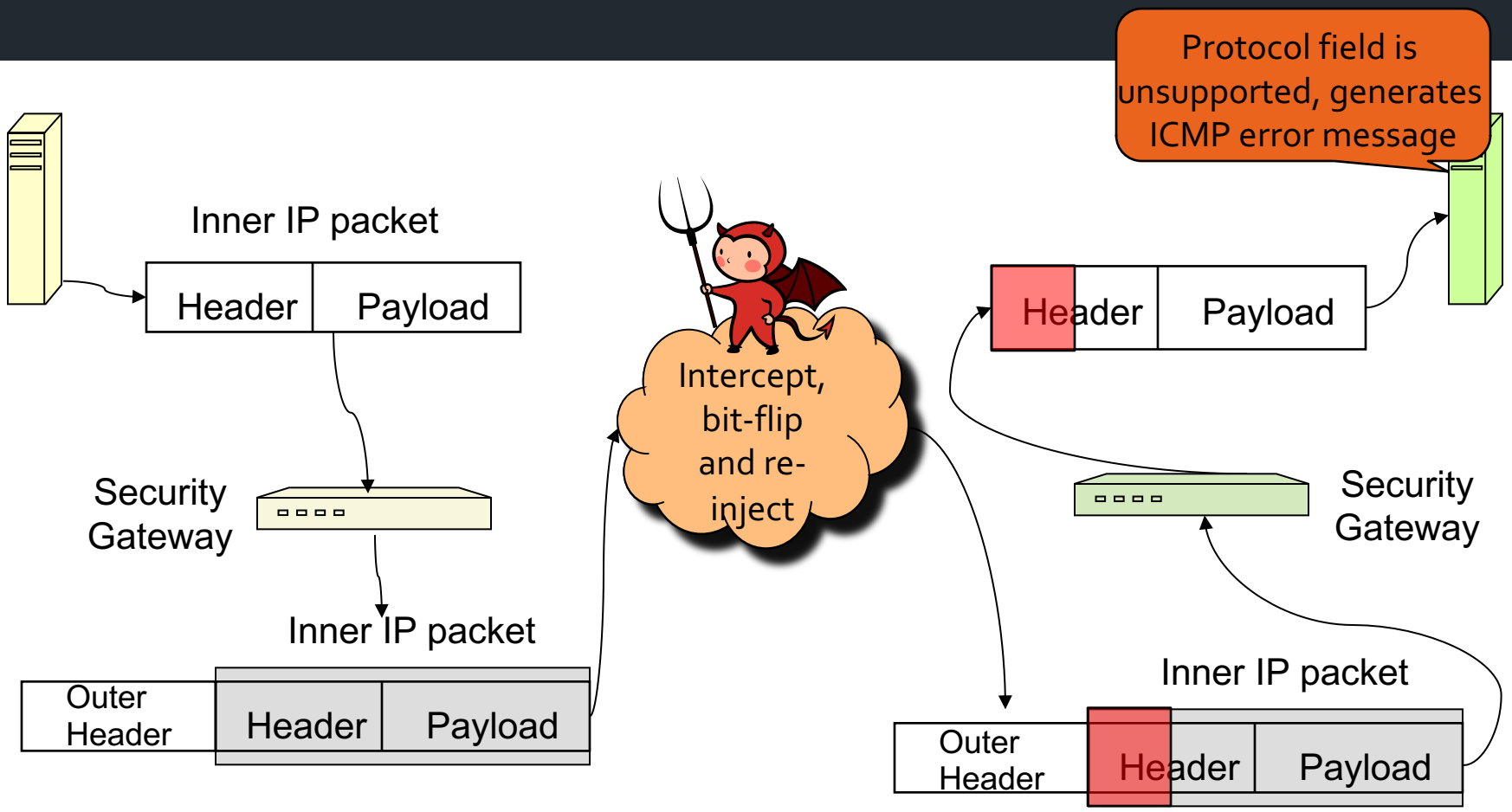
Source Address (32 bits):

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

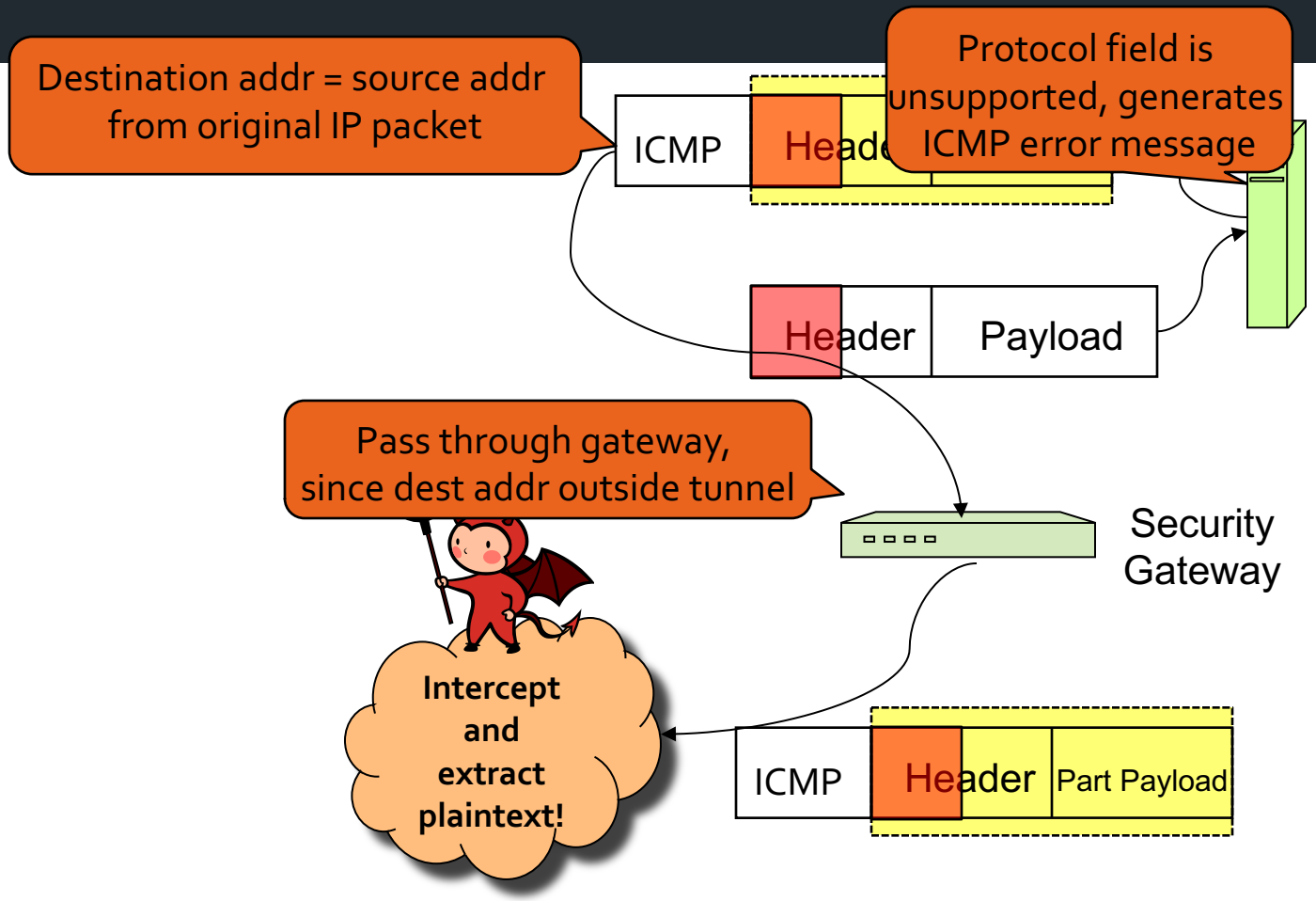
Example attack: protocol field manipulation



Attack visualisation



Attack visualisation



The attack in words

- Attacker intercepts packet, does bit flipping needed to manipulate protocol field and source address, and to correct checksum.
  - Can do better than random bit flipping for 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.
- The attack is a kind of chosen-ciphertext attack.
- Main take-away: cryptography does not exist in isolation, but interacts with its environment, e.g. a network stack.