

# Security of Internet Protocols

Kenny Paterson

Information Security Group  
Royal Holloway, University of London

[kenny.paterson@rhul.ac.uk](mailto:kenny.paterson@rhul.ac.uk)

Introduction

Theory of Symmetric Encryption

SSL/TLS Record Layer Protocol

SSH Binary Packet Protocol

Concluding Remarks

# Introduction

- The aim of these two talks is to highlight some of the challenges in the design, implementation and security analysis of secure Internet protocols.
  - SSL/TLS and SSH as case studies.
  - Jean Paul Degabriele will cover IPsec in the student talks session.
- Some theory.
- Lots of practice!

# The Secure Channel Concept



- We frequently want to guarantee the confidentiality and integrity of data travelling over untrusted networks.
  - Typically, the Internet, but other networks too.
- Applications:
  - Branch office connectivity.
  - Connecting to business partners at remote site.
  - Remote access for employees.
  - Remote administration of network devices and servers.
  - E-commerce: protecting credit card numbers in transactions.
  - Secure file transfers.
  - ...

# The Secure Channel Concept



- This is achieved by building a *secure channel*.
- Typically this secure channel will offer:
  - Data origin authentication
  - Data integrity
  - Confidentiality
  - Anti-replay
- But usually not:
  - Non-repudiation
  - Any security services once data has been received
- NB: there is a theoretical definition for “secure channel” which we will be ignoring here.

# The Secure Channel Concept



*Practical* secure channels are usually constructed in 3 steps:

- An authenticated key establishment protocol.
  - During this one or both parties is authenticated and a fresh, shared secret is established.
  - Using asymmetric (public key) or symmetric cryptography, or a combination of the two.
- A key derivation phase.
  - MAC and symmetric encryption keys are derived from the shared secret established during protocol.
- And then further traffic protected using the derived keys.
  - MAC gives data integrity mechanism and data origin authentication.
  - Encryption gives confidentiality.
    - Usually mode of operation of block cipher (CBC, CTR) or stream cipher.
  - Using symmetric cryptography for speed.

# The Secure Channel Concept



- In these two talks, we are going to focus exclusively on the last step here.
- So we will assume key exchange and key derivation have already taken place securely.
  - This is by no means a trivial issue!
  - See talks by Michele for more on key exchange.
- And we will analyse the symmetric key cryptography elements in isolation.
  - A full security analysis of a secure protocol would need to consider all 3 elements and their possible interactions.

# Introduction

# Theory of Symmetric Encryption

# Symmetric Encryption



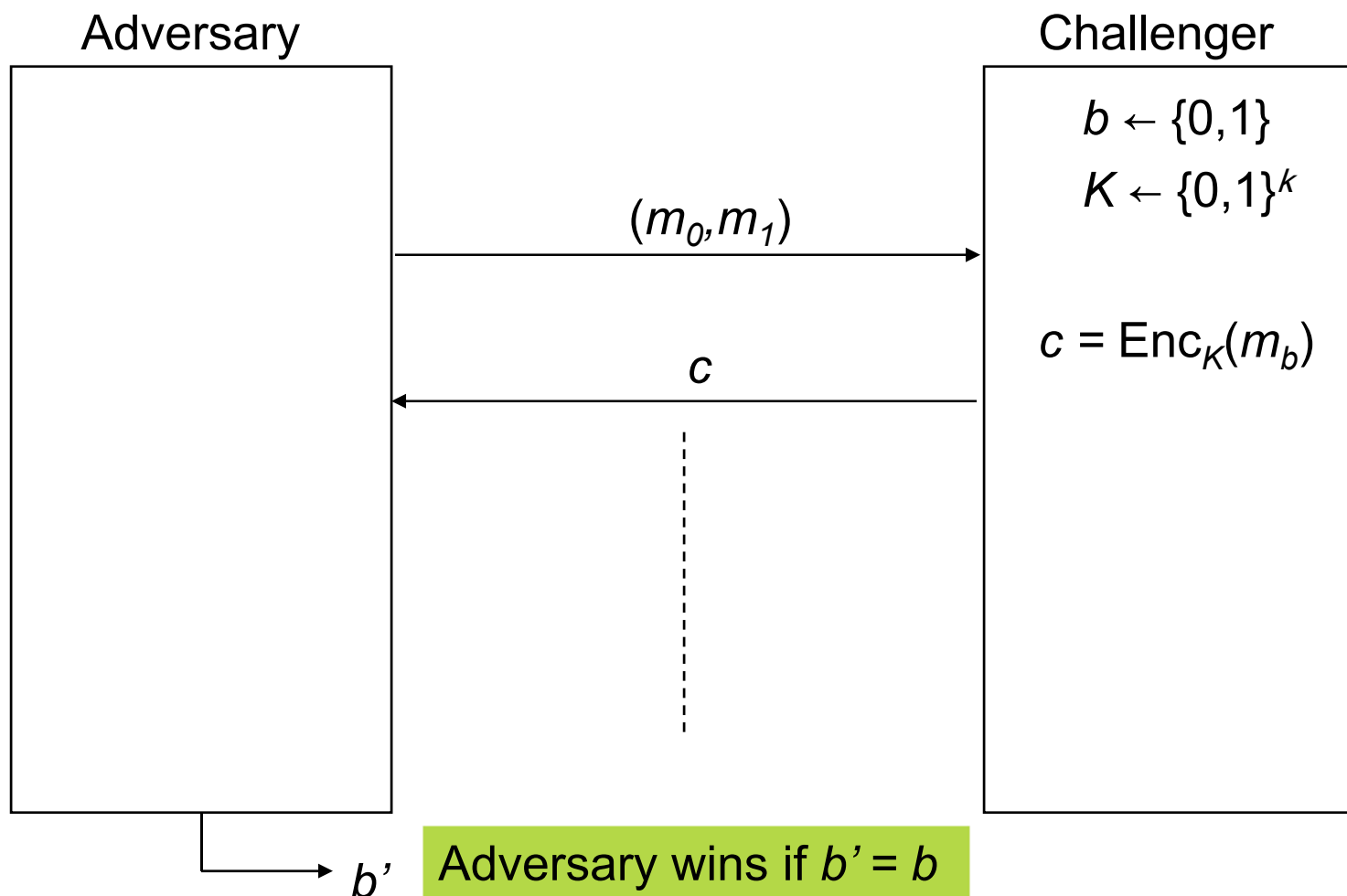
- Formal models for symmetric encryption are by now well established.
- Syntax of a symmetric encryption scheme:
  - $SE = (KGen, Enc, Dec)$ .
  - $KGen$  takes security parameter  $k$  as input and produces a symmetric key  $K$  of length  $k$ .
  - $Enc$  typically randomised;  $Dec$  usually not.
  - Obvious correctness requirement: decryption undoes encryption for everything in the space of plaintexts.
  - Decryption may fail, usually modelled by outputting  $\perp$ .

# Security for Symmetric Encryption



- IND-CPA security:
  - Adversary has access to *Left-Or-Right (LOR)* encryption oracle.
  - Adversary submits pairs of equal length messages  $(m_0, m_1)$  to this oracle.
  - Receives  $c$ , an encryption of either  $m_0$  or of  $m_1$ .
  - Adversary has to decide which message is being encrypted.
  - Adversary wins if it decides correctly.
- Originally called LOR-CPA security in [BDJR97].
- Formalised as a security game between the adversary and a challenger.

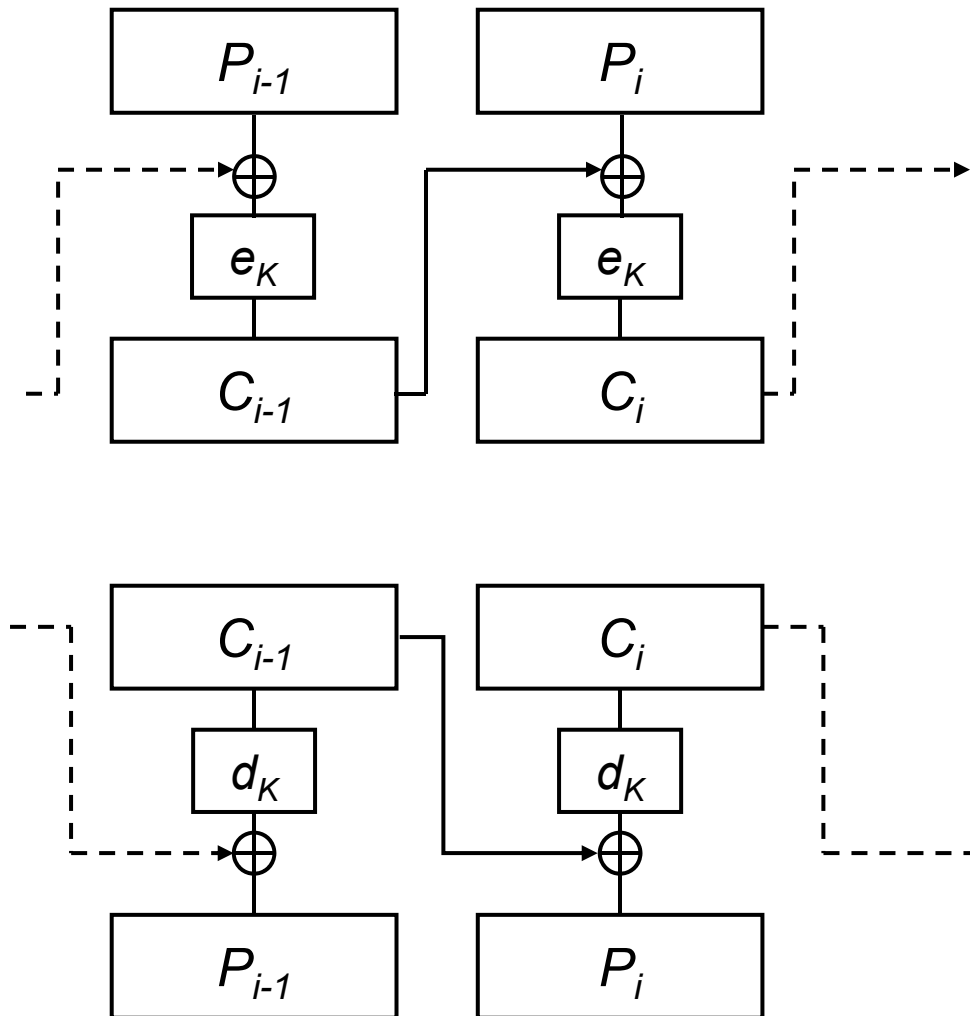
# IND-CPA Security



- Adversary's advantage is defined to be:
$$|\Pr(b' = b) - 1/2|.$$
- The scheme SE is said to be IND-CPA secure if the advantage is “small” for any adversary using “reasonable” resources.
  - IND = “Indistinguishable”.
  - CPA = Chosen Plaintext Attack.
  - Here, “small” and “reasonable” can be formalised using either an asymptotic approach or a concrete approach.

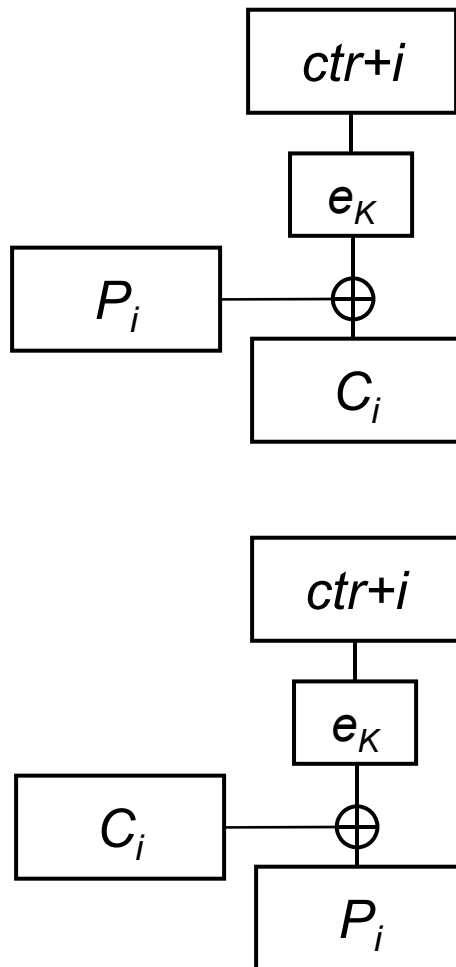
- Informally, the IND-CPA notion is a computational version of Shannon's notion of perfect secrecy.
  - Ciphertext leaks nothing about the plaintext.
  - A much stronger security notion than requiring the adversary to recover plaintext.
- Easy to achieve IND-CPA security using suitable mode of operation of block cipher:
  - Block cipher in CBC mode with random IVs;
  - Block cipher in CTR mode
  - Requires modelling of block cipher as PRP/PRF.
  - See [BDJR97] for analysis.

# Reminder of CBC Mode



- Initialisation Vector (IV) defines  $C_0$  for processing first block.
- Usually selected at random or set to be last block of previous ciphertext (IV chaining).
- CBC mode needs some form of padding if plaintext lengths are not already a multiple of block length.
- IND-CPA secure assuming block cipher is a PRF.

# Reminder of CTR Mode



- CTR mode uses a block cipher to build a stream cipher.
- Some initial value chosen for  $ctr$ .
  - Random or fixed
- Encrypt blocks  
 $ctr, ctr+1, ctr+2, \dots$   
to create a sequence of ciphertext blocks.
- Use this sequence as keystream.
- Same process to decrypt.
- IND-CPA secure assuming block cipher is a PRF.

# Other Types of CPA security



- ROR-CPA: adversary is given ROR oracle which returns either encryption of  $m$  or encryption of random message  $r$  of length  $|m|$ .
- FTG-CPA: adversary operates in stages:
  - Initial access to an encryption oracle.
  - Then selects 2 messages  $m_0, m_1$  and is given encryption  $c$  of one of them,  $m_b$ .
  - Then further access to encryption oracle, after which adversary outputs guess for  $b$ .
- SEM-CPA: adversary tries to guess a function  $f$  of a message  $m$  drawn from some distribution, given only a ciphertext  $c$ .
- [BDJR97] shows that all these notions are essentially equivalent to one another and to IND-CPA security.

# Motivating Stronger Security



- IND-CPA security only guarantees confidentiality when the adversary has access to some form of encryption oracle.
- In practice, an attacker may in addition have access to a (partial) decryption oracle.
  - For example, by interacting with an on-line entity and observing network messages that it produces.
  - These might be plaintexts or error messages indicating that decryption has failed.

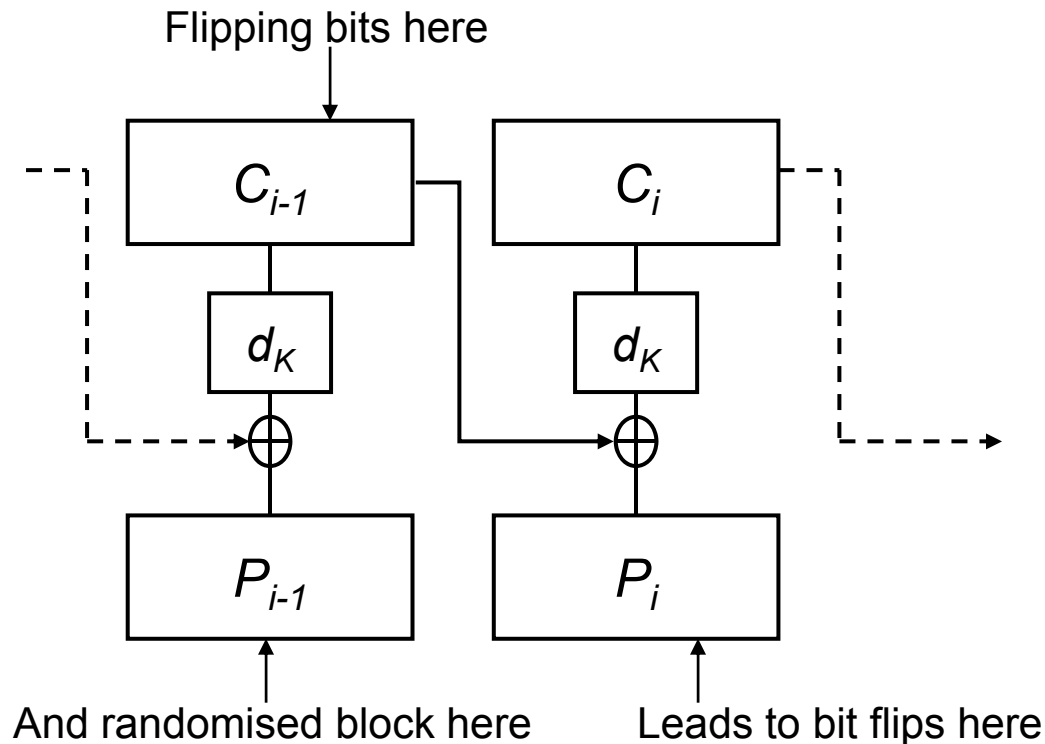
# Motivating Stronger Security



- IND-CPA security provides no integrity guarantees for plaintexts or ciphertexts.
- In practice, an attacker may be able to manipulate ciphertexts so as to make controlled changes to underlying plaintexts.
- Does not break IND-CPA security, but is clearly undesirable for a secure channel.
  - Wrong plaintext may be delivered to user.
- We really want a guarantee of integrity as well as confidentiality.

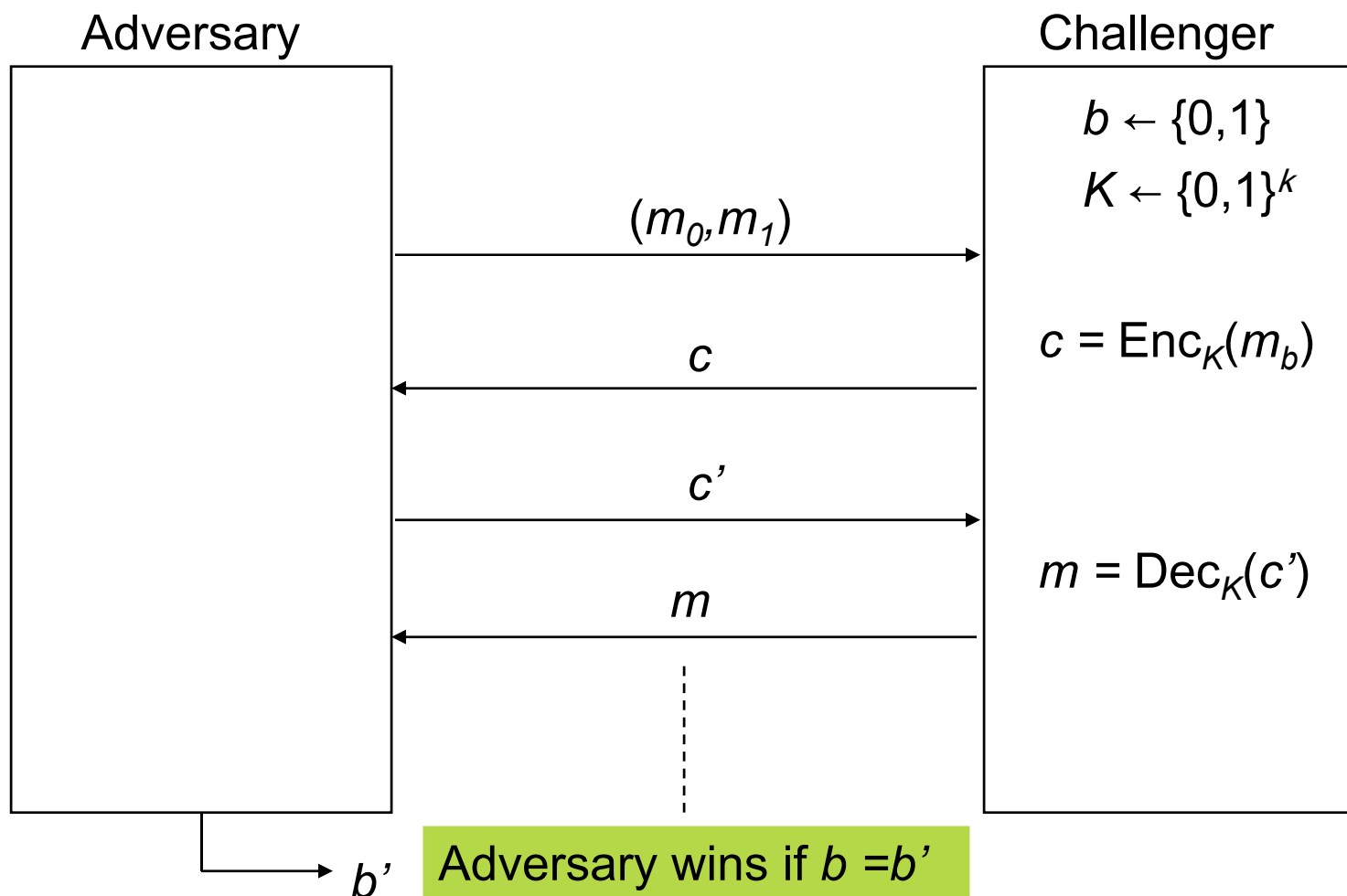
# Bit Flipping in CBC Mode

- Flipping bits in ciphertext block  $C_{i-1}$  leads to controlled changes in plaintext block  $P_i$  upon decryption.
- But block  $P_{i-1}$  is randomised (unless  $i=1$ ).



- IND-CCA security:
  - Attacker now has access to LOR encryption oracle and a decryption oracle.
  - LOR encryption oracle as before.
  - Decryption oracle takes any  $c$  as input, and outputs either  $\text{Dec}_K(c)$ , 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.
- All basic modes of operation are insecure in this model!

# IND-CCA Security



# Authenticated Encryption



- IND-CCA security does not provide any integrity guarantees for plaintexts or ciphertexts.
- In fact, there are encryption schemes that are IND-CCA secure, but where an attacker can modify a valid ciphertext  $c$  so as to produce another valid ciphertext  $c'$ .
  - Hence an attacker can get wrong message delivered to user without being able to break IND-CCA security!

# Authenticated Encryption



- [BN00] introduced security notions of *integrity of plaintexts* and *integrity of ciphertexts*.
- In each case, adversary is given access to a normal encryption oracle.
- INT-PTXT: infeasible to produce a valid ciphertext  $c$  decrypting to a message  $m$  that was not queried to the encryption oracle.
- INT-CTXT: infeasible to produce a valid ciphertext  $c$  that was not output by the encryption oracle.
- Intuitively, both notions prevent an attacker from being able to deliver the wrong message to a user.
  - But neither notion says anything about *replay attacks*.

# Authenticated Encryption



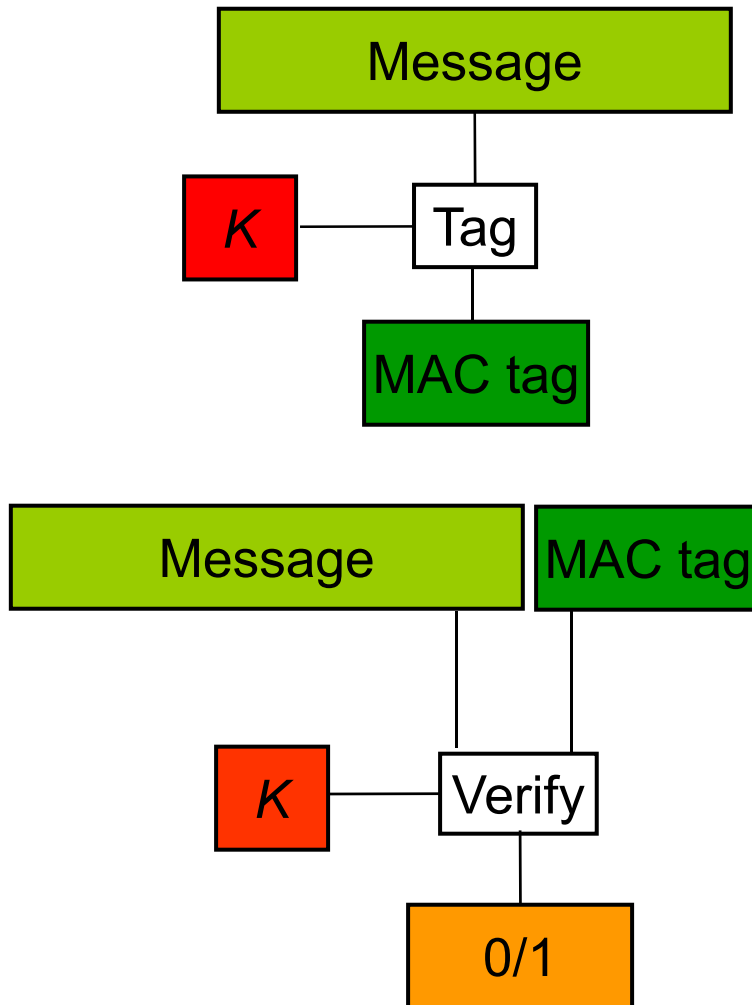
- [BN00] proved the following results:
  - INT-CTXT  $\rightarrow$  INT-PTXT
  - IND-CPA + INT-CTXT  $\rightarrow$  IND-CCA
  - IND-CCA  $\not\rightarrow$  INT-PTXT
- A scheme that is both INT-CTXT secure and IND-CPA secure is said to provide *Authenticated Encryption*.

# Achieving Strong Security Using MACs



- Message Authentication Codes (MACs) provide authenticity/integrity protection for messages.
  - Symmetric analogue of a digital signature.
- Syntax:  $\text{MAC} = (\text{KGen}, \text{Tag}, \text{Verify})$ 
  - Algorithm  $\text{Tag}$  has as input a key  $K$ , a message  $M$  of arbitrary length, and outputs a short MAC tag  $\tau$ .
  - Algorithm  $\text{Verify}$  has as input a key  $K$ , a message  $M$ , a MAC tag  $\tau$  and outputs 0 or 1, indicating correctness of tag  $\tau$ .
- HMAC is a general method for building a MAC scheme from a hash function.
  - Very widely used in secure protocols.
  - Typical output size of 96-256 bits.

# MACs



- Key security requirement is *unforgeability*:
  - Having seen MAC tags for many messages, an adversary cannot create the correct MAC tag for another chosen message.
- *Strong* versus *weak* unforgeability.

# Strong and Weak Unforgeability



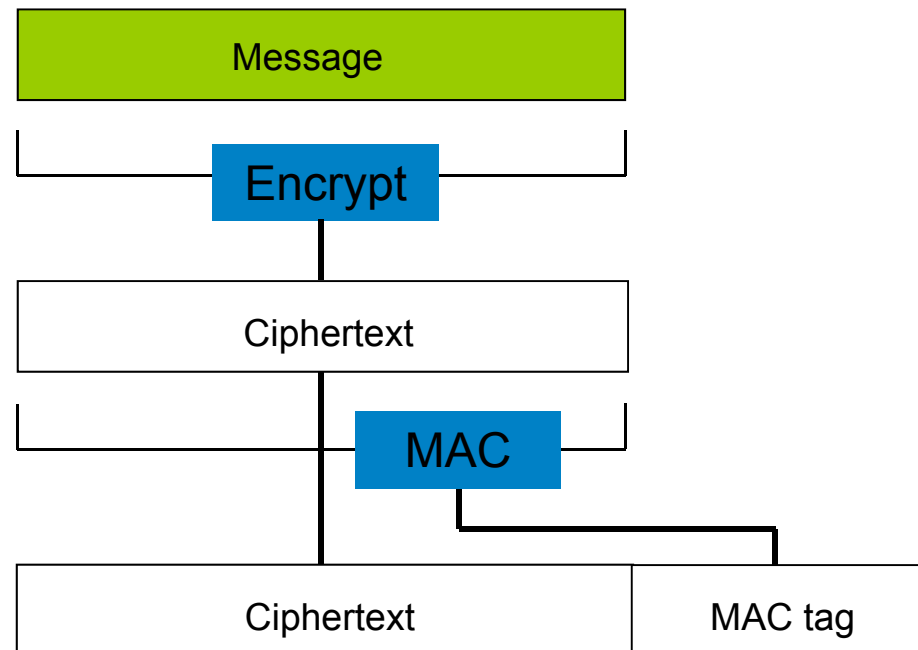
- Two notions of security for a MAC scheme.
- In both cases, adversary has access to an oracle for  $\text{Tag}_K(\cdot)$ 
  - Chosen message attack (CMA) setting.
- *Weak* unforgeability:
  - Adversary wins if it can create a valid MAC tag for a message not queried to oracle.
- *Strong* unforgeability:
  - Adversary wins if it can create a valid MAC tag for a message not queried to oracle OR if it can create a new valid MAC tag for a message previously queried to oracle.

# Achieving Strong Security



- [BN00] considered how to achieve strong security through *generic composition* of an IND-CPA secure encryption scheme and a (S/W)UF-CMA secure MAC.
  - Encrypt-then-MAC
  - Encrypt-and-MAC
  - MAC-then-Encrypt
- Also available: dedicated algorithms achieving AE – e.g. CCM, EAX, GCM, OCB,...

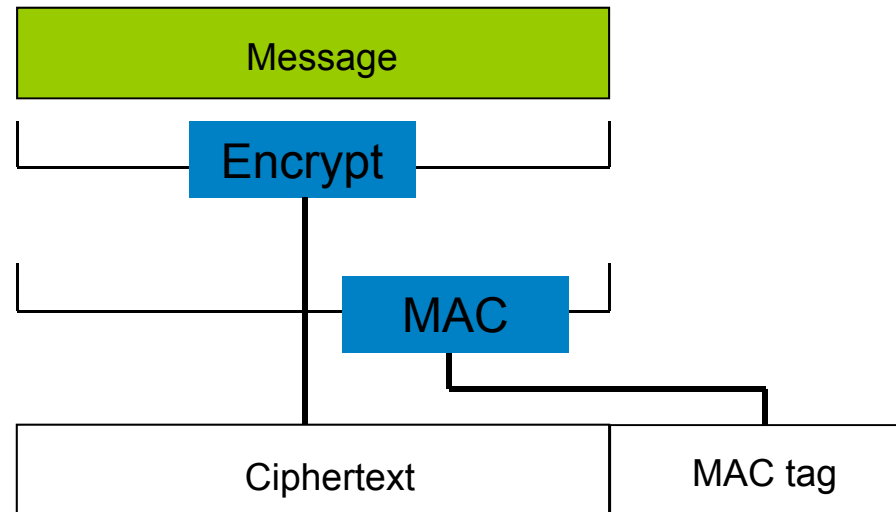
# Encrypt-then-MAC



# Encrypt-then-MAC

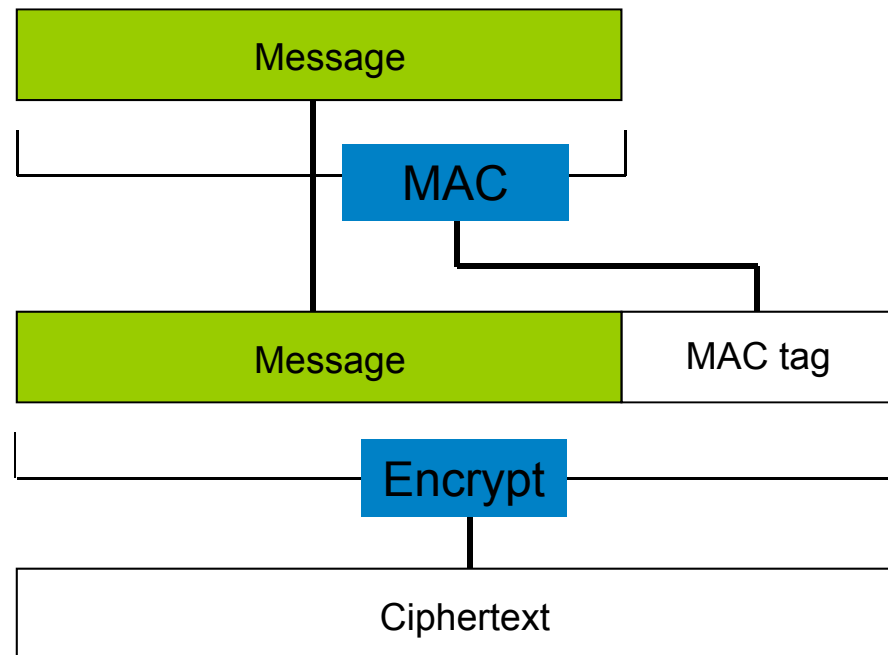
- As used by IPsec ESP in “enc+auth” configuration.
- If SE is IND-CPA secure and MAC is **W**UF-CMA secure, then the “Encrypt-then-MAC” composition is INT-PTXT and IND-CPA secure.
  - But neither INT-CTXT nor IND-CCA secure in general.
- If SE is IND-CPA secure and MAC is **S**UF-CMA secure, then the “Encrypt-then-MAC” composition is INT-CTXT and IND-CPA secure.
  - And hence also IND-CCA secure.

# Encrypt-and-MAC



- Insecure in general, even with SUF-CMA MAC and IND-CPA secure encryption.
  - Because a MAC tag for a SUF-CMA secure scheme can still leak plaintext information.
  - e.g. including first bit of plaintext in MAC tag does not break SUF-CMA security of MAC but does break confidentiality properties of composition.
- But specific instantiations may still be IND-CCA secure.
  - e.g. encode-then-encrypt-and-MAC as used in SSH [BKN02,PW10].

# MAC-then-encrypt



# MAC-then-encrypt



- An approximation of the composition used most commonly by SSL/TLS.
- If MAC is WUF-CMA secure, then MAC-then-encrypt provides INT-PTXT security.
- If SE is IND-CPA secure, then MAC-then-encrypt provides IND-CPA security.
- But this composition is not IND-CCA or INT-CTXT secure in general.
  - Encryption schemes that are IND-CPA secure but malleable can be used to show this.
- [K01] showed that MAC followed by CBC mode encryption or one-time pad provides a “secure channel functionality”.
  - And this result can be extended to show IND-CCA security in these special cases.

Generic compositions from SUF-CMA MAC and IND-CPA encryption:

- MAC-and-encrypt: **insecure**.
- Encrypt-then-MAC: **secure**.
- MAC-then-encrypt: **sometimes secure!**

- [BKN02] developed *stateful* security models for symmetric encryption.
  - Reflecting wide use of sequence numbers to prevent replay attacks in secure channel protocols.
- IND-SFCCA security:
  - Attacker has access to an LOR encryption oracle and a decryption oracle.
  - Both oracles are stateful (e.g. through use of sequence numbers).
  - Model allows adversary to advance states to any chosen value by relaying replies from LOR encryption oracle to his decryption oracle.
    - For these *in-sync* decryption queries, no plaintext is given to adversary.
  - Adversary wins game if he can guess hidden bit  $b$  of encryption oracle.

Introduction

Theory of Symmetric Encryption

SSL/TLS Record Layer Protocol

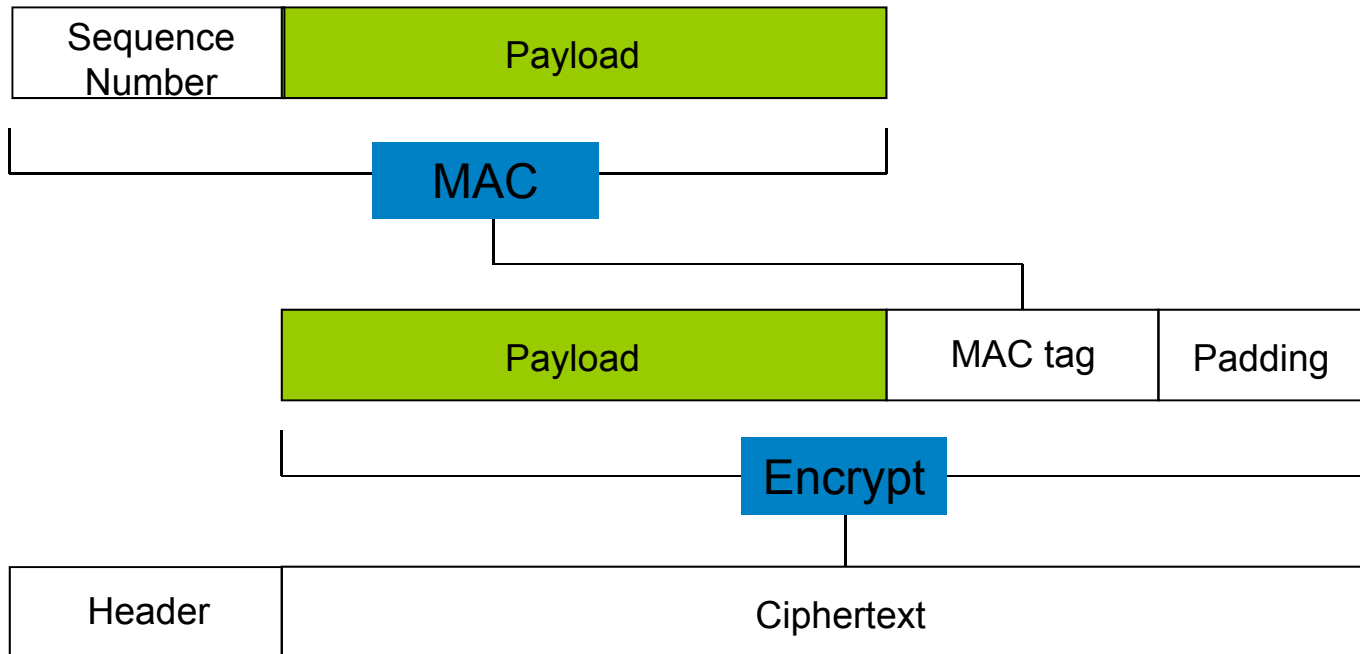
# SSL/TLS Overview



- SSL = Secure Sockets Layer.
  - unreleased v1, flawed but useful v2, good v3.
- TLS = Transport Layer Security.
  - IETF-standardised version of SSL.
  - TLS1.0=SSL3.0 with minor tweaks, RFC 2246.
  - TLS1.1=TLS1.0 with tweaks, RFC 4346 (2006).
  - TLS1.2=TLS1.1 with yet more tweaks, RFC 5246 (2008).
- SSL/TLS provides security ‘at TCP layer’.
  - Runs over TCP, using TCP to provide reliable, end-to-end transport.
  - Widely deployed in Web browsers and servers to support ‘secure e-commerce’ over HTTP.

- SSL/TLS Record Protocol provides:
  - Data origin authentication, integrity, anti-replay using a MAC and sequence number.
    - Algorithms supported in TLS1.2 are NULL, HMAC-MD5, HMAC-SHA1, HMAC-SHA256.
  - Confidentiality using symmetric algorithm.
    - Algorithms supported in TLS1.2 are NULL; 3DES, AES-128, AES-256 block ciphers, all in CBC mode; RC4-128 stream cipher.
- Earlier versions of SSL/TLS will support different sets of algorithms.
- Keys for the algorithms are supplied by the SSL/TLS Handshake Protocol.

# SSL/TLS Record Protocol (Simplified)



# Security of SSL/TLS Record Protocol



- A (stateful) MAC-then-encrypt construction.
- **Not** generically secure.
- But building on results of [K01], the basic MAC-then-encrypt construction is IND-CCA secure for CBC mode encryption.
- So we are OK, right?
- Well, SSL/TLS is actually a *MAC-then-pad-then-encrypt* construction...

# SSL/TLS Record Layer Padding



- Padding in SSL/TLS has a particular format:
  - Add a sequence of bytes after the MAC to complete the last plaintext block.
  - If  $t$  bytes are needed, then add  $t$  copies of the byte representation of  $t$ .
  - e.g. 01,  
    02 02,  
    03 03 03,...
  - Sender is allowed to include up to 255 bytes of padding.
- So what should happen if the padding format after decryption of a received ciphertext is incorrect?

# SSL/TLS Record Layer Padding



- The specification for TLS1.0 in RFC 2246 describes possible error messages, including:  
`decryption_failed`: A TLS Ciphertext decrypted in an invalid way: either it wasn't an even multiple of the block length or its padding values, when checked, weren't correct.
- This suggests that implementations should check the format of padding and terminate the connection if the padding format is incorrect.
- What are the security consequences of this?

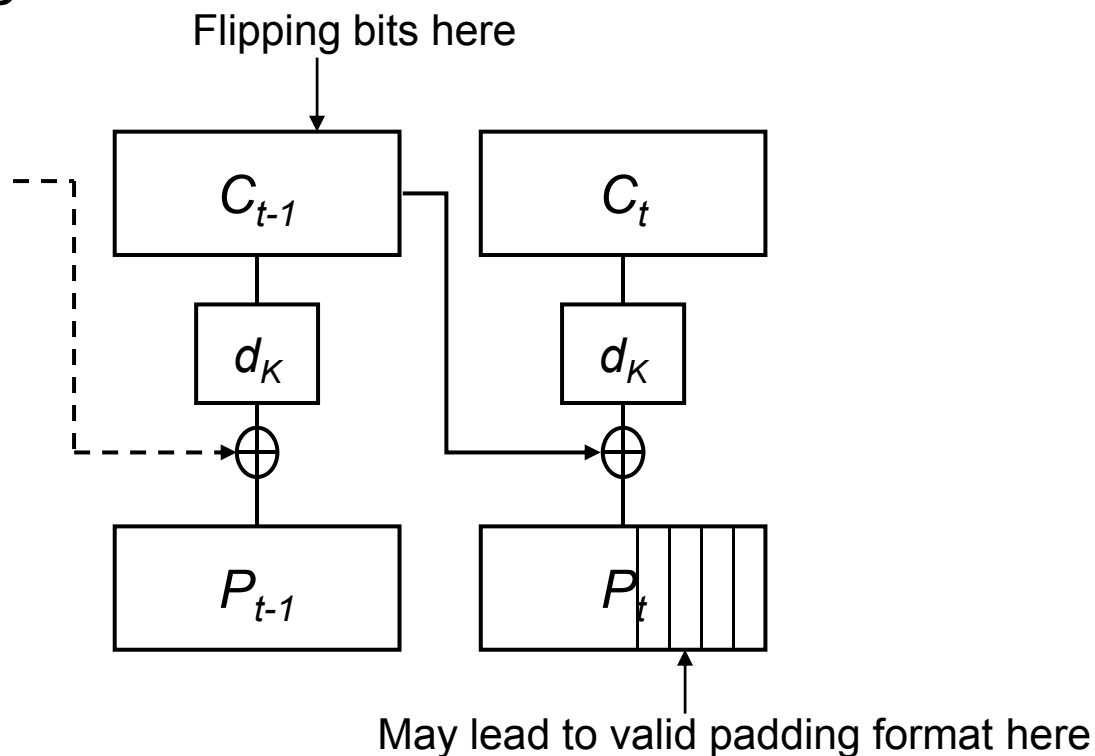
# Padding Oracle Attacks



- Vaudenay [V02] proposed the concept of a *padding oracle attack*.
- A padding oracle  $P$  takes as input a ciphertext and outputs a single bit indicating whether the underlying plaintext is correctly padded or not.
  - According to some fixed padding rule and some fixed key  $K$ .
- Vaudenay [V02] showed that, for CBC mode and for certain padding schemes, a padding oracle can be used to build a decryption oracle.
- Let's look at SSL/TLS-style padding....

# SSL/TLS Padding Oracle Attack

- Place target block  $C_t$  as last block of ciphertext.
- Flipping bits in last-but-one ciphertext block may lead to correct padding in last plaintext block.
- Padding oracle will tell attacker when this occurs.

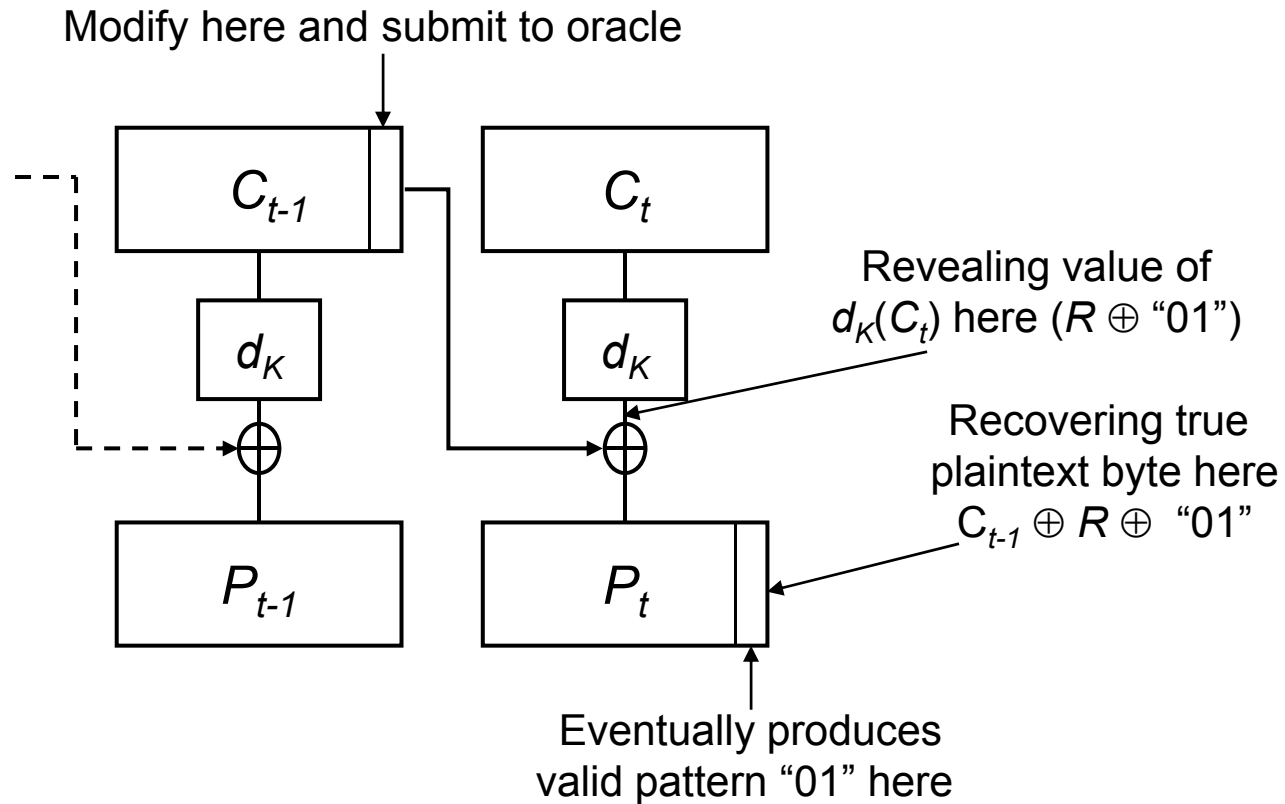


# SSL/TLS Padding Oracle Attack



- For SSL/TLS, most likely “correct” output arises from valid padding pattern of length 1, namely “01”.
- So:
  - Select a random block  $R$ .
  - Repeatedly modify last byte of  $R$  and submit ciphertext ending with blocks  $R, C_t$  to padding oracle.....

# SSL/TLS Padding Oracle Attack



# SSL/TLS Padding Oracle Attack



- An average of 128 trials are needed to extract the last byte of each plaintext block.
- Can extend to the entire block.
  - Now that last byte of  $P_t$  is known, we can modify  $R$  to set last plaintext byte to “02”.
  - Then modify second-to-last byte of  $R$  until padding oracle returns “correct”.
  - Most likely padding pattern is now “02 02”.
  - Can now recover second last byte of  $P_t$ .
  - Repeat...
- Can extend to multiple blocks.
  - Can place any target block as last block of ciphertext.

# Padding Oracles In Practice



- So is SSL/TLS vulnerable to a padding oracle attack?
- Sequence of operations during decryption:
  1. CBC-decrypt;
  2. check padding;
  3. check MAC.
- An error message can arise from either a failure of the padding check or a MAC failure.
- Padding oracle attack will produce an error of one type or the other.
  - MAC failure indicates correct padding.
- If the errors are *distinguishable*, then a padding oracle attack may be possible....

# Padding Oracles In Practice



- The error messages are different:
  - `bad_record_mac`
  - `decryption_failed`
- But they are encrypted, so cannot be seen by the attacker.
- And an error of either type is *fatal*, leading to SSL/TLS connection being torn-down.
- Meaning that the attacker only gets one chance to call the oracle.

## Canvel *et al.* [CHVV03]:

- A MAC failure error message is likely to appear on the network **later** than a padding failure error message.
- So *timing* the appearance of error messages can give us the required padding oracle.
- Because the errors are fatal, the attacker can only learn one byte of plaintext, and then with probability  $1/256$ .
- (But this does also give a probability 1 distinguishing attack.)

# OpenSSL and Padding Oracles



- Canvel *et al.* [CHVV03] also showed:
  - The attacker can still decrypt if a *fixed* plaintext is repeated in a *fixed* location in many SSL/TLS connections.
  - As is the case when SSL/TLS is used to protect an Outlook password, for example.
  - Timing differences between padding failures and MAC failures *were* detectable on a LAN for the then-current OpenSSL implementation of SSL/TLS.

# OpenSSL and Padding Oracles



- OpenSSL was subsequently patched to ensure uniform reporting and timing of error messages, thus preventing the attack.
- Advice to implementers in RFC 4346 (TLS1.1):

The receiver **MUST** check this padding and **SHOULD** use the `bad_record_mac` alert to indicate padding errors.

In order to defend against this attack, implementations **MUST** ensure that record processing time is essentially the same whether or not the padding is correct. In general, the best way to do this is to compute the MAC even if the padding is incorrect, and only then reject the packet.

- The result of [K01], while valuable, is not the whole story about security of the SSL/TLS Record Protocol.
  - Even if the subtitle of [K01] is “*How secure is SSL?*” 😊
- The IETF response was NOT to switch to a cryptographic construction that is more robustly supported by theory (e.g. encrypt-then-MAC).
  - Instead, they recommend the adoption of attack-specific countermeasures.

- The attack of [CHVV03] shows the importance of looking beyond the basic encryption and MAC algorithm components of secure channels
  - Order/timing/error events for cryptographic and other operations matters.
  - Padding matters.
  - (Statefulness/sequence numbers matter.)

- Padding oracles may seem esoteric, but can be difficult to avoid in practice.
  - e.g. in IPsec, an RFC-compliant implementation will check an extended padding format and silently drop the packet if check fails, and forward the packet if it passes.
    - Basis for various attacks against IPsec in [DP07,DP10].
  - e.g. in SSL/TLS, an RFC-compliant implementation will issue an error message if the padding check fails; it is up to the implementation to ensure this does not introduce a timing side-channel.

Introduction

Theory of Symmetric Encryption

SSL/TLS Record Layer Protocol

SSH Binary Packet Protocol

# Introducing SSH



*Secure Shell or SSH is a network protocol that allows data to be exchanged using a secure channel between two networked devices. Used primarily on Linux and Unix based systems to access shell accounts, SSH was designed as a replacement for TELNET and other insecure remote shells, which send information, notably passwords, in plaintext, leaving them open for interception. **The encryption used by SSH provides confidentiality and integrity of data over an insecure network, such as the Internet.***

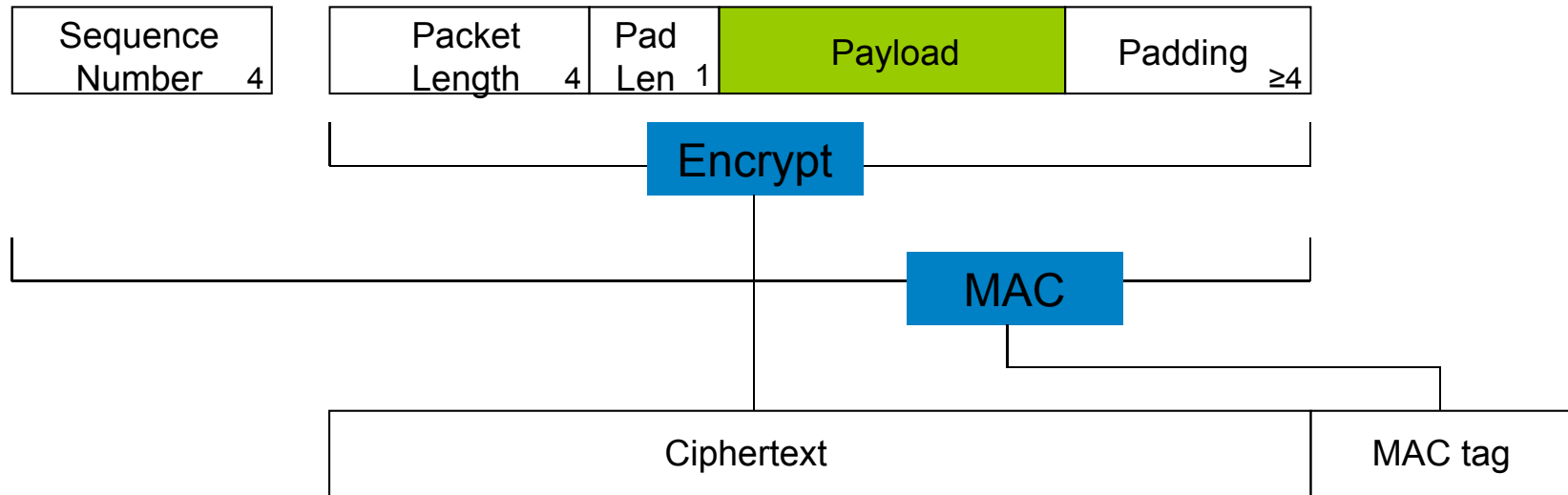
– Wikipedia

# Introducing SSH



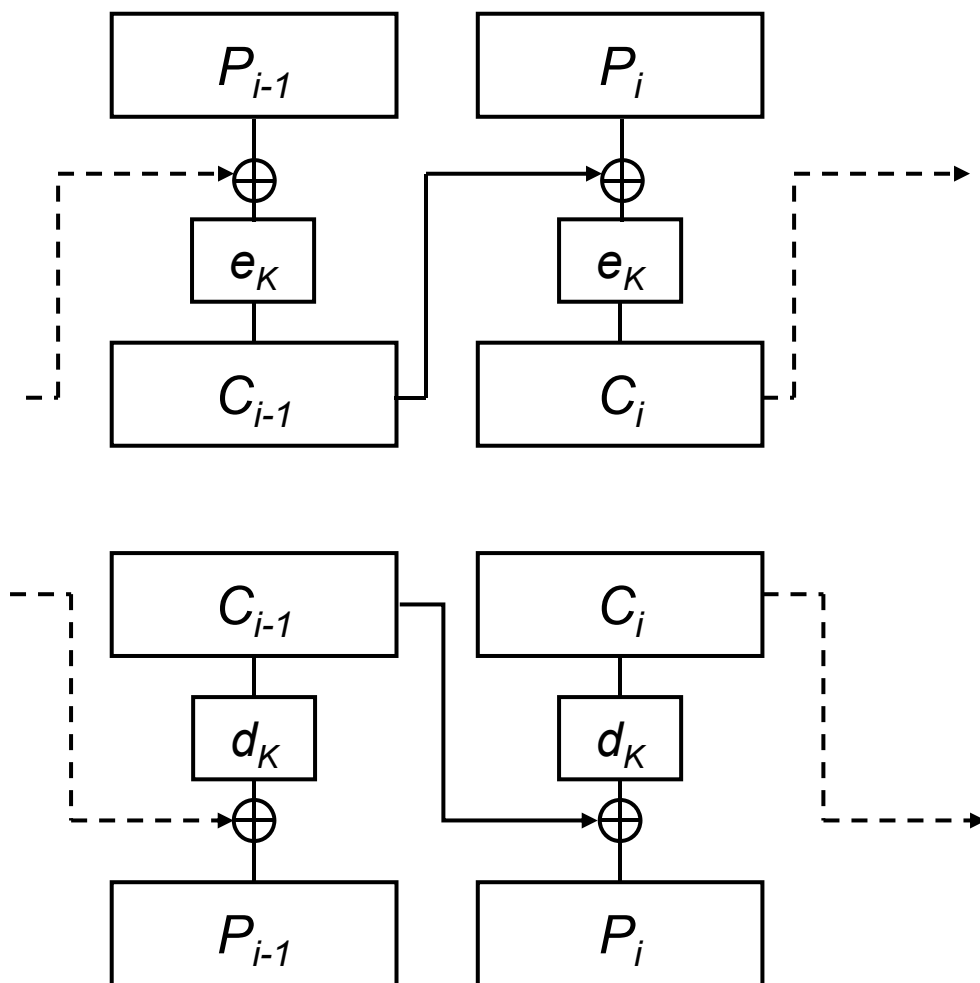
- SSHv1 had several security flaws.
  - Worst ones arising from use of CRC algorithm to provide integrity.
  - Enabling, for example, traffic injection attacks.
- SSHv2 was standardised in 2006 by the IETF in RFCs 4251-4254.
- RFC 4253 specifies the SSH Binary Packet Protocol (BPP).
  - Provides the secure channel function for SSH.
- SSHv2 is widely regarded as providing strong security.
  - Dozens of implementations.
  - Widely used to enable secure remote administration of sensitive systems.
  - One minor flaw in the BPP that allows low probability distinguishing attacks ([D02]; [BKN02]).

# The SSH BPP



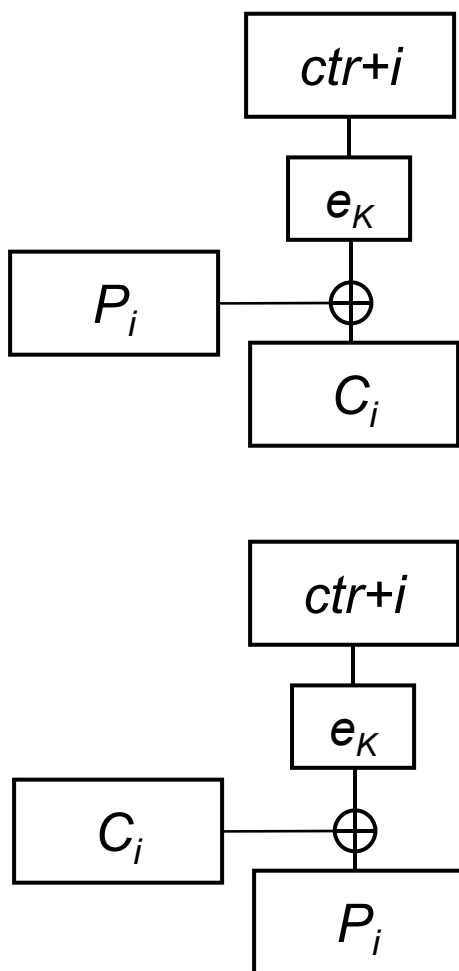
- Encode-then-Encrypt&MAC construction, **not** generically secure.
- Packet length field measures the size of the packet on the wire in bytes and is encrypted to hide the true length of SSH packets.
- Variable length padding is permissible; padding needed for CBC mode and carried over to CTR mode.
- SSH requires support for HMAC-SHA1 and recommends support for HMAC-SHA1-96.

# CBC Mode in SSH



- RFC 4253 mandates 3DES-CBC and recommends AES-CBC.
  - In fact, all originally specified optional configurations involve CBC mode, and ARCFOUR was the only optional stream cipher.
- SSH uses a chained IV in CBC mode:
  - IV for current packet is the **last** ciphertext block from the **previous** packet.
  - Effectively creates a single stream of data from multiple SSH packets.

# CTR Mode in SSH



- CTR mode uses block cipher to build a stream cipher.
- CTR mode for SSH standardised in RFC 4344.
  - Initial value of counter is obtained from handshake protocol.
  - Packet format is preserved from CBC case.
  - Recommends use of AES-CTR with 128, 192 and 256-bit keys, and 3DES-CTR.

# Security of the SSH BPP



- Attack of [D02], [BKN02] exploits chained IVs in CBC mode.
  - Breaks IND-CCA security of the SSH BPP.
    - Attacker can distinguish which one of two chosen messages was encrypted.
  - Low success probability against SSH implementations because of specifics of packet format.
  - Prevented in OpenSSH by optional use of dummy packets to hide IVs until it is too late for attacker to make use of them.
- **Basic message:** SSH BPP using CBC mode with chained IVs is **in**secure according to the standard theoretical notion of security.

# Security of the SSH BPP



- Using the IND-SFCCA model, [BKN02] proved the security of variants of the SSH BPP under reasonable assumptions concerning:
  - The encryption component.
    - Essentially, IND-CPA security.
  - The MAC component.
    - Strong unforgeability and pseudo-randomness.
  - The randomness of the padding scheme.
  - Collision properties of the encoding scheme.
    - In practice, for SSH BPP, this means not too many packets can be encrypted.

# Security of the SSH BPP



- In particular, [BKN02] established IND-SFCCA security of SSH-\$NPC and SSH-CTR.
  - SSH-\$NPC = SSH using a block cipher in CBC mode with explicit, per-packet, random IV and with random padding.
    - In contrast to chained IVs used in SSH BPP.
  - SSH-CTR = SSH using a block cipher in counter mode, with counter maintained at sender and receiver.

# Attacking the SSH BPP



- [APW09]: plaintext recovering attacks against SSH BPP.
  - Much stronger than distinguishing attack of [D02]!
- These attacks exploit the interaction of the following features of the BPP specification:
  - The attacker can send data on an SSH connection in small chunks (TCP).
  - CBC mode is mandated.
  - A MAC failure is visible on the network.
  - The packet length field encodes how much data needs to be received before the MAC is received and the integrity of the packet can be checked.

# Attacking the SSH BPP (Theory)

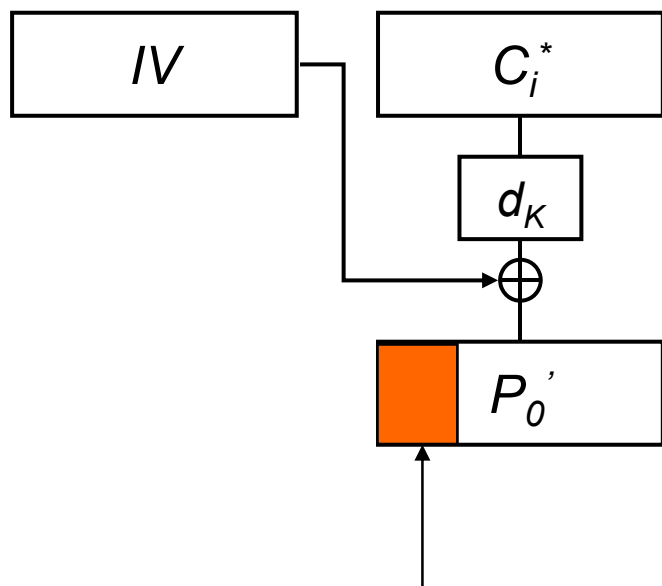


- The attacker monitors an SSH connection and selects any target ciphertext block  $C_i^*$ . Here:

$$C_i^* = e_K(C_{i-1}^* \oplus P_i^*), \text{ i.e. } P_i^* = C_{i-1}^* \oplus d_K(C_i^*)$$

- The attacker injects  $C_i^*$  so it is seen as the *first* block of a new SSH packet by the receiver...

# Attacking the SSH BPP (Theory)

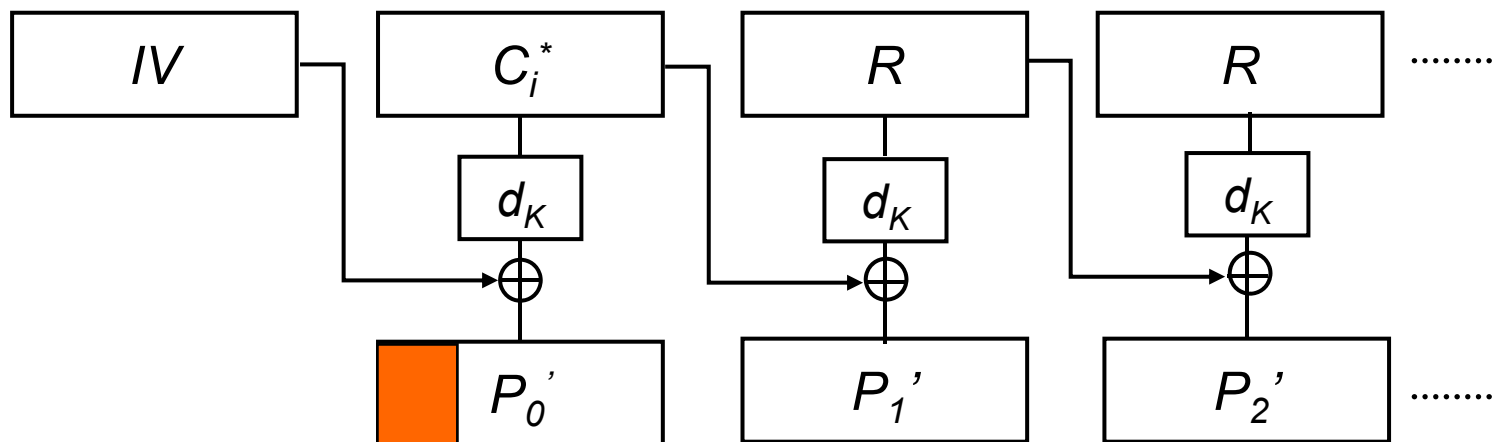


The receiver will treat the first 32 bits of the calculated plaintext block as the packet length field for the new packet. Here:

$$P_0' = IV \oplus d_K(C_i^*)$$

where  $IV$  is known from the previous packet.

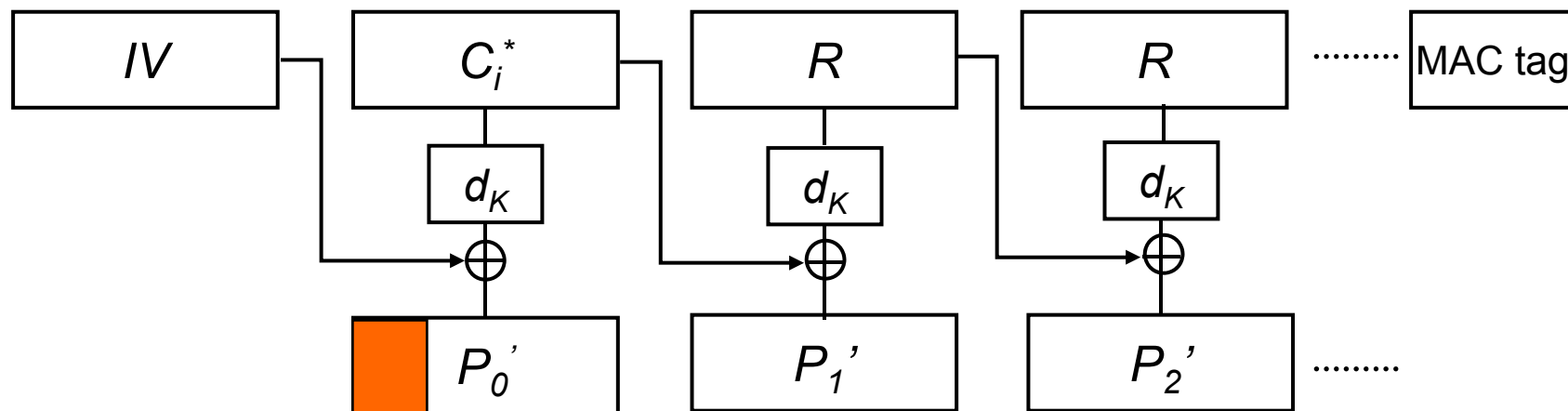
# Attacking the SSH BPP (Theory)



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.

# Attacking the SSH BPP (Theory)



- Eventually, once enough data has arrived, the receiver will receive what it thinks is the MAC tag.
- The receiver will then check the MAC.
  - This 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?

# Attacking the SSH BPP (Theory)



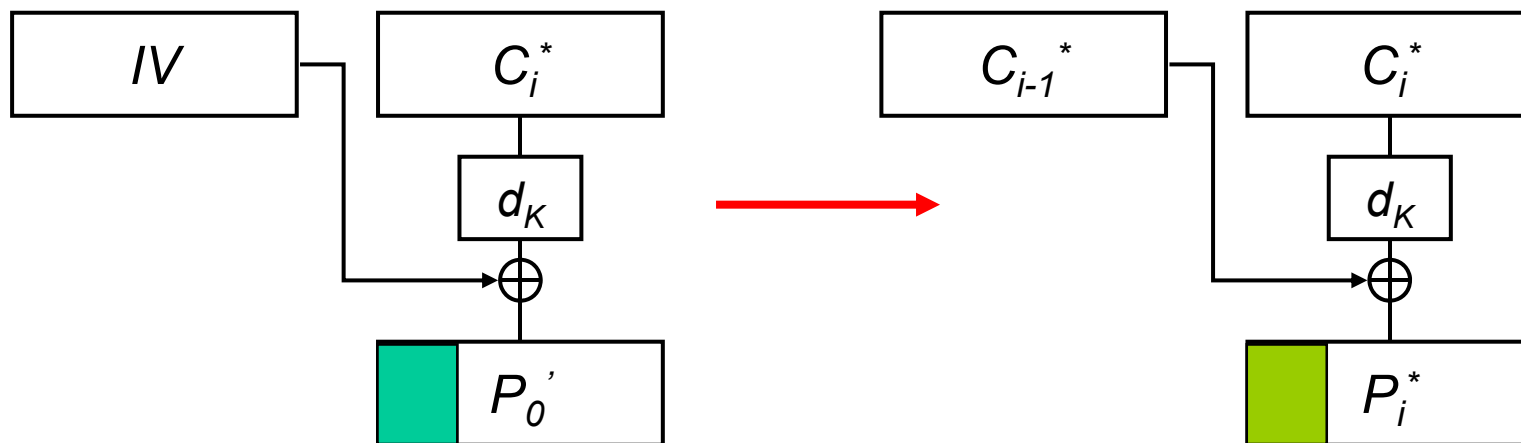
- The receiver **must** use the packet length field to decide when the MAC tag has arrived.

Implementations SHOULD decrypt the length after receiving the first 8 (or cipher block size, whichever is larger) bytes of a packet.

- Hence an attacker who counts the number of bytes needed to cause connection termination can learn the packet length field.
- That is, the attacker learns the first 32 bits of:

$$P_0' = IV \oplus d_K(C_i^*).$$

# Attacking the SSH BPP (Theory)



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

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

# Attack Performance (Theory)



- As described, this simple attack succeeds in recovering 32 bits of plaintext from an arbitrary ciphertext block with probability 1.
  - But requires the injection of about  $2^{31}$  random bytes to trigger the MAC check.
  - And leads to an SSH connection tear-down.
- The attack breaks the SSH BPP.
- The attack still works if a fresh IV is used for each new SSH packet.
  - Breaking SSH-NPC that was proven secure in [BKN02].

# Attacking OpenSSH



- OpenSSH is the most popular implementation of the SSH RFCs.
  - Open-source, distributed as part of OpenBSD.
  - OpenSSH webpages state that OpenSSH accounts for more than 80% of all deployed SSH servers.
  - [www.openssh.org/usage/index.html](http://www.openssh.org/usage/index.html)
- We worked with OpenSSH 5.1.
  - Version 5.2 released 23/02/2009 partly as a consequence of our work, current version is 5.6.

# Attacking OpenSSH



- In OpenSSH 5.1, two sanity checks are carried out on the packet length field after the first block is decrypted.
- This is in-line with advice to implementers in RFC:  
`implementations SHOULD check that the packet length is reasonable in order for the implementation to avoid denial of service and/or buffer overflow attacks`
- First check:  $5 \leq \text{length} \leq 2^{18}$ .
- Second check:  $\text{length} = 12 \pmod{16}$  (for 128 bit block cipher).

# Attacking OpenSSH



- When each of the checks fails, the SSH connection is terminated in subtly different ways.
  - This difference leaks some information, but also reduces success probability of the attack.
- If the length checks pass, then OpenSSH 5.1 waits for more bytes.
- Finally, when the MAC check fails, a third type of connection termination is seen.

# Attacking OpenSSH



- The manner in which OpenSSH 5.1 behaves on failure allows an attack recovering 32 bits of plaintext.
  - Success probability  $2^{-18}$  (for a 128-bit block cipher).
  - Requires injection of (roughly)  $2^{18}$  bytes.
- Attacks result in termination of the SSH connection.
  - But the attacks can be iterated if a fixed plaintext is repeated across multiple connections.
- The attacks worked in practice.
  - Implemented in a virtualized environment with server code patched to boost success rate.

# Disclosure of the Attacks



- We worked with the UK Centre for Protection of National Infrastructure (CPNI) to disclose the attacks.
  - `www.cpni.gov.uk/Docs/Vulnerability_Advisory_SSH.txt`
  - Advisory published 14/11/2008.
  - Vendors notified well ahead of time, giving opportunity to prepare fixes.
  - Recommends switching to stateful counter mode encryption as specified in RFC 4344, since specific attack then no longer works.

# OpenSSH Reaction



- OpenSSH released OpenSSH 5.2 (23/02/2009).
  - Offers AES in counter mode and arcfour256 stream cipher ahead of CBC mode block ciphers.
- [www.openssh.org/txt/release-5.2](http://www.openssh.org/txt/release-5.2):
  - *“This release also adds countermeasures to mitigate CPNI-957037-style attacks against the SSH protocol’s use of CBC-mode ciphers.”*

OpenSSH CBC mode countermeasure:

- If length checks fail, then set length field to  $2^{18}$  and carry on.
- This renders OpenSSH *more* vulnerable to DoS attacks!
- And there are still plaintext recovery and distinguishing attacks.
  - Attacker who knows a certain set of 18 bits of a block can recover a further 14 bits.
  - Special case gives distinguishing attack.

# Possible Countermeasures



- Randomise the length field if the length checks fail.
  - Still vulnerable to a distinguishing attack.
- Don't encrypt the length field.
  - Invasive and makes certain DoS attacks easier.
- Separately MAC the length field.
  - Invasive.
- Use authenticated encryption algorithm in place of SSH's *ad hoc* construction.
  - Invasive, and still can't safely encrypt the length field; for example, CBC-then-MAC would still be breakable.
- Use CTR mode.

# Impact of the Attacks

- The specific attacks are easily circumvented by switching to CTR mode.
- The attacks have quite low probability and recover limited amounts of plaintext.
- But SSH was meant to be bullet-proof, and our attacks are really quite simple.
  - This raises many questions!
- Our basic attack applies to the proven secure variant SSH- $\$NPC$ .
  - Pointing to problems with the provable security approach used in [BKN02].
  - So what about their proof of security for SSH- $\$CTR$ ?

# Limitations of [BKN02]



- The security model of [BKN02] *does* model errors arising during the BPP decryption process.
  - Connection teardown is modeled by disallowing access to decryption and encryption oracles after any error event.
  - Errors can arise from decryption, decoding or MAC checking.
- But only a single type of error message is output.
  - One of our attacks against OpenSSH exploits the fact that different error events *are* distinguishable.
- And the model assumes that decoding errors arise before MAC errors.
  - While the OpenSSH implementation only does decoding *after* the MAC has been checked.

# Limitations of [BKN02]



- The model assumes that plaintexts and ciphertexts are “atomic”.
  - All oracle queries in the model involve complete plaintexts or ciphertexts.
  - But the attacks exploit the ability to deliver ciphertexts one block (or even one byte!) at a time and observe behaviour.
    - For example, distinguishing the wait state from a MAC failure.
- The model does not allow for *plaintext-dependent* decryption.
  - The packet length field never appears in the model.
  - But implementations *must* make use of this field during the decryption process.
  - And, as we’ve seen, the manner in which this field is treated is critical for security.

# A New Security Analysis of SSH



- In [PW10], we:
  - Develop a new security model addressing limitations of the model used in [BKN02].
    - IND-BSF-CCA security.
    - Allows byte-by-byte delivery of ciphertexts to decryption oracle, and buffering of any as-yet-unprocessed ciphertext bytes.
    - Incorporates attacks from [APW09].
  - Build an accurate description of SSH-CTR as specified in RFCs and implemented in OpenSSH;
  - Prove the security of this description of SSH-CTR in the IND-BSF-CCA model.

# A New Security Analysis of SSH



- Our description of SSH-CTR involves:
  - Accurate modelling of errors, based on specification in RFCs and ‘C’ source code for OpenSSH.
    - Errors from length sanity checking.
    - Errors from MAC verification failure.
    - Errors from parsing failures during decoding.
    - Session teardown in event of any error.
  - Use of the packet length field from plaintext to determine the amount of ciphertext required before the MAC check is performed.
    - Plaintext-dependent decryption.

# Modelling the Encryption Algorithm



## Algorithm E-SSH-CTR<sub>Ke,Kt</sub> (*m*)

if  $st_e = \perp$  then

    return  $\perp$

$(m_e, m_t) = \text{encode}(m)$

if  $m_e = \perp$  then

$st_e \leftarrow \perp$

    return  $\perp$

else

$c \leftarrow E\text{-CTR}_{Ke}(m_e) \quad \backslash\backslash$  counter mode encryption

$\tau \leftarrow T_{Kt}(m_t) \quad \backslash\backslash$  MAC computation

    return  $c \parallel \tau$

end if

# Modelling the Decryption Algorithm



## Algorithm D-SSH-CTR<sub>Ke,Kt</sub>(c)

```
if  $st_d = \perp$  then
  return  $\perp$ 
end if
{Stage 1}
 $cbuff \leftarrow cbuff || c$ 
{Stage 2}
if  $m_e = \text{empty}$  and  $|cbuff| \geq L$  then
  Parse  $cbuff$  as  $c' || A$  (where  $|c'| = L$ )
   $m_e[1] \leftarrow D\text{-CTR}_{Ke}(c')$ 
   $LF \leftarrow \text{len}(m_e[1])$     \ \ len check
  if  $LF = \perp_L$  then
     $st_d \leftarrow \perp$ 
    return  $\perp_L$ 
  else
     $need \leftarrow 4 + LF + maclen$ 
  end if
end if
```

## **{Stage 3}**

```
if  $|cbuff| \geq L$  then
  if  $|cbuff| \geq need$  then
    Parse  $cbuff$  as  $c[1\dots n] || tau || B$ ,
    where  $|c[1\dots n] || tau| = need$ ,
    and  $|tau| = maclen$ 
     $m_e[2\dots n] \leftarrow D\text{-CTR}_{Ke}(c[2\dots n])$  \ \ CTR mode
     $m_e \leftarrow m_e[1] || m_e[2\dots n]$ 
     $m_t \leftarrow SN_d || m_e$ 
     $v \leftarrow V_{Kt}(m_t, tau)$     \ \ MAC checking
    if  $v = 0$  then                \ \ MAC failure
       $st_d \leftarrow \perp$ 
      return  $\perp_A$ 
    else
       $m \leftarrow \text{decode}(m_e)$  \ \ decoding plaintext
       $m_e \leftarrow \text{empty}; cbuff \leftarrow B$ 
      return  $m$ 
    end if
  end if
end if
```

**Theorem:** SSH-CTR is IND-BSF-CCA secure under the assumptions that:

- $F$ , the function family used to construct CTR mode is pseudo-random;
  - The MAC scheme is strongly unforgeable;
  - The MAC tagging algorithm is pseudo-random;
  - Minimal requirements on the length checking function  $\text{len}$  are met.
- The theorem can be made *concrete*.
    - The advantage of any IND-BSF-CCA adversary is meaningfully related to advantages of adversaries against  $F$  and the MAC.
    - Good for practice!

# What Does the Proof Mean?



- The model is rich enough to encompass usual IND-CCA attacker, as well as attacks of [APW09].
- The model includes all “failure modes” of SSH-CTR (as implemented in OpenSSH BPP).
  - So cryptanalysis based on error side-channels is covered.
- But:
  - Timing side-channels are not covered.
  - The model does not include anything “outside” the BPP.
  - The proof is specific to the OpenSSH implementation of SSH-CTR.
  - Completeness of model is guaranteed only by manual code inspection.

# Final Remarks on SSH



- Our attacks on SSH illustrate some limitations of current approaches to provable security.
  - How do we know we have the right model?
  - How do we know what features should be included in a protocol description?
    - What is the right amount of abstraction?
    - How close to the implementation level do we need to go to capture all significant attack vectors?
    - Does [PW10] really get it right?

Introduction

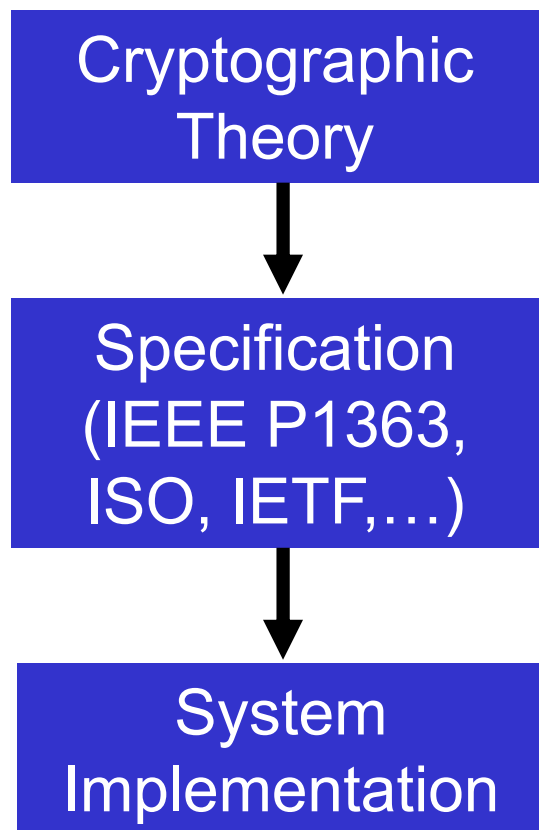
Theory of Symmetric Encryption

SSL/TLS Record Layer Protocol

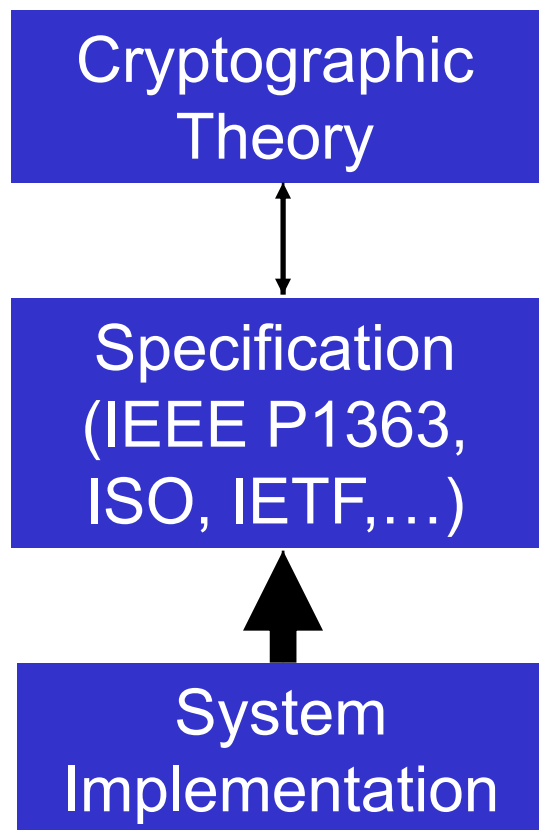
SSH Binary Packet Protocol

Concluding Remarks

# The Big Picture – Idealised View



# The Big Picture – Possible Realities



# Concluding Remarks



- There is a complex interplay between theory, specification and implementation.
  - Despite simplicity of symmetric primitives, current theory does not tell us everything about how to combine them to build secure channels.
  - Proofs are often fragile when we try to apply them in the real world.
    - Attacks outside the model.
    - Flexibility in specifications.
    - Implementation errors.
- Provable security is itself an evolutionary process.
  - But better than break-then-fix loop.
- Many practitioners are not yet convinced by what provable security has to offer.

# Concluding Remarks



- Cryptography is usually only a component in a larger system or protocol.
  - Integrating it can introduce security weaknesses.
- Implementing cryptography in real systems is fraught with dangers.
  - We have focussed almost exclusively on error-based side channels and format-based attacks here.
  - Many other types of implementation-based attack are known.

# Last Words

A (mis)quote from Eugene Spafford:

*“Using encryption on the Internet is the equivalent of arranging an armored car to deliver credit-card information from someone living in a cardboard box to someone living on a park bench.”*

**Thank You**

# Some Further Reading



- [APW09]: Albrecht *et al.*, IEEE S&P 2009.
- [BDJR97]: Bellare *et al.*, FOCS 1997.
- [BKN02]: Bellare *et al.*, ACM-CCS, 2002.
- [BN00]: Bellare and Namprempre, Asiacrypt 2000.
- [D02]: Dai, unpublished, 2002.
- [DP07]: Degabriele and Paterson, IEEE S&P 2007.
- [DP10]: Degabriele and Paterson, ACM-CCS, 2010.
- [CHVV03]: Canvel *et al.*, Crypto 2003.
- [K01]: Krawczyk, Crypto 2001.
- [PW10]: Paterson and Watson, Eurocrypt 2010.
- [V02]: Vaudenay, Eurocrypt 2002.