# SSH: A Case Study of Cryptography in Theory and Practice

Kenny Paterson

Information Security Group

Royal Holloway, University of London

Joint work with Martin Albrecht and Gaven Watson

# Outline

- Introduction to SSH
- Security Proofs for SSH
- Breaking SSH
- A New Security Analysis of SSH
- Concluding Remarks

# Introduction to SSH

# Introduction to 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

# Introduction to SSH

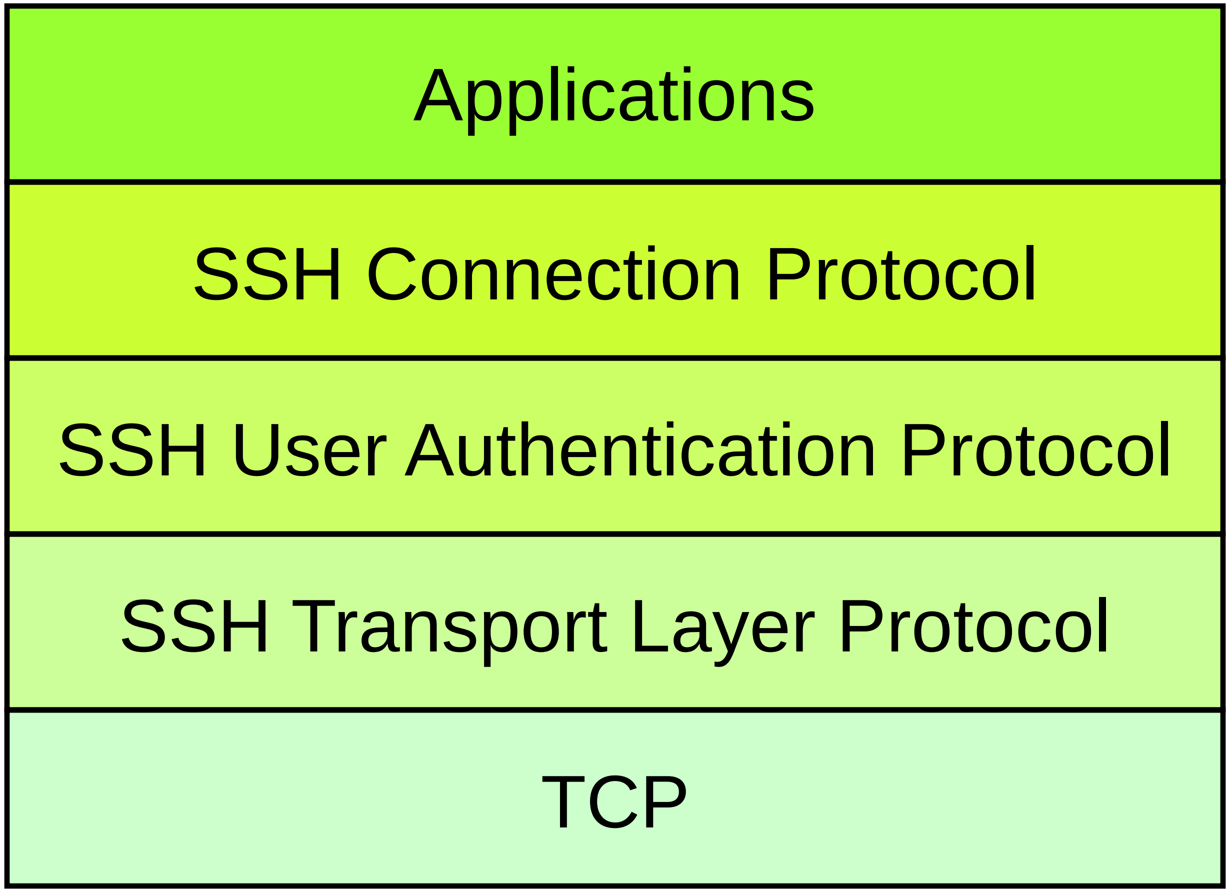
- SSHv1 had several security flaws.
  - Worst ones arising from use of CRC algorithm to provide integrity.
  - Enabling, for example, traffic injection attacks.
- SSHv2 was standardised in 2006 by the IETF in RFCs 4251-4254.
  - But basic specification dates from the late 1990s.
- SSHv2 is widely regarded as providing strong security.
  - Widely used to enable secure remote administration of sensitive systems.
  - One minor flaw in the BPP that *in theory* allows distinguishing attacks ([D02]; [BKN02]).
  - Simple countermeasure adopted in, for example, OpenSSH.
  - Dozens of different implementations of SSH.

# SSHv2 Architecture

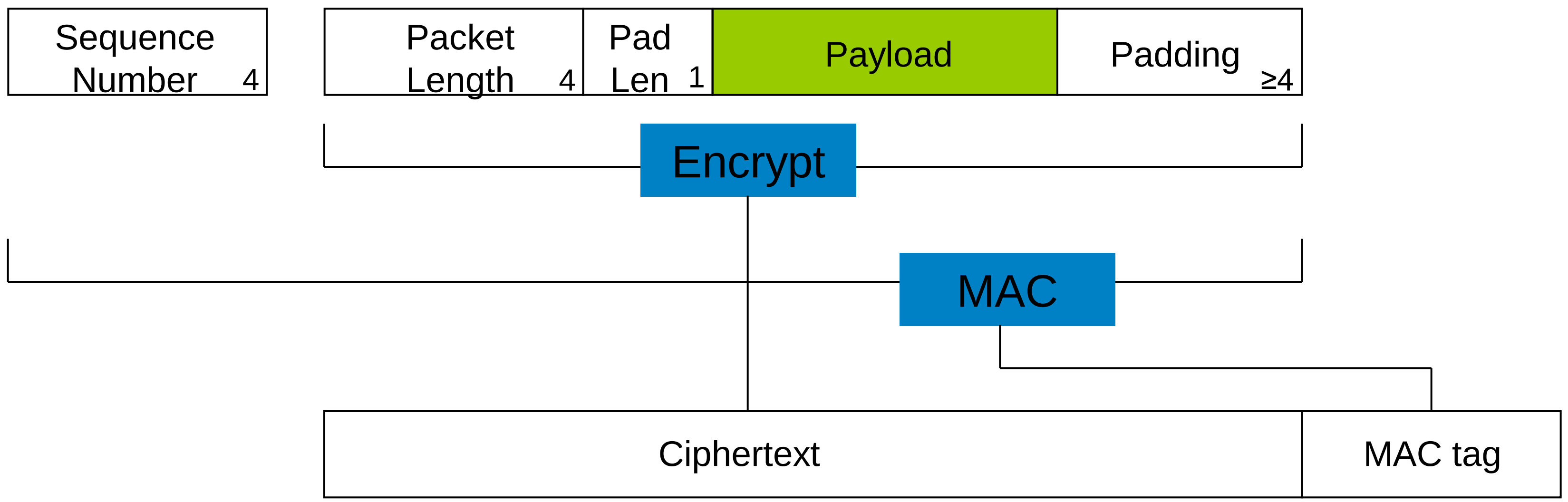
SSHv2 adopts a three layer architecture:

- SSH Transport Layer Protocol.
  - Initial connection establishment and key exchange.
  - Server authentication (almost always).
  - Sets up a secure channel between client and server, using the **SSH Binary Packet Protocol** specified in RFC 4253.
- SSH User Authentication Protocol.
  - Client authentication over secure Transport Layer channel.
- SSH Connection Protocol.
  - Supports multiple concurrent connections over a single Transport Layer secure channel.
  - Efficiency (session re-use) and support for multiple applications.

# SSHv2 Architecture

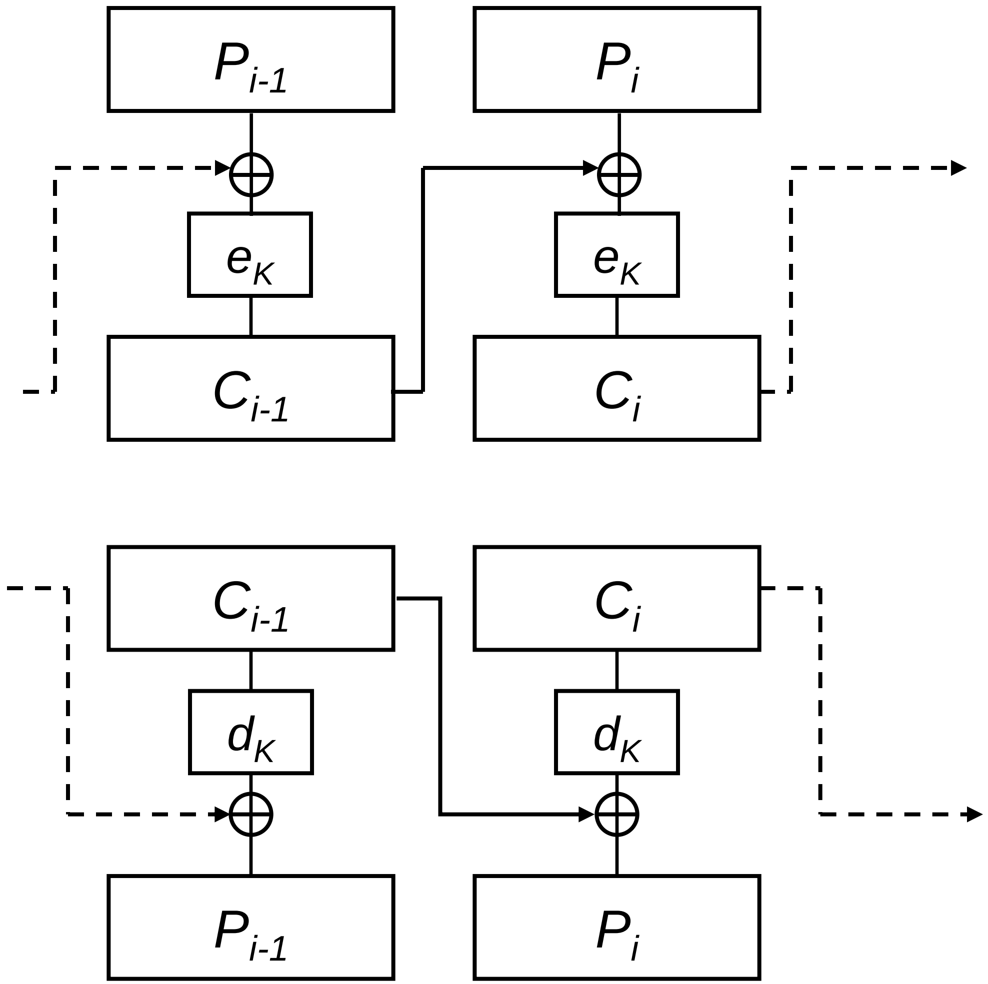


# The SSH BPP



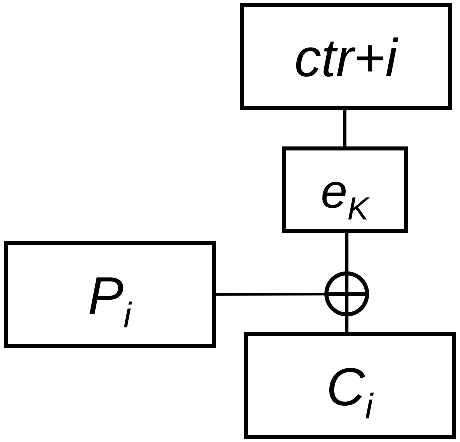

- Encode-then-Encrypt&MAC construction, **not** generically secure.
  - Because secure MAC can leak plaintext information.
- Packet length field measures the size of the packet on the wire in bytes and is encrypted to hide the true length of SSH packets.
- Variable length padding is permissible; padding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

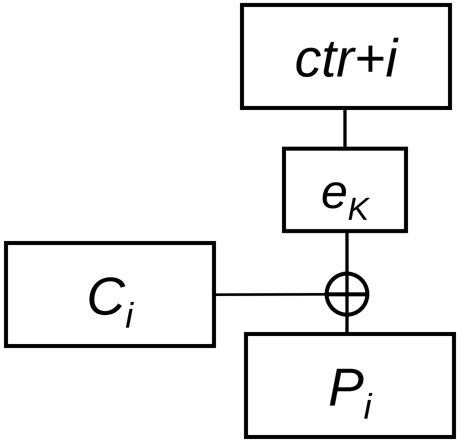
# CBC Mode in SSH




- 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.

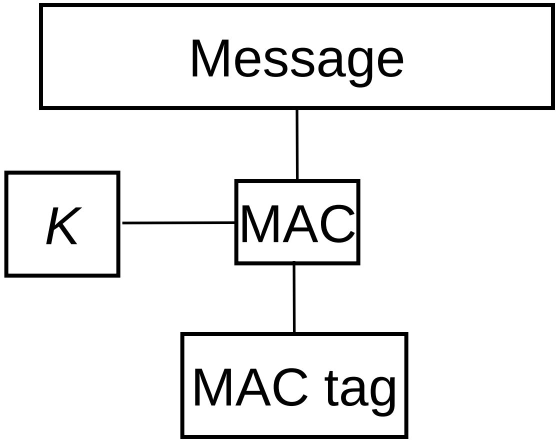
# CTR Mode in SSH




- 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.

# MACs in SSH




- A MAC algorithm has two inputs:
  - A message.
  - A symmetric key *K.*
- Output is a (short) MAC tag.
- Key requirement is unforgeability:
  - Having seen MAC tags for many chosen messages, an adversary cannot create the correct MAC tag for another chosen message.
- SSH requires support for HMAC-SHA1 and recommends support for HMAC-SHA1-96.

Introduction to SSH

Security Proofs for SSH

# Security of the SSH BPP

- Attack of [D02], [BKN02] exploits chained IVs in CBC mode.
  - Breaks semantic security of the SSH BPP in a chosen ciphertext attack model.
    - 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

- [BKN02] developed a stateful security model for general encode-then-encrypt&MAC schemes.
  - IND-SFCCA security, where SF=Stateful.
  - Attacker has access to an LoR encryption oracle and a decryption oracle.
  - Both oracles are stateful, with states parameterised by SSH sequence numbers.
  - Model allows adversary to advance states to any chosen value via queries to encryption and decryption oracles.
    - Adversary can submit output of encryption oracle at SN to decryption oracle at SN, but receives no output from decryption oracle.
  - Adversary wins game if he can guess hidden bit *b* of encryption oracle.

# Security of the SSH BPP

- Using this 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 the 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.

Introduction to SSH

Security Proofs for SSH

Breaking SSH

# Attacking the SSH BPP

- [APW09]: plaintext recovering attacks against SSH BPP.
  - Much stronger than distinguishing attack of [D02], [BKN02]!
- 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.

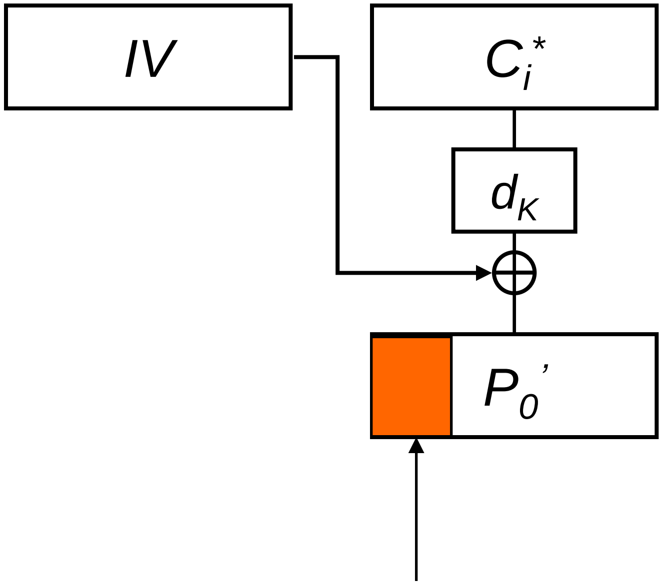
# Attacking the SSH BPP (Theory)

- 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 as seen as the *first* block of a new SSH packet by the receiver…

# Attacking the SSH BPP (Theory)

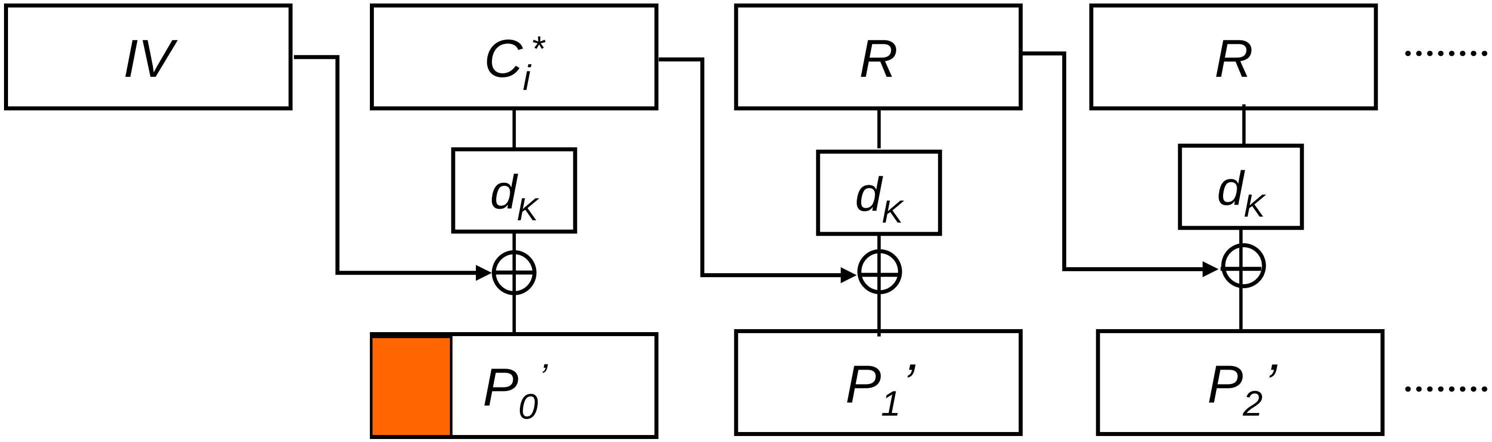



The receiver will treat the first 32 bits of the calculated plaintext block as the packet length field for the new packet. Here:

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

where *IV* is known from the previous packet.

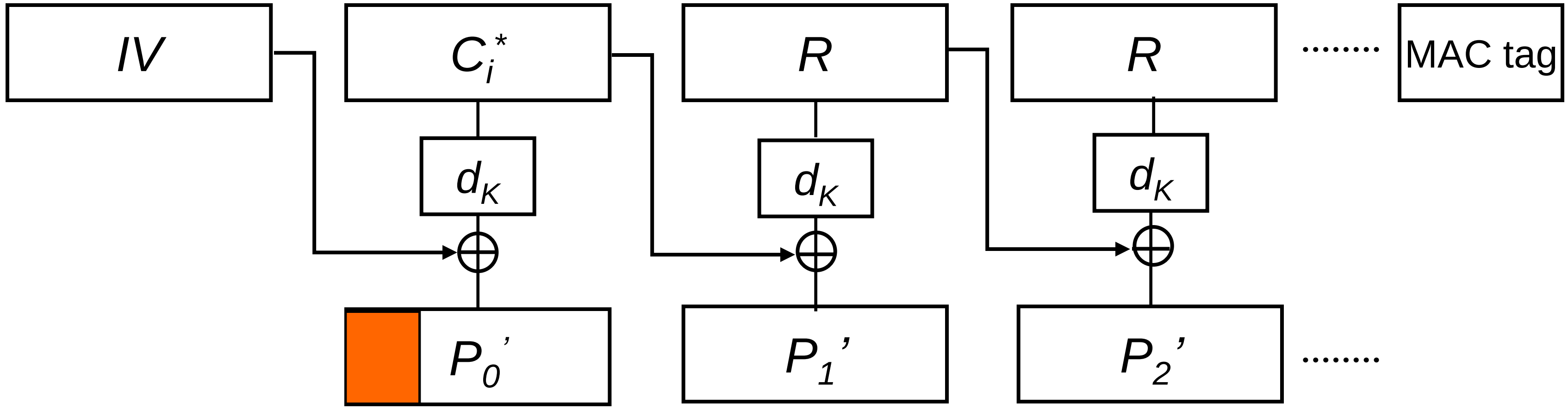
# Attacking the SSH BPP (Theory)




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)



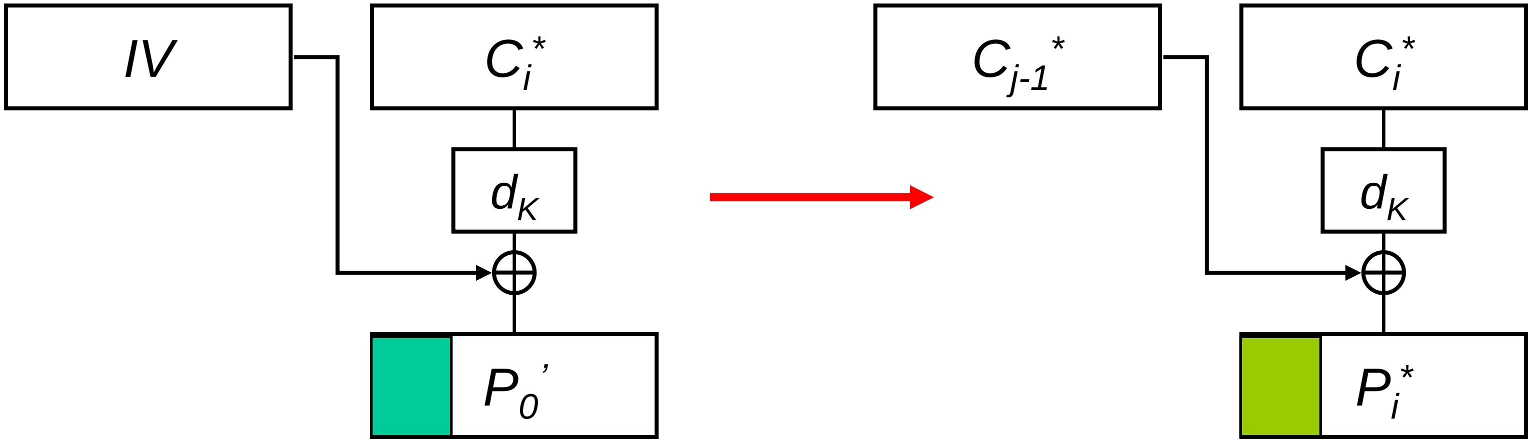

- Eventually, once enough data has arrived, the receiver will receive what it thinks is the MAC tag.
- The receiver will then check the MAC.
  - This check will fail with overwhelming probability.
  - Consequently the connection is terminated (with an error message).
- How much data is “enough” so that the receiver decides to check the MAC?

# Attacking the SSH BPP (Theory)

- The receiver **has** to use the packet length field to decide when the MAC tag has arrived.
- Hence an attacker who counts the number of blocks needed to cause connection termination learns 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
- We worked with OpenSSH 5.1.
  - Version 5.2 released 23/02/2009 partly as a consequence of our work, current version is 5.3.

# Attacking OpenSSH

- In OpenSSH 5.1, two sanity checks are carried out on the packet length field after the first block is decrypted.
- When each of the checks fails, the SSH connection is terminated in subtly different ways.
  - This difference leaks some information, but also reduces success prob. 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:
  - A first attack verifiably recovering 14 bits of plaintext with probability $2^{-14}$.
  - A second attack verifiably recovering 32 bits of plaintext with probability $2^{-18}$ (for a 128-bit block cipher).
  - The attacks require injection of (roughly) $2^{18}$ bytes.
- Both attacks result in termination of the SSH connection.
  - But the attacks can be iterated if a plaintext is repeated across multiple connections.
- The attacks worked in practice.

# Iterating the attacks

- If a *fixed* plaintext is repeated at a *fixed* position in SSH packets over *multiple* connections, then the attacks can be iterated to boost success rate.
  - Application to password extraction.
  - Some clients automatically reconnect on session termination.
  - By carefully selecting after which IV to inject the target ciphertext block, we can reduce the number of connections consumed during the attack to $2^{14} + 2^4$.

# Disclosure

- 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 counter mode encryption.

# Reactions and Countermeasures

- OpenSSH published a statement and committed a first fix (21/11/2008).
  - www.openssh.com/txt/cbc.adv
  - Both the statement and the bugfix addressed only the $2^{-14}$ attack.
- Then OpenSSH released OpenSSH 5.2 (23/02/2009).
  - Offers AES in counter mode and arcfour256 stream cipher ahead of CBC mode block ciphers.

# Reactions and Countermeasures

- 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."*
  - 20-30 lines of new code with no comments.
  - If length checks fail, then set length field to $2^{18}$ and carry on.
  - This renders OpenSSH *more* vulnerable to DoS attacks!
  - And there's still a distinguishing attack.

# Further Vendor Reactions

- **SunSSH** increased the version number because of a security vulnerability "for the first time".
  - However, it seems they only addressed the $2^{-14}$ attack.
- **SSH.com** acknowledged that their products are vulnerable and claim to have addressed the issue.
- **Bitvise** acknowledged that their WinSSHD product is vulnerable and issued an update.
  - Randomisation of length field after failure of sanity checking.
- **Dropbear** added support for counter mode.
- Partial list of affected vendors and products at:

  http://www.kb.cert.org/vuls/id/958563

# Some Countermeasures

- Use counter mode.
  - *Stateful* version of counter mode needed, as standardised in RFC 4344.
  - Our attacks no longer apply.
- *Enforce* use of counter mode.
  - Not standards compliant with the RFCs as they are currently written.
  - Some implementations do not support counter mode at all, creating backwards compatibility issue.
  - "*Only a cryptographer would suggest this...*"

# Further Countermeasures

- Randomise the length field if the length checks fail.
  - The Bitvise solution.
- 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.

# Impact of the Attacks

- SSH was meant to be bullet-proof, but our attacks are really quite simple.
- The specific attacks are easily circumvented by switching to CTR mode or by modifying error handling in CBC mode.
- Unfortunately, this does not constitute a proof of security against attacks of the type presented here.
- And the basic attack applied to the proven secure variant SSH-$NPC
  – Hinting at inadequacies of the approach used in [BKN02].

Introduction to SSH

Security Proofs for SSH

Breaking SSH

A New Security Analysis of SSH

# 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.
  - The $2^{-14}$ attack 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]
    - LOR-BSF-CCA security;
  - 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 our new model.

# A New Security Analysis of SSH

- Our model extends the model from [BKN02]:
  - Attacker has access to stateful LoR encryption oracle and stateful decryption oracle.
  - *Byte-by-byte delivery of ciphertexts to decryption oracle, and buffering of any as-yet-unprocessed ciphertext bytes.*
  - Adversary can advance oracles to arbitrary states by submitting output of encryption oracle to decryption oracle (“in-sync queries”).
  - Adversary wins game if he can guess the hidden bit *b* of the encryption oracle.

# A New Security Analysis of SSH

- Our model does **not** include:
  - Byte-by-byte delivery of plaintexts to encryption oracle.
    - Because (Open)SSH is not implemented this way, though the model and proofs can be adapted to handle it.
  - Confidentiality of the packet length field.
    - Because it is easy to show that it is impossible to provide this in practice in a reasonable attack model.
- Still, the model is powerful enough to capture the attacks of [APW09].

# 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$_{Ke,Kt}$ (*m*)**

if $st_e$ = *fail* then
  return *fail*
$(m_e, m_t)$ = encode($m$)
if $m_e$ = *fail* then
  $st_e$ = *fail*
  return *fail*
else
  $c = E\text{-CTR}_{Ke}(m_e)$ \\ counter mode encryption
  $tau = T_{Kt}(m_t)$ \\ MAC computation
  return $c \,||\, tau$
end if

# Modelling the Decryption Algorithm

**Algorithm D-SSH-CTR$_{Ke,Kt}$ (*c*)**
if $st_d$ = *fail* then
  return *fail*
end if
***{Stage 1}***
*cbuff* = *cbuff* || *c*
***{Stage 2}***
if $m_e$ = *empty* and |*cbuff*| >= *L* then
  Parse *cbuff* as *c'*||*A* (where |*c'*| = *L*)
  $m_e[1]$ = *D*-CTR$_{Ke}$ (*c'*)
  *LF* = len($m_e[1]$)   \\ len checking
  if *LF* = $fail_L$ then
    $st_d$ = *fail*
    return $fail_L$
  else
    *need* = *4* + *LF* + *maclen*
  end if
end if

***{Stage 3}***
if |*cbuff*| >= *L* then
  if |*cbuff*| >= *need* then
    Parse *cbuff* as *c*[1…*n*] || *tau* || *B*,
    where |*c*[1…*n*] || *tau*| = *need*,
    and |*tau*| = *maclen*
    $m_e[2…n]$ = *D*-CTR$_{Ke}$ (*c*[2…*n*])  \\ CTR mode
    $m_e$ = $m_e[1]$ **||** $m_e[2…n]$
    $m_t$ = *SN* || $m_e$
    *v* = $V_{Kt}$ ($m_t$, *tau*)   \\ MAC checking
    if *v* = 0 then   \\ MAC failure
      $st_d$ = *fail*
      return $fail_A$
    else
      *m* = decode($m_e$)  \\ decoding plaintext
      $m_e$ = *empty*; *cbuff* = *B*
      return *m*
    end if
  end if
end if

# Main Security Result

**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 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!

# Outline of Proof

- Broad outline is similar to proofs in [BKN02], but with complications because of the need to handle decryption queries and length checking.
- Reduce in first step to "security against LOR-LL-CPA adversary" + "security against ciphertext forgery".
  - LL = length leakage – usual CPA adversary but given an extra length oracle.
- Security against LOR-LL-CPA adversary then reduces to pseudo-randomness of F and of MAC tags.
- Security against ciphertext forgery reduces to strong unforgeability of MAC scheme.

# What Does the Proof Mean?

- The model is rich enough to encompass usual LOR-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.

Introduction to SSH

Security Proofs for SSH

Breaking SSH

A New Security Analysis of SSH

Concluding Remarks

# Concluding Remarks

- We have given an overview of recent attacks against the SSH BPP and how they illustrate limitations of security analysis of [BKN02].
- We have motivated the introduction of a new security model for SSH.
- We have sketched how to prove SSH-CTR secure in this new model.
  – With an accurate description of SSH-CTR based on RFCs and OpenSSH source code.

# The Theory/Practice Disconnect

- Theory has long suggested that encrypt-and-MAC constructions are a **bad** idea in general.
- Yet SSH uses it and we are probably stuck with the current SSH BPP design for many years to come.
- What *useful, accessible* theory was available for the SSH BPP designers to draw upon in the late 1990s?
- How useful is today's theory we can find such a simple attack just outside the model?
- Incorporating all the security-relevant details of the implementation was very hard work using our current analysis tools.
- We need better theory, better tools, and a better understanding of how to apply the theory to practice.

# Further Reading


[APW09]: Albrecht, Paterson and Watson, Plaintext-recovery attacks against SSH, IEEE S&P 2009.

[BKN02]: Bellare, Kohno and Namprempre, Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the encode-then-encrypt-and-MAC paradigm, ACM-CCS, 2002.

[D02]: Dai, An Attack Against SSH2 Protocol, unpublished, 2002.

[PW10]: Paterson and Watson, Plaintext-Dependent Decryption: A Formal Security Treatment of SSH-CTR, Eurocrypt 2010.