Plaintext-Dependent Decryption: A Formal Security Treatment of SSH-CTR

Kenneth G. Paterson and Gaven J. Watson

Information Security Group,
Royal Holloway, University of London

1st June 2010

Outline

SSH

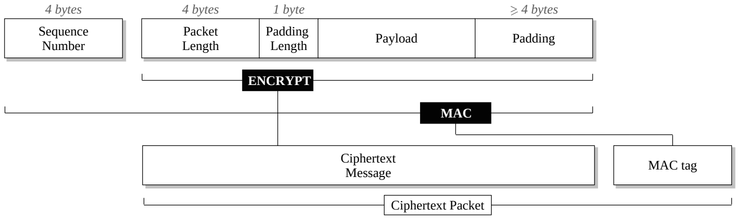
Wikipedia's Description:

"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."

SSH RFCs

- SSHv2 was standardised in 2006 by the IETF in RFCs 4251-4254.
- RFC 4253 specifies the SSH Binary Packet Protocol (BPP).
  - Symmetric encryption and integrity protection for SSH packets, using keys agreed in an earlier exchange.
- SSHv2 is widely regarded as being secure.
  - One minor flaw in the BPP that allows distinguishing attacks (Dai; Bellare, Kohno and Namprempre).
- Several minor variants of the SSH BPP were proven secure by Bellare, Kohno and Namprempre (BKN) (ACM CCS 2002)

SSH Binary Packet Protocol



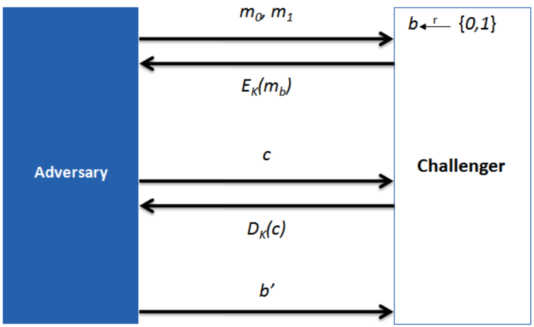
- Encode-then-Encrypt&MAC construction.
- Packet length field measures total size of packet on the wire in bytes and is encrypted to hide true length of SSH packets.
- RFC 4253 mandates 3DES-CBC and recommends AES-CBC.

Outline

The First Formal Security Treatment

- A formal security analysis of the SSH BPP was first performed by Bellare, Kohno and Namprempre (BKN).
- They proposed variants of the SSH BPP and proved them to be secure.
- They prove security in a slightly extended version of the normal IND-CCA (or LOR-CCA) model.

LOR-CCA



The adversary wins if $b' = b$.
We define the adversary's advantage to be:

$$\mathbf{Adv}^{\text{lor-cca}} = \left| \Pr[b' = b] - \frac{1}{2} \right|$$

IND-SFCCA

- BKN's extended model is denoted IND-SFCCA meaning indistinguishability under chosen-ciphertext attack for stateful decryption.
- Again the adversary has access to a left-or-right encryption oracle and a decryption oracle.
- But the oracles are stateful.
- Why?
  - Recall that the SSH BPP uses a sequence number when calculating the MAC.
  - This sequence number does not form part of the plaintext or ciphertext message; it is only used in the MAC calculation.

IND-SFCCA

- BKN's extended model is denoted IND-SFCCA meaning indistinguishability under chosen-ciphertext attack for stateful decryption.
- Again the adversary has access to a left-or-right encryption oracle and a decryption oracle.
- But the oracles are stateful.
- Why?
  - Recall that the SSH BPP uses a sequence number when calculating the MAC.
  - This sequence number does not form part of the plaintext or ciphertext message; it is only used in the MAC calculation.

IND-SFCCA

- BKN's extended model is denoted IND-SFCCA meaning indistinguishability under chosen-ciphertext attack for stateful decryption.
- Again the adversary has access to a left-or-right encryption oracle and a decryption oracle.
- But the oracles are stateful.
- Why?
  - Recall that the SSH BPP uses a sequence number when calculating the MAC.
  - This sequence number does not form part of the plaintext or ciphertext message; it is only used in the MAC calculation.

IND-SFCCA

- BKN's extended model is denoted IND-SFCCA meaning indistinguishability under chosen-ciphertext attack for stateful decryption.
- Again the adversary has access to a left-or-right encryption oracle and a decryption oracle.
- But the oracles are stateful.
- Why?
  - Recall that the SSH BPP uses a sequence number when calculating the MAC.
  - This sequence number does not form part of the plaintext or ciphertext message; it is only used in the MAC calculation.

IND-SFCCA

- BKN's extended model is denoted IND-SFCCA meaning indistinguishability under chosen-ciphertext attack for stateful decryption.
- Again the adversary has access to a left-or-right encryption oracle and a decryption oracle.
- But the oracles are stateful.
- Why?
    - Recall that the SSH BPP uses a sequence number when calculating the MAC.
    - This sequence number does not form part of the plaintext or ciphertext message; it is only used in the MAC calculation.

Provably Secure SSH Alternatives

BKN propose various SSH alternatives and prove them secure in their IND-SFCCA model.

- SSH-$NPC, CBC mode with randomised per-packet IV and random padding.
- SSH-CTR, Counter mode encryption.
- SSH-CTRIV-CBC, CBC mode with counter IVs.
- SSH-EIV-CBC, CBC mode with encrypted IVs.

Outline

But....

- Albrecht, Paterson and W. (APW) (IEEE S&P 2009) presented plaintext-recovery attacks against the SSH-BPP.
- The attacks exploit features of the BPP not covered by BKN's analysis.
- The attacks are even applicable to one of the provably secure variants of SSH proposed by BKN, namely SSH-$NPC.
- The attacks demonstrate that the existing IND-SFCCA model fails to capture all security critical features of the SSH RFCs and SSH implementations.

But....

- Albrecht, Paterson and W. (APW) (IEEE S&P 2009) presented plaintext-recovery attacks against the SSH-BPP.
- The attacks exploit features of the BPP not covered by BKN's analysis.
- The attacks are even applicable to one of the provably secure variants of SSH proposed by BKN, namely SSH-$NPC.
- The attacks demonstrate that the existing IND-SFCCA model fails to capture all security critical features of the SSH RFCs and SSH implementations.

But....

- Albrecht, Paterson and W. (APW) (IEEE S&P 2009) presented plaintext-recovery attacks against the SSH-BPP.
- The attacks exploit features of the BPP not covered by BKN's analysis.
- The attacks are even applicable to one of the provably secure variants of SSH proposed by BKN, namely SSH-$NPC.
- The attacks demonstrate that the existing IND-SFCCA model fails to capture all security critical features of the SSH RFCs and SSH implementations.

But....

- Albrecht, Paterson and W. (APW) (IEEE S&P 2009) presented plaintext-recovery attacks against the SSH-BPP.
- The attacks exploit features of the BPP not covered by BKN's analysis.
- The attacks are even applicable to one of the provably secure variants of SSH proposed by BKN, namely SSH-$NPC.
- The attacks demonstrate that the existing IND-SFCCA model fails to capture all security critical features of the SSH RFCs and SSH implementations.

What Makes the Attacks Possible?

APW's attacks exploit an interaction of the following design features of SSH:

- The packet length field encodes how much data needs to be received before the MAC can be checked.
- The attacker can send data on an SSH connection in small chunks (TCP) and observe how the receiver reacts.
- CBC mode is mandated.
- A MAC failure is visible on the network.

The attacks were implemented against OpenSSH, one of the most popular implementations of the SSH RFCs, and apply up to and including OpenSSH version 5.1.

What Makes the Attacks Possible?

APW's attacks exploit an interaction of the following design features of SSH:

- The packet length field encodes how much data needs to be received before the MAC can be checked.
- The attacker can send data on an SSH connection in small chunks (TCP) and observe how the receiver reacts.
- CBC mode is mandated.
- A MAC failure is visible on the network.

The attacks were implemented against OpenSSH, one of the most popular implementations of the SSH RFCs, and apply up to and including OpenSSH version 5.1.

What Makes the Attacks Possible?

APW's attacks exploit an interaction of the following design features of SSH:

- The packet length field encodes how much data needs to be received before the MAC can be checked.
- The attacker can send data on an SSH connection in small chunks (TCP) and observe how the receiver reacts.
- CBC mode is mandated.
- A MAC failure is visible on the network.

The attacks were implemented against OpenSSH, one of the most popular implementations of the SSH RFCs, and apply up to and including OpenSSH version 5.1.

What Makes the Attacks Possible?

APW's attacks exploit an interaction of the following design features of SSH:

- The packet length field encodes how much data needs to be received before the MAC can be checked.
- The attacker can send data on an SSH connection in small chunks (TCP) and observe how the receiver reacts.
- CBC mode is mandated.
- A MAC failure is visible on the network.

The attacks were implemented against OpenSSH, one of the most popular implementations of the SSH RFCs, and apply up to and including OpenSSH version 5.1.

What Makes the Attacks Possible?

APW's attacks exploit an interaction of the following design features of SSH:

- The packet length field encodes how much data needs to be received before the MAC can be checked.
- The attacker can send data on an SSH connection in small chunks (TCP) and observe how the receiver reacts.
- CBC mode is mandated.
- A MAC failure is visible on the network.

The attacks were implemented against OpenSSH, one of the most popular implementations of the SSH RFCs, and apply up to and including OpenSSH version 5.1.

What Makes the Attacks Possible?

APW's attacks exploit an interaction of the following design features of SSH:

- The packet length field encodes how much data needs to be received before the MAC can be checked.
- The attacker can send data on an SSH connection in small chunks (TCP) and observe how the receiver reacts.
- CBC mode is mandated.
- A MAC failure is visible on the network.

The attacks were implemented against OpenSSH, one of the most popular implementations of the SSH RFCs, and apply up to and including OpenSSH version 5.1.

What Does This Mean in Practice?

- UK Centre for Protection of National Infrastructure (CPNI) released a vulnerability advisory.
- This advisory recommends a switch to counter mode encryption.
  - Counter mode is resistant to APW's attacks.
  - RFC 4344 already existed to standarise the use of counter mode encryption in SSH.
- As a result implementations of SSH (including OpenSSH) have been updated to use counter mode in preference to CBC mode.

What Does This Mean in Practice?

- UK Centre for Protection of National Infrastructure (CPNI) released a vulnerability advisory.
- This advisory recommends a switch to counter mode encryption.
  - Counter mode is resistant to APW's attacks.
  - RFC 4344 already existed to standarise the use of counter mode encryption in SSH.
- As a result implementations of SSH (including OpenSSH) have been updated to use counter mode in preference to CBC mode.

What Does This Mean in Practice?

- UK Centre for Protection of National Infrastructure (CPNI) released a vulnerability advisory.
- This advisory recommends a switch to counter mode encryption.
  - Counter mode is resistant to APW's attacks.
  - RFC 4344 already existed to standarise the use of counter mode encryption in SSH.
- As a result implementations of SSH (including OpenSSH) have been updated to use counter mode in preference to CBC mode.

...and wrt the Security Analysis?

- There was a formal security analysis.
- We have just seen that due to some aspects not covered by this existing analysis, attacks were still possible.
- So despite SSH-CTR appearing to be resistant to APW style attacks the existing security analysis does not provide any security guarantees against these attacks.
- Can we add these missing aspects to the analysis and provide a new proof of security for SSH-CTR?

...and wrt the Security Analysis?

- There was a formal security analysis.
- We have just seen that due to some aspects not covered by this existing analysis, attacks were still possible.
- So despite SSH-CTR appearing to be resistant to APW style attacks the existing security analysis does not provide any security guarantees against these attacks.
- Can we add these missing aspects to the analysis and provide a new proof of security for SSH-CTR?

...and wrt the Security Analysis?

- There was a formal security analysis.
- We have just seen that due to some aspects not covered by this existing analysis, attacks were still possible.
- So despite SSH-CTR appearing to be resistant to APW style attacks the existing security analysis does not provide any security guarantees against these attacks.
- Can we add these missing aspects to the analysis and provide a new proof of security for SSH-CTR?

...and wrt the Security Analysis?

- There was a formal security analysis.
- We have just seen that due to some aspects not covered by this existing analysis, attacks were still possible.
- So despite SSH-CTR appearing to be resistant to APW style attacks the existing security analysis does not provide any security guarantees against these attacks.
- Can we add these missing aspects to the analysis and provide a new proof of security for SSH-CTR?

Outline

This Paper

- Aim:
  - New analysis which more closely captures the capabilities of real world attackers.
  - Our new proofs of security must imply security against a much wider array of attacks including APW's plaintext-recovery attacks.

What Went Wrong I

- The security model does model errors during the BPP decryption process but only a single type of error message is output.
  - One of APW's attacks against OpenSSH exploits the fact that the errors are distinguishable.
  - But even allowing only one error type, there is a very simple distinguishing attack having probability 1 against SSH-$NPC!
- One good point is that connection teardowns are modelled by disallowing access to decryption after an error event.

What Went Wrong I

- The security model does model errors during the BPP decryption process but only a single type of error message is output.
  - One of APW's attacks against OpenSSH exploits the fact that the errors are distinguishable.
  - But even allowing only one error type, there is a very simple distinguishing attack having probability 1 against SSH-$NPC!
- One good point is that connection teardowns are modelled by disallowing access to decryption after an error event.

What Went Wrong II

Model:

- The security model of BKN assumes that ciphertexts are "self-describing" in terms of their lengths.
- The model does not allow for the possibility that the amount of data needed to complete the decryption process might be governed by data produced during the decryption process.
- Ciphertexts and plaintexts are handled in the model as "atomic" strings.

Reality:

- In practice, APW's attacks exploit the fact that SSH is run over TCP and hence data can be sent in small chunks.
- Crucially, the SSH packet length field tells the decryptor how much data it should expect to receive.

What Went Wrong II

Model:

- The security model of BKN assumes that ciphertexts are "self-describing" in terms of their lengths.
- The model does not allow for the possibility that the amount of data needed to complete the decryption process might be governed by data produced during the decryption process.
- Ciphertexts and plaintexts are handled in the model as "atomic" strings.

Reality:

- In practice, APW's attacks exploit the fact that SSH is run over TCP and hence data can be sent in small chunks.
- Crucially, the SSH packet length field tells the decryptor how much data it should expect to receive.

What Went Wrong II

Model:

- The security model of BKN assumes that ciphertexts are "self-describing" in terms of their lengths.
- The model does not allow for the possibility that the amount of data needed to complete the decryption process might be governed by data produced during the decryption process.
- Ciphertexts and plaintexts are handled in the model as "atomic" strings.

Reality:

- In practice, APW's attacks exploit the fact that SSH is run over TCP and hence data can be sent in small chunks.
- Crucially, the SSH packet length field tells the decryptor how much data it should expect to receive.

What Went Wrong II

Model:

- The security model of BKN assumes that ciphertexts are "self-describing" in terms of their lengths.
- The model does not allow for the possibility that the amount of data needed to complete the decryption process might be governed by data produced during the decryption process.
- Ciphertexts and plaintexts are handled in the model as "atomic" strings.

Reality:

- In practice, APW's attacks exploit the fact that SSH is run over TCP and hence data can be sent in small chunks.
- Crucially, the SSH packet length field tells the decryptor how much data it should expect to receive.

What Went Wrong II

Model:

- The security model of BKN assumes that ciphertexts are "self-describing" in terms of their lengths.
- The model does not allow for the possibility that the amount of data needed to complete the decryption process might be governed by data produced during the decryption process.
- Ciphertexts and plaintexts are handled in the model as "atomic" strings.

Reality:

- In practice, APW's attacks exploit the fact that SSH is run over TCP and hence data can be sent in small chunks.
- Crucially, the SSH packet length field tells the decryptor how much data it should expect to receive.

Improving the Security Analysis

So what can we do to improve the existing analysis?

- Accurate description of the BPP
  - Distinguishable Error Outputs:
    - Length check errors $\perp_L$
    - MAC verification errors $\perp_A$
    - Parsing or padding check errors $\perp_P$
  - Make decryption plaintext-dependent.
    - Decryption is governed by plaintext received during the decryption process.
    - In SSH, the packet length field tells the recipient how much data it must wait for.
- Improving the model
  - Allow buffered decryption.

Improving the Security Analysis

So what can we do to improve the existing analysis?

- Accurate description of the BPP
  - Distinguishable Error Outputs:
    - Length check errors $\perp_L$
    - MAC verification errors $\perp_A$
    - Parsing or padding check errors $\perp_P$
  - Make decryption plaintext-dependent.
    - Decryption is governed by plaintext received during the decryption process.
    - In SSH, the packet length field tells the recipient how much data it must wait for.
- Improving the model
  - Allow buffered decryption.

Improving the Security Analysis

So what can we do to improve the existing analysis?

- Accurate description of the BPP
  - Distinguishable Error Outputs:
    - Length check errors $\perp_L$
    - MAC verification errors $\perp_A$
    - Parsing or padding check errors $\perp_P$
  - Make decryption plaintext-dependent.
    - Decryption is governed by plaintext received during the decryption process.
    - In SSH, the packet length field tells the recipient how much data it must wait for.
- Improving the model
  - Allow buffered decryption.

Improving the Security Analysis

Royal Holloway University of London Information Security Group

So what can we do to improve the existing analysis?

- Accurate description of the BPP
  - Distinguishable Error Outputs:
    - Length check errors $\perp_L$
    - MAC verification errors $\perp_A$
    - Parsing or padding check errors $\perp_P$
  - Make decryption plaintext-dependent.
    - Decryption is governed by plaintext received during the decryption process.
    - In SSH, the packet length field tells the recipient how much data it must wait for.
- Improving the model
  - Allow buffered decryption.

Improving the Security Analysis

So what can we do to improve the existing analysis?

- Accurate description of the BPP
  - Distinguishable Error Outputs:
    - Length check errors $\perp_L$
    - MAC verification errors $\perp_A$
    - Parsing or padding check errors $\perp_P$
  - Make decryption plaintext-dependent.
    - Decryption is governed by plaintext received during the decryption process.
    - In SSH, the packet length field tells the recipient how much data it must wait for.
- Improving the model
  - Allow buffered decryption.

Improving the Security Analysis

So what can we do to improve the existing analysis?

- Accurate description of the BPP
  - Distinguishable Error Outputs:
    - Length check errors $\perp_L$
    - MAC verification errors $\perp_A$
    - Parsing or padding check errors $\perp_P$
  - Make decryption plaintext-dependent.
    - Decryption is governed by plaintext received during the decryption process.
    - In SSH, the packet length field tells the recipient how much data it must wait for.
- Improving the model
  - Allow buffered decryption.

SSH-CTR

- We now give our description of SSH-CTR, with plaintext-dependent decryption.
- The scheme combines:
  - the encoding scheme for the SSH-BPP, $\mathcal{EC} = (\text{enc}, \text{dec})$,
  - a length checking algorithm, len,
  - counter mode encryption $\text{CTR}[F]$,
  - and a message authentication scheme $\mathcal{MA}$.

SSH-CTR

- We now give our description of SSH-CTR, with plaintext-dependent decryption.
- The scheme combines:
  - the encoding scheme for the SSH-BPP, $\mathcal{EC} = (\mathsf{enc}, \mathsf{dec})$,
  - a length checking algorithm, len,
  - counter mode encryption CTR[F],
  - and a message authentication scheme $\mathcal{MA}$.

SSH-CTR – Encoding Scheme

```
Algorithm enc($m$)
  if $st_e = \perp$ then
    return $(\perp, \perp)$
  end if
  if $SN_e \geq 2^{32}$ or $|m| \geq 2^{32} - 5$ then
    $st_e \leftarrow \perp$
    return $(\perp, \perp)$
  else
    $PL \leftarrow L - ((|m| + 5) \bmod L)$
    if $PL < 4$ then
      $PL \leftarrow PL + L$
    end if
    $PD \xleftarrow{r} \{0,1\}^{8 \cdot PL}$
    $LF \leftarrow (1 + |m| + PL)$
    $m_e \leftarrow \langle LF \rangle_4 \| \langle PL \rangle_1 \| m \| PD$
    $m_t \leftarrow SN_e \| m_e$
    $SN_e \leftarrow SN_e + 1$
    return $(m_e, m_t)$
  end if
```

```
Algorithm dec($m_e$)
  if $st_d = \perp$ then
    return $\perp$
  end if
  if $SN_d \geq 2^{32}$ then
    $st_d \leftarrow \perp$
    return $\perp$
  else
    Attempt to parse $m_e$ as:
    $\langle LF \rangle_4 \| \langle PL \rangle_1 \| m \| PD$ where
    $PL \geq 4$, $|PD| = PL$ and $|m| \geq 0$.
    if parsing fails then
      $st_d \leftarrow \perp$
      return $\perp_P$
    else
      $SN_d \leftarrow SN_d + 1$
      return $m$
    end if
  end if
```

SSH-CTR – Length Checks

```
**Algorithm** len($m$) ($|m| = L$)
    Parse $m$ as $\langle LF \rangle_4 \| R$
    **if** $LF \leq 5$ or $LF \geq 2^{18}$ **then**
        **return** $\perp_L$
    **else if** $LF + 4 \bmod L \neq 0$ **then**
        **return** $\perp_L$
    **else**
        **return** $LF$
    **end if**
```

- These length checks are specific to OpenSSH.

SSH-CTR – Main Scheme

```
Algorithm $\mathcal{K}$-SSH-CTR$(k)$
    $K_e \xleftarrow{r} \mathcal{K}_e(k)$
    $K_t \xleftarrow{r} \mathcal{K}_t(k)$
    $ctr \xleftarrow{r} \{0,1\}^l$
    return $K_e, K_t, ctr$
```

```
Algorithm $\mathcal{E}$-SSH-CTR$_{K_e,K_t}(m)$
    if $st_e = \perp$ then
        return $\perp$
    end if
    $(m_e, m_t) \leftarrow \text{enc}(m)$
    if $m_e = \perp$ then
        $st_e \leftarrow \perp$
        return $\perp$
    else
        $c \leftarrow \mathcal{E}\text{-CTR}_{K_e}(m_e)$
        $\tau \leftarrow \mathcal{T}_{K_t}(m_t)$
        return $c \| \tau$
    end if
```

SSH-CTR – Main Scheme

```
Algorithm $\mathcal{D}$-SSH-CTR$_{K_e,K_t}(c)$
  if $st_d = \perp$ then
      return $\perp$
  end if
  cbuff $\leftarrow$ cbuff$\|c$
  if $m_e = \varepsilon$ and $|$cbuff$| \geq L$ then
      Parse cbuff as $\tilde{c}\|A$ (where $|\tilde{c}| = L$)
      $m_e[1] \leftarrow \mathcal{D}$-CTR$_{K_e}(\tilde{c})$
      $LF \leftarrow$ len$(m_e[1])$
      if $LF = \perp_L$ then
          $st_d \leftarrow \perp$
          return $\perp_L$
      else
          need $= 4 + LF +$ maclen
      end if
  end if
  if $|$cbuff$| \geq L$ then
      if $|$cbuff$| \geq$ need then
          Parse cbuff as $\bar{c}[1...n]\|\tau\|B$,
          where $|\bar{c}[1...n]\|\tau| =$ need,
          and $|\tau| =$ maclen
          $m_e[2...n] \leftarrow \mathcal{D}$-CTR$_{K_e}(\bar{c}[2...n])$
          $m_e \leftarrow m_e[1]\|m_e[2...n]$
          $m_t \leftarrow SN_d\|m_e$
          $v \leftarrow \mathcal{V}_{K_t}(m_t, \tau)$
          if $v = 0$ then
              $st_d \leftarrow \perp$
              return $\perp_A$
          else
              $m \leftarrow$ dec$(m_e)$
              $m_e \leftarrow \varepsilon$, cbuff $\leftarrow B$
              return $m$
          end if
      end if
  end if
```

Stage 1:

Arbitrary length input c is appended to the ciphertext buffer `cbuff`.

Stage 2:

Once `cbuff` contains the first block of ciphertext, the packet length field is extracted, and length checking is performed.

Stage 3:

Once `cbuff` contains sufficient data (as determined by the variable `need` in stage 2), decryption and MAC verification are performed. Any remaining bytes (B) are used to reinitatialise `cbuff`.

SSH-CTR – Main Scheme

```
Algorithm $\mathcal{D}$-SSH-CTR$_{K_e,K_t}(c)$
    if $st_d = \perp$ then
        return $\perp$
    end if
    $\texttt{cbuff} \leftarrow \texttt{cbuff} \| c$
    if $m_e = \varepsilon$ and $|\texttt{cbuff}| \geq L$ then
        Parse $\texttt{cbuff}$ as $\tilde{c} \| A$ (where $|\tilde{c}| = L$)
        $m_e[1] \leftarrow \mathcal{D}\text{-CTR}_{K_e}(\tilde{c})$
        $LF \leftarrow \text{len}(m_e[1])$
        if $LF = \perp_L$ then
            $st_d \leftarrow \perp$
            return $\perp_L$
        else
            $\texttt{need} = 4 + LF + \texttt{maclen}$
        end if
    end if
    if $|\texttt{cbuff}| \geq L$ then
        if $|\texttt{cbuff}| \geq \texttt{need}$ then
            Parse $\texttt{cbuff}$ as $\bar{c}[1...n] \| \tau \| B$,
            where $|\bar{c}[1...n] \| \tau| = \texttt{need}$,
            and $|\tau| = \texttt{maclen}$
            $m_e[2...n] \leftarrow \mathcal{D}\text{-CTR}_{K_e}(\bar{c}[2...n])$
            $m_e \leftarrow m_e[1] \| m_e[2...n]$
            $m_t \leftarrow SN_d \| m_e$
            $v \leftarrow \mathcal{V}_{K_t}(m_t, \tau)$
            if $v = 0$ then
                $st_d \leftarrow \perp$
                return $\perp_A$
            else
                $m \leftarrow \text{dec}(m_e)$
                $m_e \leftarrow \varepsilon$, $\texttt{cbuff} \leftarrow B$
                return $m$
            end if
        end if
    end if
```

Stage 1:

Arbitrary length input c is appended to the ciphertext buffer `cbuff`.

Stage 2:

Once `cbuff` contains the first block of ciphertext, the packet length field is extracted, and length checking is performed.

Stage 3:

Once `cbuff` contains sufficient data (as determined by the variable `need` in stage 2), decryption and MAC verification are performed. Any remaining bytes (B) are used to reinitatialise `cbuff`.

SSH-CTR – Main Scheme

```
Algorithm $\mathcal{D}$-SSH-CTR$_{K_e,K_t}(c)$
  if $st_d = \perp$ then
      return $\perp$
  end if
  cbuff $\leftarrow$ cbuff$||c$
  if $m_e = \varepsilon$ and $|$cbuff$| \geq L$ then
      Parse cbuff as $\tilde{c}||A$ (where $|\tilde{c}| = L$)
      $m_e[1] \leftarrow \mathcal{D}$-CTR$_{K_e}(\tilde{c})$
      $LF \leftarrow$ len($m_e[1]$)
      if $LF = \perp_L$ then
          $st_d \leftarrow \perp$
          return $\perp_L$
      else
          need $= 4 + LF +$ maclen
      end if
  end if
  if $|$cbuff$| \geq L$ then
      if $|$cbuff$| \geq$ need then
          Parse cbuff as $\bar{c}[1\ldots n]||\tau||B$,
          where $|\bar{c}[1\ldots n]||\tau| =$ need,
          and $|\tau| =$ maclen
          $m_e[2\ldots n] \leftarrow \mathcal{D}$-CTR$_{K_e}(\bar{c}[2\ldots n])$
          $m_e \leftarrow m_e[1]||m_e[2\ldots n]$
          $m_t \leftarrow SN_d||m_e$
          $v \leftarrow \mathcal{V}_{K_t}(m_t, \tau)$
          if $v = 0$ then
              $st_d \leftarrow \perp$
              return $\perp_A$
          else
              $m \leftarrow$ dec($m_e$)
              $m_e \leftarrow \varepsilon$, cbuff $\leftarrow B$
              return $m$
          end if
      end if
  end if
```

Stage 1:

Arbitrary length input c is appended to the ciphertext buffer `cbuff`.

Stage 2:

Once `cbuff` contains the first block of ciphertext, the packet length field is extracted, and length checking is performed.

Stage 3:

Once `cbuff` contains sufficient data (as determined by the variable `need` in stage 2), decryption and MAC verification are performed. Any remaining bytes (B) are used to reinitatialise `cbuff`.

SSH-CTR – Main Scheme

```
Algorithm $\mathcal{D}$-SSH-CTR$_{K_e,K_t}(c)$
  if $st_d = \perp$ then
      return $\perp$
  end if
  cbuff $\leftarrow$ cbuff $\|$ $c$
  if $m_e = \varepsilon$ and $|$cbuff$| \geq L$ then
      Parse cbuff as $\tilde{c} \| A$ (where $|\tilde{c}| = L$)
      $m_e[1] \leftarrow \mathcal{D}$-CTR$_{K_e}(\tilde{c})$
      $LF \leftarrow$ len$(m_e[1])$
      if $LF = \perp_L$ then
          $st_d \leftarrow \perp$
          return $\perp_L$
      else
          need $= 4 + LF +$ maclen
      end if
  end if
  if $|$cbuff$| \geq L$ then
      if $|$cbuff$| \geq$ need then
          Parse cbuff as $\bar{c}[1...n] \| \tau \| B$,
          where $|\bar{c}[1...n] \| \tau| =$ need,
          and $|\tau| =$ maclen
          $m_e[2...n] \leftarrow \mathcal{D}$-CTR$_{K_e}(\bar{c}[2...n])$
          $m_e \leftarrow m_e[1] \| m_e[2...n]$
          $m_t \leftarrow SN_d \| m_e$
          $v \leftarrow \mathcal{V}_{K_t}(m_t, \tau)$
          if $v = 0$ then
              $st_d \leftarrow \perp$
              return $\perp_A$
          else
              $m \leftarrow$ dec$(m_e)$
              $m_e \leftarrow \varepsilon$, cbuff $\leftarrow B$
              return $m$
          end if
      end if
  end if
```

Stage 1:

Arbitrary length input c is appended to the ciphertext buffer `cbuff`.

Stage 2:

Once `cbuff` contains the first block of ciphertext, the packet length field is extracted, and length checking is performed.

Stage 3:

Once `cbuff` contains sufficient data (as determined by the variable `need` in stage 2), decryption and MAC verification are performed. Any remaining bytes (B) are used to reinitatialise `cbuff`.

SSH-CTR – Main Scheme

```
Algorithm $\mathcal{D}$-SSH-CTR$_{K_e,K_t}(c)$
  if $st_d = \perp$ then
      return $\perp$
  end if
  cbuff $\leftarrow$ cbuff $\| c$
  if $m_e = \varepsilon$ and $|$cbuff$| \geq L$ then
      Parse cbuff as $\tilde{c} \| A$ (where $|\tilde{c}| = L$)
      $m_e[1] \leftarrow \mathcal{D}$-CTR$_{K_e}(\tilde{c})$
      $LF \leftarrow$ len$(m_e[1])$
      if $LF = \perp_L$ then
          $st_d \leftarrow \perp$
          return $\perp_L$
      else
          need $= 4 + LF +$ maclen
      end if
  end if
  if $|$cbuff$| \geq L$ then
      if $|$cbuff$| \geq$ need then
          Parse cbuff as $\bar{c}[1 \ldots n] \| \tau \| B$,
          where $|\bar{c}[1 \ldots n] \| \tau| =$ need,
          and $|\tau| =$ maclen
          $m_e[2 \ldots n] \leftarrow \mathcal{D}$-CTR$_{K_e}(\bar{c}[2 \ldots n])$
          $m_e \leftarrow m_e[1] \| m_e[2 \ldots n]$
          $m_t \leftarrow SN_d \| m_e$
          $v \leftarrow \mathcal{V}_{K_t}(m_t, \tau)$
          if $v = 0$ then
              $st_d \leftarrow \perp$
              return $\perp_A$
          else
              $m \leftarrow$ dec$(m_e)$
              $m_e \leftarrow \varepsilon$, cbuff $\leftarrow B$
              return $m$
          end if
      end if
  end if
```

Stage 1:

Arbitrary length input c is appended to the ciphertext buffer `cbuff`.

Stage 2:

Once `cbuff` contains the first block of ciphertext, the packet length field is extracted, and length checking is performed.

Stage 3:

Once `cbuff` contains sufficient data (as determined by the variable `need` in stage 2), decryption and MAC verification are performed. Any remaining bytes (B) are used to reinitatialise `cbuff`.

New Security Model

- We now need a new security model which considers decryption with buffers.
- Our analysis can then capture exactly how an implementation operates.
- In particular, this gives adversaries all the capabilities required for APW style plaintext-recovery attacks.
- This makes our analysis much more meaningful in practice.

New Security Model

- We now need a new security model which considers decryption with buffers.
- Our analysis can then capture exactly how an implementation operates.
- In particular, this gives adversaries all the capabilities required for APW style plaintext-recovery attacks.
- This makes our analysis much more meaningful in practice.

New Security Model

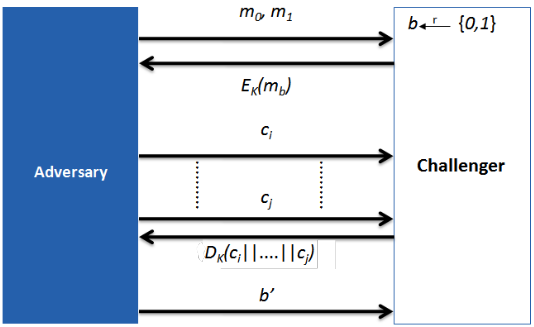
- We now need a new security model which considers decryption with buffers.
- Our analysis can then capture exactly how an implementation operates.
- In particular, this gives adversaries all the capabilities required for APW style plaintext-recovery attacks.
- This makes our analysis much more meaningful in practice.

New Security Model

- We now need a new security model which considers decryption with buffers.
- Our analysis can then capture exactly how an implementation operates.
- In particular, this gives adversaries all the capabilities required for APW style plaintext-recovery attacks.
- This makes our analysis much more meaningful in practice.

LOR-BSF-CCA

Left-or-right indistinguishability under chosen-ciphertext attack with buffered stateful decryption (LOR-BSF-CCA).



The adversary wins if $b' = b$. With advantage,

$$\mathbf{Adv}^{\text{lor-bsf-cca}} = \left|\Pr[b' = b] - \frac{1}{2}\right|.$$

LOR-BSF-CCA

- We refer to decryption of a full ciphertext packet as a sequence of decryption queries.
- As with the decryption oracle in BKN's model, our buffered decryption oracle is stateful.

LOR-BSF-CCA

- We refer to decryption of a full ciphertext packet as a sequence of decryption queries.
- As with the decryption oracle in BKN's model, our buffered decryption oracle is stateful.

Our Main Result

Theorem

SSH-CTR[F] is LOR-BSF-CCA secure if:

- F is a pseudorandom function family,
- $\mathcal{T}$ (the tagging algorithm from $\mathcal{MA}$) is a pseudorandom function family,
- $\mathcal{MA}$ is strongly unforgeable (SUF-CMA secure),
- the adversary is restricted to at most 2^{32} encryption queries and 2^{32} sequences of decryption queries.

$$\mathbf{Adv}^{\text{lor-bsf-cca}}_{\text{SSH-CTR}[F]} \leq 2\mathbf{Adv}^{\text{suf-cma}}_{\mathcal{MA}} + 2\mathbf{Adv}^{\text{prf}}_{F} + 4\mathbf{Adv}^{\text{prf}}_{\mathcal{T}}$$

Outline

Summary

- We have expanded the existing analysis of BKN by:
  - Developing a security model considering buffered decryption.
  - Giving a definition of SSH using counter mode that is closely linked to the SSH RFCs and the OpenSSH implementation.
    - Drawback: Despite being as general as possible in our modelling of SSH, our security results are now specific to OpenSSH.
- Our approach is sufficiently powerful to incorporate the attacks of APW.
- This closes the gap that existed between the formal security analysis of SSH and the way in which SSH should be (and is in practice) implemented.

Summary

- We have expanded the existing analysis of BKN by:
  - Developing a security model considering buffered decryption.
  - Giving a definition of SSH using counter mode that is closely linked to the SSH RFCs and the OpenSSH implementation.
    - Drawback: Despite being as general as possible in our modelling of SSH, our security results are now specific to OpenSSH.
- Our approach is sufficiently powerful to incorporate the attacks of APW.
- This closes the gap that existed between the formal security analysis of SSH and the way in which SSH should be (and is in practice) implemented.

Summary

- We have expanded the existing analysis of BKN by:
    - Developing a security model considering buffered decryption.
    - Giving a definition of SSH using counter mode that is closely linked to the SSH RFCs and the OpenSSH implementation.
        - Drawback: Despite being as general as possible in our modelling of SSH, our security results are now specific to OpenSSH.
- Our approach is sufficiently powerful to incorporate the attacks of APW.
- This closes the gap that existed between the formal security analysis of SSH and the way in which SSH should be (and is in practice) implemented.

Summary

- We have expanded the existing analysis of BKN by:
  - Developing a security model considering buffered decryption.
  - Giving a definition of SSH using counter mode that is closely linked to the SSH RFCs and the OpenSSH implementation.
    - Drawback: Despite being as general as possible in our modelling of SSH, our security results are now specific to OpenSSH.
- Our approach is sufficiently powerful to incorporate the attacks of APW.
- This closes the gap that existed between the formal security analysis of SSH and the way in which SSH should be (and is in practice) implemented.

Thanks

Questions?