Encyption In Protocols

Viet Tung Hoang
Agenda

1. Nonced-based AE with Associated Data

2. SSH Encryption

3. Onion encryption and Tagging Attack
Classical Encryption Needs Random IVs

CBC fails if IV is predictable
But Generating **Good Randomness Is Not Easy**

A bug in Debian Linux causes OpenSSL to get entropy only from process ID

**Dual EC: A Standardized Back Door**

The NIST standard Dual EC is NSA-backdoored

**Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices**

Linux /dev/urandom produces output even if entropy pool is depleted
Nonce-based Encryption

Nonce, a (user-provided) string that should **never repeat**. Implemented as a random string or a counter.

Nonce is **not** a part of the ciphertext

It can be sent along the ciphertext, or is implicit (as a synchronized counter)
Example: Nonce-based CTR

Assume that nonces are 96-bit

\[ E_K \]

\[ M_1 \rightarrow C_1 \]

\[ M_2 \rightarrow C_2 \]

\[ M_3 \rightarrow C_3 \]

32-bit counter
When Some Data Can’t Be Encrypted

Issue: Can’t encrypt packet headers, because intermediate routers need to read them

Associated data (AD): a string that can’t be encrypted but should be authenticated
Encrypt-then-MAC with Associated Data

Security **breaks down** if the AD length is not fed into MAC.
Real-world Nonce-based AE with AD

Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality

NIST Special Publication 800-38C
Morris Dworkin

CCM: Used in IPSec and WPA2 (WiFi encryption)

Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC

NIST Special Publication 800-38D
November, 2007
Morris Dworkin

GCM: Used in SSH and TLS 1.3

Both (loosely) follow the Encrypt-then-MAC pattern
Caveat: Nonces May Be Repeated

We assume that nonces don’t repeat, but in practice they do.

- Devices reboot and reset counters.
- QUIC generates hundreds of millions of random 96-bit nonces per second.
- KRACK attack on WPA2: Exploit a bug to force devices to reset nonces.

Most existing schemes break down completely if nonces repeat.
Agenda

1. Nonced-based AE with Associated Data

2. SSH Encryption

3. Onion Encryption and Tagging Attack
SSH

Aim to replace insecure Unix tools (rlogin, telnet) by adding encryption and authentication
SSH Encryption: Encrypt-and-MAC

\[ M \rightarrow \text{Encode} \rightarrow \text{CBC} \rightarrow C \]

\[ \text{len}(M) \| \text{len}(\text{pad}) \rightarrow M \rightarrow \text{pad} \rightarrow T \rightarrow \text{MAC} \]

Privacy | Authenticity
---|---
Yes | Yes

---

CBC

PAD
SSH Boundary Hiding

When there are many encrypted SSH packets sent over network

**SSH’s design goal:** boundary hiding

Adversary shouldn’t be able to tell the boundary of packets

**Reason:** Frustrate traffic analysis that learns info of data from size
An Issue: Non-atomic CBC Decryption

Receiver doesn’t know the boundary of packets

Decrypt the first 32 bits to know the length of packet 1

Decrypt the rest of packet 1

Non-atomic decryption: CBC-decryption is broken into two steps
An Attack On Non-atomic Decryption

Goal: Recover the first 4 bytes of the stream
An Attack On Non-atomic Decryption

Send the first 4B of the ciphertext stream as a part of a new stream
An Attack On Non-atomic Decryption

Decrypt and interpret as a length

Wait for 96 bytes for message, and 16 bytes for MAC
An Attack On Non-atomic Decryption

Send an additional byte

Wait for MAC tag to authenticate
An Attack On Non-atomic Decryption

Send an additional byte

Wait for MAC tag to authenticate
An Attack On Non-atomic Decryption

Eventually send 112 bytes

MAC tag is invalid, reject

Learn that the message is 96
Agenda

1. Nonced-based AE with Associated Data

2. SSH Encryption

3. Onion Encryption and Tagging Attack
Recap: Tor ("The Onion Router")

Tor operates by tunnelling traffic through three random "onion routers"
Knows Alice is using Tor and the identity of the middle node, but not the destination
Who Knows What

Knows someone is connecting to destination, but not which user
Onion routing

- Src: 1.2.3.4, Dest: entry
- Encrypted with entry’s key
- Src: entry, Dest: middle
- Encrypted with middle’s key
- Src: middle, Dest: exit
- Encrypted with exit’s key
- Src: exit, Dest: 5.6.7.8
- HTTP packet
MAC-then-Enc; encryption is CTR

CTR mode

CTR mode
Tagging Attack

Malicious routers want to identify what service user $U$ is using
Tagging Attack

Suppose malicious nodes are chosen to be entry and exit

**Problem:** How does exit know that it is processing user $U$?

- CTR with exit’s key
- CTR with middle’s key
Tagging Attack

CTR is **malleable**: XOR $X$ to ciphertext $\rightarrow$ XOR $X$ to data

Pre-shared $X$

<table>
<thead>
<tr>
<th>Entry</th>
<th>Middle</th>
<th>Exit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T \oplus X$</td>
<td>$M$</td>
<td>Pre-shared $X$</td>
</tr>
</tbody>
</table>
Tagging Attack

Pre-shared $X$  \hspace{1cm}  Middle  \hspace{1cm}  Pre-shared $X$

Entry  \hspace{2cm}  $T \oplus X$  \hspace{1cm}  $M$  \hspace{1cm}  Exit
Tagging Attack

Pre-shared $X$

Entry → Middle → Exit

Pre-shared $X$

$T \oplus X \quad M$

- MAC checking fails if use given tag
- Pass if xor $X$ to the tag
Tagging Attack

What if only one malicious node is chosen?

Pre-shared $X$

Tag checking fails at exit; this route is less likely to be chosen

Reinforce the routes of two malicious nodes