CNT 5412, Spring 2025

Encryption In Protocols

Viet Tung Hoang

Agenda

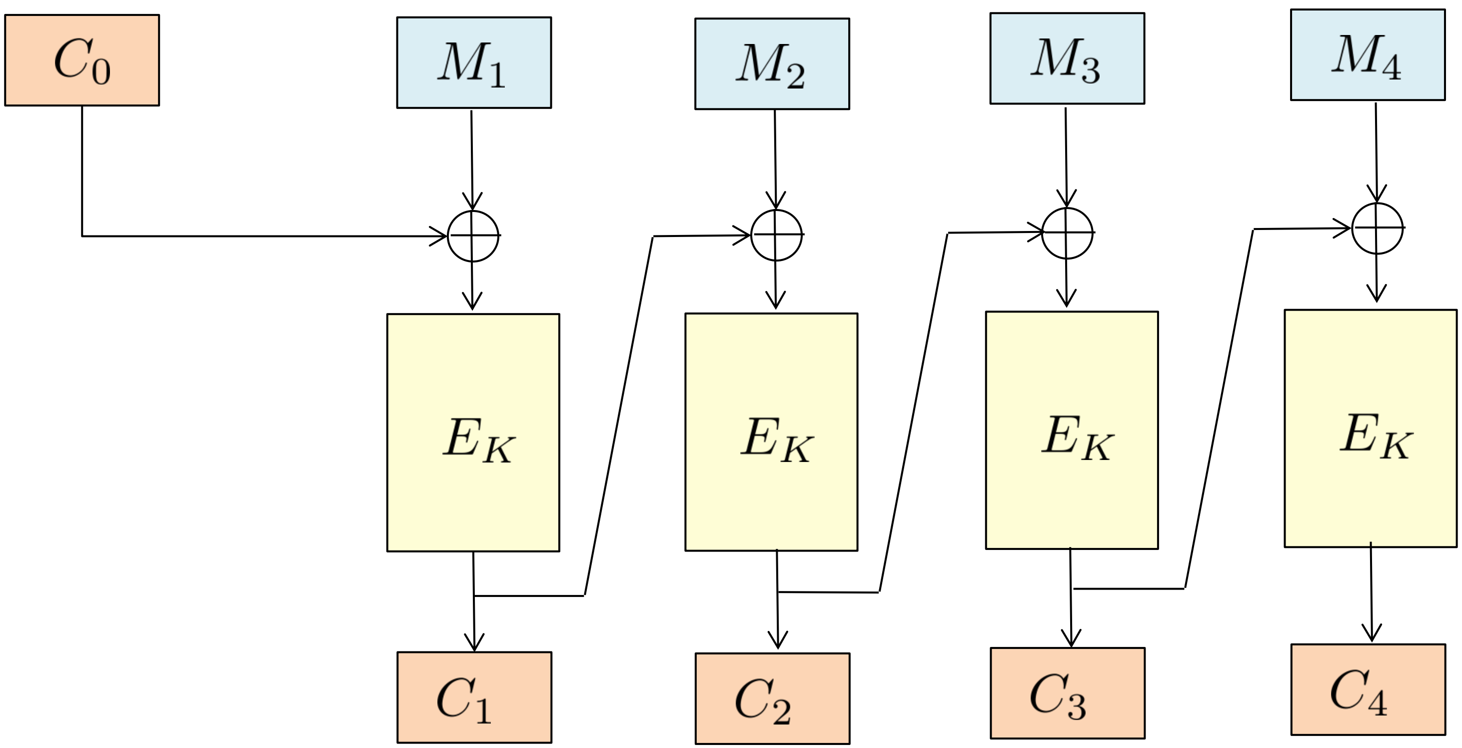
1. Nonced-based AE with Associated Data

2. SSH Encryption

3. Streaming Encryption

4. Onion encryption and Tagging Attack

Classical Encryption Needs Random IVs



CBC fails if IV is predictable

But Generating Good Randomness Is Not Easy

OpenSSL Cryptography and SSL/TLS Toolkit

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

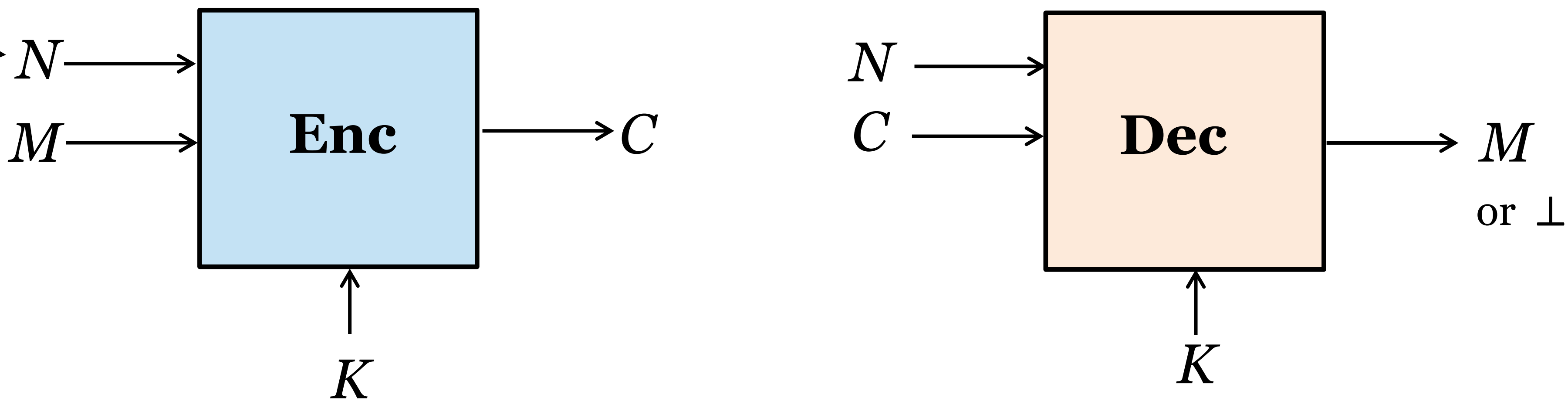
Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices

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.

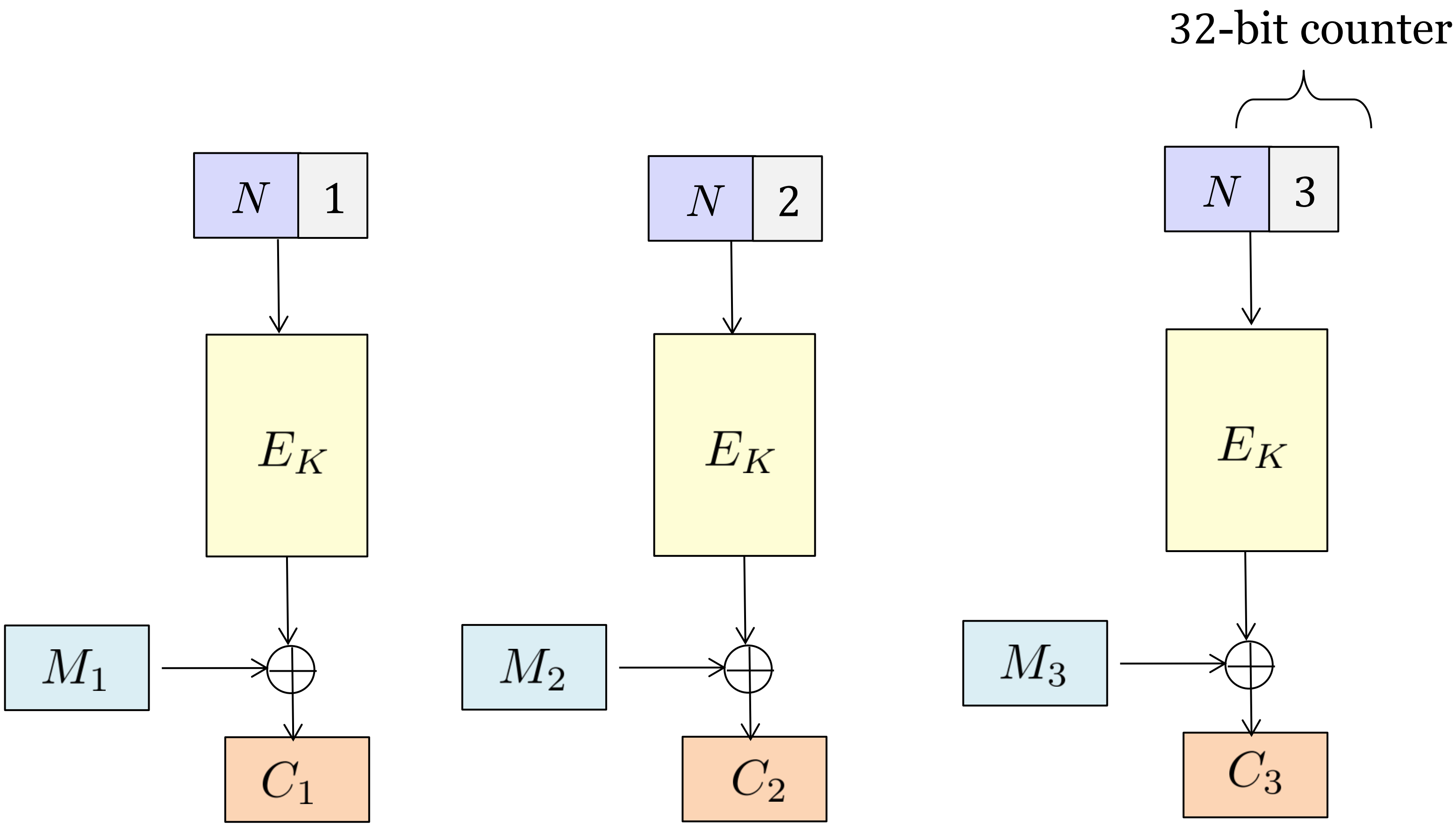


Nonce is **not** a part of the ciphertext

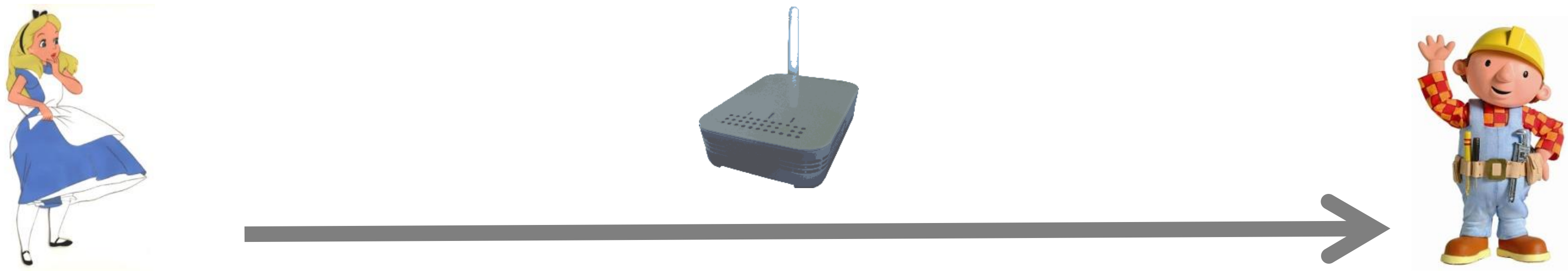
It can be sent along the ciphertext, or is implicit (as a synchronized counter)

Example: Nonce-based CTR

Assume that nonces are 96-bit

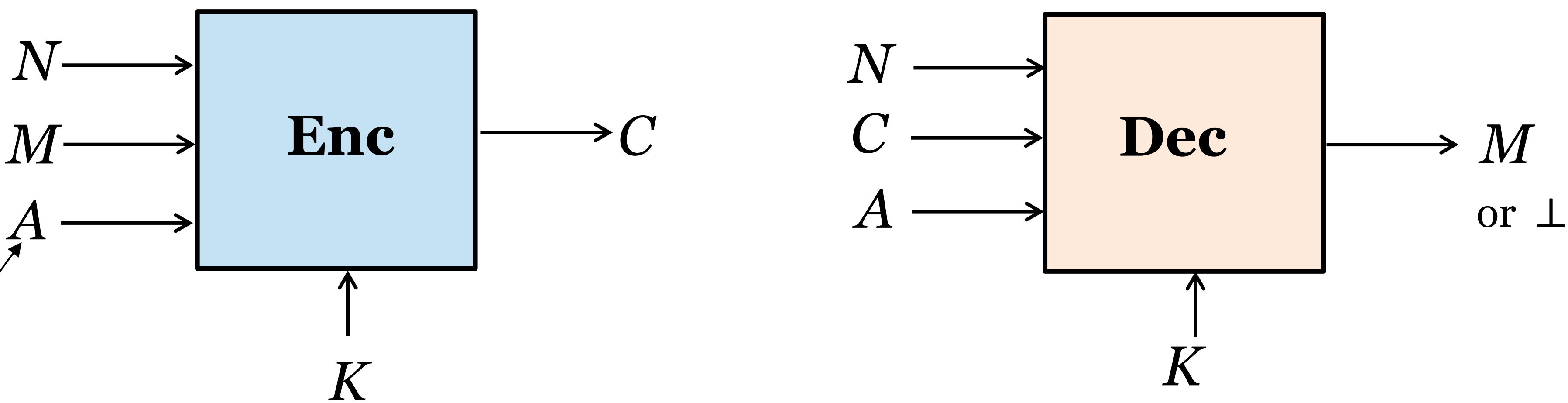


When Some Data Can't Be Encrypted



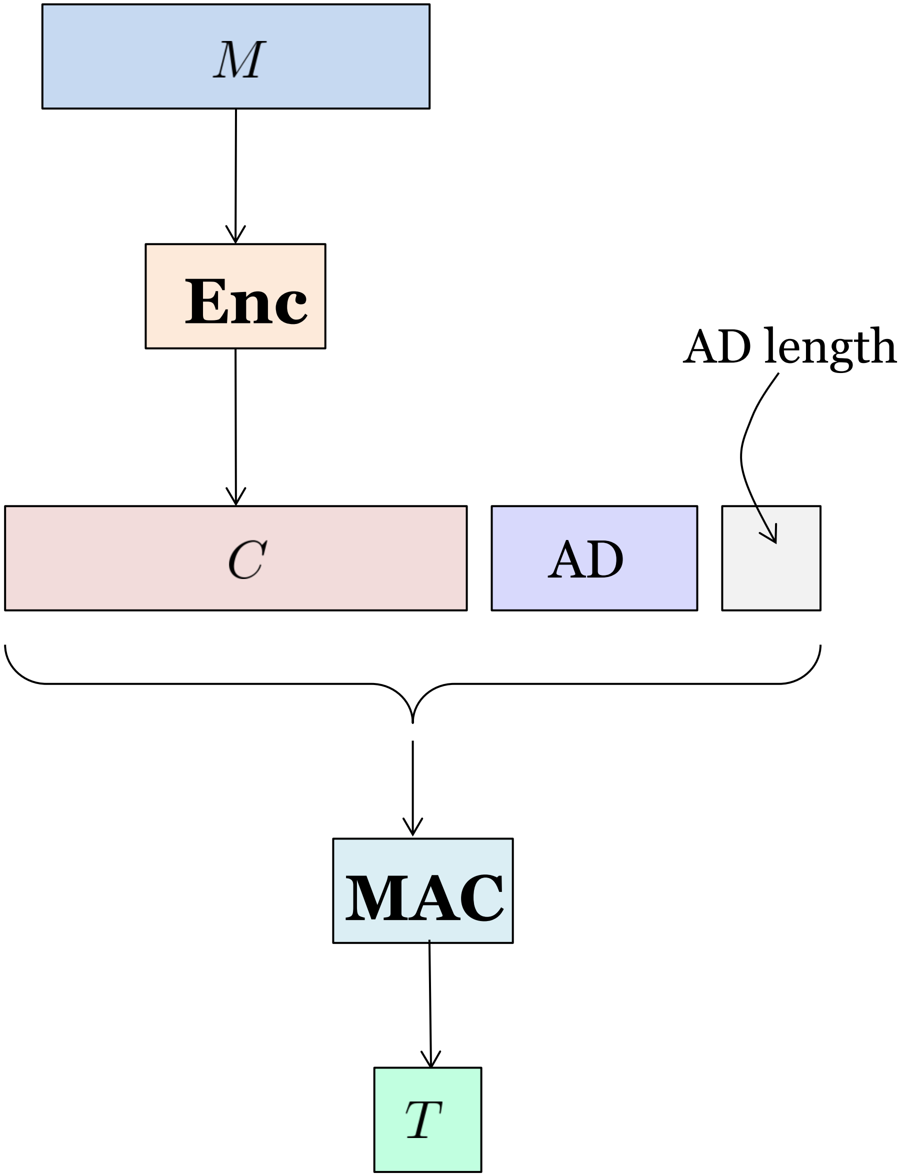
Header	Payload: $$$

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

NIST Special Publication 800-38C

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

National Institute of Standards and Technology
Technology Administration
U.S. Department of Commerce

Morris Dworkin

CCM: Used in IPSec and WPA2 (WiFi encryption)

NIST Special Publication 800-38D
November, 2007

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

National Institute of Standards and Technology
U.S. Department of Commerce

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 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. Streaming Encryption

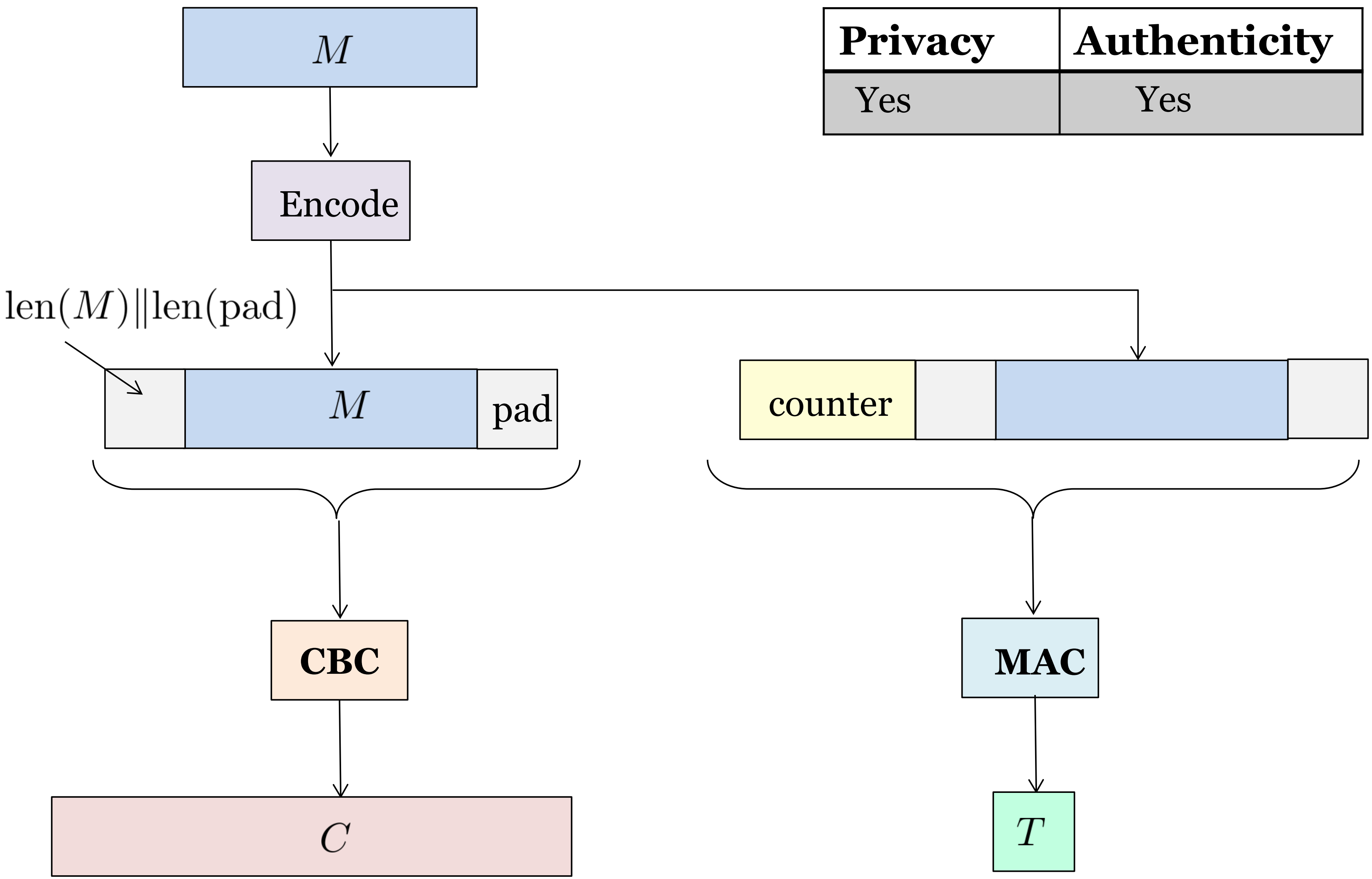
4. Onion Encryption and Tagging Attack

SSH

Aim to replace insecure Unix tools (rlogin, telnet) by adding encryption and authentication

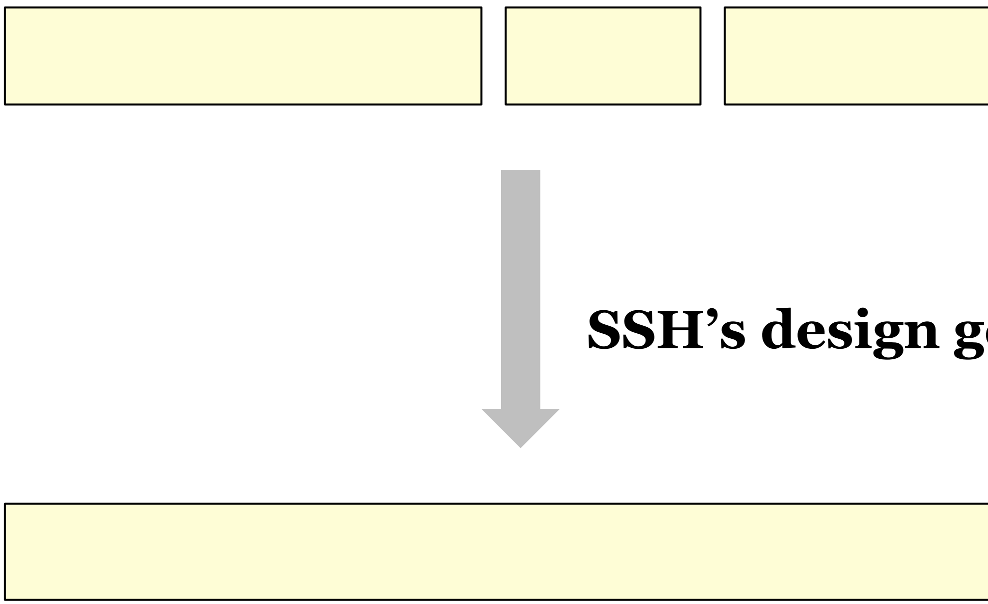
SSH Encryption: Encrypt-and-MAC

Privacy	**Authenticity**
Yes	Yes



SSH Boundary Hiding

When there are many encrypted SSH packets sent over network

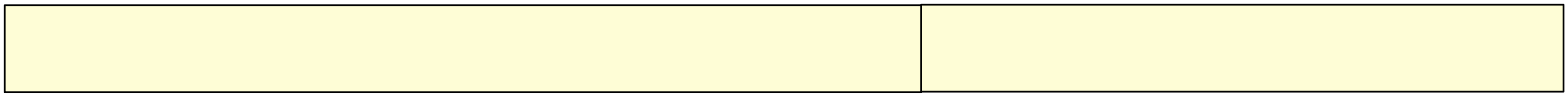


SSH's design goal: boundary hiding

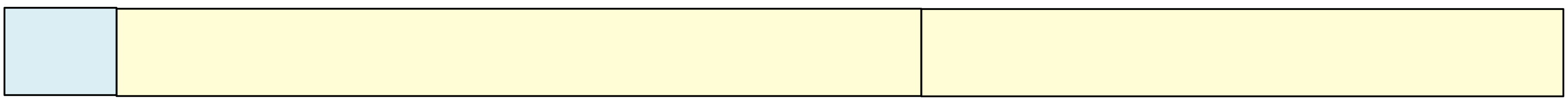
Adversary shouldn't be able to tell the boundary of packets

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

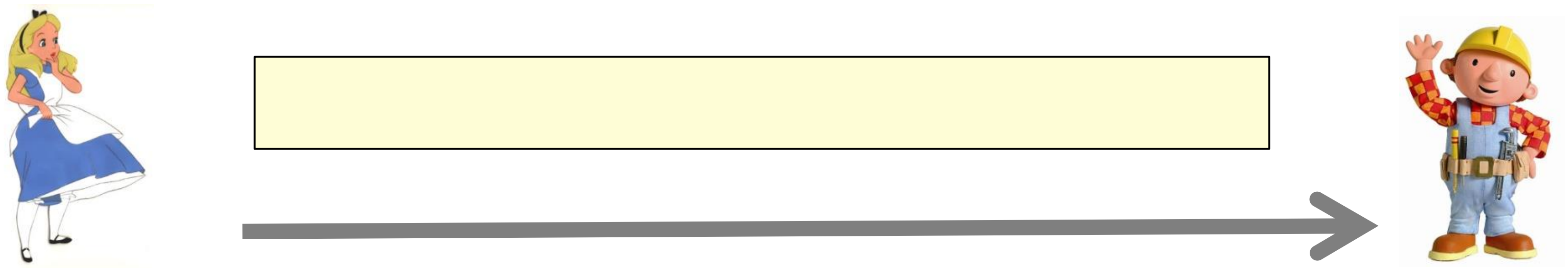


Decrypt the first 32 bits to know the length of packet 1

Decrypt the rest of packet 1

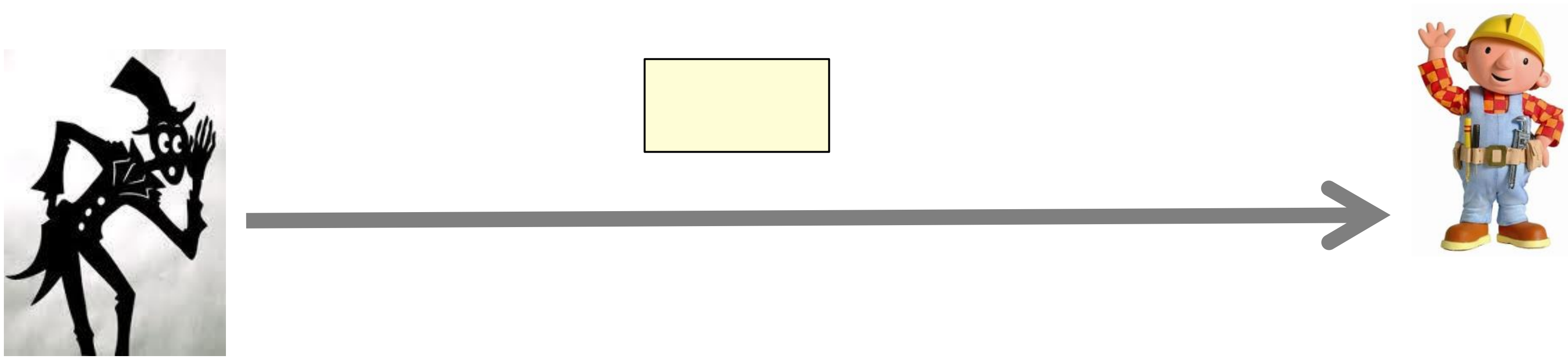
Non-atomic decryption: CBC-decryption is broken into two steps

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

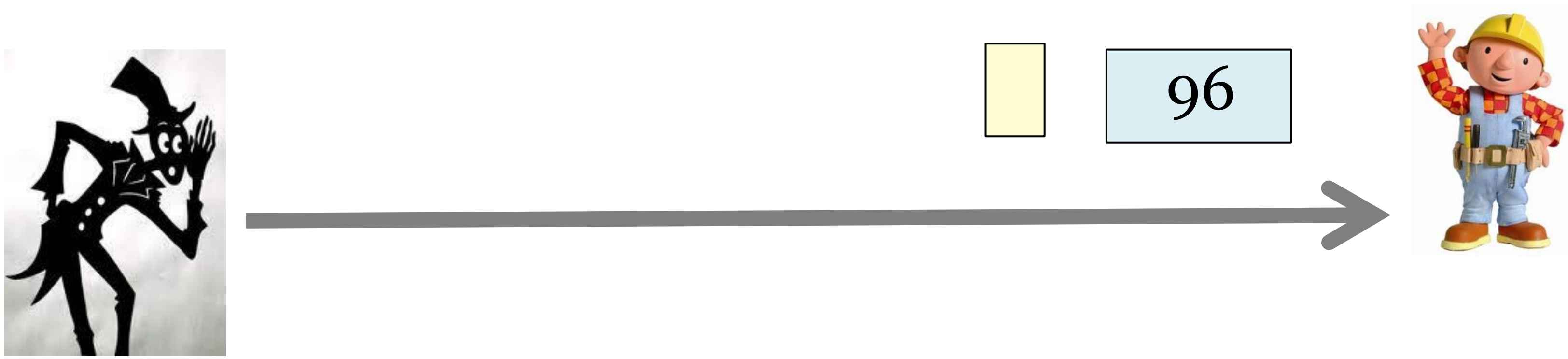
96

Decrypt and interpret as a length

Wait for 96 bytes for message,

and 16 bytes for MAC

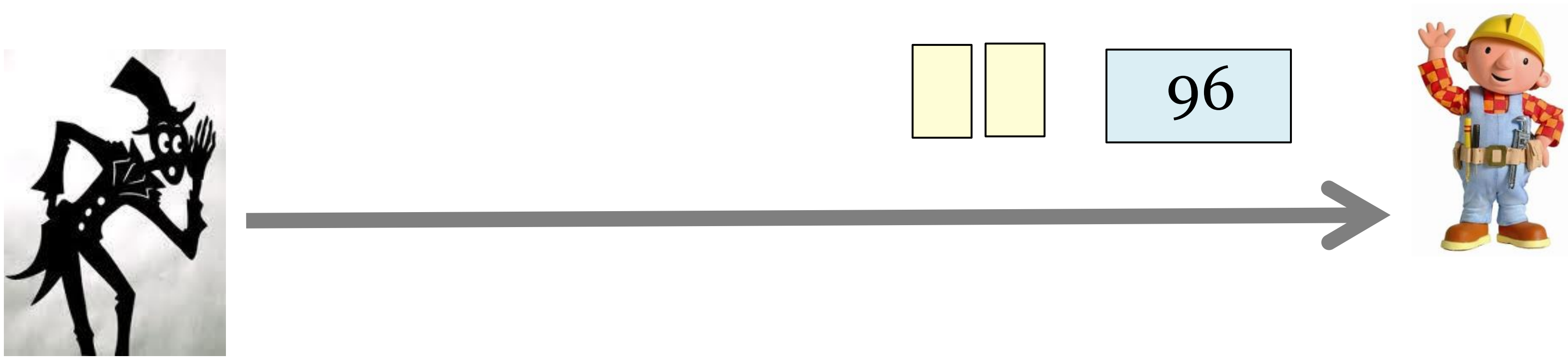
An Attack On Non-atomic Decryption



Send an additional byte

Wait for MAC tag to authenticate

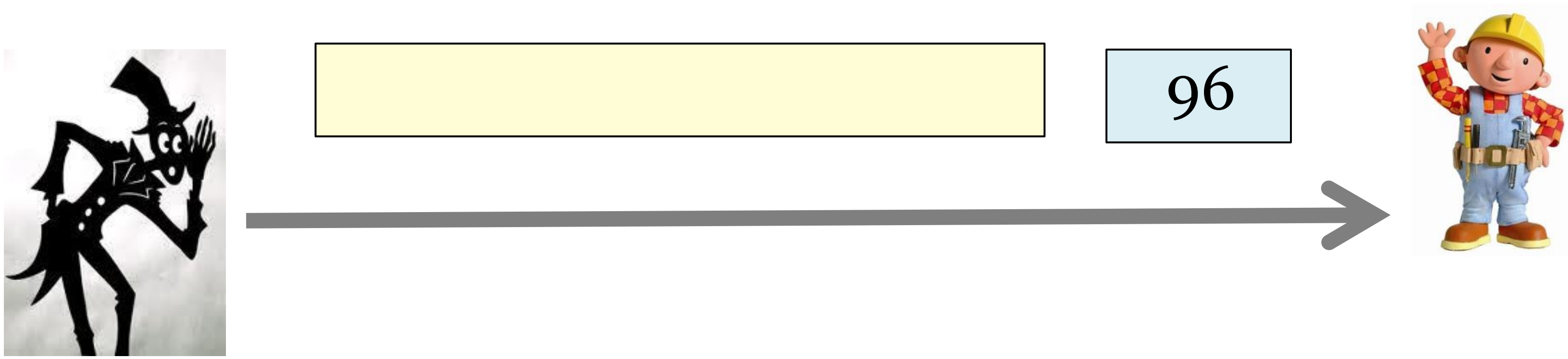
An Attack On Non-atomic Decryption



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. **Streaming Encryption**

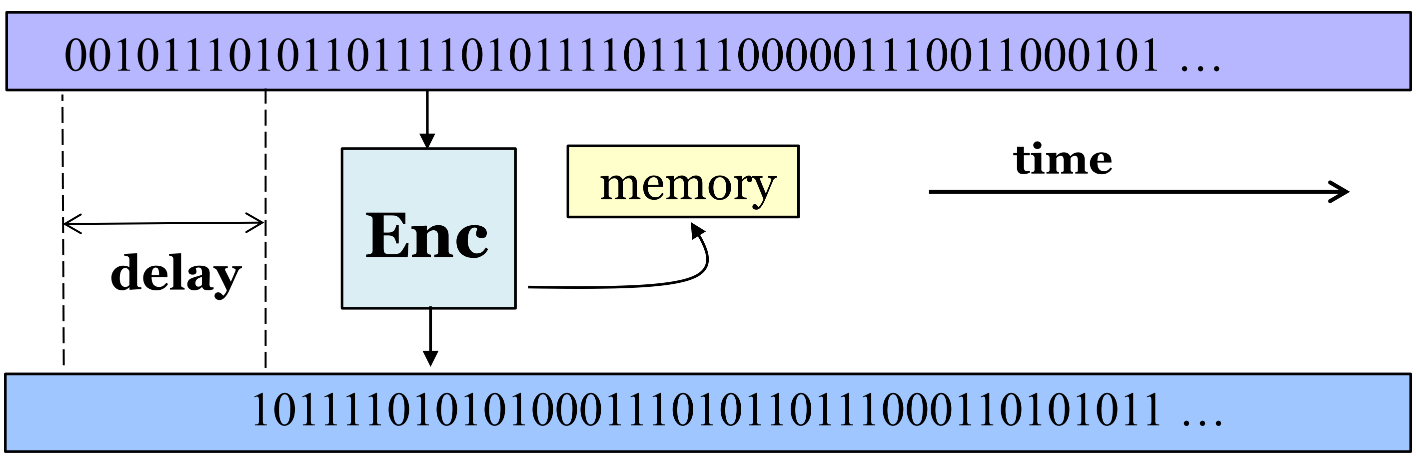
4. Onion Encryption and Tagging Attack

The Stream Setting

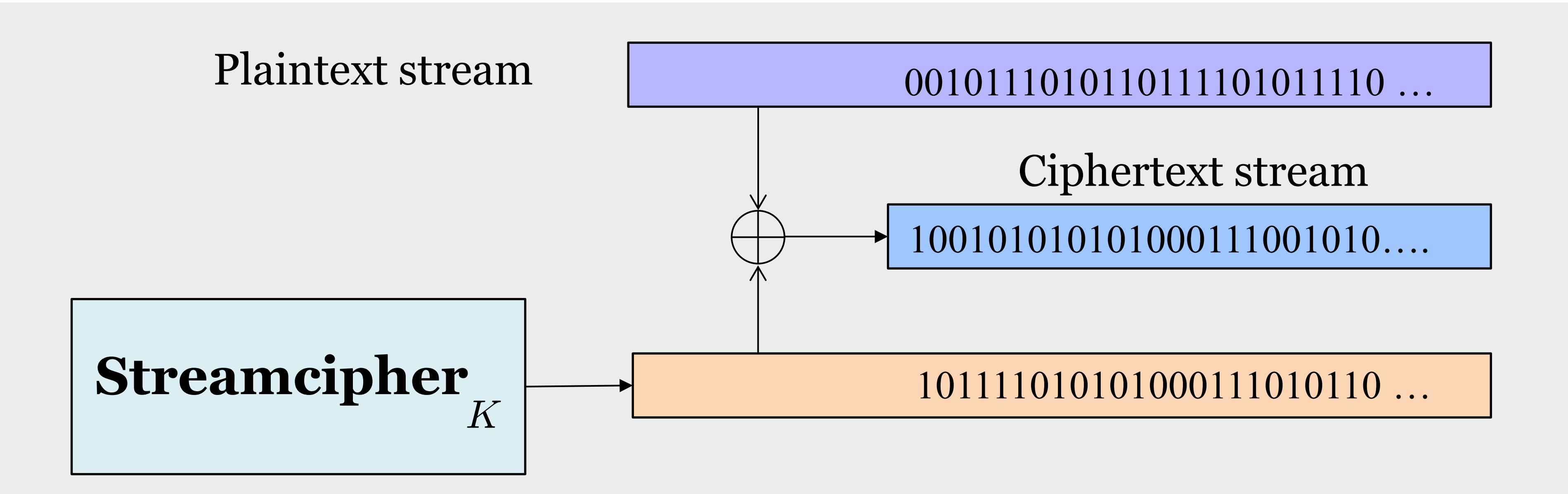
Streaming apps,

low-end devices,

real-time apps



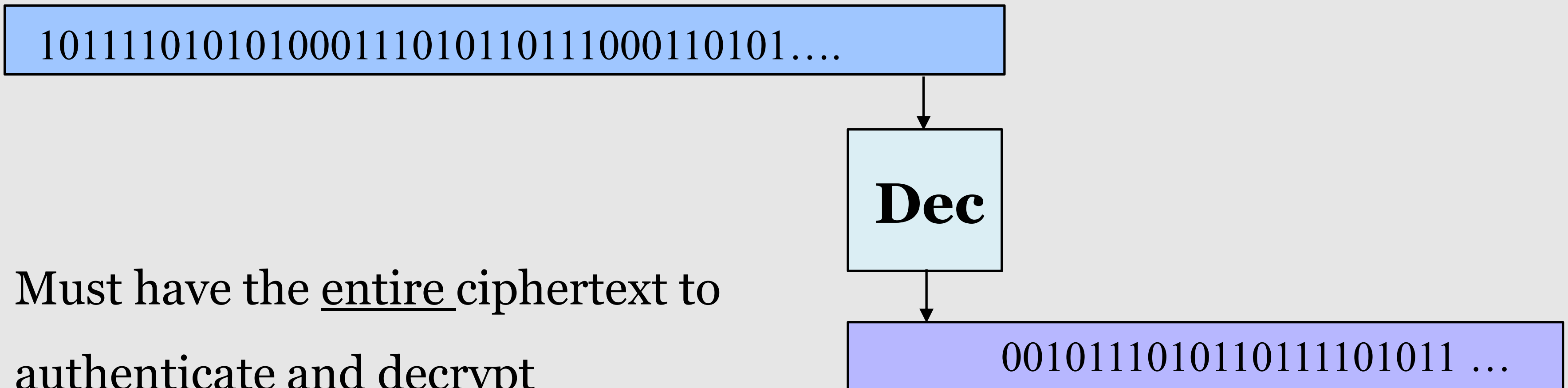
A Naïve Way To Encrypt Stream



Issue: No authenticity

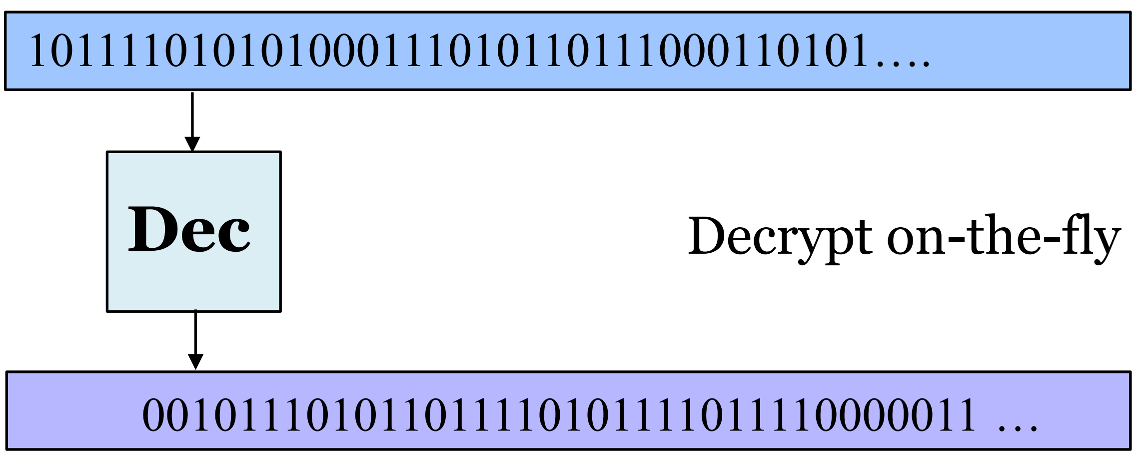
But Adding Authenticity Breaks Usability

What standard AE provides



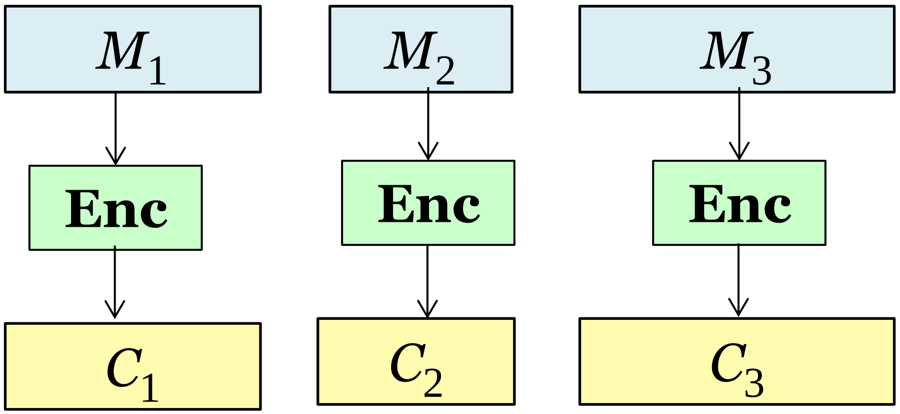
Must have the <u>entire</u> ciphertext to authenticate and decrypt

What users want

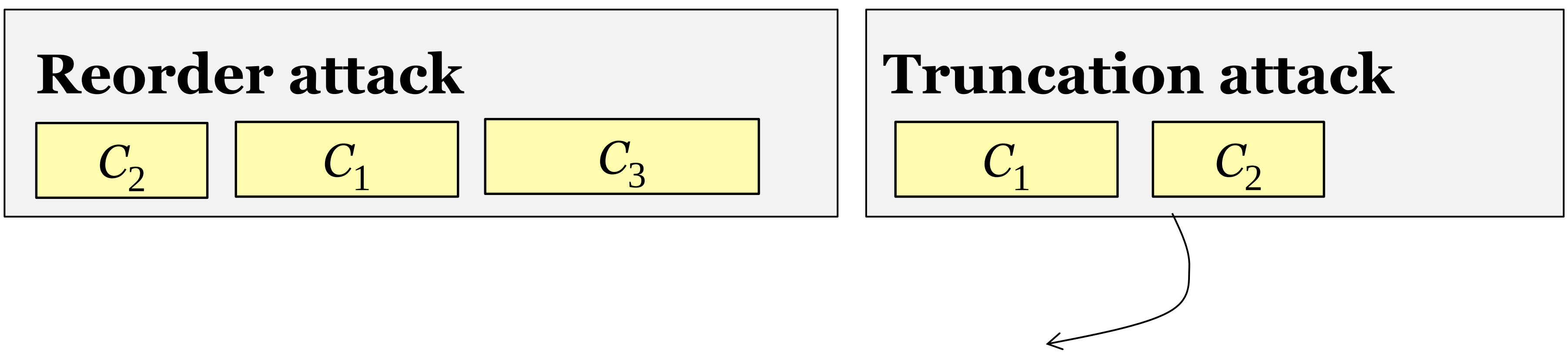


Decrypt on-the-fly

Chop A Long Message Into Small Chunks?



This leads to more authenticity issues



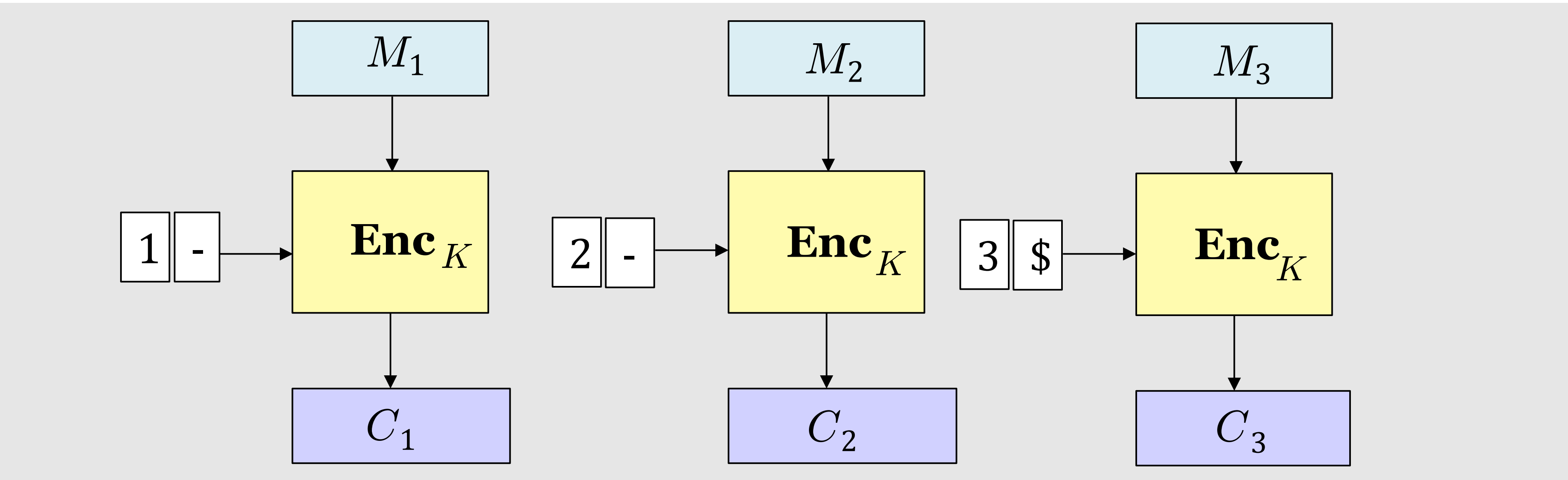
Cookie Cutter attack on TLS: Steal TLS cookie

How To Encrypt Stream

Hoang et al, CRYPTO 2015,

adopted by Google's Tink library

Chop a long message to small chunks



Chunk size is user-selectable

1 MB

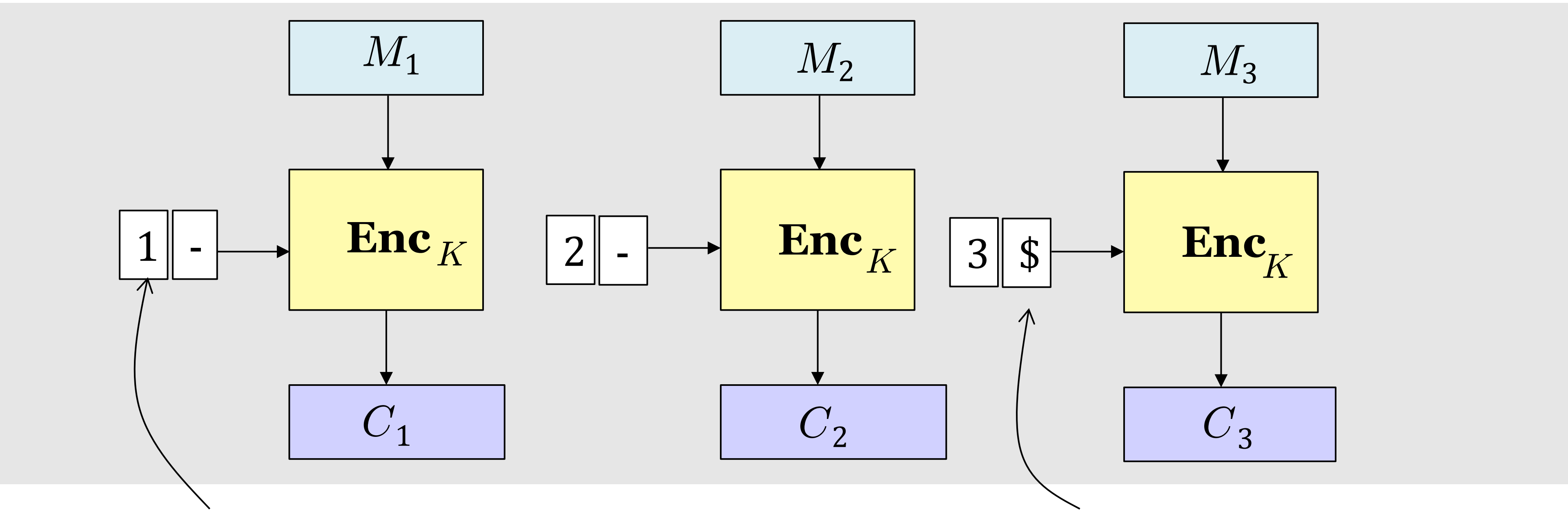
1 KB

1 character

How To Encrypt Stream

Hoang et al, CRYPTO 2015, adopted by Google's Tink library

Chop a long message to small chunks



Use a counter to enforce order

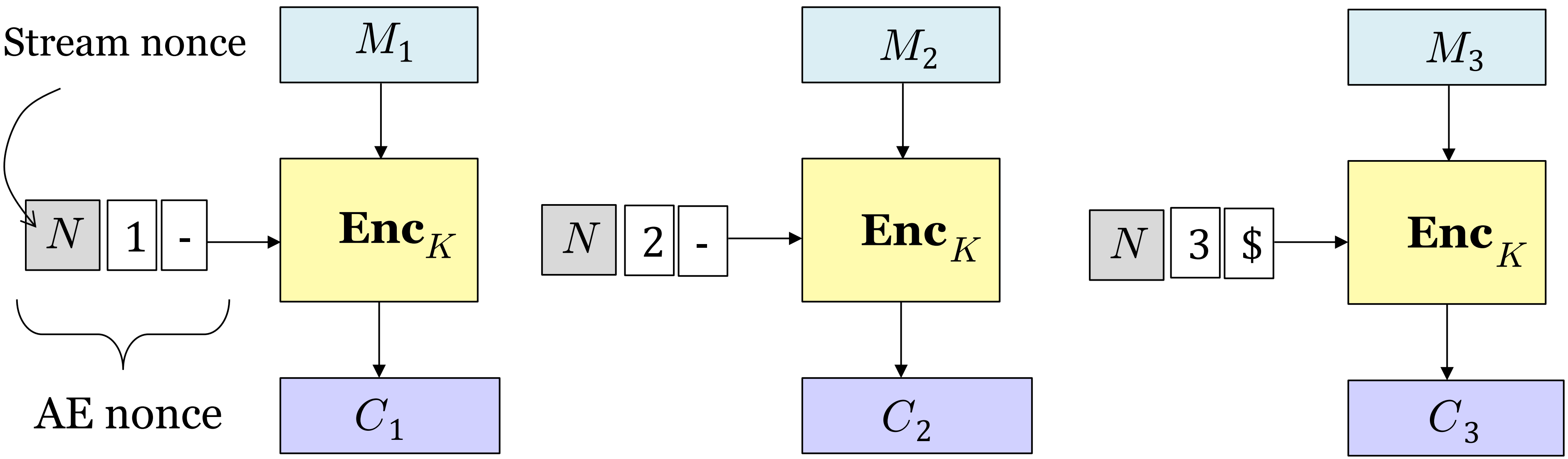
Signal the last chunk

(TLS relies on apps to enforce this)

Include counter and signal **without** extra cost

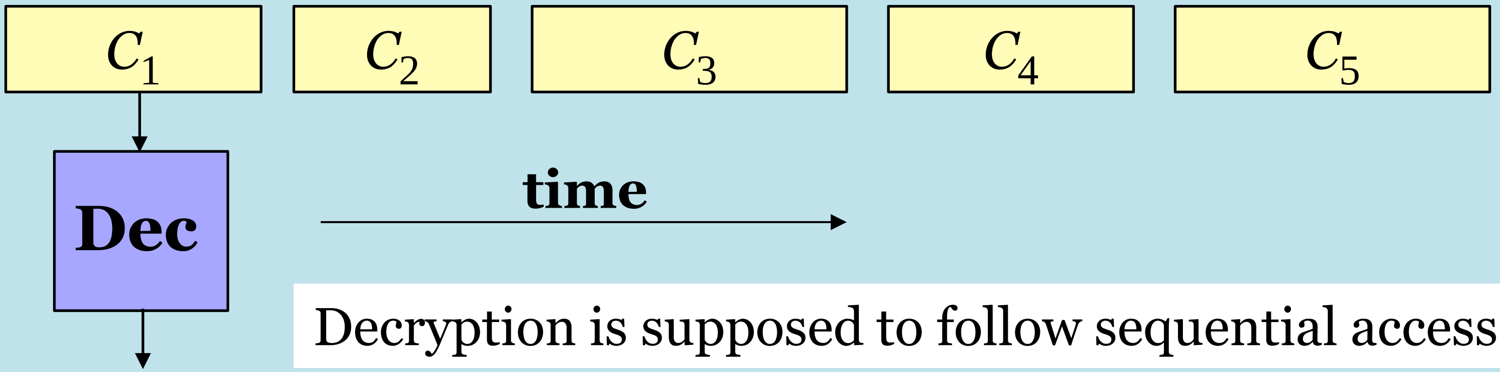
The Trick of Having No Extra Cost

Embed counter and signal into the nonce



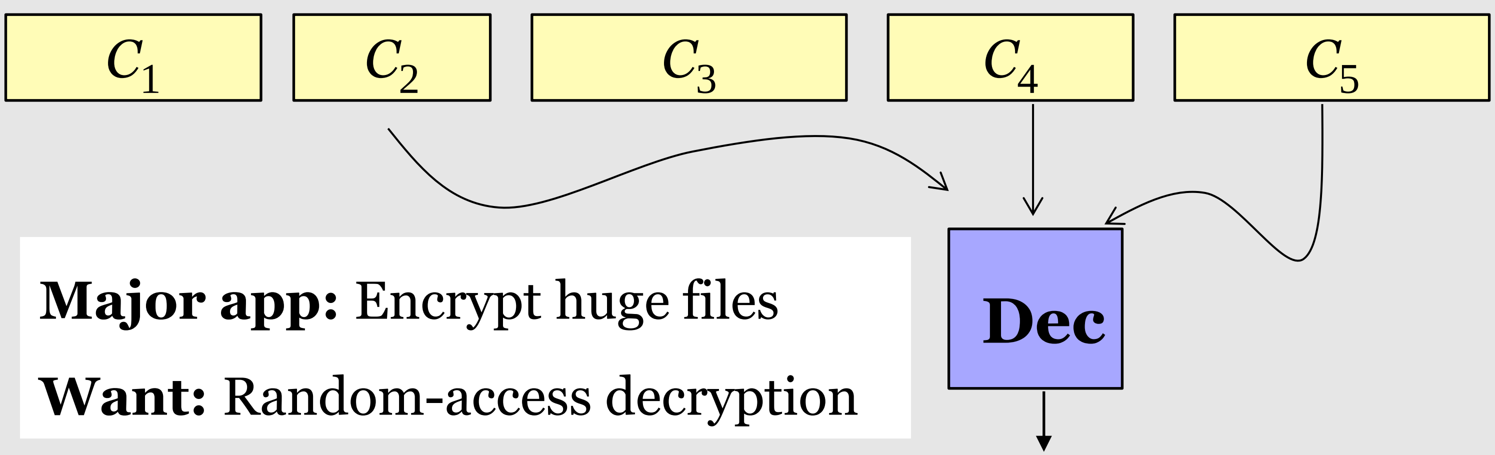
Subtlety in Security Modeling

What “streaming decryption” intuitively suggests



Decryption is supposed to follow sequential access?

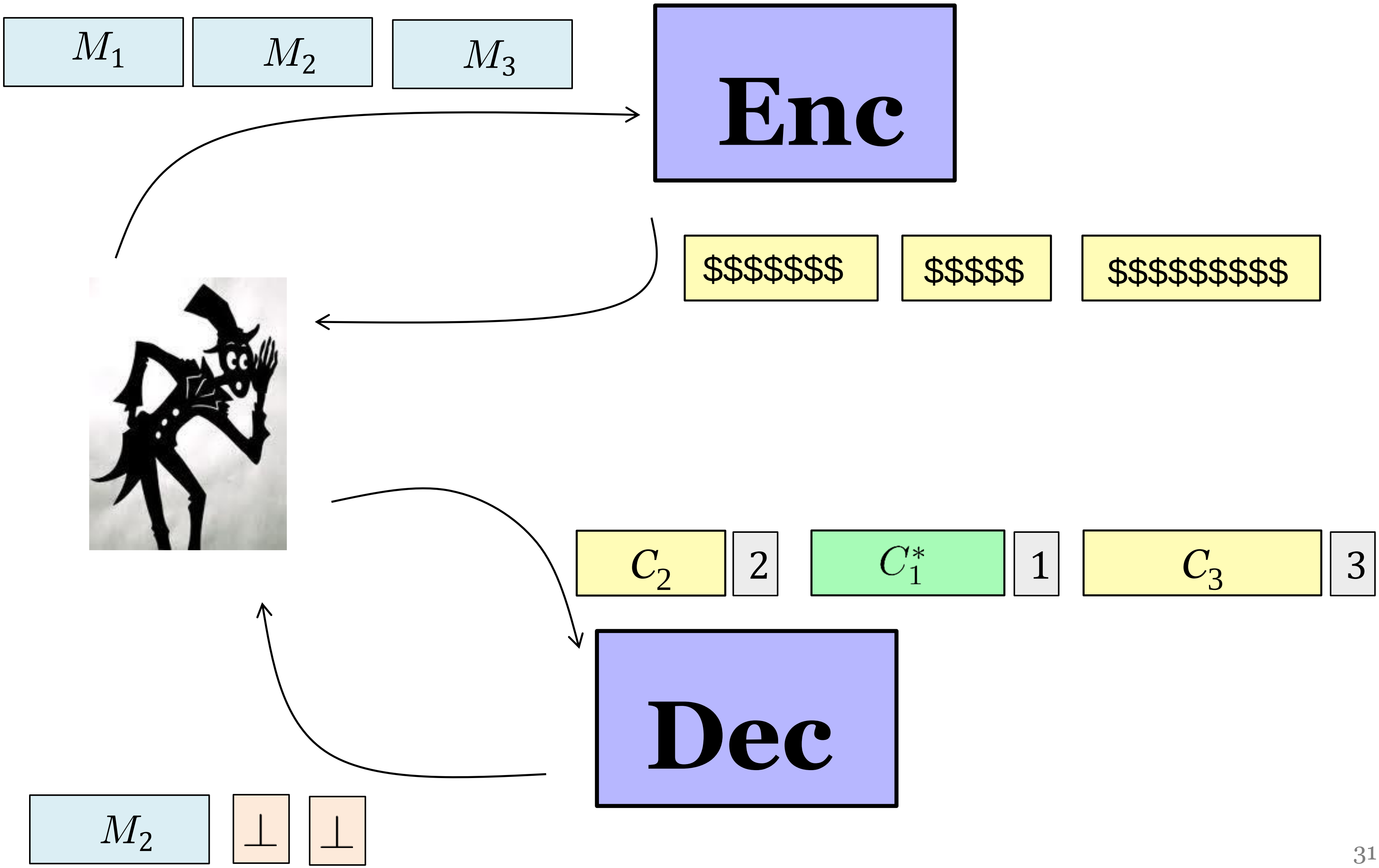
What applications actually demand



Major app: Encrypt huge files

Want: Random-access decryption

How The Model Looks Like (Very Informally)



Agenda

1. Nonced-based AE with Associated Data

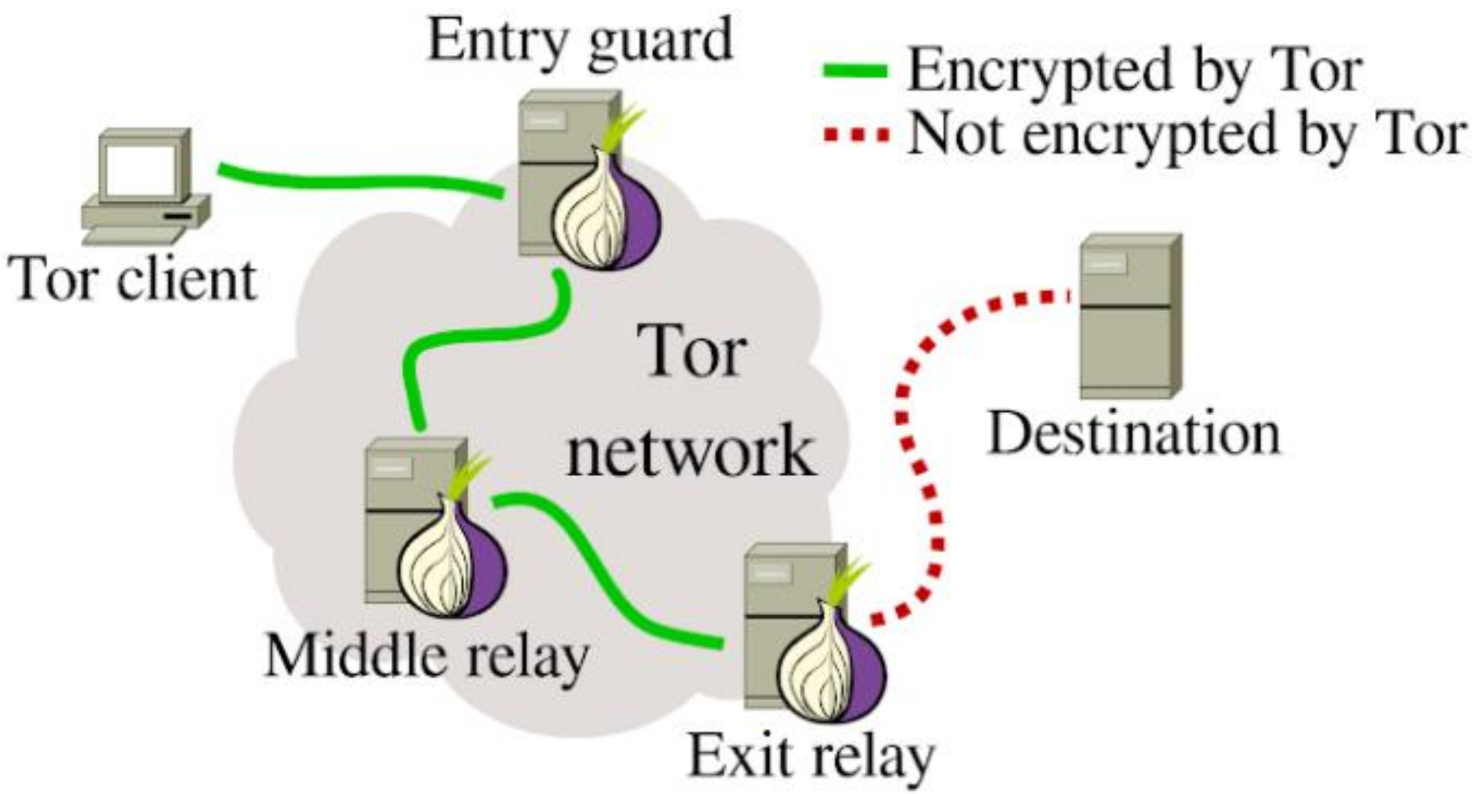
2. SSH Encryption

3. Streaming Encryption

4. **Onion Encryption and Tagging Attack**

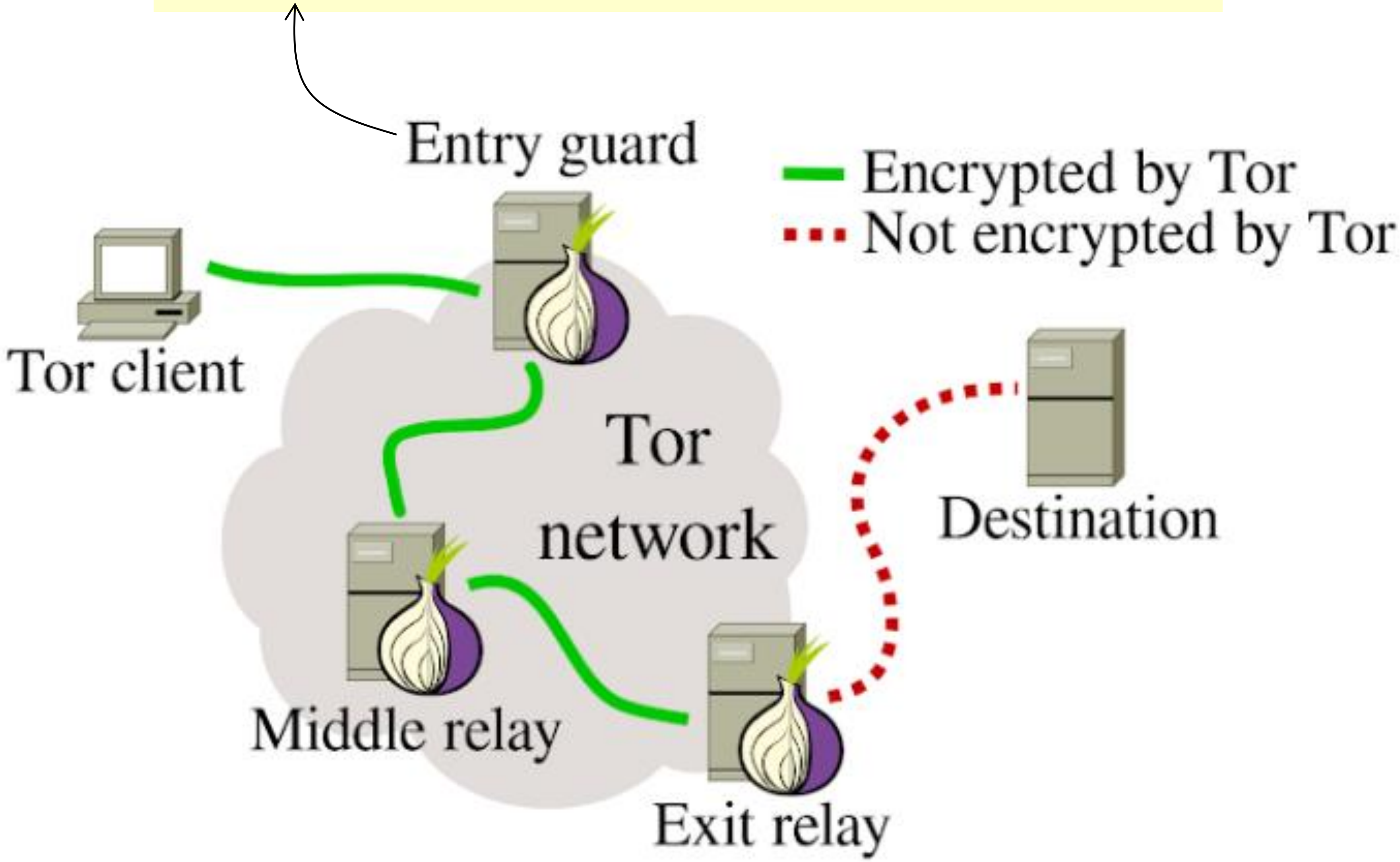
Recap: Tor (“The Onion Router”)

Tor operates by tunnelling traffic through three **random** “onion routers”

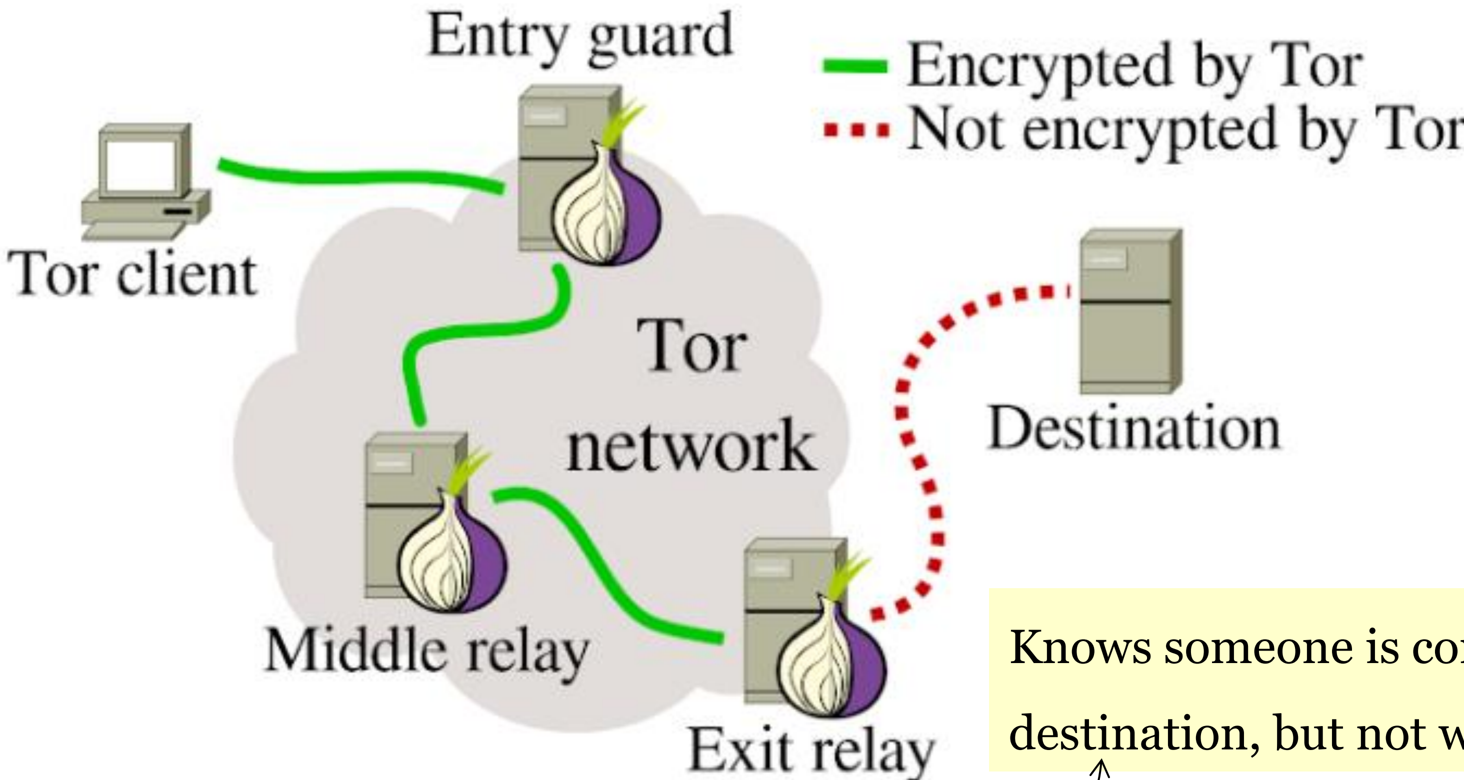


Who Knows What

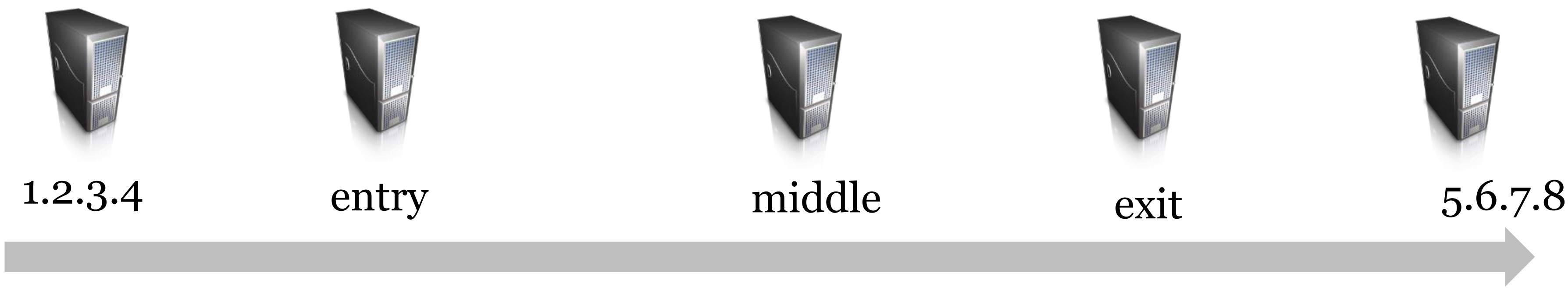
Knows Alice is using Tor and the identity of the middle node, but not the destination



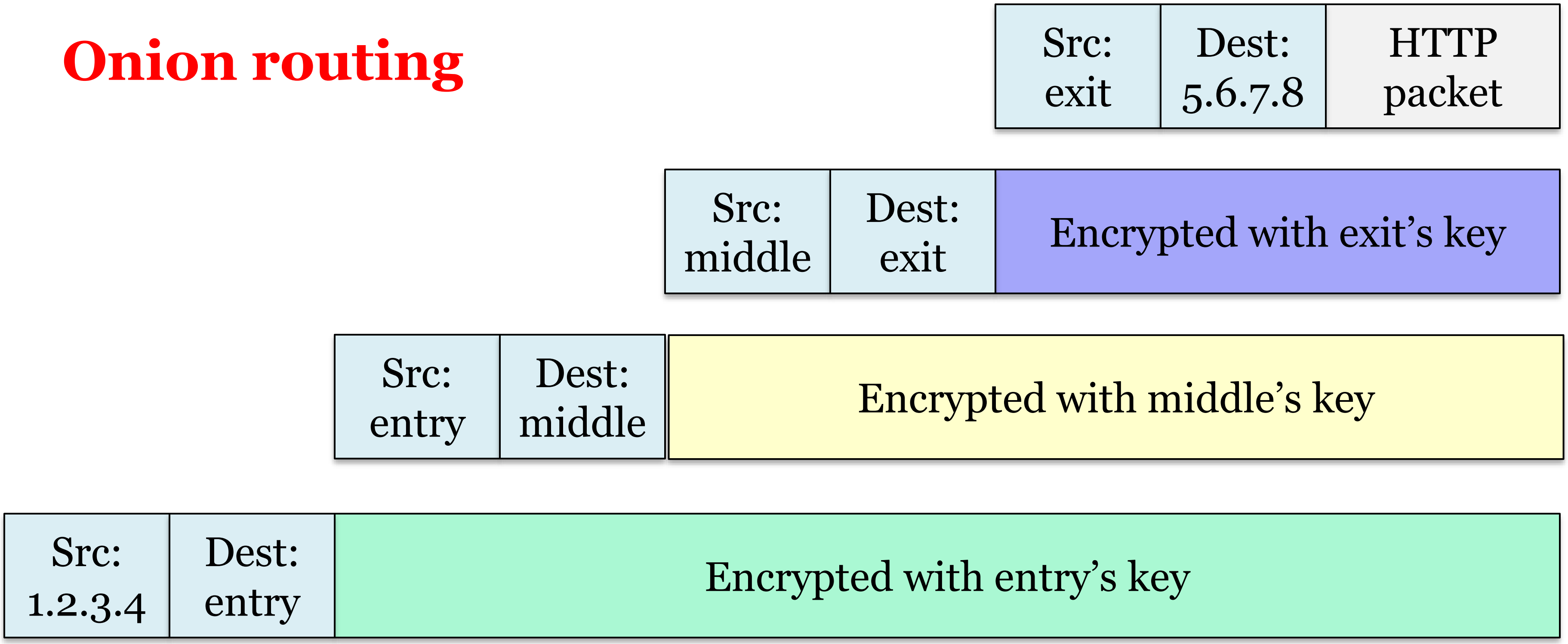
Who Knows What

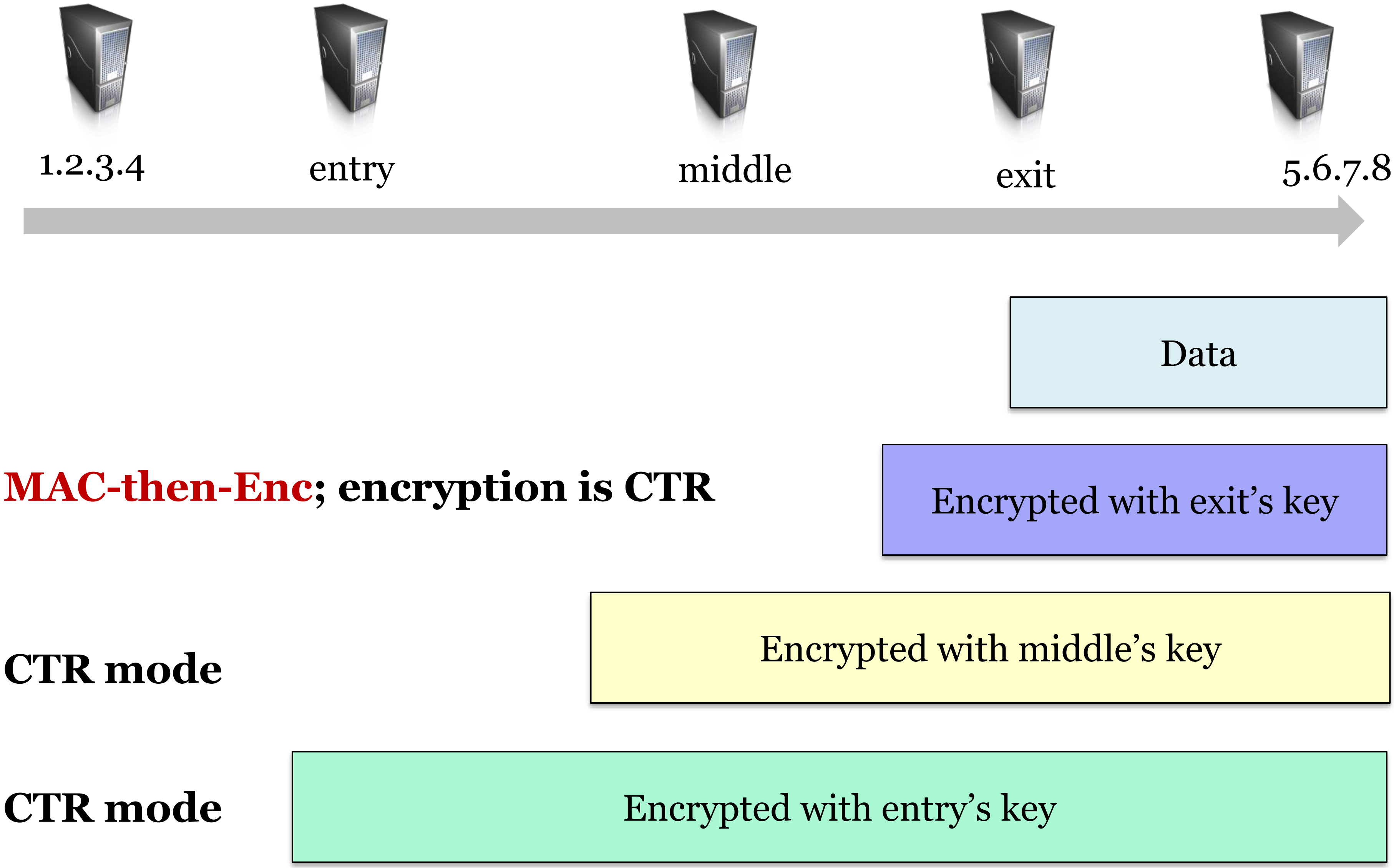


Knows someone is connecting to destination, but not which user



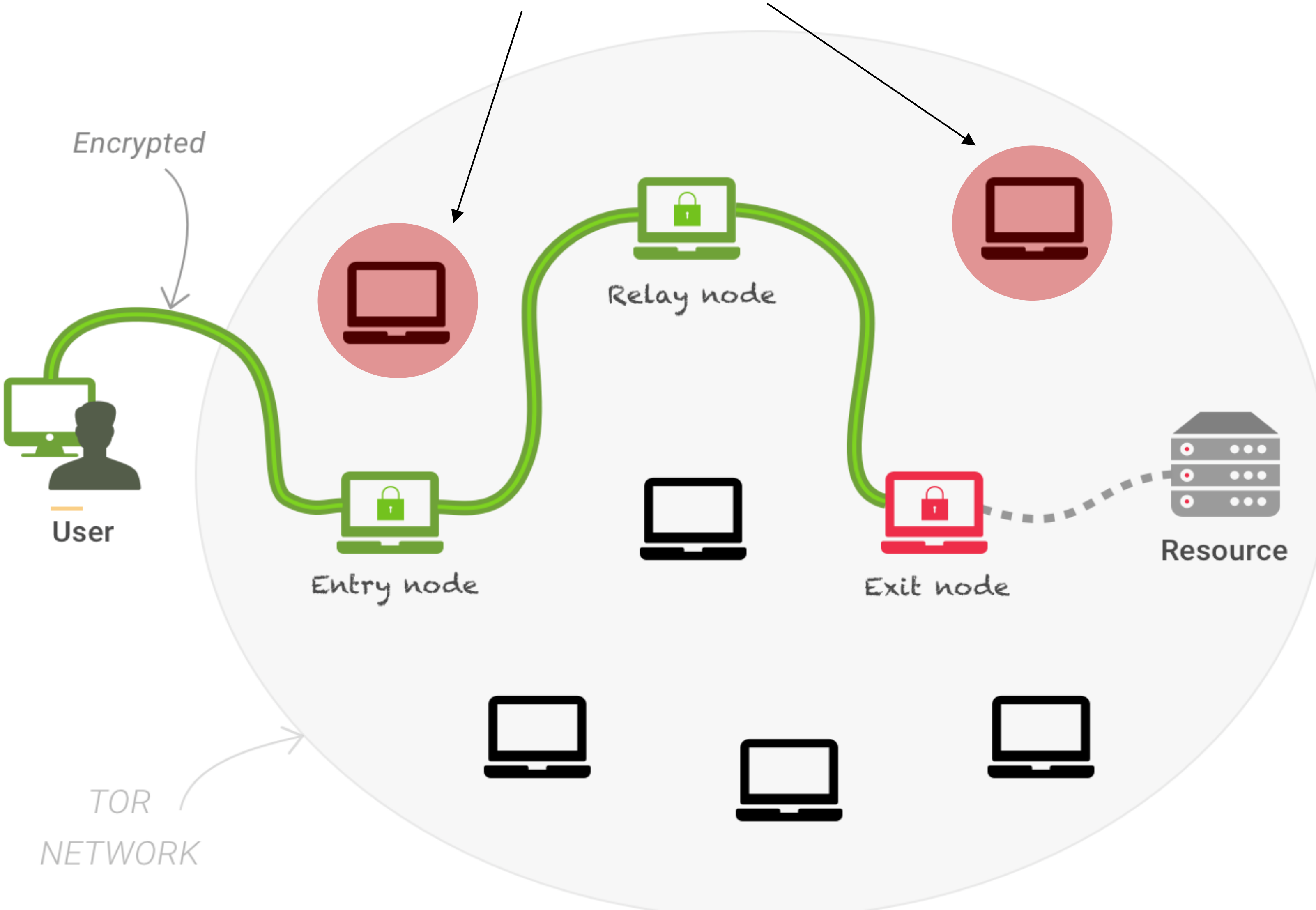
Onion routing




1.2.3.4
entry
middle
exit
5.6.7.8
Data
MAC-then-Enc; encryption is CTR
Encrypted with exit's key
CTR mode
Encrypted with middle's key
CTR mode
Encrypted with entry's key

Tagging Attack

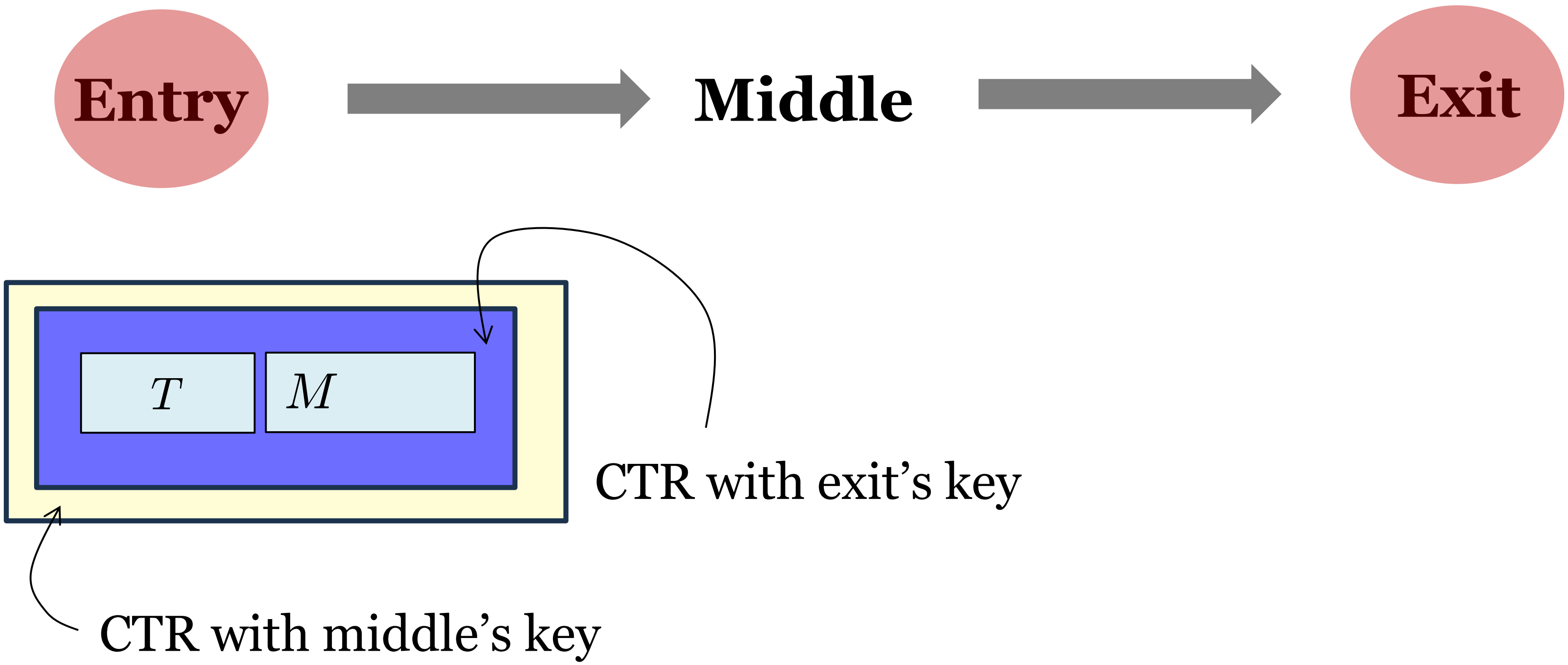
Malicious routers want to identify what service user U is using



Tagging Attack

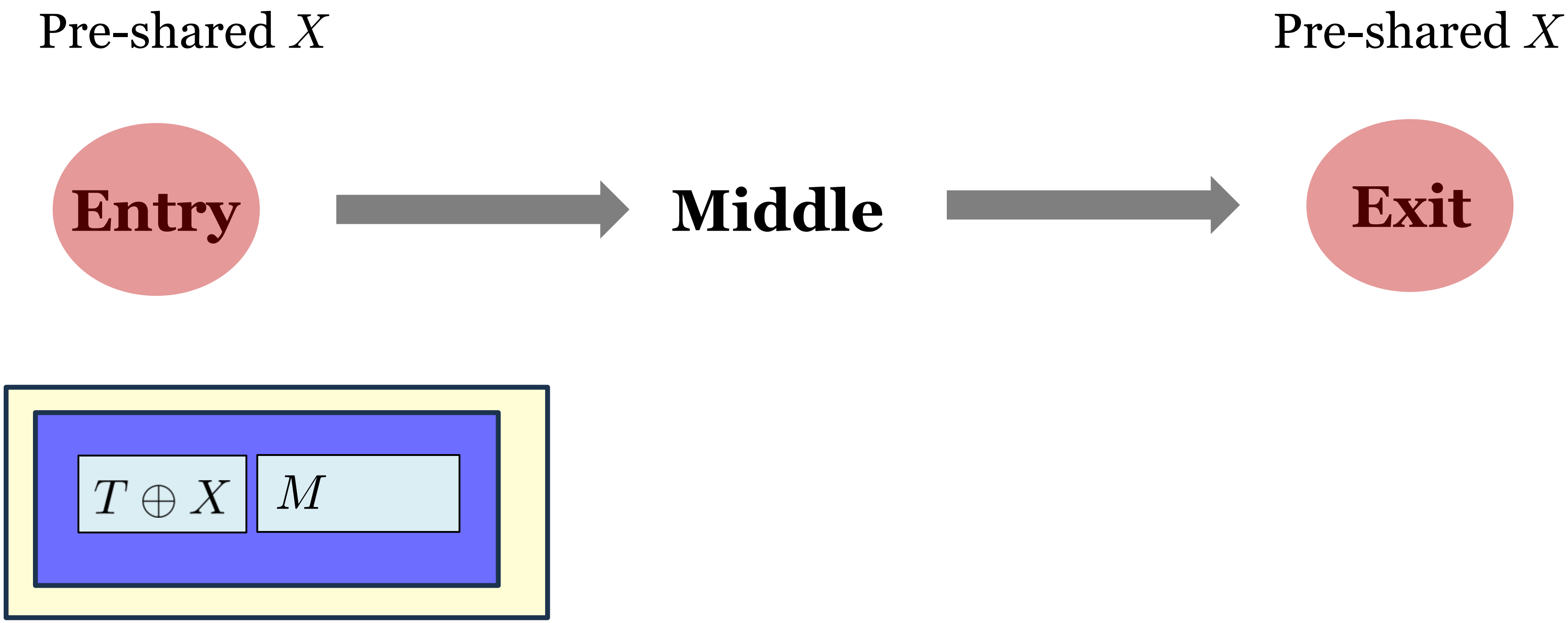
Suppose malicious nodes are chosen to be entry and exit

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

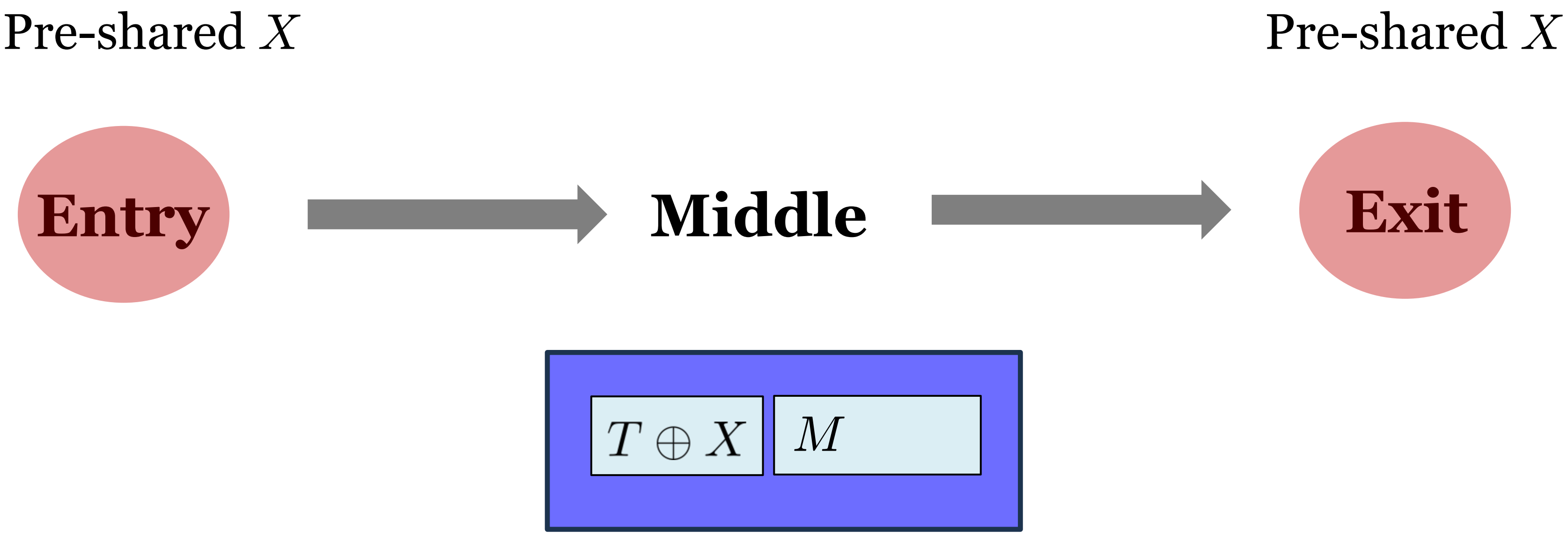


Tagging Attack

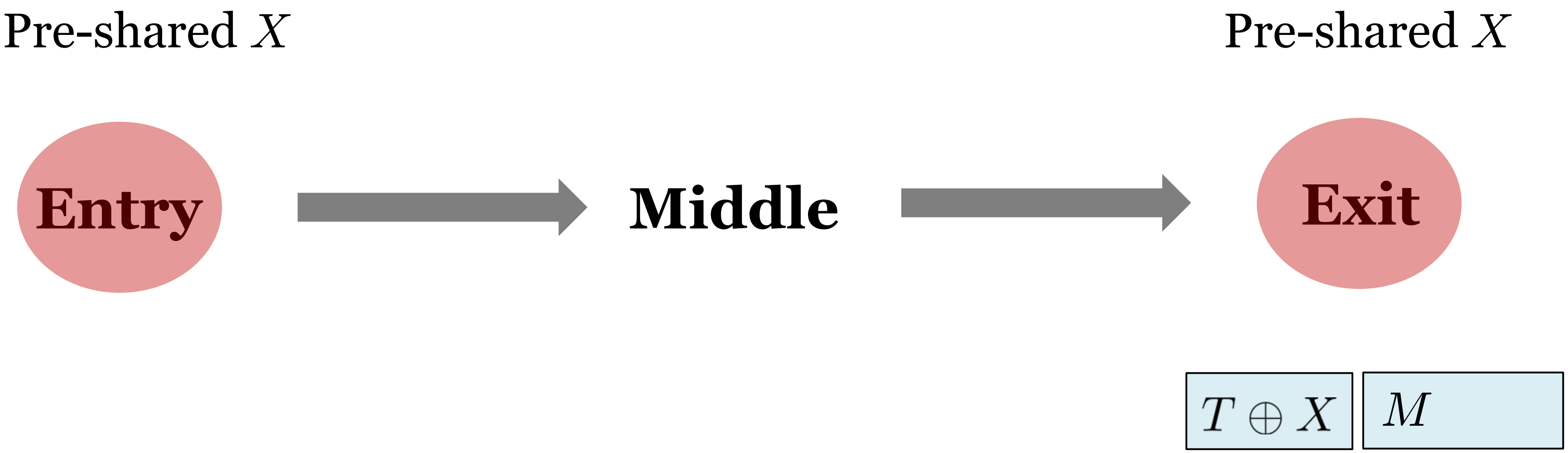
CTR is **malleable**: XOR X to ciphertext $\longrightarrow$ XOR X to data



Tagging Attack



Tagging Attack

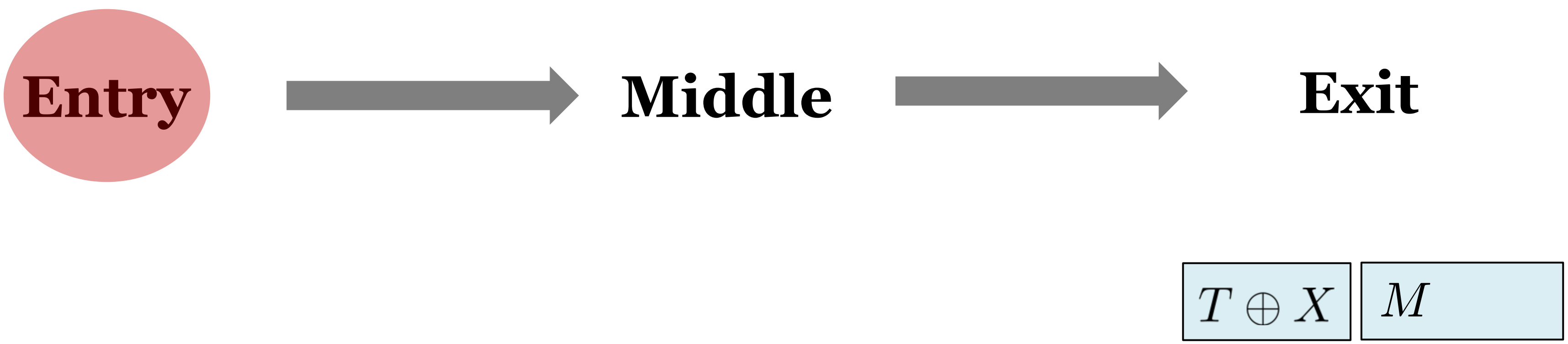


- MAC checking fails if use given tag
- Pass if xor X to the tag

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