
Lecture 5 – Authenticated encryption

TEK4500

22.09.2021

Håkon Jacobsen

hakon.jacobsen@its.uio.no

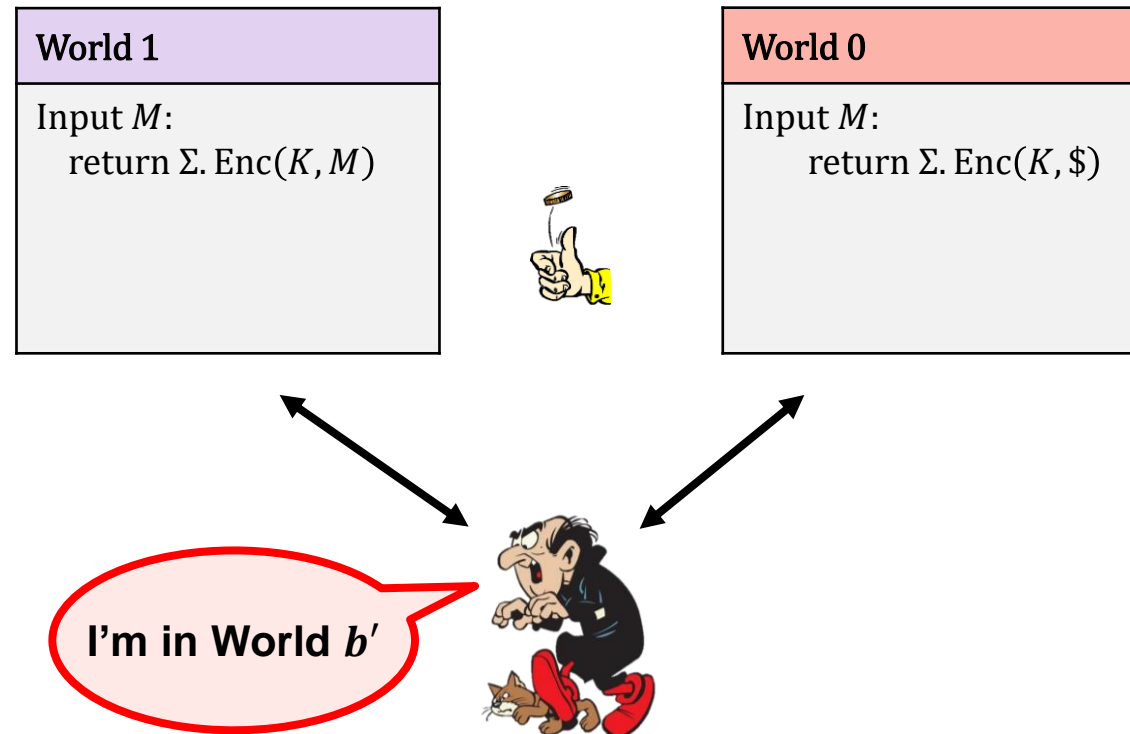
Midterm exam

- Available next week (Wednesday 29. September, 14:00)
- Due: **two weeks** later (Wednesday 13. October, 23:59)
- **Take-home** exam
- **Individual:** corporation is *not* allowed
- **Mandatory:** need to pass in order to be eligible for the exam
- All sources allowed (save for explicitly searching for the solution)
- Submission: Canvas (file type = PDF; strongly prefer if you use provided LaTeX template)
- Start early! The assignment may be more challenging than you expect

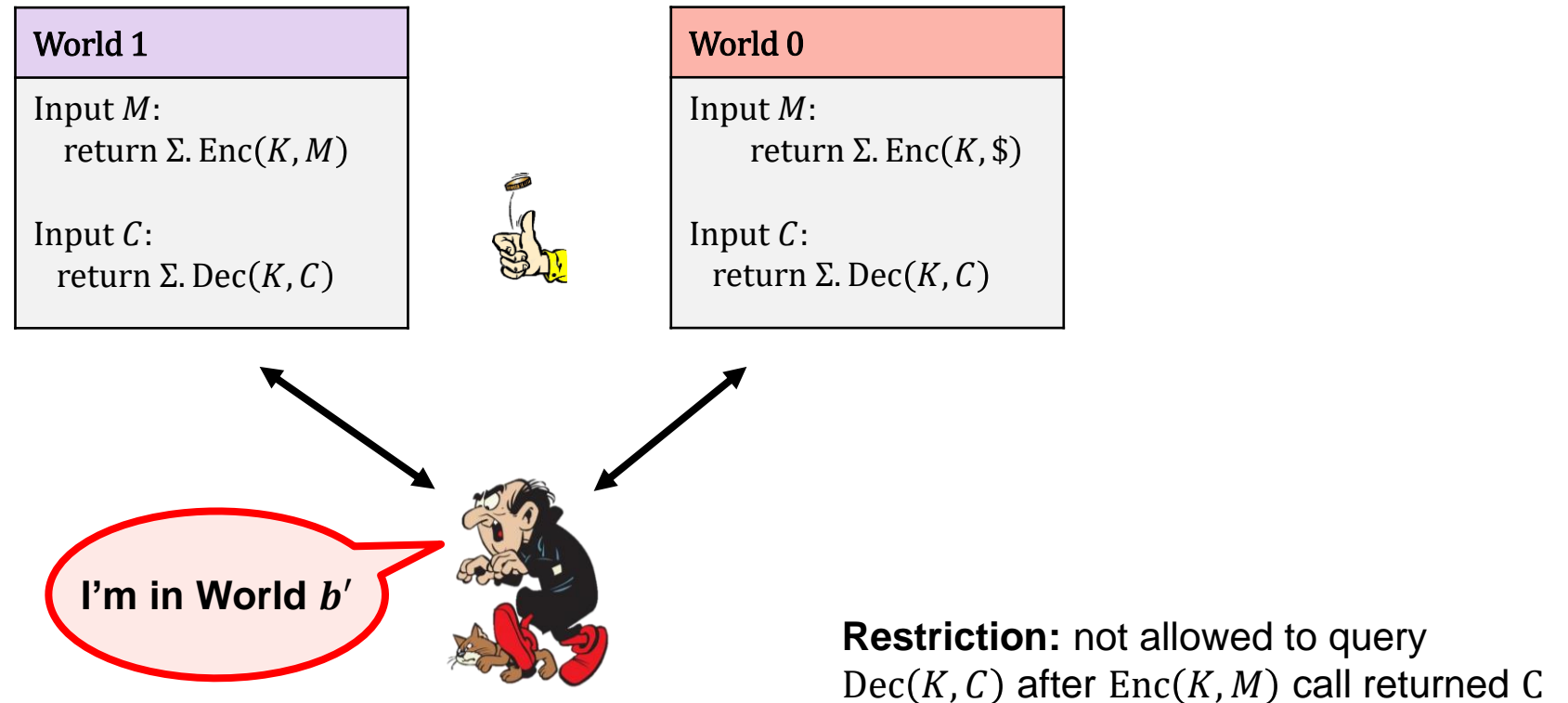
Basic goals of cryptography

	Message privacy	Message integrity / authentication
Symmetric keys	Symmetric encryption	Message authentication codes (MAC)
Asymmetric keys	Asymmetric encryption (a.k.a. public-key encryption)	Digital signatures

IND-CPA – Indistinguishability against chosen-plaintext attacks



IND-CCA – Indistinguishability against chosen-ciphertext attacks



Xilinx FPGA – Starbleed attack


Security Tools and Resources Privacy Policy FAQ's


ROOTDAEMON.COM // N

SECURITY NEWS

Unpatchable 'Starbleed' Exposes Critical Design Flaw

BY ROOTDAEMON © APRIL 21, 2020

+ 



A hand is shown holding a circuit board with a red glow. The word 'Starbleed' is written in a stylized font on the board. A 'DIGILENT' logo is also visible.

A newly discovered unpatchable hardware vulnerability could allow an attacker to break bitstream encryption on Xilinx 7-Series FPGAs.

The Unpatchable Silicon: A Full Break of the Bitstream Encryption of Xilinx 7-Series FPGAs

Maik Ender*, Amir Moradi* and Christof Paar*†

*Horst Goertz Institute for IT Security, Ruhr University Bochum, Germany

†Max Planck Institute for Cyber Security and Privacy, Germany

Abstract

The security of FPGAs is a crucial topic, as any vulnerability within the hardware can have severe consequences, if they are used in a secure design. Since FPGA designs are encoded in a bitstream, securing the bitstream is of the utmost importance. Adversaries have many motivations to recover and manipulate the bitstream, including design cloning, IP theft, manipulation of the design, or design subversions e.g., through hardware Trojans. Given that FPGAs are often part of cyber-physical systems e.g., in aviation, medical, or industrial devices, this can even lead to physical harm. Consequently, vendors have introduced bitstream encryption, offering authenticity and confidentiality. Even though attacks against bitstream encryption have been proposed in the past, e.g., side-channel analysis and probing, these attacks require sophisticated equipment and considerable technical expertise.

In this paper, we introduce novel low-cost attacks against the Xilinx 7-Series (and Virtex-6) bitstream encryption, resulting in the total loss of authenticity and confidentiality. We exploit a design flaw which piecewise leaks the decrypted bitstream. In the attack, the FPGA is used as a decryption oracle, while only access to a configuration interface is needed. The attack does not require any sophisticated tools and, depending on the target system, can potentially be launched remotely. In addition to the attacks, we discuss several countermeasures.

1 Introduction

Nowadays, Field Programmable Gate Arrays (FPGAs) are common in consumer electronic devices, aerospace, financial computing, and military applications. Additionally, given the trend towards a connected world, data-driven practices, and artificial intelligence, FPGAs play a significant role as hardware platforms deployed in the cloud and in end devices. Hence, trust in the underlying platform for all these applications is vital. Altera, who are (together with Xilinx) the FPGA market leader, was acquired by Intel in 2015.

FPGAs are reprogrammable ICs, containing a repetitive logic area with a few hundred up to millions of reprogrammable gates. The bitstream configures this logic area; in analogy to software, the bitstream can be considered the 'binary code' of the FPGA. On SRAM-based FPGAs, which are the dominant type of FPGA in use today, the bitstream is stored on an external non-volatile memory and loaded into the FPGA during power-up.

In order to protect the bitstream against malicious actors, its confidentiality and authenticity must be assured. If an attacker has access to the bitstream and breaks its confidentiality, he can reverse-engineer the design, clone intellectual property, or gather information for subsequent attacks e.g., by finding cryptographic keys or other design aspects of a system. If the adversary succeeds in violating the bitstream authenticity, he can then change the functionality, implant hardware Trojans, or even physically destroy the system in which the FPGA is embedded by using configuration outside the specifications. These problems are particularly relevant since access to bitstream is often effortlessly possible due to the fact that, for the vast majority of devices, it resides in the external non-volatile memory, e.g., flash chips. This memory can often either be read out directly, or the adversary wiretaps the FPGA's configuration bus during power-up. Alternatively, a microcontroller can be used to configure the FPGA, and consequently, the microcontroller's firmware includes the bitstream. When the adversary gains access to the microcontroller, he also gains access to the configuration interface and the bitstream. Thus, if the microcontroller is connected to a network, remotely attacking the FPGA becomes possible.

In order to protect the design, the major FPGA vendors introduced bitstream encryption around the turn of the millennium, a technique which nowadays is available in most mainstream devices [1,56]. In this paper, we investigate the security of the Xilinx 7-Series and Virtex-6 bitstream encryption. On these devices, the bitstream encryption provides authenticity by using an SHA-256 based HMAC and also provides confidentiality by using CBC-AES-256 for encryption. By our attack, we can circumvent the bitstream encryption and decrypt an assumedly secure bitstream on all Xilinx 7-Series devices completely and on the Virtex-6 devices partially. Adversaries can then change the functionality, implant hardware Trojans, or even physically destroy the system in which the FPGA is embedded by using configuration outside the specifications.

Nowadays, Field Programmable Gate Arrays (FPGAs) are common in consumer electronic devices, aerospace, financial computing, and military applications. Additionally, given the trend towards a connected world, data-driven practices, and artificial intelligence, FPGAs play a significant role as hardware platforms deployed in the cloud and in end devices. Hence, trust in the underlying platform for all these applications is vital. Altera, who are (together with Xilinx) the FPGA market leader, was acquired by Intel in 2015.

NEWS HACKING TOOLS CTF SHOP COURSES AFFILIATES



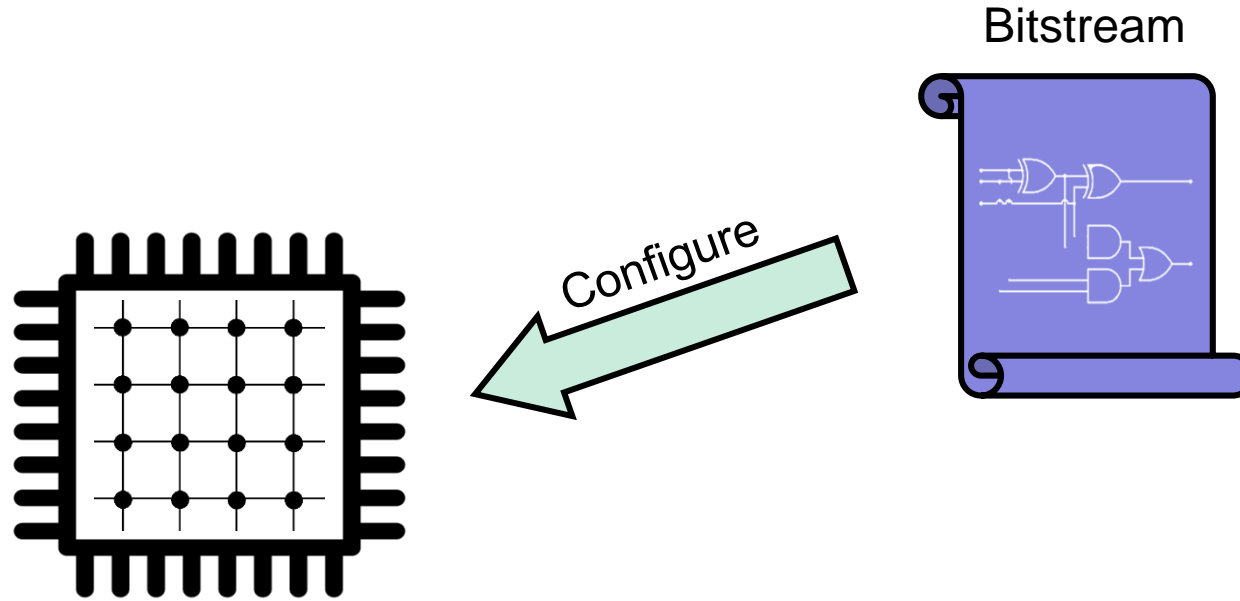
News Vulnerabilities

Vulnerability Discovered In FPGA

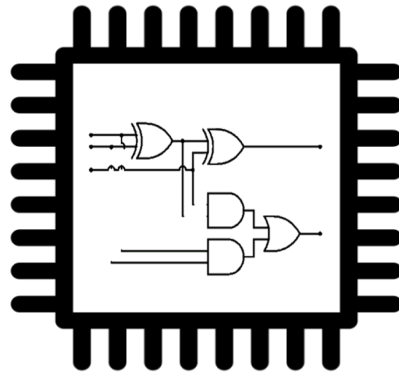
bitstream, bitstream encryption, bug, Chips, decryption, decryption key,

Gate Arrays, flaw, FPGA chips, Hardware, hardware encryption, hardware

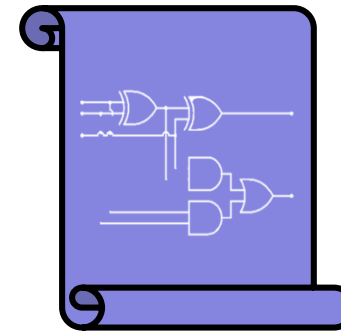
FPGA – Field Programmable Gate Array



FPGA – Field Programmable Gate Array

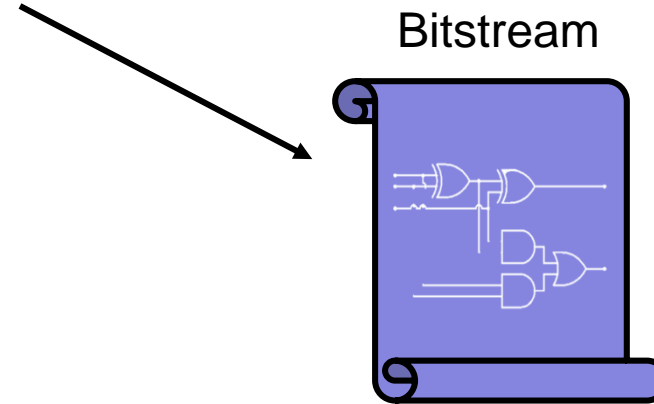
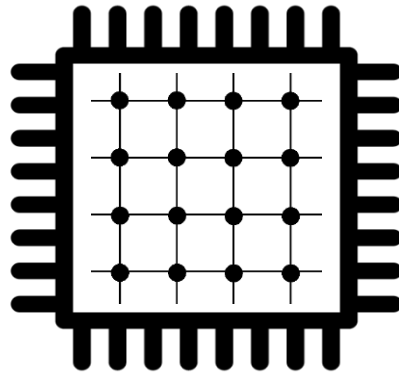


Bitstream



FPGA – Field Programmable Gate Array

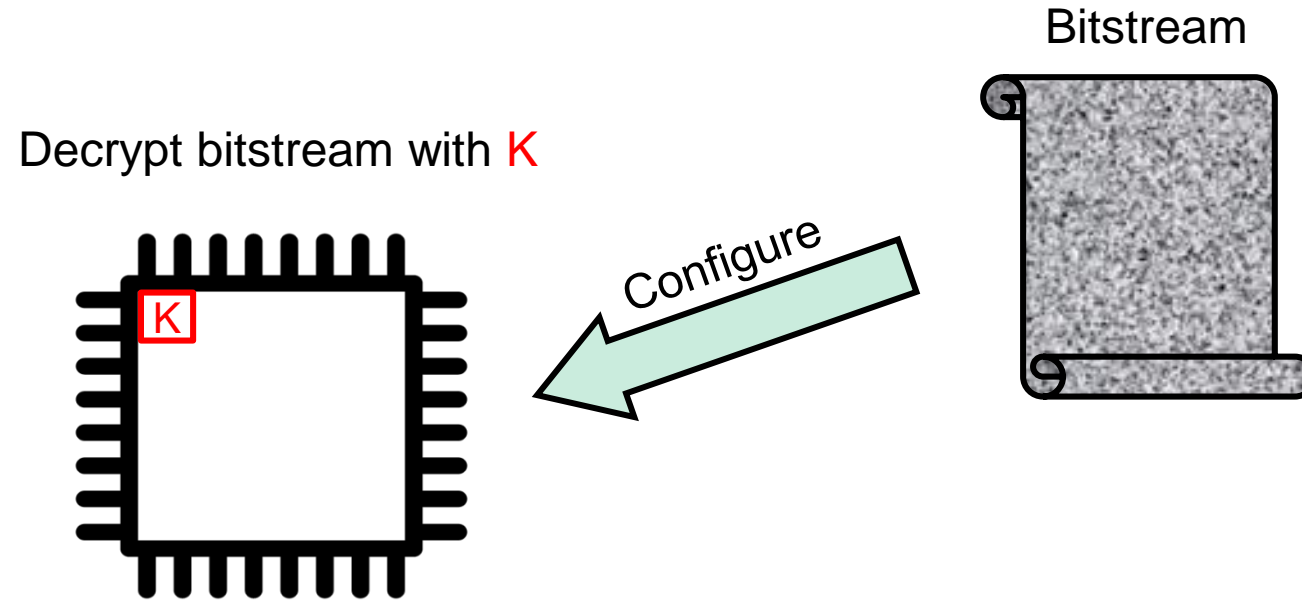
Bitstream design typically a business secret
(or even a national/military secret)



FPGA applications:

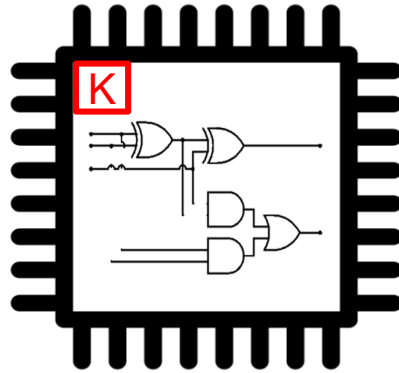
- Aerospace and avionics
- Digital signal processors
- Defense and military
- Medical devices
- General hardware accelerators (e.g. cryptography)

FPGA – Field Programmable Gate Array

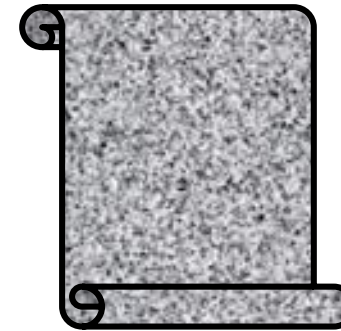


FPGA – Field Programmable Gate Array

Decrypt bitstream with **K**

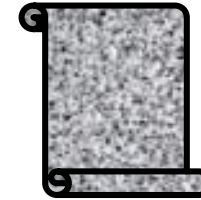


Bitstream

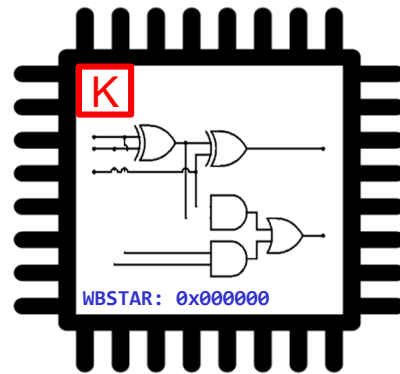


Xilinx Starbleed attack

Bitstream



Decrypt bitstream with **K**



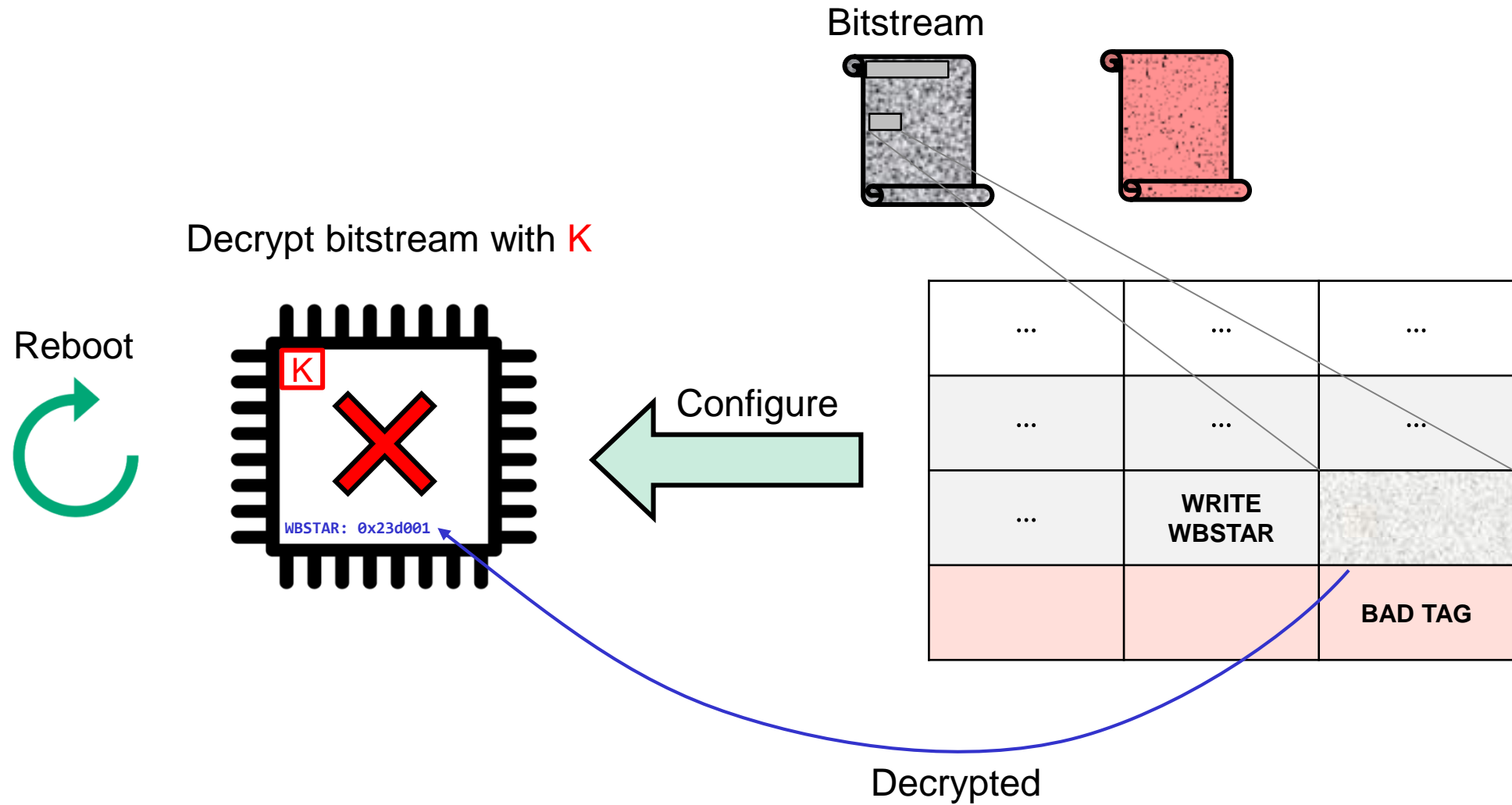
Header

AES-CBC
encrypted

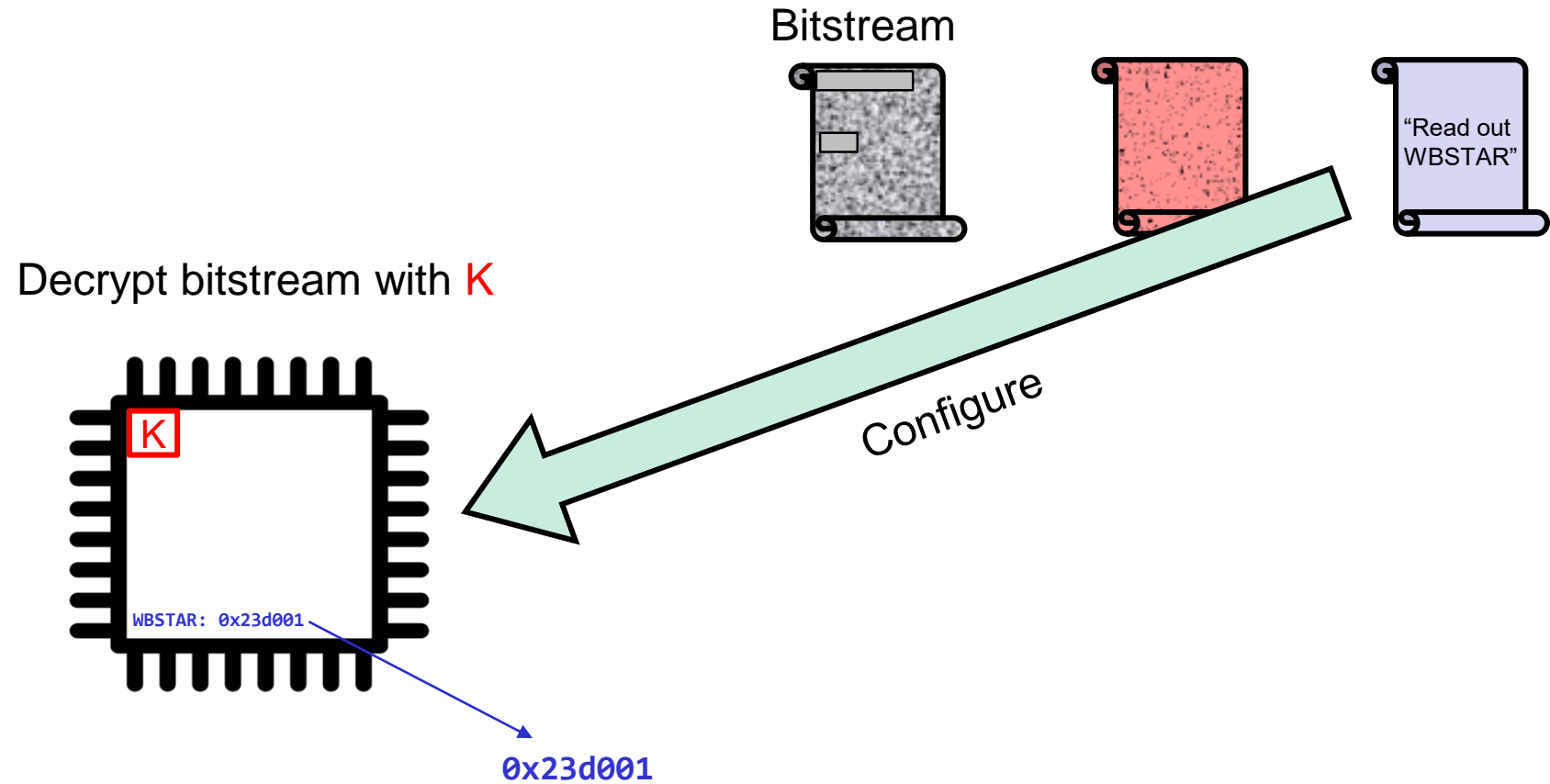
...
K_{MAC}
...	WRITE WBSTAR	0x00000000
		MAC TAG

WBSTAR = **W**arm-**B**oot **S**tart-address

Xilinx Starbleed attack

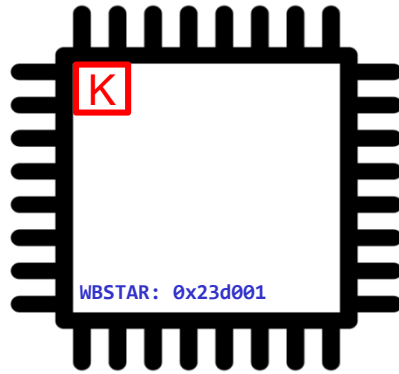


Xilinx Starbleed attack

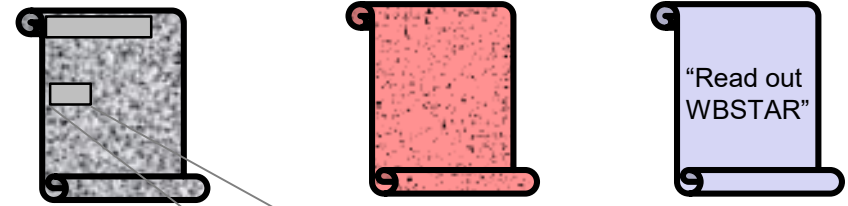


Xilinx Starbleed attack

Decrypt bitstream with **K**

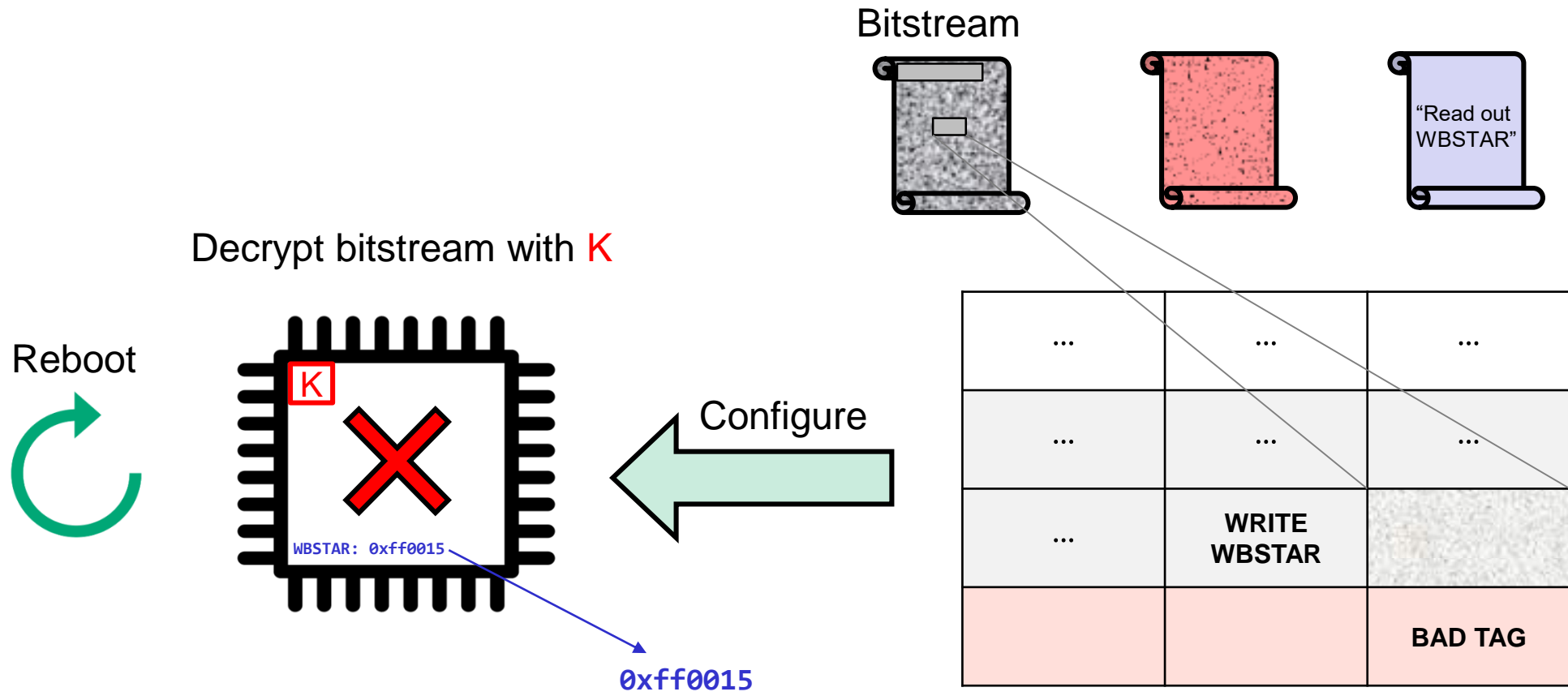


Bitstream

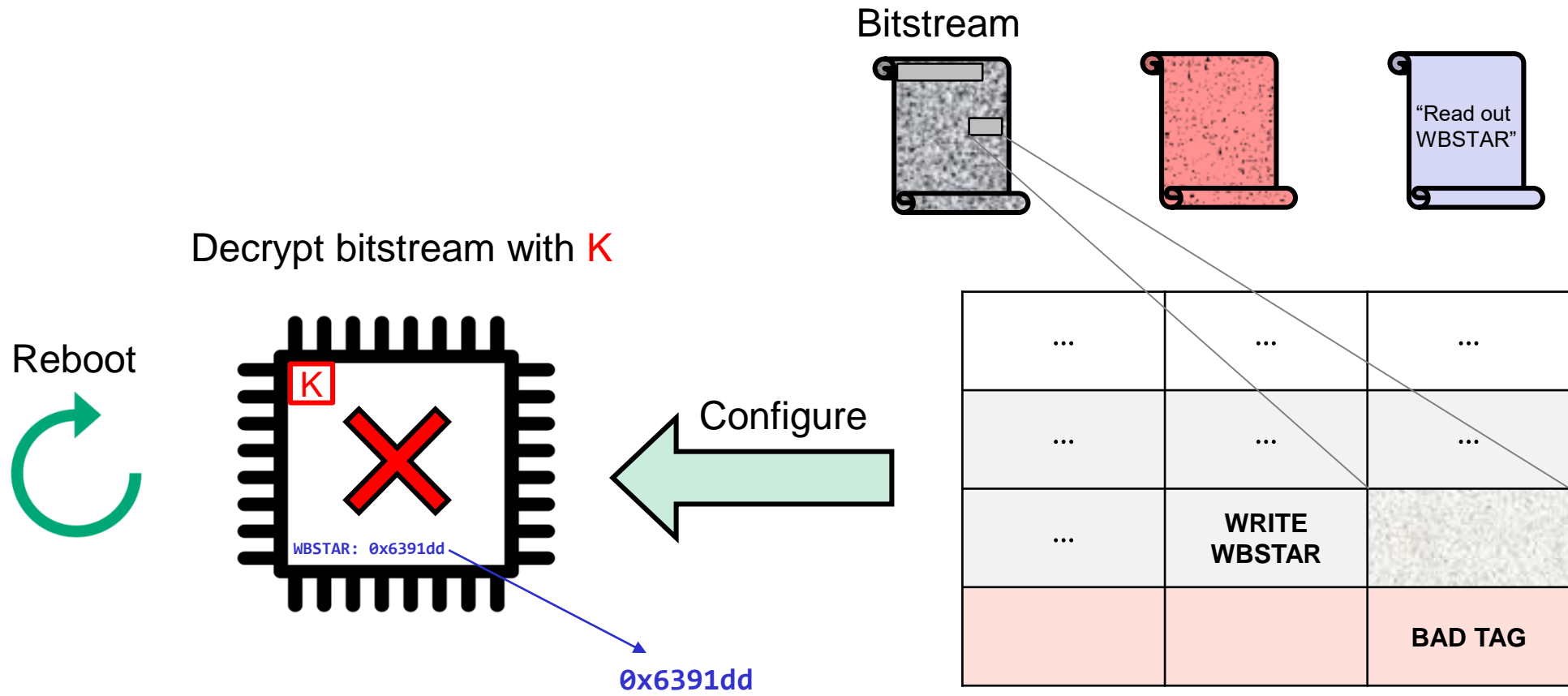


...
...
...	WRITE WBSTAR	...
		BAD TAG

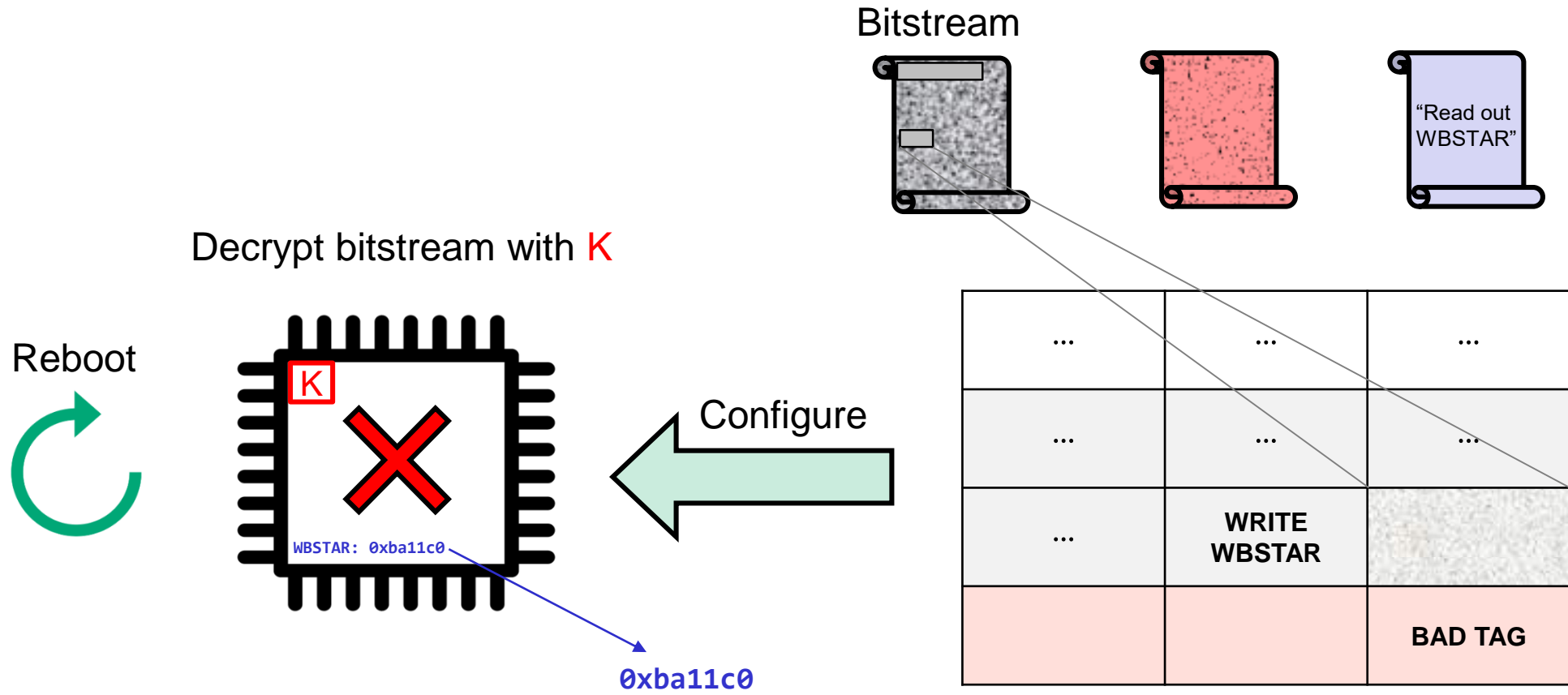
Xilinx Starbleed attack



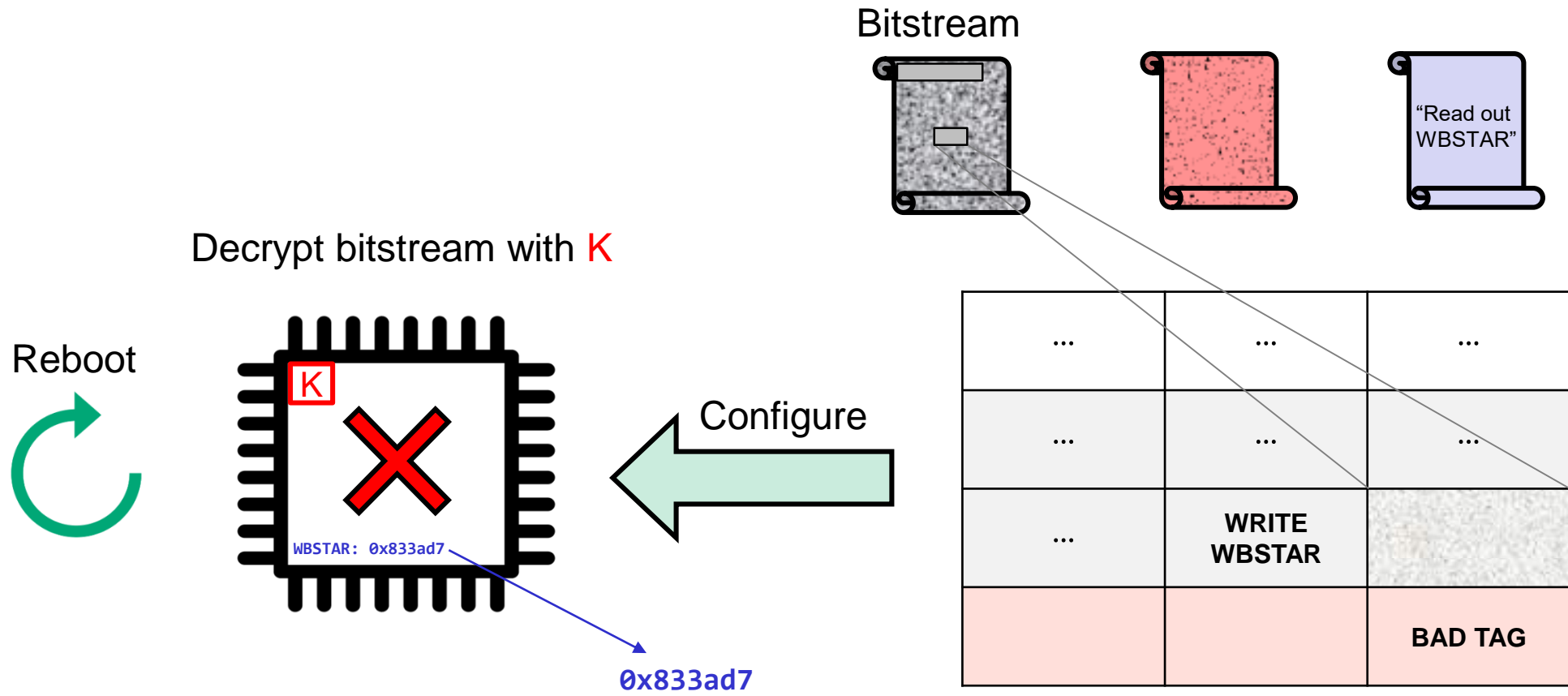
Xilinx Starbleed attack



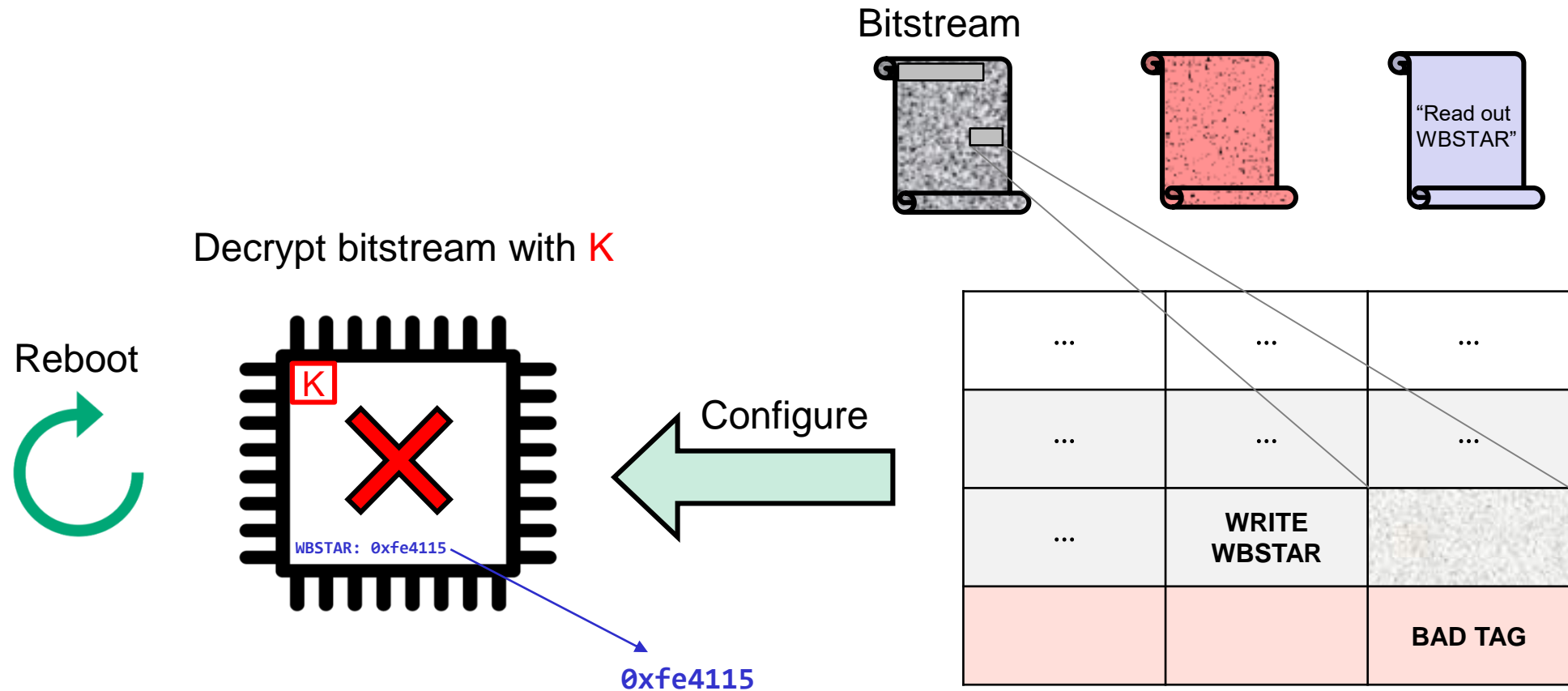
Xilinx Starbleed attack



Xilinx Starbleed attack



Xilinx Starbleed attack



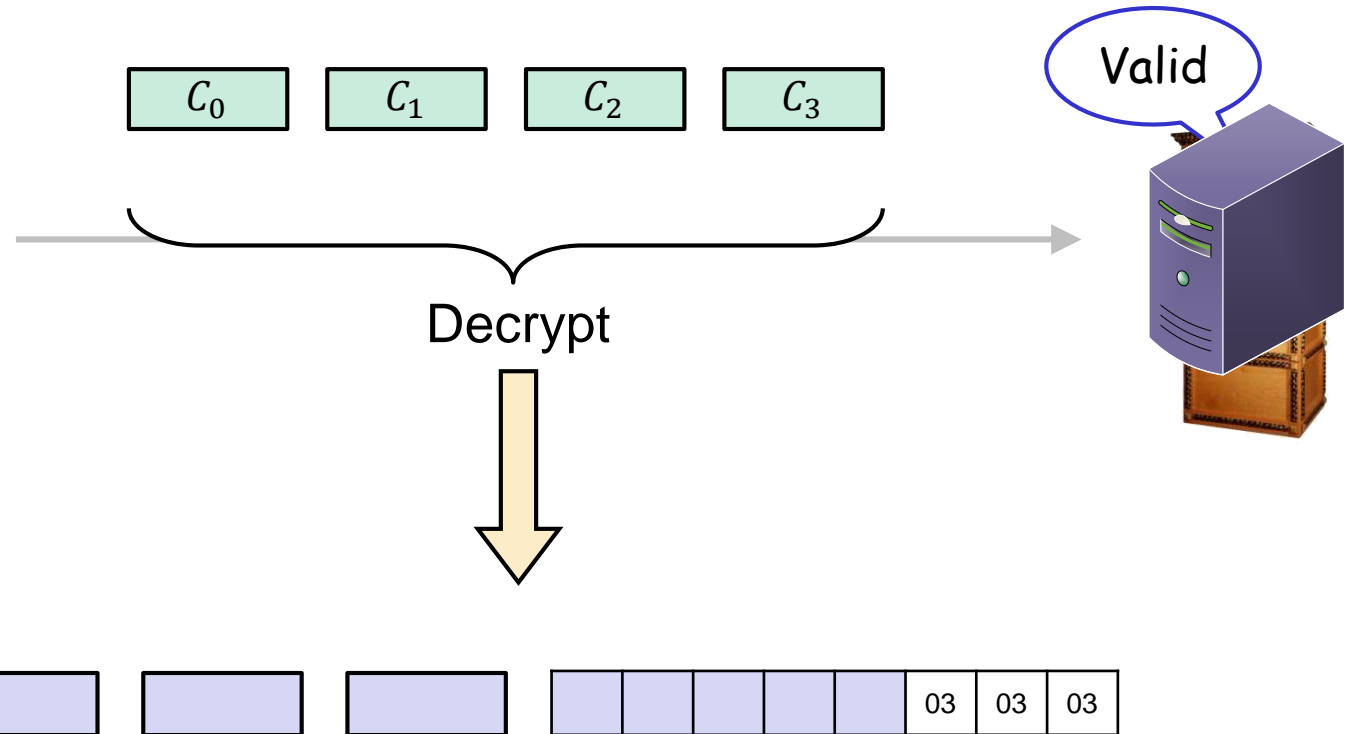
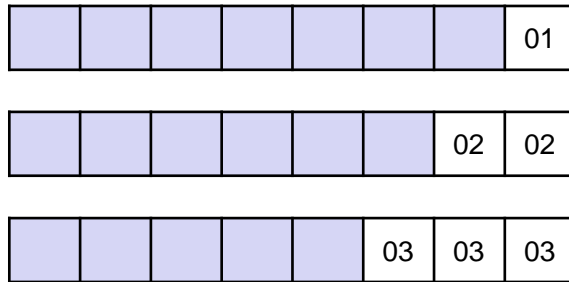
Time to fully decrypt 48 MB bitstream: 26 hours

Padding oracles

- Full decryption oracles are not always available/needed
- Padding attacks:
 - Chosen-ciphertext attacks that exploit small timing variations when decrypting different ciphertexts
 - Only a limited "decryption" oracle available to the attacker
 - First described by Serge Vaudenay (2002) based on CBC-encryption in TLS and IPsec

DTLS padding oracle attack

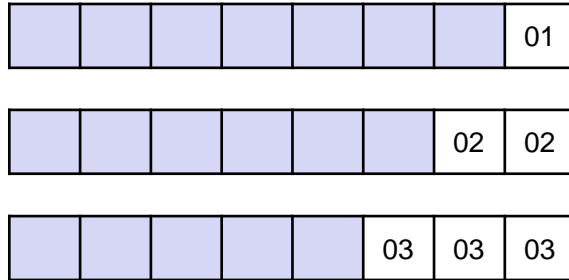
DTLS padding



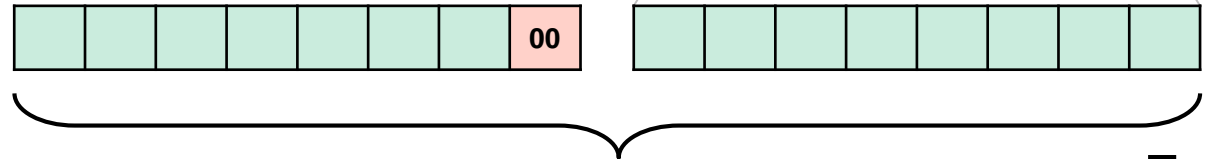
$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$

DTLS padding oracle attack

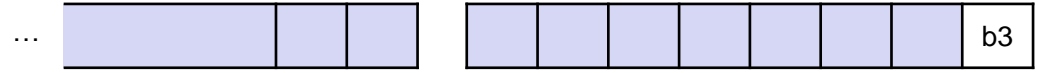
DTLS padding



Invalid

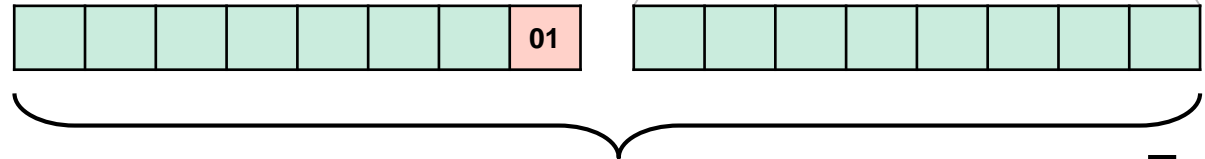
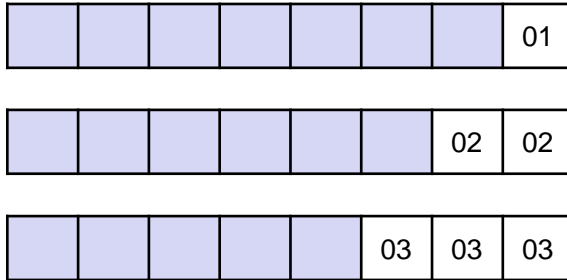


$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$



DTLS padding oracle attack

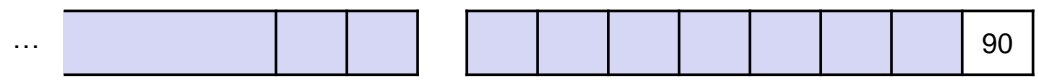
DTLS padding



Invalid

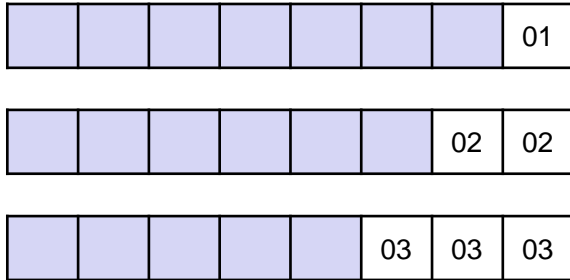


$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$

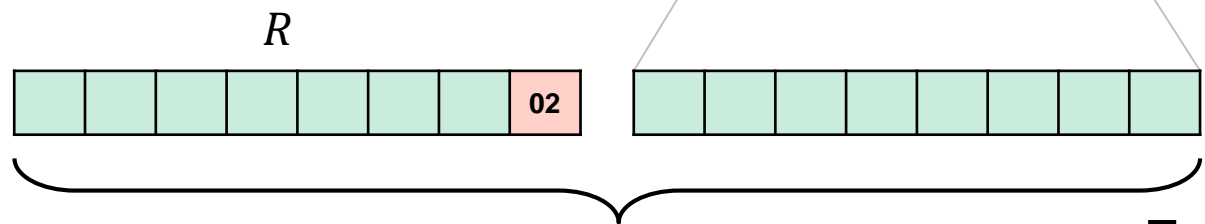


DTLS padding oracle attack

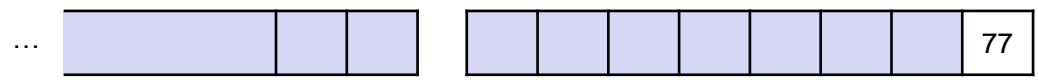
DTLS padding



Invalid

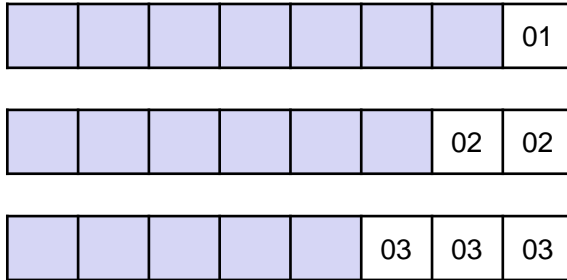


$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$

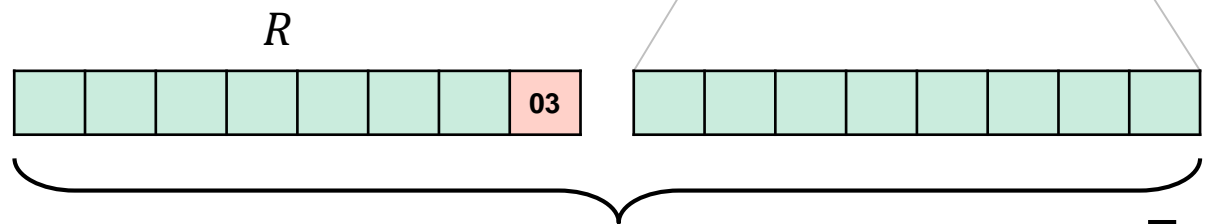


DTLS padding oracle attack

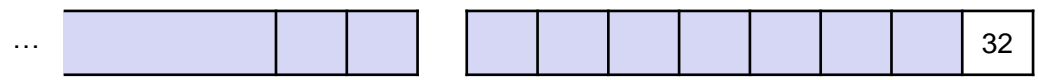
DTLS padding



Invalid

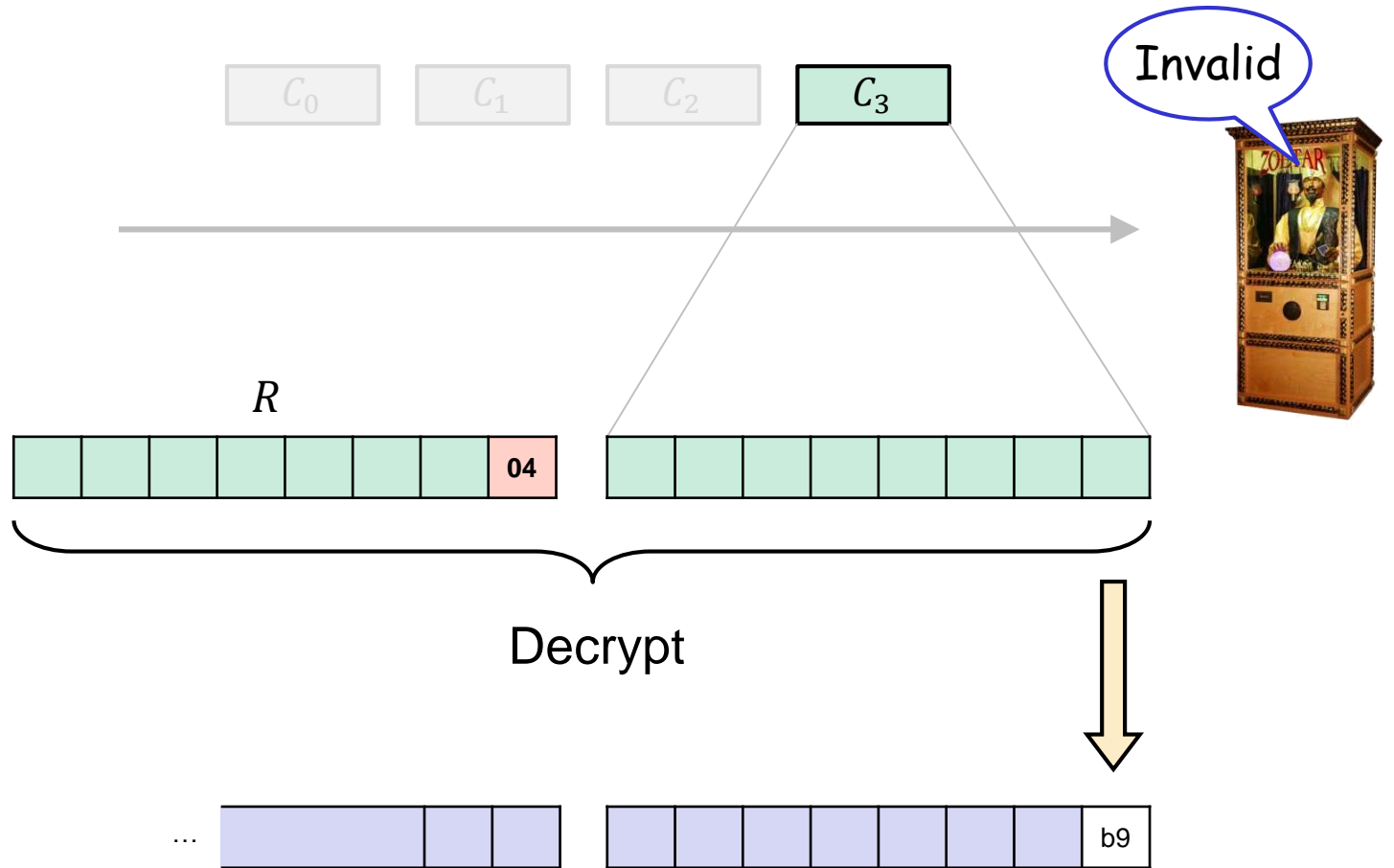
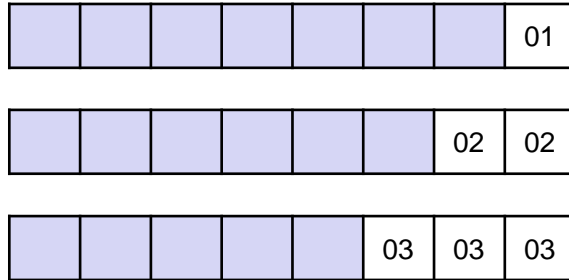


$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$



DTLS padding oracle attack

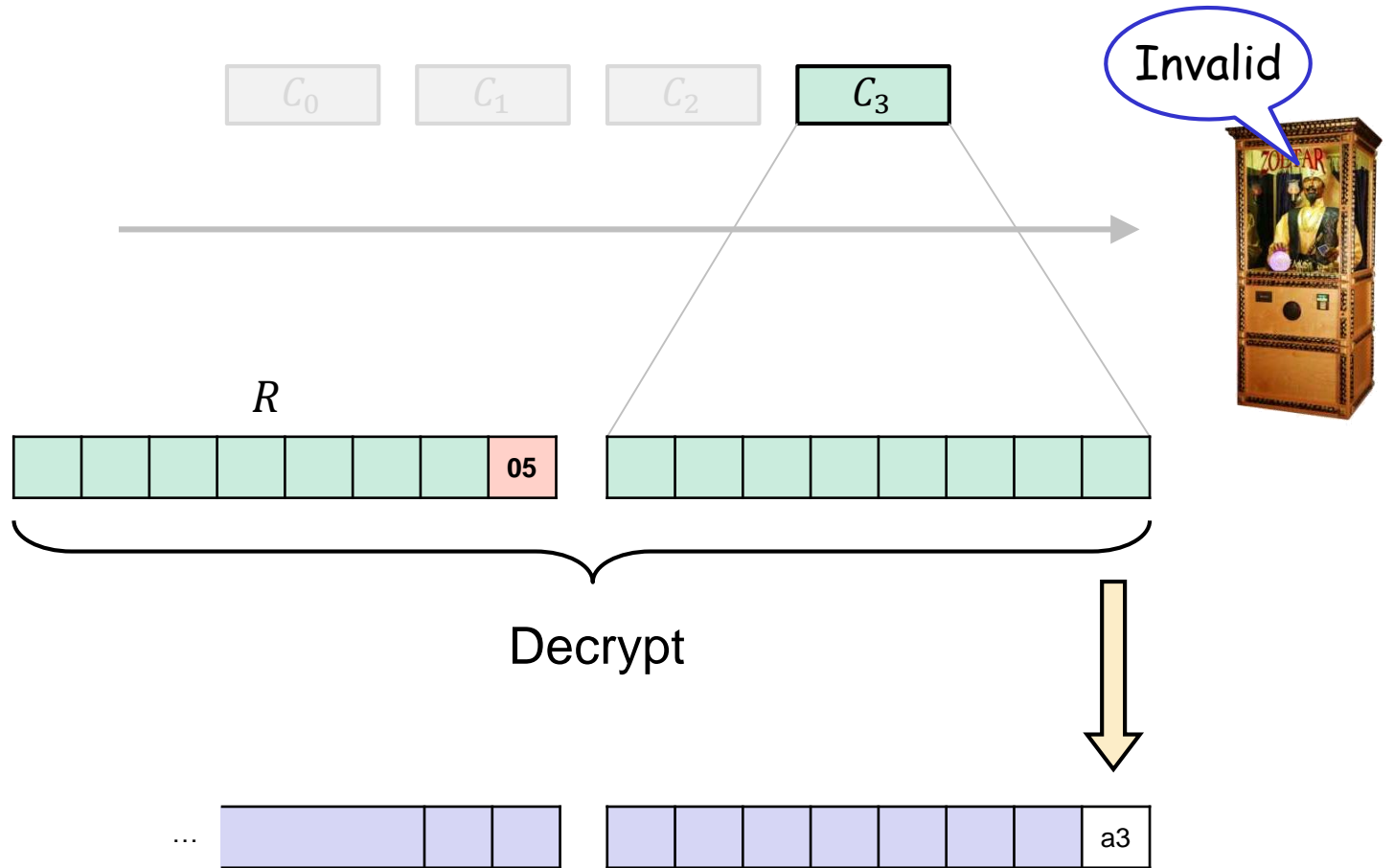
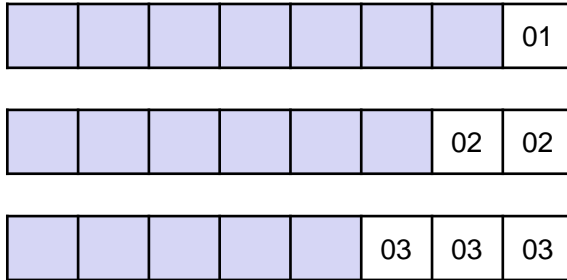
DTLS padding



$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$

DTLS padding oracle attack

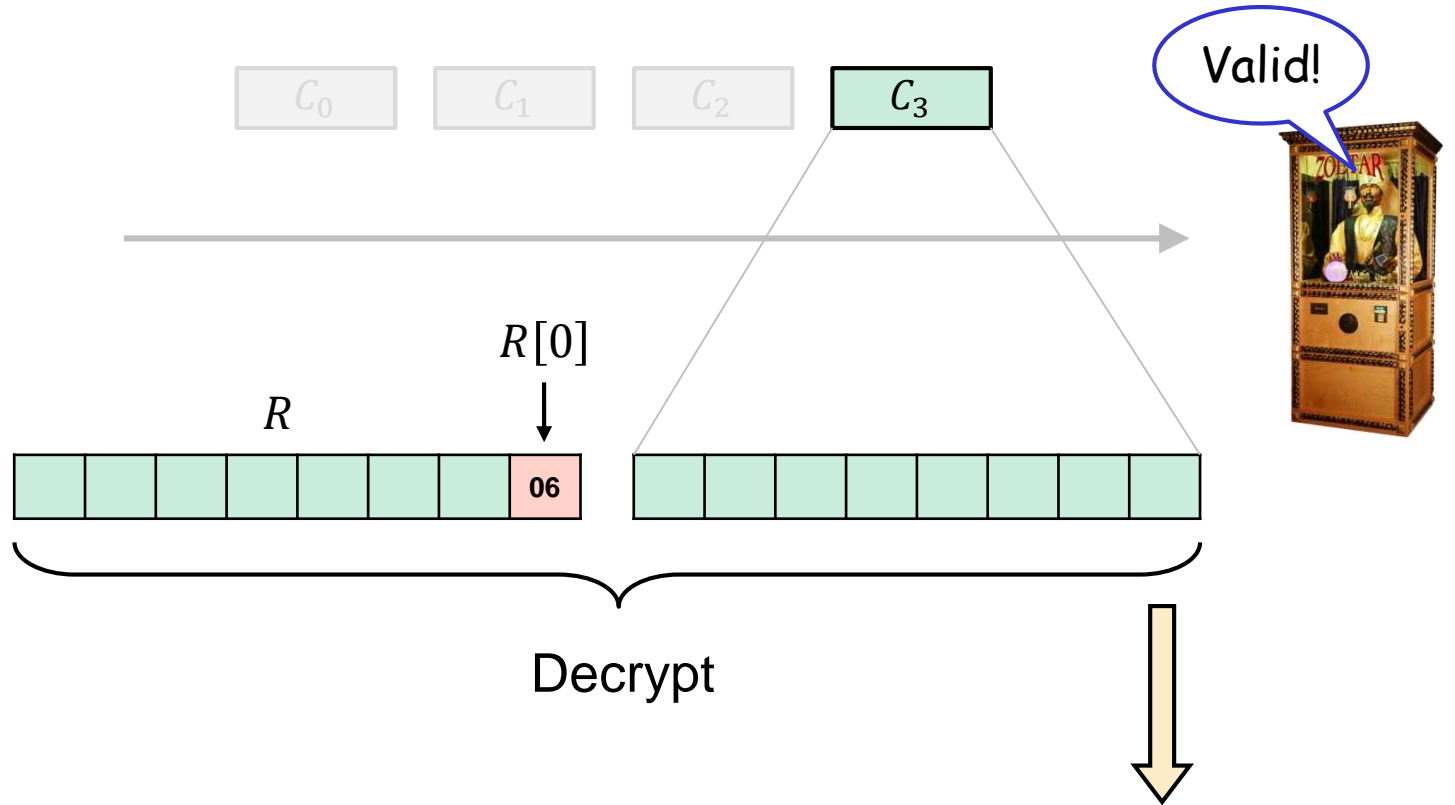
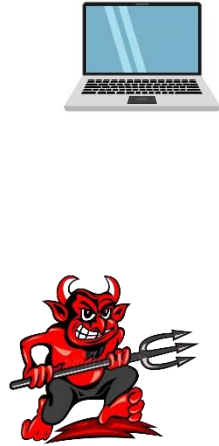
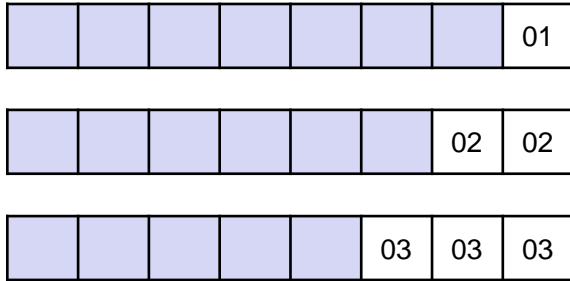
DTLS padding



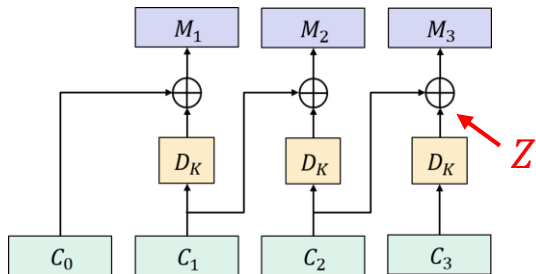
$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$

DTLS padding oracle attack

DTLS padding



$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$



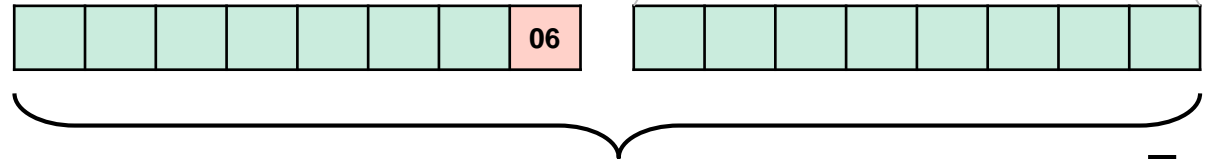
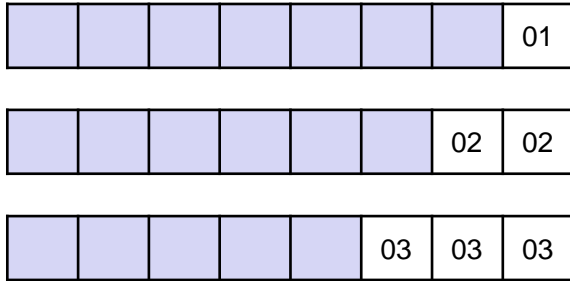
$$M_3[0] = R[0] \oplus C_2[0] \oplus 01$$

$$R[0] \oplus Z[0] = 01$$

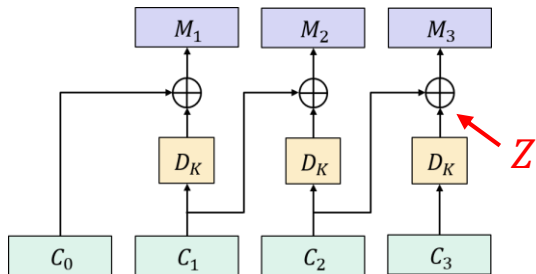
$$R[0] \oplus (C_2[0] \oplus M_3[0]) = 01$$

DTLS padding oracle attack

DTLS padding



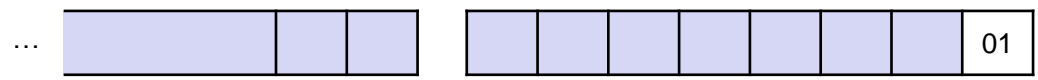
$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$



$$M_3[0] = R[0] \oplus C_2[0] \oplus 01$$

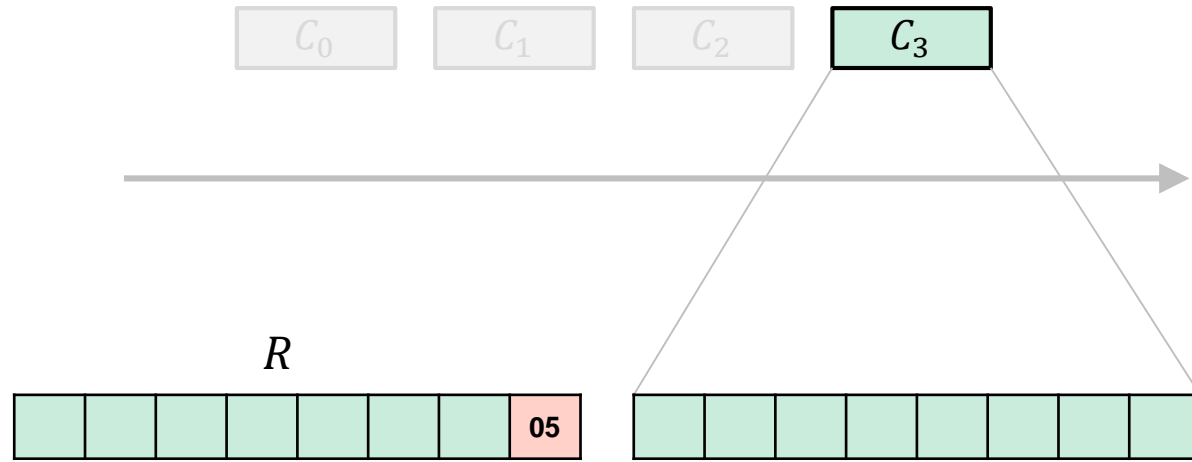
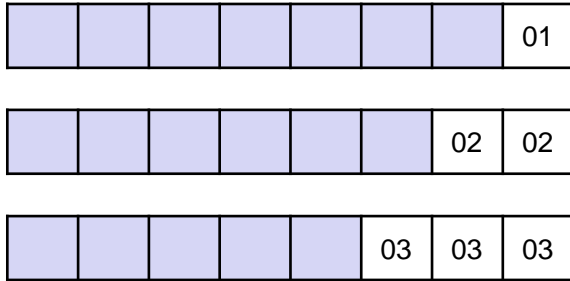
$$R[0] \oplus Z[0] = 01$$

$$R[0] \leftarrow R[0] \oplus 01 \oplus 02 (= 05)$$

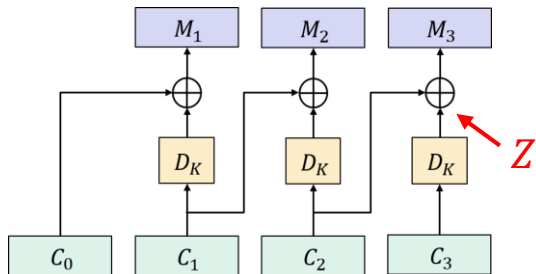


DTLS padding oracle attack

DTLS padding



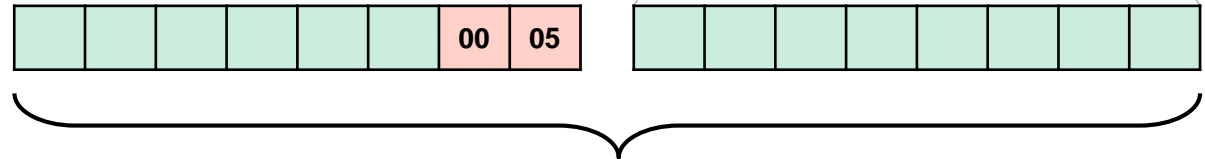
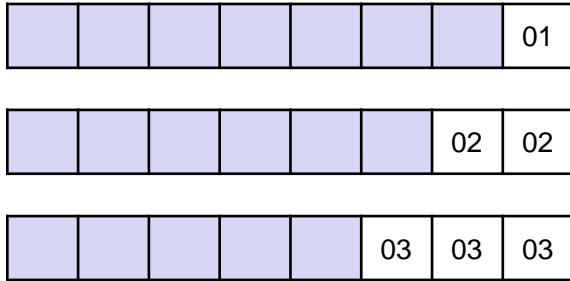
$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$



$$M_3[0] = R[0] \oplus C_2[0] \oplus 01$$

DTLS padding oracle attack

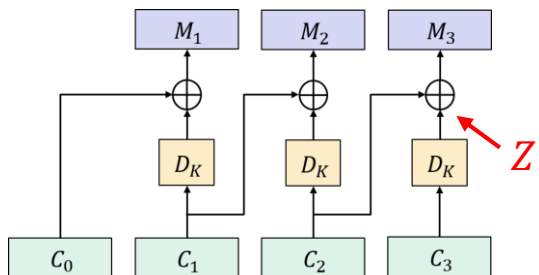
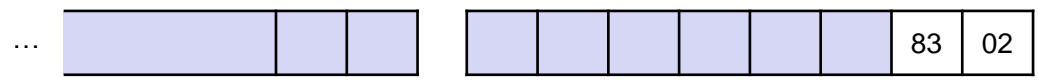
DTLS padding



Invalid



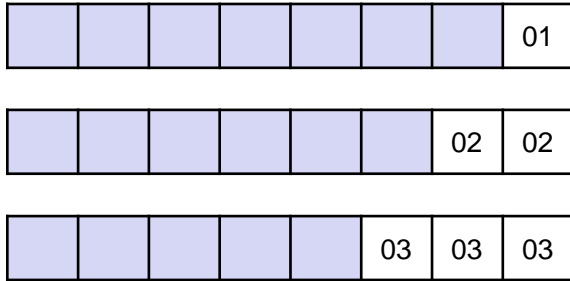
$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$



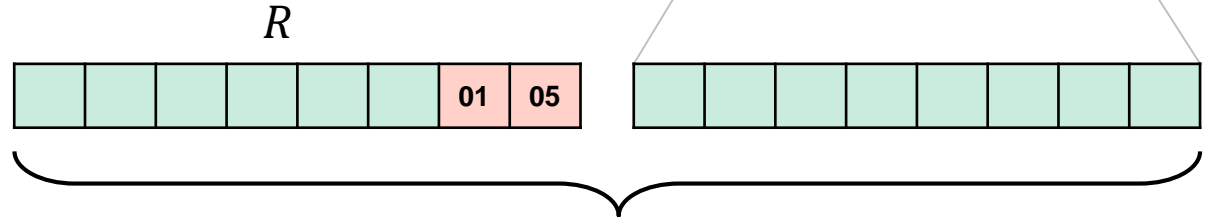
$$M_3[0] = R[0] \oplus C_2[0] \oplus 01$$

DTLS padding oracle attack

DTLS padding



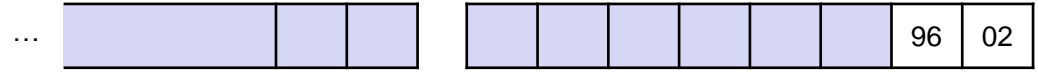
Invalid



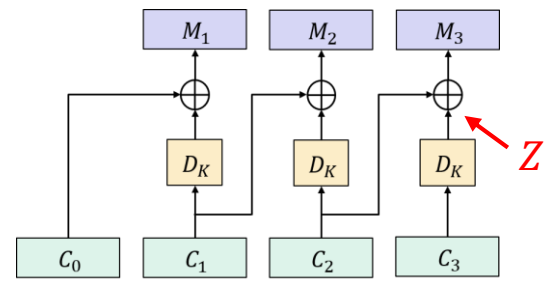
Decrypt



$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$

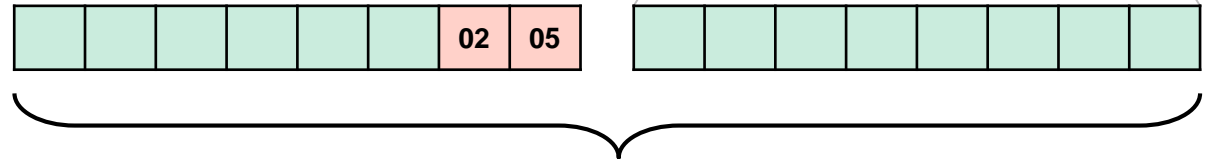
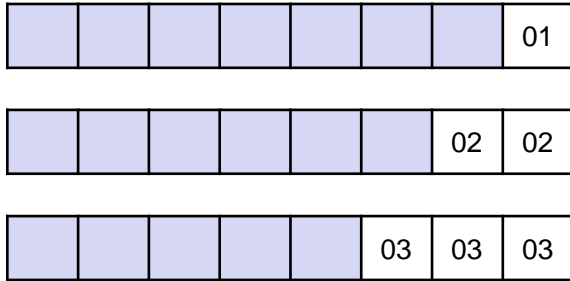


$$M_3[0] = R[0] \oplus C_2[0] \oplus 01$$



DTLS padding oracle attack

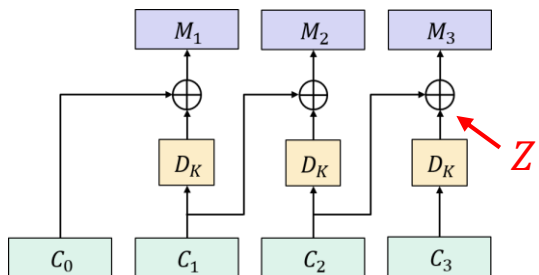
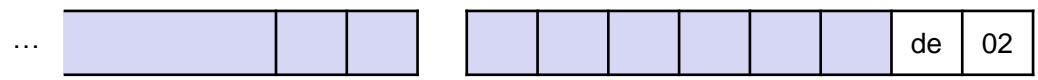
DTLS padding



Invalid



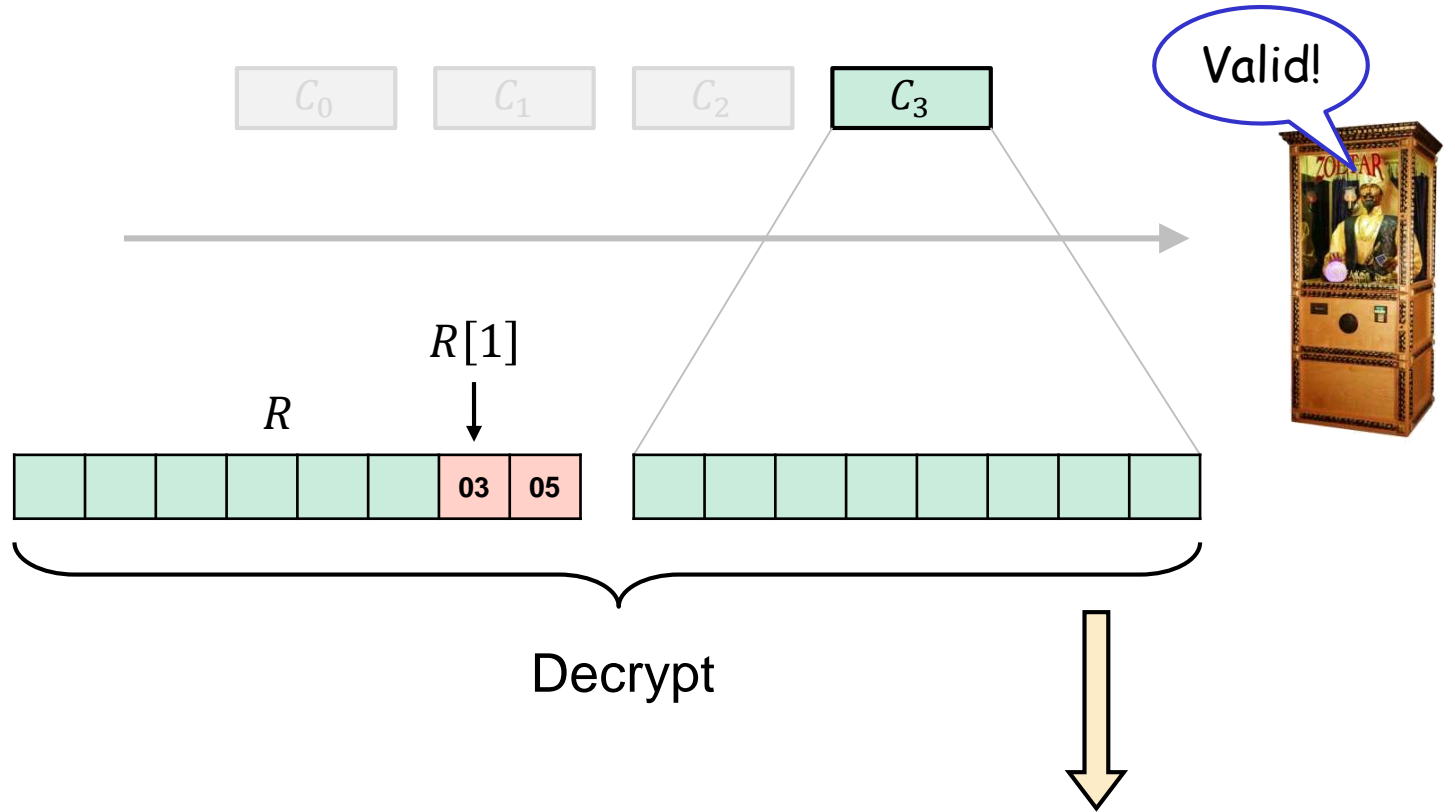
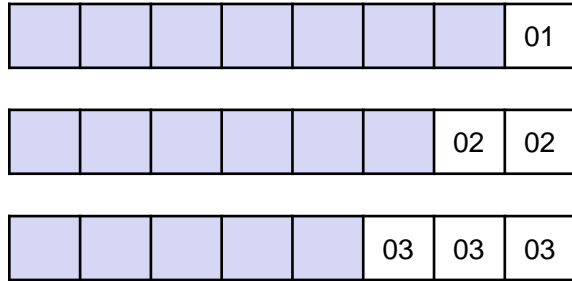
$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$



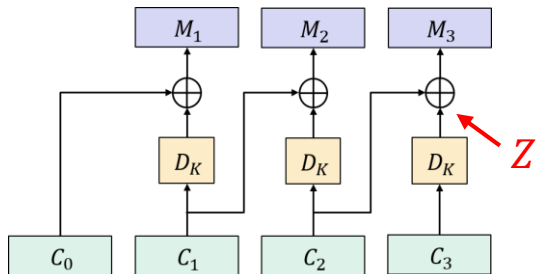
$$M_3[0] = R[0] \oplus C_2[0] \oplus 01$$

DTLS padding oracle attack

DTLS padding



$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$



$$M_3[0] = R[0] \oplus C_2[0] \oplus 01$$

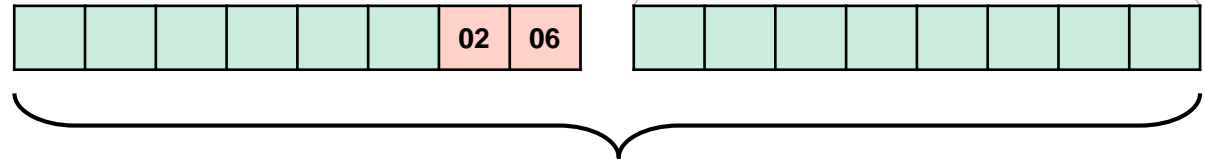
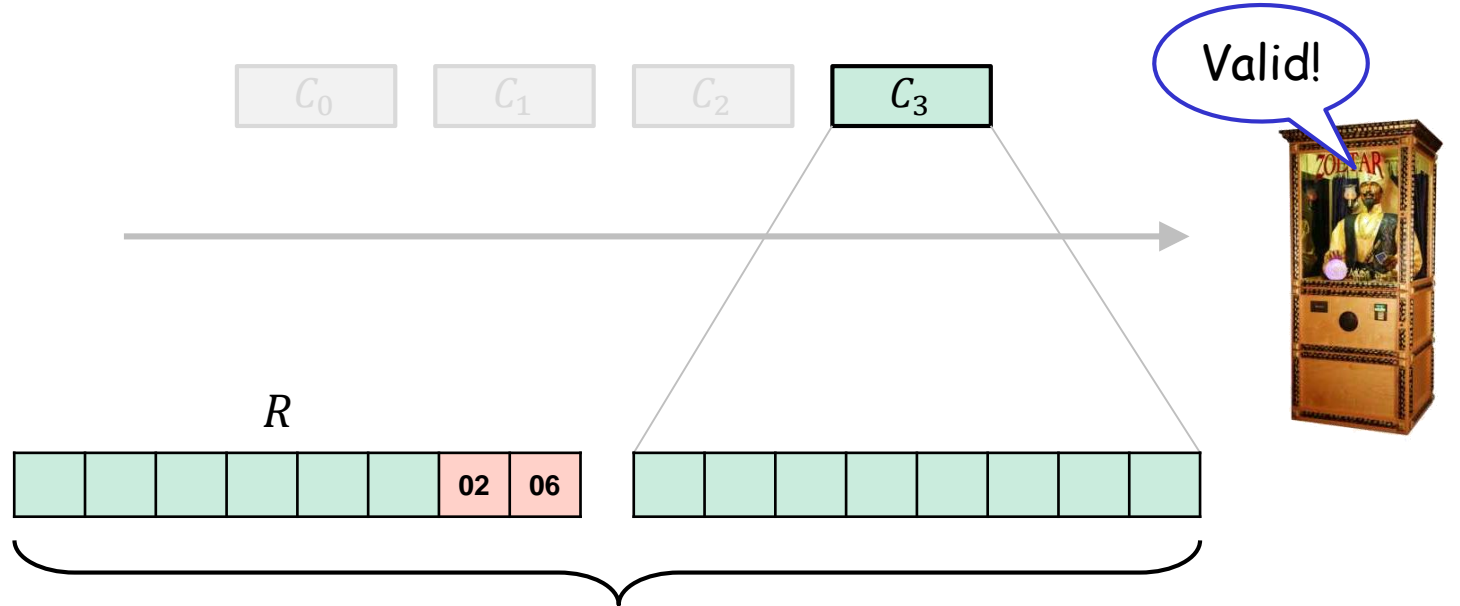
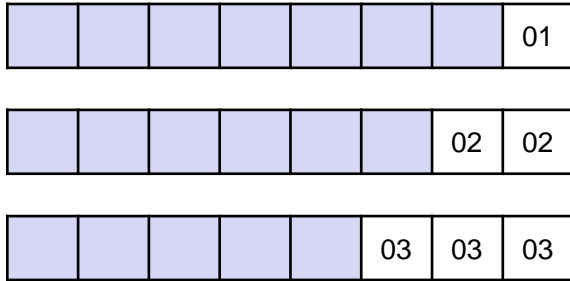
$$M_3[1] = R[1] \oplus C_2[1] \oplus 02$$

$$R[1] \oplus Z[1] = 02$$

$$R[1] \oplus (C_2[1] \oplus M_3[1]) = 02$$

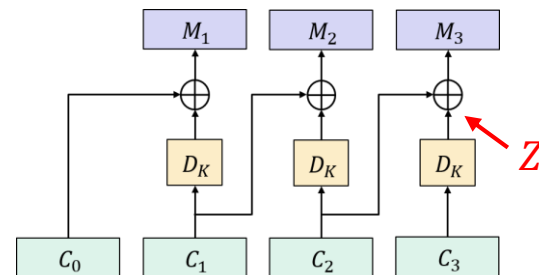
DTLS padding oracle attack

DTLS padding



Decrypt

$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$



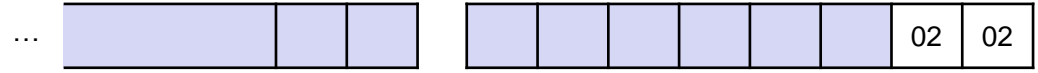
$$M_3[0] = R[0] \oplus C_2[0] \oplus 01$$

$$M_3[1] = R[1] \oplus C_2[1] \oplus 02$$

$$R[1] \oplus Z[1] = 02$$

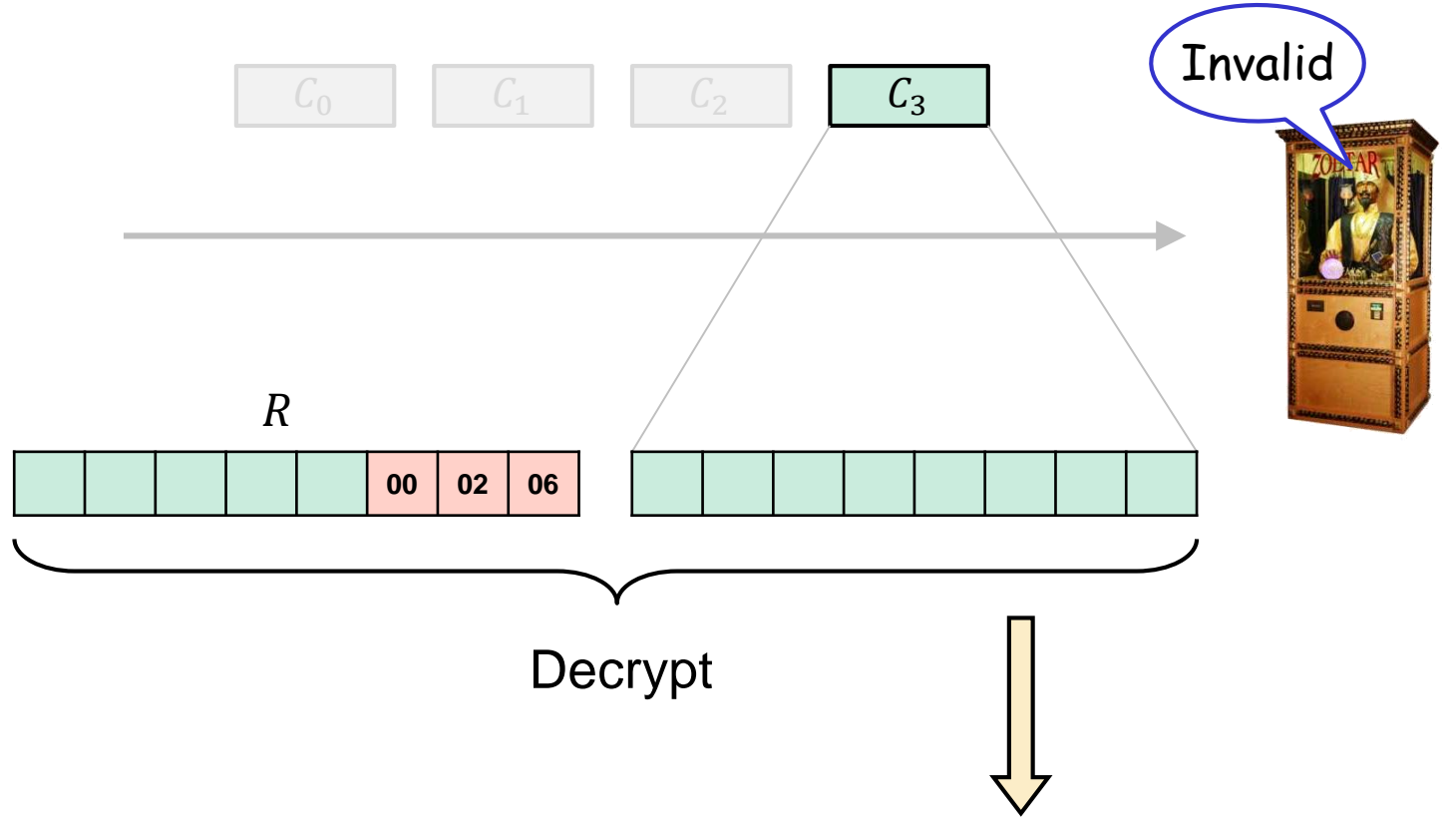
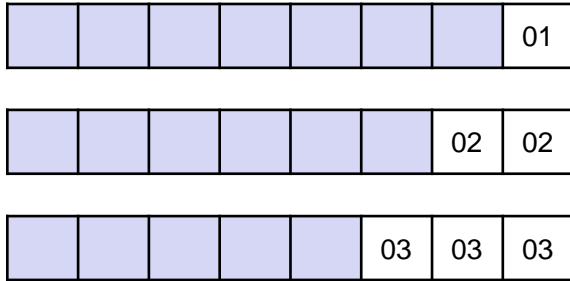
$$R[0] \leftarrow R[0] \oplus 02 \oplus 03 (= 06)$$

$$R[1] \leftarrow R[1] \oplus 02 \oplus 03 (= 02)$$

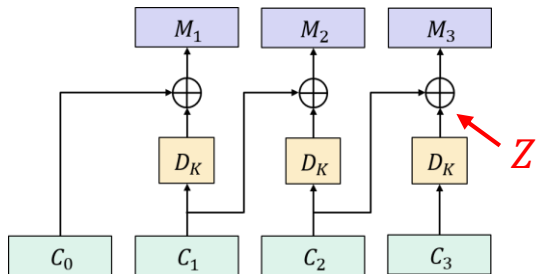


DTLS padding oracle attack

DTLS padding



$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$

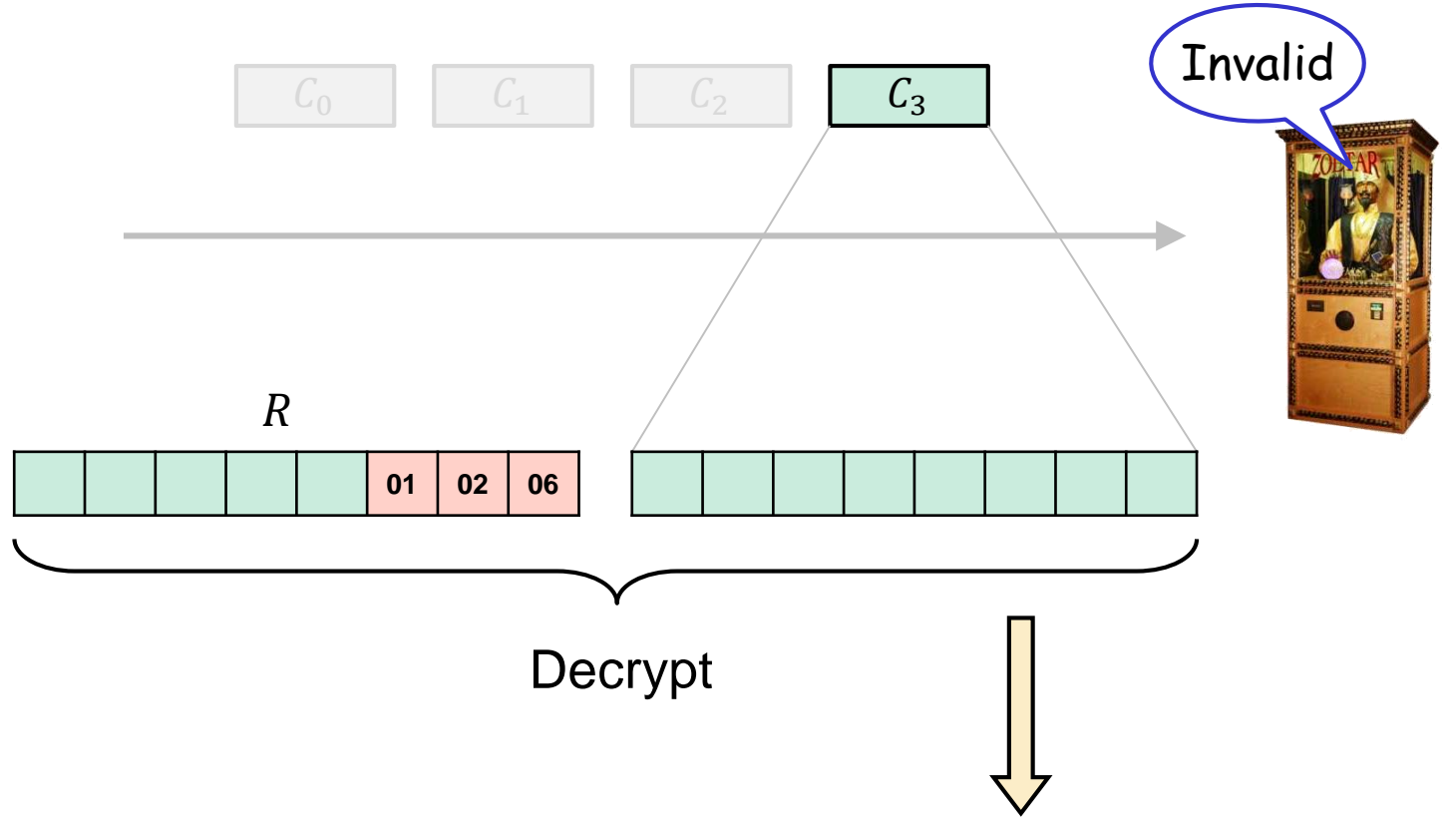
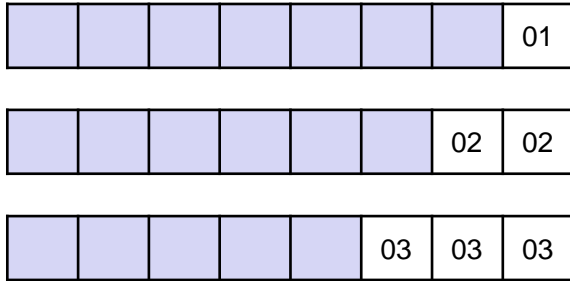


$$M_3[0] = R[0] \oplus C_2[0] \oplus 01$$

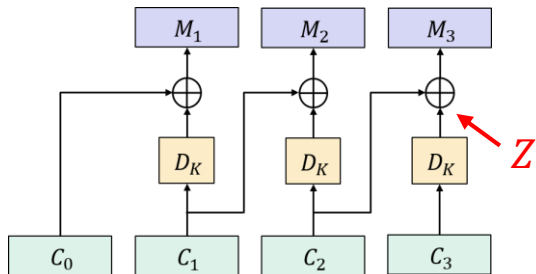
$$M_3[1] = R[1] \oplus C_2[1] \oplus 02$$

DTLS padding oracle attack

DTLS padding

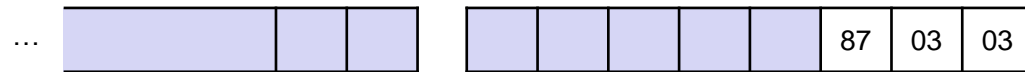


$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$



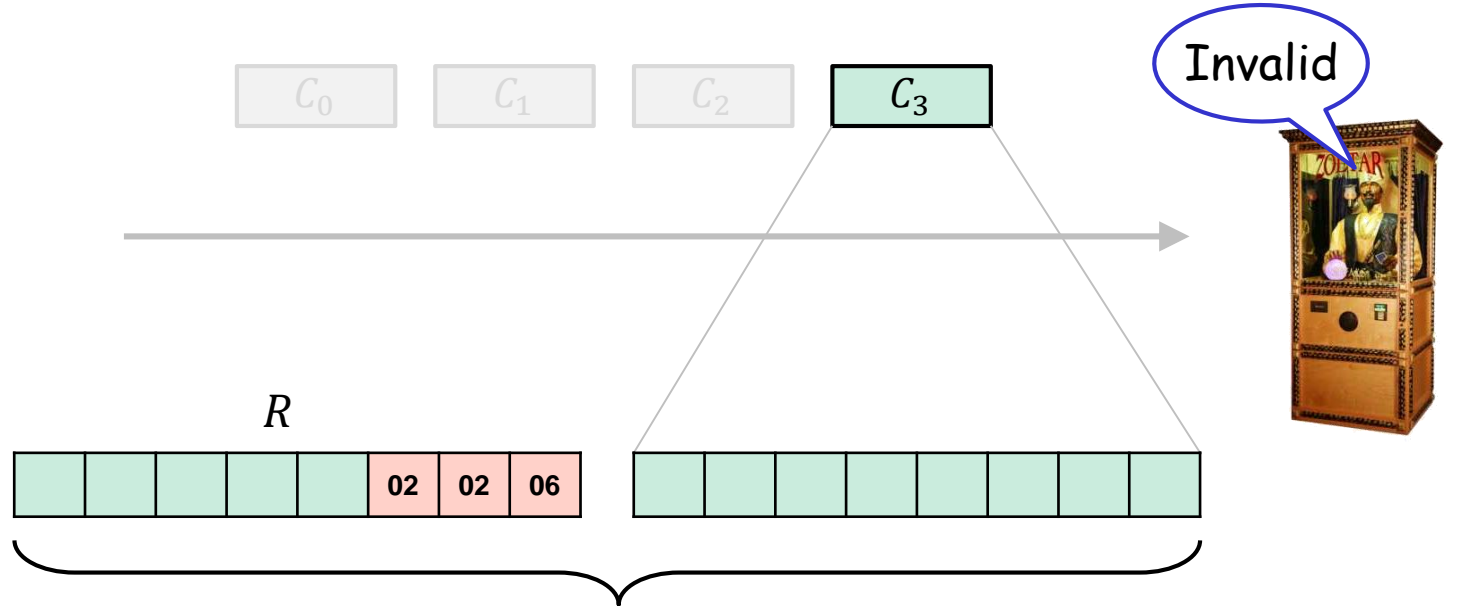
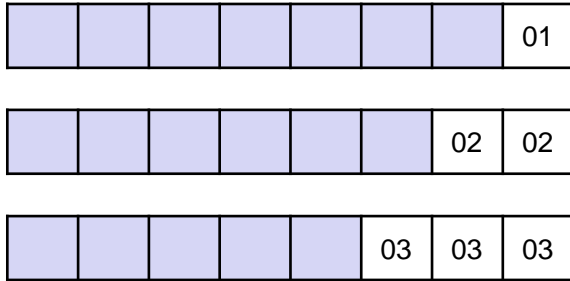
$$M_3[0] = R[0] \oplus C_2[0] \oplus 01$$

$$M_3[1] = R[1] \oplus C_2[1] \oplus 02$$



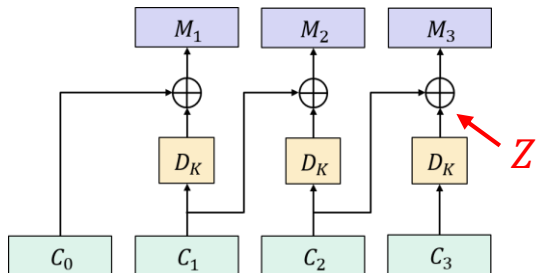
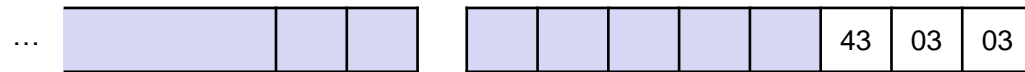
DTLS padding oracle attack

DTLS padding



Decrypt

$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$

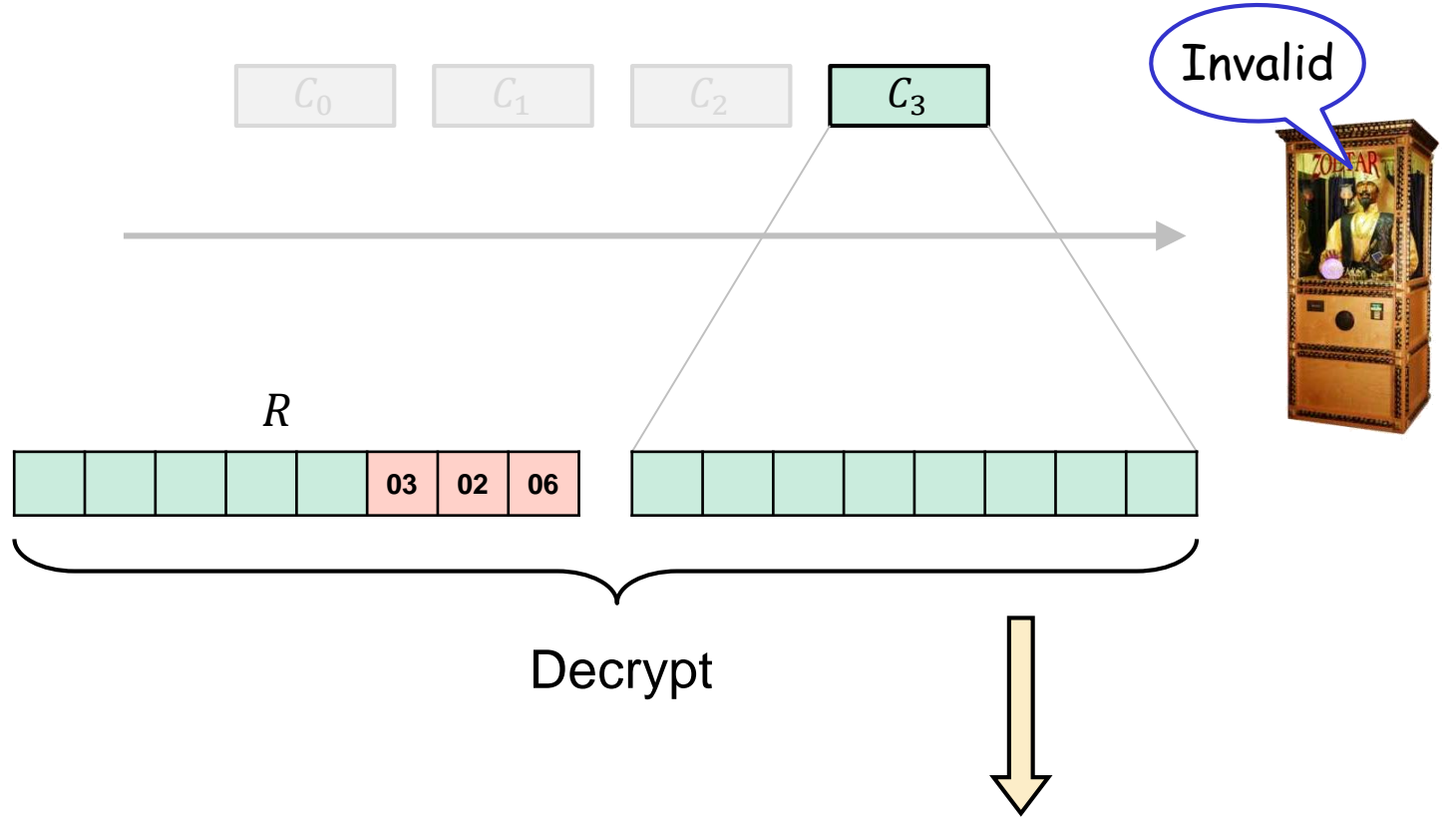
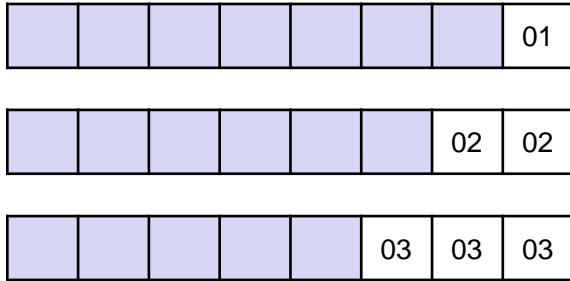


$$M_3[0] = R[0] \oplus C_2[0] \oplus 01$$

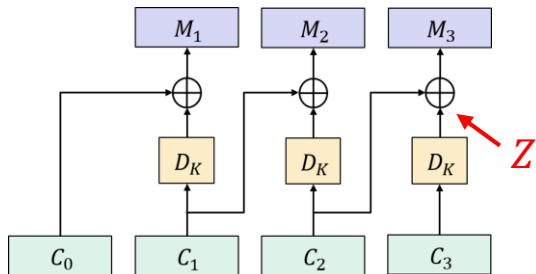
$$M_3[1] = R[1] \oplus C_2[1] \oplus 02$$

DTLS padding oracle attack

DTLS padding

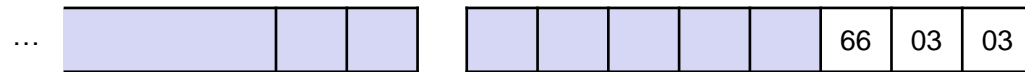


$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$



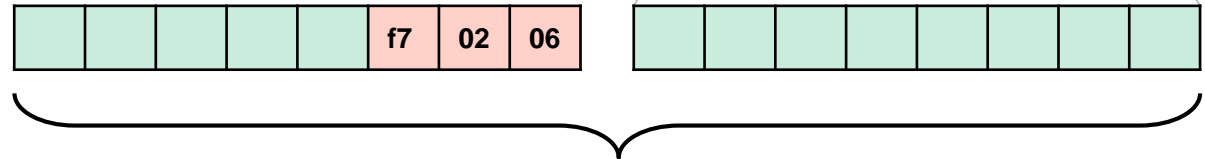
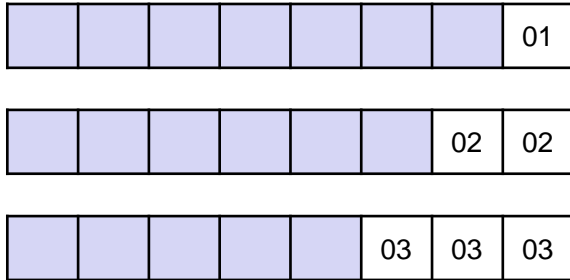
$$M_3[0] = R[0] \oplus C_2[0] \oplus 01$$

$$M_3[1] = R[1] \oplus C_2[1] \oplus 02$$



DTLS padding oracle attack

DTLS padding

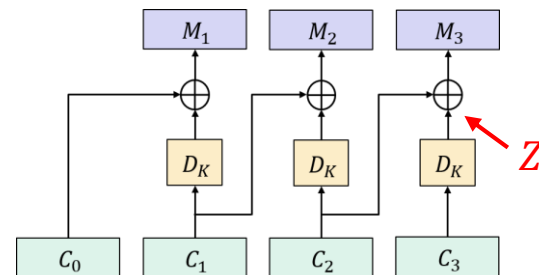
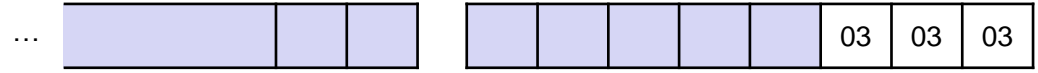


Valid!



Decrypt

$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$



$$M_3[0] = R[0] \oplus C_2[0] \oplus 01$$

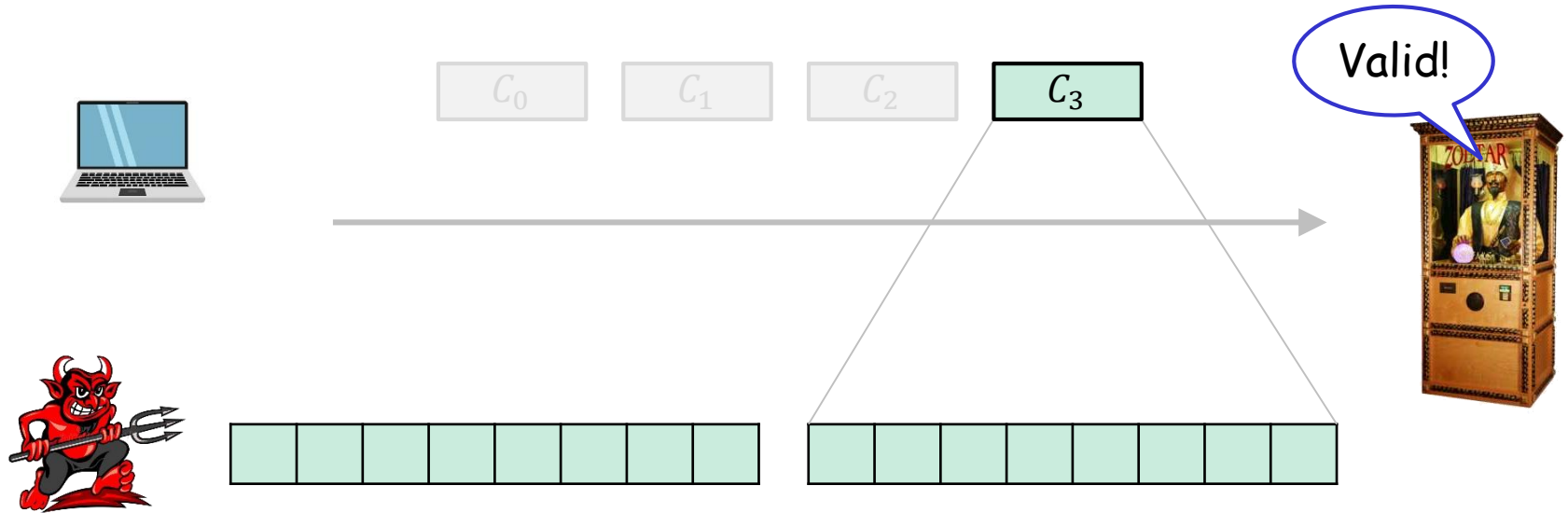
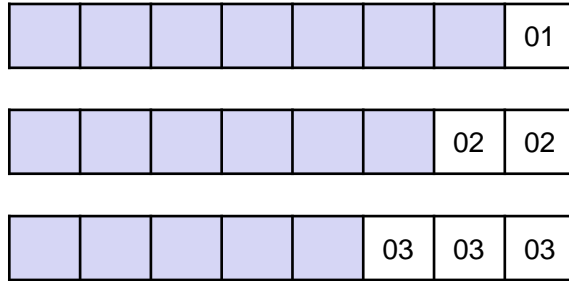
$$M_3[1] = R[1] \oplus C_2[1] \oplus 02$$

$$M_3[2] = R[2] \oplus C_2[2] \oplus 03$$

...continue process until we've found all of M_3

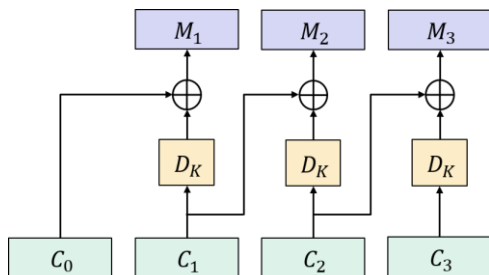
DTLS padding oracle attack

DTLS padding



But how do we get  to act like  ?

$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$



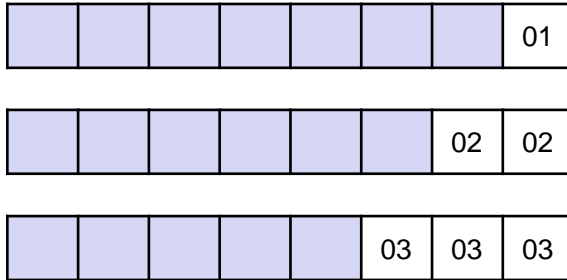
Answer: Timing differences:

Repeat attack to decrypt C_2, C_1, \dots

- If padding is invalid then server does not compute MAC
- If padding is valid then server computes MAC \Rightarrow longer time!

DTLS padding oracle attack

DTLS padding



Plaintext-Recovery Attacks Against Datagram TLS

Nadhem J. AlFardan and Kenneth G. Paterson*
 Information Security Group
 Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK
 {nadhem.alfardan.2009, kenny.paterson}@rhul.ac.uk

Abstract

The Datagram Transport Layer Security (DTLS) protocol provides confidentiality and integrity of data exchanged between a client and a server. We describe an efficient and full plaintext recovery attack against the OpenSSL implementation of DTLS, and a partial plaintext recovery attack against the GnuTLS implementation of DTLS. The attack against the OpenSSL implementation is a variant of Vaudenay's padding oracle attack and exploits small timing differences arising during the cryptographic processing of DTLS packets. It would have been prevented if the OpenSSL implementation had been in accordance with the DTLS RFC. In contrast, the GnuTLS implementation does follow the DTLS RFC closely, but is still vulnerable to attack. The attacks require new insights to overcome the lack of error messages in DTLS and to amplify the timing differences. We discuss the reasons why these implementations are insecure, drawing lessons for secure protocol design and implementation in general.

Keywords TLS, DTLS, CBC-mode encryption, padding oracle, attack, timing, OpenSSL, GnuTLS.

1 Introduction

DTLS, OpenSSL and GnuTLS: The Datagram Transport Layer Security (DTLS) protocol was first introduced at NDSS in 2004 [10]. Two years later, the Internet Engineering Task Force (IETF) assigned Request for Comments (RFC) 4347 [11] to DTLS. The aim of DTLS is to provide a datagram-compatible variant of TLSv1.1 [6] that eliminates the dependency on the Transport Control Protocol (TCP). Since its introduction, there has been a growing interest in the security services offered by DTLS. Leading implementations of DTLS can be found in OpenSSL¹ and

*This author's research supported by an EPSRC Leadership Fellowship, EP/H005455/1.
¹<http://www.openssl.org>

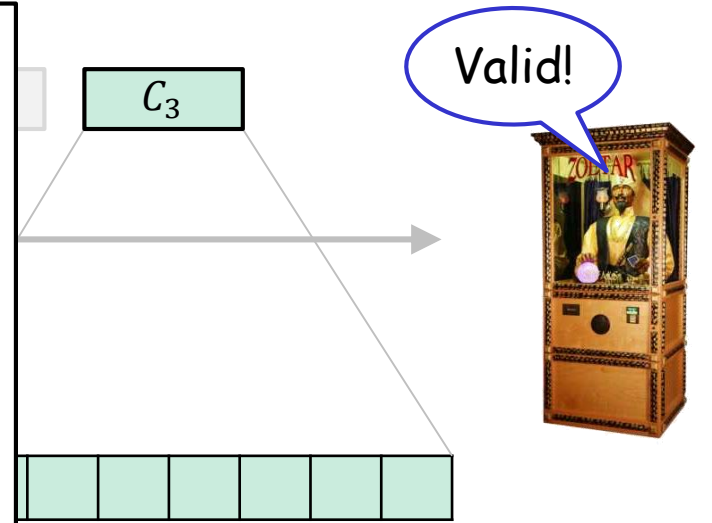
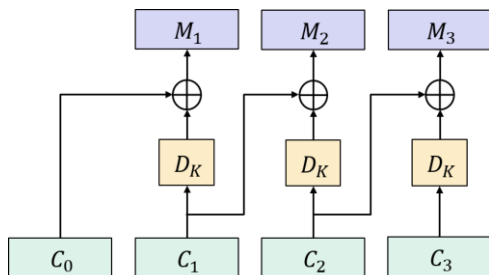
GnuTLS². Both of these provide source toolkits that implement TLS and DTLS as well as being general purpose cryptographic libraries that software developers can use. The first release of OpenSSL to implement DTLS was 0.9.8. Since its release, DTLS has become a mainstream protocol in OpenSSL. There are also a number of commercial products that have taken advantage of DTLS. For example, DTLS is used to secure Virtual Private Networks (VPNs)^{3,4} and wireless traffic⁵. Platforms such as Microsoft Windows, Microsoft .NET and Linux can also make use of DTLS⁶. In addition, the number of RFC documents that are being published on DTLS is increasing. Recent examples include RFC 5415 [1], RFC 5953 [8] and RFC 6012 [13]. A new version of DTLS is currently under development in the IETF to bring DTLS into line with TLSv1.2.

Padding oracle attacks: According to [11], the DTLS protocol is based on TLSv1.1 and provides equivalent security guarantees. In particular, then, one would expect implementations of DTLS to be resilient to attacks on TLS known prior to the development of TLSv1.1, especially those attacks explicitly mentioned in RFC 4346 [6], the specification for TLSv1.1.

One such example is the padding oracle attack introduced by Vaudenay in [15] and applied to OpenSSL by Canvel *et al.* in [2]. This attack exploits the MAC-then-Pad-then-Encrypt construction used by TLS and makes use of subtle timing differences that may arise in the cryptographic processing carried out during decryption, in order to glean information about the correctness or otherwise of the plaintext format underlying a target ciphertext. Specifically, Canvel *et al.* used timing of encrypted TLS error messages in order to distinguish whether the padding occurring

²<http://www.gnu.org/software/gnutls>
³<http://www.cisco.com/en/US/products/ps10884/index.html>
⁴<http://campagnol.sourceforge.net>
⁵http://www.cisco.com/en/US/docs/wireless/controller/7.0MR1/configuration/guide/cgi_lwap.html
⁶<http://www.eldos.com/sbb/deac-ssl.php>

$$C = \text{CBC}(K, M \parallel \text{MAC}_K(M) \parallel \text{pad})$$



act like



?

at C_2, C_1, \dots

es not compute MAC

utes MAC \Rightarrow longer time!

DTLS padding oracle attack

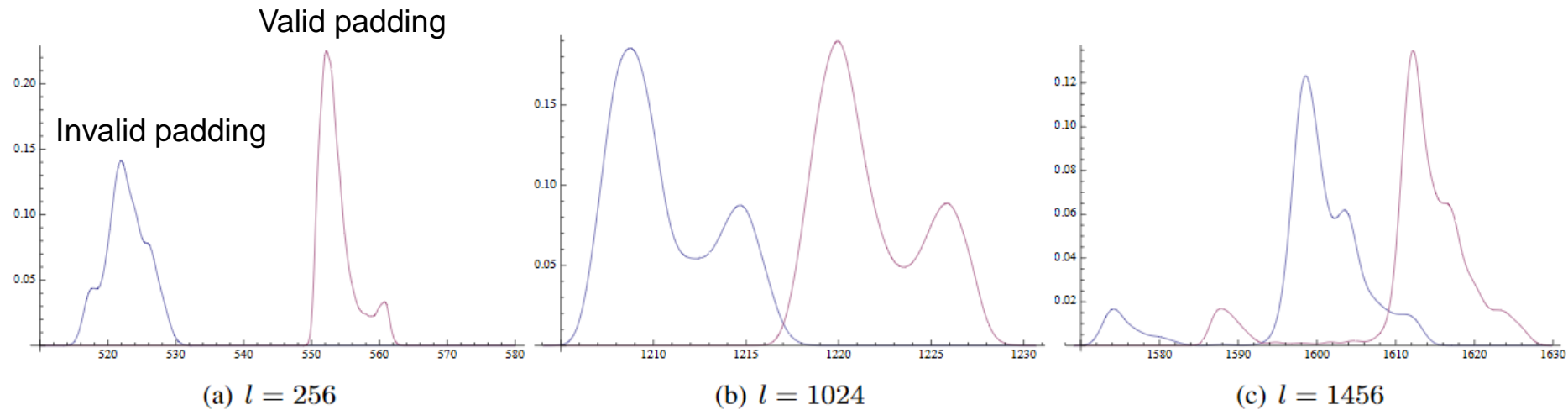
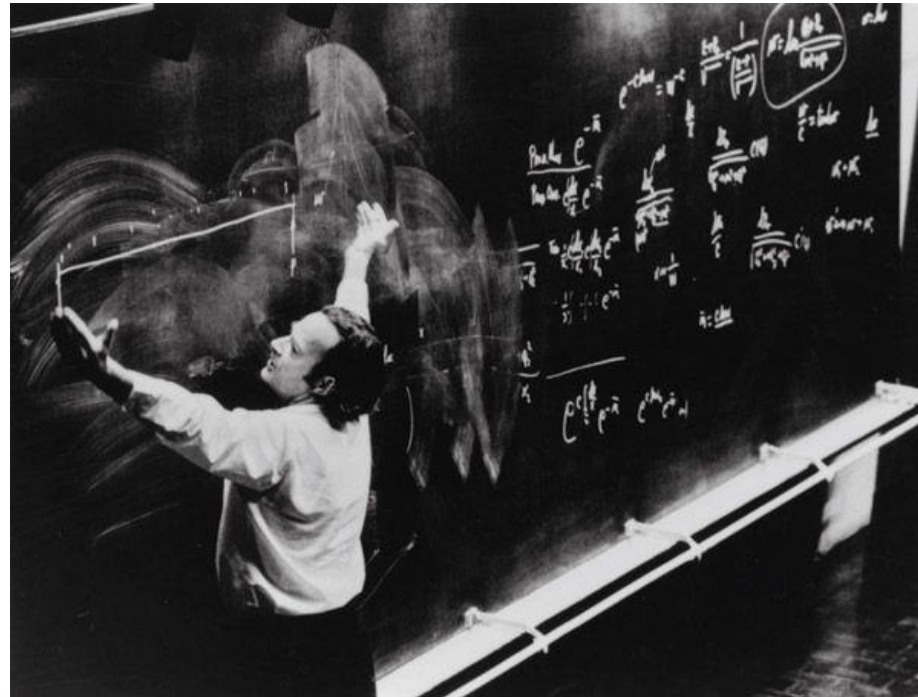


Figure 3. AES-256 – PDFs for $n = 10$ and varying l .

- Timing difference between valid and invalid padding: a few μs
 - Enough to carry out the attack
 - Repeat attack $n = 10 - 50$ times per byte to filter out timing noise
 - Able to decrypt each byte with probability 0.97–0.99

DTLS padding oracle attack – conclusions

- Decryption oracles are **real** \Rightarrow chosen-**ciphertext** security is *necessary*
- Even a limited oracle that only leaks padding validity can be used to decrypt full ciphertext
- Tiny timing differences enough to induce oracle
- Many other timing attacks on TLS/DTLS done since
 - Lucky-13



Authenticated encryption

Authenticated encryption

- Want privacy *and* integrity from a single primitive:
an **authenticated encryption** scheme
- Syntactically (almost) the same as a normal encryption scheme

Authenticated encryption – syntax

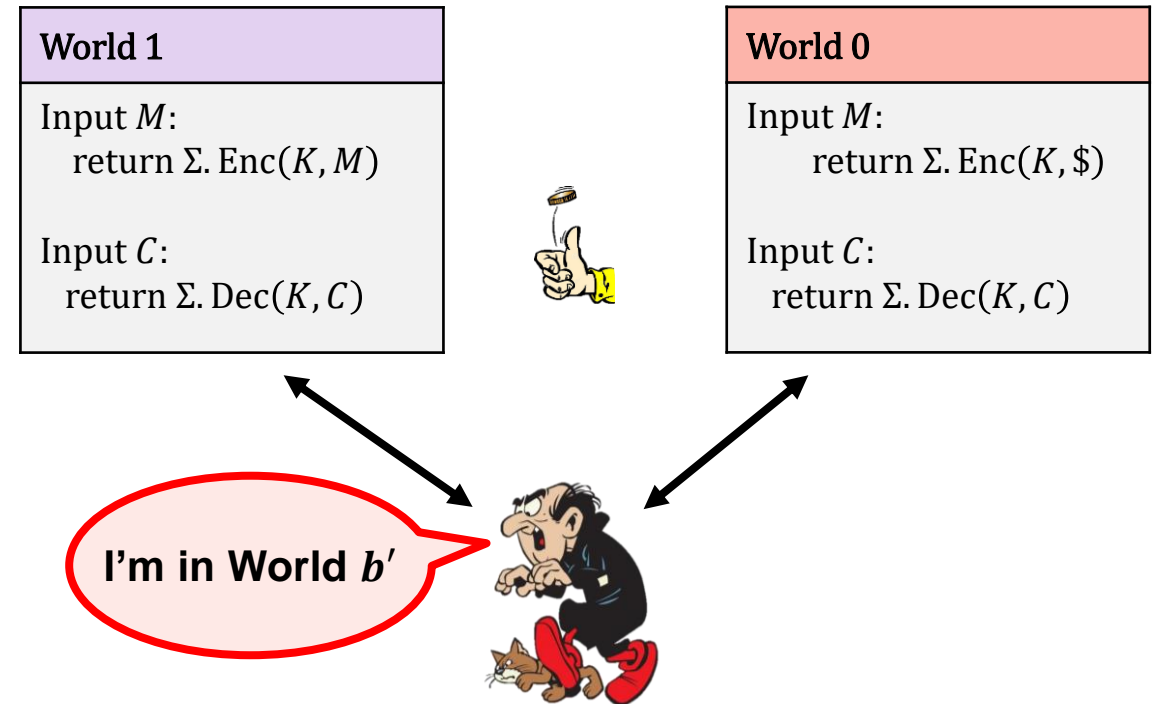
$$\text{Enc} : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{C}$$

$$\text{Enc}(K, M) = \text{Enc}_K(M) = C$$

$$\text{Dec} : \mathcal{K} \times \mathcal{C} \rightarrow \mathcal{M} \cup \{\perp\}$$

$$\text{Dec}(K, C) = \text{Dec}_K(C) = M / \perp$$

Authenticated encryption – security



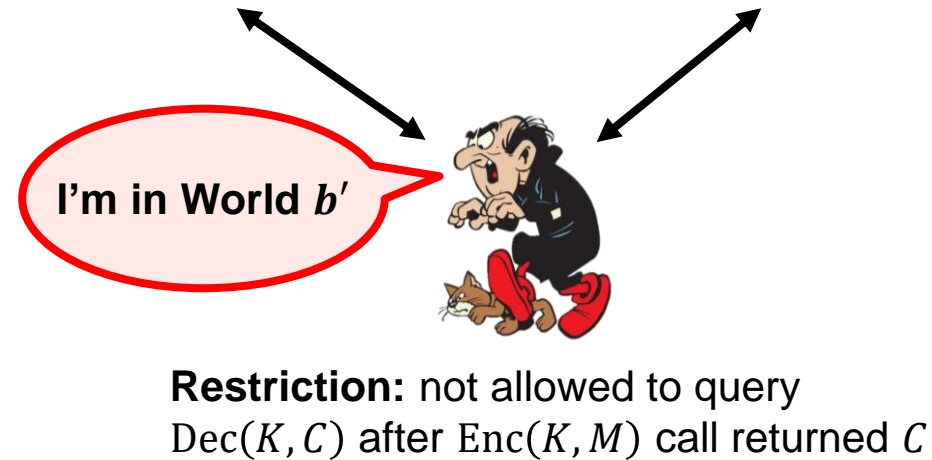
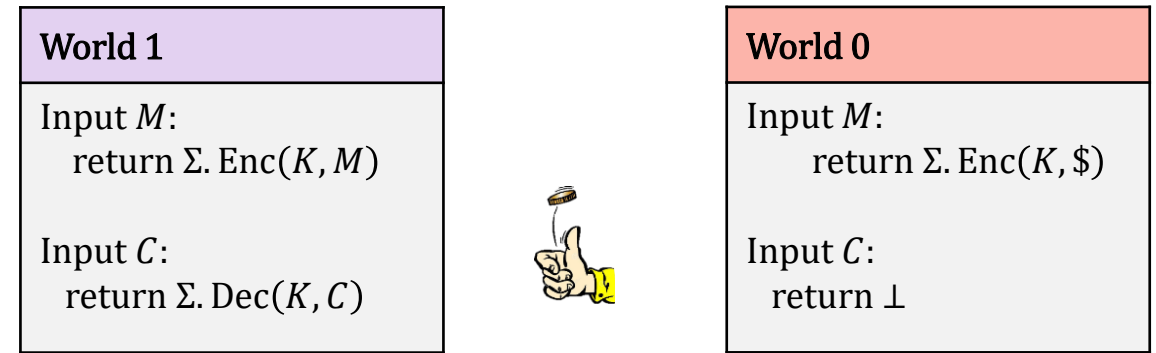
Restriction: not allowed to query $\text{Dec}(K, C)$ after $\text{Enc}(K, M)$ call returned C

Authenticated encryption – security

$\text{Exp}_{\Sigma}^{\text{ae}}(A)$	
1.	$b \xleftarrow{\$} \{0,1\}$
2.	Ciphertexts $\leftarrow []$
3.	$K \xleftarrow{\$} \Sigma.\text{KeyGen}$
4.	$b' \leftarrow A^{\mathcal{E}(\cdot), \mathcal{D}(\cdot)}$
5.	return $b' \stackrel{?}{=} b$
$\mathcal{E}(M)$	

1.	$R \xleftarrow{\$} \{0,1\}^{ M }$
2.	$C_0 \leftarrow \Sigma.\text{Enc}(K, R)$
3.	$C_1 \leftarrow \Sigma.\text{Enc}(K, M)$
4.	Ciphertexts.add(C_b)
5.	return C_b
$\mathcal{D}(C)$	

1.	if $C \in \text{Ciphertexts}$ then // cheating!
2.	return \perp
3.	$M_0 \leftarrow \perp$
4.	$M_1 \leftarrow \Sigma.\text{Dec}(K, C)$
5.	return M_b



Definition: The **AE-advantage** of an adversary A is

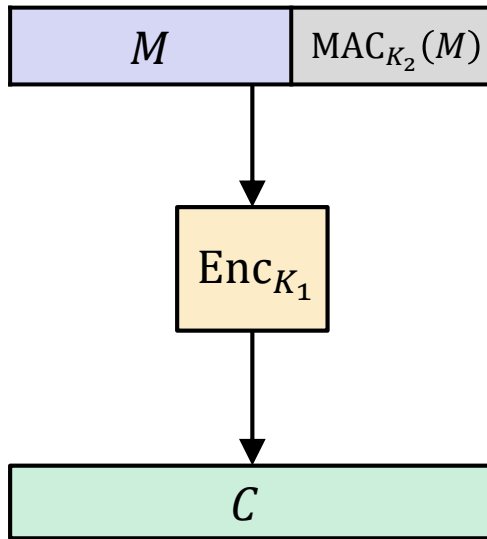
$$\text{Adv}_{\Sigma}^{\text{ae}}(A) = |2 \cdot \Pr[\text{Exp}_{\Sigma}^{\text{ae}}(A) \Rightarrow \text{true}] - 1|$$

AE security definition – implications

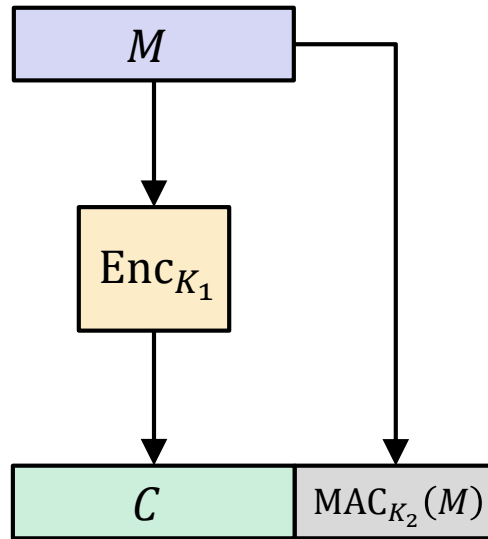
- **Privacy:** adversary cannot distinguish encryption of real messages from encryption random messages
- **Integrity:** adversary is not able to **forge** ciphertexts: any ciphertext *not* produced by the legitimate sender will decrypt to \perp
- Does *not* protect against **replay attacks**

Generic composition: AE from Encryption + MAC

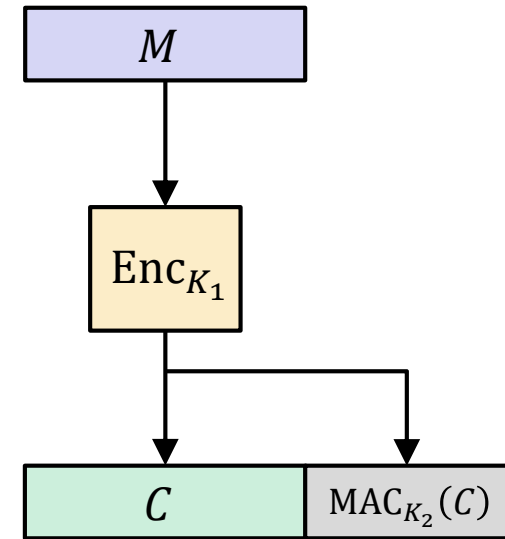
MAC-then-Encrypt (MtE)



Encrypt-and-MAC (E&M)



Encrypt-then-MAC (EtM)



Generic composition: secure?

Enc is IND-CPA secure and MAC is UF-CMA secure; which combination is secure?

- E&M: $C || T \leftarrow \text{Enc}(K_1, M) || \text{MAC}(K_2, M)$
 - Used in SSH



$$\text{Enc}(K_1, M) || \underbrace{M || \text{MAC}'(K_2, M)}_{\text{MAC}(K_2, M)}$$

- MtE: $C \leftarrow \text{Enc}(K_1, M || \text{MAC}(K_2, M))$
 - Used in TLS up to TLS 1.2



$$0000 || \underbrace{\text{Enc}(K_1, M, \text{MAC}(K_2, M))}_C$$

- EtM: $C || T \leftarrow \text{Enc}(K_1, M) || \text{MAC}(K_2, C)$
 - Used in IPsec
 - MAC must cover all of C (including IVs)

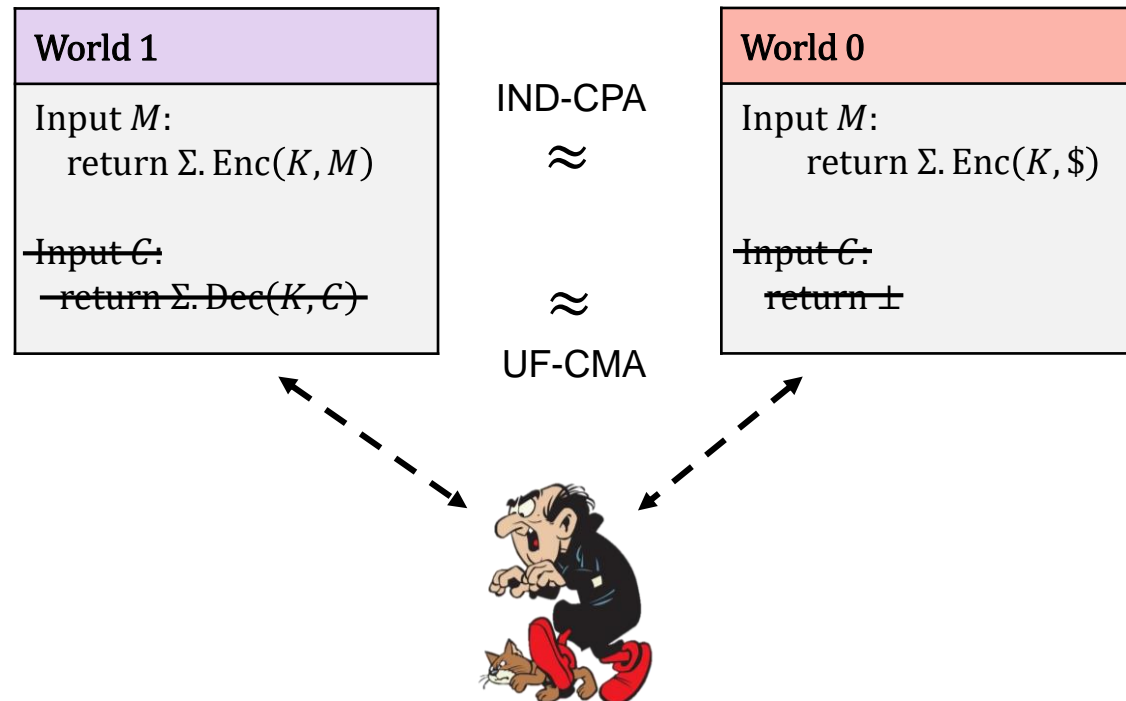


EtM – security

Theorem: for *any* AE adversary A against EtM there are adversaries B and C against Enc and MAC such that

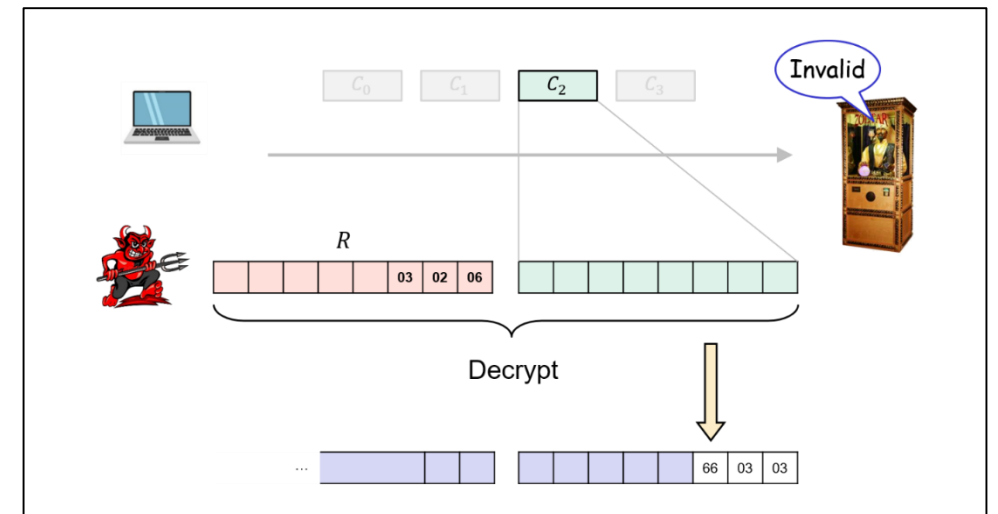
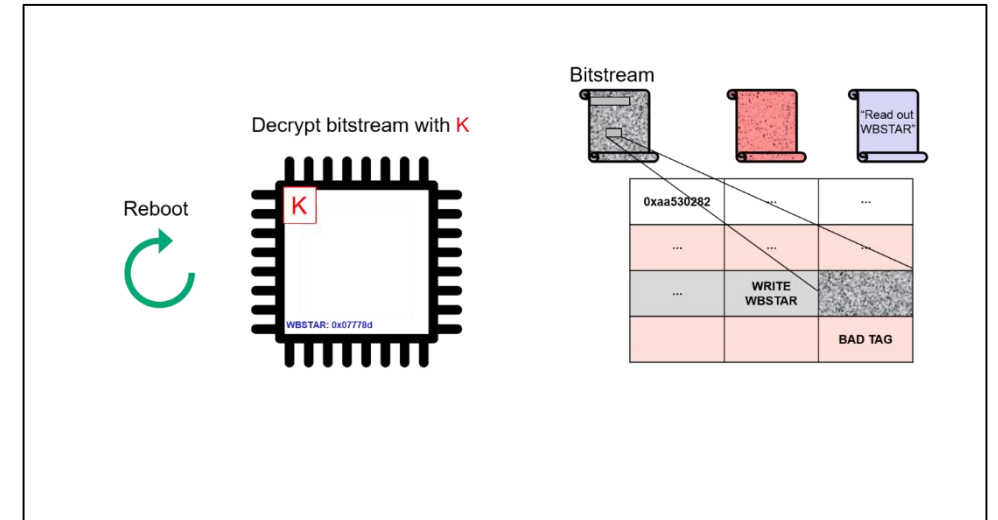
$$\mathbf{Adv}_{\text{EtM}}^{\text{ae}}(A) \leq \mathbf{Adv}_{\text{Enc}}^{\text{ind-cpa}}(B) + 2 \cdot \mathbf{Adv}_{\text{MAC}}^{\text{uf-cma}}(C)$$

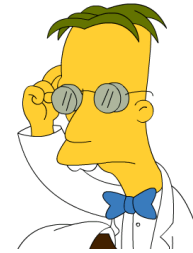
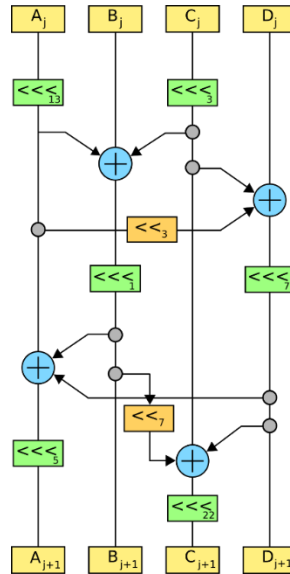
Proof:



CCA-attacks – revisited

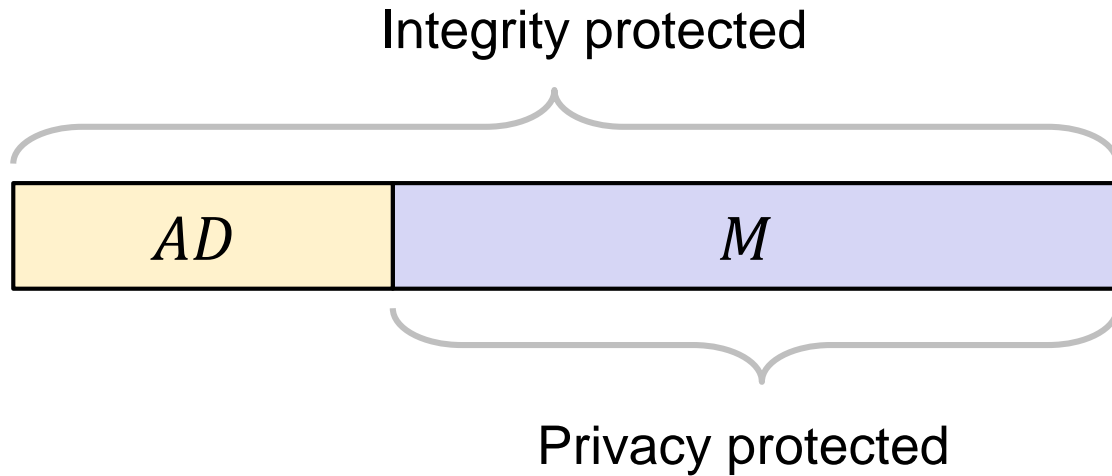
- MtE used in both Starbleed attack and in DTLS padding oracle attack
- Gave rise to (partial) decryption oracles
- Attacks would not have been possible with an AE secure scheme



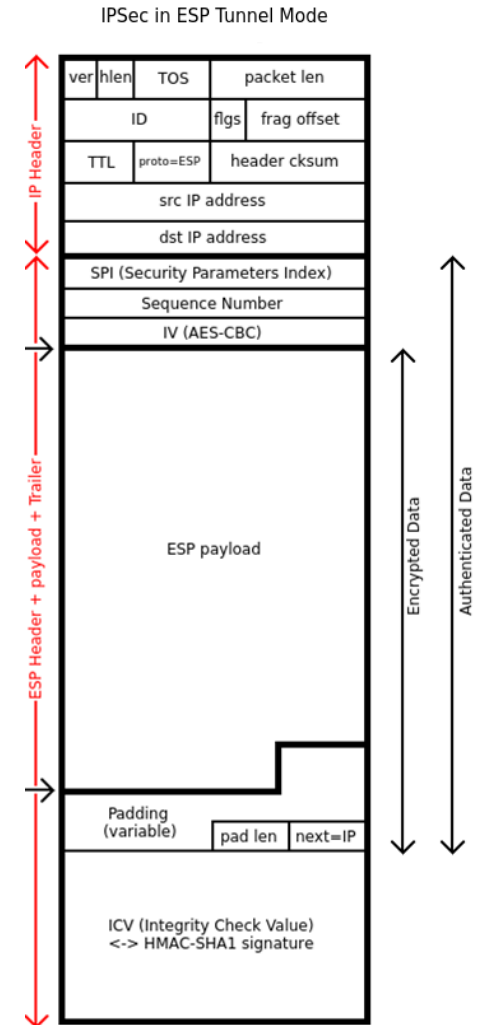


AEAD – Authenticated encryption with associated data

AEAD – AE with associated data (AD)



- AD – data that can't be encrypted but still need integrity protection
 - Headers in protocols
 - Configuration data
 - Metadata



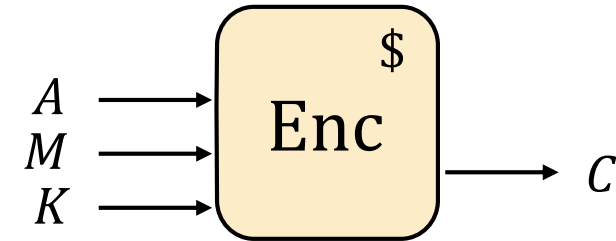
Authenticated encryption w/associated data (AEAD) – syntax

$$\text{Enc} : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{C}$$

$$\text{Enc}(K, M) = \text{Enc}_K(M) = C$$

$$\text{Dec} : \mathcal{K} \times \mathcal{C} \rightarrow \mathcal{M} \cup \{\perp\}$$

$$\text{Dec}(K, C) = \text{Dec}_K(C) = M/\perp$$



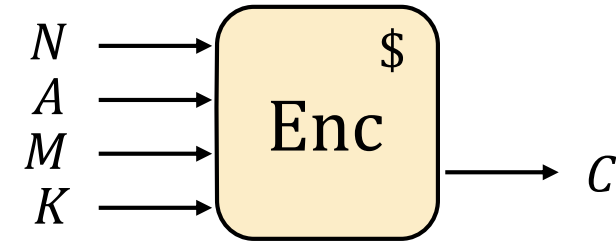
Authenticated encryption w/associated data (AEAD) – syntax

$$\text{Enc} : \mathcal{K} \times \mathcal{A} \times \mathcal{M} \rightarrow \mathcal{C}$$

$$\text{Enc}(K, A, M) = \text{Enc}_K(A, M) = \text{Enc}_K^A(M) = C$$

$$\text{Dec} : \mathcal{K} \times \mathcal{A} \times \mathcal{C} \rightarrow \mathcal{M} \cup \{\perp\}$$

$$\text{Dec}(K, A, C) = \text{Dec}_K(A, C) = \text{Dec}_K^A(C) = M/\perp$$



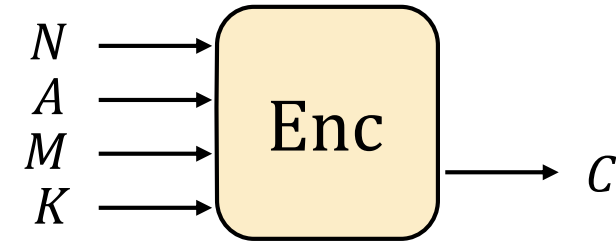
Authenticated encryption w/associated data (AEAD) – syntax

$$\text{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \rightarrow \mathcal{C}$$

$$\text{Enc}(K, N, A, M) = \text{Enc}_K^N(A, M) = \text{Enc}_K^{N,A}(M) = C$$

$$\text{Dec} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \rightarrow \mathcal{M} \cup \{\perp\}$$

$$\text{Dec}(K, N, A, C) = \text{Dec}_K^N(A, C) = \text{Dec}_K^{N,A}(C) = M/\perp$$



\mathcal{K} – key space

\mathcal{N} – nonce space

\mathcal{A} – associated data space

\mathcal{M} – message space

\mathcal{C} – ciphertext space

Correctness requirement: $\forall K \in \mathcal{K}, \forall N \in \mathcal{N}, \forall A \in \mathcal{A}, \forall M \in \mathcal{M}$:

$$\text{Dec}(K, N, A, \text{Enc}(K, N, A, M)) = M$$

AEAD security (nonce-based)

$\text{Exp}_{\Sigma}^{\text{aead}}(A)$

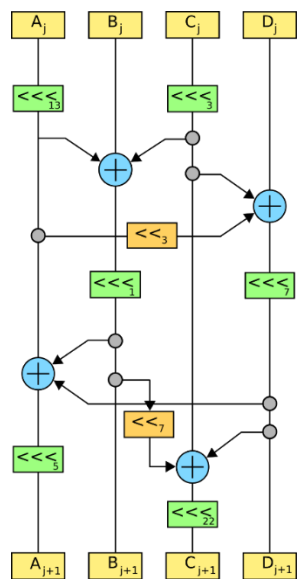
<ol style="list-style-type: none"> 1. $b \xleftarrow{\\$} \{0,1\}$ 2. Nonces $\leftarrow []$ 3. Ciphertexts $\leftarrow []$ 4. $K \xleftarrow{\\$} \Sigma.\text{KeyGen}$ 5. $b' \leftarrow A^{\mathcal{E}(\cdot, \cdot), \mathcal{D}(\cdot, \cdot)}$ 6. return $b' \stackrel{?}{=} b$ 	<p>$\mathcal{E}(N, A, M)$</p> <p>-----</p> <ol style="list-style-type: none"> 1. if $N \in \text{Nonces}$ then // cheating! 2. return \perp 3. $R \xleftarrow{\\$} \{0,1\}^{ M }$ 4. $C_0 \leftarrow \Sigma.\text{Enc}(K, N, A, R)$ 5. $C_1 \leftarrow \Sigma.\text{Enc}(K, N, A, M)$ 6. Nonces.add(N) 7. Ciphertexts.add(C_b) 8. return C_b
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

$\mathcal{D}(N, A, C)$

1. **if** $C \in \text{Ciphertexts}$ **then** // cheating!
2. **return** \perp
3. $C_0 \leftarrow \perp$
4. $C_1 \leftarrow \Sigma.\text{Dec}(K, N, A, C)$
5. **return** C_b

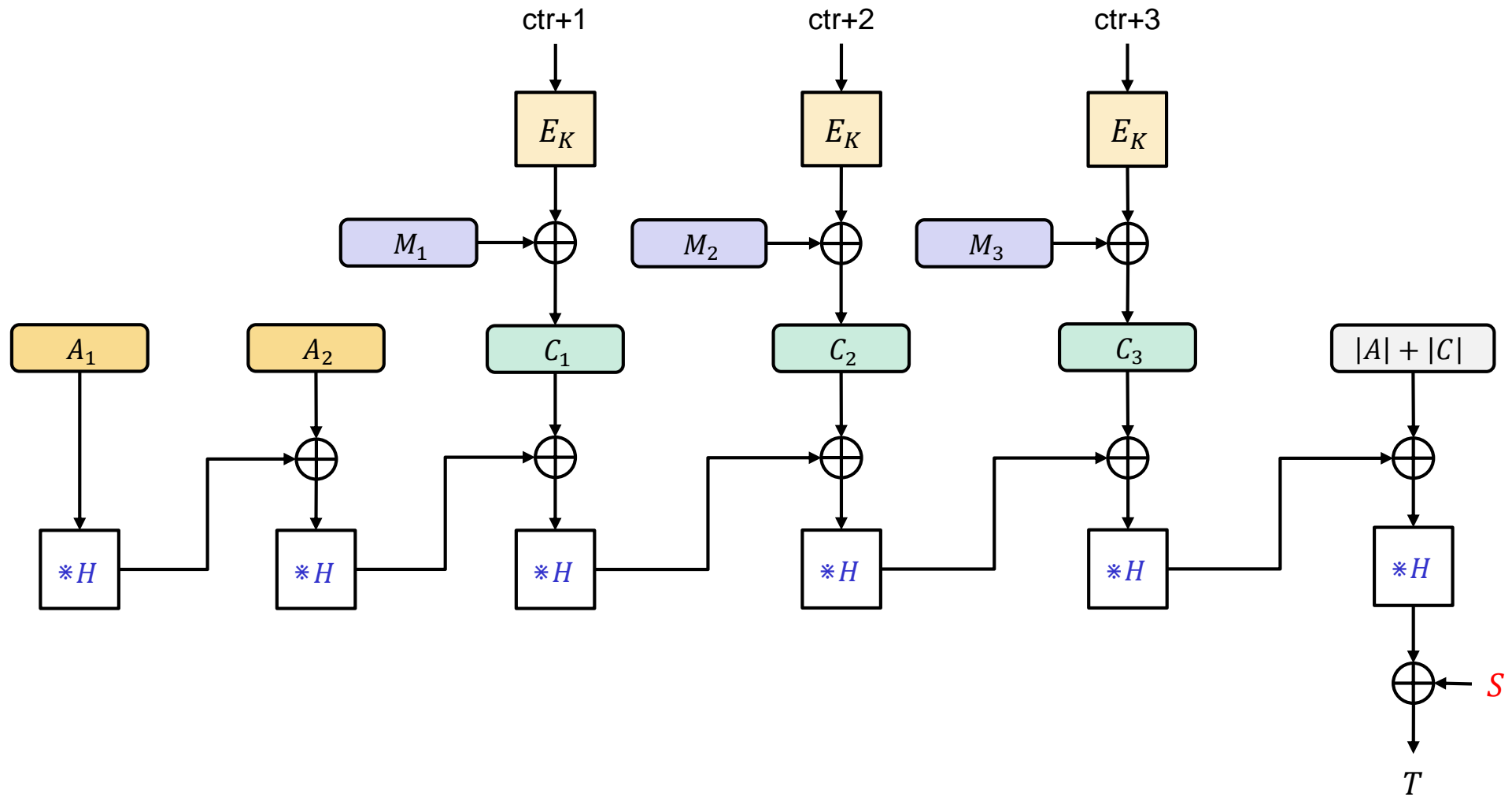
Definition: The **AEAD-advantage** of an adversary A is

$$\text{Adv}_{\Sigma}^{\text{aead}}(A) = |2 \cdot \Pr[\text{Exp}_{\Sigma}^{\text{aead}}(A) \Rightarrow \text{true}] - 1|$$



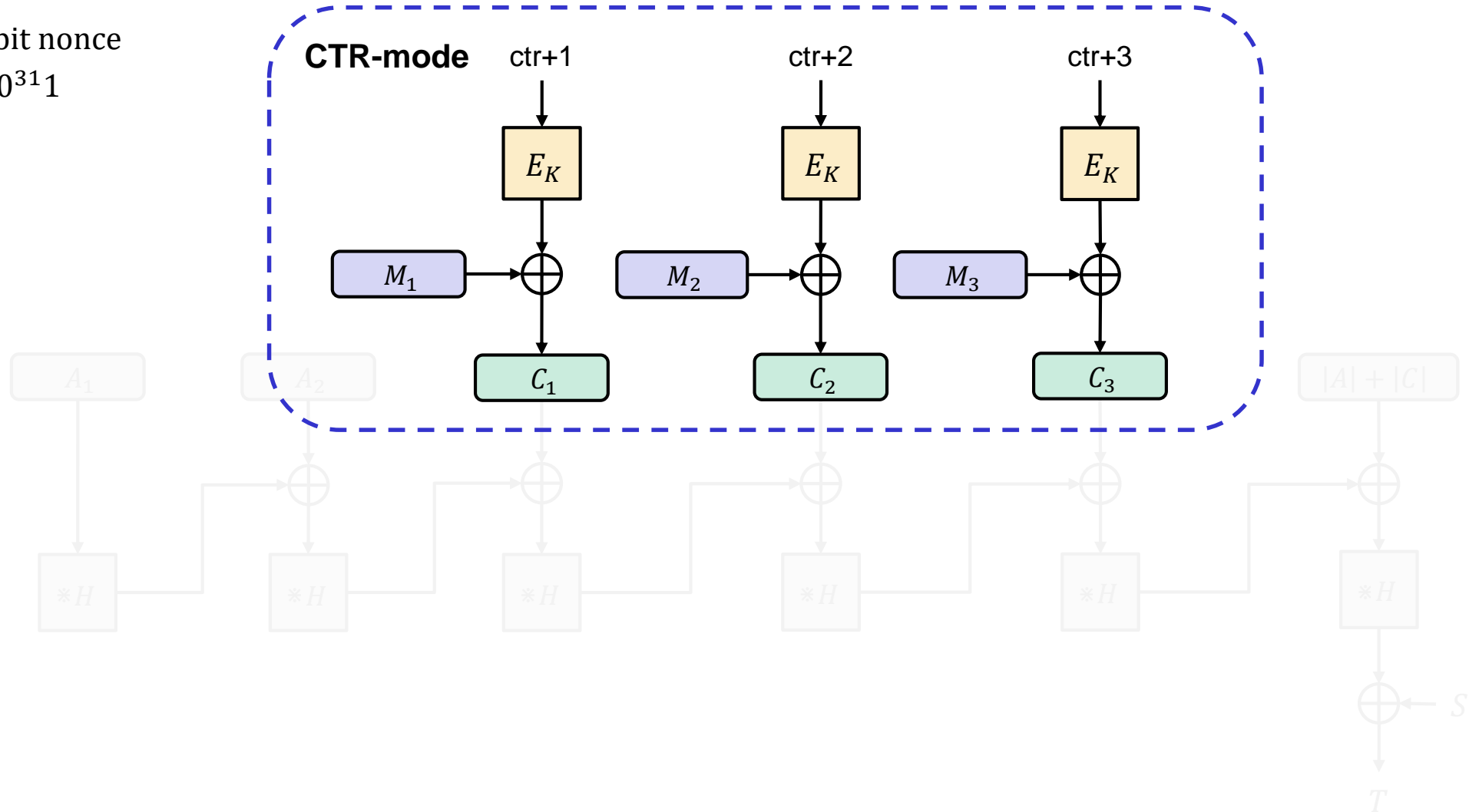
Constructing AEAD schemes

GCM – Galois/Counter Mode



GCM – Galois/Counter Mode

$N = 96$ bit nonce
 $\text{ctr} = N || 0^{31} 1$



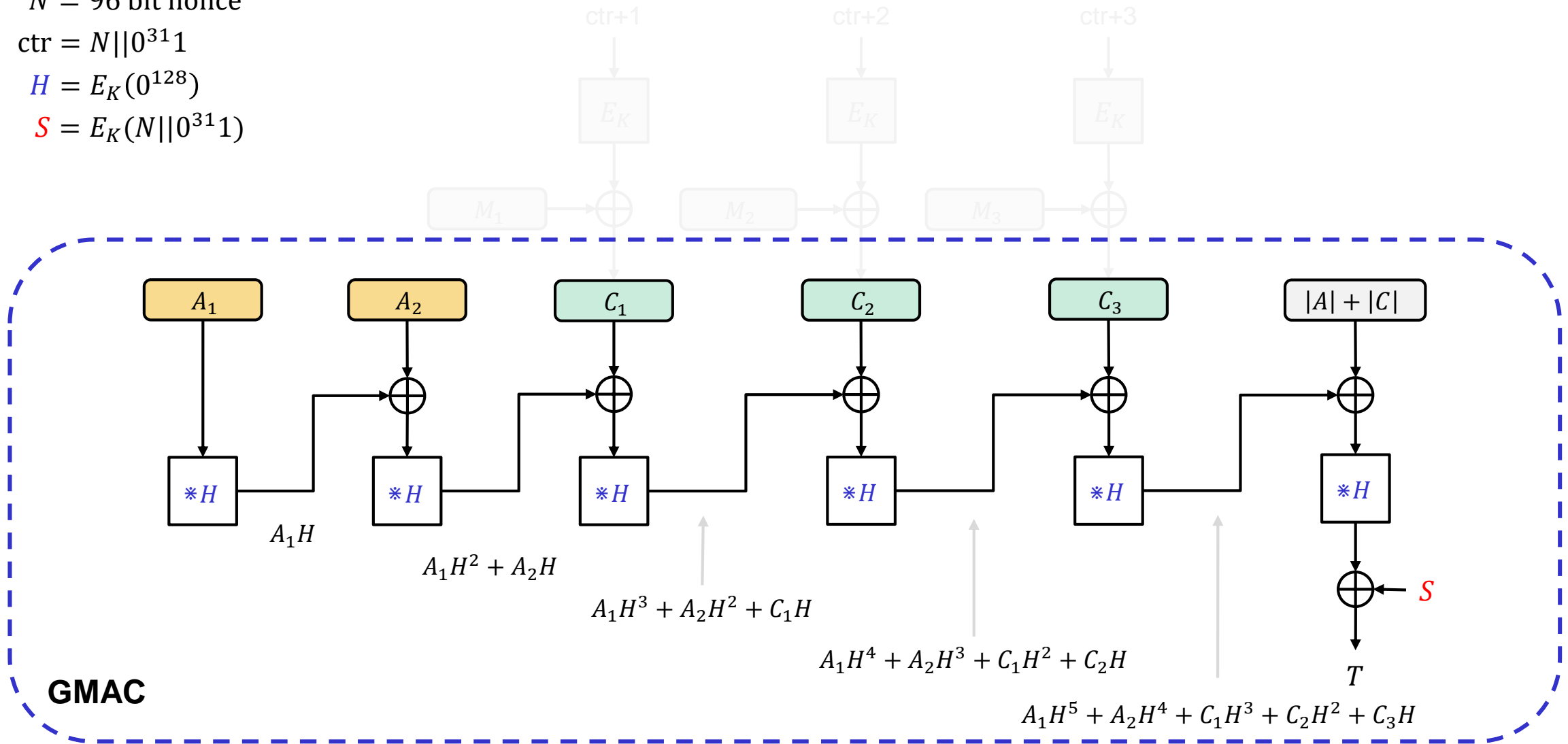
GCM – Galois/Counter Mode

$N = 96$ bit nonce

$\text{ctr} = N || 0^{31} 1$

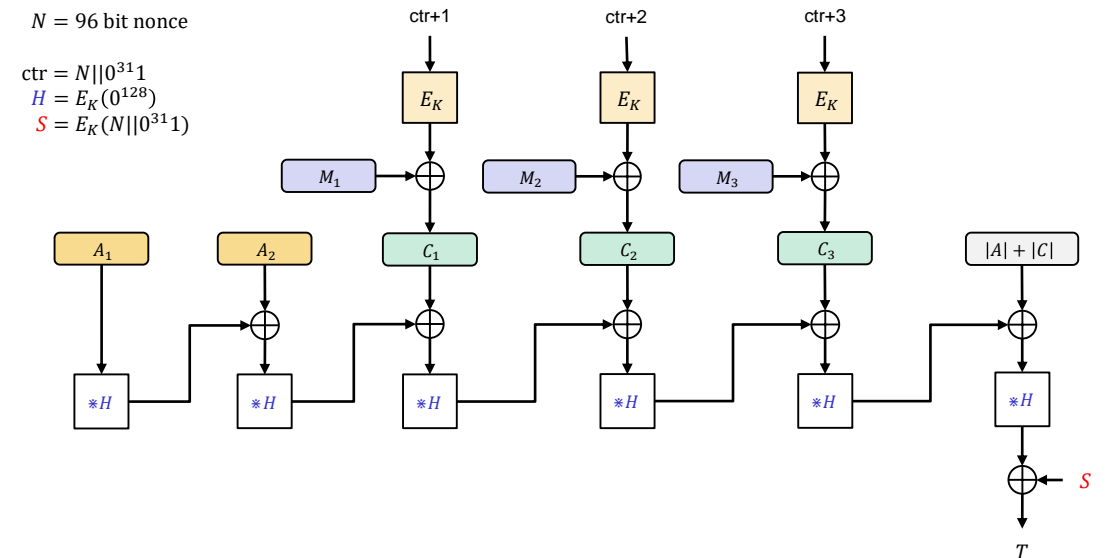
$H = E_K(0^{128})$

$S = E_K(N || 0^{31} 1)$

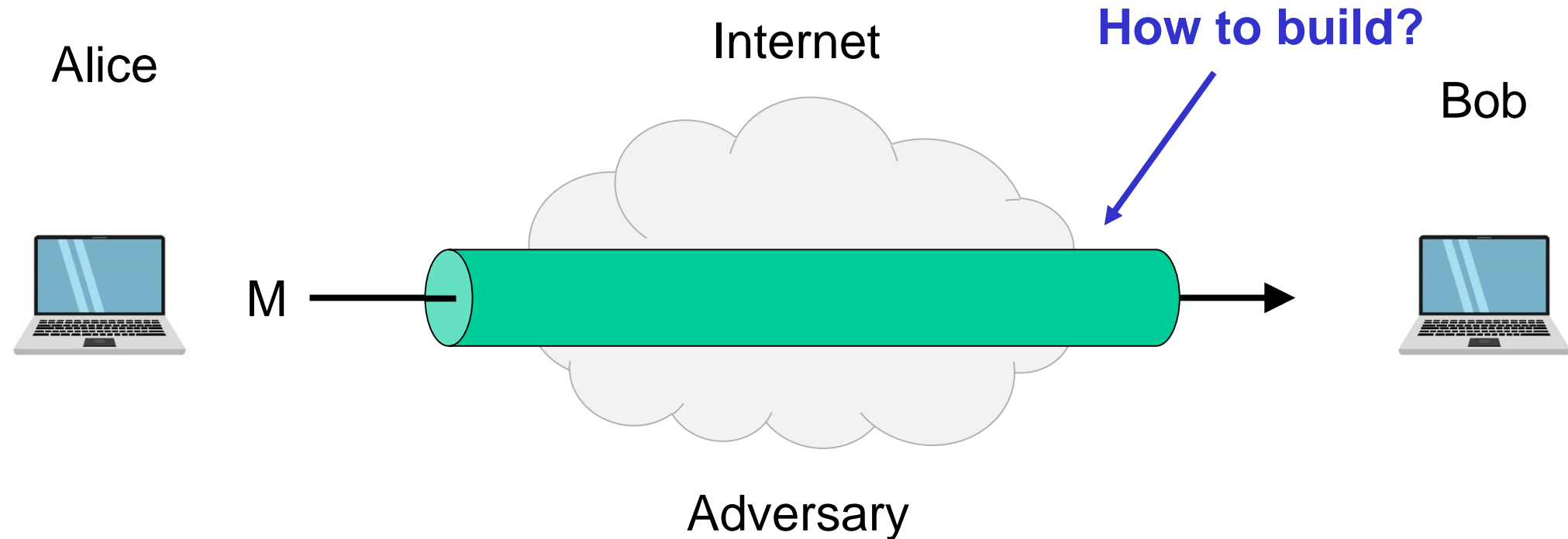


GCM – properties

- **Theorem:** AE-secure if E is a secure PRF/PRP
- Very fast
 - Especially with AES-NI and Intel PCLMULQDQ instructions
- Online
 - Doesn't need to know the length of the message before starting encryption
- Brittle
 - Nonce-reuse is *very* bad (see Problem set 5)
 - Tricky to implement correctly
- Used everywhere
 - Probably the most used mode on the Internet



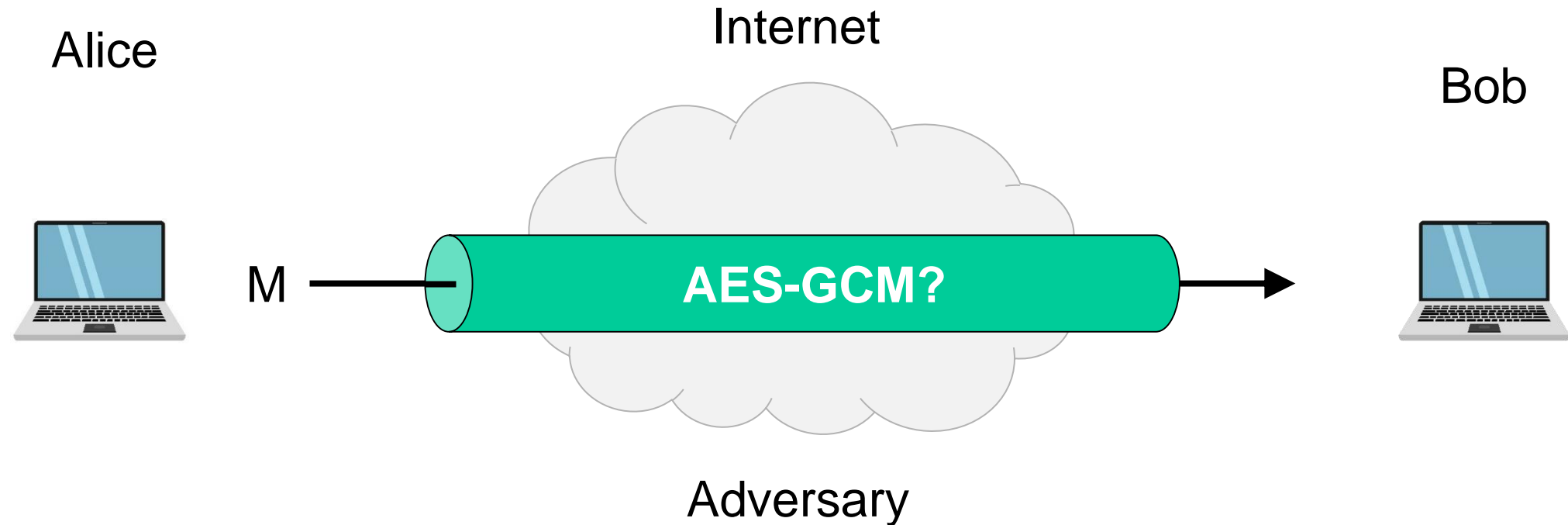
Ideal solution: secure channels



Security goals:

- **Data privacy:** adversary should not be able to read message M ✓
- **Data integrity:** adversary should not be able to modify message M ✓
- **Data authenticity:** message M really originated from Alice ✓

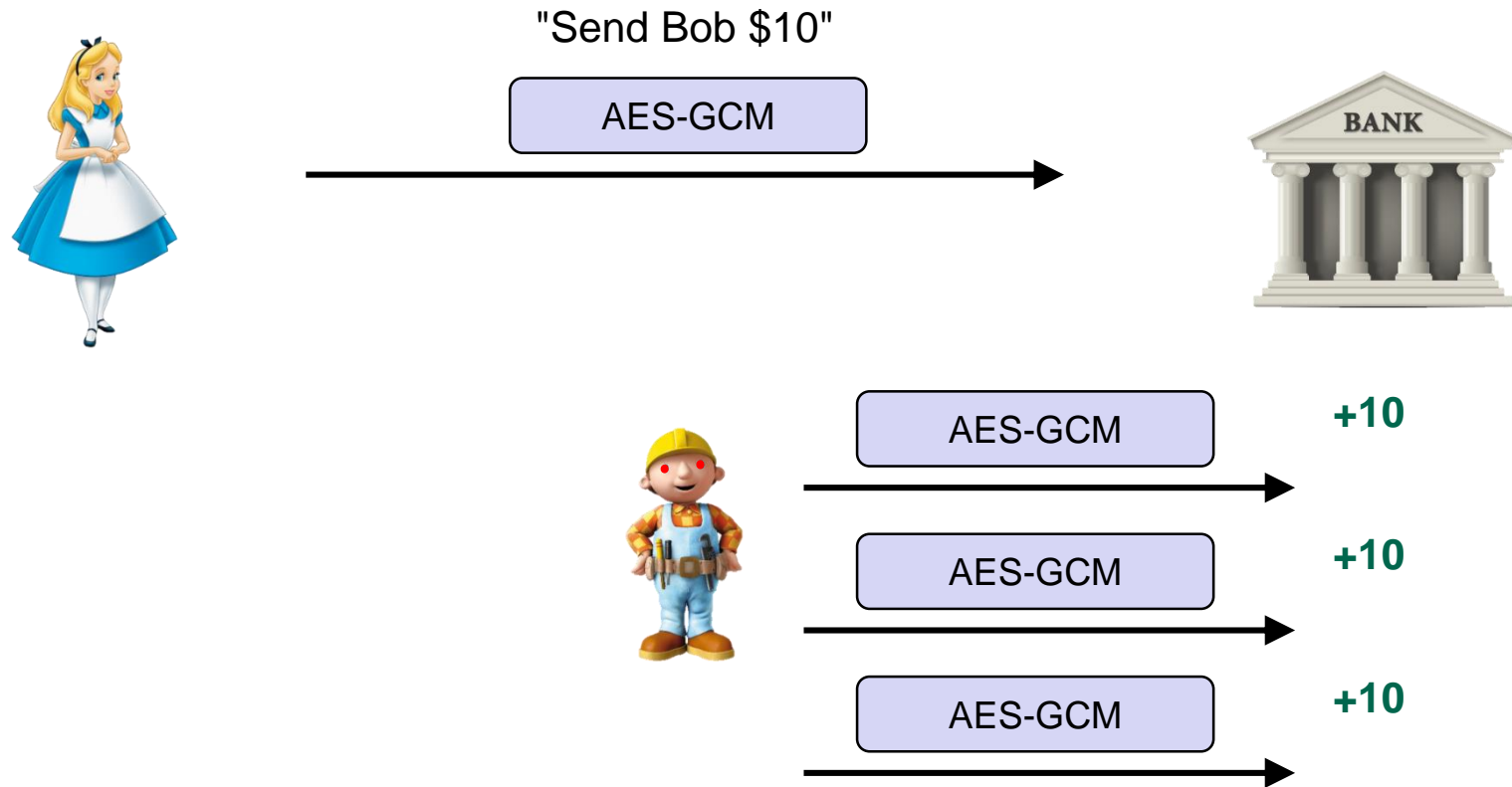
Ideal solution: secure channels



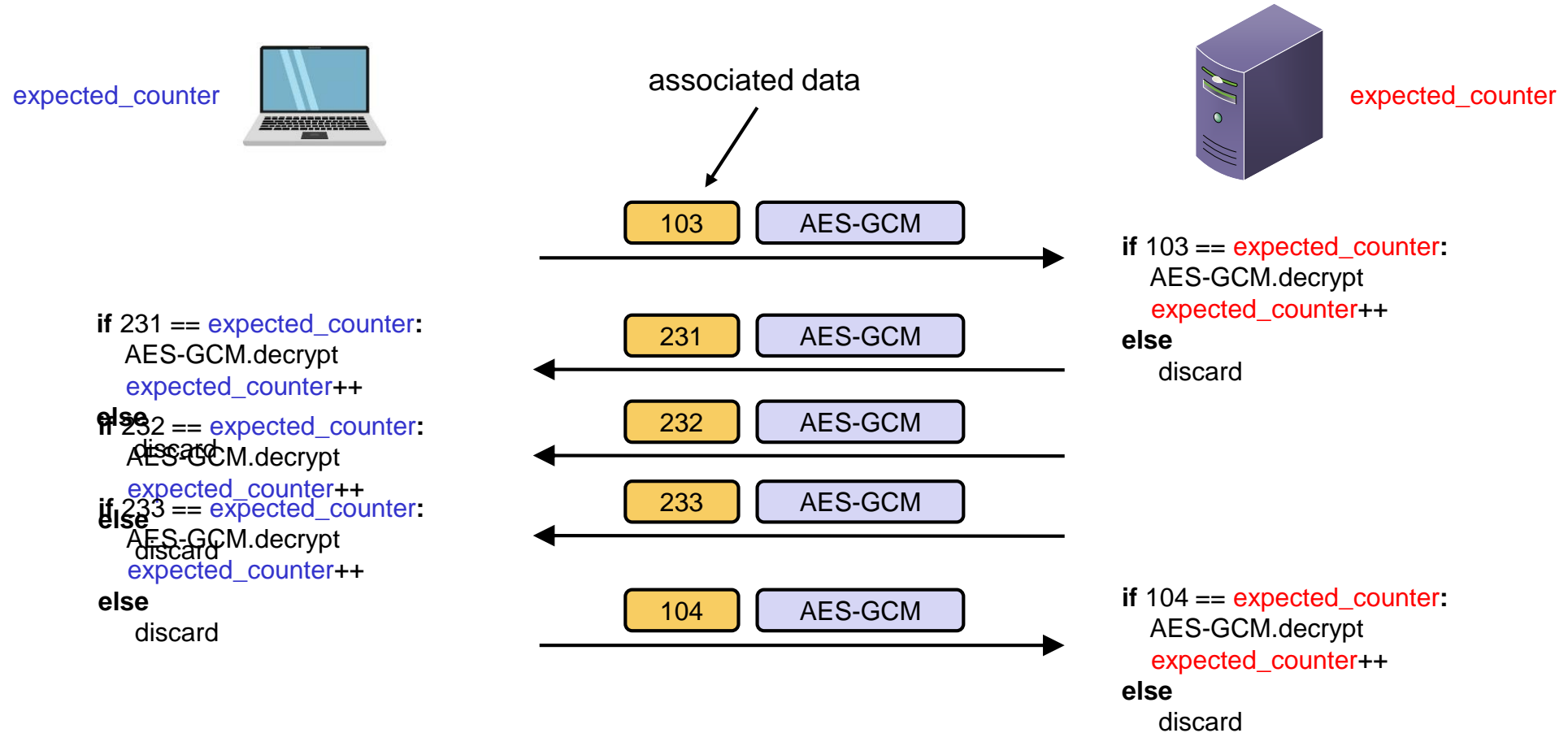
Security goals:

- **Data privacy:** adversary should not be able to read message M ✓
- **Data integrity:** adversary should not be able to modify message M ✓
- **Data authenticity:** message M really originated from Alice ✓

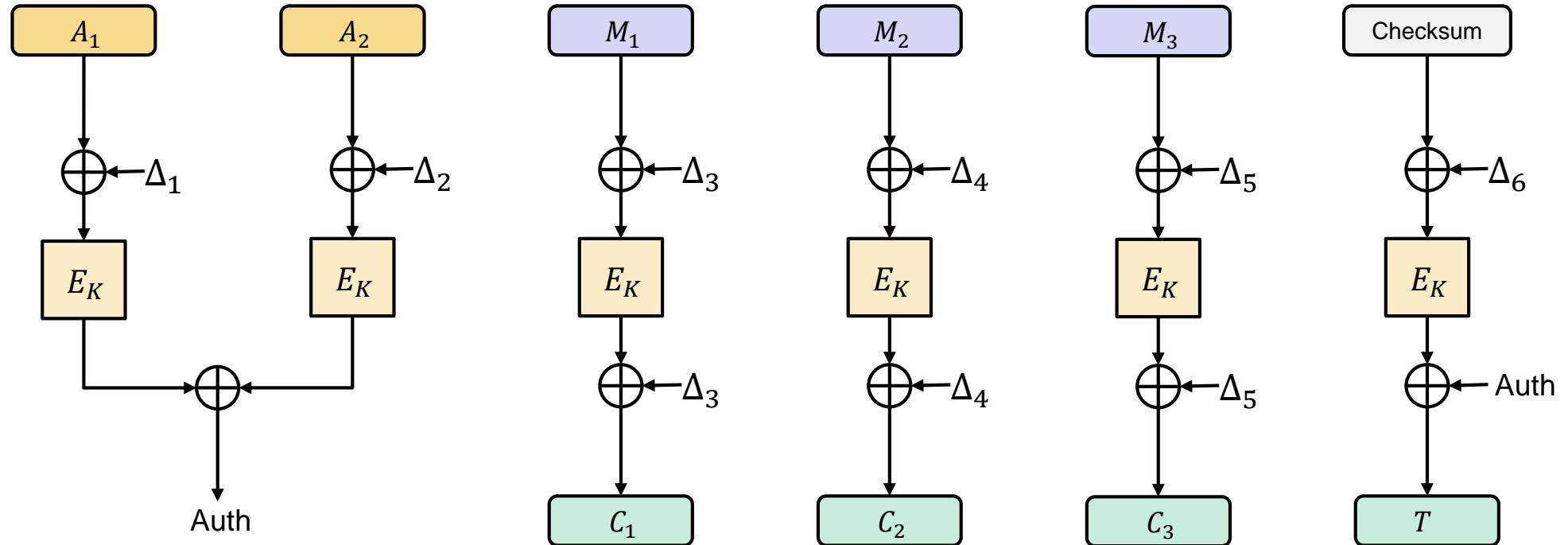
AES-GCM = secure channel?



Secure channels – TLS / IPsec



OCBv3 – Offset Codebook Mode



Δ_i - derived from 96-bit nonce N

$$\text{Checksum} = M_1 \oplus M_2 \oplus M_3$$

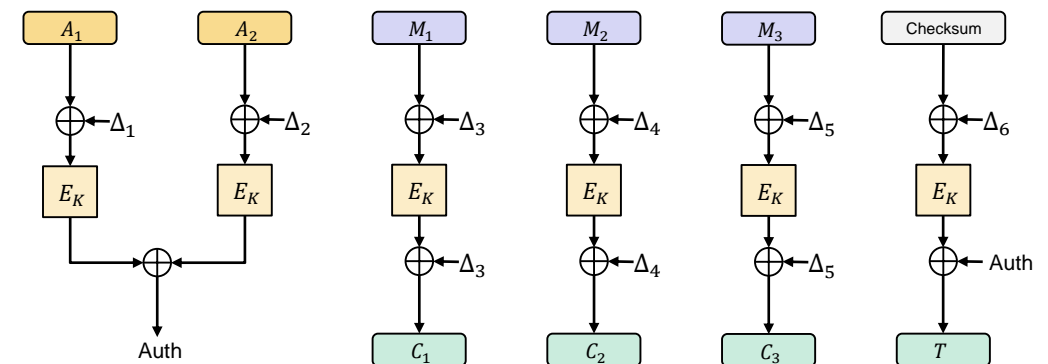
OCBv3 – properties

- **Theorem:** AE secure if E is a secure PRF
- The fastest AEAD algorithm in the west
- Fully parallelizable and online
- Incremental
- Patented by Rogaway
 - Hardly used anywhere

“ If OCB was your kid, he’d play three sports and be on his way to Harvard. You’d brag about him to all your friends. ”

“ Unfortunately OCB is *not* your kid. It belongs to Philip Rogaway, who also happens to hold a patent on it.”

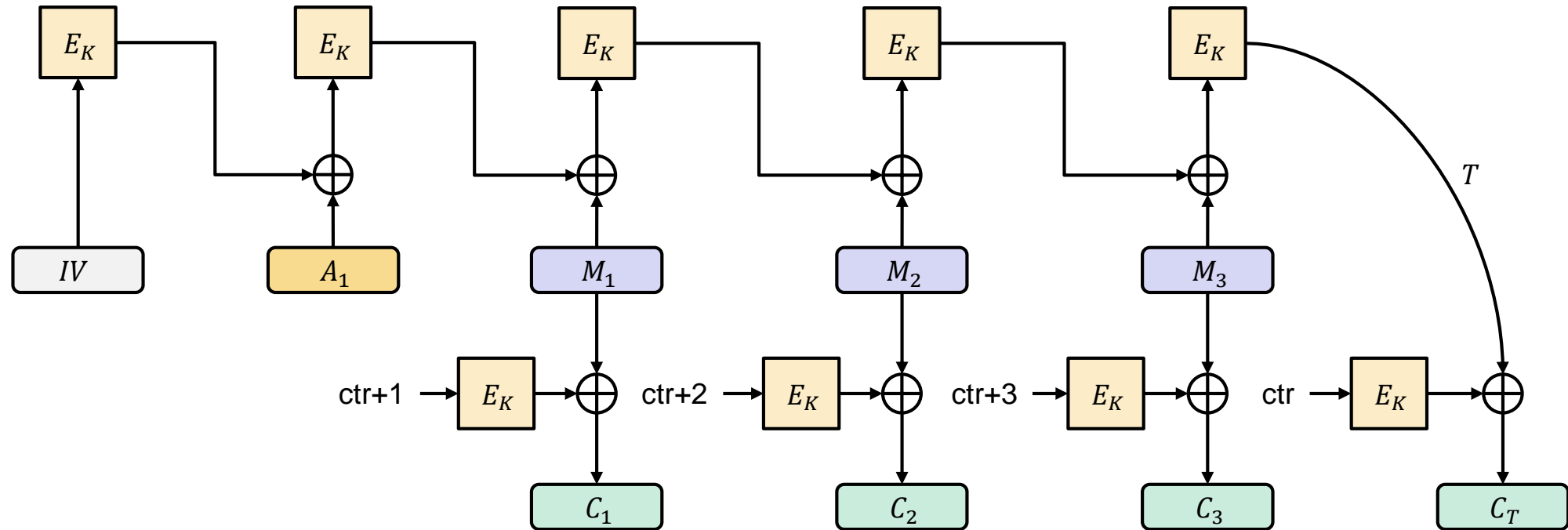
Matthew Green



Δ_i - derived from 96-bit nonce N

Checksum = $M_1 \oplus M_2 \oplus M_3$

CCM – Counter Mode with CBC-MAC

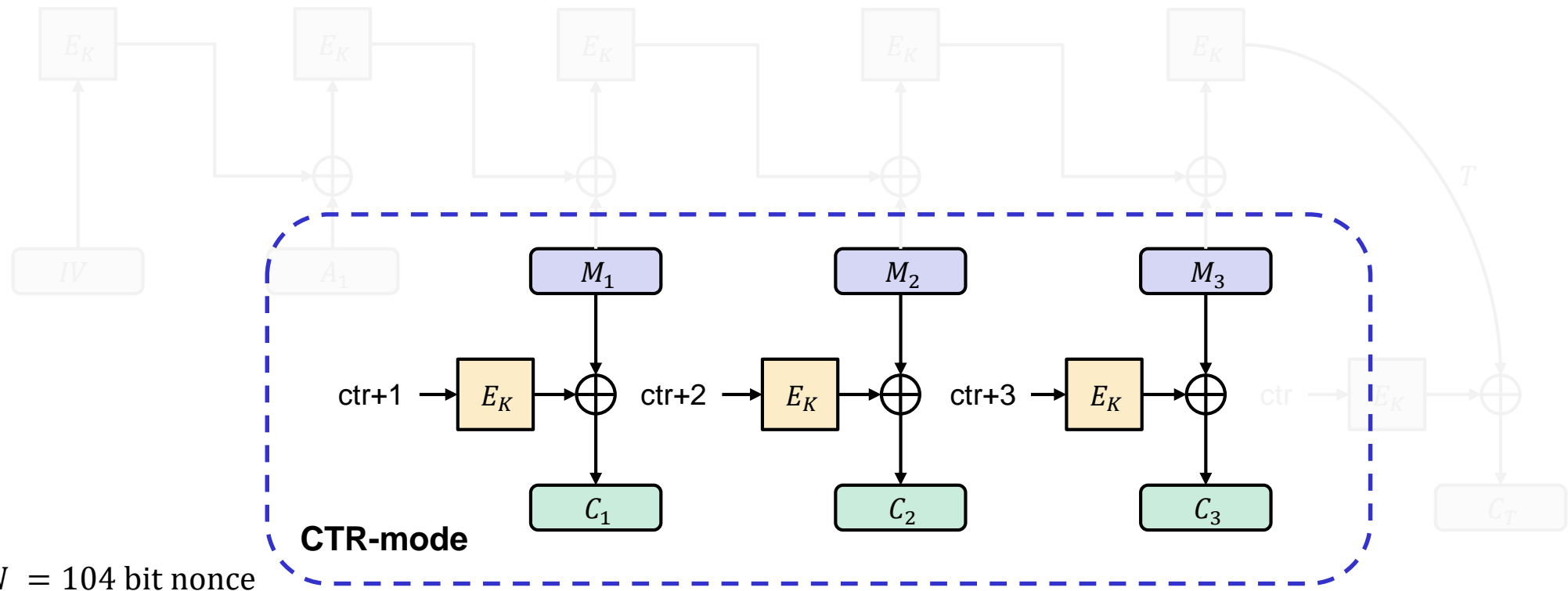


$N = 104$ bit nonce

$IV = 0^8 || N || \text{len}_{16}(AD + M)$

$ctr = 1^8 || N || 0^{16}$

CCM – Counter Mode with CBC-MAC

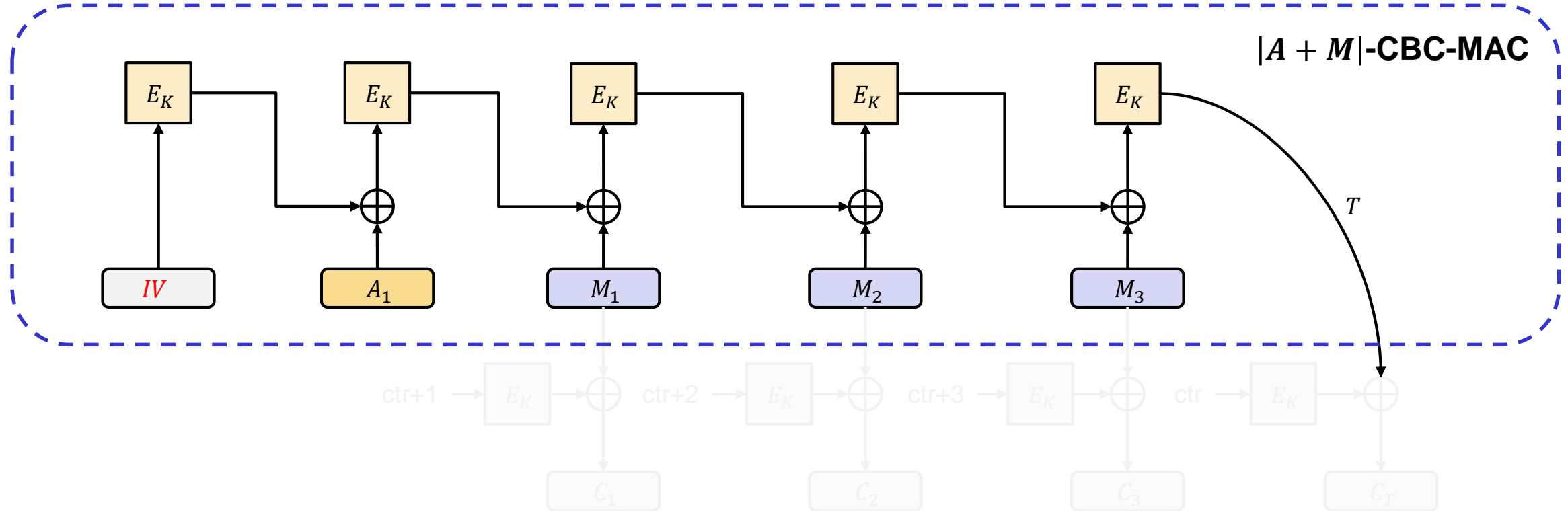


$N = 104$ bit nonce

$$IV = 0^8 || N || \text{len}_{16}(AD + M)$$

$$ctr = 1^8 || N || 0^{16}$$

CCM – Counter Mode with CBC-MAC

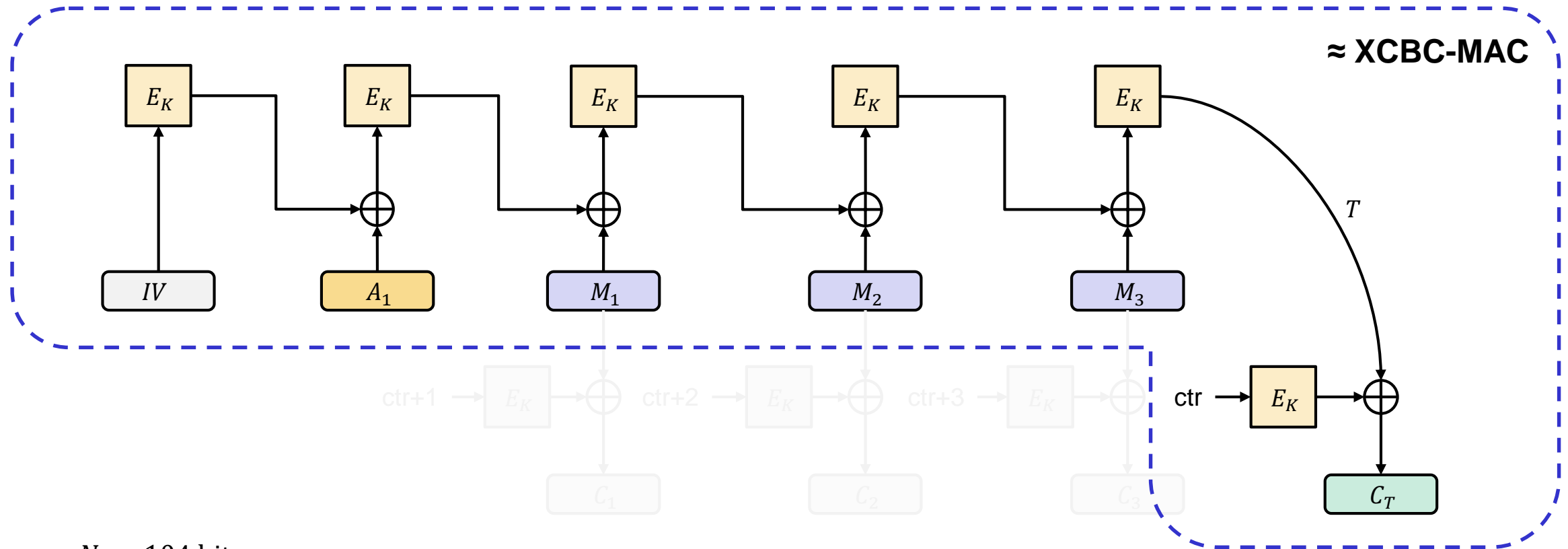


$N = 104$ bit nonce

$IV = 0^8 || N || \text{len}_{16}(AD + M)$

$ctr = 1^8 || N || 0^{16}$

CCM – Counter Mode with CBC-MAC



$N = 104$ bit nonce

$IV = 0^8 || N || \text{len}_{16}(AD + M)$

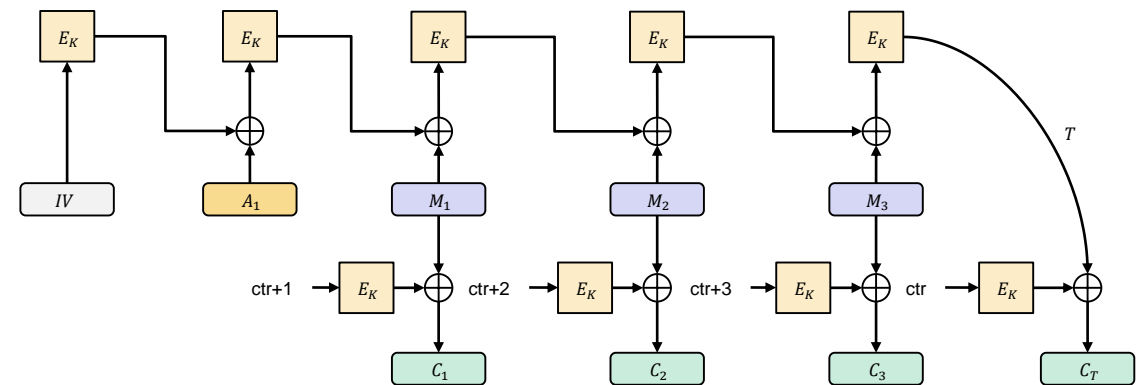
$ctr = 1^8 || N || 0^{16}$

CCM – properties

- Slow; needs 2 block cipher calls per message block
- Sequential; not online
- Clunky message encoding
- Royalty-free
 - Designed specifically as an alternative to OCB for use in WiFi (WPA2)
- Widely used
 - Default encryption algorithm in WPA2

“ CCM is the 1989 Volvo station wagon of AEAD modes. It'll get you to your destination reliably, just not in a hurry. ”

Matthew Green



$N = 104$ bit nonce

$IV = 0^8 || N || \text{len}_{16}(AD + M)$

$ctr = 1^8 || N || 0^{16}$

Summary

- Authenticated encryption: privacy + integrity in one primitive
- AEAD: AE + *associated* data: data that gets integrity protection, but not privacy protection (e.g. protocol headers)
- AEAD examples:
 - GCM
 - CCM
 - OCB
- AEAD is *not* a secure channel!
 - Does not provide *replay* protection
 - Secure channels from AEAD: add counters/nonces/timers
- Next week: hash functions