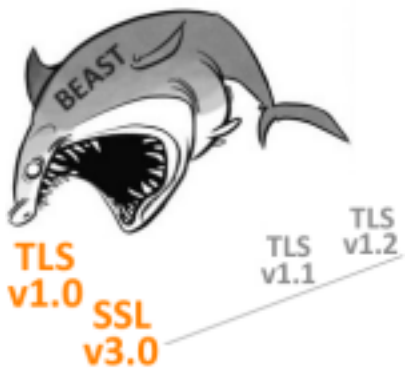
Netzsicherheit 2
3. Angriffe auf TLS

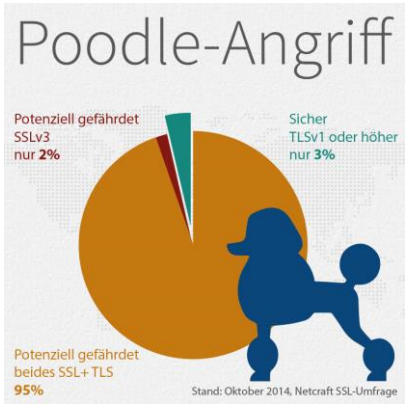
Prof. Dr. Jörg Schwenk

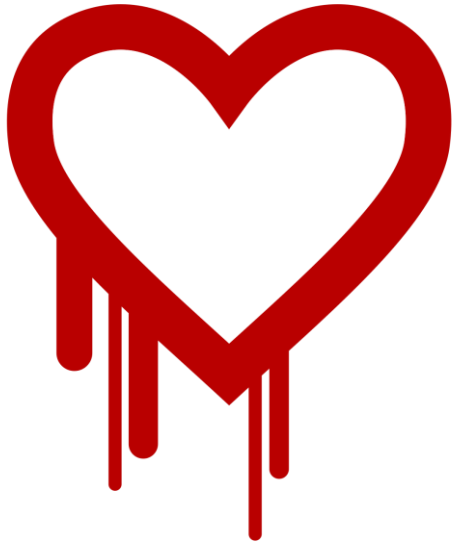
www.nds.rub.de

Angriffe auf TLS



Bleichenbacher





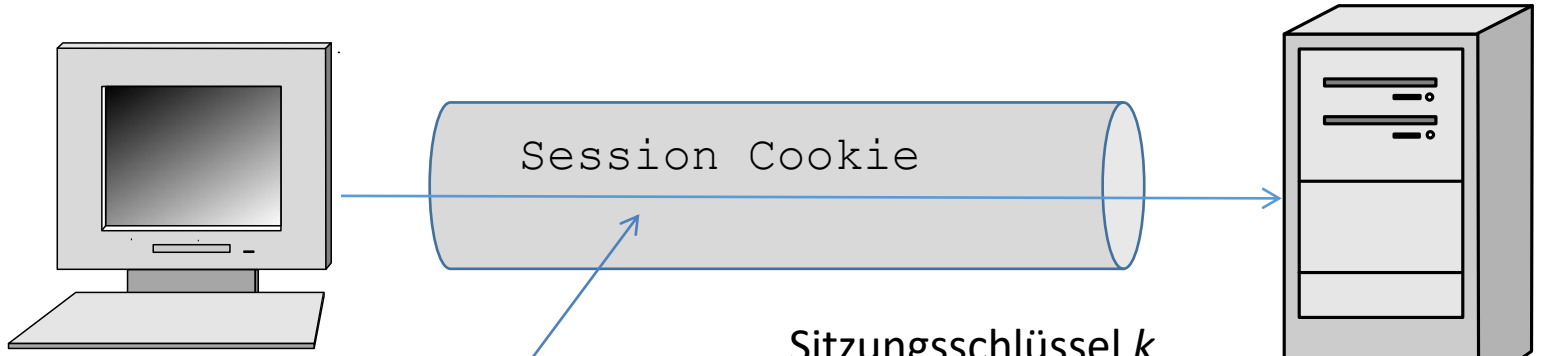
Lucky13

Invalid Curve

3.1 Übersicht

Angriffsziele bei TLS

Angriffe auf Zertifikate und PKI
z.B. Diginotar (2011)



privater Schlüssel sk_S des Servers

Sitzungsschlüssel k

Angriffe auf GUI
z.B. SSLStrip

BEAST (2010)
CRIME (2011)
BREACH (2013)
Lucky13 (2013)
POODLE (2014)
...

Bleichenbacher (1998)
B-Versionsnummer (2003)
B-Timing (2014)
DROWN (2016)
ROBOT (2017)
RACCOON (2021)

Timing Attacks (2003)
Heartbleed (2014)
Invalid Point Attacks (2015)

3.2 Angreifermodelle

Angreifermodell

- beschreibt die Fähigkeiten des Angreifers
- dient dazu, die Gefährlichkeit eines Angriffs einzuschätzen
- schwaches Angreifermodell <==> starker (gefährlicher) Angriff

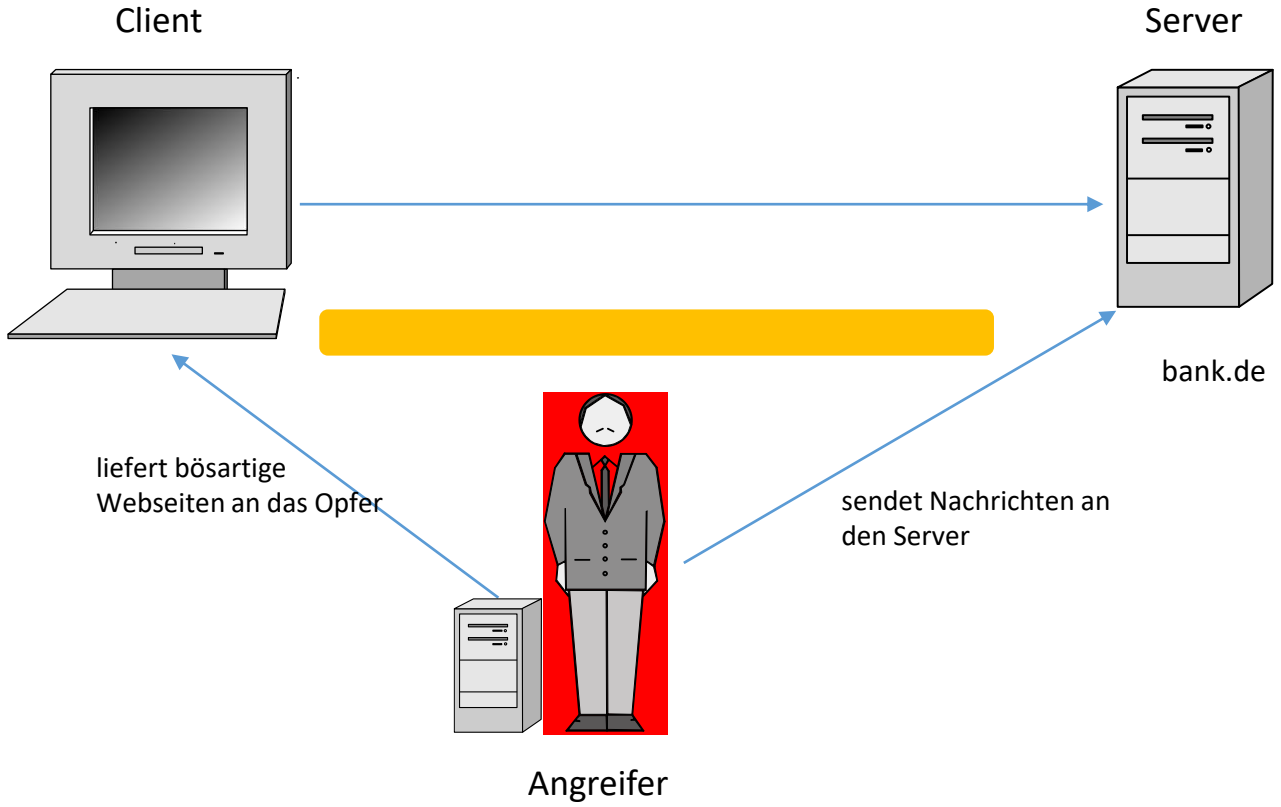
Angreifermodelle TLS

- Wir unterscheiden drei grundlegende Angreifermodelle
- ... die aber auch kombiniert werden können

Web Attacker Model

Starkes, realistisches Modell da schwache Fähigkeiten des Angreifers. Dieser darf:

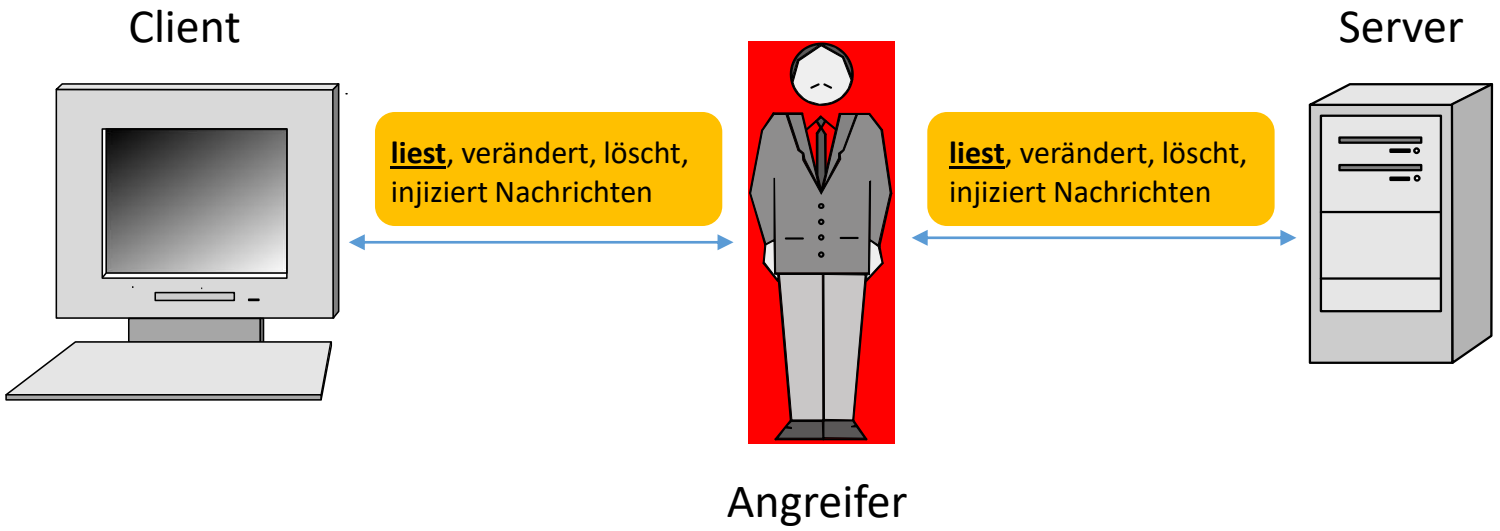
- bösartige Server im Internet betreiben
- beliebige Nachrichten (z.B. Email) senden
- Angreifer ist kein Man-in-the-Middle!



Man-in-the-Middle Attacker Model

Teilweise realistisches Modell (z.B. WLAN, Geheimdienst).
Angreifer:

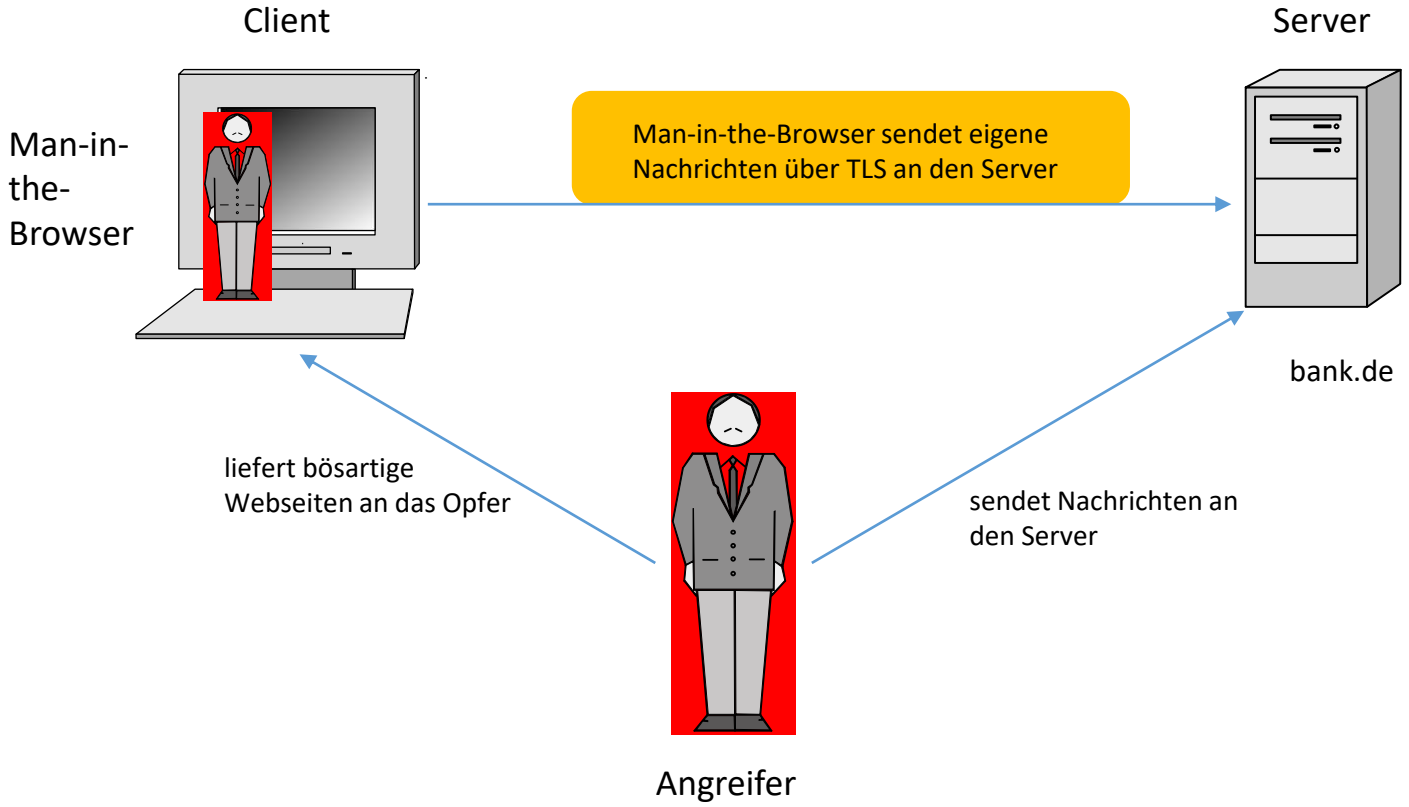
- sieht alle Nachrichten, die zwischen Client und Server ausgetauscht werden (Chiffretext!)
- kann Nachrichten verändern, löschen, hinzufügen



Man-in-the-Browser Attacker Model

Variante des Web Attacker Model für HTML/HTTPS/Javascript. Dieser darf:

- bösartige Server im Internet betreiben
- beliebige Nachrichten (z.B. Email) senden
- Über JavaScript beliebige HTTP-Requests vom Client an den Server senden



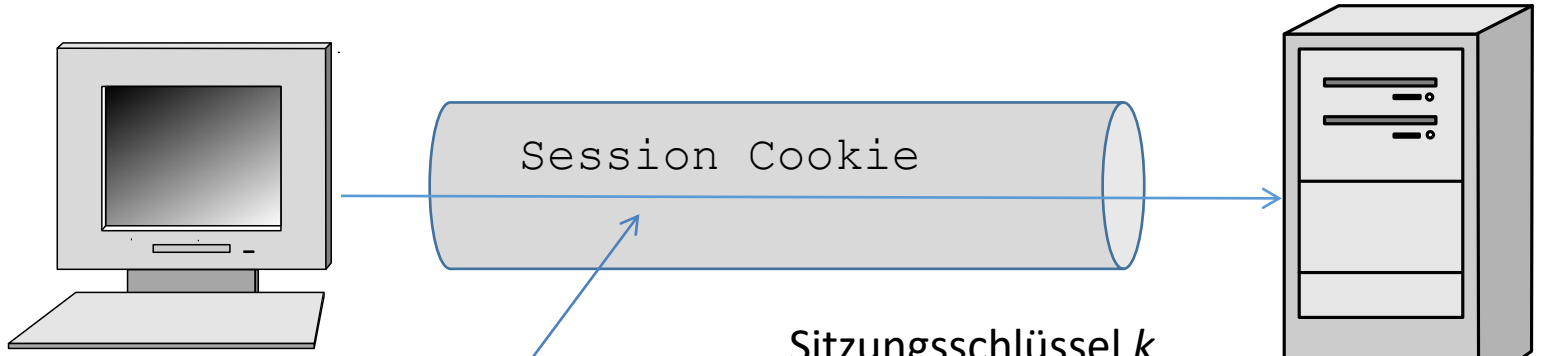
Fazit

Angreifermodell	Fähigkeiten des Angreifers	Gefährlichkeit des Angriffs
Web Attacker (schwach)	realistisch	hoch
Man-in-the-Middle (stark)	stark, nur teilweise realistisch	mittel (hoch in bestimmten Situationen, z.B. WLAN)
Man-in-the-Browser (schwach)	realistisch für HTTPS	hoch

3.3 Angriffe auf den Record Layer

Angriffsziele bei TLS

Angriffe auf Zertifikate und PKI
z.B. Diginotar (2011)



privater Schlüssel sk_S des Servers

Sitzungsschlüssel k

Angriffe auf GUI
z.B. SSLStrip

BEAST (2010)
CRIME (2011)
BREACH (2013)
Lucky13 (2013)
POODLE (2014)
...

Bleichenbacher (1998)
B-Versionsnummer (2003)
B-Timing (2014)
DROWN (2016)
ROBOT (2017)

Timing Attacks (2003)
Heartbleed (2014)
Invalid Point Attacks (2015)

3.3 Angriffe auf den Record Layer

3.3.1 Wörterbuch aus Chiffretextlängen

Wörterbuch aus Chiffretextlängen

[CWWZ10] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10. IEEE Computer Society, May 2010.

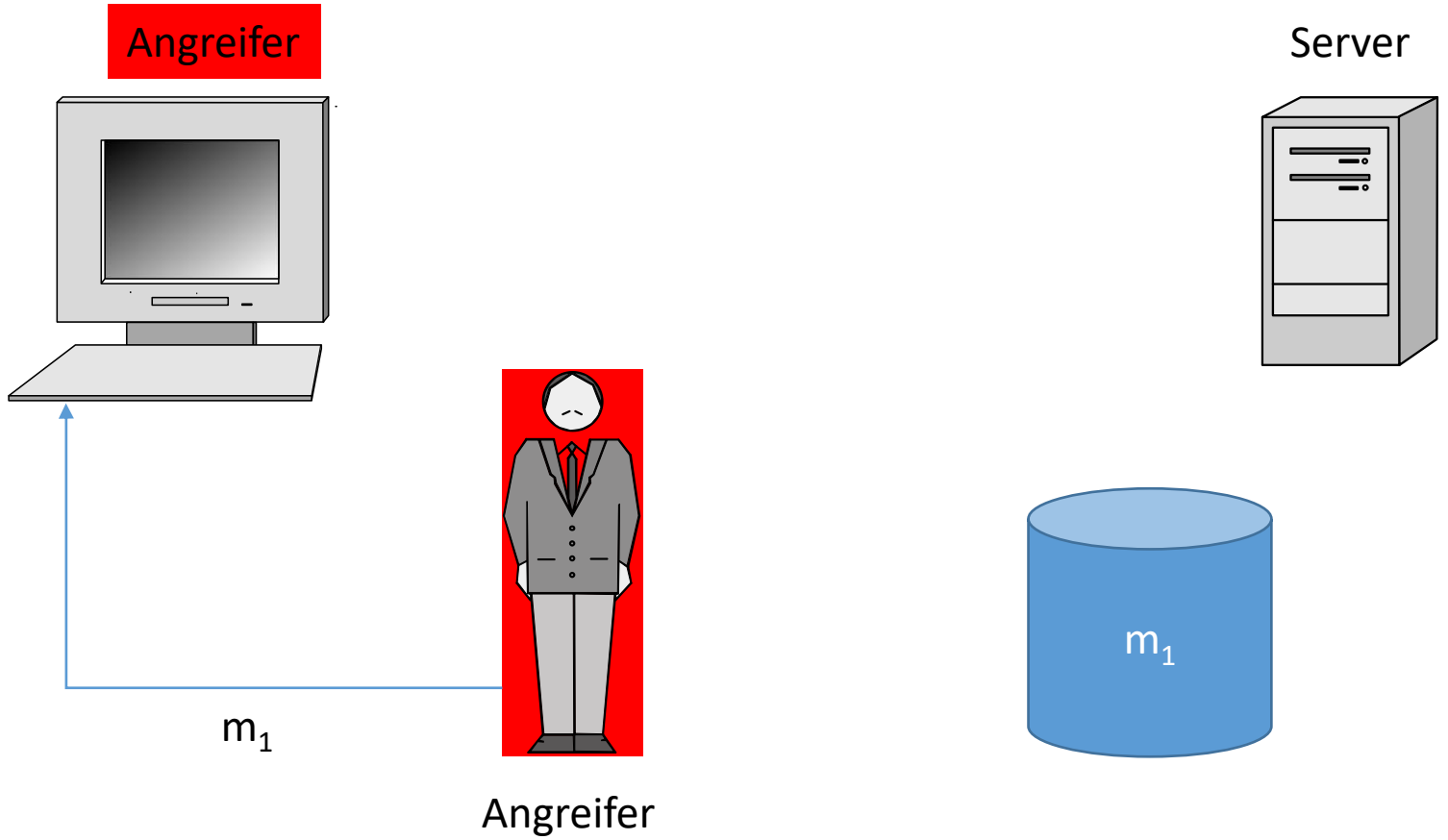
Beispiel: Online-Apotheke, per HTTPS geschützt

- Kaum Texteingabe
- Auswahl von Medikament sendet Anfrage fester Länge an Server und erhält Antwort(en) fester Länge (Text, Bilder)
- Angreifer
  1. besucht Apotheke und navigiert in sensiblen Bereich (z.B. chronische Krankheiten wie AIDS und Diabetes)
  2. zeichnet dabei die Länge des gesendeten und empfangenen Chiffretexte auf
  3. da in jedem Navigationspfad andere Daten abgerufen werden, sind Anzahl und Länge der in jede Richtung übertragenen Chiffretexte unterschiedlich

Wörterbuch aus Chiffretextlängen

Zwei Phasen

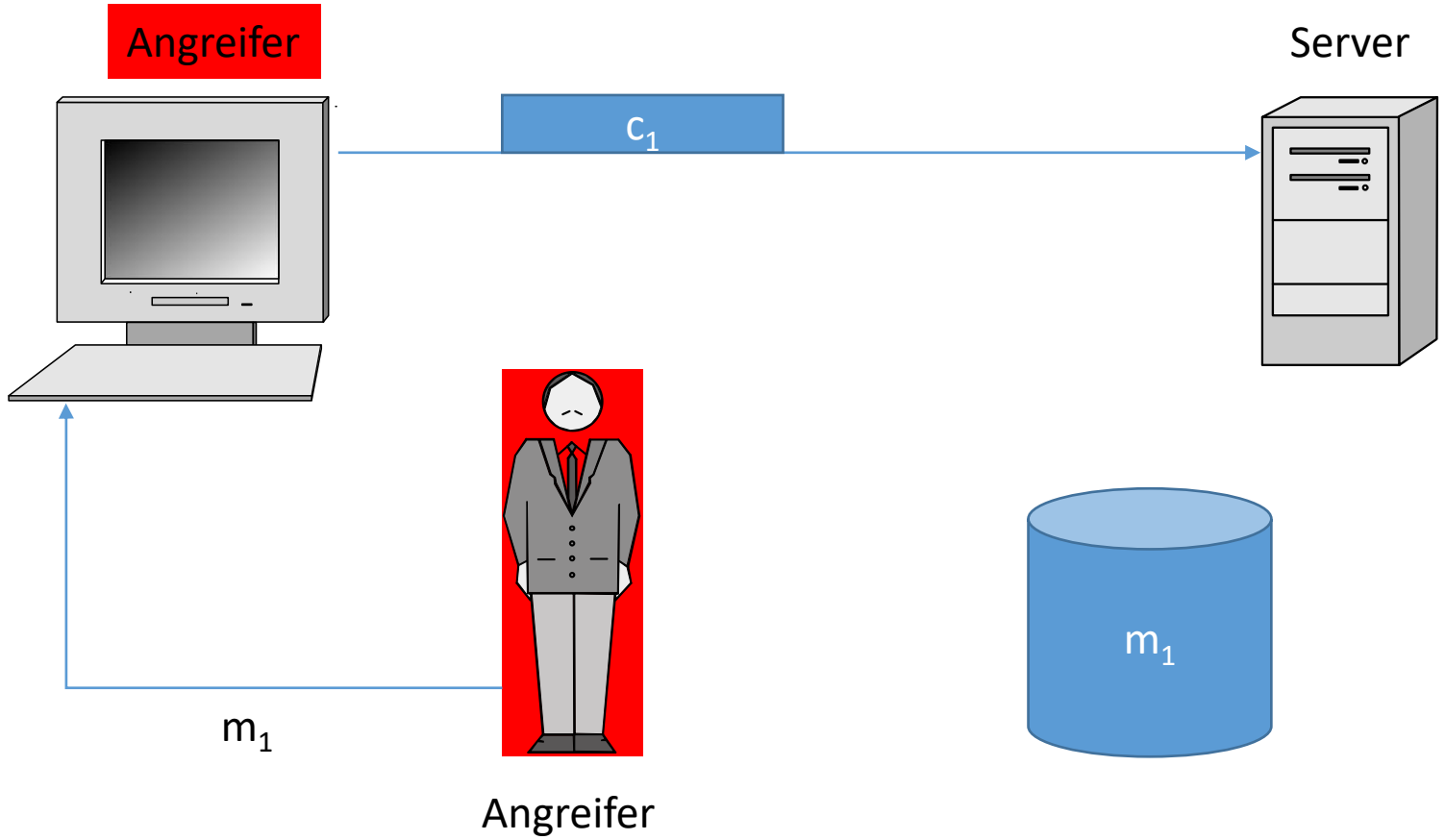
1. Angreifer nutzt die Webanwendung und zeichnet zu jedem Klartext, den er eingibt,



Wörterbuch aus Chiffretextlängen

Zwei Phasen

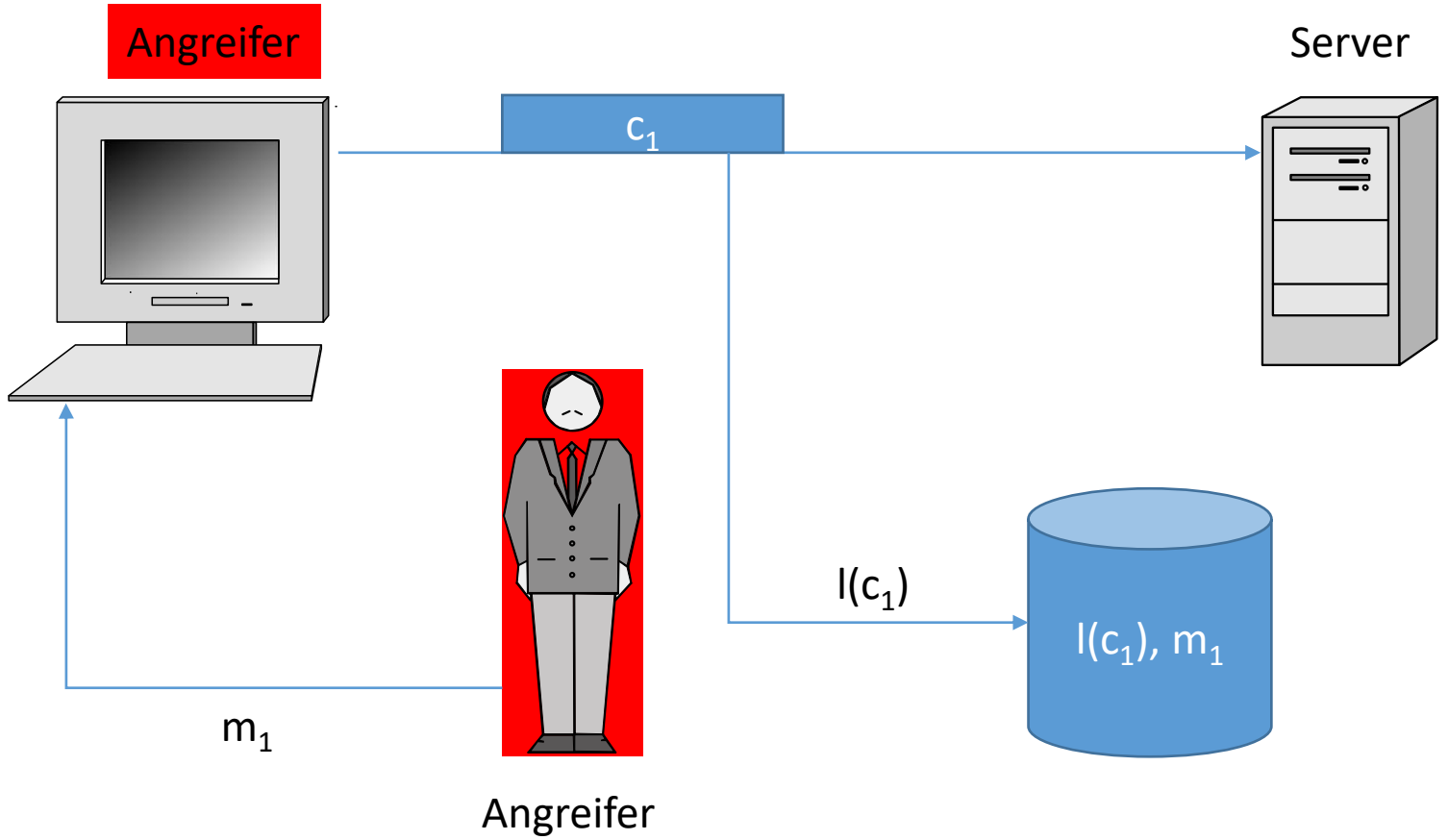
1. Angreifer nutzt die Webanwendung und zeichnet zu jedem Klartext, den er eingibt, die Chiffretextlänge auf



Wörterbuch aus Chiffretextlängen

Zwei Phasen

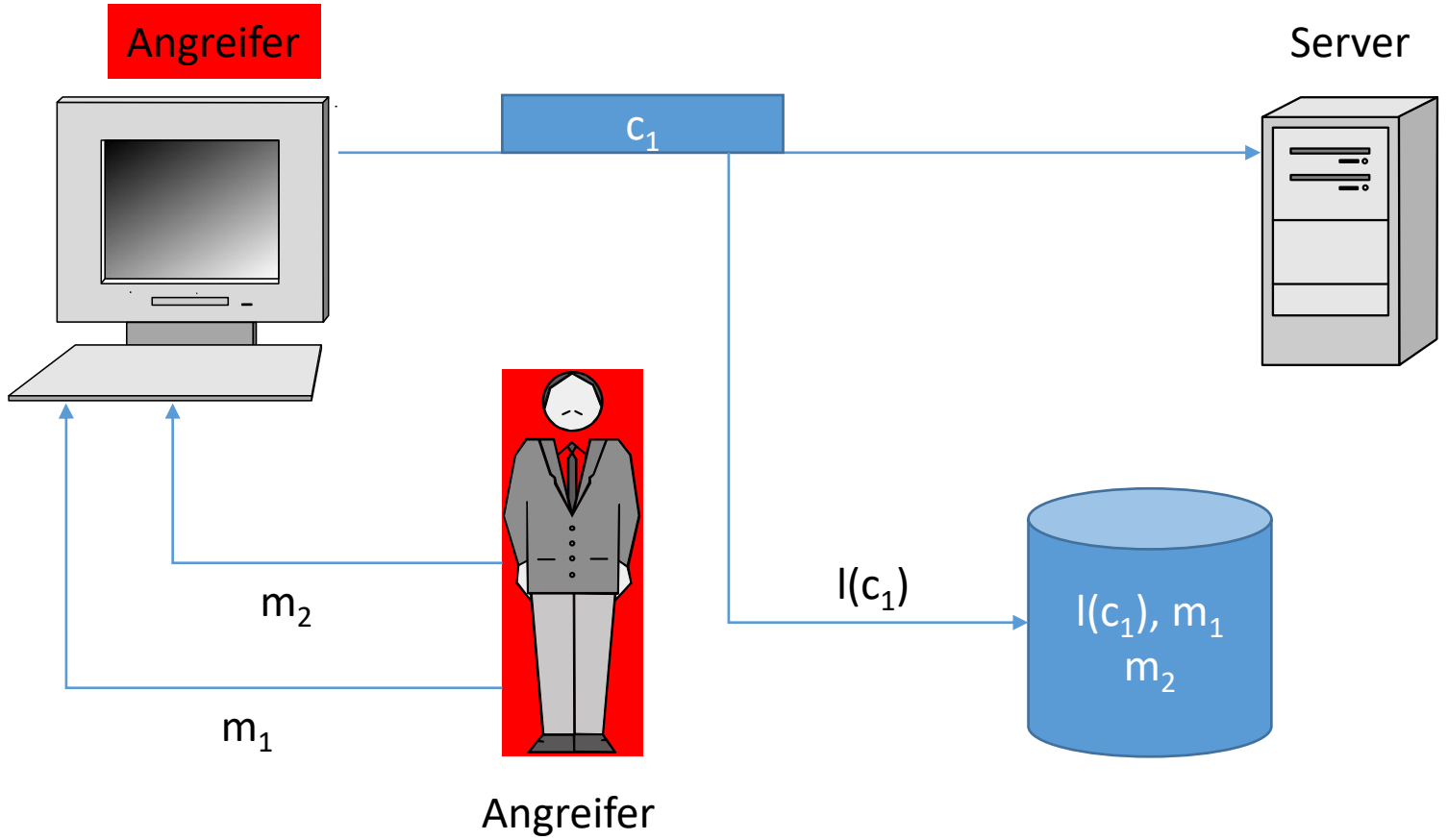
1. Angreifer nutzt die Webanwendung und zeichnet zu jedem Klartext, den er eingibt, die Chiffretextlänge auf



Wörterbuch aus Chiffretextlängen

Zwei Phasen

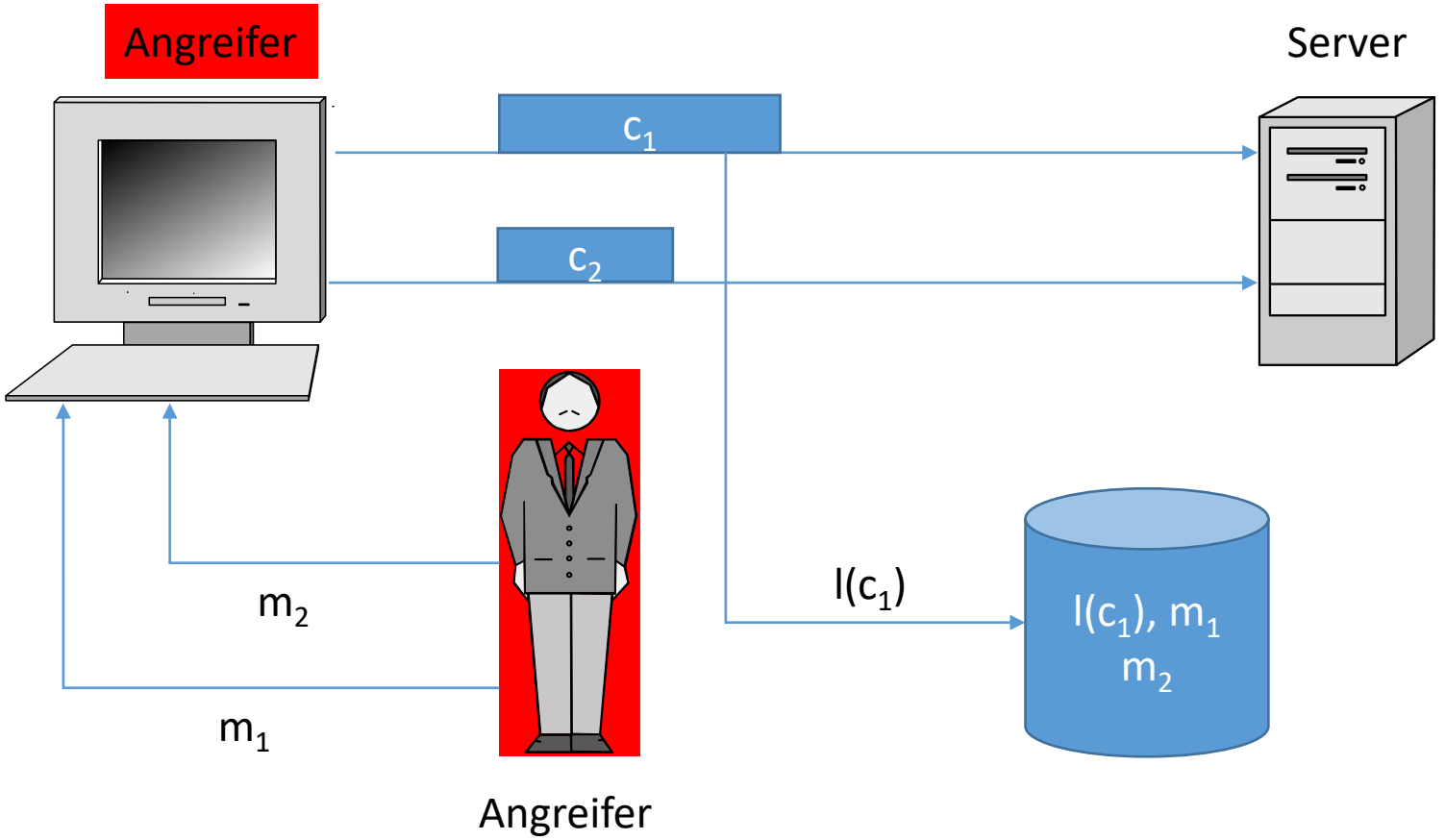
1. Angreifer nutzt die Webanwendung und zeichnet zu jedem Klartext, den er eingibt, die Chiffretextlänge auf



Wörterbuch aus Chiffretextlängen

Zwei Phasen

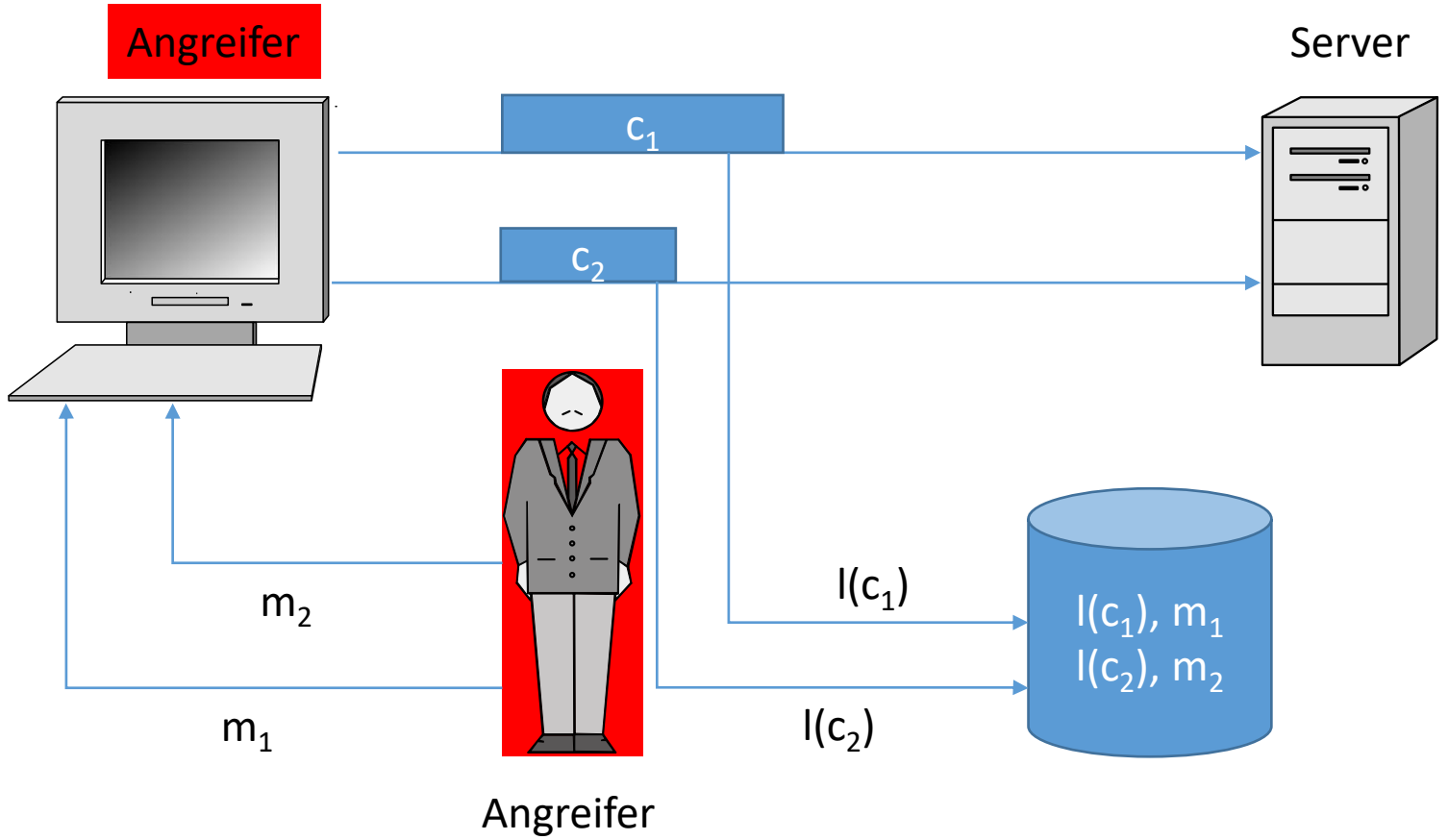
1. Angreifer nutzt die Webanwendung und zeichnet zu jedem Klartext, den er eingibt, die Chiffretextlänge auf



Wörterbuch aus Chiffretextlängen

Zwei Phasen

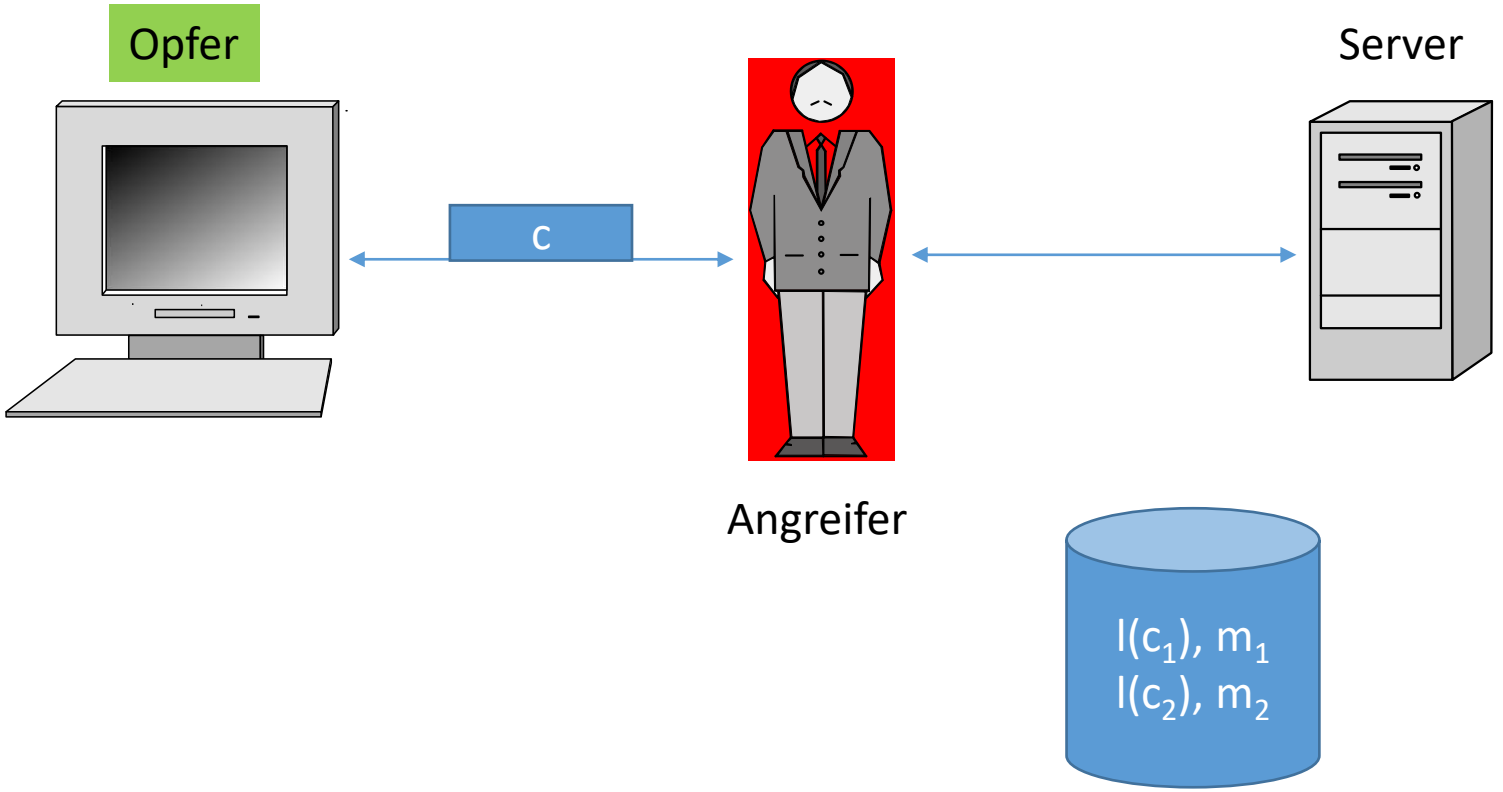
1. Angreifer nutzt die Webanwendung und zeichnet zu jedem Klartext, den er eingibt, die Chiffretextlänge auf



Wörterbuch aus Chiffretextlängen

Zwei Phasen

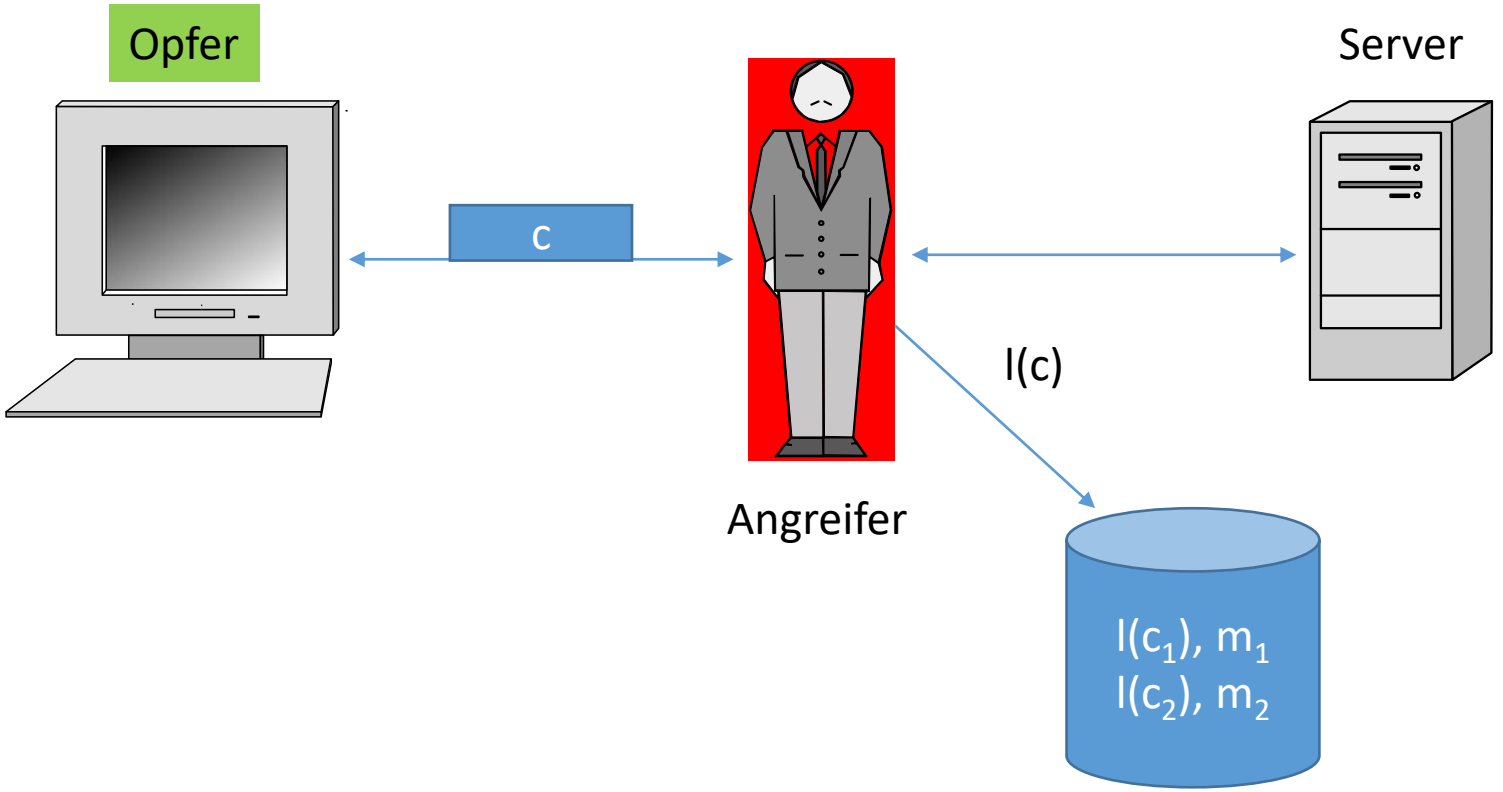
1. Angreifer nutzt die Webanwendung und zeichnet zu jedem Klartext, den er eingibt, die Chiffretextlänge auf
2. Angreifer agiert als MitM, sieht die Längen der Chiffretexte



Wörterbuch aus Chiffretextlängen

Zwei Phasen

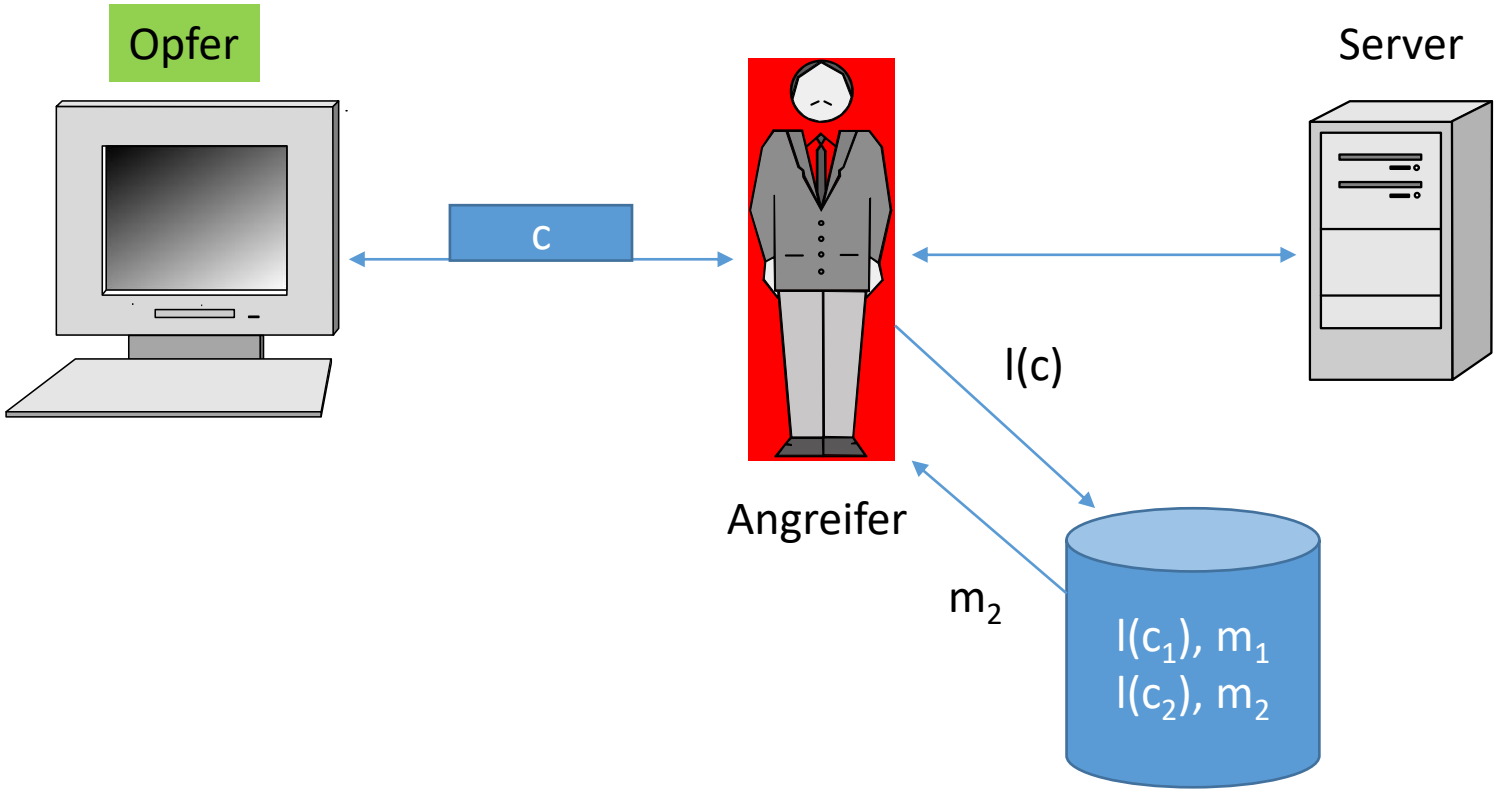
1. Angreifer nutzt die Webanwendung und zeichnet zu jedem Klartext, den er eingibt, die Chiffretextlänge auf
2. Angreifer agiert als MitM, sieht die Längen der Chiffretexte und übersetzt diese mithilfe des Wörterbuch in Klartexte



Wörterbuch aus Chiffretextlängen

Zwei Phasen

1. Angreifer nutzt die Webanwendung und zeichnet zu jedem Klartext, den er eingibt, die Chiffretextlänge auf
2. Angreifer agiert als MitM, sieht die Längen der Chiffretexte und übersetzt diese mithilfe des Wörterbuch in Klartexte



Wörterbuch aus Chiffretextlängen: Fazit

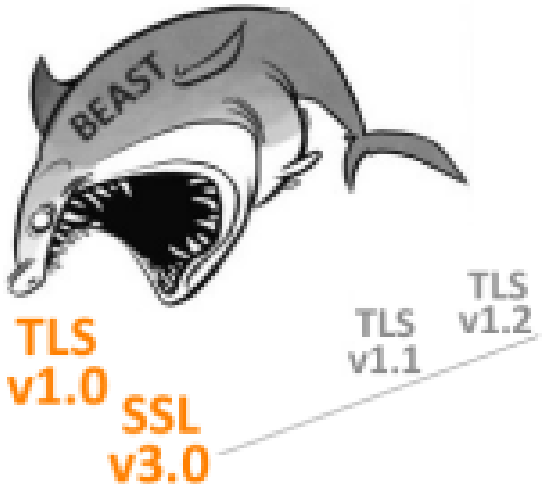
- Vom Nutzer gewählte Navigationspfade in Webanwendungen produzieren Chiffretexte unterschiedlicher Anzahl und Länge
- Durch Beobachtung der Anzahl und Länge der Chiffretexte kann die Navigation nachvollzogen werden

- Gegenmaßnahme: TLS-Recordlängen vereinheitlichen durch Padding

3.3 Angriffe auf den Record Layer

3.3.2 BEAST

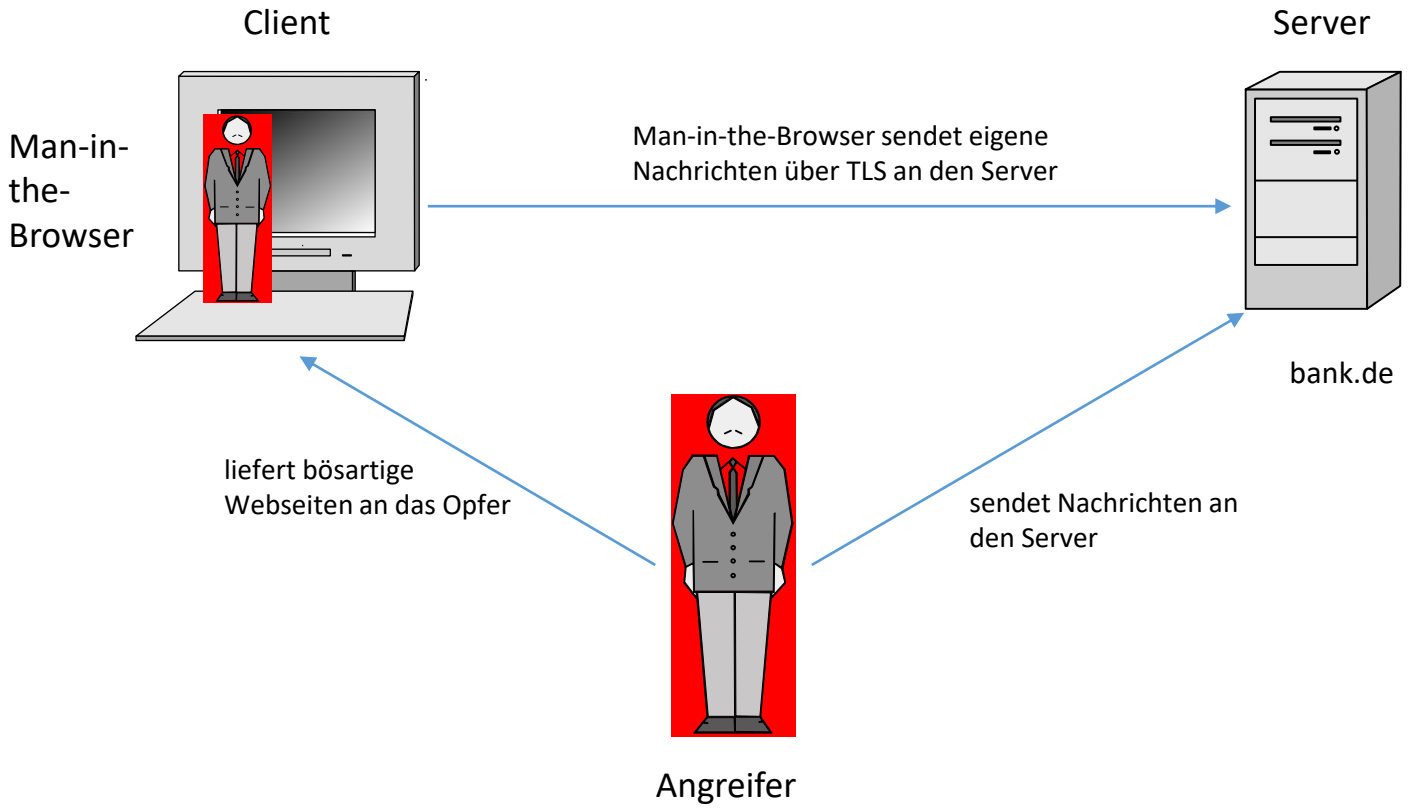
BEAST



[RD11] Juliano Rizzo and Thai Duong. Here Come The XOR Ninjas. `https://nerdoholic.org/uploads/dergln/beast_part2/ssl_jun21.pdf`, May 2011.

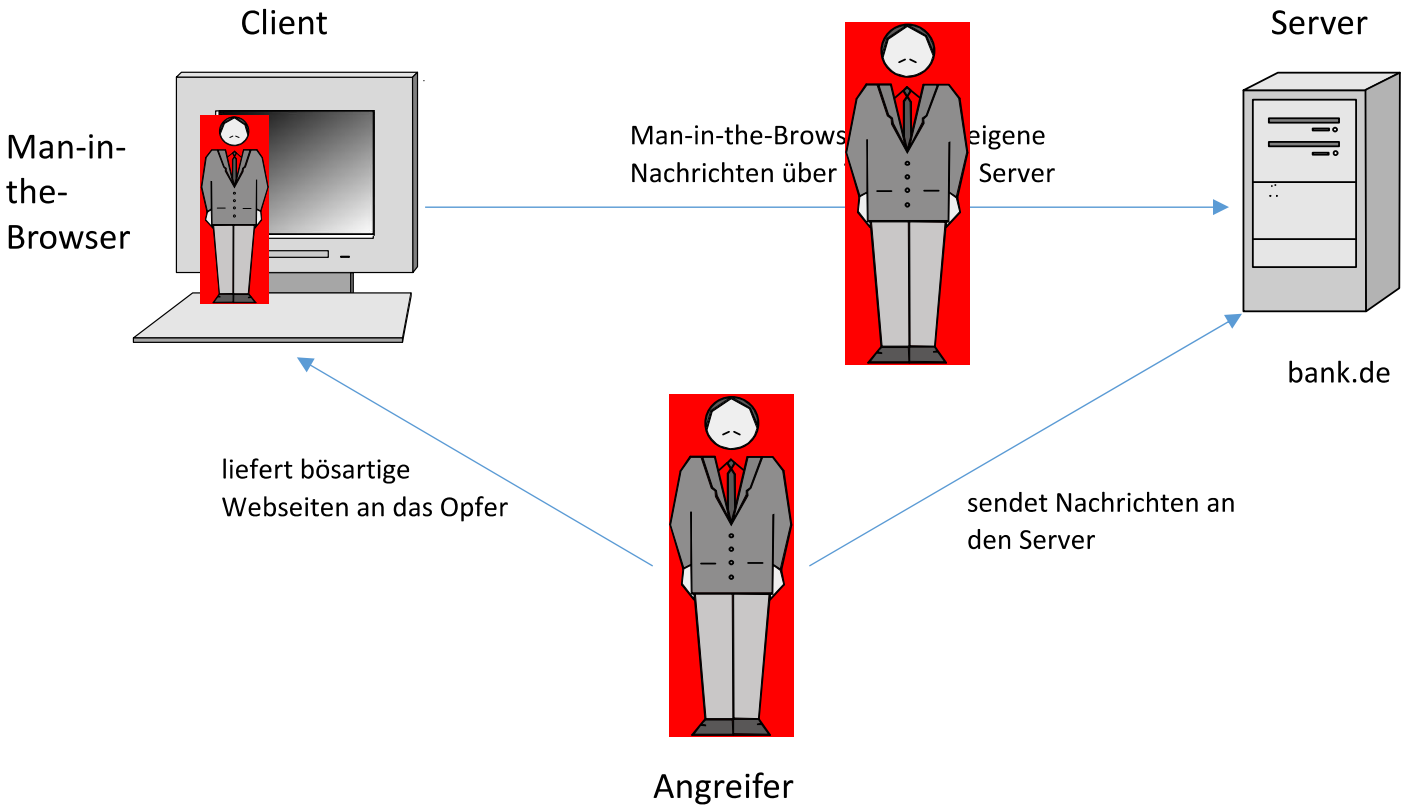
BEAST-Angreifermodell

Man-in-the-Browser +



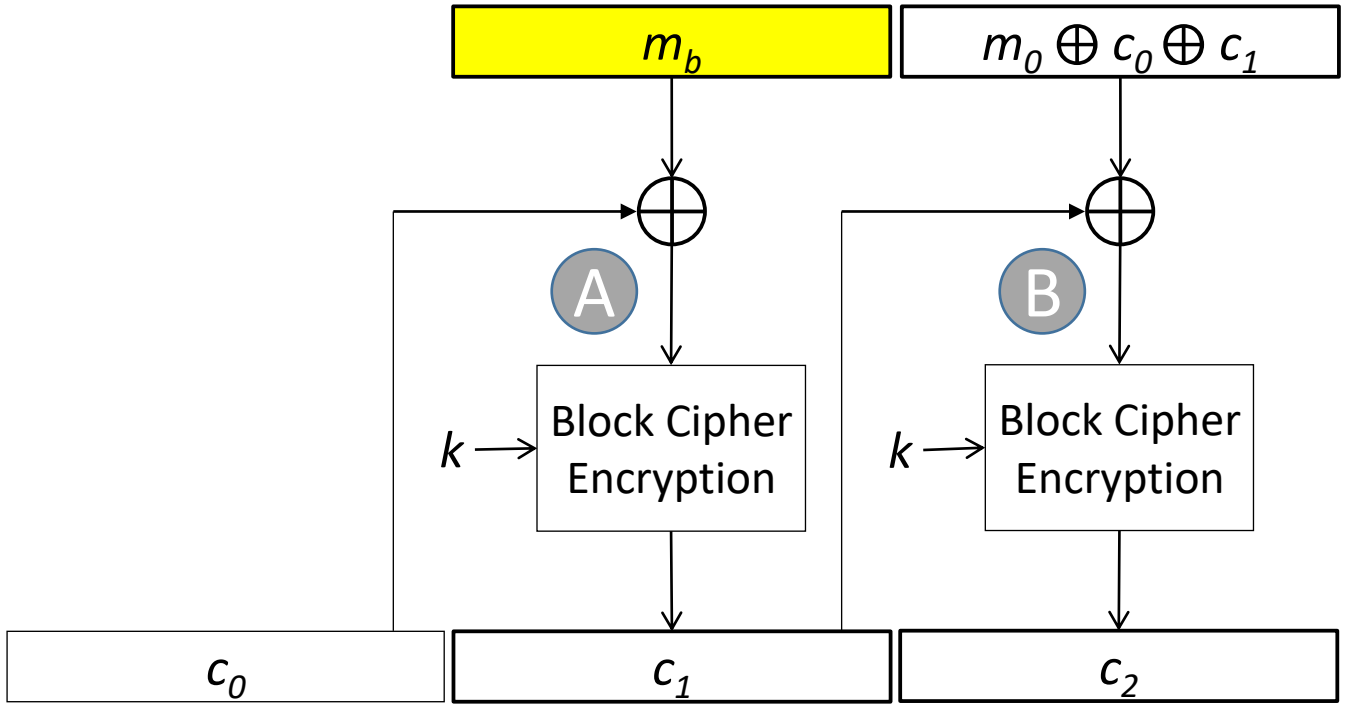
BEAST-Angreifermodell

Man-in-the-Browser +
Man-in-the-Middle



BEAST-Angriffsidee

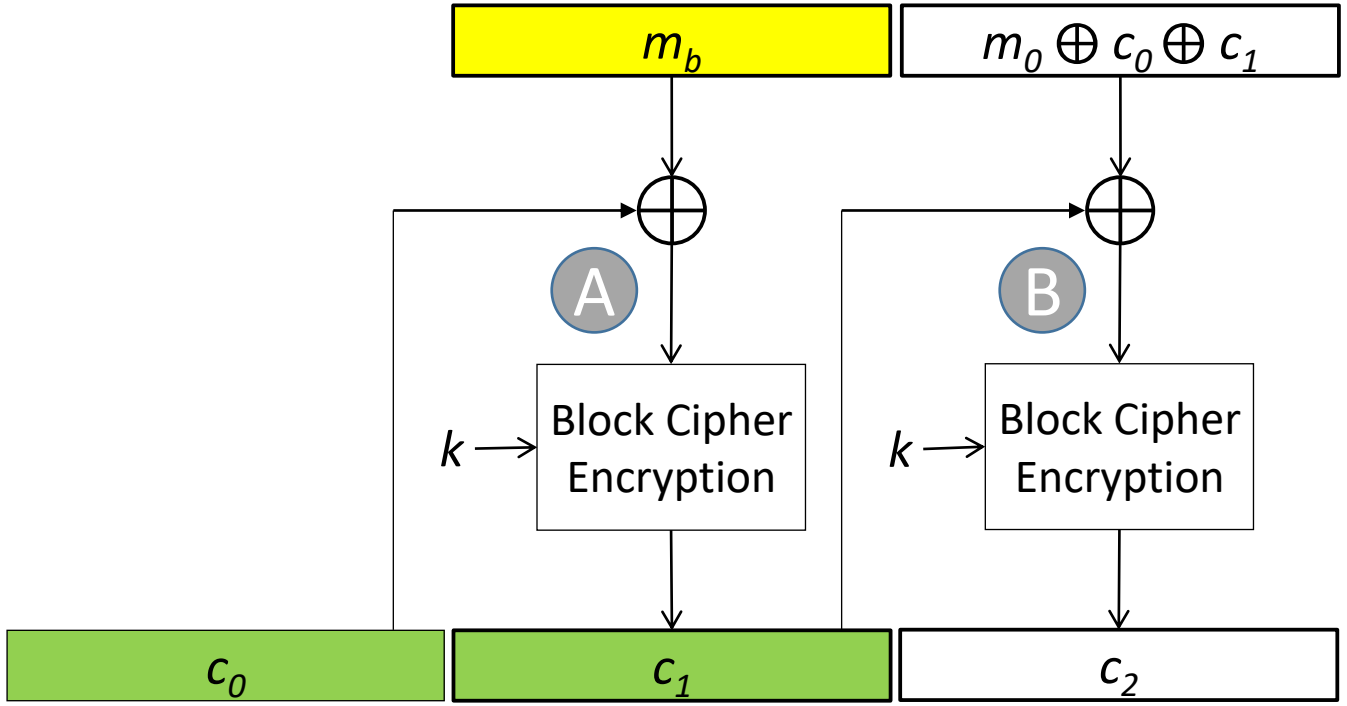
Angreifer kann erkennen, ob im ersten Klartextblock m_0 oder m_1 verschlüsselt wurde, wenn er



BEAST-Angriffsidee

Angreifer kann erkennen, ob im ersten Klartextblock m_0 oder m_1 verschlüsselt wurde, wenn er

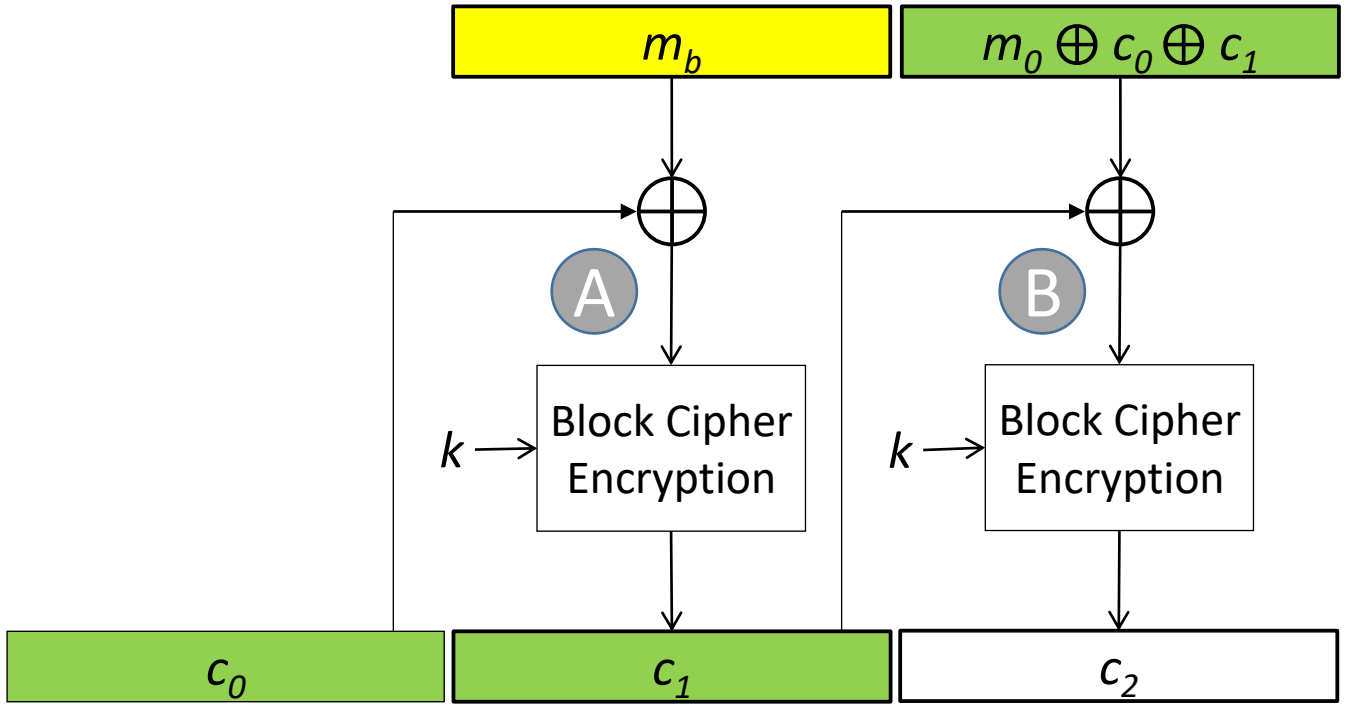
- den IV c_0 und den Block c_1 des CBC-Chiffretexts kennt



BEAST-Angriffsidee

Angreifer kann erkennen, ob im ersten Klartextblock m_0 oder m_1 verschlüsselt wurde, wenn er

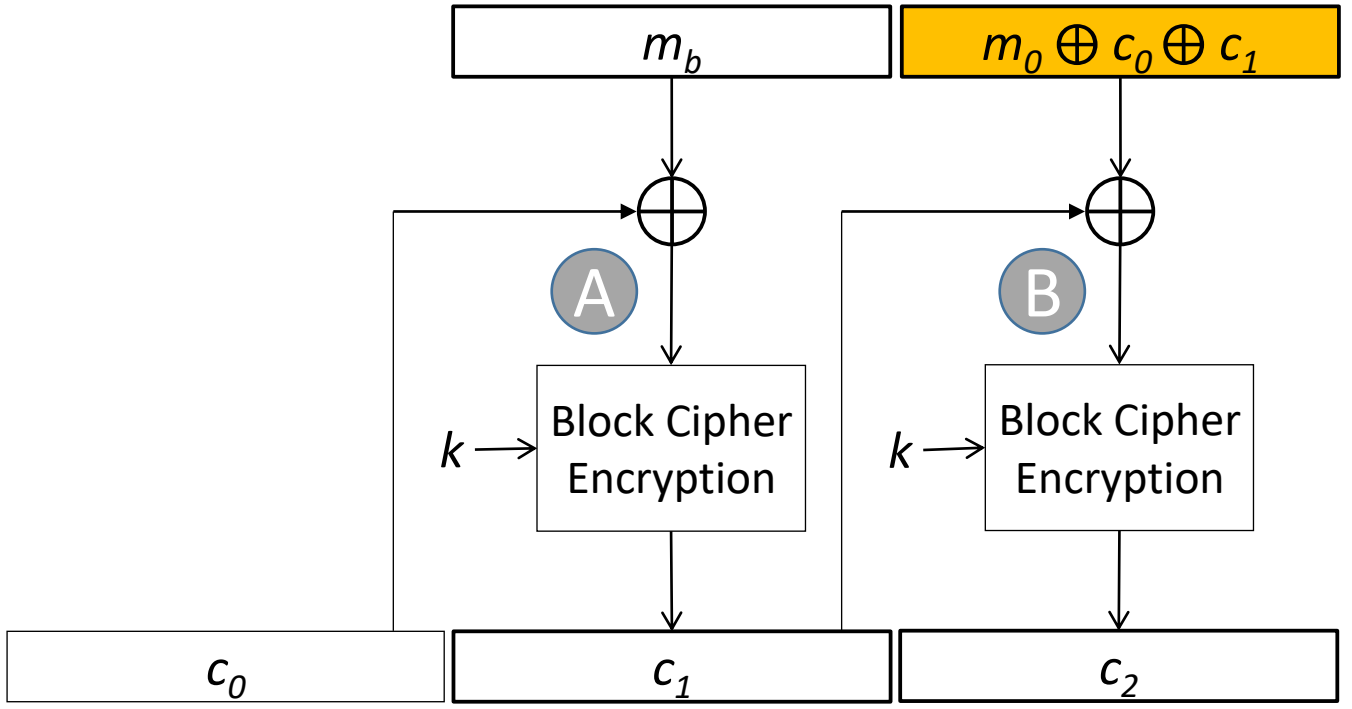
- den IV c_0 und den Block $_{c1}$ des CBC-Chiffretexts kennt
- einen Klartext eigener Wahl im Block 2 verschlüsseln kann



BEAST-Angriffsidee

BEAST

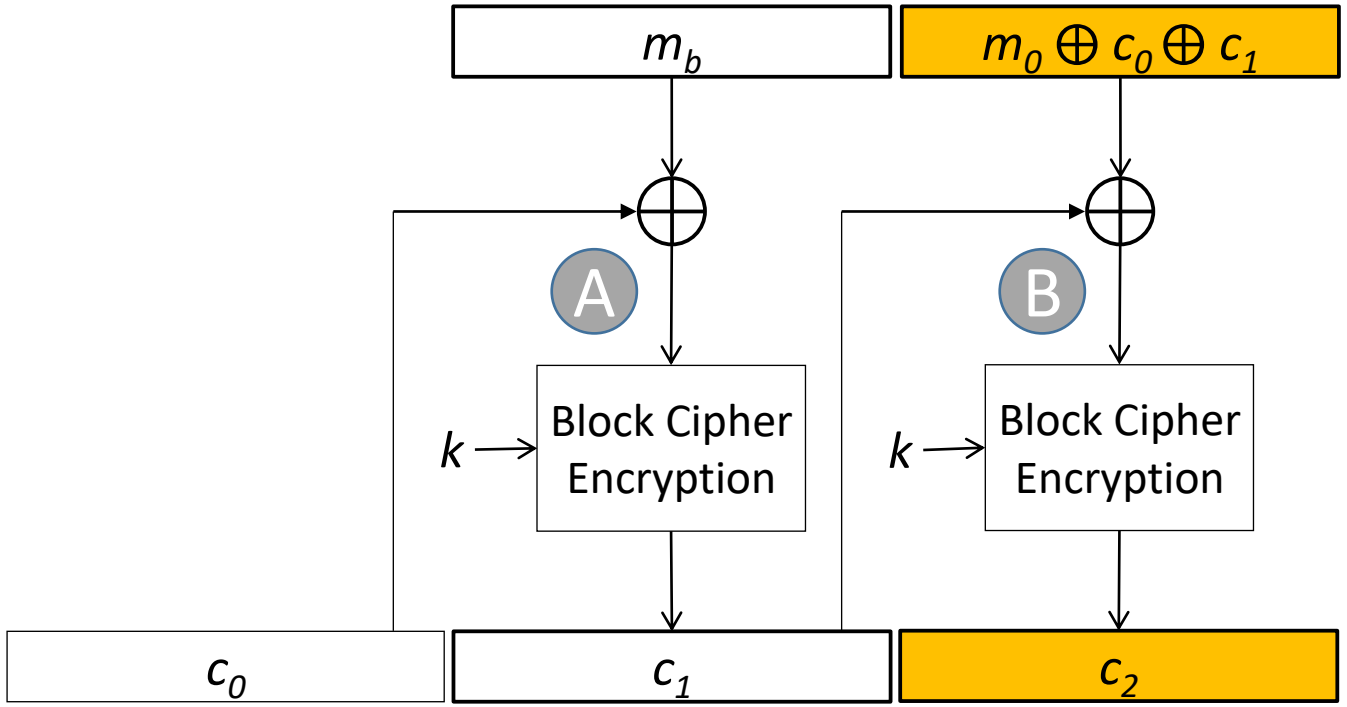
1. Angreifer fügt $m_0 \oplus c_0 \oplus c_1$ als zweiten Klartextblock in die HTTP-Anfrage ein



BEAST-Angriffsidee

BEAST

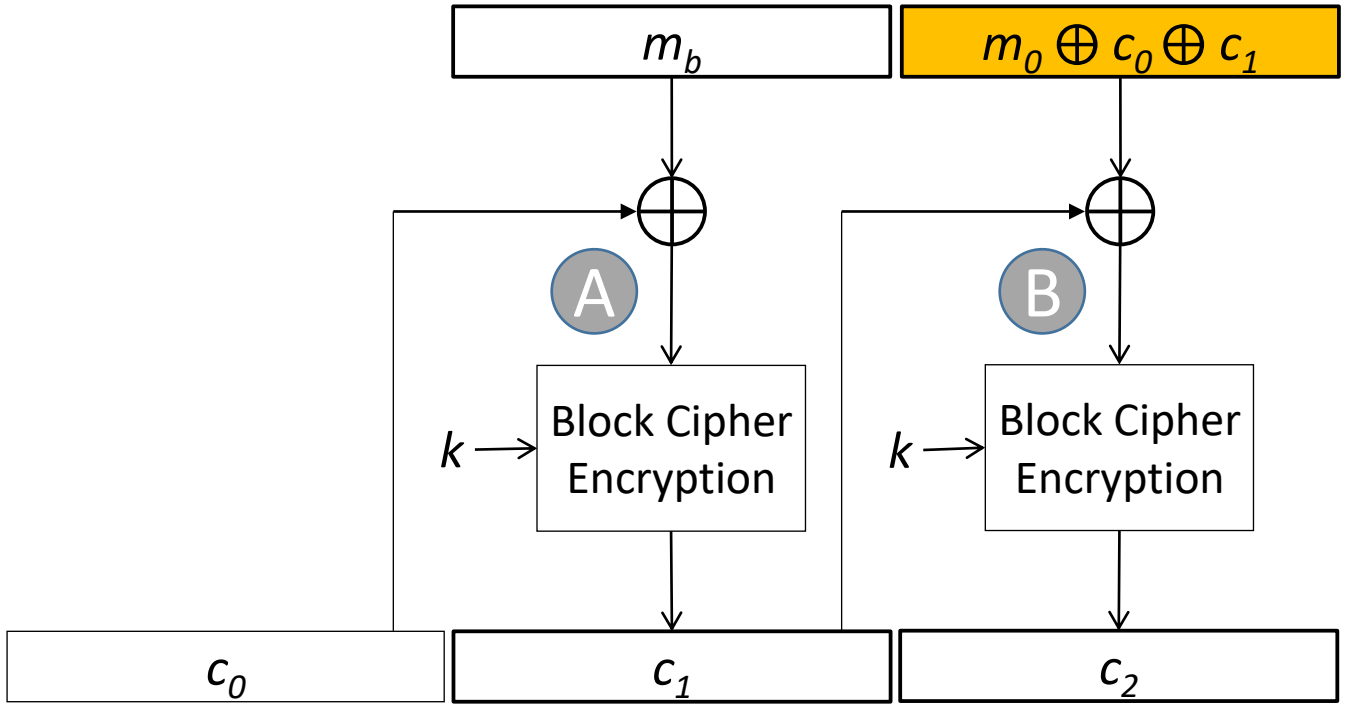
1. Angreifer fügt $m_0 \oplus c_0 \oplus c_1$ als zweiten Klartextblock in die HTTP-Anfrage ein
2. Er zeichnet c_2 als MitM auf



BEAST-Angriffsidee

BEAST

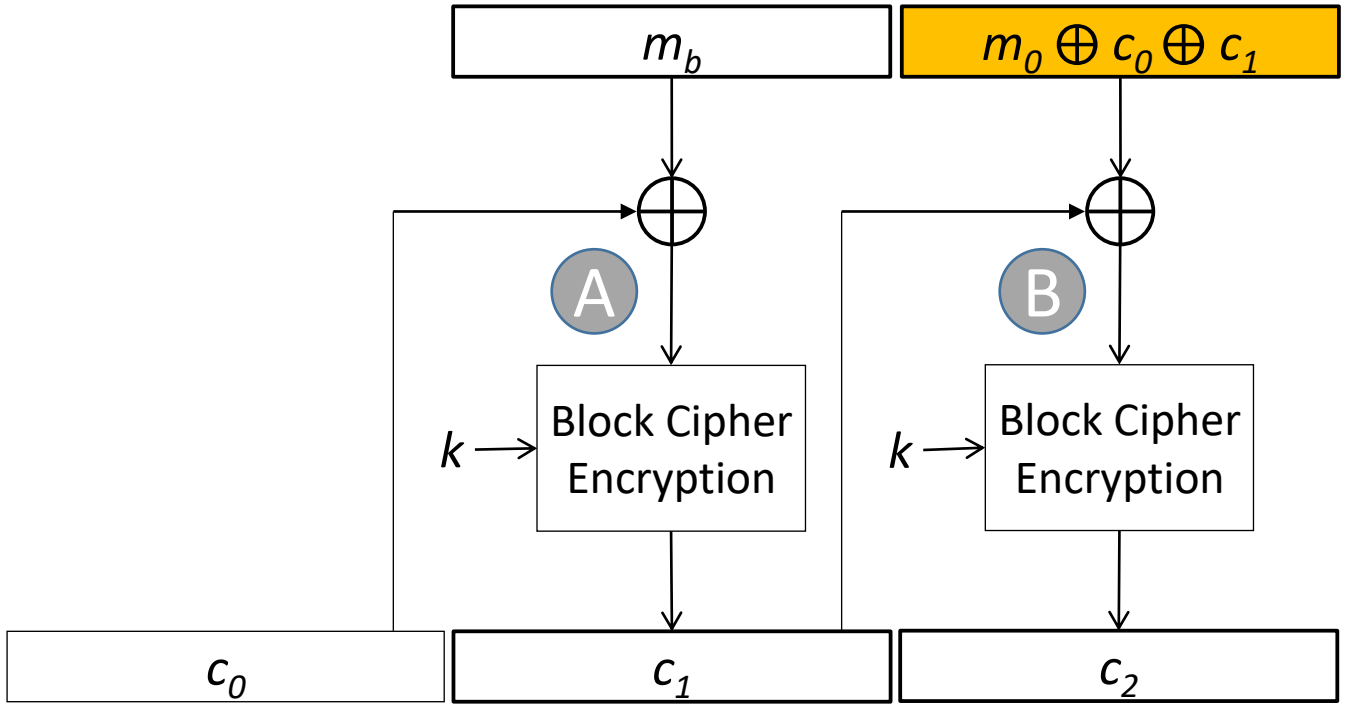
3. Betrachten wir die Werte A und B



BEAST-Angriffsidee

BEAST

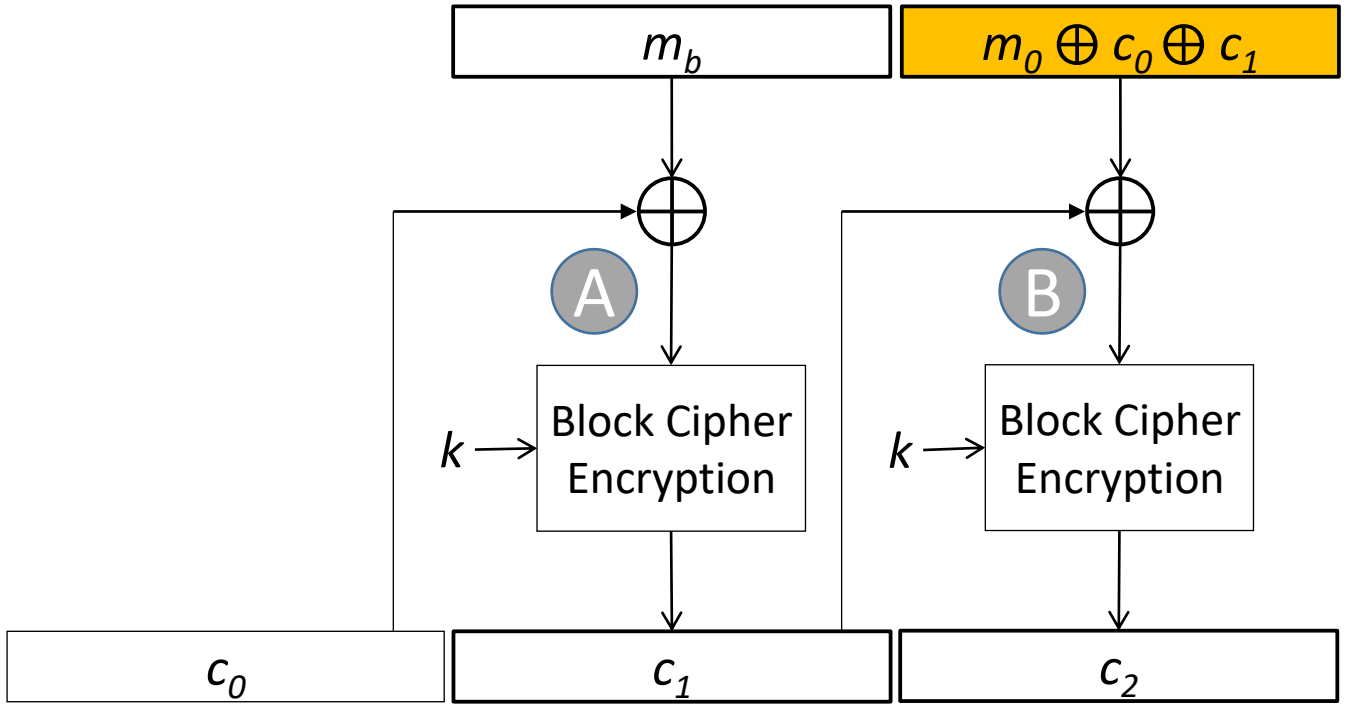
3. Betrachten wir die Werte A und B
   - Fall 1: $m_b = m_0$



BEAST-Angriffsidee

BEAST

3. Betrachten wir die Werte A und B
   - Fall 1: $m_b = m_0$

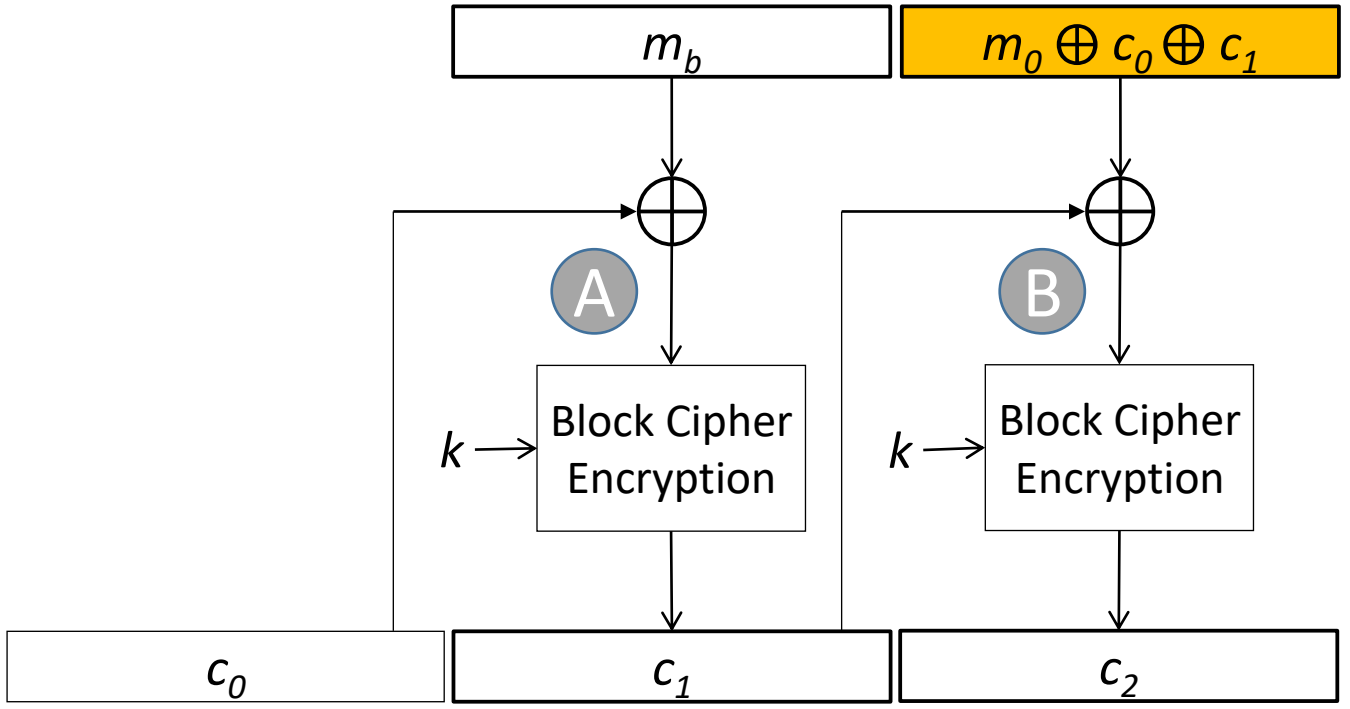


$$A = m_0 \oplus c_0$$

BEAST-Angriffsidee

BEAST

3. Betrachten wir die Werte A und B
   - Fall 1: $m_b = m_0$



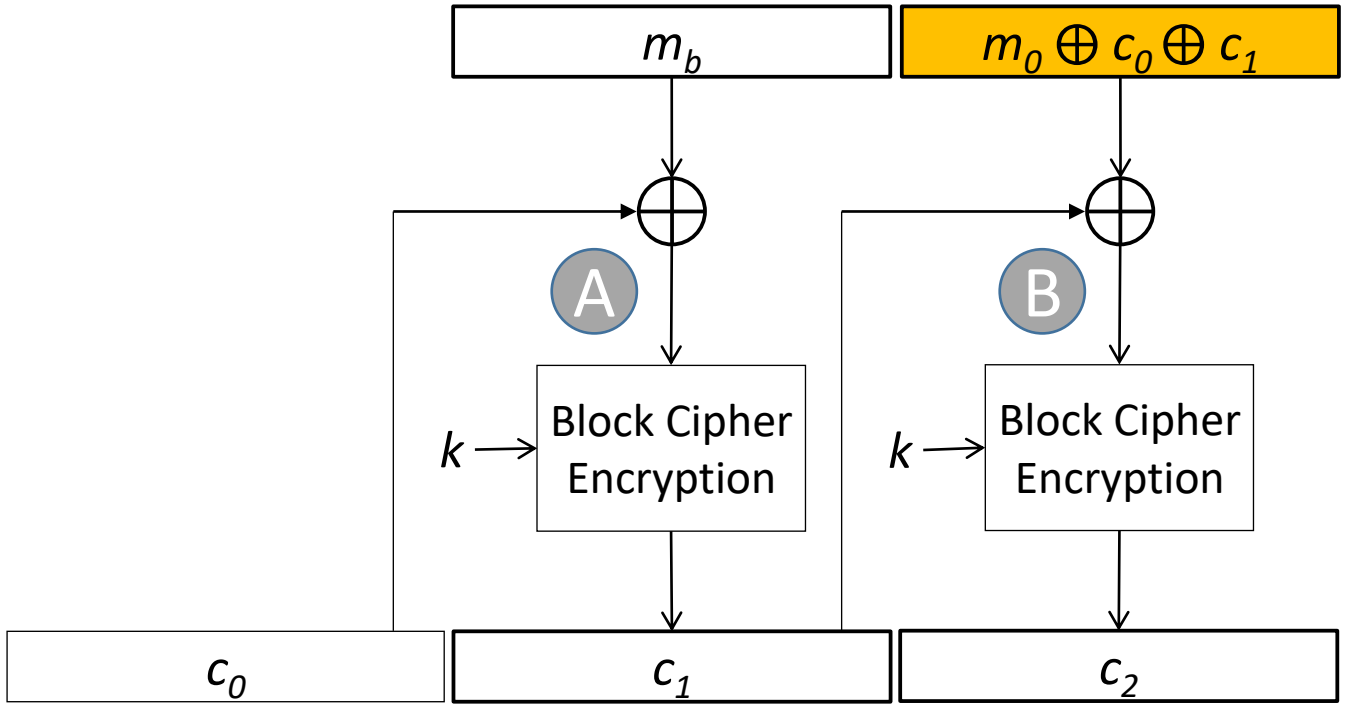
$A = m_0 \oplus c_0$

$B = (m_0 \oplus c_0 \oplus c_1) \oplus c_1$

BEAST-Angriffsidee

BEAST

3. Betrachten wir die Werte A und B
   - Fall 1: $m_b = m_0$



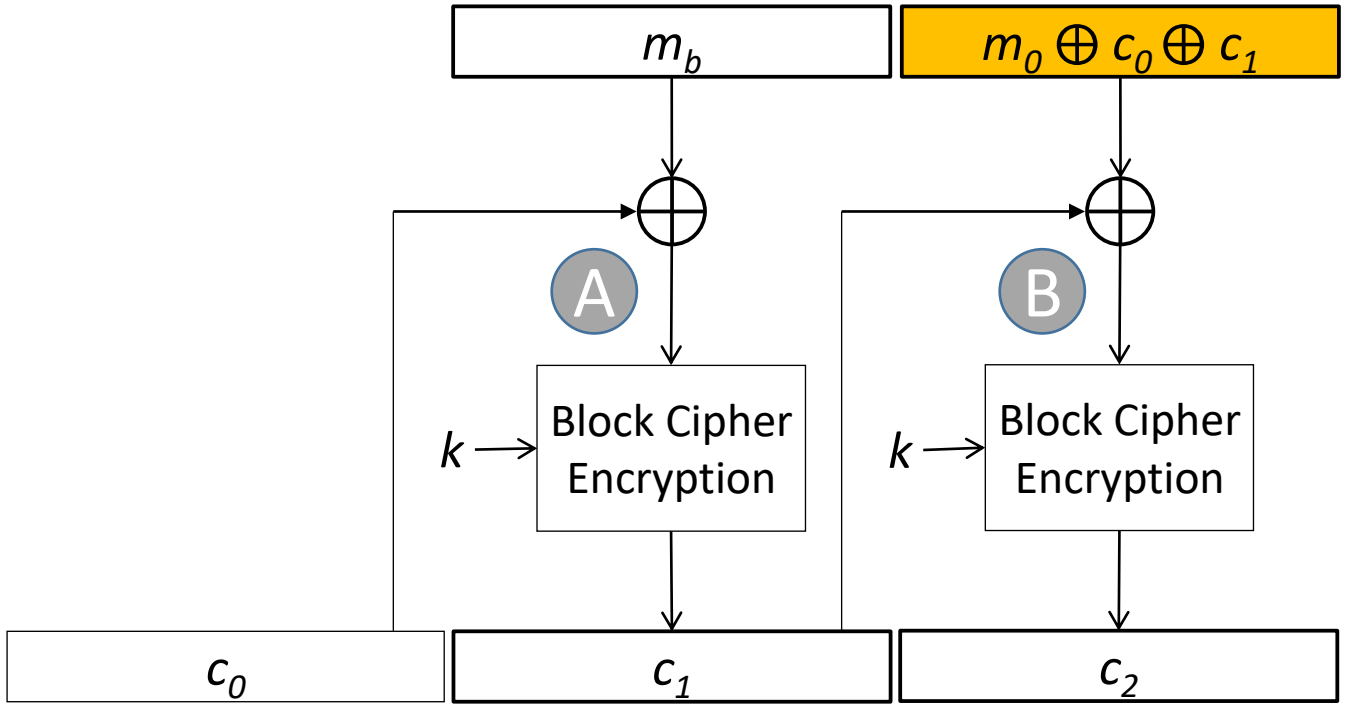
$A = m_0 \oplus c_0$

$B = (m_0 \oplus c_0 \oplus c_1) \oplus c_1 = m_0 \oplus c_0$

BEAST-Angriffsidee

BEAST

3. Betrachten wir die Werte A und B
   - Fall 1: $m_b = m_0$



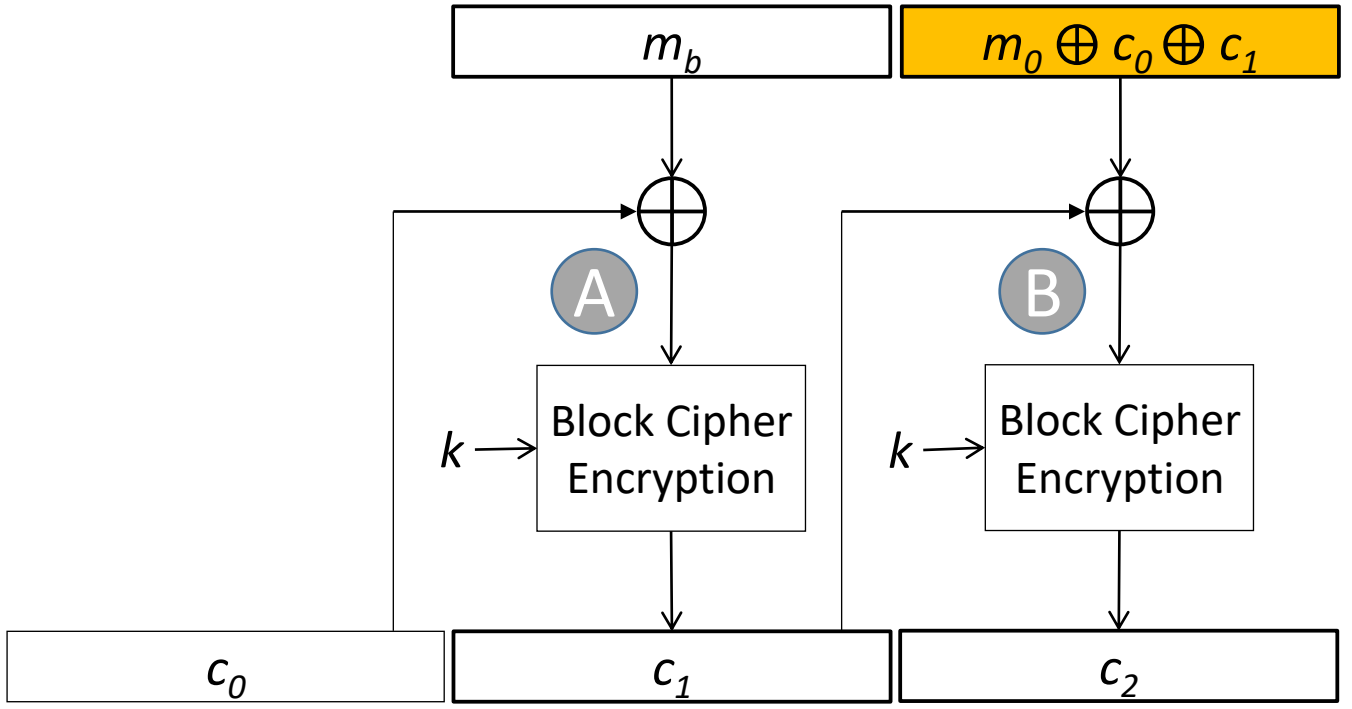
$A = m_0 \oplus c_0$ $B = (m_0 \oplus c_0 \oplus c_1) \oplus c_1 = m_0 \oplus c_0$

Wir haben A=B, und daher auch c_1=c_2

BEAST-Angriffsidee

BEAST

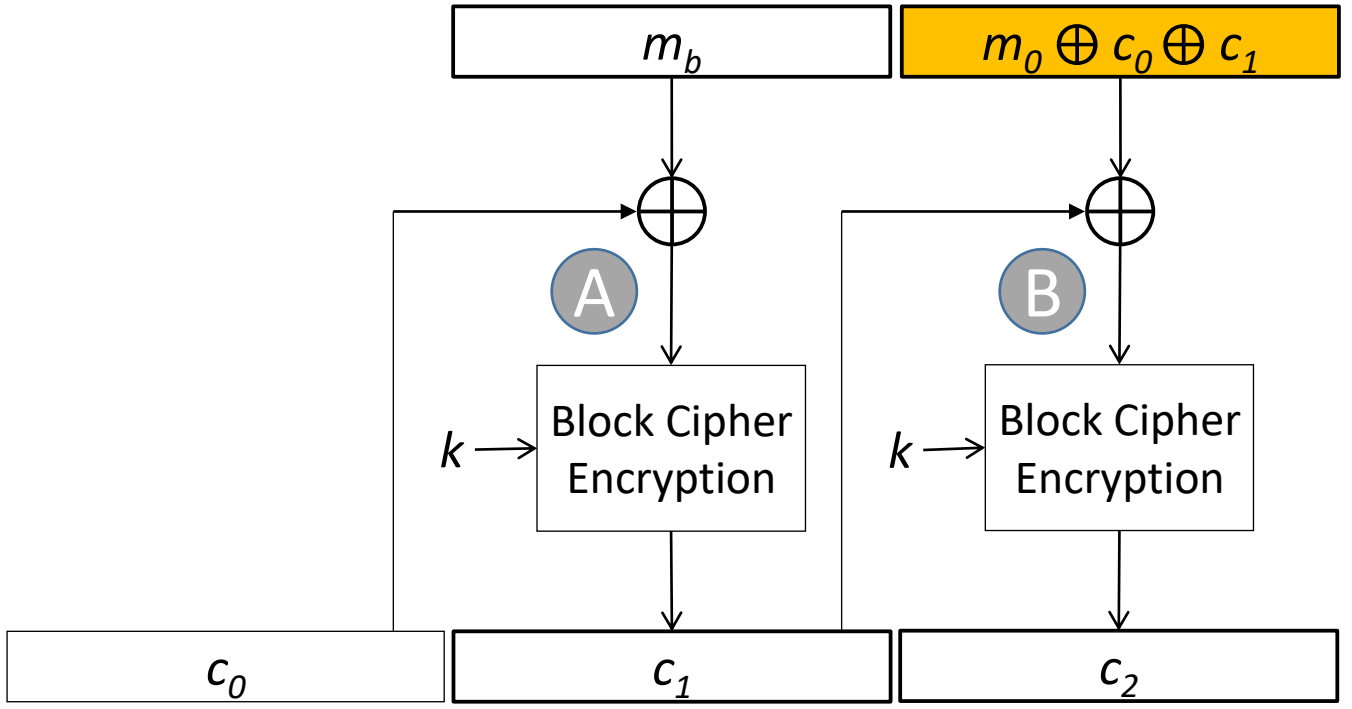
3. Betrachten wir die Werte A und B
   - Fall 1: $m_b = m_0$
   - Fall 2: $m_b \neq m_0$



BEAST-Angriffsidee

BEAST

3. Betrachten wir die Werte A und B
   - Fall 1: $m_b = m_0$
   - Fall 2: $m_b \neq m_0$

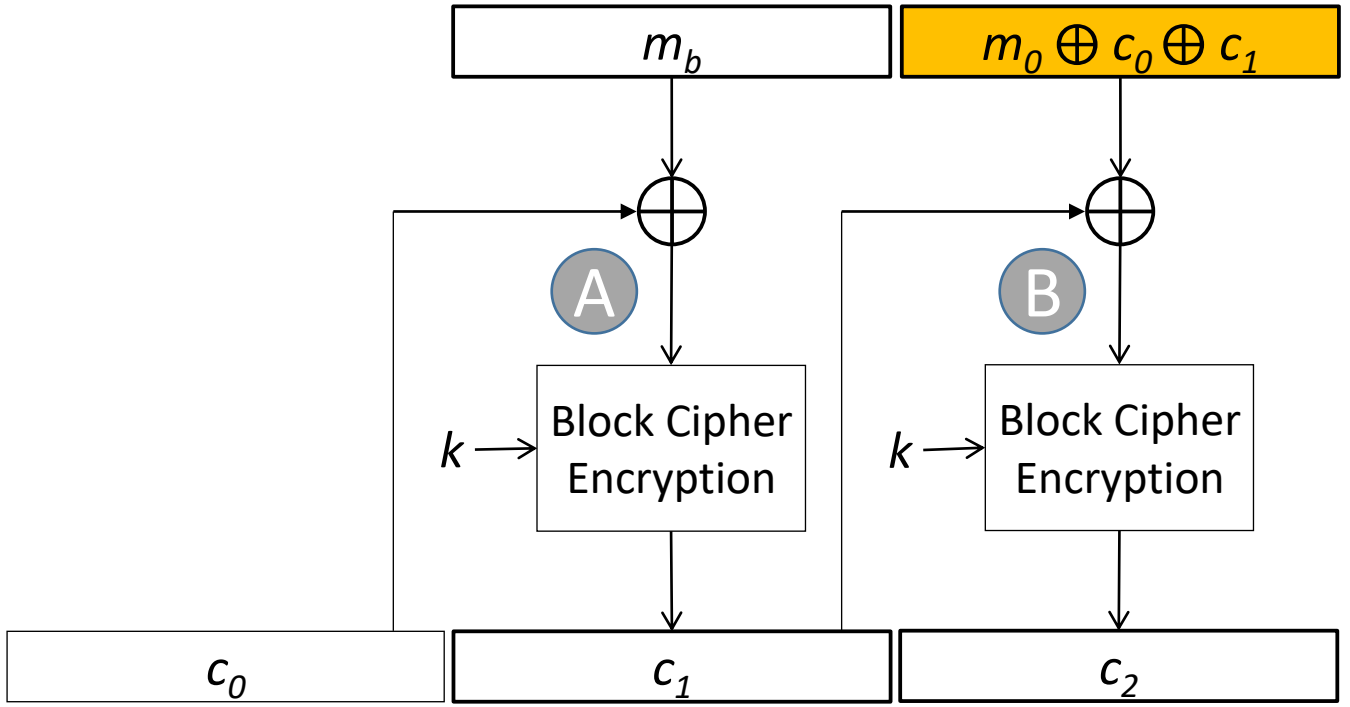


Wir haben A≠B, und daher auch $c_1 \neq c_2$

BEAST-Angriffsidee

BEAST

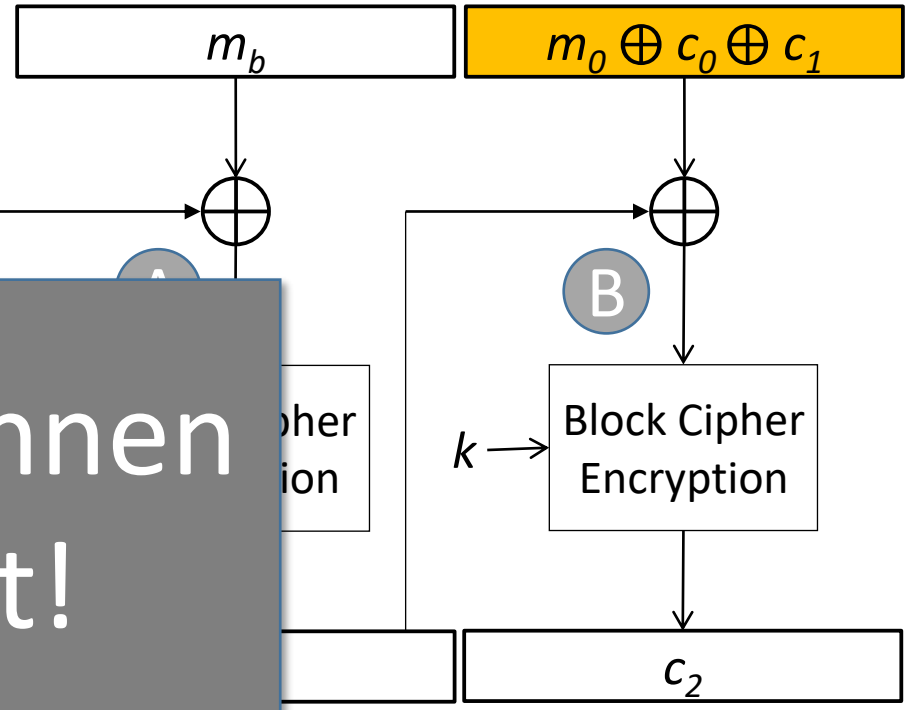
5. Fazit: Wenn $c_1=c_2$ dann ist $m_b=m_0$



BEAST-Angriffsidee

BEAST

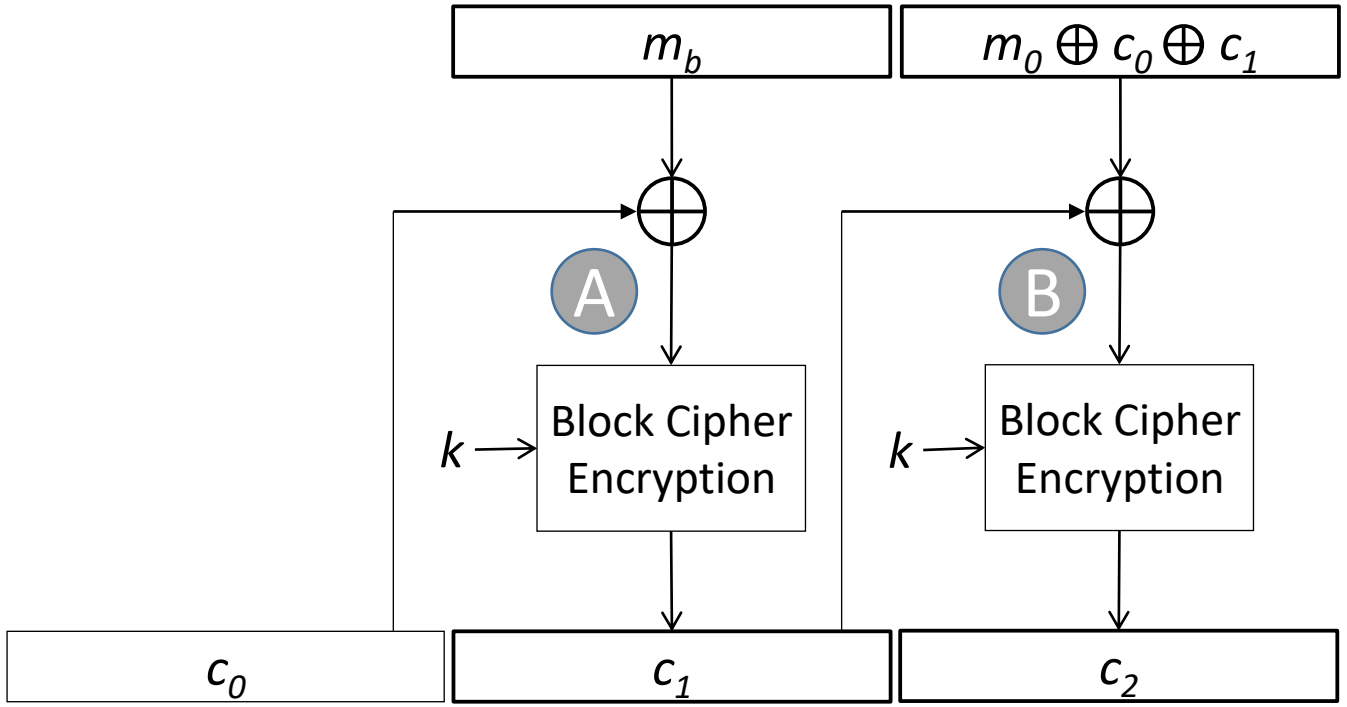
5. Fazit: Wenn $c_1=c_2$ dann ist $m_b=m_0$



Problem: Wir kennen c_0 und c_1 nicht!

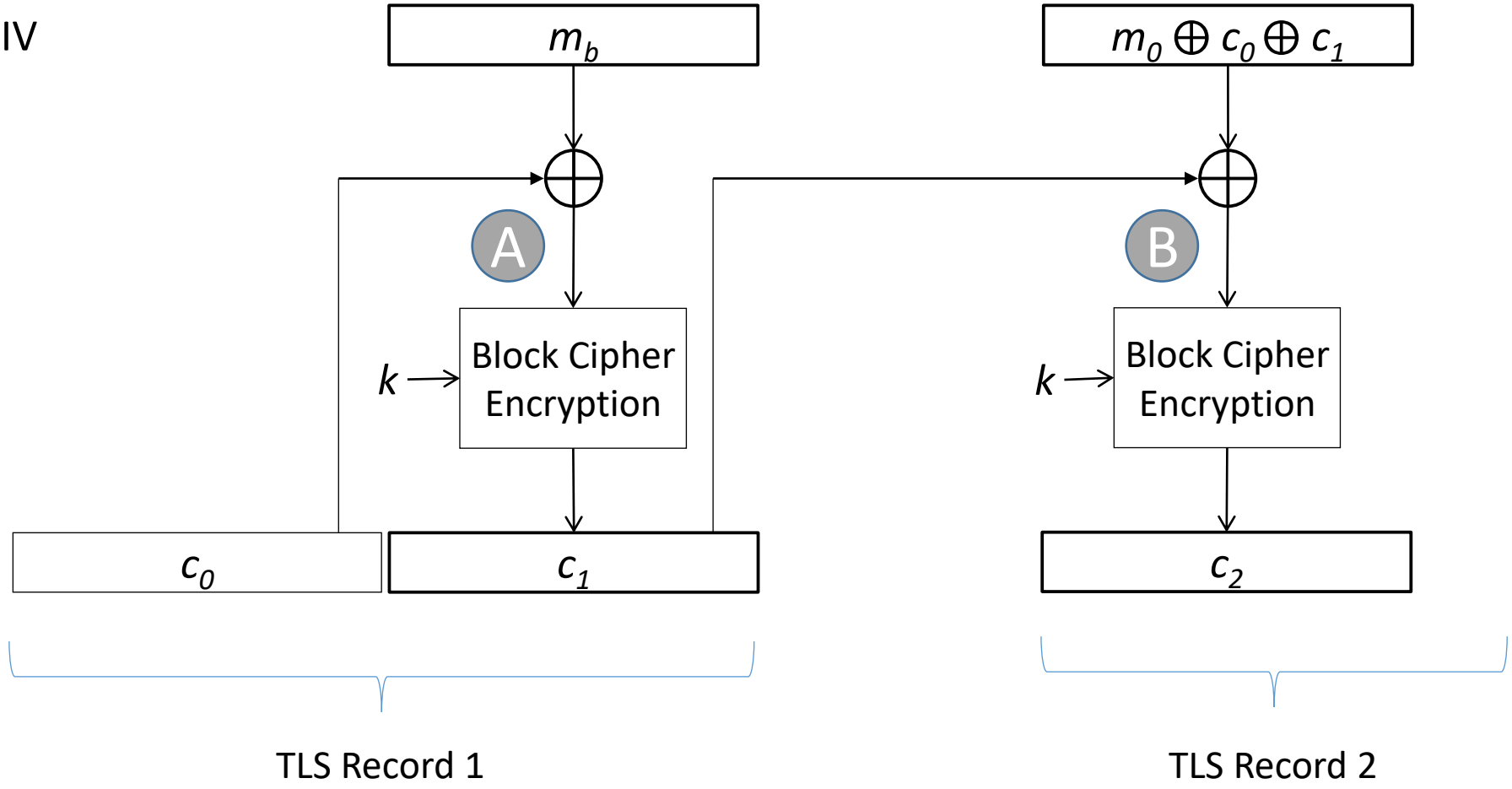
BEAST: Voraussetzungen in SSL 3.0 und TLS 1.0 erfüllt

SSL 3.0 und TLS 1.0



BEAST: Voraussetzungen in SSL 3.0 und TLS 1.0 erfüllt

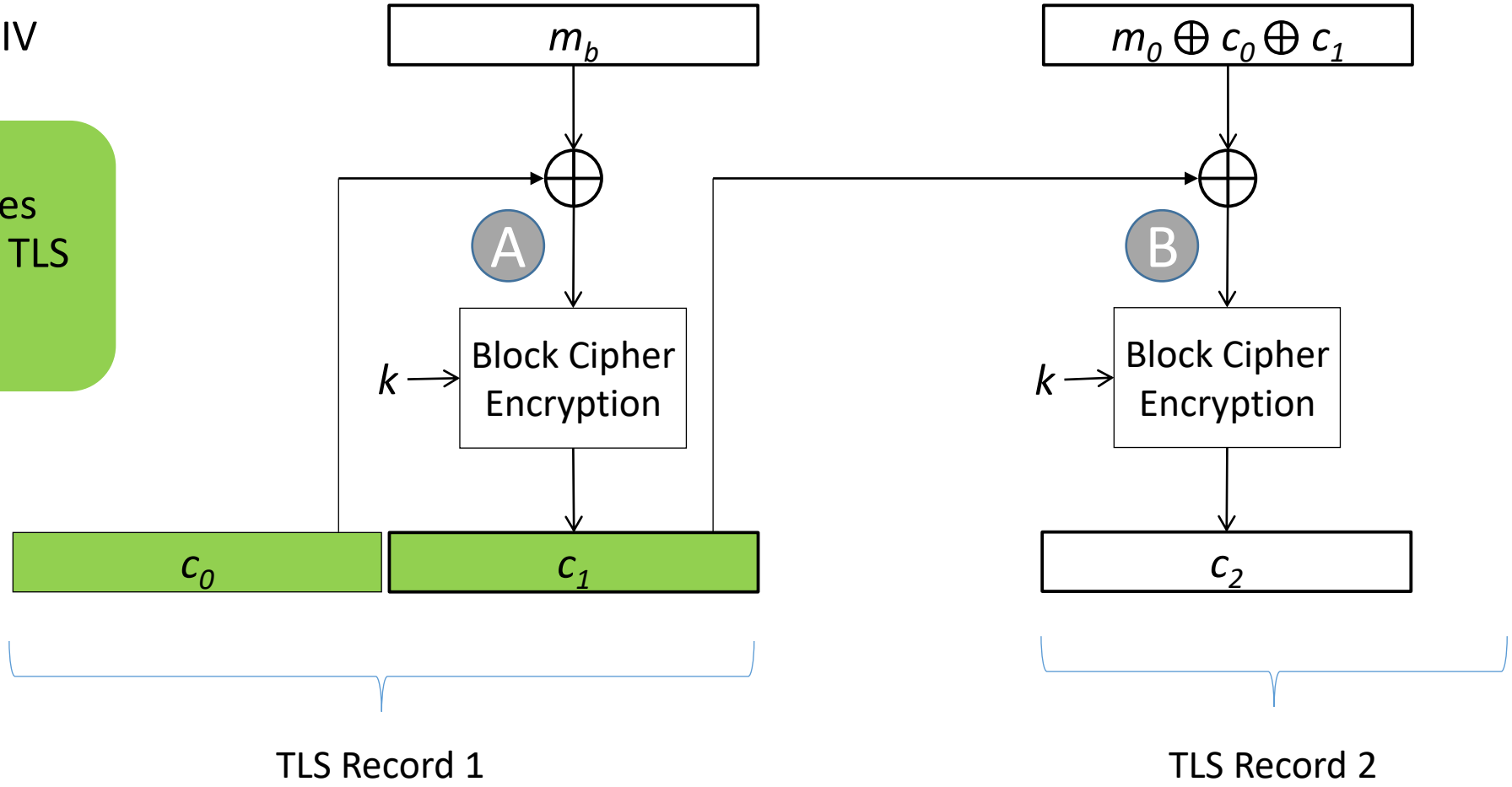
SSL 3.0 und TLS 1.0: IV Chaining



BEAST: Voraussetzungen in SSL 3.0 und TLS 1.0 erfüllt

SSL 3.0 und TLS 1.0: IV Chaining

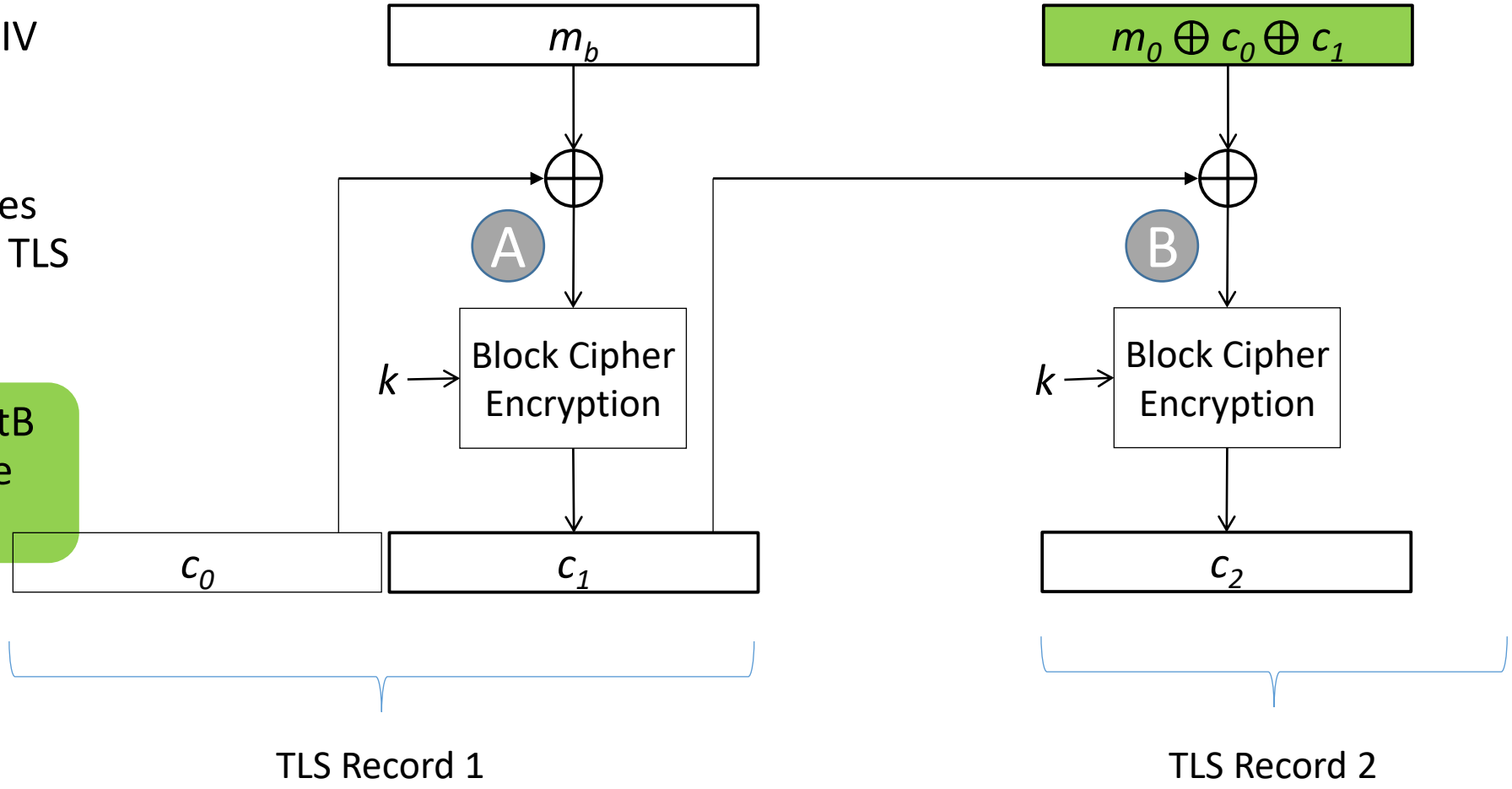
- c_1 ist der letzte Chiffretextblock des vorangegangenen TLS Record, c_0 der vorletzte Block



BEAST: Voraussetzungen in SSL 3.0 und TLS 1.0 erfüllt

SSL 3.0 und TLS 1.0: IV Chaining

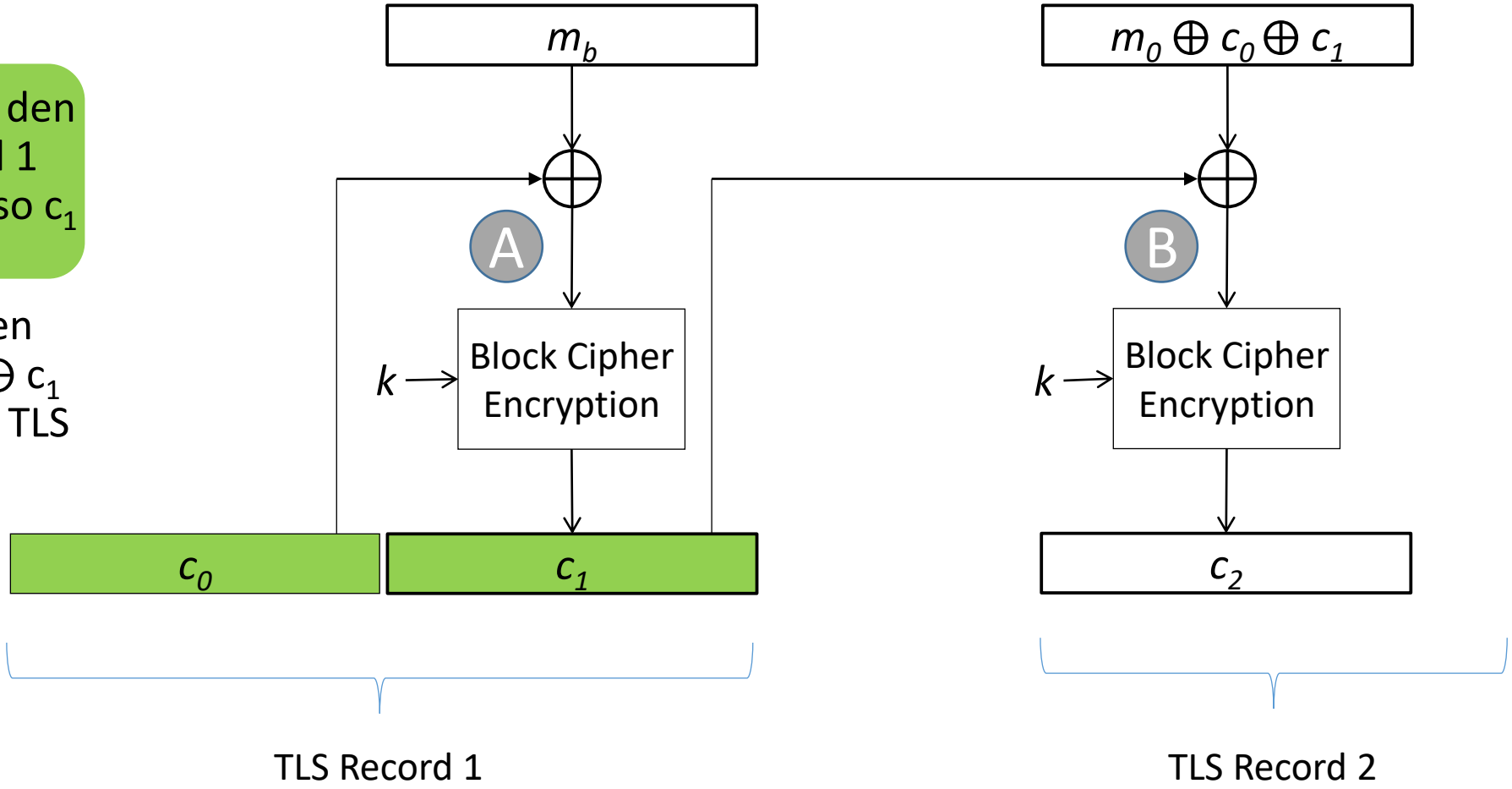
- c_1 ist der letzte Chiffretextblock des vorangegangenen TLS Record, c_0 der vorletzte Block
- wir können als MitB beliebige Klartexte senden



BEAST: Angriff

Angreifer

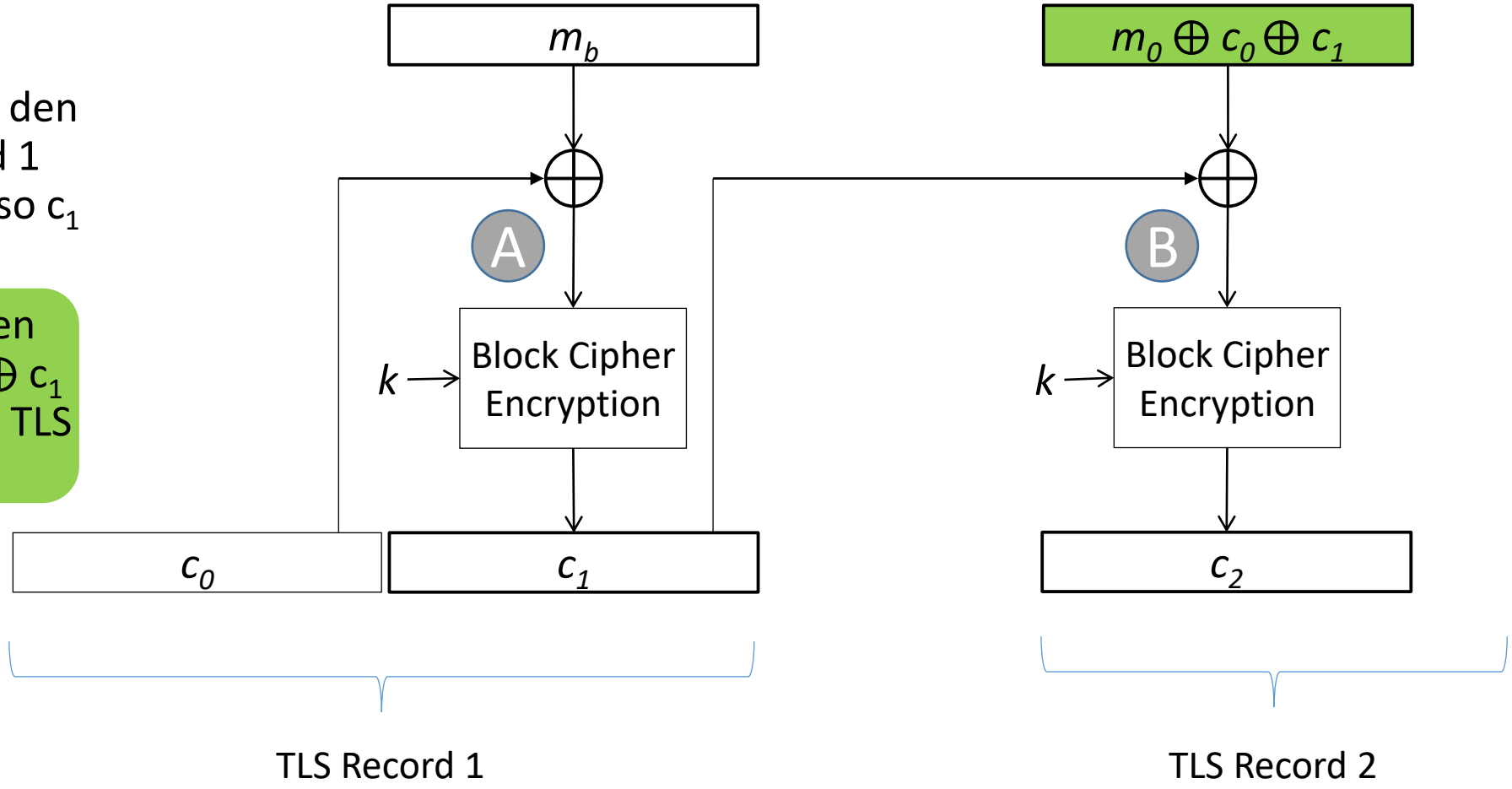
- zeichnet als MitM den letzten TLS Record 1 auf und ermittelt so c_1 und c_0
- sendet als MitB den Klartext $m_0 \oplus c_0 \oplus c_1$ als ersten Block in TLS Record 2
- überprüft ob $c_2 = c_1$



BEAST: Angriff

Angreifer

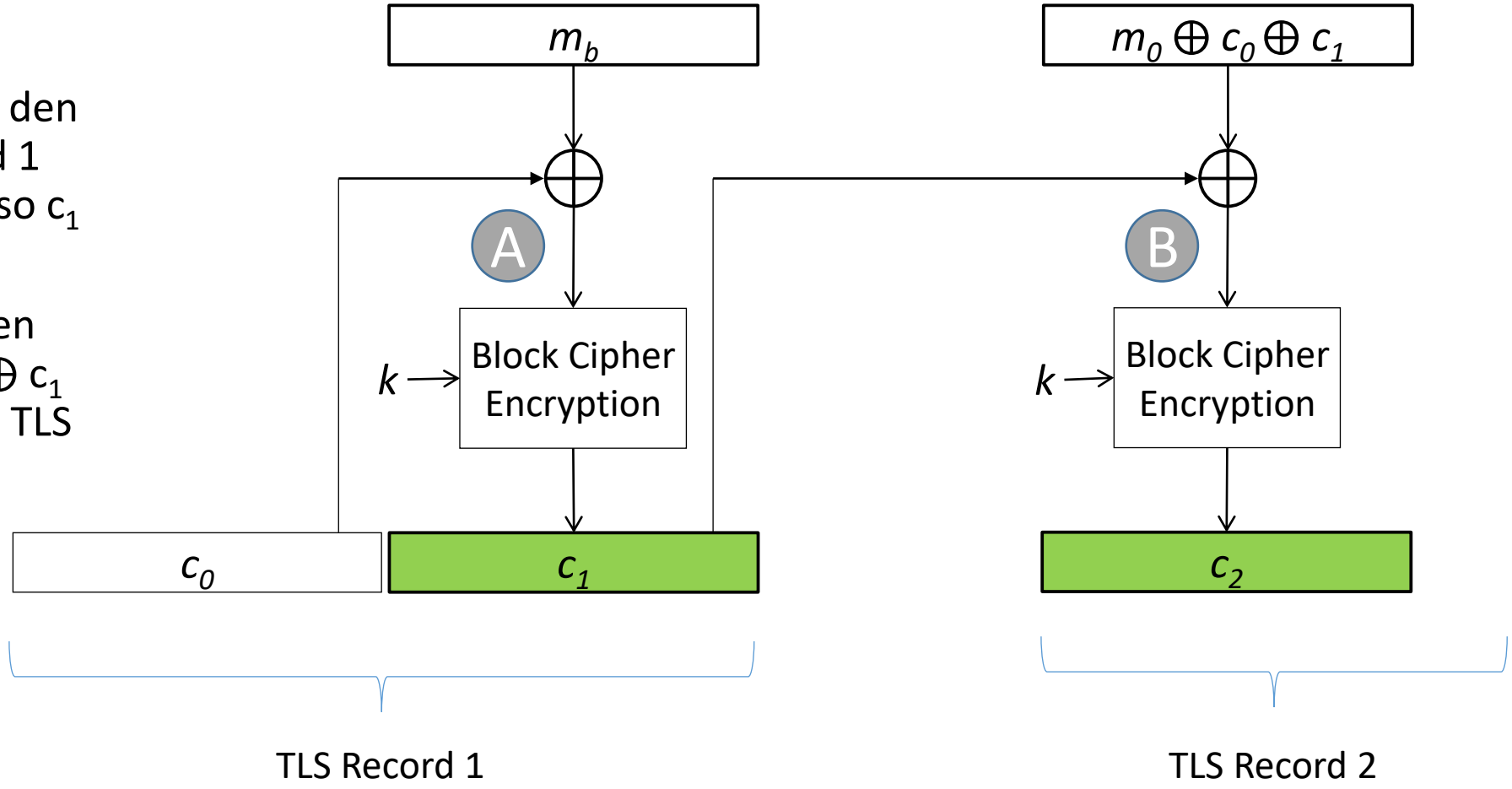
- zeichnet als MitM den letzten TLS Record 1 auf und ermittelt so c_1 und c_0
- sendet als MitB den Klartext $m_0 \oplus c_0 \oplus c_1$ als ersten Block in TLS Record 2
- überprüft ob $c_2 = c_1$



BEAST: Angriff

Angreifer

- zeichnet als MitM den letzten TLS Record 1 auf und ermittelt so c_1 und c_0
- sendet als MitB den Klartext $m_0 \oplus c_0 \oplus c_1$ als ersten Block in TLS Record 2
- überprüft ob $c_2 = c_1$



Bytewise Privileges

BEAST: Problem

Bytewise Privileges

BEAST: Problem

- m_b ist zwischen 64 (3DES) und 128 (AES) Bit lang

Bytewise Privileges

BEAST: Problem

- m_b ist zwischen 64 (3DES) und 128 (AES) Bit lang
- bei unbekanntem m_b benötigt der Angreifer also im Worst Case zwischen 2^{64} und 2^{128} Versuche, um m_b so zu entschlüsseln

Bytewise Privileges

BEAST: Problem

- m_b ist zwischen 64 (3DES) und 128 (AES) Bit lang
- bei unbekanntem m_b benötigt der Angreifer also im Worst Case zwischen 2^{64} und 2^{128} Versuche, um m_b so zu entschlüsseln

Lösung: Bytewise Privileges

Bytewise Privileges

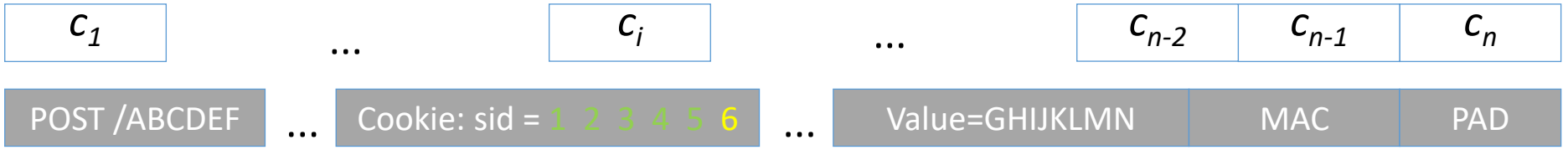
BEAST: Problem

- m_b ist zwischen 64 (3DES) und 128 (AES) Bit lang
- bei unbekanntem m_b benötigt der Angreifer also im Worst Case zwischen 2^{64} und 2^{128} Versuche, um m_b so zu entschlüsseln

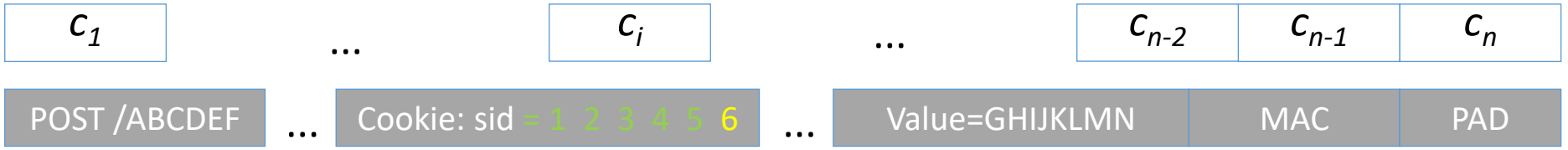
Lösung: Bytewise Privileges

- Sind alle Byte von m_b bis auf eines bekannt, so benötigt der Angreifer nur maximal 2^8 Versuche, um dieses Byte zu bestimmen

Bytewise Privileges

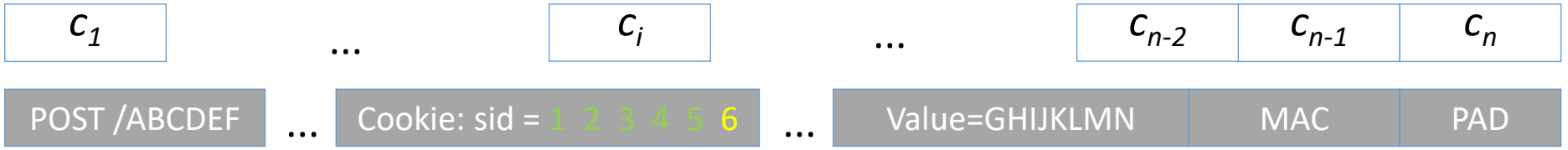


Bytewise Privileges



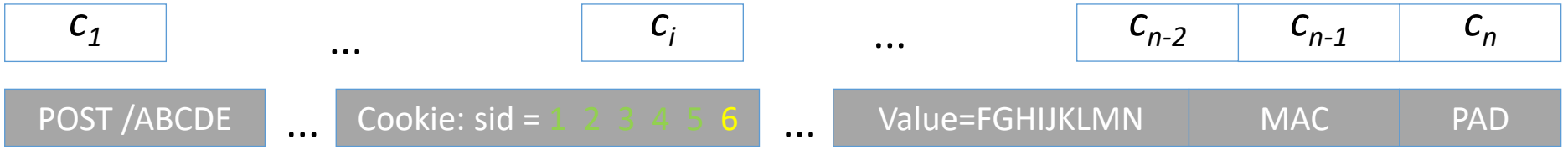
- BEAST-Angreifer kann als MitB die URL variieren, die aufgerufen wird

Bytewise Privileges



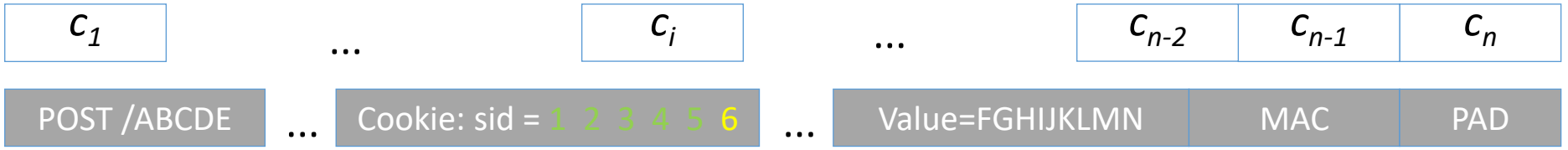
- BEAST-Angreifer kann als MitB die URL variieren, die aufgerufen wird
- Sind vom Session Cookie also schon die Bytewerte 1 2 3 4 5 bekannt, so kann der Angreifer

Bytewise Privileges



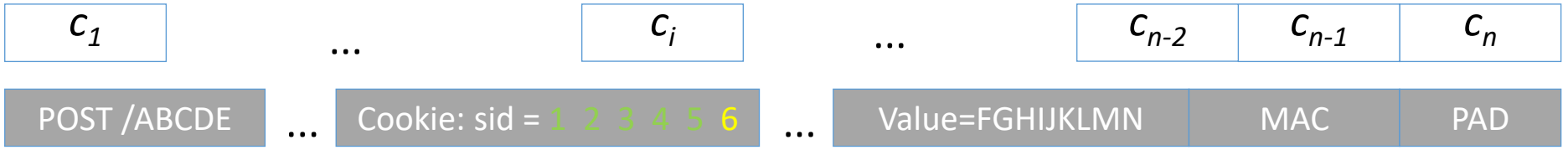
- BEAST-Angreifer kann als MitB die URL variieren, die aufgerufen wird
- Sind vom Session Cookie also schon die Bytewerte 1 2 3 4 5 bekannt, so kann der Angreifer
  - das ASCII-Zeichen F aus der Pfadangabe entfernen
  - ... und in den Wert Value einfügen

Bytewise Privileges



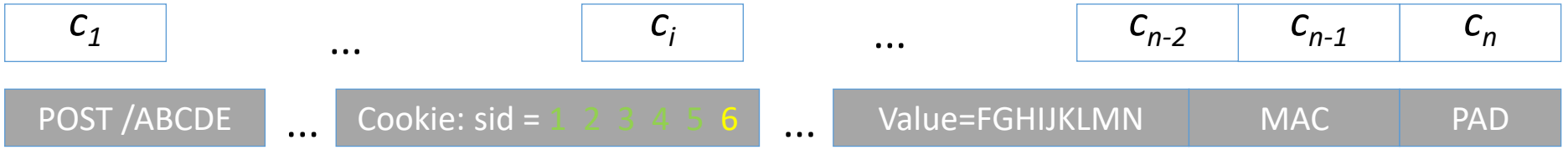
- BEAST-Angreifer kann als MitB die URL variieren, die aufgerufen wird
- Sind vom Session Cookie also schon die Bytewerte 1 2 3 4 5 bekannt, so kann der Angreifer
  - das ASCII-Zeichen F aus der Pfadangabe entfernen
  - ... und in den Wert Value einfügen
- Die Byte in c_i werden um eine Stelle nach links gezogen,

Bytewise Privileges



- BEAST-Angreifer kann als MitB die URL variieren, die aufgerufen wird
- Sind vom Session Cookie also schon die Bytewerte 1 2 3 4 5 bekannt, so kann der Angreifer
  - das ASCII-Zeichen F aus der Pfadangabe entfernen
  - ... und in den Wert Value einfügen
- Die Byte in c_i werden um eine Stelle nach links gezogen, und c_i enthält jetzt die bekannten Byte 2 3 4 5

Bytewise Privileges



- BEAST-Angreifer kann als MitB die URL variieren, die aufgerufen wird
- Sind vom Session Cookie also schon die Bytewerte 1 2 3 4 5 bekannt, so kann der Angreifer
  - das ASCII-Zeichen F aus der Pfadangabe entfernen
  - ... und in den Wert Value einfügen
- Die Byte in c_i werden um eine Stelle nach links gezogen, und c_i enthält jetzt die bekannten Byte 2 3 4 5 und das eine unbekannte Byte 6

BEAST: Fazit

- BEAST hat praktisch nie richtig funktioniert

BEAST: Fazit

- BEAST hat praktisch nie richtig funktioniert
  - bei HTTPS ist der 1. Klartextblock durch den MitB nicht kontrollierbar

BEAST: Fazit

- BEAST hat praktisch nie richtig funktioniert
  - bei HTTPS ist der 1. Klartextblock durch den MitB nicht kontrollierbar
  - Anwendbarkeit auf andere Protokolle wird nur angedeutet

BEAST: Fazit

- BEAST hat praktisch nie richtig funktioniert
  - bei HTTPS ist der 1. Klartextblock durch den MitB nicht kontrollierbar
  - Anwendbarkeit auf andere Protokolle wird nur angedeutet

- ABER: Die Idee der Bytewise Privileges haben alle nachfolgenden Angriffe auf TLS erst möglich gemacht

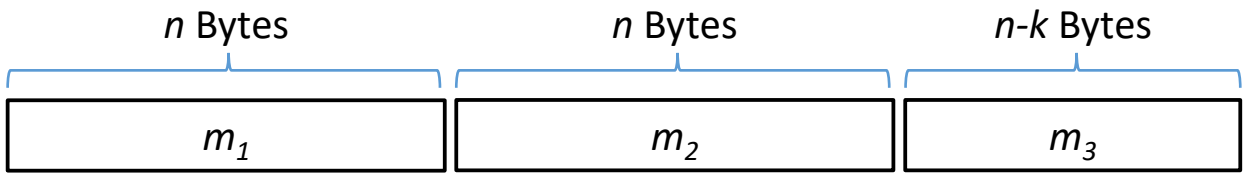
3.3 Angriffe auf den Record Layer

3.3.3 Padding-Oracle-Angriffe: RFC 2440

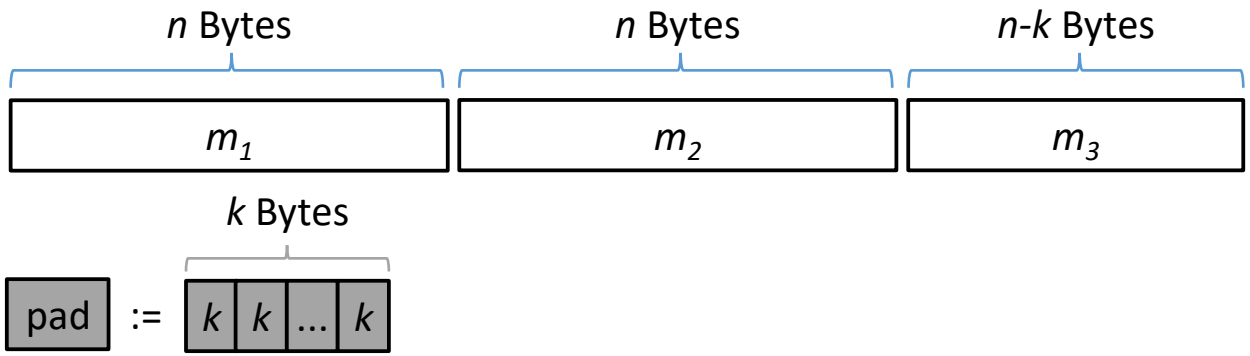
RFC 2440: Serge Vaudenay

[Vau02] Serge Vaudenay. Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS... In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 534–546, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany.

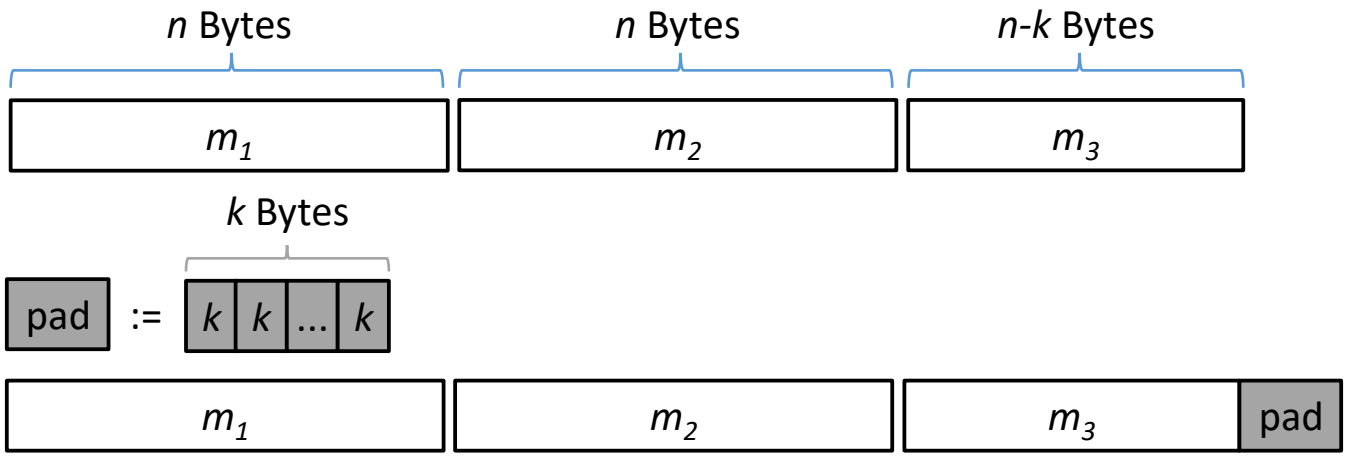
Padding nach RFC 2440



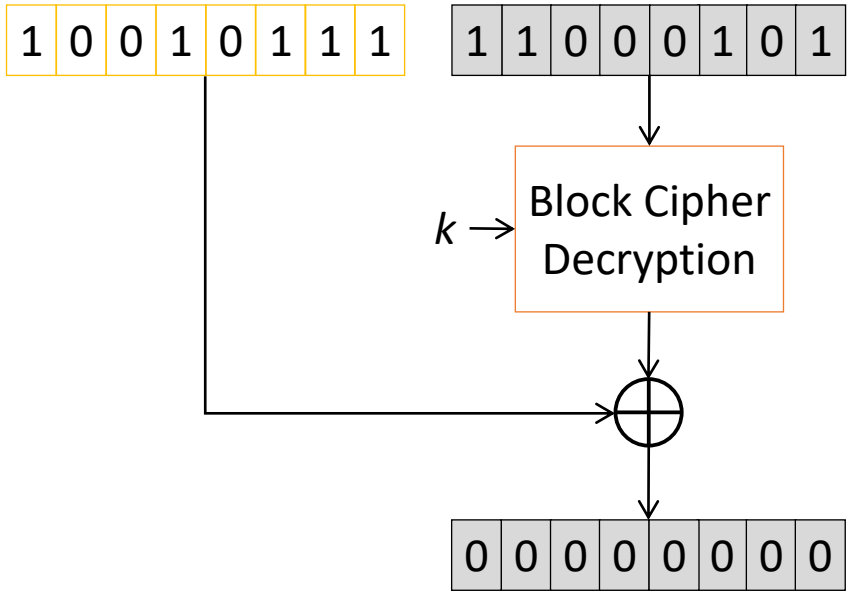
Padding nach RFC 2440



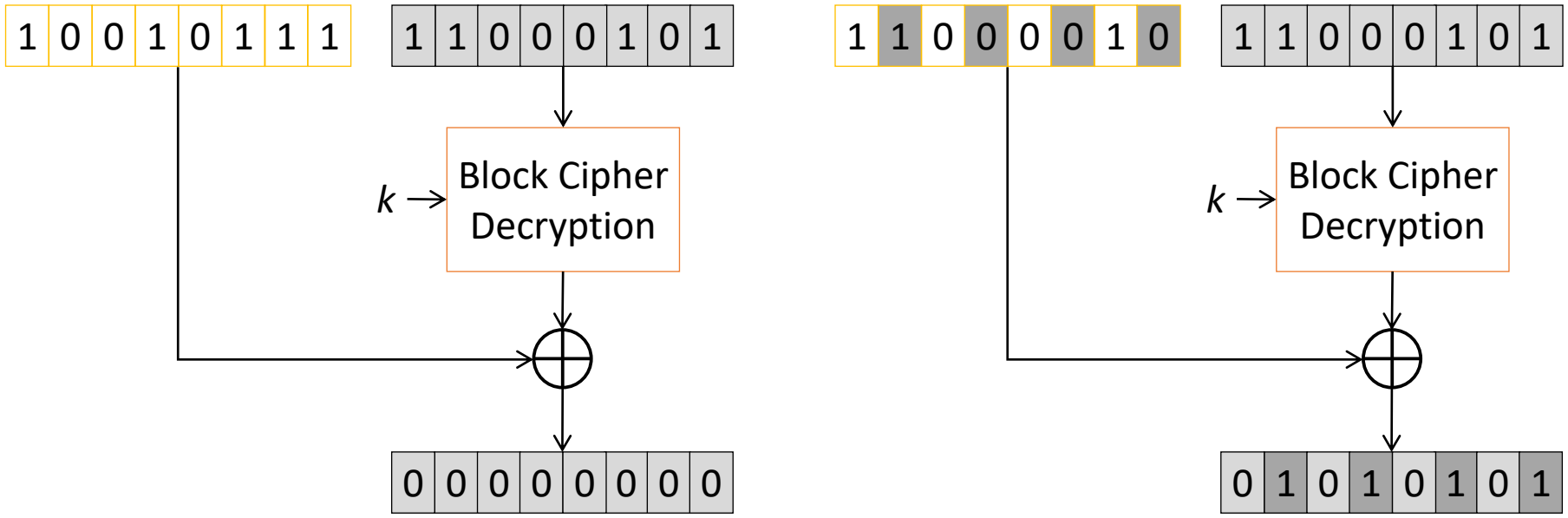
Padding nach RFC 2440



Malleability des CBC-Modus

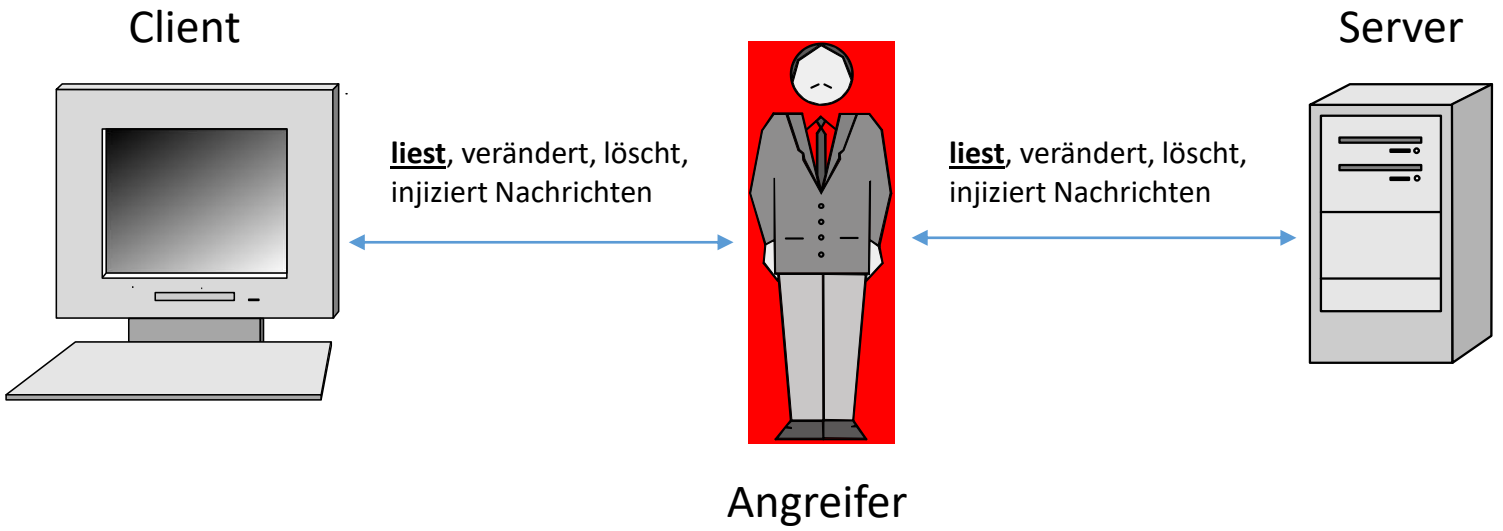


Malleability des CBC-Modus



Padding-Orakel-Angriff auf RFC 2440

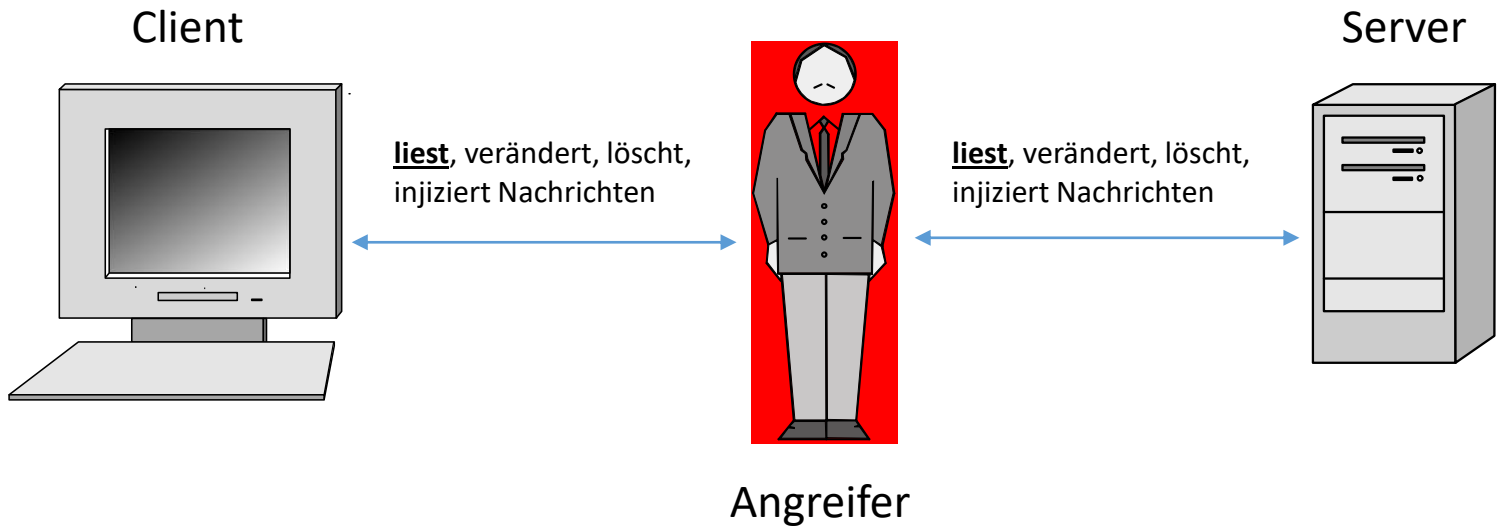
Man-in-the-Middle



Padding-Orakel-Angriff auf RFC 2440

Man-in-the-Middle

Phase 1: Angreifer zeichnet einen CBC-verschlüsselten Chiffretext auf

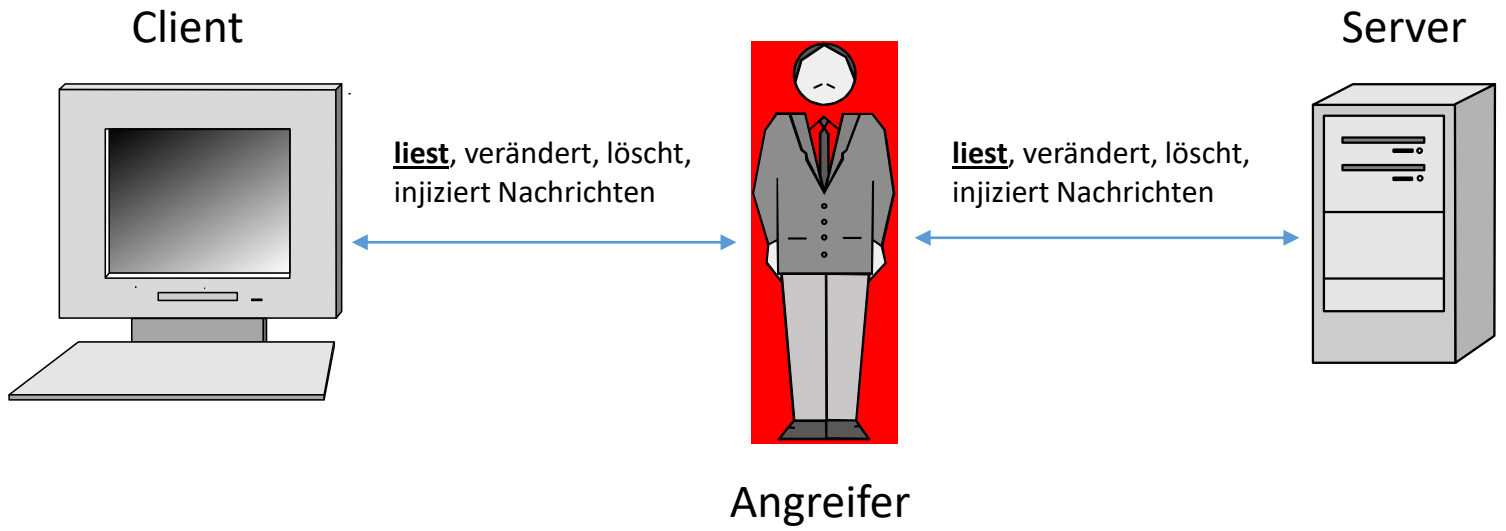


Padding-Orakel-Angriff auf RFC 2440

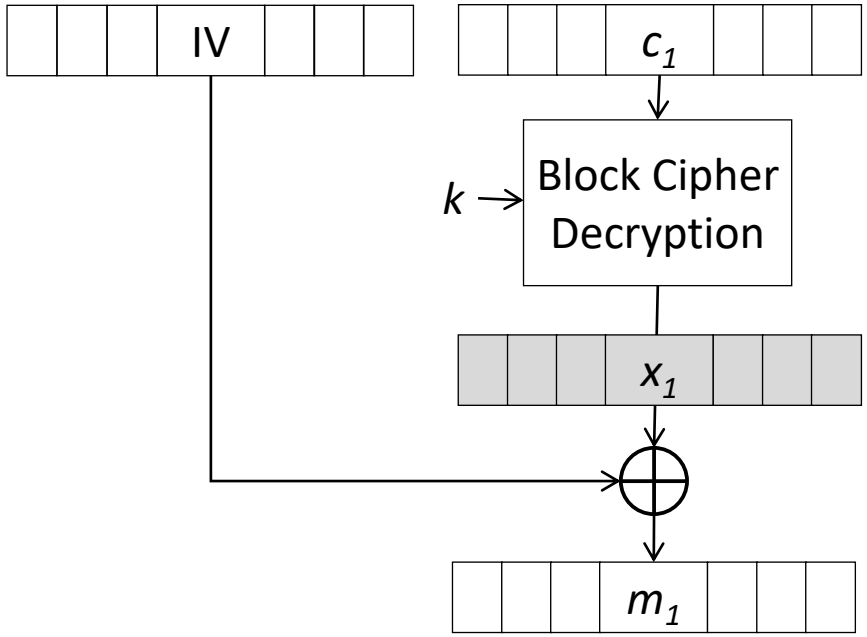
Man-in-the-Middle

Phase 1: Angreifer zeichnet einen CBC-verschlüsselten Chiffretext auf

Phase 2: Angreifer sendet modifizierten Chiffretext an den Server und beobachtet, ob dieser eine RFC-2440-Padding-Fehlermeldung sendet

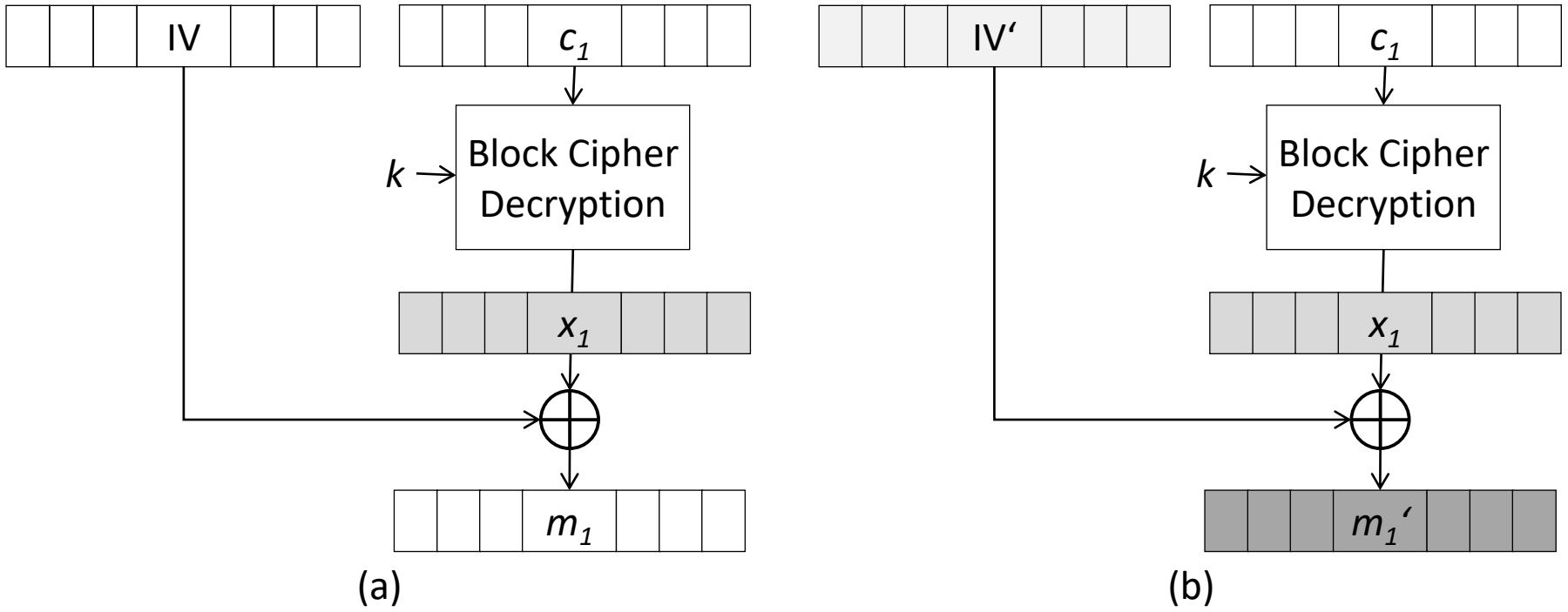


Padding-Orakel-Angriff auf RFC 2440



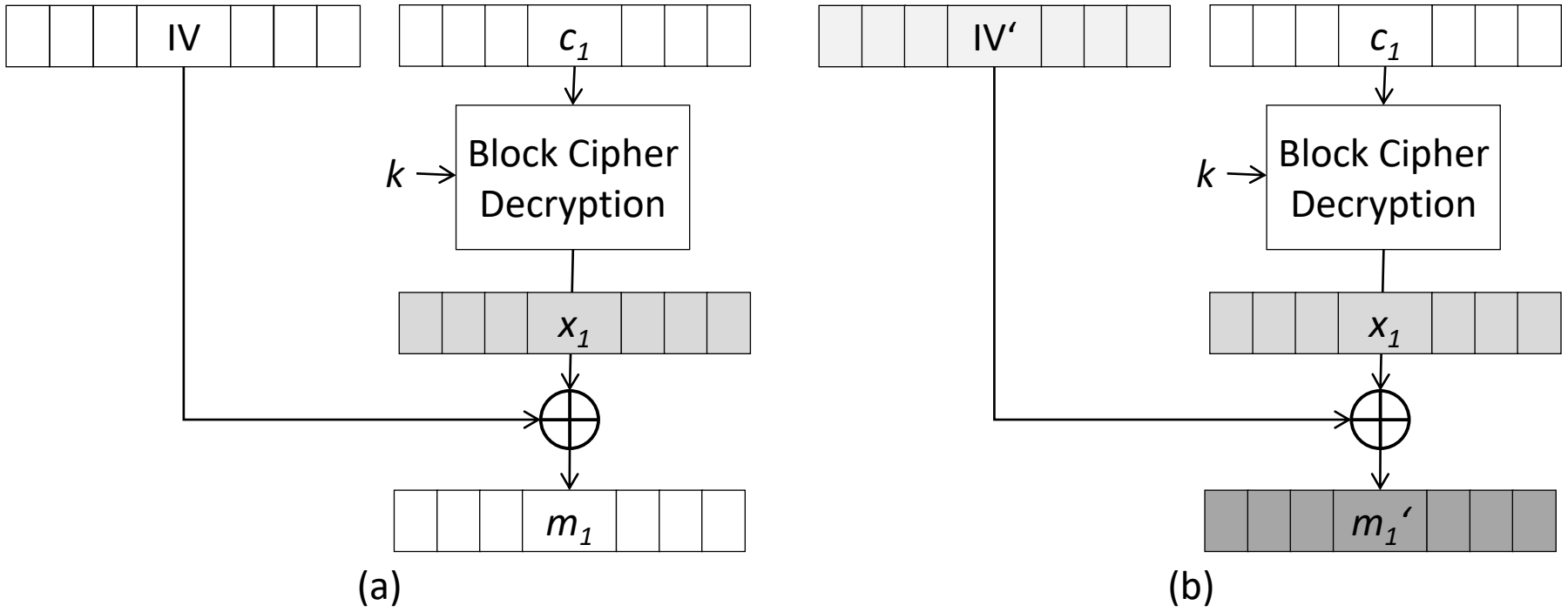
(a) Originalchiffretext mit Original-IV

Padding-Orakel-Angriff auf RFC 2440



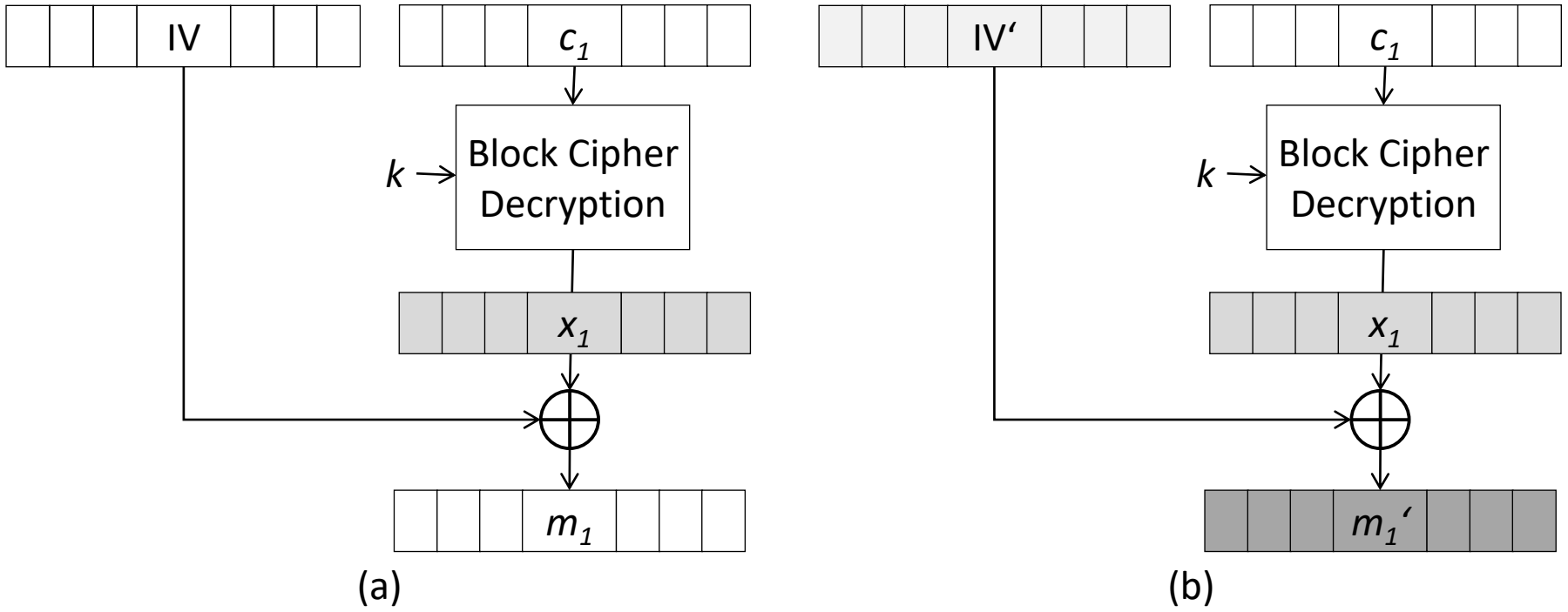
(b) Im ersten Schritt wählen wir einen völlig neuen IV'

Padding-Orakel-Angriff auf RFC 2440



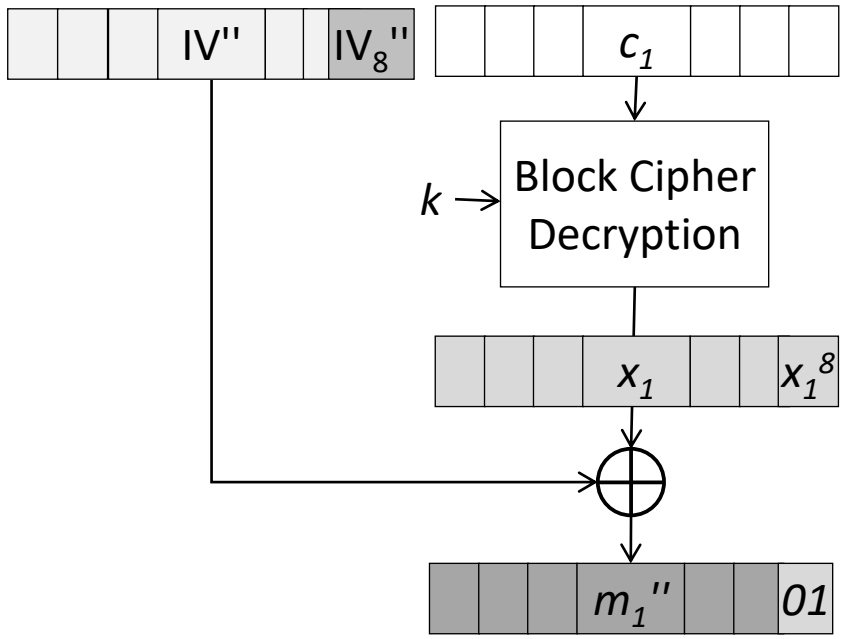
Wenn wir den IV ändern, ändert sich der Klartext;

Padding-Orakel-Angriff auf RFC 2440



Wenn wir den IV ändern, ändert sich der Klartext; der Zwischenwert x_1 bleibt aber gleich!

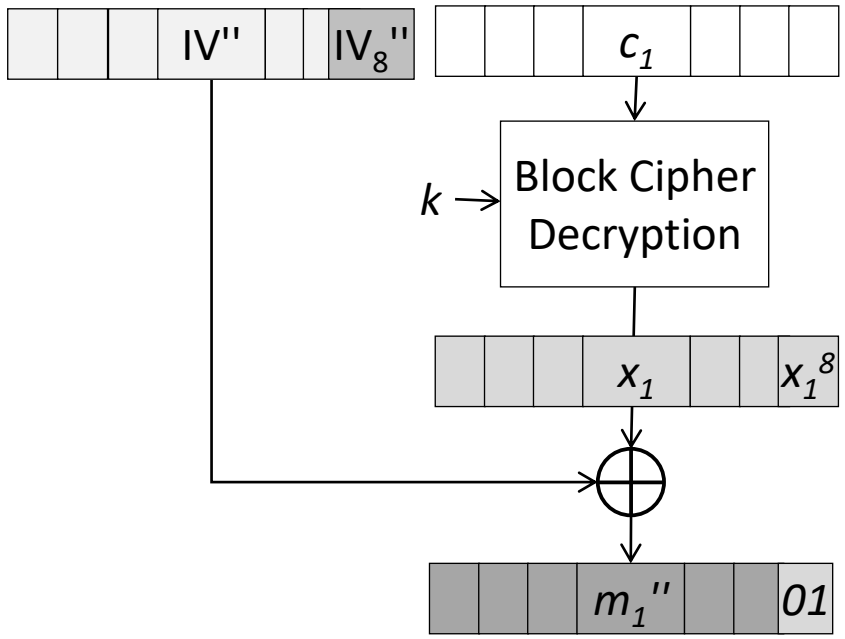
Padding-Orakel-Angriff auf RFC 2440



(c)

(c) Wir variieren das letzte Byte des IV so lange, bis der Server keinen Padding-Fehler mehr meldet

Padding-Orakel-Angriff auf RFC 2440

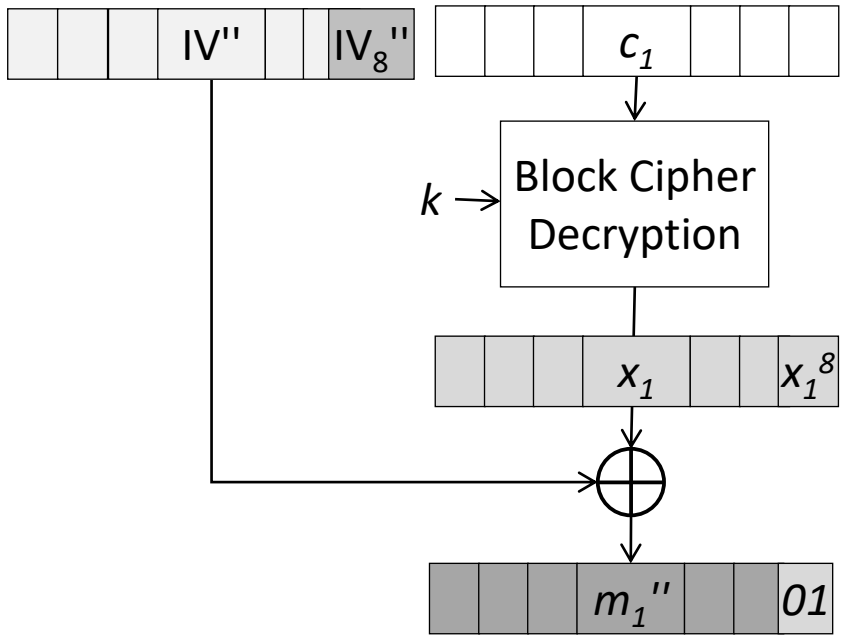


(c)

Das Padding ist also jetzt korrekt, es gibt mehrere Möglichkeiten

(c) Wir variieren das letzte Byte des IV so lange, bis der Server keinen Padding-Fehler mehr meldet

Padding-Orakel-Angriff auf RFC 2440



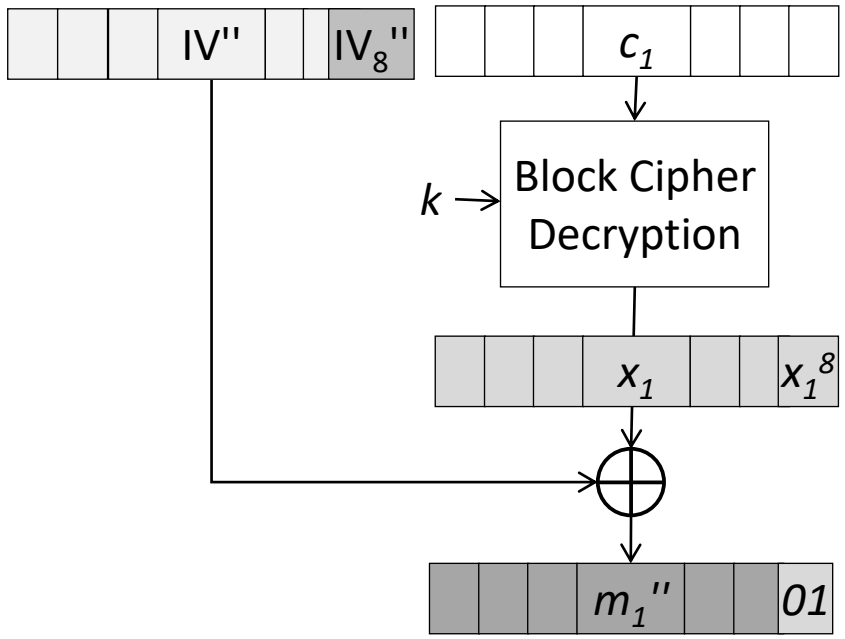
(c)

Das Padding ist also jetzt korrekt, es gibt mehrere Möglichkeiten

1. pad = 01 (tritt mit W. $1/2^8$ auf)

(c) Wir variieren das letzte Byte des IV so lange, bis der Server keinen Padding-Fehler mehr meldet

Padding-Orakel-Angriff auf RFC 2440



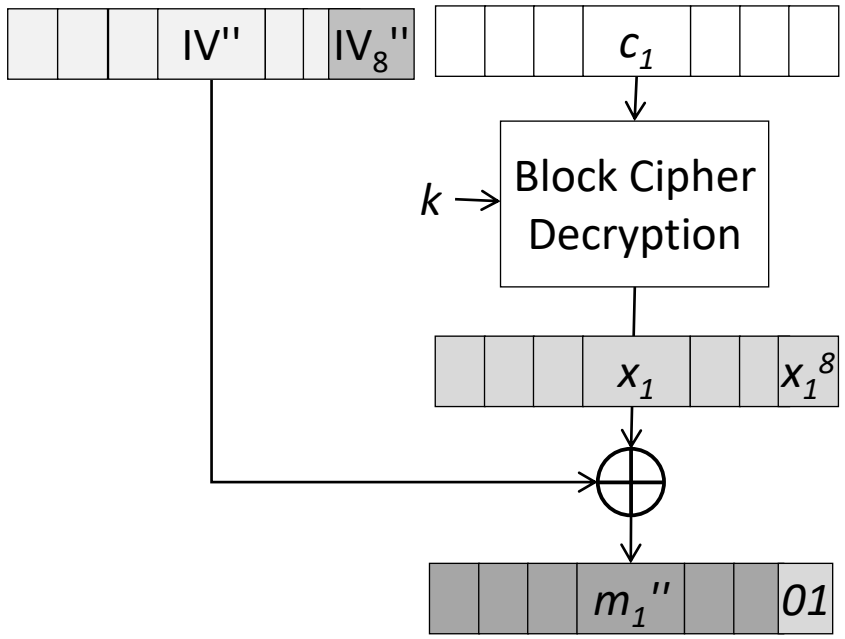
(c)

Das Padding ist also jetzt korrekt, es gibt mehrere Möglichkeiten

1. pad = 01 (tritt mit W. $1/2^8$ auf)
2. pad = 02 02 (tritt mit W. $1/2^{16}$ auf)

(c) Wir variieren das letzte Byte des IV so lange, bis der Server keinen Padding-Fehler mehr meldet

Padding-Orakel-Angriff auf RFC 2440



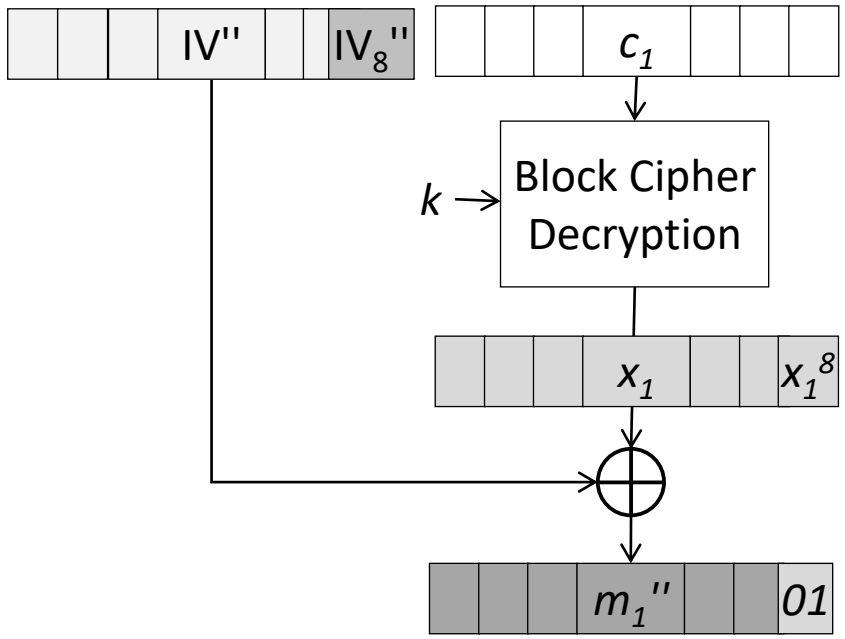
(c)

Das Padding ist also jetzt korrekt, es gibt mehrere Möglichkeiten

1. pad = 01 (tritt mit W. $1/2^8$ auf)
2. pad = 02 02 (tritt mit W. $1/2^{16}$ auf)
3. pad = 03 03 03 (tritt mit W. $1/2^{24}$ auf)
4. ...

(c) Wir variieren das letzte Byte des IV so lange, bis der Server keinen Padding-Fehler mehr meldet

Padding-Orakel-Angriff auf RFC 2440

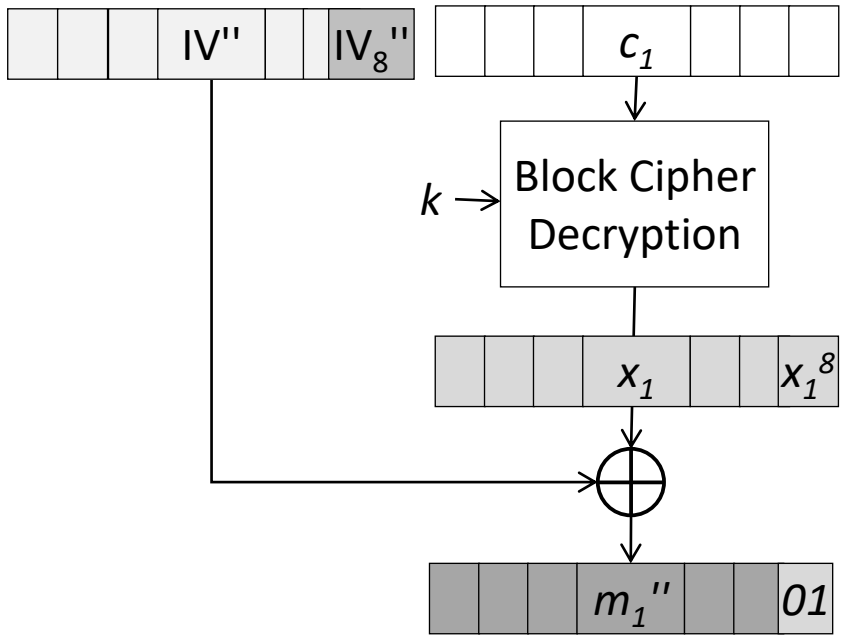


(c)

Wir vermuten, dass das wahrscheinlichste Padding pad = 01 in m_1'' enthalten ist und testen das wie folgt

(c) Wir variieren das letzte Byte des IV so lange, bis der Server keinen Padding-Fehler mehr meldet

Padding-Orakel-Angriff auf RFC 2440



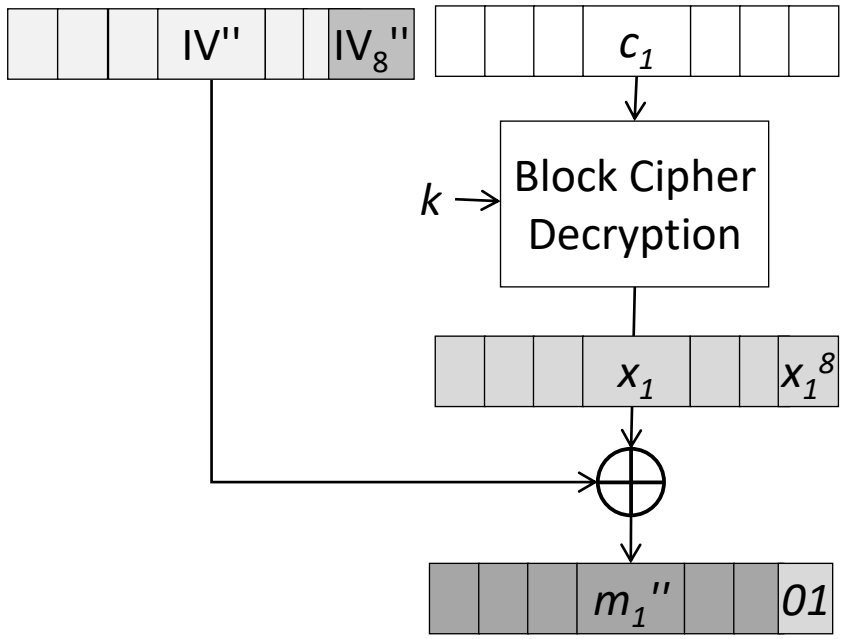
(c)

Wir vermuten, dass das wahrscheinlichste Padding pad = 01 in m_1'' enthalten ist und testen ads wie folgt

- wir flippen ein beliebiges Bit im vorletzten Byte von IV''

(c) Wir variieren das letzte Byte des IV so lange, bis der Server keinen Padding-Fehler mehr meldet

Padding-Orakel-Angriff auf RFC 2440



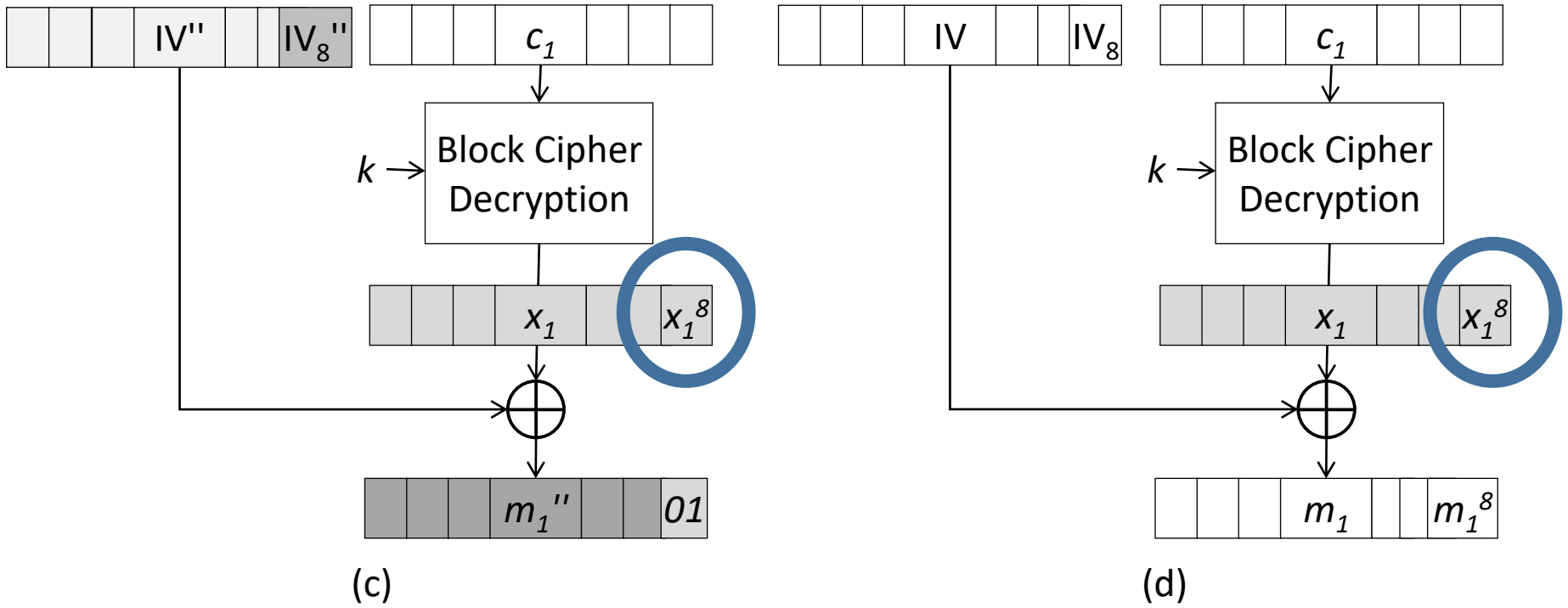
(c)

Wir vermuten, dass das wahrscheinlichste Padding pad = 01 in m_1'' enthalten ist und testen das wie folgt

- wir flippen ein beliebiges Bit im vorletzten Byte von IV''
- Wenn das Padding korrekt bleibt, haben wir pad = 01 verifiziert

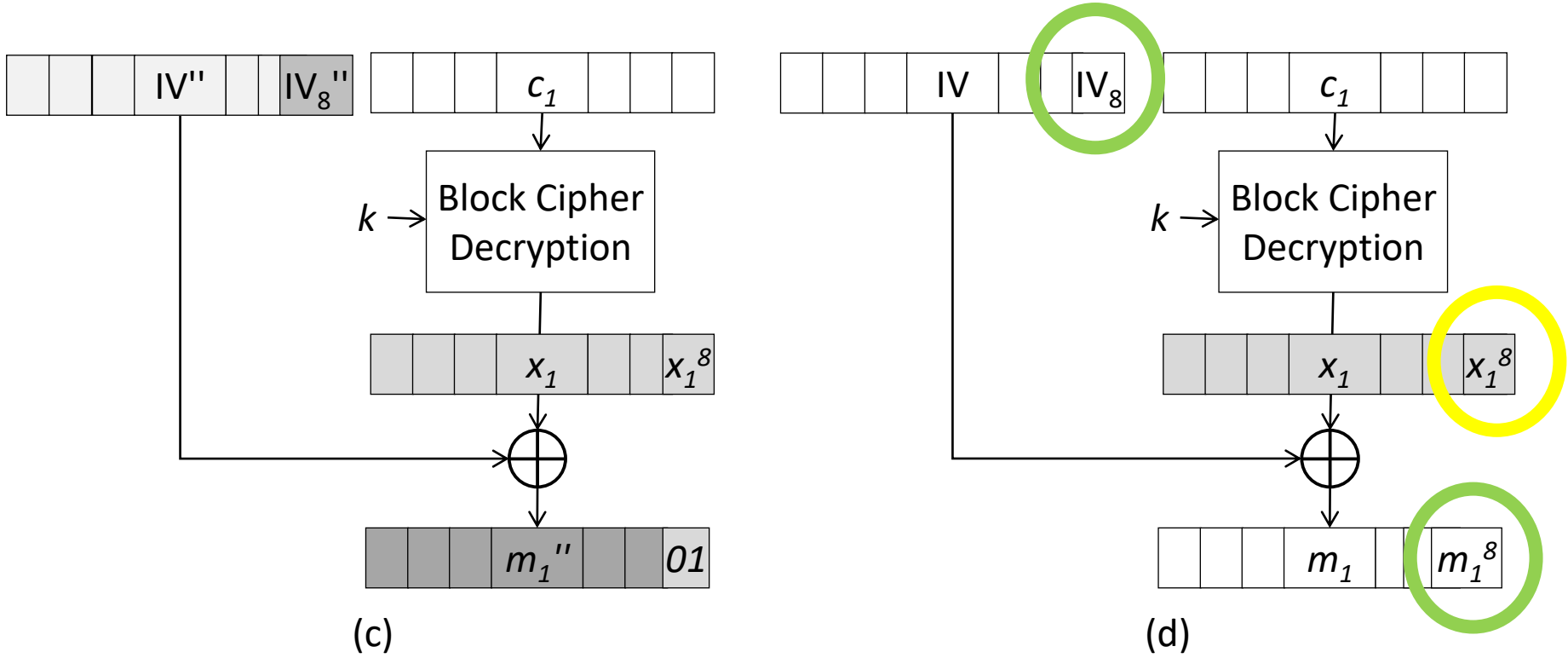
(c) Wir variieren das letzte Byte des IV so lange, bis der Server keinen Padding-Fehler mehr meldet

Padding-Orakel-Angriff auf RFC 2440



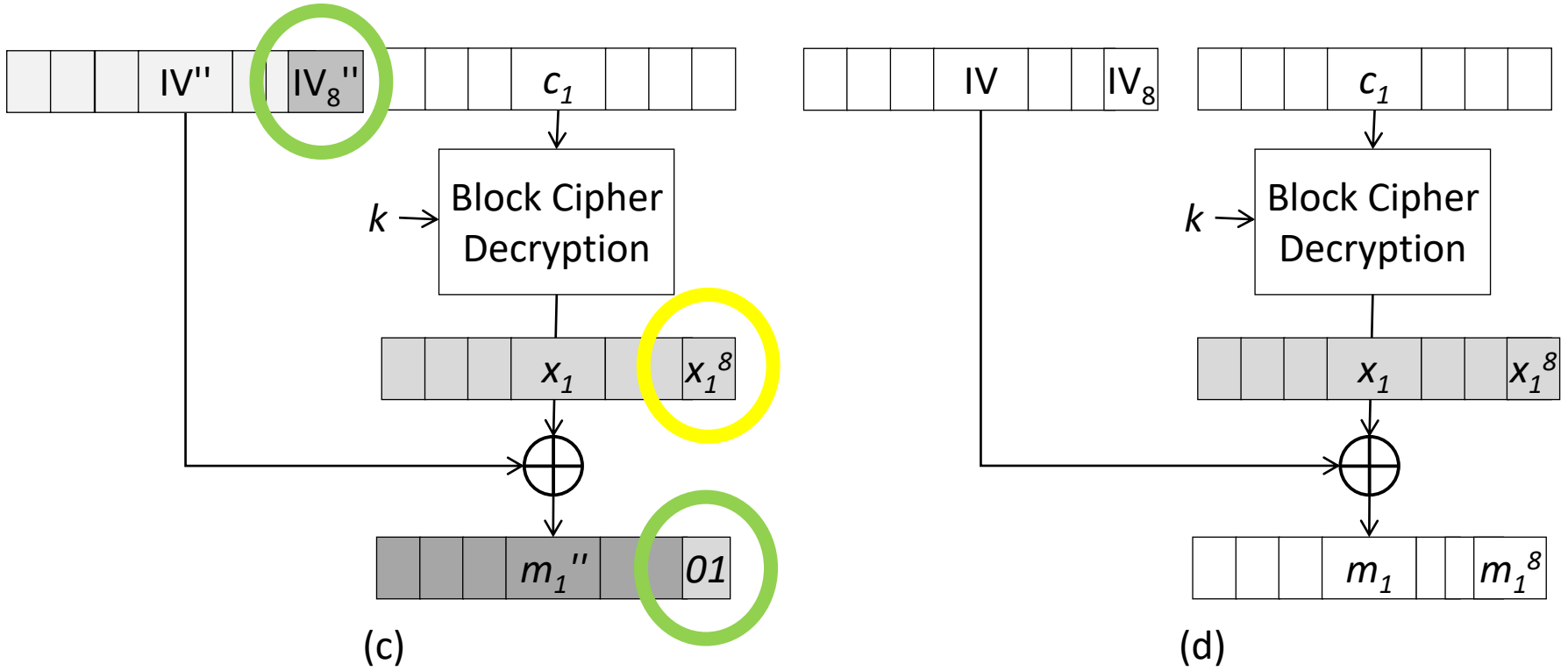
Der Wert x_1^8 wird durch all diese Manipulationen am IV nicht geändert.

Padding-Orakel-Angriff auf RFC 2440



Wir können das letzte Klartextbyte m_1^8 jetzt wie folgt berechnen: $m_1^8 = x_1^8 \oplus IV_8$

Padding-Orakel-Angriff auf RFC 2440



Wir können das letzte Klartextbyte m_1^8 jetzt wie folgt berechnen: $m_1^8 = x_1^8 \oplus IV_8 = (01 \oplus IV_8) \oplus IV_8$''

Vaudenay: Berechnung weiterer Byte

- Wir wissen jetzt, dass mit IV_8'' das letzte Klartextbyte den Wert 01 hat

Vaudenay: Berechnung weiterer Byte

- Wir wissen jetzt, dass mit IV_8'' das letzte Klartextbyte den Wert 01 hat
- Wir flippen die letzten beiden Bit in IV_8''; das letzte Klartextbyte hat jetzt den Wert 02

Vaudenay: Berechnung weiterer Byte

- Wir wissen jetzt, dass mit IV_8'' das letzte Klartextbyte den Wert 01 hat
- Wir flippen die letzten beiden Bit in IV_8''; das letzte Klartextbyte hat jetzt den Wert 02
- Wir fixieren diesen neuen Wert für IV_8'' und variieren das vorletzte Byte IV_7', bis für einen Wert IV_7'' KEIN Padding-Fehler mehr auftritt

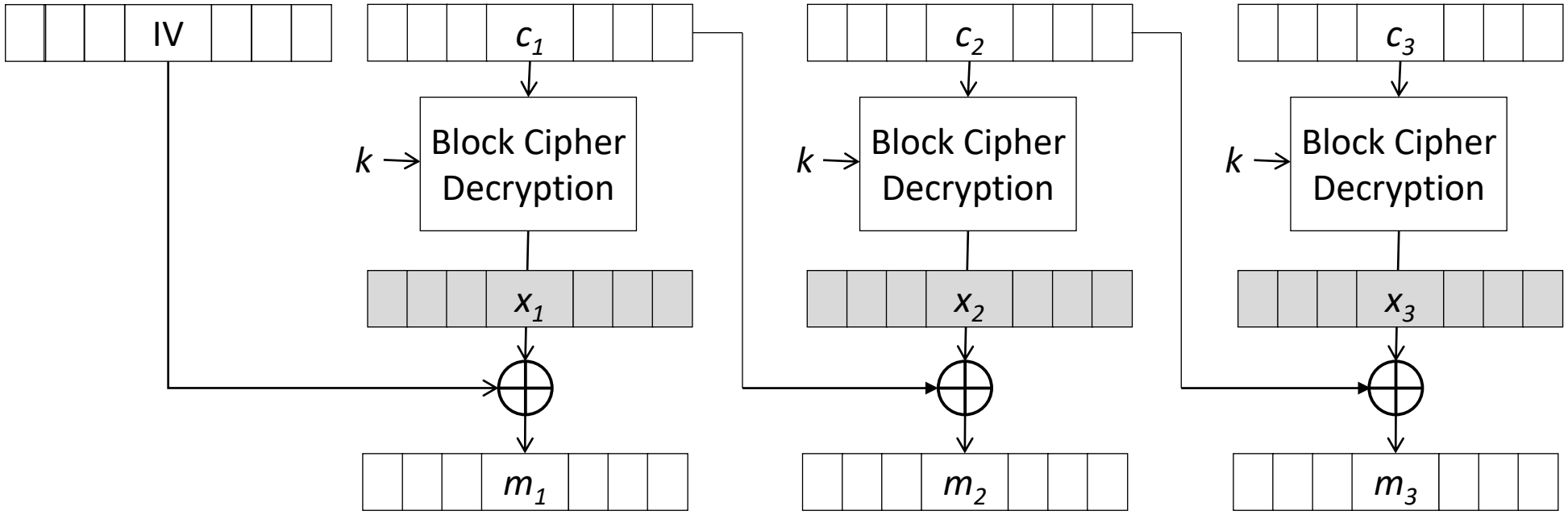
Vaudenay: Berechnung weiterer Byte

- Wir wissen jetzt, dass mit IV_8'' das letzte Klartextbyte den Wert 01 hat
- Wir flippen die letzten beiden Bit in IV_8''; das letzte Klartextbyte hat jetzt den Wert 02
- Wir fixieren diesen neuen Wert für IV_8'' und variieren das vorletzte Byte IV_7', bis für einen Wert IV_7'' KEIN Padding-Fehler mehr auftritt
  - dann ist das Padding im neuen Klartext pad = 02 02

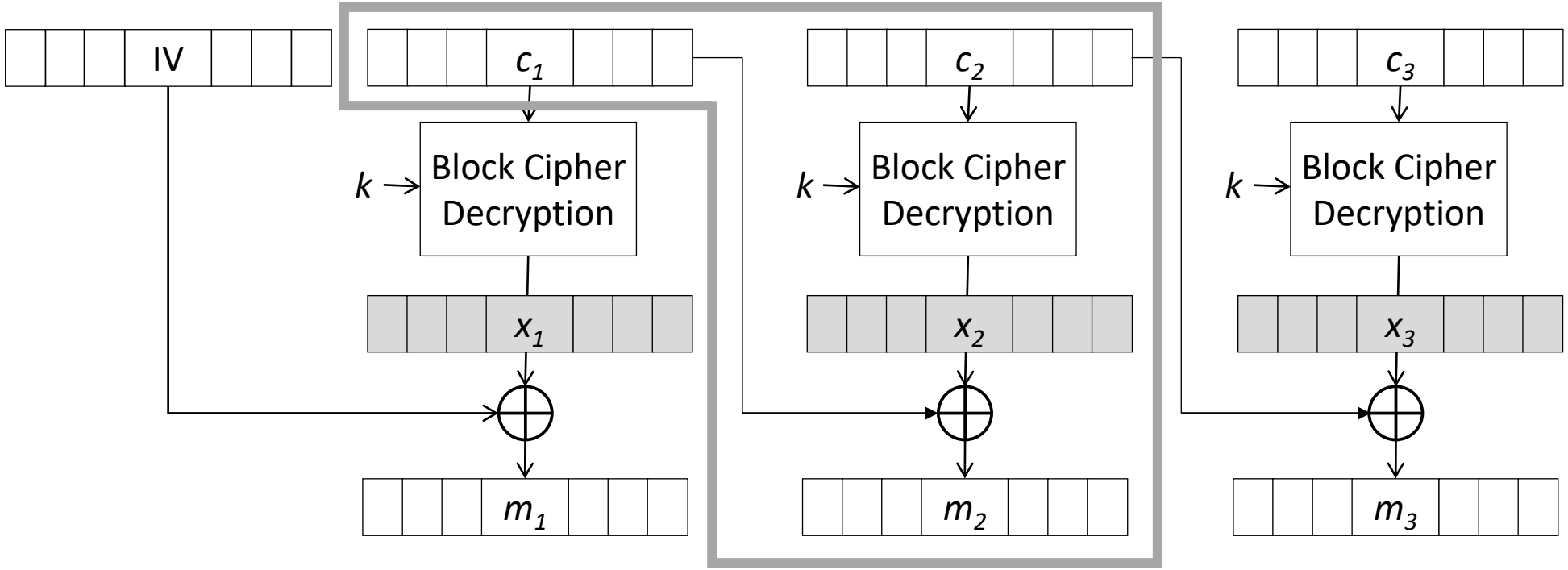
Vaudenay: Berechnung weiterer Byte

- Wir wissen jetzt, dass mit IV_8'' das letzte Klartextbyte den Wert 01 hat
- Wir flippen die letzten beiden Bit in IV_8''; das letzte Klartextbyte hat jetzt den Wert 02
- Wir fixieren diesen neuen Wert für IV_8'' und variieren das vorletzte Byte IV_7', bis für einen Wert IV_7'' KEIN Padding-Fehler mehr auftritt
  - dann ist das Padding im neuen Klartext pad = 02 02
  - wir können m_1^7 jetzt analog zu oben bestimmen

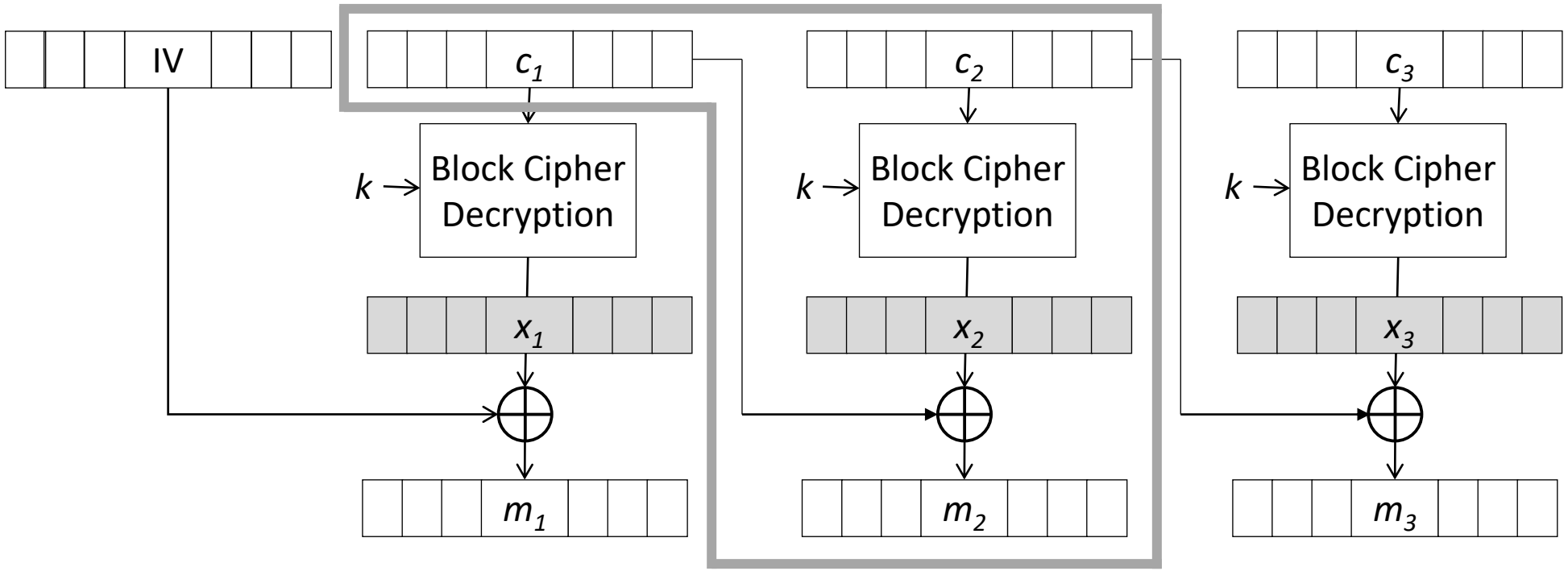
Vaudenay: Berechnung weiterer Blöcke



Vaudenay: Berechnung weiterer Blöcke



Vaudenay: Berechnung weiterer Blöcke



- Um den Klartextblock m_2 zu bestimmen verwenden wir c_1 als IV und gehen analog zu oben vor

Vaudenay: Fazit

- Anwendbar auf Verschlüsselungsverfahren mit RC 2440 Padding, bei denen der Schlüssel k 'lange' genutzt wird

Vaudenay: Fazit

- Anwendbar auf Verschlüsselungsverfahren mit RC 2440 Padding, bei denen der Schlüssel k 'lange' genutzt wird

- Nicht direkt anwendbar auf SSL/TLS

Vaudenay: Fazit

- Anwendbar auf Verschlüsselungsverfahren mit RC 2440 Padding, bei denen der Schlüssel k 'lange' genutzt wird

- Nicht direkt anwendbar auf SSL/TLS
  - SSL 3.0 verwendet anderes Padding-Verfahren

Vaudenay: Fazit

- Anwendbar auf Verschlüsselungsverfahren mit RC 2440 Padding, bei denen der Schlüssel k 'lange' genutzt wird

- Nicht direkt anwendbar auf SSL/TLS
  - SSL 3.0 verwendet anderes Padding-Verfahren
  - TLS 1.0, 1.1 und 1.2 verwenden zwar ähnliches Padding-Verfahren (k mal das Byte k-1), aber

Vaudenay: Fazit

- Anwendbar auf Verschlüsselungsverfahren mit RC 2440 Padding, bei denen der Schlüssel k 'lange' genutzt wird

- Nicht direkt anwendbar auf SSL/TLS
  - SSL 3.0 verwendet anderes Padding-Verfahren
  - TLS 1.0, 1.1 und 1.2 verwenden zwar ähnliches Padding-Verfahren (k mal das Byte k-1), aber
    - fehlerhaftes Padding impliziter den 'fatalen' bad_record_mac-Alert (Hausaufgabe: Warum?)

Vaudenay: Fazit

- Anwendbar auf Verschlüsselungsverfahren mit RC 2440 Padding, bei denen der Schlüssel k 'lange' genutzt wird

- Nicht direkt anwendbar auf SSL/TLS
  - SSL 3.0 verwendet anderes Padding-Verfahren
  - TLS 1.0, 1.1 und 1.2 verwenden zwar ähnliches Padding-Verfahren (k mal das Byte k-1), aber
    - fehlerhaftes Padding impliziter den 'fatalen' bad_record_mac-Alert (Hausaufgabe: Warum?)
    - Schlüssel k wird dann gelöscht

Vaudenay: Fazit

- Anwendbar auf Verschlüsselungsverfahren mit RC 2440 Padding, bei denen der Schlüssel k 'lange' genutzt wird

- Nicht direkt anwendbar auf SSL/TLS
  - SSL 3.0 verwendet anderes Padding-Verfahren
  - TLS 1.0, 1.1 und 1.2 verwenden zwar ähnliches Padding-Verfahren (k mal das Byte k-1), aber
    - fehlerhaftes Padding löst den 'fatalen' bad_record_mac'-Alert aus
    - Schlüssel k wird dann gelöscht
- Funktioniert in DTLS weil da Alerts meist nicht 'fatal' sind

[PA12] Kenneth G. Paterson and Nadhem J. AlFardan. Plaintext-recovery attacks against datagram TLS. In *ISOC Network and Distributed System Security Symposium – NDSS 2012*, San Diego, CA, USA, February 5–8, 2012. The Internet Society.

3.3 Angriffe auf den Record Layer

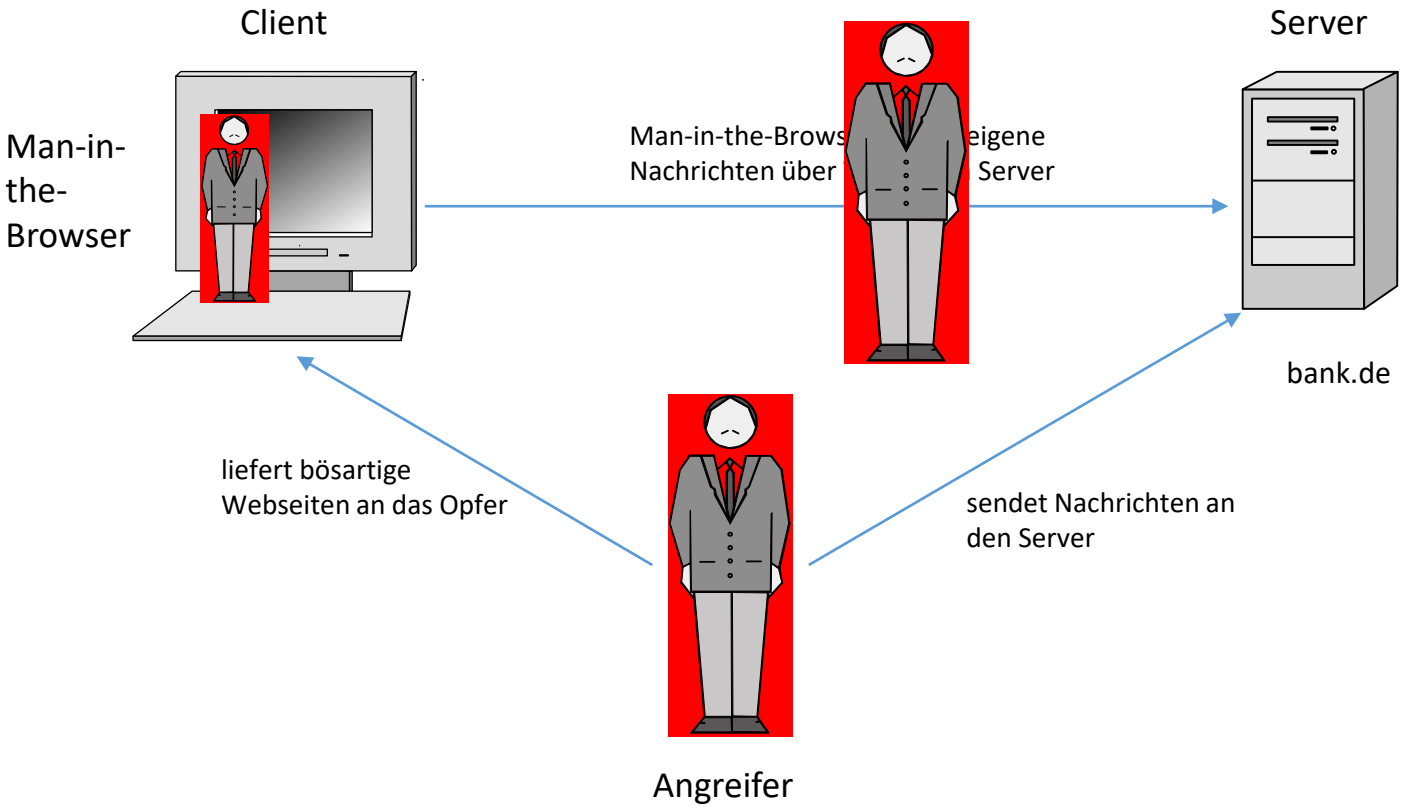
3.3.3 Padding-Oracle-Angriffe: POODLE

POODLE: Padding Oracle On Downgrade Legacy Encryption

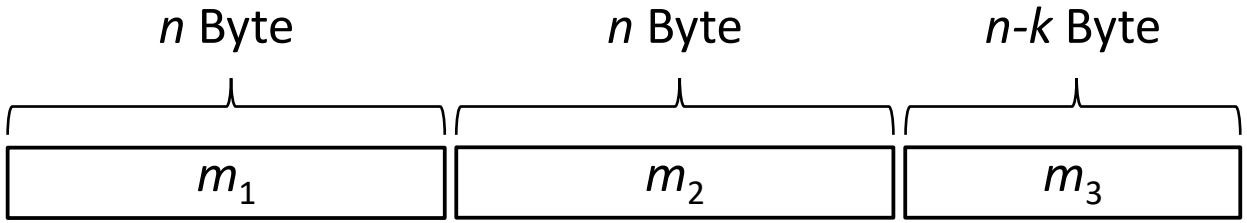
[MDK14] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. This poodle bytes: Exploiting the ssl 3.0 fallback. `https://www.openssl.org/~bodo/ssl-poodle.pdf`, September 2014.

POODLE = BEAST-Angreifermodell

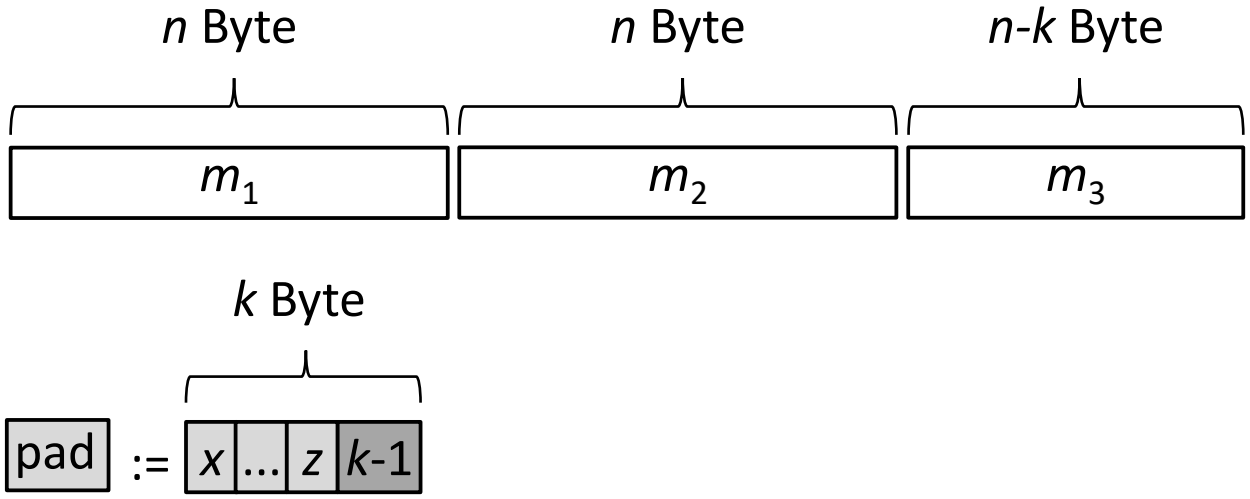
Man-in-the-Browser +
Man-in-the-Middle



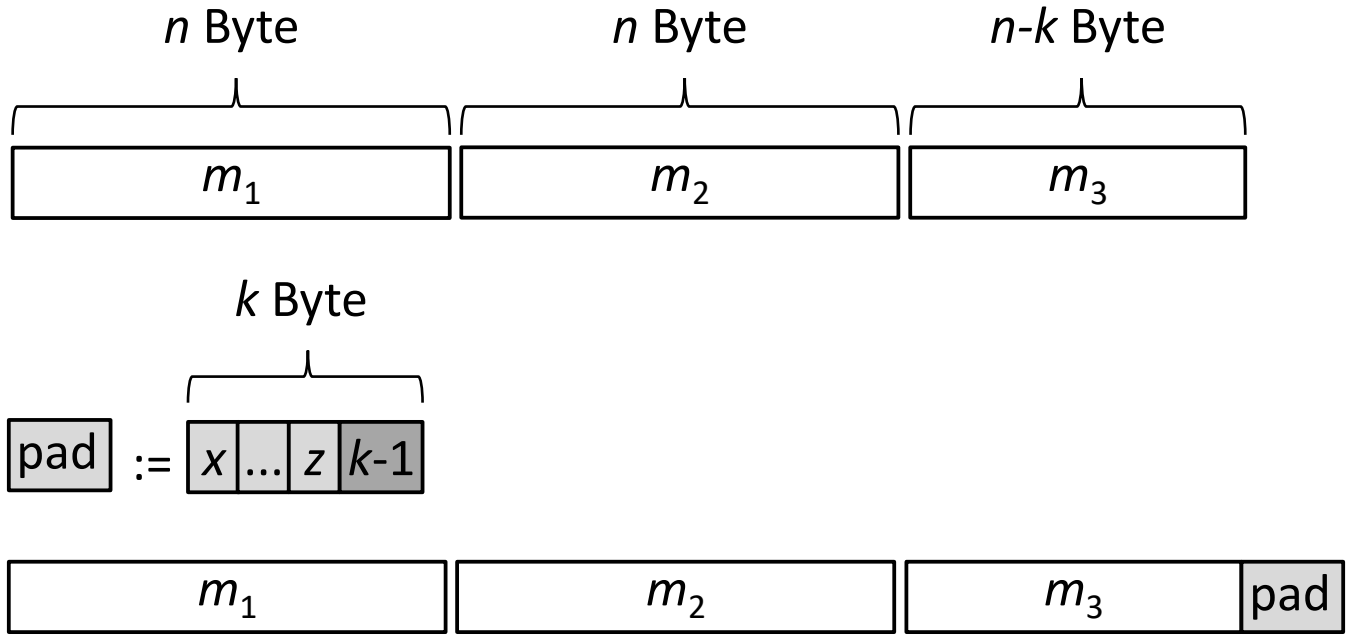
Padding in SSL 3.0



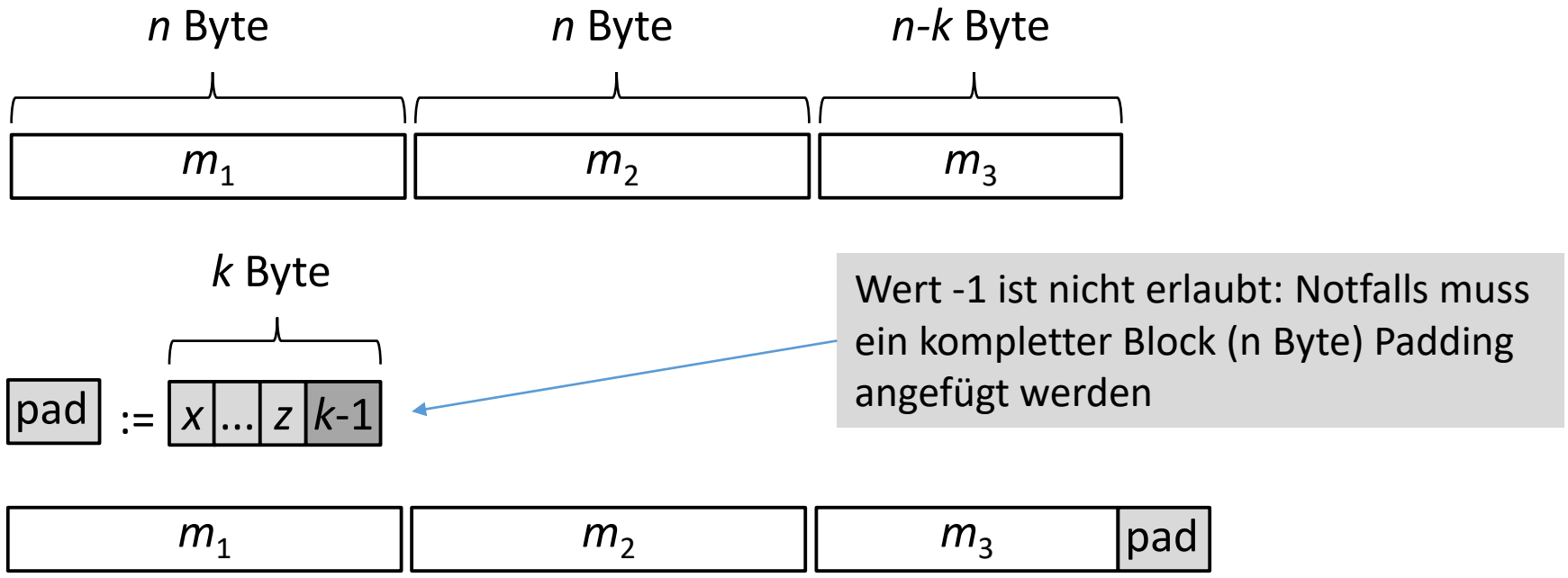
Padding in SSL 3.0



Padding in SSL 3.0



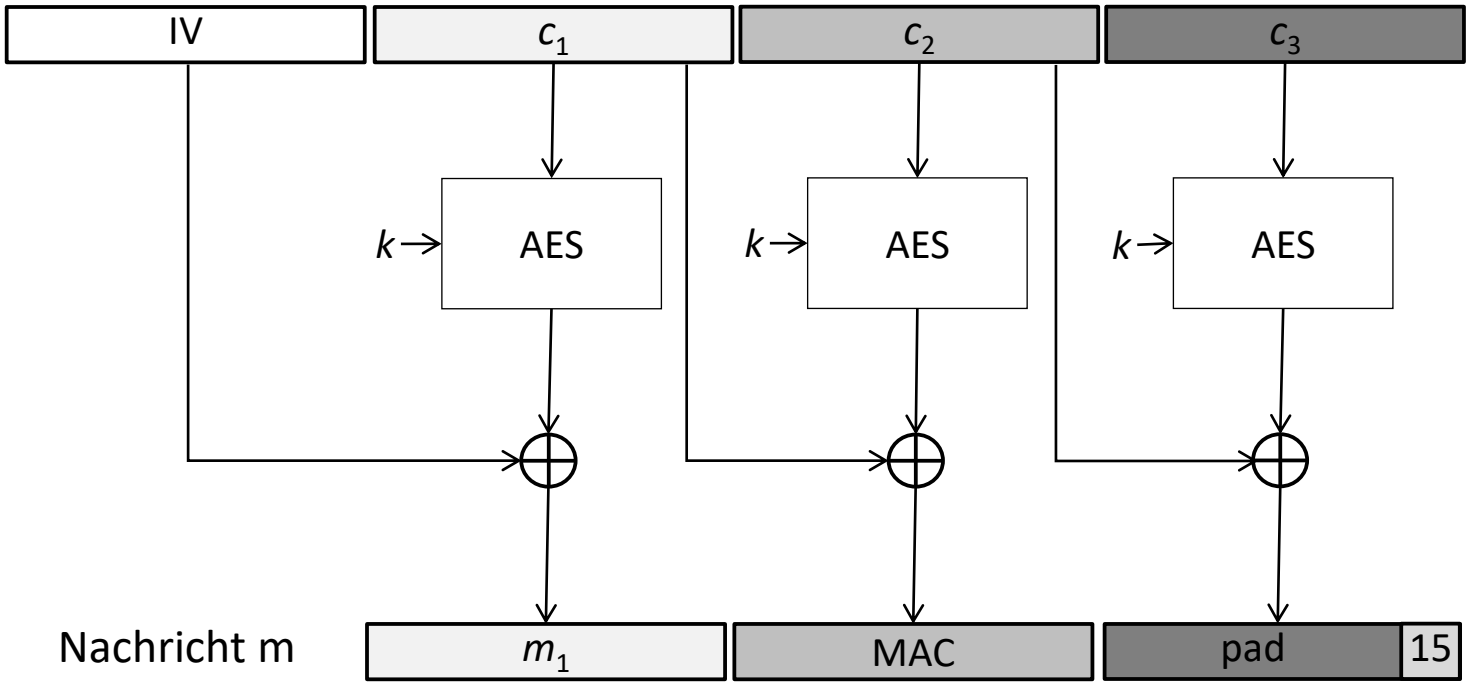
Padding in SSL 3.0



POODLE-Angriff

POODLE

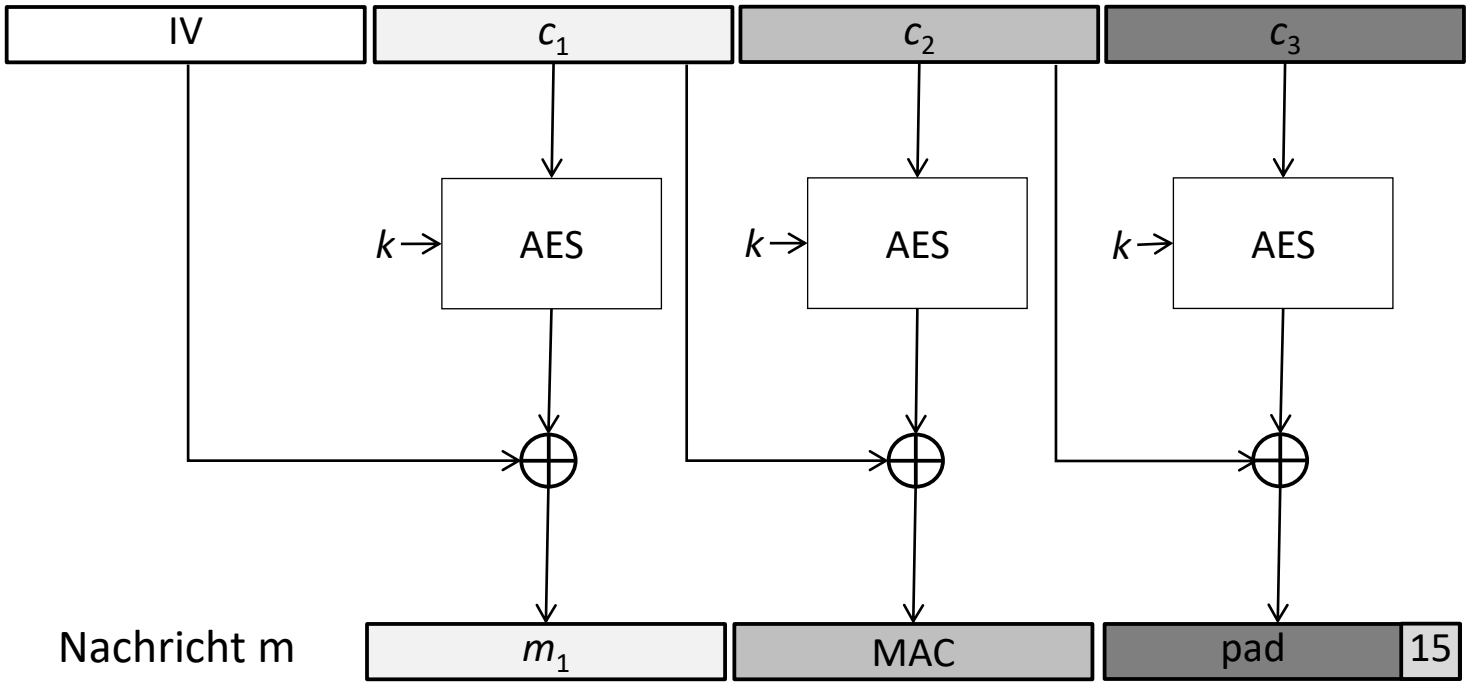
- Ziel des Angriffs ist es, wiederholt gesendete Klartexte zu entschlüsseln



POODLE-Angriff

POODLE

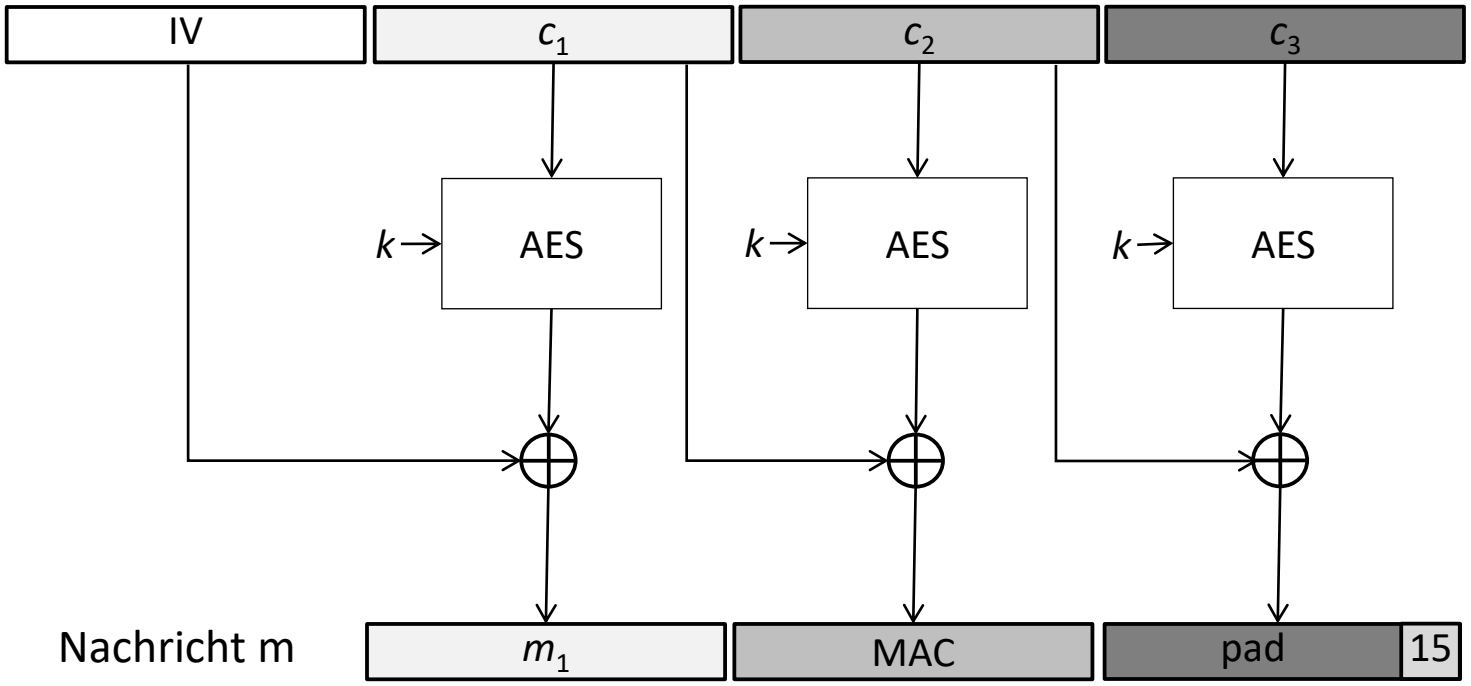
- Ziel des Angriffs ist es, wiederholt gesendete Klartexte zu entschlüsseln
- Beispiel: HTTP Session Cookies



POODLE-Angriff

POODLE

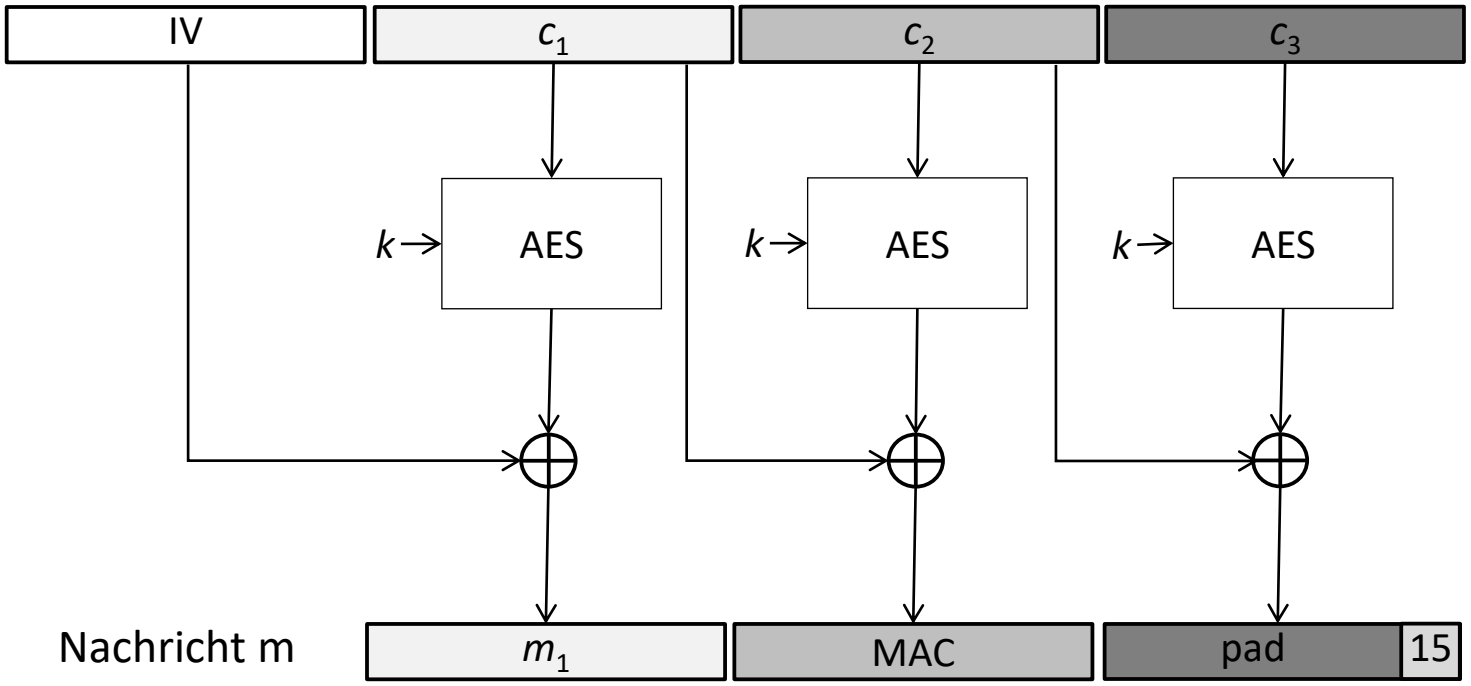
1. Das Opfer lädt die Webseite des Angreifers



POODLE-Angriff

POODLE

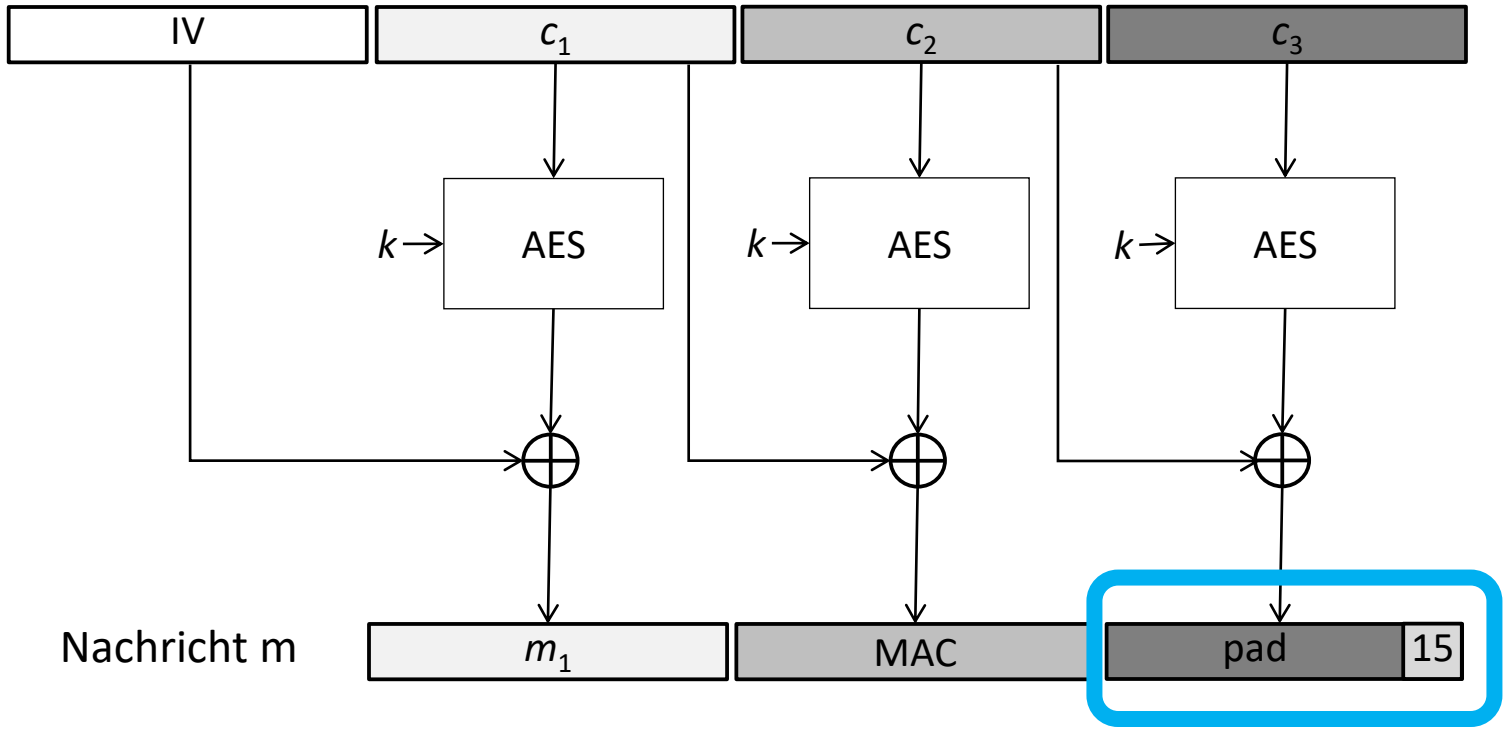
1. Das Opfer lädt die Webseite des Angreifers
2. Der MITB-Angreifer ruft eine speziell präparierte URL auf.



POODLE-Angriff

POODLE

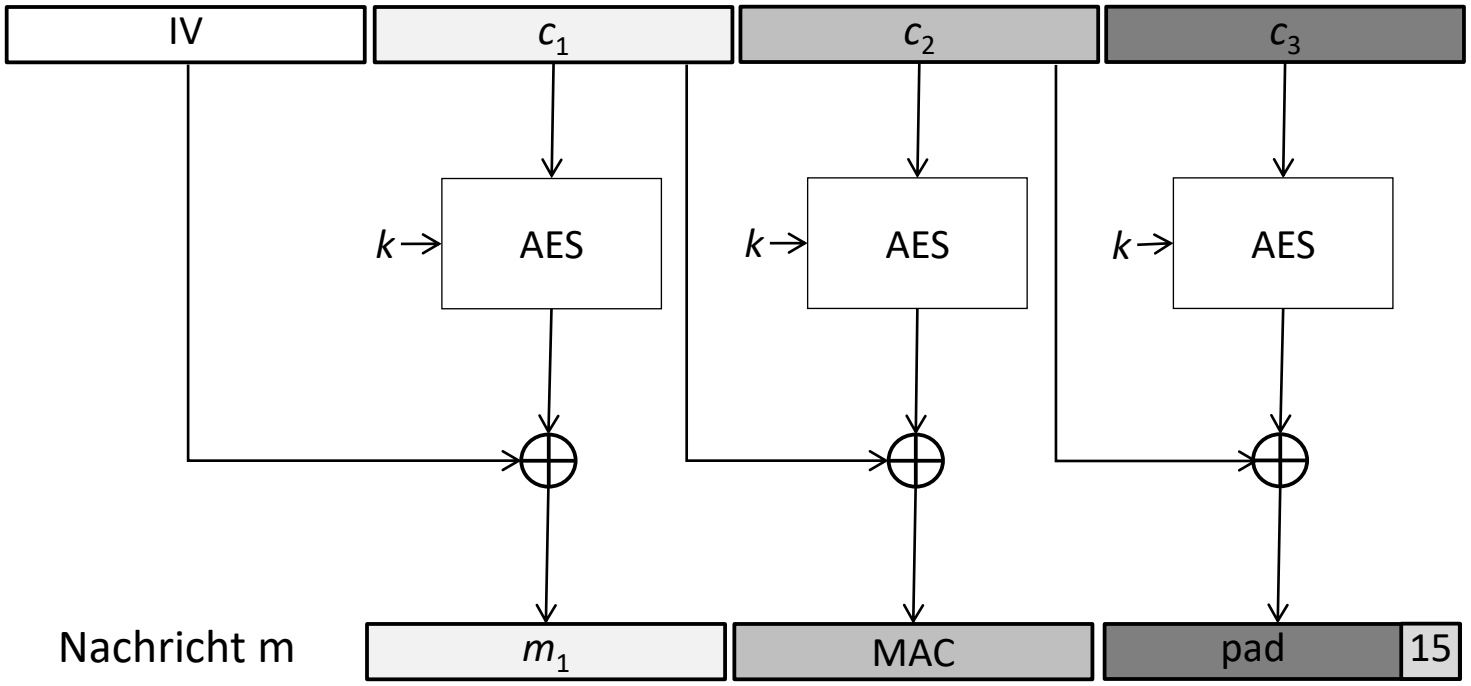
1. Das Opfer lädt die Webseite des Angreifers
2. Der MITB-Angreifer ruft eine speziell präparierte URL auf. Diese URL stellt sicher, dass ein kompletter Block (8 bzw. 16 Byte) Padding angefügt wird



POODLE-Angriff

POODLE

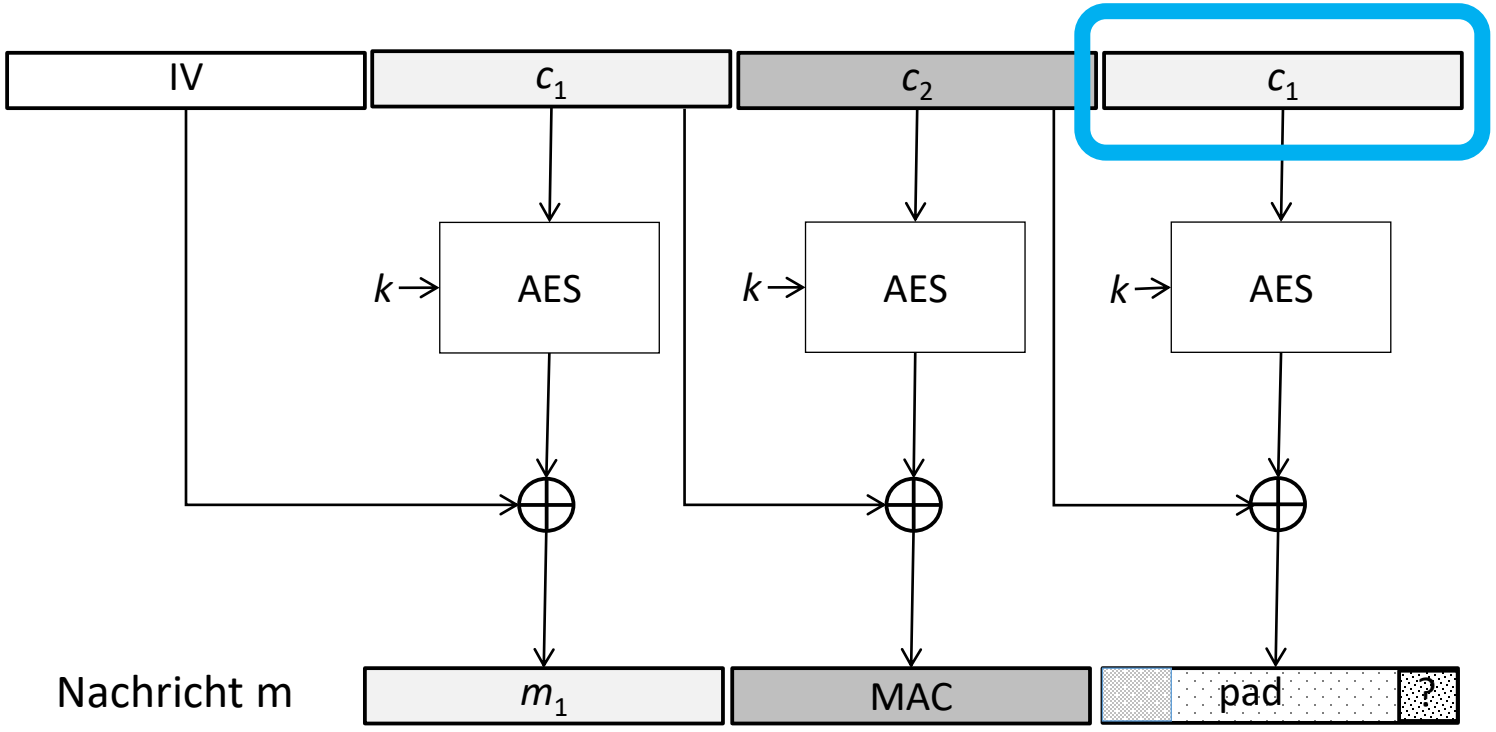
1. Das Opfer lädt die Webseite des Angreifers
2. Der MITB-Angreifer ruft eine speziell präparierte URL auf. Diese URL stellt sicher, dass ein kompletter Block (8 bzw. 16 Byte) Padding angefügt wird
3. Der Angreifer fängt den TLS Record ab



POODLE-Angriff

POODLE

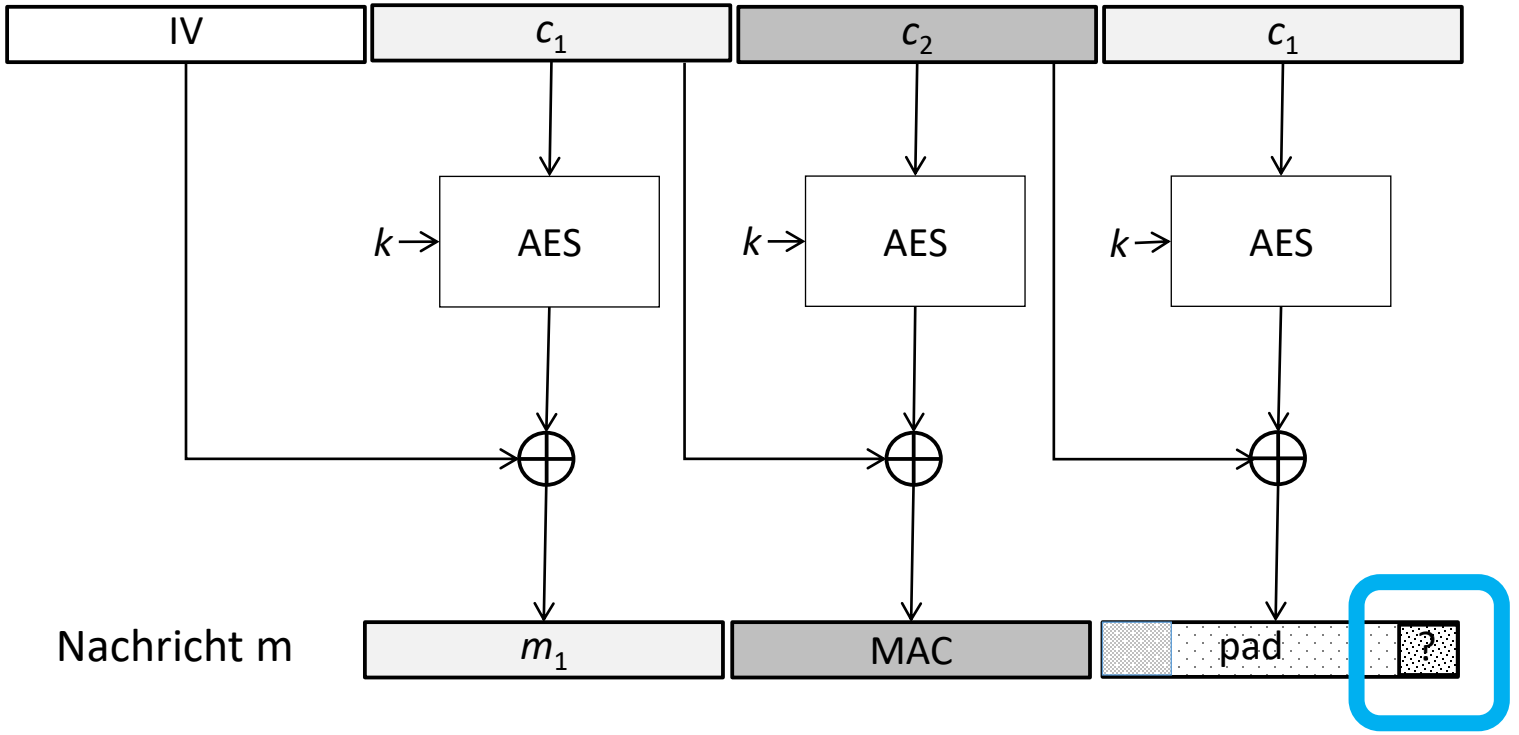
4. Der Angreifer ersetzt c_3 durch c_1



POODLE-Angriff

POODLE

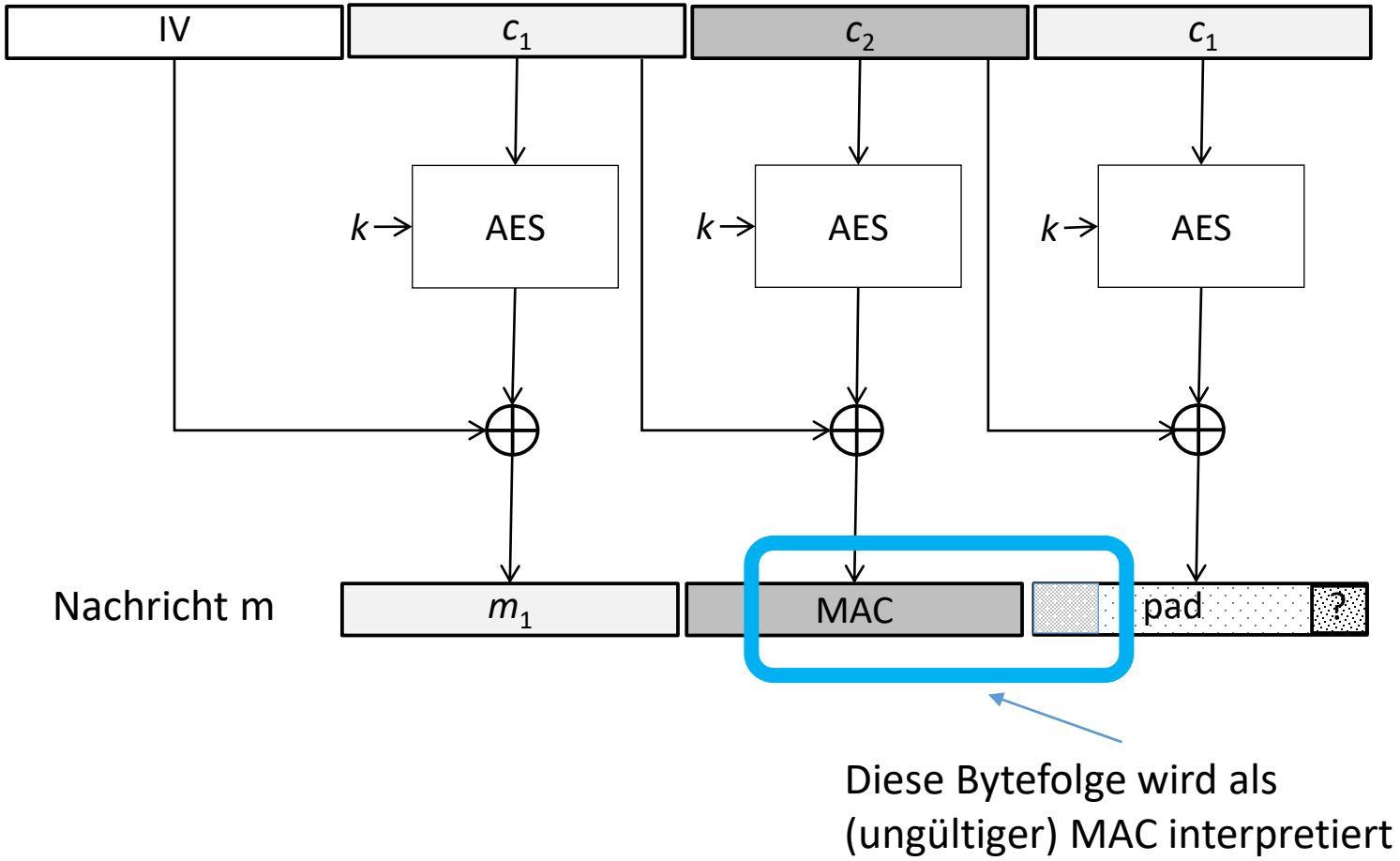
4. Der Angreifer ersetzt c_3 durch c_1
5. Nach der Entschlüsselung wird das letzte Byte als Paddinglänge interpretiert



POODLE-Angriff

POODLE

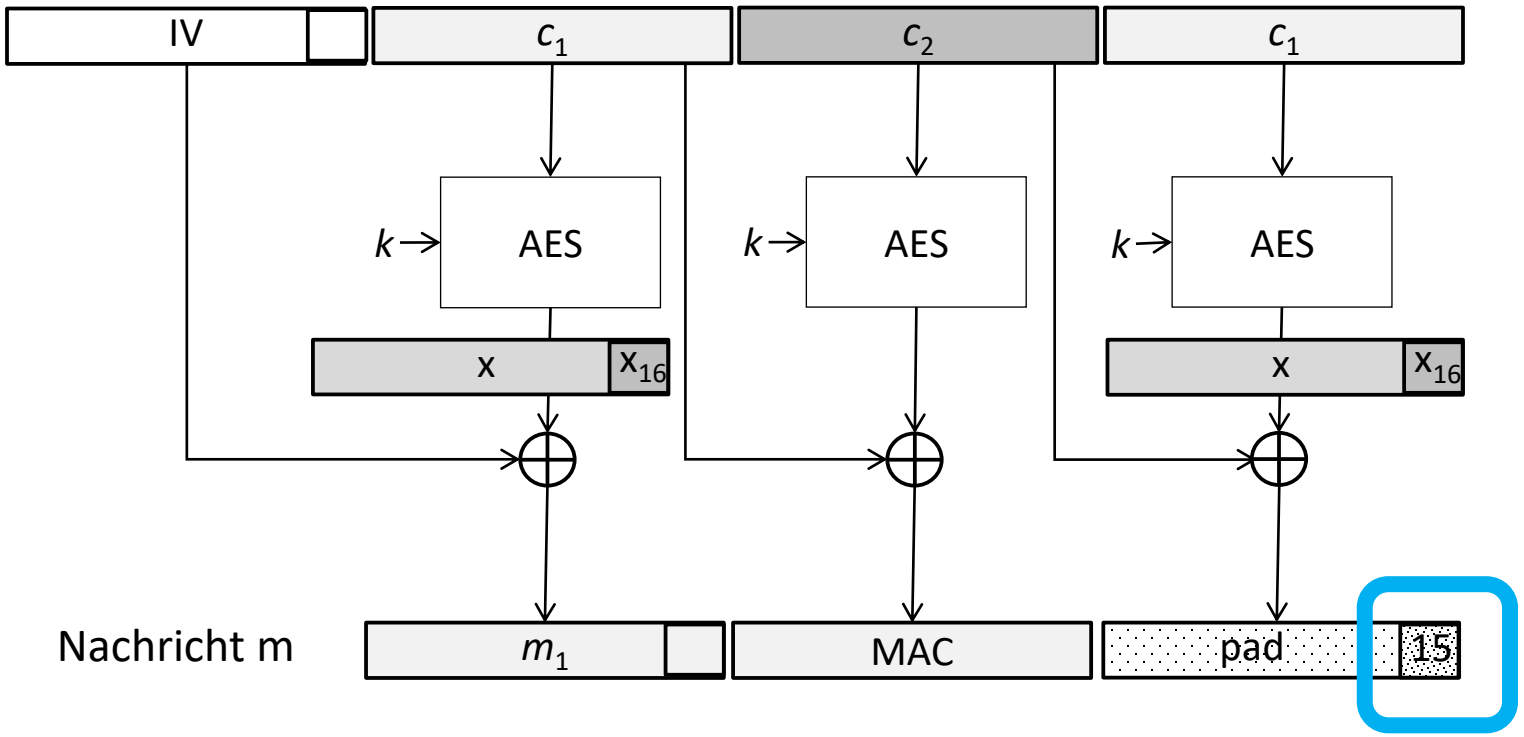
4. Der Angreifer ersetzt c_3 durch c_1
5. Nach der Entschlüsselung wird das letzte Byte als Paddinglänge interpretiert
6. Hat dieses Byte einen Wert ≠ 15 (AES) bzw 7 (3DES), so wird nach Entfernen des Padding der MAC nicht gefunden



POODLE-Angriff

POODLE

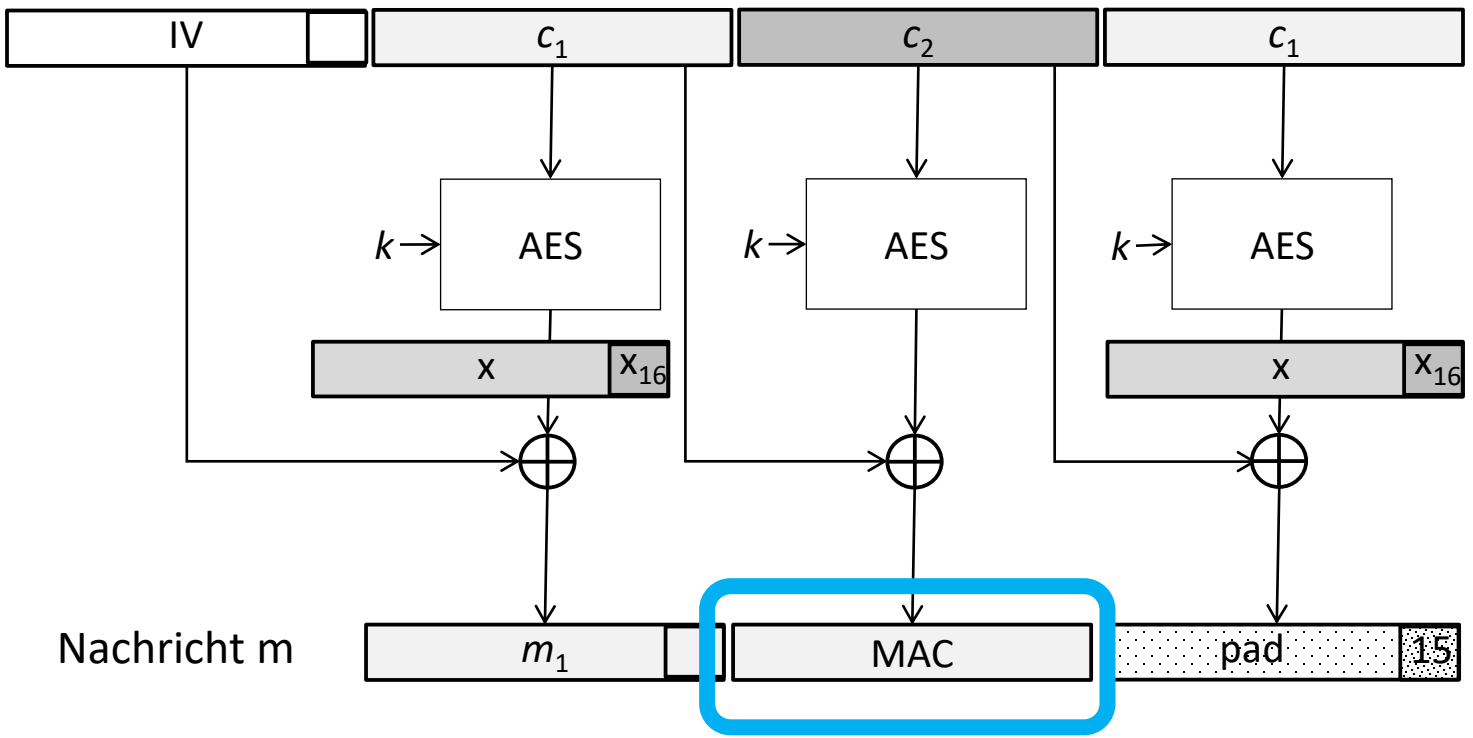
7. Mit Wahrscheinlichkeit 1/256 enthält das letzte Byte den korrekten Wert 15 (bzw. 7)



POODLE-Angriff

POODLE

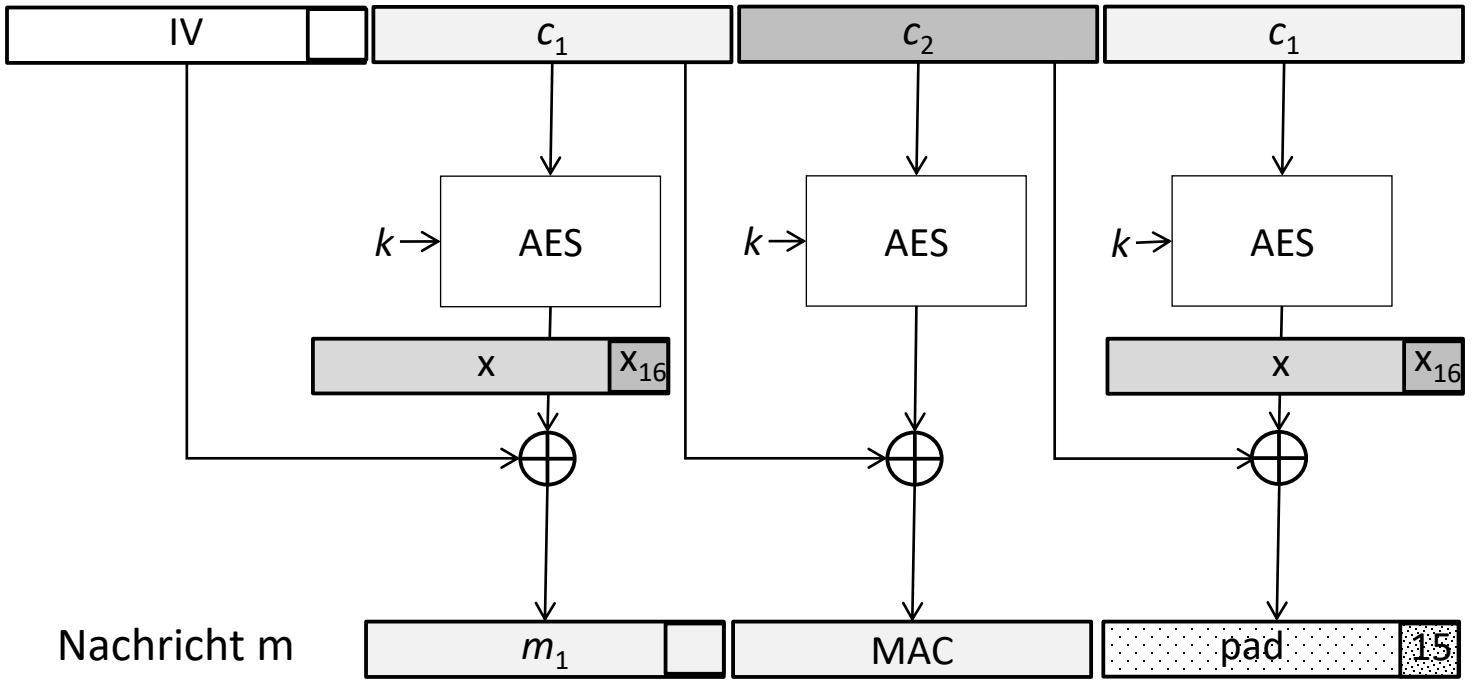
7. Mit Wahrschein-lichkeit 1/256 enthält das letzte Byte den korrekten Wert 15 (bzw. 7)
8. In diesem Fall wird der korrekte MAC gefunden, es tritt kein Entschlüsselungs-fehler auf



POODLE-Angriff

POODLE

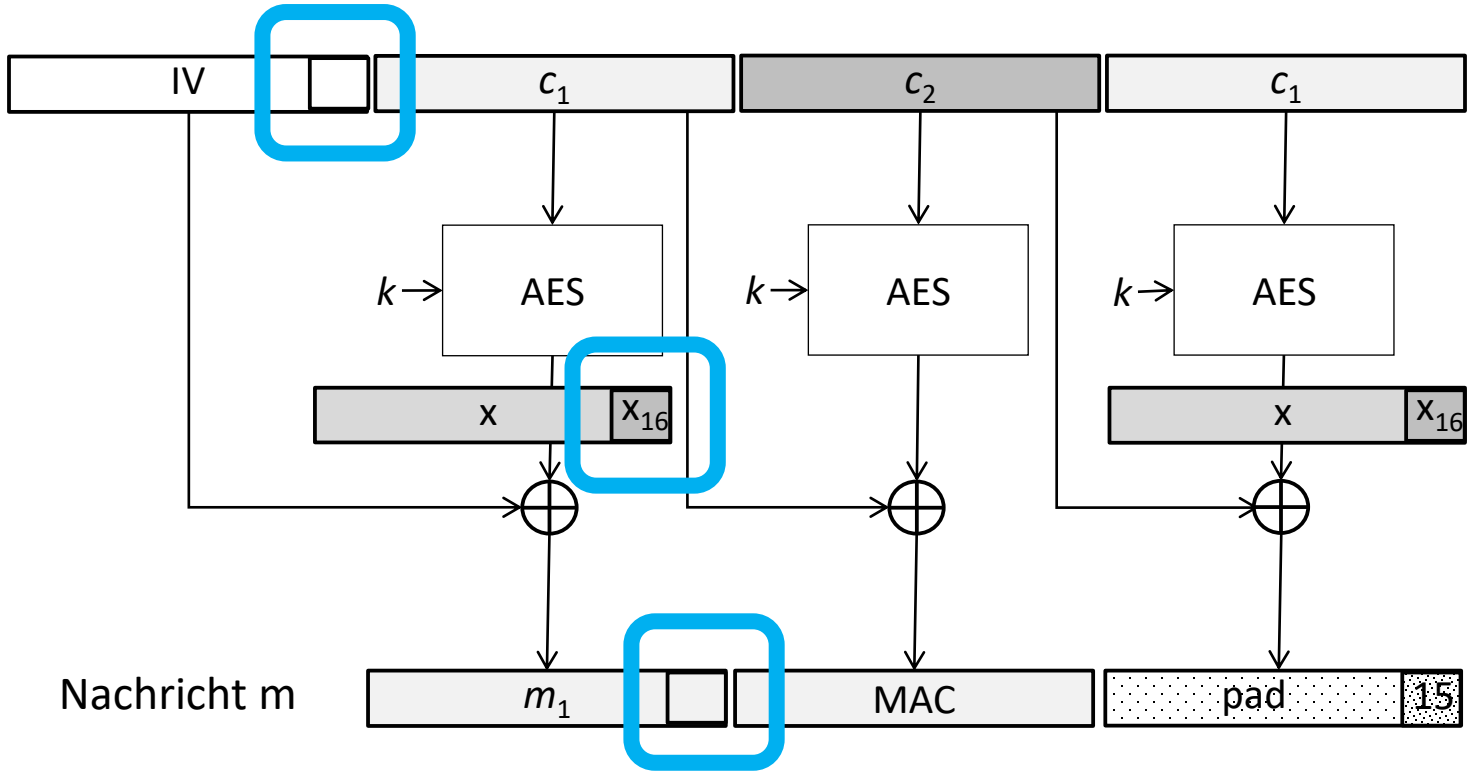
7. Mit Wahrschein-lichkeit 1/256 enthält das letzte Byte den korrekten Wert 15 (bzw. 7)
8. In diesem Fall wird der korrekte MAC gefunden, es tritt kein Entschlüsselungs-fehler auf
9. Der Klartext von m_1^{16} kann analog zum Vaudenauy-Angriff berechnet werden



POODLE-Angriff

POODLE

7. Mit Wahrschein-lichkeit 1/256 enthält das letzte Byte den korrekten Wert 15 (bzw. 7)
8. In diesem Fall wird der korrekte MAC gefunden, es tritt kein Entschlüsselungs-fehler auf
9. Der Klartext von $m_1{}^{16}$ kann analog zum Vaudenauy-Angriff berechnet werden

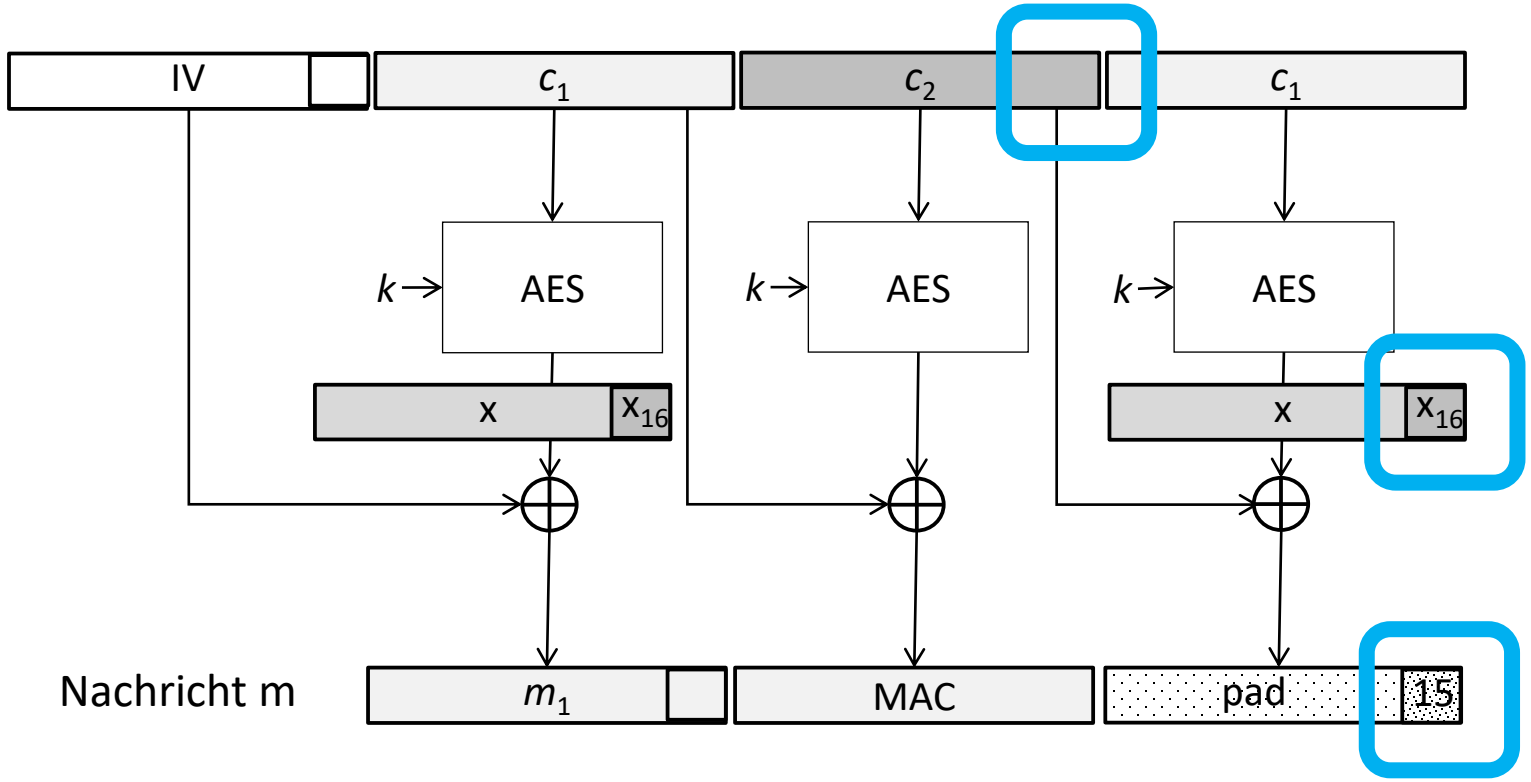


$$m_1{}^{16} = x_1{}^{16} \oplus IV_{16}$$

POODLE-Angriff

POODLE

7. Mit Wahrschein-lichkeit 1/256 enthält das letzte Byte den korrekten Wert 15 (bzw. 7)
8. In diesem Fall wird der korrekte MAC gefunden, es tritt kein Entschlüsselungs-fehler auf
9. Der Klartext von $m_1{}^{16}$ kann analog zum Vaudenauy-Angriff berechnet werden



$$m_1{}^{16} = x_1{}^{16} \oplus IV_{16} = (15 \oplus c_2{}^{16}) \oplus IV_{16}$$

POODLE: Bytewise Privileges

- Der beschriebene Angriff funktioniert nur für das letzte Byte von c_1

POODLE: Bytewise Privileges

- Der beschriebene Angriff funktioniert nur für das letzte Byte von c_1
- Zur Berechnung weiterer Klartextbytes kann der Angreifer die 'bytewise privileges'-Technik aus BEAST verwenden

POODLE: Downgrade Dance

Frage: Ist eine TLS-1.2-Verbindung mit POODLE angreifbar?

POODLE: Downgrade Dance

Frage: Ist eine TLS-1.2-Verbindung mit POODLE angreifbar?

- Undokumentiertes Browserverhalten: Downgrade-Dance

POODLE: Downgrade Dance

Frage: Ist eine TLS-1.2-Verbindung mit POODLE angreifbar?

- Undokumentiertes Browserverhalten: Downgrade-Dance
  1. Browser sendet ClientHello mit TLS 1.2

POODLE: Downgrade Dance

Frage: Ist eine TLS-1.2-Verbindung mit POODLE angreifbar?

- Undokumentiertes Browserverhalten: Downgrade-Dance
  1. Browser sendet ClientHello mit TLS 1.2
  2. MitM-Angreifer signalisiert Netzwerkfehler nach ClientHello

POODLE: Downgrade Dance

Frage: Ist eine TLS-1.2-Verbindung mit POODLE angreifbar?

- Undokumentiertes Browserverhalten: Downgrade-Dance
  1. Browser sendet ClientHello mit TLS 1.2
  2. MitM-Angreifer signalisiert Netzwerkfehler nach ClientHello
  3. Browser sendet ClientHello mit TLS 1.1

POODLE: Downgrade Dance

Frage: Ist eine TLS-1.2-Verbindung mit POODLE angreifbar?

- Undokumentiertes Browserverhalten: Downgrade-Dance
  1. Browser sendet ClientHello mit TLS 1.2
  2. MitM-Angreifer signalisiert Netzwerkfehler nach ClientHello
  3. Browser sendet ClientHello mit TLS 1.1
  4. MitM-Angreifer signalisiert Netzwerkfehler nach ClientHello

POODLE: Downgrade Dance

Frage: Ist eine TLS-1.2-Verbindung mit POODLE angreifbar?

- Undokumentiertes Browserverhalten: Downgrade-Dance
  1. Browser sendet ClientHello mit TLS 1.2
  2. MitM-Angreifer signalisiert Netzwerkfehler nach ClientHello
  3. Browser sendet ClientHello mit TLS 1.1
  4. MitM-Angreifer signalisiert Netzwerkfehler nach ClientHello
  5. Browser sendet ClientHello mit TLS 1.0

POODLE: Downgrade Dance

Frage: Ist eine TLS-1.2-Verbindung mit POODLE angreifbar?

- Undokumentiertes Browserverhalten: Downgrade-Dance
  1. Browser sendet ClientHello mit TLS 1.2
  2. MitM-Angreifer signalisiert Netzwerkfehler nach ClientHello
  3. Browser sendet ClientHello mit TLS 1.1
  4. MitM-Angreifer signalisiert Netzwerkfehler nach ClientHello
  5. Browser sendet ClientHello mit TLS 1.0
  6. MitM-Angreifer signalisiert Netzwerkfehler nach ClientHello

POODLE: Downgrade Dance

Frage: Ist eine TLS-1.2-Verbindung mit POODLE angreifbar?

- Undokumentiertes Browserverhalten: Downgrade-Dance
  1. Browser sendet ClientHello mit TLS 1.2
  2. MitM-Angreifer signalisiert Netzwerkfehler nach ClientHello
  3. Browser sendet ClientHello mit TLS 1.1
  4. MitM-Angreifer signalisiert Netzwerkfehler nach ClientHello
  5. Browser sendet ClientHello mit TLS 1.0
  6. MitM-Angreifer signalisiert Netzwerkfehler nach ClientHello
  7. Browser sendet ClientHello mit SSL 3.0

POODLE: Downgrade Dance

Frage: Ist eine TLS-1.2-Verbindung mit POODLE angreifbar?

- Undokumentiertes Browserverhalten: Downgrade-Dance
  1. Browser sendet ClientHello mit TLS 1.2
  2. MitM-Angreifer signalisiert Netzwerkfehler nach ClientHello
  3. Browser sendet ClientHello mit TLS 1.1
  4. MitM-Angreifer signalisiert Netzwerkfehler nach ClientHello
  5. Browser sendet ClientHello mit TLS 1.0
  6. MitM-Angreifer signalisiert Netzwerkfehler nach ClientHello
  7. Browser sendet ClientHello mit SSL 3.0
  8. Angreifer lässt ClientHello durch, SSL 3.0 wird verwendet

POODLE: Fazit

- Record Layer von SSL 3.0 komplett gebrochen

POODLE: Fazit

- Record Layer von SSL 3.0 komplett gebrochen
- Downgrade Dance macht den Angriff gefährlich

POODLE: Fazit

- Record Layer von SSL 3.0 komplett gebrochen
- Downgrade Dance macht den Angriff gefährlich

- Einzige Lösung: Komplette Deaktivierung von SSL 3.0

3.3 Angriffe auf den Record Layer

3.3.3a Allgemeine Padding-Oracle-Angriffe

Allgemeine Padding-Oracle-Angriffe

[PDF] Scalable Scanning and Automatic Classification of TLS **Padding** Oracle Vulnerabilities. [PDF] usenix.org

R Merget, J **Somorovsky**, N Aviram, C Young… - USENIX Security …, 2019 - usenix.org

… One prominent class of such attacks is CBC **padding** oracle attacks. These attacks allow an … of CBC **padding**. We present the first large-scale scan for CBC **padding** oracle vulnerabilities …

☆ Speichern 99 Zitieren Zitiert von: 21 Ähnliche Artikel Alle 5 Versionen »

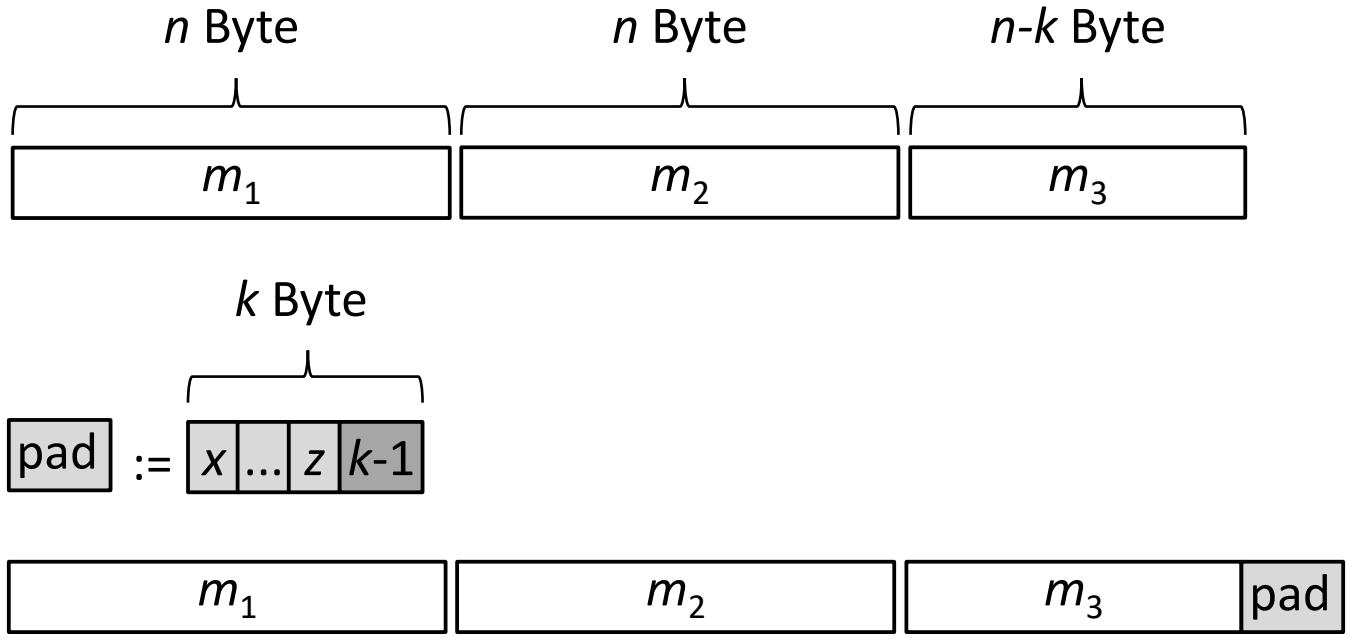
Systematic fuzzing and testing of TLS libraries [PDF] acm.org

J **Somorovsky** - Proceedings of the 2016 ACM SIGSAC conference on …, 2016 - dl.acm.org

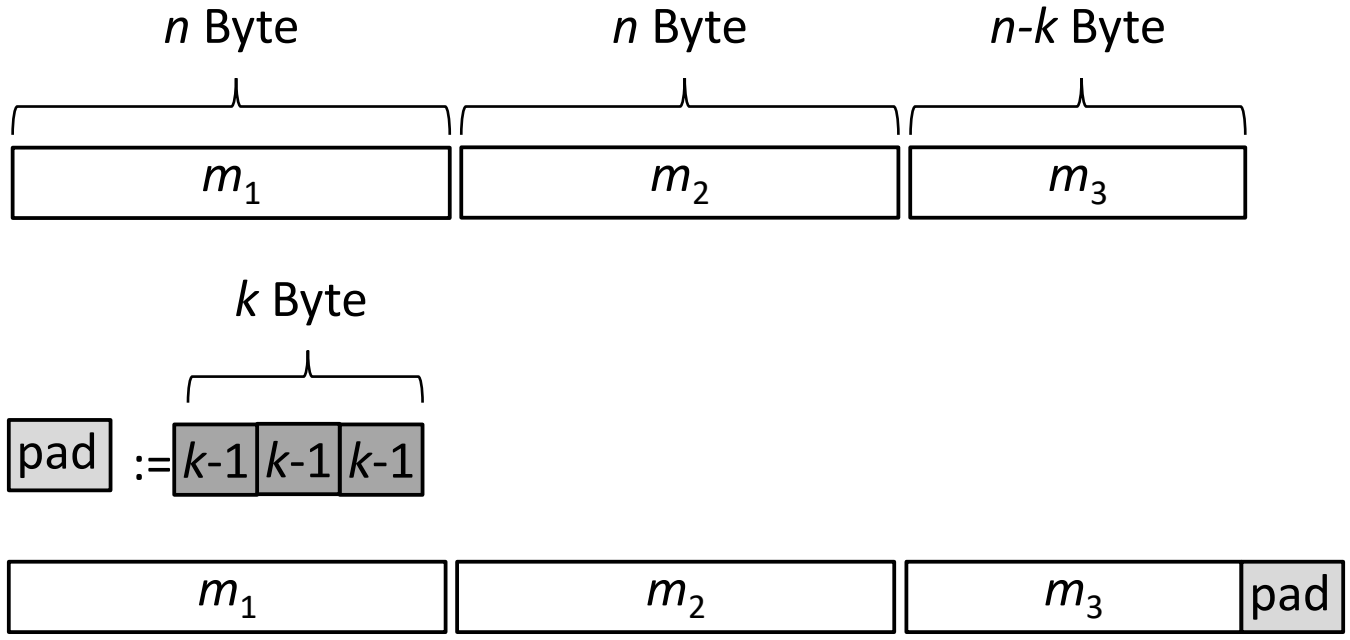
… In the first stage, we introduce cryptographic fuzzing for known vulnerabilities like **padding** oracle attacks [55] or Bleichenbacher attacks [23]. In the second stage, we then systematically …

☆ Speichern 99 Zitieren Zitiert von: 125 Ähnliche Artikel Alle 8 Versionen »

Padding in SSL 3.0

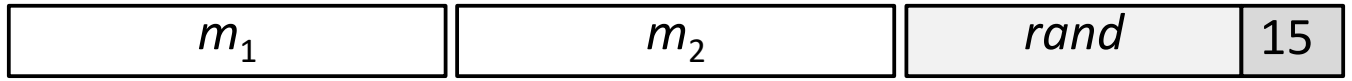


Padding in TLS



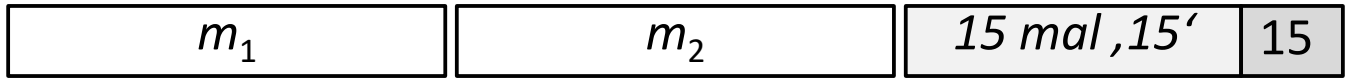
POODLE funktioniert nicht in TLS

SSL 3.0



passt mit W. 2^{-8}

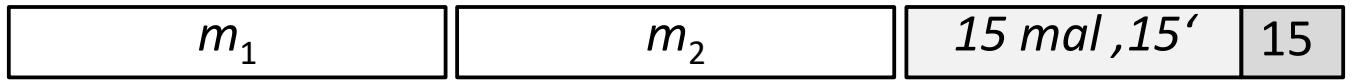
TLS 1.0, …



passt nur mit W. 2^{-128}

Allgemeine Padding-Oracle-Angriffe

TLS 1.0, …



… aber was wäre wenn die Implementierung nur das letzte Padding-Byte überprüft?

passt mit W. 2^{-8}

Allgemeine Padding-Oracle-Angriffe

TLS 1.0, …

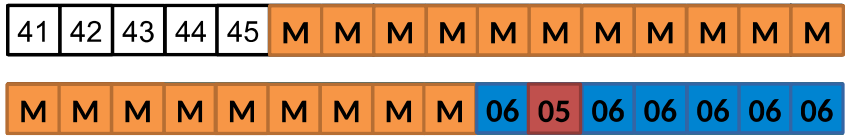
… oder wenn die Implementierung nur das erste Byte des MAC überprüft?

passt mit W. 2^{-8}

passt mit W. 2^{-8}

Allgemeine Padding-Oracle-Angriffe

BAD_RECORD_MAC



DECODING_ERROR

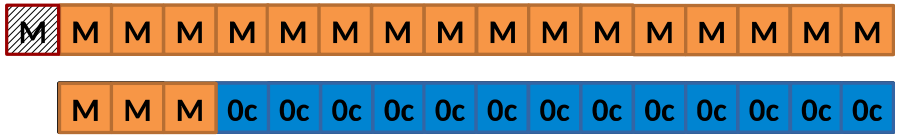


Figure 5: Direct padding oracle provided by Botan 1.11.21. In case of an invalid padding, Botan responds with `BAD_RECORD_MAC`. In case of a valid padding, Botan attempts to process the HMAC. It responds with a `DECODE_ERROR` if the number of remaining bytes is insufficient for HMAC validation.

```
if(record_len < mac_pad_iv_size)
  throw Decoding_Error("Record sent with invalid
      length");
```

Somorovsky, ACM CCS 2016

Allgemeine Padding-Oracle-Angriffe

Nr.	MAC			Padding		
	Len	Pos	Modification	Len	Pos	Modification
1	20	20	⊕ 0x01	56	–	–
2	20	11	⊕ 0x08	56	–	–
3	20	1	⊕ 0x80	56	–	–
4	19	1	DEL	56	–	–
5	19	20	DEL	56	–	–
6	0	–	–	80	ALL	0x4F
7	0	–	–	80	ALL	0xFF
8	20	–	–	60	1	⊕ 0x80
9	20	–	–	60	31	⊕ 0x08
10	20	–	–	60	60	⊕ 0x01
11	20	1	⊕ 0x80	60	–	–
12	20	9	⊕ 0x08	60	–	–
13	20	16	⊕ 0x01	60	–	–
14	20	1	⊕ 0x01	60	1	⊕ 0x80
15	20	1	⊕ 0x01	60	31	⊕ 0x08
16	20	1	⊕ 0x01	60	60	⊕ 0x01
17	20	–	–	6	1	⊕ 0x80
18	20	–	–	6	3	⊕ 0x08
19	20	–	–	6	6	⊕ 0x01
20	20	1	⊕ 0x80	6	–	–
21	20	9	⊕ 0x08	6	–	–
22	20	16	⊕ 0x01	6	–	–
23	20	1	⊕ 0x01	6	1	⊕ 0x80
24	20	1	⊕ 0x01	6	3	⊕ 0x08
25	20	1	⊕ 0x01	6	6	⊕ 0x01

Table 1: A summary of our malformed records, as constructed for `TLS_RSA_WITH_AES_128_CBC_SHA`. The columns indicate length, position, and modification for MAC and padding bytes, respectively. ⊕ denotes XOR'ing the listed value in the listed position. DEL denotes deleting one byte in the listed position.

Amazon MAC errors:

- Invalid Padding: Close TCP
- Valid Padding: Abort TLS, keep TCP

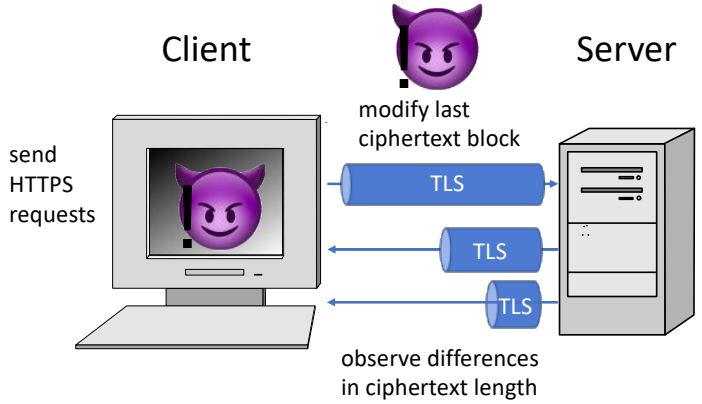


Figure 5: Exploiting observable error-based padding oracles in a BEAST scenario. Differences in total ciphertext length result from different numbers of TLS alerts being sent.

Merget et al., USENIX 2019