Authenticated Encryption (2)

Krypto II

VL 19

06.12.2018

Buch des Tages

Titel: Network Security (2nd Edition)

Autor(en): Charlie Kaufman, Radia Perlman, Mike Speciner

Verlag: Prentice Hall, 2002

Umfang: ca. 700 Seiten

Hinweise zum Inhalt:

Trotz des allgemeinen Titels liegt der Schwerpunkt auf Kryptographie. Die Beschreibung der mathematischen Grundlagen sowie der Algorithmen ist sehr gelungen, gut verständlich und mit Beispielen angereichert.

Leider ist das Buch schon etwas veraltet und es wurde nach 2002 keine Aktualisierung mehr vorgenommen. Aus diesem Grund finden sich mehrere Algorithmen, die detailliert erläutert werden, doch mittlerweile nicht mehr verwendet werden (sollen). Der größte Teil der Inhalte ist jedoch unverändert gültig. Wen es nicht stört, dass nicht immer die aktuellsten Praxisbeispiele erörtert werden, für den ist das Buch eine echte Empfehlung.

Simpler Angriff ohne Schlüssel

- Es gibt zahlreiche Vorschläge für die Erzeugung kryptographischer Prüfsummen. Betrachten wir die folgende Idee:
- Die Nachricht M bestehe aus den Blöcken $M = M_1 \| M_2 \| ... \| M_n$.
- Nun berechnen wir $f(M) := M_1 \oplus M_2 \oplus ... \oplus M_n$.
- Auf f(M) wenden wir eine Blockchiffre $E_K()$ an und erhalten unsere kryptographische Prüfsumme $MAC_K(M) := E_K(f(M))$.
- Angenommen, wir haben eine beliebige (aus einem oder mehreren Blöcken bestehende) Nachricht M'.
- Definieren wir einen Block $Y := f(M') \oplus f(M)$. Es gilt $f(Y) = Y$.
- Damit erhalten wir:

$$MAC_K(M' \| Y) = E_K(f(M' \| Y)) = E_K(f(M') \oplus f(Y)) = E_K(f(M') \oplus Y) =$$

$$E_K(f(M') \oplus f(M') \oplus f(M)) = E_K(f(M)) = MAC_K(M).$$

Die Nachricht $M' \| Y$ hat also dieselbe Prüfsumme wie M und wir können Nachrichten beliebig fälschen. Lediglich der letzte Nachrichtenblock ist hierbei festgelegt auf den Wert Y.

Keyed Hashing

- Ein naheliegender Ansatz zum Design eines Message Authentication Codes ist die Verwendung einer Hashfunktion, die als Input zusätzlich zur Nachricht einen geheimen Key erhält.
- Zu diesem Zweck fügen wir einfach einen geheimen Schlüssel K zur Nachricht hinzu, bevor wir den Hashwert bilden. Zur Positionierung gibt es verschiedene Varianten:
  - $MAC_K(M) = hash(M \| K)$
  - $MAC_K(M) = hash(K \| M)$
  - $MAC_K(M) = hash(K \| M \| K)$
  - weitere Kombinationen...(?)
- Der Empfänger erhält die Nachricht M, zusammen mit dem Hashwert $MAC_K(M)$. Da er im Besitz des geheimen Schlüssels K ist, kann er verifizieren, ob der MAC zur Nachricht M gehört.
- Grundsätzlich ist die Kombination aus Nachricht und Schlüssel, die als Input einer Hashfunktion dienen, eine gute Idee. Man muss bei der Konstruktion eines solchen MAC jedoch sorgfältig vorgehen. Mögliche Lösungen stellen wir nun vor.

Collision Attack auf hash(M ‖ K)

- Betrachten wir zunächst den mittels Anfügen eines geheimen Schlüssels erzeugten Message Authentication Code.

  $MAC_K(M) = hash(M \| K)$

- Angenommen, ein Angreifer ist in der Lage, ein Second Preimage M’ zu finden, das denselben Hashwert wie M besitzt. Verwenden wir eine Hashfunktion wie z.B. MD5 oder SHA-2 nach dem Merkle-Damgard Prinzip (siehe VL Krypto 1), so gilt:

  $hash(M') = hash(M) \Rightarrow hash(M' \| K) = hash(M \| K)$

- Dies bedeutet, dass ein Angreifer ohne Kenntnis des geheimen Schlüssels K eine Nachricht mit gültigem MAC konstruieren kann, sofern er ein Second Preimage findet.

Length Extension auf hash(K ‖ M)

- Nachdem die Erzeugung eines Message Authentication Codes durch Bildung von $MAC_K(M) = hash(M \| K)$ unerwünschte Eigenschaften aufweist, betrachten wir stattdessen die Variante, in der zuerst der Schlüssel K und danach die Nachricht M aneinander gefügt werden:

  $MAC_K(M) = hash(K \| M)$

- Betrachten wir nun eine Nachricht der Form M ‖ M'. Dann gilt:

  $MAC_K(M \| M') = hash(K \| M \| M')$

- Wird die Hashfunktion so berechnet (sog. Merkle Damgard Konstruktion), dass der Hash für den i-ten Datenblock ermittelt wird auf Basis des Hashwertes der vorherigen i-1 Datenblöcke, so kann ein Angreifer mit Kenntnis von $MAC_K(M) = hash(K \| M)$, ohne Kenntnis des Schlüssels K, einfach die Hashfunktion weiterführen auf die folgenden Datenblöcke M' und erhält dadurch einen gültigen Hashwert $hash(K \| M \| M') = MAC_K(M \| M')$.

HMAC (1)

- ***HMAC*** ist ein weit verbreitetes und standardisiertes Verfahren (siehe u.a. FIPS PUB 198, FIPS PUB 198-1, RFC 2104). Die Abkürzung steht für ***Keyed-Hash Message Authentication Code*** bzw. ***Hash-based Message Authentication Code***.
- Das Verfahren erlaubt es, eine beliebige (sichere) Hashfunktion als Komponente einzusetzen. So gibt es beispielsweise HMAC-SHA256, HMAC-SHA3, etc.
- Designmerkmal bei HMAC ist die zweifache ("innere" und "äußere") Anwendung der Hashfunktion.
- HMAC bietet keine Verschlüsselung, sondern ausschließlich Message Authentication.
- Zahlreiche Protokolle verwenden HMAC, so beispielsweise IPSec und TLS.

RFC 2104

Network Working Group
Request for Comments: 2104
Category: Informational

H. Krawczyk
IBM
M. Bellare
UCSD
R. Canetti
IBM
February 1997

HMAC: Keyed-Hashing for Message Authentication

Status of This Memo

Abstract

This document describes HMAC, a mechanism for message authentication using cryptographic hash functions. HMAC can be used with any iterative cryptographic hash function, e.g., MD5, SHA-1, in combination with a secret shared key. The cryptographic strength of HMAC depends on the properties of the underlying hash function.

1. Introduction

Providing a way to check the integrity of information transmitted over or stored in an unreliable medium is a prime necessity in the world of open computing and communications. Mechanisms that provide such integrity check based on a secret key are usually called "message authentication codes" (MAC). Typically, message authentication codes are used between two parties that share a secret key in order to validate information transmitted between these parties. In this document we present such a MAC mechanism based on cryptographic hash functions. This mechanism, called HMAC, is based on work by the authors [BCK1] where the construction is presented and cryptographically analyzed. We refer to that work for the details on the rationale and security analysis of HMAC, and its comparison to other keyed-hash methods.

FIPS PUB 198-1

FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION

The Keyed-Hash Message Authentication Code (HMAC)

CATEGORY: COMPUTER SECURITY **SUBCATEGORY: CRYPTOGRAPHY**

Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8900

July 2008

U.S. Department of Commerce
Carlos M. Gutierrez, Secretary

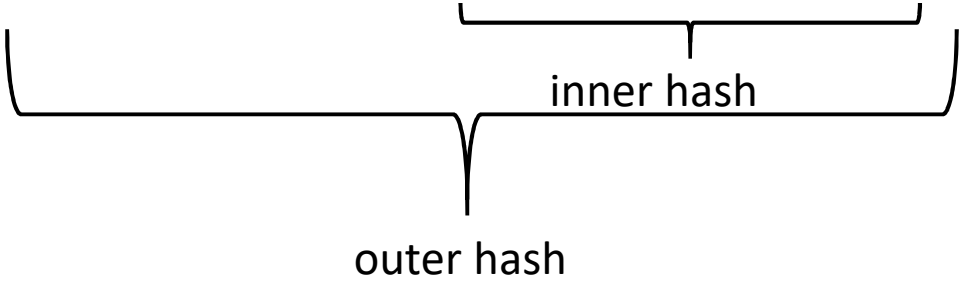
National Institute of Standards and Technology
James M. Turner, Deputy Director

FIPS PUB 198-1

HMAC (2)

HMAC funktioniert wie folgt:

$\text{HMAC}_K(M) = \text{hash}((K' \oplus \text{opad}) \| \text{hash}(K' \oplus \text{ipad} \| M))$



Hierbei bezeichne

- hash(): kryptographische Hashfunktion, beispielsweise SHA-2
- K: der geheime Schlüssel, und K' einen hiervon abgeleiteten Schlüssel
- ipad: eine im inneren Hash verwendete Konstante zwecks Padding:
  ipad = 00110110
- opad: eine im äußeren Hash verwendete Konstante zwecks Padding:
  opad = 01011100

HMAC (3)

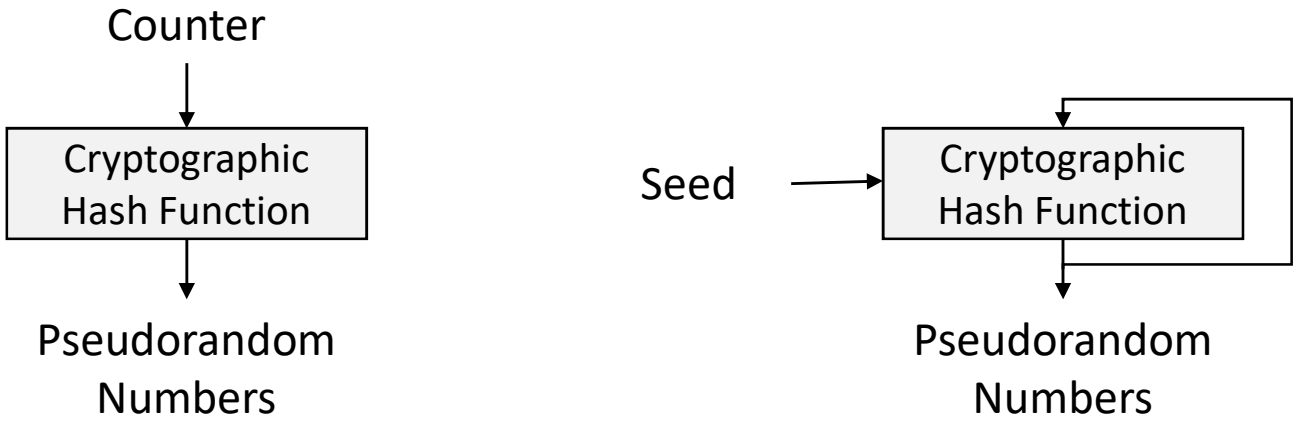
- Sei n die Blocklänge des Hashwerts (Output der verwendeten Hashfunktion, beispielsweise 256 Bit bei SHA-256).
- Sei b die Blocklänge der Inputblöcke der Hashfunktion (beispielsweise 512 Bit bei SHA-2).
- K' wird aus K abgeleitet in Abhängigkeit von der Länge k des Schlüssels K und der Länge n der Inputblocks des Hashverfahrens:
  - ist der Schlüssel K länger als ein Inputblock, so setzen wir K' = hash(K)
  - ist der Schlüssel K kürzer als ein Inputblock (bzw. $k \leq n$), so wird K rechtsbündig in einen Block der Länge n geschrieben und die ggf. noch fehlenden n-k Bits links von K werden mit Nullen aufgefüllt.
  - Anschließend haben wir einen Schlüssel K', dessen Länge genau einem Inputblock entspricht.

Online Brute Force Attacke auf HMAC

- Angenommen wir haben eine Hashfunktion für HMAC verwendet, deren Output eine Länge von 128 Bit hat.
- Unter Berücksichtigung des Birthday Paradoxons wissen wir, dass wir "nur" ca. 2^{64} HMAC Outputs sammeln müssen, um voraussichtlich eine Kollission zu finden.
- Da wir jedoch nicht über den Key K verfügen, können wir nicht einfach Offline-Berechnungen ausführen. Stattdessen muss ein Angreifer beispielsweise den Datenverkehr einer mit HMAC abgesicherten Verbindung abhören und Datenpakete auswerten.
- Legt man (Beispiel siehe Stallings, Cryptography and Network Security) eine Datenleitung mit 1 Gbps zugrunde, so müsste man den Datenstrom rund 150,000 Jahre beobachten, um genügend Material für einen Angriff zu haben.
- Ein Brute Force Angriff auf HMAC ist insofern schwieriger als ein Brute Force Angriff auf die zugrunde liegende Hashfunktion allein.

HMAC als PRNG (1)

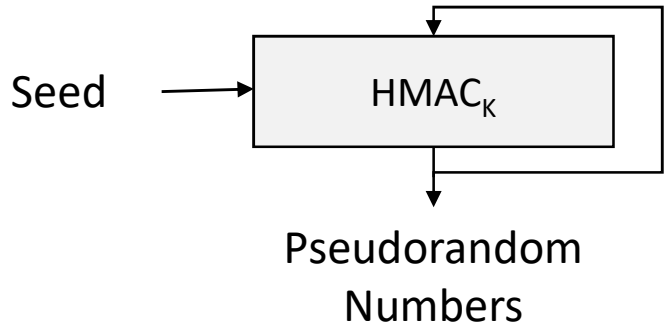
- Der Output einer kryptographischen Hashfunktion sollte nicht unterscheidbar sein vom Output eines kryptographischen Pseudozufallsgenerators.
- Insofern ist es naheliegend, einen solchen PRNG auf Grundlage einer Hashfunktion zu konstruieren.
- In der Tat gibt es einfache Designs, die wie folgt aufgebaut sind:



- Diese können funktionieren, doch erhöhte Sicherheit verspricht man sich durch Verwendung eines kryptographischen Message Authentication Codes wie z.B. HMAC, anstelle einer einfachen Hashfunktion.

HMAC als PRNG (2)

- Die Verwendung von HMAC als Deterministic Random Bit Generator (DRBG) ist sogar standardisiert und beschrieben in NIST SP800-90A.
- Die Funktionsweise ist analog zum zuvor beschriebenen Beispiel mit Hashfunktionen, nur dass diesmal HMAC verwendet wird.
- Der Output des HMAC dient in der nächsten Iteration wieder als Input.



- Der erste Input erfolgt in Form eines Seed Values bzw. Initialisierungsvektors. Dieser dient, gemeinsam mit dem geheim zu haltenden Key K, zur Initialisierung des DRBG.
- Kennt man einen Outputwert, so kann man – im Gegensatz zur Verwendung einer einfachen (schlüssellosen) Hashfunktion, nicht automatisch den Folge-Output berechnen. Hierzu fehlt der Schlüssel K.

NIST SP800-90A

NIST SP800-90A

10.1.2 HMAC_DRBG

HMAC_DRBG uses multiple occurrences of an **approved** keyed hash function, which is based on an **approved** hash function. This DRBG mechanism uses the **HMAC_DRBG_Update** function specified in Section 10.1.2.2 and the **HMAC** function within the **HMAC_DRBG_Update** function as the derivation function during instantiation and reseeding. The same hash function **shall** be used throughout an **HMAC_DRBG** instantiation. The hash function used **shall** meet or exceed the security strength required by the consuming application for DRBG output (see [SP 800-57]).

Figure 9 depicts the **HMAC_DRBG** in three stages. **HMAC_DRBG** is specified using an internal function (**HMAC_DRBG_Update**). This function is called during the **HMAC_DRBG** instantiate, generate and reseed algorithms to adjust the internal state when new entropy or additional input is provided, as well as to update the internal state after pseudorandom bits are generated. The operations in the top portion of the figure are only performed if the additional input is not null. Figure 10 depicts the **HMAC_DRBG_Update** function.

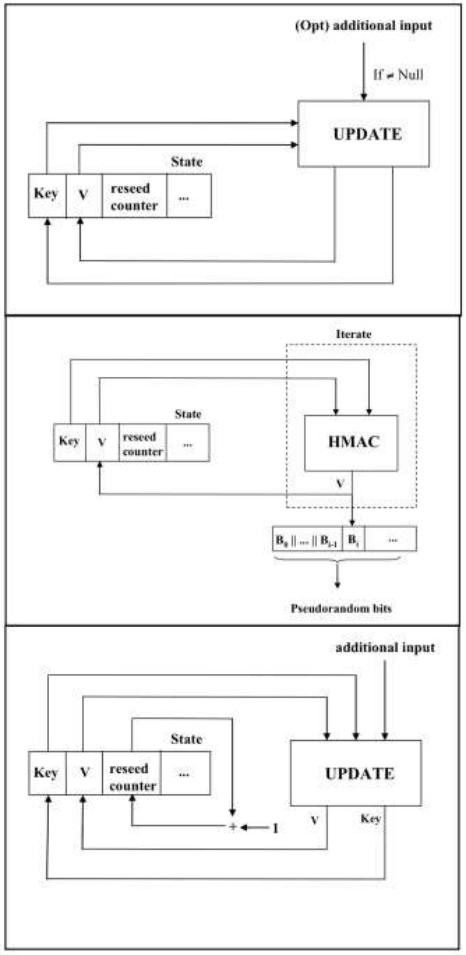


Figure 9: HMAC_DRBG Generate Function

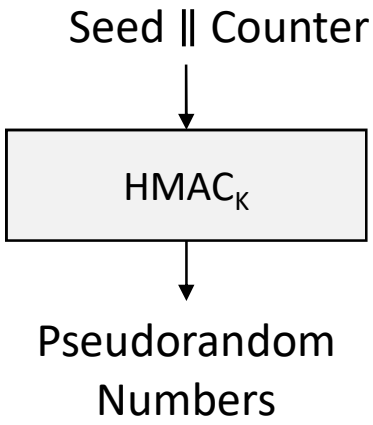
10.1.2.1 HMAC_DRBG Internal State

The internal state for **HMAC_DRBG** consists of:

1. The *working_state*:
   a. The value *V* of *outlen* bits, which is updated each time another *outlen* bits of output are produced (where *outlen* is specified in Table 2 of Section 10.1).
   b. The *outlen*-bit *Key*, which is updated at least once each time that the DRBG mechanism generates pseudorandom bits.
   c. A counter (*reseed_counter*) that indicates the number of requests for pseudorandom bits since instantiation or reseeding.

HMAC in IEEE 802.11i

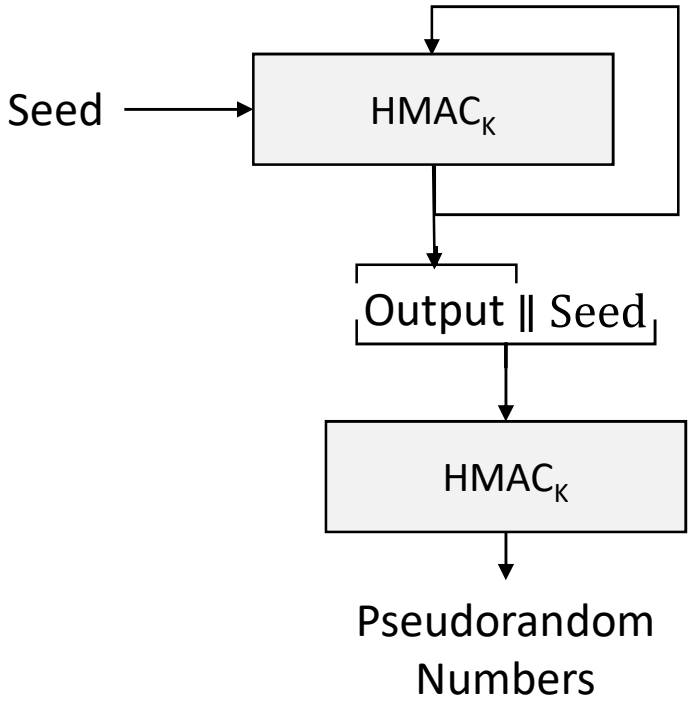
- In der Praxis finden sich nicht nur Implementierungen gemäß NIST SP800-90A, sondern auch andere, leicht abgewandelte Versionen für einen HMAC DRBG.
- Ein Beispiel ist der Wireless LAN Standard IEEE 802.11i.



- Hier wird nicht der Output des HMAC verwendet als Input für die nächste Iteration. Stattdessen dient der Seed Value als Input, in Verbindung mit einem Counter, der nach jeder Iteration inkrementiert wird.

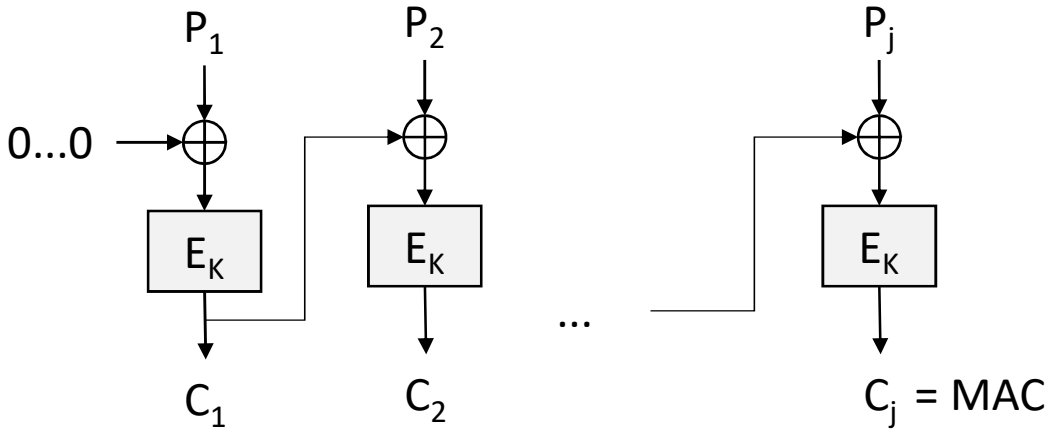
HMAC in TLS / WTLS

- Eine weitere Variante verwendet das Transport Layer Security Protocol TLS bzw. Wirelesss Transport Layer Security Protocol WTLS.
- Dort wird der HMAC zweimal hintereinander ausgeführt pro Iteration.



Data Authentication Algorithm DAA

- Ein anderer Message Authentication Code ist der sogenannte ***DAA Data Authentication Algorithm***. Dieser hat das gleiche Konstruktionsprinzip wie der CBC Betriebsmodus für Blockchiffren. Als Initialisierungsvektor wird hier der Nullvektor gewählt, was praktisch bedeutet, dass kein IV verwendet wird.
- DAA hatte eine hohe Verbreitung und wurde standardisiert in FIPS PUB 113 und in ANSI X9.17 . Die Tatsache, dass DAA schon recht alt ist, lässt sich daran erkennen, dass als Verschlüsselungsalgorithmus E_K DES spezifiziert wurde.



- Der MAC besteht aus dem letzten Output Block (oder einem Teilblock hiervon).

PUBLICATIONS

FIPS 113

Computer Data Authentication

Date Published: May 1985

Withdrawn: September 01, 2008

Abstract

This standard specifies a Data Authentication Algorithm (DAA) which may be used to detect unauthorized modifications, both intentional and accidental, to data, The standard is based on the algorithm specified in the Data Encryption Standard (DES) Federal Information Processing Standards Publication (FIPS PUB) 46, and is compatible with both the Department of the Treasury's Electronic Funds and Security Transfer Policy and the American National Standards Institute (ANSI) Standard for Financial Institution Message Authentication (see cross index). The Message Authentication Code (MAC) as specified in ANSI X9.9 is computed in the same manner as the Data Authentication Code (DAC) specified in this standard. Similarly, the Data Identifier (DID) specified in this standard is sometimes referred to as a Message Identifier (MID) in standards related to message communications.

Control Families

None selected

DOCUMENTATION

Publication:
None available

Supplemental Material:
None available

Length Extension Attack (2)

- Betrachten wir nochmals den Data Authentication Algorithm. Angenommen wir haben eine Nachricht P_1, die nur einen Block lang ist. Diese möchten wir mit einem MAC versehen:

  $MAC(P_1) = C_1 = E_K(P_1)$.

- Nun setzen wir $P_2 := C_1 \oplus P_1$ und betrachten die neue Nachricht, bestehend aus $P_1 \| P_2$: deren MAC errechnet sich wie folgt:

  $MAC(P_1 \| P_2) = C_2 = E_K(C_1 \oplus P_2) = E_K(C_1 \oplus C_1 \oplus P_1) = E_K(P_1) = MAC(P_1)$.

- Die um einen Block verlängerte Nachricht hat also denselben MAC!

- Wir könnten diese Nachricht nun Block für Block weiter verlängern: angenommen wir haben die Nachricht $P_1 \| P_2 \ldots \| P_j$ und einen zugehörigen gültigen MAC in Form von C_j. Zusätzlich benötigen wir den Outputblock C_{j-1} bzw. den MAC zu $P_1 \| P_2 \ldots \| P_{j-1}$.

- Dann setzen wir $P_{j+1} = C_j \oplus C_{j-1} \oplus P_j$ und berechnen den neuen MAC für die Nachricht $P_1 \| P_2 \ldots \| P_{j+1}$:

  $MAC(P_1 \| P_2 \ldots \| P_{j+1}) = C_{j+1} = E_K(C_j \oplus P_{j+1}) = E_K(C_j \oplus C_j \oplus C_{j-1} \oplus P_j) = E_K(C_{j-1} \oplus P_j) = C_j = MAC(P_1 \| P_2 \ldots \| P_j)$.

- Erneut hat die um einen Block verlängerte Nachricht denselben MAC!