Counter with CBC-MAC (CCM)

Blockwoche Informationssicherheit Wintersemester 2024/25

Olaf Bergmann

obgm@uni-bremen.de

Werbung: tinydtls

- DTLS für „kleine Geräte“
- AEAD mit PSK oder ECC
- wenig Speicherbedarf, Codegröße

Zu finden unter https://www.eclipse.org/tinydtls

Authenticated Encryption with Associated Data (AEAD)

Authenticated Encryption

- Vertraulichkeit + Integrität + Authentizität
- effizient mit Block-Verschlüsselung realisierbar
  $\rightarrow$ benötigt Counter und CBC-MAC

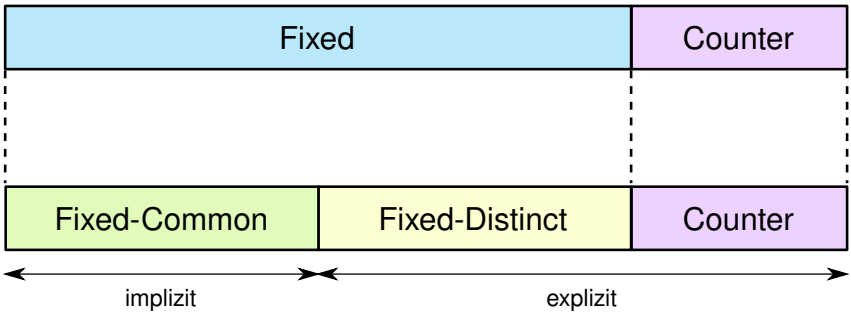
Mechanismus: RFC 5116

- Verschlüsselung: $K \times N \times A \times P \rightarrow A \times C$
- Entschlüsselung: $K \times N \times A \times C \rightarrow A \times P$
- AEAD-Algorithmus legt Länge von K (= `K_LEN`), fest (16 B für AES-128)

K: Key
N: Nonce
A: Associated Data
P: Plaintext
C: Ciphertext

Nonce

- (K,N) muss für jeden Aufruf von `encrypt()` unterschiedlich sein
- häufig fester Anteil + Zähler (siehe Abbildung)



Konstanten

Name	Bedeutung	typischer Wert (RFC 3610)
K_LEN	Schlüssellänge in Bytes (fest)	16
P_MAX	max. Plaintextlänge in Bytes	$2^{24} - 1$
A_MAX	max. Länge der Associated Data in Bytes	$2^{64} - 1$
N_MIN	min. Länge der Nonce in Bytes	7
N_MAX	max. Länge der Nonce in Bytes	13
C_MAX	max. Länge des Ciphertext in Bytes	$P\_MAX + 16$

RFC 3610 definiert das Verfahren „Counter with CBC-MAC (CCM)“
$\rightarrow$ konkrete Anwendung des Mechanismus aus RFC 5116

Counter with CBC-MAC (CCM, RFC 3610)

Authenticated Encryption mit einer Blockchiffre
$K \times N \times A \times P \rightarrow A \times C$

Konstanten

Name	Bedeutung	Wertebereich	typischer Wert
M	Größe des Authentisierungsfelds	$\{4, 6, 8, 10, 12, 14, 16\}$	16
L	Größe des Längenfelds	$2 \leq L \leq 8$	2

Eingabeparameter

Name	Bedeutung	Größe	Beispiel
K	Schlüssel für Blockchiffre		16
N	Nonce	$15 - L$ Bytes	13
m	Plaintext	$l(m)$ Bytes	
a	Additional data	$l(a)$ Bytes	0

Grundfunktionalität

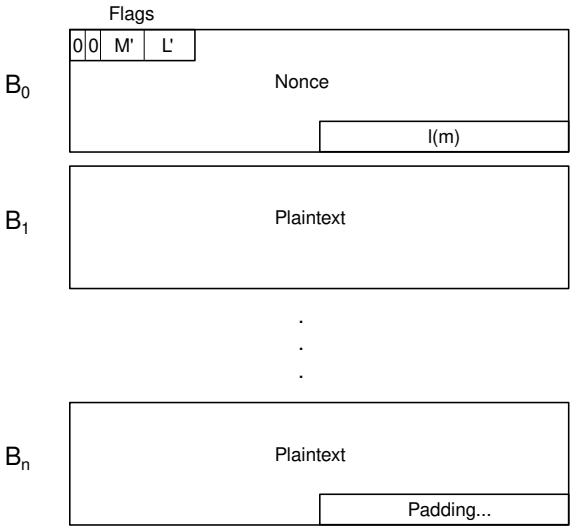
Basisoperationen

- Verschlüsseln mit Blockchiffre (z. B. AES-128)
  $\rightarrow$ 16-Byte-Blöcke als Grundbaustein
- Exklusiv-Oder (für jedes Bit in einem Block)

RFC 3610: Blockströme

- Vier Arten von Blöcken: A, B, S, X
- A_0, B_0 aus Eingabeparametern erzeugen
  - B_1, … B_n: enthalten Plaintext (mit n = $\lceil l(m)/16 \rceil$)
- S_i aus A_i erzeugt, bildet One-Time Pad zum Verschlüsseln/Entschlüsseln
- X_{i+1} aus X_i und B_i erzeugt, bildet Grundlage für Authentisierungswert

$B_0, \ldots B_n$



$$M' = (M-2)/2$$
$$L' = L-1$$

Achtung: MSB first („big endian“)

Message Authentication Code (MAC)

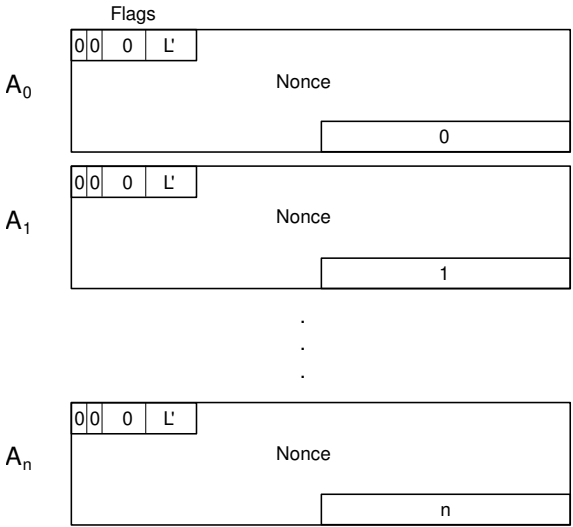
T = `first-M-bytes`(X_{n+1}) berechnen mit
$X_1 := E(K, B_0)$
$X_{i+1} := E(K, X_i \otimes B_i)$, i = 1, …, n

E: `EVP_EncryptUpdate()`
K: 16-Byte-Schlüssel (bei Initialisierung gesetzt)
$\otimes$: Bit-weises Exklusiv-Oder
T: MAC-Tag (= Eingabe für Berechnung des Authentisierungswerts)

Verschlüsseln im CTR-Mode

1. A_i berechnen $\rightarrow$
2. Schlüsselstrom: $S_i := E(K, A_i)$, $i = 0, 1, 2, \ldots$
3. blockweise verschlüsseln: $D := (P \otimes S_1 \cdot S_2 \cdot \ldots).substr(0, l(m))$
4. Authentisierungswert berechnen: $U := T \otimes \texttt{first-M-bytes}(S_0)$
5. und anhängen: $C := D \cdot U$

$A_0 \ldots A_n$



$L' = L - 1$

Counter i

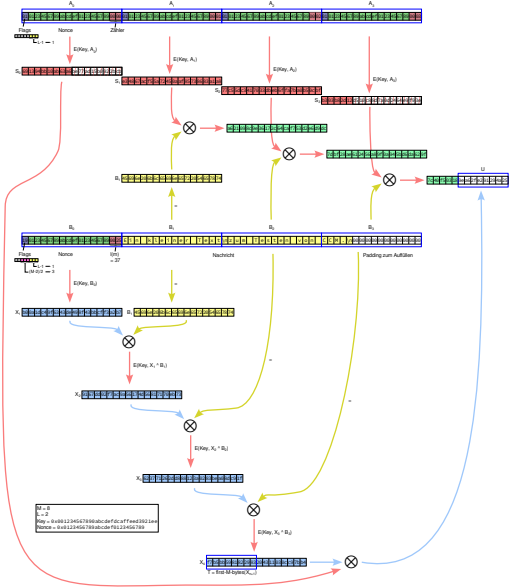
Achtung: MSB first („big endian“)

Entschlüsseln und Integritätscheck

$K \times N \times A \times C \rightarrow A \times P$

1. Schlüsselstrom S_i berechnen
2. T aus U und S_0 ermitteln ($T = U \otimes \texttt{first-M-bytes}(S_0)$)
3. Plaintext P aus D und $S_1 \cdot S_2 \cdot S_n$ … ermitteln
4. MAC berechnen
5. `MAC == T`? (alle `M` Bytes vergleichen — warum?)

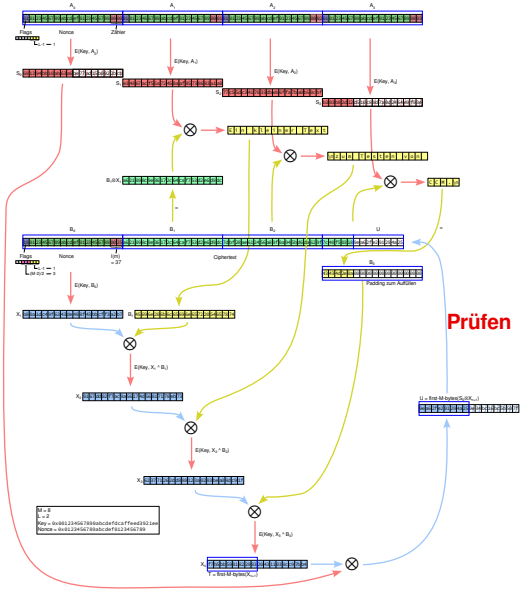
CCM Cheat Sheet: Verschlüsseln



Verschlüsseln

Integritäts-Anhängsel herstellen

CCM Cheat Sheet: Entschlüsseln



Entschlüsseln

Integrität prüfen

OpenSSL

1. Initialisierung: `EVP_CIPHER_CTX` vorbereiten
   `EVP_CIPHER_CTX_new()`
   `EVP_CIPHER_CTX_init()`, `EVP_CIPHER_CTX_set_padding(ctx, 0)`
2. Schlüssel setzen:
   `EVP_EncryptInit_ex(ctx, EVP_aes_128_ecb(), NULL, aes_key, NULL)`
3. B_0, …, B_n berechnen
   (dabei ggf. Daten blockweise einlesen)
4. Blöcke verschlüsseln mit `EVP_EncryptUpdate(…)`

Makefile

```
LDLIBS=-lcrypto
CFLAGS=-g -O2 -Wall -Wextra -Werror

# Flags for libressl on MacOS
__opensslpath=/opt/local/libexec/openssl3
LDFLAGS=-L$(__opensslpath)/lib
CPPFLAGS=-I$(__opensslpath)/include
```

C++-Gerüst: Platzhalter für `decrypt` und `encrypt`

- Vorgabe: `myccm.cc` mit Platzhaltern `MyCCM::decrypt` und `MyCCM::encrypt`

Vergleiche `ccm.hh`:

```
/**
 * Decrypts the given @p ciphertext using the parameters that have
 * been set in the constructor. The resulting plaintext is appended
 * to @p plaintext. This function returns @c true if the @p
 * ciphertext was successfully decrypted.
 */
virtual bool decrypt(const std::string &ciphertext, std::string &plaintext) = 0;

/**
 * Encrypts the given @p plaintext using the parameters that have
 * been set in the constructor. The resulting ciphertext is appended
 * to @p ciphertext, including a trailing authentication tag of
 * mac_tag_length bytes. This function returns @c true if the @p
 * plaintext was successfully ecrypted.
 */
virtual bool encrypt(const std::string &plaintext, std::string &ciphertext) = 0;
```

C++-Gerüst: Basisoperationen

Vergleiche `ccm.hh`:

```
/**
 * Encrypts a single block @p src using the encryption key that has
 * been set in the constructor. The encrypted block is written to @p
 * dst. As a general principle. the source and destination block
 * objects should be created according to the needs of the cipher
 * block chaining mode. For example, to create Block X_1 from B_0,
 * encrypt_block should be invoked with a BlockB as @p src and a
 * BlockX as @p dst. @p src and @p dst may be the same blocks.
 */
void encrypt_block(const Block &src, Block &dst);

/**
 * This block calculates the bit-wise exclusive or from the data of
 * the given blocks @p a and @p b. The result is stored in @p dst.
 * The destination block @p dst may be one of the source blocks @p a
 * or @p b.
 */
void xor_blocks(const Block &a, const Block &b, Block &dst) const;
```

C++-Gerüst: Beispiel: Erzeugen von X_{i+1} aus B_i und X_i

Gegeben: `BlockB b`, `BlockX x` und `BlockX ziel`.

```
xor_blocks(b, x, ziel);
```

- Überschreibt den Inhalt von Block `ziel` mit `b^x` (Exklusiv-Oder).
- Zielblock darf mit einem der Eingabeblöcke identisch sein
  → Typische Anwendung: `xor_blocks(b, x, x)`

Initialisieren von B_0

```
BlockB b{mac_tag_length, l_length, nonce, plaintext};
```

Das CCM-Objekt…

- bietet einen bereits initialisierten `EVP_CIPHER_CTX`
- kennt `mac_tag_length` (M), `l_length` (L) und `nonce`.

C++-Gerüst: Beispiel: Erzeugen von S_0 aus A_0

```
BlockA a{l_length, nonce}; // Initialisiert A_0 mit Zählerwert 0
BlockS s;                  // leerer Block S_0 für Ergebnis

a.debug();                 // Optional: Block ausgeben lassen
encrypt_block(a, s);       // S_0 aus A_0 erzeugen
s.debug();                 // Optional: Block ausgeben lassen
```

Zielblock darf mit einem der Eingabeblöcke identisch sein
→ Typische Anwendung:

```
xor_blocks(b, x, x);
encrypt_block(x, x);
```

C++-Gerüst: Die Klasse Block

- abstrakte Basisklasse für Operationen auf Blöcken
- intern: Byte-Array der Größe `Block::size()` (hier immer 16)
- `Block::debug` für Debug-Ausgaben
- Lesender Zugriff über Funktion `to_string()`
  - Liefert `string_view` mit der „richtigen“ Länge als Ergebnis
  - **nicht** Null-terminiert (Binärdaten!)
  - bietet **keine** Vergleichsoperationen mit konstanter Ausführungszeit

C++-Gerüst: Spezialisierungen von `Block`

- `BlockA`: Basis für A_0 … A_i
  - Präfix-Inkrement bildet A_{i+1} aus A_i (= Zähler hochzählen)
- `BlockB`: Basis für B_0 … B_i
  - Wird mit Nachricht initialisiert (Plaintext oder Ciphertext)
  - Präfix-Inkrement bildet B_{i+1} aus B_i
  - Schaltet nur weiter, bis n = $\lceil l(m)/16 \rceil$ erreicht
    Typische Nutzung: `while (b) { ++b; ...encrypt/decrypt... }`
- `BlockX`, `BlockS`: Nutzung als Ziel für Block-basierte Funktionen
- `BlockT`: zwei Varianten
  1. Berechnet MAC-Tag aus Ciphertext D und S_0
  2. Berechnet MAC aus S_0 und X_{n+1}
     - Methode `tag()` liefert `string`-Objekt der richtigen Länge

C++-Gerüst: Nutzung

- C++20 (kann jeder halbwegs aktuelle C++-Compiler. Bei Bedarf Linux aktualisieren oder im Makefile `-std=c++2a` einstellen.)
- Bauen unter Linux oder MacOS: `make`
  - benötigt ggf. Paket `libssl-dev` (bzw. libressl unter MacOS)
- Kommandozeilenparameter für Krypto-Parameter usw.
  - `-T`: Verzeichnis mit Testvektoren angeben
    Beispiel: `./myccm -T testvectors`
  - `-v 0`: Debug-Output von `Block::debug()` unterdrücken
  - Achtung: Leerzeichen zwischen Optionsbuchstabe und Argument (noch) zwingend erforderlich
  - Näheres in der Datei `README.md`

C++-Gerüst: Nutzung

```
$ ls -1 testvectors-one
testdata2.M8_L2_K2_N3-encrypted
testdata2.txt
$ ./myccm -T testvectors-one -v 0
Running unit tests (-T dir). Other options are ignored.
Found 1 test case in testvectors-one
"testdata2.M8_L2_K2_N3-encrypted"  Key id: 2, Nonce id: 3, M: 8, L: 2 \
  [decrypt OK] [encrypt OK]
2 tests were successful.
0 tests failed.
```

C++-Gerüst: Nutzung mit Debug-Ausgaben

```
$ ./myccm -T testvectors-one
./myccm -T testvectors-one
Running unit tests (-T dir). Other options are ignored.
Found 1 test case in testvectors-one
*******************************************************
"testdata2.M8_L2_K2_N3-encrypted"  Key id: 2, Nonce...
A0: 01 01 23 45 67 89 ab cd ef 01 23 45 67 89 00 00
S0: 69 13 94 bb 10 ab 63 a6 be 77 ad 15 b0 92 2b cb
T: f7 b5 bb 59 81 82 29 83
B0: 19 01 23 45 67 89 ab cd ef 01 23 45 67 89 00 25
X1: b0 0a 1d c4 9f 63 43 de 49 8f 43 bb cf f3 a2 b7
A1: 01 01 23 45 67 89 ab cd ef 01 23 45 67 89 00 01
S1: e3 48 67 ac f5 5a 72 45 0a af 85 73 86 83 a1 a8
B1: 45 69 6e 20 6b 6c 65 69 6e 65 72 20 54 65 78 74
X2: 91 67 dd 02 f7 ac 9a 56 17 4d 56 81 71 70 eb 73
A2: 01 01 23 45 67 89 ab cd ef 01 23 45 67 89 00 02
S2: 77 c5 5d c3 41 70 33 db eb 6f fa 7b a0 b5 0c bf
B2: 0a 7a 75 6d 20 54 65 73 74 65 6e 20 76 6f 6e 20
X3: 62 37 71 26 2d d8 88 12 3a 02 83 be a0 8d c9 1f
A3: 01 01 23 45 67 89 ab cd ef 01 23 45 67 89 00 03
S3: 3f 03 b8 2d 12 d5 10 c8 8d 7a 0d 24 64 e9 f9 3e
B3: 43 43 4d 2e 0a d5 10 c8 8d 7a 0d 24 64 e9 f9 3e
X4: f7 b5 bb 59 81 82 29 83 20 43 11 03 0c c9 7b b4
 [decrypt OK]
```

```
A0: 01 01 23 45 67 89 ab cd ef 01 23 45 67 89 00 00
B0: 19 01 23 45 67 89 ab cd ef 01 23 45 67 89 00 25
S0: 69 13 94 bb 10 ab 63 a6 be 77 ad 15 b0 92 2b cb
X1: b0 0a 1d c4 9f 63 43 de 49 8f 43 bb cf f3 a2 b7
A1: 01 01 23 45 67 89 ab cd ef 01 23 45 67 89 00 01
B1: 45 69 6e 20 6b 6c 65 69 6e 65 72 20 54 65 78 74
S1: e3 48 67 ac f5 5a 72 45 0a af 85 73 86 83 a1 a8
X2: 91 67 dd 02 f7 ac 9a 56 17 4d 56 81 71 70 eb 73
A2: 01 01 23 45 67 89 ab cd ef 01 23 45 67 89 00 02
B2: 0a 7a 75 6d 20 54 65 73 74 65 6e 20 76 6f 6e 20
S2: 77 c5 5d c3 41 70 33 db eb 6f fa 7b a0 b5 0c bf
X3: 62 37 71 26 2d d8 88 12 3a 02 83 be a0 8d c9 1f
A3: 01 01 23 45 67 89 ab cd ef 01 23 45 67 89 00 03
B3: 43 43 4d 2e 0a 00 00 00 00 00 00 00 00 00 00 00
S3: 3f 03 b8 2d 12 d5 10 c8 8d 7a 0d 24 64 e9 f9 3e
X4: f7 b5 bb 59 81 82 29 83 20 43 11 03 0c c9 7b b4
T: 9e a6 2f e2 91 29 4a 25
 [encrypt OK]
2 tests were successful.
0 tests failed.
```

Weitere Informationen

C- und C++-Standardbibliotheken:
https://cppreference.com
(Inhalte auch auf DevDocs verfügbar)

Referenz zum C++-Gerüst für myccm

Im Ordner `doc`:

- als PDF-Dokument `refman.pdf`
- als HTML-Dokumentsammlung mit Einstiegspunkt `html/index.html`
- unter https://rangpur.informatik.uni-bremen.de/ccm
  (mindestens für die Dauer der Blockwoche)