7. Übungsblatt
Kryptographie und Mediensicherheit (SoSe 2018)

Bauhaus-Universität Weimar, Professur für Mediensicherheit (Prof. Lucks)
Betreuer: Eik List
URL: `http://www.uni-weimar.de/de/medien/professuren/mediensicherheit/teaching`
Abgabe: 10.07.2018, 11:00 Uhr vor Beginn der Übung oder an `eik.list(at)uni-weimar.de`.
LaTeX-Template: template.tex

Aufgabe 1 – Simple MACs (5 Punkte)
Erläutern Sie für jeden der folgenden MACs ob sie sicher im SUF-CMA-Modell sind oder beschreiben Sie einen effizienten Angriff der eine Fälschung produziert.

a) $MAC_K(M) := \bigoplus_{i=1}^{m} E_K(M_i)$.

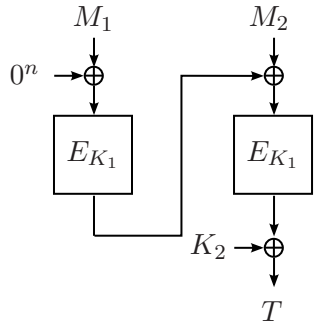
b) $MAC_K(M) := \bigoplus_{i=1}^{m} E_K(M_i \oplus \langle i \rangle)$. $\langle i \rangle$ bezeichne die n-bit-Repräsentation von i.

c) $MAC_K(M) := \bigoplus_{i=1}^{m} E_{K \oplus \langle i \rangle}(M_i)$.

Aufgabe 2 – CBC-MAC' (4 Punkte)
Sei $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ eine sichere Blockchiffre und $K_1, K_2 \twoheadleftarrow \{0,1\}^n$ zwei geheime zufällige unabhängige Schlüssel. Wir betrachten die folgende Variante von CBC-MAC: CBC-MAC'. Für eine m-Block- Nachricht $M = (M_1, \ldots, M_m)$ berechnet sich der Authentisierungstag mit $\text{CBC-MAC}_{K_1,K_2}(M)$ wie folgt

$$\begin{aligned} C_0 &= 0^n, \\ C_i &= E_{K_1}(C_{i-1} \oplus M_i), \qquad \text{for } 1 \le i \le m, \\ \text{CBC-MAC}'_{K_1,K_2}(M) &:= C_m \oplus K_2, \end{aligned}$$

Beschreiben Sie **entweder** einen effizienten Fälschungsangriff mit möglichst wenigen Anfragen **oder** erläutern Sie kurz warum dieser MAC sicher im UF-CMA-Modell ist. Ihre Anfragen dürfen beliebige (auch variable) Längen haben.



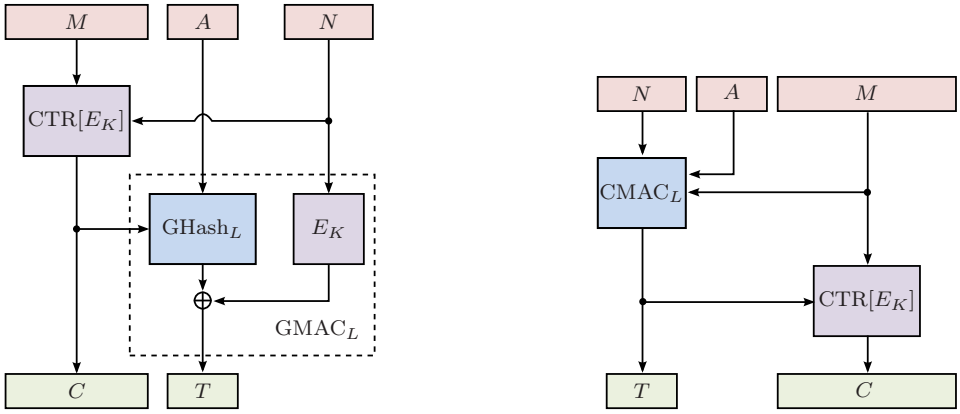
Aufgabe 3 – Authentisierte Verschlüsselung (4 Punkte)
Unten sehen Sie den vereinfachten schematischen Aufbau der Verschlüsselung mit GCM und SIV. Informieren Sie sich selbständig weiter. Die Details von GMAC und CMAC können Sie dabei ignorieren: Die Ausgaben von GMAC und CMAC sind nicht von zufälligen Bits unterscheidbar. Die Blockchiffre $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ sei sicher, K, L sind zwei geheime zufällige unabhängige Schlüssel.

GCM und SIV sind sichere Nonce-basierte Authentisierte Verschlüsselungsverfahren sofern Sie Nonces **NICHT** wiederholen. Sie nehmen jeweils eine Nonce N, Associated Data $A \in \{0,1\}^*$ und eine Nachricht $M \in \{0,1\}^*$ und erstellen einen Chiffretext C und einen Authentisierungstag T. Die Associated Data sind öffentliche Daten (IP-Adresse etc.) die nicht verschlüsselt aber mitauthentisiert werden. Sie können A in der Folge beliebig wählen, gerne auch konstant lassen. Der Counter-Mode in GCM nutzt eine 96-bit Nonce und hängt einen 32-bit-Counter $i = 1, 2, \ldots$ für die Verschlüsselung von Block i: $C_i = M_i \oplus E_K(N \,\|\, \langle i \rangle)$ for $1 \leq i < 2^{32}$.

Angenommen, Sie dürfen genau einmal eine Nonce wiederholen, d. h., das gleiche N für die Verschlüsselung zweier verschiedener Tupel (A, M) und (A', M') verwenden.

a) Beschreiben Sie **entweder** einen effizienten Angriff auf GCM im RoR-CPA-Modell mit möglichst wenigen Anfragen **oder** erläutern Sie kurz warum GCM auch bei Nonce-Wiederholung immer noch sicher ist.

b) Beschreiben Sie **entweder** einen effizienten Angriff auf SIV im RoR-CPA-Modell mit möglichst wenigen Anfragen **oder** erläutern Sie kurz warum SIV auch bei Nonce-Wiederholung immer noch sicher ist.



Schematischer Aufbau von GCM (links) und SIV (rechts).

Aufgabe 4 – Cipher Suites (2 Punkte)
Ihr Browser bietet die folgenden Optionen um die Kommunikation zwischen Ihnen und dem Server Ihrer Bank zu sichern. Recherchieren Sie notfalls selbständig. Wählen Sie für Schlüsselaustausch, Chiffre und Verschlüsselungsverfahren die jeweils sicherste Option aus und begründen Sie Ihre Wahl kurz:

- Schlüsselaustausch: (1) DH-3072, (2) DH-3072-Ephemeral, (3) Elliptic-Curve-DH-25519, (4) Elliptic-Curve-DH-25519-Ephemeral.

- Chiffre: (1) Salsa-20, (2) RC4-128, (3) 3DES-168, (4) AES-256.
- Verschlüsselungsverfahren: (1) CTR und HMAC, (2) CBC und CMAC, (3) GCM, (4) Salsa20 und Poly1305.

Aufgabe 5 – Programmieraufgabe (5 Punkte)
CBC muss Nachrichten padden (verlängern), deren Länge nicht ein Vielfaches der Blocklänge von n bytes ist. Eine beliebte Methode ist PKCS#7 (RFC 2315): Sei $M = M_1, \ldots, M_m$ eine Nachricht aus m Blöcken. Angenommen, dem letzten Block fehlen p Bytes um den letzten Block zu komplettieren. Dann werden p Bytes mit dem Wert p angehangen:

$$\mathrm{pad}(M_m) = M_m \,\|\, \underbrace{p \,\|\, p \,\|\, \ldots \,\|\, p}_{p \text{ mal}}.$$

Fehlen z. B. zwei Bytes, dann ist $\mathrm{pad}(M_m) = M_m \,\|\, \texttt{0x02} \,\|\, \texttt{0x02}$. Informieren Sie sich selbständig zum Padding-Orakel-Angriff auf CBC.

a) Implementieren Sie eine Python3-Klasse `CBCPaddingOracle` welche (IV, M) entgegennimmt, M mit PKCS#7 paddet und unter dem IV mit AES-CBC unter einem geheimen Schlüssel verschlüsselt. Sie können das `pycrypto`-Modul nutzen.

b) Implementieren Sie eine Funktion `verify_padding` in der Klasse `CBCPaddingOracle` die einen Chiffretext entgegennimmt und `True` zurückgibt wenn das Padding nach der Entschlüsselung stimmte und ansonsten `False` zurückgibt.

c) Implementieren Sie eine Klasse `CBCPaddingOracleAttack` die vom Padding-Orakel lediglich die Funktion `verify_padding` nutzt um zu einem gegebenen IV-Chiffretext-Paar erfolgreich den dazugehörenden Klartext wiederherzustellen.

Sie können Startcode für beide Klassen und einen Test auf der Webseite der Übung finden. Schicken Sie Ihre Lösung als Anhang einer E-Mail an `eik.list(at)uni-weimar.de` mit dem Betreff `[Krypto SS18] Beleg 7`. Es reicht dabei eine Matrikelnummer aus der Gruppe anzugeben. Die vollständigen Namen und Matrikelnummern sollen als Kommentar im Python-Skript stehen.
Hinweis: Sie können die `pycrypto`-Bibliothek für AES nutzen. Beachten Sie, dass einige Versionen der `pycrypto`-Implementation von AES-CBC den letzten Nachrichtenblock speichert und Sie den Zustand eventuell zurücksetzen müssen.