Karlsruhe Institute of Technology

Institut für Theoretische Informatik
Jun.-Prof. Dr. D. Hofheinz

Stammvorlesung Sicherheit im Sommersemester 2014

Übungsblatt 5

Hinweis: Übungsblätter können freiwillig bei Jessica Koch, Raum 256, Geb. 50.34 („Info-Bau“) bis zur Übung am 30.6.14 zur Korrektur abgegeben werden. Die Korrektur dient nur der Selbstkontrolle; es gibt keine Punkte und keinen Klausur-Bonus.

Aufgabe 1.

(a) In der Vorlesung wurde gezeigt, dass Lehrbuch-RSA-Signaturen nicht EUF-CMA-sicher sind, da ein Angreifer $\mathcal{A}$ beliebige Signaturen zu unsinnigen Nachrichten fälschen kann. Wir ändern nun das EUF-CMA Sicherheitsexperiment leicht ab, indem wir fordern, dass der Angreifer $\mathcal{A}$ zu einer bestimmten vom Challenger vorgegebenen Nachricht m^* eine Signatur σ^* fälschen muss, um das Spiel zu gewinnen. Er hat weiterhin ein Signaturorakel zur Verfügung, jedoch darf die Nachricht m^* trivialerweise nicht angefragt werden. Erfüllt das Lehrbuch-RSA-Signaturverfahren diesen neuen Sicherheitsbegriff? Beweisen Sie dies oder geben Sie einen erfolgreichen Angriff an.

(b) Wir beschreiben nun ein für die Praxis relevantes Signaturverfahren, das sogenannte Schnorr-Verfahren. Sei im Folgenden $\mathbb{G}$ eine Gruppe primer Ordnung p mit Generator g und $H : \{0,1\}^* \to \mathbb{Z}_p$ eine kryptographische Hashfunktion.

$\mathsf{Gen}(1^k)$**.** Der Schlüsselerzeugungsalgorithmus wählt einen Zufallswert $x \xleftarrow{\$} \mathbb{Z}_p$ und berechnet $y := g^x$. Das erzeugte Schlüsselpaar ist (pk, sk) mit

$$pk = (g, y) \qquad \text{und} \qquad sk = (g, x).$$

$\mathsf{Sig}(sk, m)$**.** Um eine Nachricht $m \in \{0,1\}^*$ zu signieren, wird ein Zufallswert $r \xleftarrow{\$} \mathbb{Z}_p$ gewählt und die Werte

$$t := g^r \in \mathbb{G}, \qquad c := H(t||m) \in \mathbb{Z}_p, \qquad s := cx + r \in \mathbb{Z}_p$$

berechnet. Die Signatur ist $\sigma := (t, s) \in \mathbb{G} \times \mathbb{Z}_p$.

$\mathsf{Ver}(pk, m, \sigma)$**.** Der Verifikationsalgorithmus gibt 1 aus, wenn die Gleichung

$$g^s \stackrel{?}{=} y^{H(t||m)} \cdot t$$

gilt, und ansonsten 0.

Aufgrund von schlechten Zufallszahlengeneratoren oder Implementierungsfehlern kann es vorkommen, dass der gleiche Zufall r mehrmals verwendet wird für verschiedene Signaturen. Was kann ein Angreifer dadurch lernen?

Aufgabe 2. Wir betrachten den Digital-Signature-Algorithmus (DSA) über der Gruppe $\mathbb{G} = Q(\mathbb{Z}_p^*)$, für ungerades primes $p \in \mathbb{N}$. Dabei sei $Q(\mathbb{Z}_p^*) := \{x^2 : x \in \mathbb{Z}_p^*\}$ die Menge der Quadrate in $\mathbb{Z}_p^*$.

(a) Erstellen Sie einen (DSA-)Public-Key $pk := (\mathbb{G}, g, g^x, (\mathsf{H}, h_1, h_2))$ sowie einen (DSA-)Secret-Key $sk := (\mathbb{G}, g, x, (\mathsf{H}, h_1, h_2))$. Dabei seien $p := 2q + 1$ (eine strong prime für q prim), $q = 11$, und eine Hashfunktion $\mathsf{H} : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{Z}_q^*$, $(x_1, x_2) \mapsto h_1^{x_1} h_2^{x_2} \bmod q$ gegeben. (Wir setzen $g := 8, h_1 := 4, h_2 := 2$.)

(b) Signieren Sie die Nachricht $M = (7, 3)$ mithilfe des Secret-Keys aus (a).

(c) Verifizieren Sie die Signatur zur Nachricht M aus (b) mittels des Public-Keys aus (a).

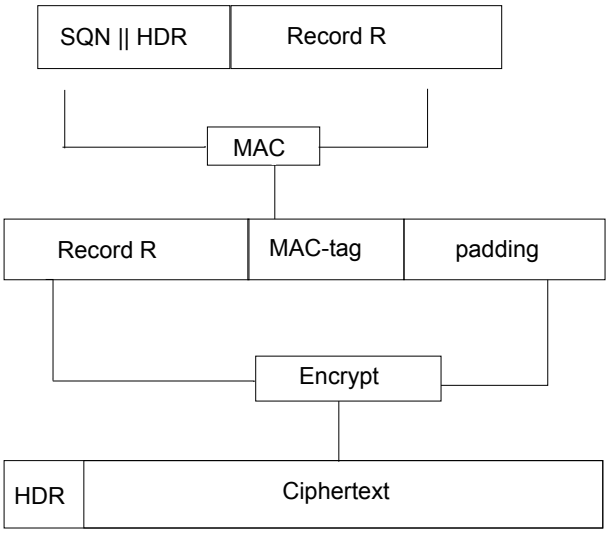

Abbildung 1: TLS Record Protocol

Aufgabe 3. Sei ein 2-Parteien-2-Nachrichten-Schlüsselaustauschverfahren $\mathsf{KE} = (\mathsf{KE.Gen}, \mathsf{KE.Encap}, \mathsf{KE.Decap})$ gegeben, wobei die KE-Algorithmen wie folgt beschrieben sind:

- Die Parametergenerierung $\mathsf{KE.Gen}(1^k)$ erhält als Eingabe den Sicherheitsparameter $k \in \mathbb{N}$ und gibt einen State s und eine Nachricht X aus. (Die Nachricht X stellt dabei die erste Nachricht im Schlüsselaustauschverfahren dar.)
- Die Schlüssel-Encapsulation $\mathsf{KE.Encap}(X)$ erhält als Eingabe eine Nachricht X, gibt eine Nachricht Y und einen Schlüssel K aus. (Die Nachricht Y entspricht dabei der zweiten ausgetauschten Nachricht im Schlüsselaustauschverfahren.)
- Die Schlüssel-Decapsulation $\mathsf{KE.Decap}(s, Y)$ erhält als Eingabe zusammen mit einem State s die Nachricht Y und gibt einen Schlüssel K' aus.

Wir fordern Korrektheit: Für alle $k \in \mathbb{N}$, für alle $(s, X) \leftarrow \mathsf{KE.Gen}(1^k)$, für $(K, Y) \leftarrow \mathsf{KE.Encap}(X)$ gilt $\mathsf{KE.Decap}(s, Y) = K$.

Ein Benutzer A berechnet demnach $(s, X) \leftarrow \mathsf{KE.Gen}(1^k)$ und sendet X an einen Benutzer B. B wiederum berechnet $(Y, K) \leftarrow \mathsf{KE.Encap}(X)$ und sendet Y an A. Schließlich erhält A den Schlüssel $K \leftarrow \mathsf{KE.Decap}(s, Y)$.

Konstruieren Sie ein Public-Key-Verschlüsselungssystem $\mathsf{PKE} = (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ aus KE.

Hinweis: Denken Sie an den Diffie-Hellman Schlüsselaustausch und seiner Beziehung zum ElGamal-Verschlüsselungsverfahren.

Aufgabe 4. Wir betrachten folgende CBC-Mode Verschlüsselung in TLS (siehe Abbildung 1). Alle benötigten Schlüssel wurden zuvor im Handshake Protocol ausgehandelt:

- Zu einer Sequenznummer SQN (8 Byte), einem Header HDR (5 Byte) und einer beliebigen Nachricht R wird durch ein MAC-Verfahren ein MAC-tag erzeugt.
- Danach wird $R||$MAC-tag$||$padding für ein geeignetes padding mit einer Blockchiffre E im CBC-Mode verschlüsselt.

Folgende Spezifikationen werden getroffen:

- Wir verwenden als Blockchiffre AES, welche die Nachricht in Blöcke der Länge 16 Byte aufteilt.
- Als MAC verwenden wir ein HMAC, der Iterationen von SHA-1 verwendet und somit eine Ausgabelänge von 20 Byte hat. Dabei gilt folgendes für die Eingabelänge $|\text{SQN}||\text{HDR}||R| =: L$:

    * Falls $L \leq 55$ Byte werden 4 Berechnungen durchgeführt.
    * Falls $L \geq 56$ Byte werden 5 oder mehr Berechnungen durchgeführt.

- Als padding verwenden wir folgendes Muster:
    * für 1 Byte nehmen wir die Byte-Darstellung von 00
    * für 2 Byte nehmen wir die Byte-Darstellung von 0101
    * für 3 Byte nehmen wir die Byte-Darstellung von 020202
    * $\vdots$
    * für 256 Byte nehmen wir die Byte-Darstellung von FF...FF
- Zur Verschlüsselung verwenden wir AES im CBC-Mode mit einem Schlüssel K, d.h. für
    * $P := R||\text{MAC-tag}||\text{padding} = P_1 \dots P_{18}$ gilt
    * $C_j := E_K(P_j \oplus C_{j-1})$, $C_0 :=$ IV zufällig
- Zur Entschlüsselung berechnen wir:
    * $P_j = D_K(C_j) \oplus C_{j-1}$
    * danach wird das padding entfernt
    * danach wird der MAC überprüft, indem die Header-Informationen verwendet werden

  Bei ungültigem padding werden die letzten 20 Byte als MAC-tag interpretiert und überprüft. Bei ungültigem MAC-tag wird eine Fehlermeldung *bad record mac* zurückgegeben.

Überlegen Sie, wie ein aktiver Angreifer das hier verwendete Design MAC-then-encrypt mithilfe von einem Seitenkanalangriff (timing attack) ausnutzen könnte.

Hinweis: Beachten Sie die Reihenfolge der Entschlüsselung, bei der selbst bei veränderten ungültigen Chiffraten eine MAC-Überprüfung durchgeführt wird. Die Zeit bis eine Fehlermeldung zurückgegeben wird ist für den Angreifer messbar und hängt von der Anzahl der Berechnungen (Länge der Eingabe) im MAC-Verfahren ab. Konstruieren Sie anhand dieser Informationen zwei Nachrichten, die Sie als aktiver Angreifer voneinander unterscheiden können.