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?

Lösungsvorschlag zu Aufgabe 1.

(a) Lehrbuch-RSA Signaturen erfüllen diesen Sicherheitsbegriff nicht.

Ein Angreifer erhält als Eingabe vom Challenger eine Nachricht m^*. Sein Ziel ist die Berechnung einer Signatur σ^* mit $(\sigma^*)^e \equiv m^* \bmod N$. Er geht dazu folgendermaßen vor:

1. Der Angreifer wählt einen zufälligen Wert $x \xleftarrow{\$} \mathbb{Z}_N^* \setminus \{1\}$ und berechnet $y \equiv x^e \bmod N$.

2. Dann berechnet der Angreifer $m_1 := m^* \cdot y \bmod N$, und fragt den Challenger nach einer Signatur für m_1. Weil $x \neq 1 \bmod N$ gewählt wurde, ist auch $y \neq 1 \bmod N$. Daher ist $m_1 \neq m^*$. Als Antwort erhält der Angreifer daher einen Wert σ_1 mit $\sigma_1^e \equiv m_1 \bmod N$.
3. Zum Schluss berechnet der Angreifer $\sigma^* \equiv \sigma_1 \cdot x^{-1} \bmod N$, und gibt σ^* aus. Dies ist möglich, weil $x \in \mathbb{Z}_N^*$ invertierbar ist. Ausserdem ist dies eine gültige Signatur für m^*, denn

$$(\sigma^*)^e \equiv (\sigma_1 \cdot x^{-1})^e \equiv \sigma_1^e \cdot (x^e)^{-1} \equiv m_1 \cdot y^{-1} \equiv m^* \cdot y \cdot y^{-1} \equiv m^* \bmod N.$$

Die Tatsache, dass Lehrbuch-RSA Signaturen multiplikativ homomorph sind, erlaubt also diesen Angriff.

(b) Sei $(g, y) = (g, g^x)$ ein öffentlicher Schlüssel des Schnorr-Verfahrens. Angenommen wir erhalten zwei gültige Signaturen $\sigma_1 = (t_1, s_1)$ und $\sigma_2 = (t_2, s_2)$ für zwei Nachrichten $m_1 \neq m_2$, sodass $t_1 = t_2$ gilt. Sei $r = \log_g t_1 = \log_g t_2$ der diskrete Logarithmus von $t_1 = t_2$ zur Basis g. Dann erhalten wir durch s_1 und s_2 zwei Gleichungen der Form

$$s_1 \equiv x \cdot H(t||m_1) + r \bmod p \qquad \text{und} \qquad s_2 \equiv x \cdot H(t||m_2) + r \bmod p.$$

Wenn wir diese voneinander abziehen, erhalten wir

$$s_1 - s_2 \equiv x \cdot H(t||m_1) - x \cdot H(t||m_2) \bmod p.$$

Falls $H(t||m_1) \neq H(t||m_2)$ gilt (weil $m_1 \neq m_2$ ist, gilt dies für eine kollisionsresistente Hashfunktion H mit einer Wahrscheinlichkeit von nahezu 1), so gilt $H(t||m_1) - H(t||m_2) \not\equiv 0 \bmod p$. Wir können also aus (s_1, s_2) den geheimen Schlüssel x berechnen durch

$$x \equiv \frac{s_1 - s_2}{H(t||m_1) - H(t||m_1)} \bmod p.$$

Der hier beschriebene Angriff widerspricht natürlich nicht dem Sicherheitsbeweis des Schnorr-Signaturverfahrens, da im Beweis angenommen wird, dass gute Zufallszahlen verwendet werden.

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).

Lösungsvorschlag zu Aufgabe 2. Der DSA ist Teil des Digital-Signature-Standards, der beispielsweise für US-Behörden gilt. Als mögliche Hashfunktionen werden darin Algorithmen der SHA-Familie empfohlen. In unserem Beispiel wählten wir jedoch eine beweisbar sichere, aber ineffizientere, Hashfunktion von Chaum, van Heijst und Pfitzmann. Um längere Nachrichten signieren zu können, benötigen wir allerdings eine Anpassung beziehungsweise eine Erweiterung der Hashfunktion H. In realen Anwendungen sollten p und q große Primzahlen sein, sodass p eine Bitlänge von mindestens 1024 Bit und q eine Länge von mindestens 160 Bit besitzen.

(a) Für $p = 2q + 1 = 23$ mit $q = 11$, ergibt sich

$$\mathbb{G} := Q(\mathbb{Z}_{23}^*) = \{1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18\}.$$

Wir ziehen x zufällig und gleichverteilt aus $\{0, \ldots, q-1\}$; und erhalten $x := 10$. Wir setzen den Public-Key als

$$pk := (\mathbb{G}, g, g^x \bmod p, (\mathsf{H}, h_1, h_2)) = (Q(\mathbb{Z}_{23}^*), 8, 3, (\mathsf{H}, 4, 2))$$

und den Secret-Key als

$$sk := (\mathbb{G}, g, x, (\mathsf{H}, h_1, h_2)) = (Q(\mathbb{Z}_{23}^*), 8, 10, (\mathsf{H}, 4, 2)).$$

(b) Wir führen den Signaturalgorithmus $\mathsf{Sig}(sk, M)$ mit $sk = (Q(\mathbb{Z}_{23}^*), 8, 10, (\mathsf{H}, 4, 2)) =: (\mathbb{G}, g, x, (\mathsf{H}, h_1, h_2))$ und $M = (7, 3)$ aus. Wir wählen zuerst ein e aus $\{0, \ldots, q-1\}$ zufällig und gleichverteilt; erhalten hier $e = 5$ und setzen $a := g^e \bmod p = 8^5 \bmod 23 = 16$. Der Hashwert der Nachricht ergibt sich zu

$$\mathsf{H}(M = (7, 3)) = h_1^7 h_2^3 \bmod q = 4^7 2^3 \bmod 11 = 7.$$

Wir lösen nun das Gleichungssystem

$$ax + eb = \mathsf{H}(M) \bmod q$$

nach b auf und erhalten

$$b = (\mathsf{H}(M) - ax)e^{-1} \bmod q = (7 - 16 \cdot 10) \cdot 9 \bmod 11 = 9.$$

Die Signatur zur Nachricht $M = (7, 3)$ lautet $\sigma := (a, b) = (16, 9)$.

(c) Die Ausführung des Verifizierungsalgorithmus $\mathsf{Ver}(pk, M, \sigma)$ mit $pk = (Q(\mathbb{Z}_{23}^*), 8, 3, (\mathsf{H}, 4, 2)) =: (\mathbb{G}, g, \tilde{g}, (\mathsf{H}, h_1, h_2))$, $M = (7, 3)$, und $\sigma = (16, 9) =: (a, b)$ überprüft, ob die Gleichung

$$\tilde{g}^a \cdot a^b \bmod p = g^{\mathsf{H}(M)} \bmod p$$

erfüllt ist. Wir erhalten $3^{16} 16^9 \bmod 23 = 8^7 \bmod 23 = 12$ und somit ist die Signatur σ zur Nachricht M verifiziert.

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.

Lösungsvorschlag zu Aufgabe 3. Wir beschreiben die Algorithmen für PKE :

- Die Schlüsselgenerierung $\mathsf{PKE.Gen}(1^k)$ erhält als Eingabe den Sicherheitsparameter $k \in \mathbb{N}$, führt $(s, X) \leftarrow \mathsf{KE.Gen}(1^k)$ aus, setzt den Public-Key als $pk := X$, den Secret-Key als $sk := s$ und gibt (pk, sk) aus.
- Die Verschlüsselung $\mathsf{PKE.Enc}(pk, M)$ erhält als Eingabe den Public-Key pk und eine Nachricht M, berechnet $(Y, K) \leftarrow \mathsf{KE.Encap}(pk)$ und gibt ein Chiffrat $C := (Y, K \oplus M)$ aus. (Hinweis: Y entspricht hier Zufall, der im Chiffrat mitübergeben werden muss. Würde man nochmals $\mathsf{KE.Encap}(pk)$ ausführen, würde man andere Y', K' erhalten, da $\mathsf{KE.Encap}(pk)$ ein probabilistischer Algorithmus ist.)
- Die Entschlüsselung $\mathsf{PKE.Dec}(sk, C)$ erhält als Eingabe den Secret-Key sk und das Chiffrat $C =: (C_1, C_2)$, berechnet $K' := \mathsf{KE.Decap}(sk, C_1)$ und gibt $M := C_2 \oplus K'$ aus.

Korrektheit überträgt sich von KE auf PKE. (Es gilt demnach $K = K'$.) Somit gilt für alle $k \in \mathbb{N}$, alle $(pk, sk) \leftarrow \mathsf{PKE.Gen}(1^k)$, alle M und alle $C \leftarrow \mathsf{PKE.Enc}(pk, M)$, dass $\mathsf{PKE.Dec}(sk, C) = M$ erfüllt.

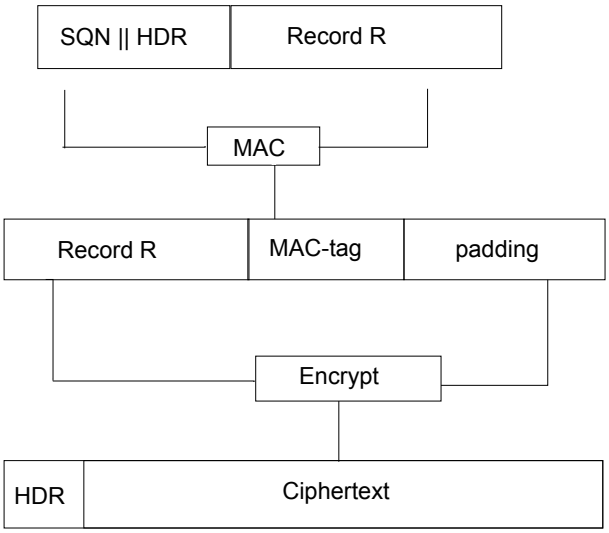

Abbildung 1: TLS Record Protocol

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
  * $P := R||$MAC-tag$||$padding $= P_1 \dots P_n$ 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 gleicher Länge, die Sie als aktiver Angreifer voneinander unterscheiden können.

Lösungsvorschlag zu Aufgabe 4. Ein Angreifer $\mathcal{A}$ der zwei selbst gewählte Nachrichten M_0 und M_1 gleicher Länge voneinander unterscheiden möchte geht folgendermaßen vor:

- Zuerst konstruiert $\mathcal{A}$ zwei Nachrichten der Länge 288 Byte (Vielfaches von 16), wobei $M_0 := R^{287}||00$ und $M_1 := R^{32}||FF^{256}$ für beliebiges R der Länge 287 bzw. 32 Byte und einem gültigen padding der Länge 1 bzw 256 Byte.
- M_b für $b \in \{0,1\}$ wird entsprechend der Beschreibung verschlüsselt und $\mathcal{A}$ erhält den Initialisierungsvektor IV und HDR$||C_b$, wobei $C_b = E_K(M_b||\text{MAC-tag}||\text{padding})$ mit $|C_b| = 288 + x$ Byte.
- $\mathcal{A}$ schneidet die letzten x Byte von C_b ab, so dass das veränderte Chiffrat C'_b eine Länge von 288 Byte besitzt. $\mathcal{A}$ lässt C'_b entschlüsseln und misst die Zeit bis eine Fehlermeldung kommt.

Folgende Fälle kann $\mathcal{A}$ unterscheiden:

1. Falls $D_K(C'_b) = M_0 = R^{287}||00$ wird danach entsprechend der Beschreibung der Entschlüsselung ein gültiges padding, in diesem Falle 00 entfernt und die letzten 20 Byte als MAC-tag interpretiert und überprüft. Dazu wird der SHA-1 Wert der restlichen Nachricht R^{267} zusammen mit der Sequenznummer SQN und dem Header HDR berechnet, wobei $|\text{SQN}||\text{HDR}||R^{267}| = 280$ Byte.
2. Falls $D_K(C'_b) = M_1 = R^{32}||FF^{256}$ wird danach entsprechend der Beschreibung der Entschlüsselung ein gültiges padding, in diesem Falle FF^{256} entfernt und die letzten 20 Byte als MAC-tag interpretiert und überprüft. Dazu wird der SHA-1 Wert der restlichen Nachricht R^{12} zusammen mit der Sequenznummer SQN und dem Header HDR berechnet, wobei $|\text{SQN}||\text{HDR}||R^{12}| = 25$ Byte.

In beiden Fällen wird eine Fehlermeldung zurückgegeben, da die Überprüfung des MAC-tag mit großer Wahrscheinlichkeit nicht erfolgreich war. Die Anzahl der Berechnungen zur Überprüfung sind in beiden Fällen unterschiedlich viele und führen zu einem messbaren Zeitunterschied, den $\mathcal{A}$ ausnutzen kann um M_0 und M_1 zu unterscheiden.

Fazit: Die Überprüfung der Integrität sollte zuerst erfolgen! (Encrypt-then-MAC)