Institut für Theoretische Informatik
Prof. Dr. Dennis Hofheinz
Übungsleiter: Thomas Agrikola

Stammvorlesung Sicherheit im Sommersemester 2017

Übungsblatt 5

Aufgabe 1.

(1.) 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^* natürlich nicht angefragt werden. Erfüllt das Lehrbuch-RSA-Signaturverfahren diesen neuen Sicherheitsbegriff? Beweisen Sie dies oder geben Sie einen erfolgreichen Angriff an.

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

(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 $\mathsf{H} : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{Z}_q^\times,\ (x_1, x_2) \mapsto h_1^{x_1} h_2^{x_2} \bmod q$ 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 1.

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

Der Angreifer $\mathcal{A}$ erhält als Eingabe vom Challenger eine Nachricht m^* und den Public-Key $(e, N = P \cdot Q)$ (P, Q prim). Das Ziel des Angreifers $\mathcal{A}$ ist die Berechnung einer Signatur σ^* mit $(\sigma^*)^e \equiv m^* \bmod N$.

Angenommen m^* wäre nicht aus $\mathbb{Z}_N^\times$, also nicht invertierbare modulo N. Wir wissen, dass ein Element $x \in \mathbb{Z}_N$ genau dann invertierbar ist, wenn $ggT(x, N) = 1$ gilt. Ist m^* also nicht invertierbar, gilt somit $ggT(m^*, N) \neq 1$. Da $m^* < N$ muss dann aber entweder $ggT(m^*, N) = P$ oder $ggT(m^*, N) = Q$ gelten, da N nur die Teiler N, P, Q und 1 hat. Somit könnte der Angreifer also mithilfe eines nichtinvertierbaren m^* direkt N faktorisieren und damit den geheimen Schlüssel berechnen. Kennt er den geheimen Schlüssel, kann er natürlich leicht eine Signatur fälschen.

Wir müssen also noch den Fall betrachten, dass m^* invertierbar ist. $\mathcal{A}$ geht in diesem Fall folgendermaßen vor:

1. $\mathcal{A}$ wählt einen zufälligen Wert $x \xleftarrow{\$} \mathbb{Z}_N^\times \setminus \{1\}$ und berechnet $y \equiv x^e \bmod N$.
2. $\mathcal{A}$ berechnet nun $m_1 := m^* \cdot y \bmod N$ und fragt sein Sig-Orakel nach einer Signatur für m_1. Als Antwort erhält der Angreifer nun eine Signatur σ_1 mit $\sigma_1^e \equiv m_1 \bmod N$. Weil $x \neq 1 \bmod N$ gewählt wurde, ist auch $y \neq 1 \bmod N$ (dies folgt im Wesentlichen aus der Korrektheit von RSA bzw. daher, dass Exponieren mit $e \bmod N$ eine Bijektion ist).

   Wir müssen zeigen, dass $m_1 \neq m^* \bmod N$ (da Sig-Orakelanfragen für m^* nicht erlaubt sind). Dies folgt daraus, dass m^* invertierbar ist und $y \neq 1 \bmod N$ gilt. Denn wäre $m_1 = m^* \bmod N$, so müsste $y = 1 \bmod N$ sein. Dies sieht man folgendermaßen ein:

$$\begin{aligned} & m = m^* && \bmod N \\ \Leftrightarrow\ & m^* \cdot y = m^* && \bmod N \\ \Leftrightarrow\ & y = m^* \cdot (m^*)^{-1} && \bmod N \\ \Leftrightarrow\ & y = 1 && \bmod N. \end{aligned}$$

3. Zum Schluss berechnet der Angreifer $\sigma^* \equiv \sigma_1 \cdot x^{-1} \bmod N$, und gibt σ^* aus. Dies ist möglich, weil $x \in \mathbb{Z}_N^\times$ invertierbar ist. Außerdem ist dies eine gültige Signatur für m^*, denn

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

In beiden Fällen ist die Erfolgwahrscheinlichkeit des Angreifers gleich 1 und seine Laufzeit polynomiell. Die Tatsache, dass Lehrbuch-RSA Signaturen multiplikativ homomorph sind, erlaubt also diesen Angriff.

(b) 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

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

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

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

und den Secret-Key als

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

(b) Wir führen den Signaturalgorithmus $\mathsf{Sig}(sk, M)$ mit $sk = (Q(\mathbb{Z}_{23}^\times), 8, 10, (\mathsf{H}, 4, 2)) =: (\mathbb{G}, g, x, (\mathsf{H}, h_1, h_2))$ 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 $pk = (Q(\mathbb{Z}_{23}^\times), 8, 3, (\mathsf{H}, 4, 2)) =: (\mathbb{G}, g, \tilde{g}, (\mathsf{H}, h_1, h_2))$, $M = (7, 3)$, 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 2. Führen Sie einen Diffie-Hellman-Schlüsselaustausch über der Gruppe $\mathbb{G} = Q(\mathbb{Z}_p^\times)$ ($Q(\mathbb{Z}_p^\times)$ bezeichne wie in Aufgabe 1 die Untergruppe der Quadrate), für ungerades primes $p \in \mathbb{N}$, durch. Wir setzen $p := 23$. Folgende Schritten werden ausgeführt:

(i) A und B einigen sich auf einen Generator g der Gruppe $\mathbb{G}$, sodass $\mathbb{G} = \langle g \rangle$ gilt. Für diese Aufgabe sei $g := 18$. Die Ordnung von $\mathbb{G}$ ist 11.

(ii) A wählt zufällig gleichverteilt ein $x \in \{0, \ldots, |\mathbb{G}| - 1\}$ und schickt $X := g^x \bmod p$ an B. A wählt hier $x := 6$.

(iii) B wählt zufällig gleichverteilt ein $y \in \{0, \ldots, |\mathbb{G}| - 1\}$ und schickt $Y := g^y \bmod p$ an A. B wählt hier $y := 8$.

(iv) A berechnet $K := Y^x = (g^y)^x = g^y \bmod p$ und B berechnet $K := X^y = (g^x)^y = g^{xy} \bmod p$. Somit haben sich beide auf K als gemeinsamen Schlüssel geeinigt.

Berechnen Sie X, Y und K.

Lösungsvorschlag zu Aufgabe 2. Es ergeben sich die folgenden Werte:

$$\begin{aligned} X &= g^x \bmod 23 \\ &= 18^6 \bmod 23 \\ &= (-5)^6 \bmod 23 \\ &= 2^3 \bmod 23 \\ &= 8 \bmod 23, \end{aligned}$$

$$\begin{aligned} Y &= g^y \bmod 23 \\ &= 18^8 \bmod 23 \\ &= (-5)^8 \bmod 23 \\ &= 2^4 \bmod 23 \\ &= 16 \bmod 23, \end{aligned}$$

$$\begin{aligned} K &= g^{xy} \bmod 23 \\ &= 18^{48} \bmod 23 \\ &\overset{(*)}{=} (-5)^4 \bmod 23 \\ &= 2^2 \bmod 23 \\ &= 4 \bmod 23. \end{aligned}$$

Bei $(*)$ nutzen wir aus, dass die Ordnung von $\mathbb{G}$ gleich 11 ist.

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.

Aufgabe 4. Der ebenso geniale wie modebewusste Wissenschaftler und Superbösewicht Doktor Meta ist betrübt. Seinen Recherchen zufolge sind Säurebecken, Todesstrahlen und Raubtierkäfige nicht mehr zeitgemäß. Er möchte sein geheimes Untergrundlabor deshalb neu einrichten. Nachdem die Einführung seiner Kryptowährung MetaCoin nicht den gewünschten Erfolg hatte, reichen seine Finanzen dafür aber nicht aus. Im Internet hat er von einem Angriff namens „CRIME“ auf das TLS-Protokoll gelesen. Einerseits entspricht der Name genau seinem Geschmack, andererseits erhofft sich Doktor Meta mit Hilfe dieses Angriffs, die Sessioncookies der Kunden von *Villains'R'Us*[1] auszuspionieren. Dadurch könnte er sowohl sein Labor neu einrichten, als auch gleichzeitig konkurrierenden Superbösewichten schaden.

(a) Wir wollen das „Nachricht komprimieren, dann verschlüsseln“-Prinzip (welches den CRIME-Angriff ermöglicht) zuerst aus theoretischer Perspektive untersuchen. Sei dazu $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ ein IND-CPA-sicheres Public-Key-Verschlüsselungsverfahren mit dem Nachrichtenraum $\mathcal{M}$. Zu PKE ist ein PPT-Algorithmus $\mathcal{L}$ bekannt, für den die Wahrscheinlichkeit

$$\Pr\left[\mathcal{L}(pk, C) = |M| \mid (pk, sk) \leftarrow \mathsf{Gen}(1^k), C \leftarrow \mathsf{Enc}(pk, M)\right]$$

für alle Nachrichten $M \in \mathcal{M}$ gleich 1 ist, d.h. $\mathcal{L}$ kann aus einem Chiffrat die Länge des verschlüsselten Klartextes berechnen.

Zusätzlich sei $\mathsf{C} = (\mathsf{Comp}, \mathsf{Decomp})$ ein Kompressionsalgorithmus (Comp komprimiert, Decomp dekomprimiert). Gehen Sie für die Bearbeitung davon aus, dass Nachrichten gleicher Länge nicht zwingend auf die gleiche Länge komprimiert werden, sondern diese Länge umso kürzer ist, je redundanter die Nachricht war (beispielsweise würde die Nachricht $aaaaaaaaaa$ auf einen kürzeren Text komprimiert als $abcdefghij$).

Es sei nun $\mathsf{PKE}' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ gegeben durch:

- $\mathsf{Gen}'(1^k) = \mathsf{Gen}(1^k)$,
- $\mathsf{Enc}'(pk, M) = \mathsf{Enc}(pk, \mathsf{Comp}(M))$,

[1] Der führende Online-Händler für den Bedarf eines Superbösewichts – Superlaser, Haifischbecken, Reagenzgläser und viele weitere Artikel sind ständig auf Lager.

- $\mathsf{Dec}'(sk, C) = \mathsf{Decomp}(\mathsf{Dec}(sk, C))$.

Zeigen Sie: PKE' ist nicht IND-CPA-sicher.

Überlegen Sie, wie die Aufgabe zu lösen ist, wenn die Wahrscheinlichkeit, dass $\mathcal{L}$ die richtige Länge berechnet, nicht gleich 1, sondern nur *überwältigend* ist, d.h.

$$\Pr\left[\mathcal{L}(pk, C) = |M| \mid (pk, sk) \leftarrow \mathsf{Gen}(1^k), C \leftarrow \mathsf{Enc}(pk, M)\right] = 1 - f(k),$$

wobei $f(k)$ eine in k vernachlässigbare Funktion ist.

(b) Schreiben Sie ein Programm, beispielsweise ein Pythonskript, das den CRIME-Angriff bei eingeschalteter Komprimierung in TLS simuliert. (Nutzen Sie dazu auch die entsprechenden Folien aus der Vorlesung.) Vereinfacht können wir annehmen, dass ein POST-HTTP-Header wie folgt aussieht:

```
POST / HTTP/1.1
Host: host.edu
Cookie: <cookie data>
```

Das Ziel ist es das Cookie, also `<cookie data>`, aus den komprimierten und verschlüsselten Daten zu extrahieren. Die Komprimierung kann über Kompressionsbibliotheken, wie zlib in Python, realisiert werden. Das folgende unvollständige Pythonskript kann verwendet werden.

```
# This python script is based on a script by Tibor Jager.

import string
import zlib
import random

# random cookie data with length 8 to 20 consisting of letters, digits, and punctuation
cookie_data = ''.join(random.choice(string.letters + string.digits + string.punctuation)
  for x in range(random.randint(8,20)))

# POST HTTP header
HEADER = ("POST / HTTP/1.1\r\n"
          "Host: host.edu\r\n"
          "Cookie: " + cookie_data + "\r\n")

# use Python's compression funcion zlib
def comp(string):
    return zlib.compress(string)

# to make it easier, the chosen plaintext oracle only outputs
#  the length of a compressed message of the form HEADER + M
def chosen_plaintext_oracle(M):
    # concatenate HEADER and M
    ...
    # use compression comp
    c = ...
    # return length of a compressed message
    return len(c)

# we try to extract the cookie data here
cookie_extr = ""
cnt = 0
while len(cookie_extr) < len(cookie_data):
    minimum = 2**80
    candidate = ""
    # try different candidates and compare chosen_plaintext_oracle outputs
```

```
    for x in (string.letters + string.digits + string.punctuation):
        cnt+=1
        # query oracle with chosen plain texts
        ctxt_len = chosen_plaintext_oracle("Cookie: " + cookie_extr + ...)
        if ctxt_len < minimum:
           minimum = ctxt_len
           candidate = x
    cookie_extr += candidate

# print out result
...
```

Lösungsvorschlag zu Aufgabe 4.

(a) Wir betrachten den folgenden PPT-Angreifer $\mathcal{A}$: $\mathcal{A}$ erhält zuerst den Public Key pk. Danach geht $\mathcal{A}$ folgendermaßen vor:

- $\mathcal{A}$ erhält pk und wählt zwei Nachrichten *gleicher Länge* M_0, M_1 mit $|\mathsf{Comp}(M_0)| \neq |\mathsf{Comp}(M_1)|$.
- $\mathcal{A}$ schickt M_0, M_1 als Challenge an das Experiment.
- Das Experiment zieht $b \leftarrow \{0,1\}$ zufällig und berechnet $C^* \leftarrow \mathsf{Enc}'(pk, M_b) = \mathsf{Enc}(pk, \mathsf{Comp}(M_b))$.
- $\mathcal{A}$ erhält das Challenge-Chiffrat C^*.
- Er führt nun $\mathcal{L}(pk, C^*)$ aus und erhält von $\mathcal{L}$ einen Wert x.
- $\mathcal{A}$ überprüft, ob $|\mathsf{Comp}(M_0)| = x$, wenn ja, gibt er 0 aus, sonst geht er weiter zum nächsten Schritt.
- $\mathcal{A}$ überprüft, ob $|\mathsf{Comp}(M_1)| = x$, wenn ja, gibt er 1 aus, sonst rät er ein zufälliges Bit.

Wir müssen nun zeigen, dass $\Pr[\mathcal{A} \text{ gewinnt}] - 1/2$ nicht vernachlässigbar ist. Bezeichne dazu E das Ereignis, dass $\mathcal{L}$ für C^* die korrekte Klartextlänge ausgibt. Nach Voraussetzung ist dann $\Pr[\mathsf{E}] = 1$ und $\Pr[\overline{E}] = 0$. Wir können nun $\Pr[\mathcal{A} \text{ gewinnt}] - 1/2$ folgendermaßen analysieren:

$$\begin{aligned}\Pr[\mathcal{A} \text{ gewinnt}] - 1/2 &= \Pr[(\mathcal{A} \text{ gewinnt } \wedge \mathsf{E}) \vee (\mathcal{A} \text{ gewinnt } \wedge \overline{\mathsf{E}})] - 1/2 \\ &= \Pr[\mathsf{E}] \Pr[\mathcal{A} \text{ gewinnt } | \mathsf{E}] + \Pr[\overline{\mathsf{E}}] \Pr[\mathcal{A} \text{ gewinnt } | \overline{\mathsf{E}}] - 1/2 \\ &= 1 \cdot \Pr[\mathcal{A} \text{ gewinnt } | \mathsf{E}] + 0 - 1/2\end{aligned}$$

Die Wahrscheinlichkeit $\Pr[\mathcal{A} \text{ gewinnt } | \mathsf{E}]$ ist ebenfalls genau 1, denn wenn E eintritt, wird $\mathcal{A}$ ebenfalls mit Wahrscheinlichkeit 1 bestimmen, ob M_0 oder M_1 verschlüsselt wurde. Beide Nachrichten wurden so gewählt, dass sie zu verschieden langen Nachrichten komprimiert werden, weshalb er in diesem Fall eindeutig entscheiden kann. Insgesamt gilt also

$$\Pr[\mathcal{A} \text{ gewinnt}] - 1/2 = \Pr[\mathcal{A} \text{ gewinnt } | \mathsf{E}] - 1/2 = 1/2,$$

was nicht vernachlässigbar ist.

Zur Bonus-Frage: In diesem Fall gilt nicht $\Pr[\mathsf{E}] = 1$ und $\Pr[\overline{\mathsf{E}}] = 0$, sondern

$$\begin{aligned}\Pr[\mathsf{E}] &= 1 - f(k), \\ \Pr[\overline{\mathsf{E}}] &= 1 - \Pr[\mathsf{E}] = f(k),\end{aligned}$$

wobei f eine in k vernachlässigbare Funktion ist, weshalb wir etwas genauer bei der Analyse vorgehen müssen. Tritt E ein, so ist die Erfolgswahrscheinlichkeit von $\mathcal{A}$ auch hier gleich 1. Wir können nun die folgende Abschätzung vornehmen:

$$\begin{aligned}\Pr[\mathcal{A} \text{ gewinnt}] - 1/2 &= \Pr[\mathsf{E}] \Pr[\mathcal{A} \text{ gewinnt } | \mathsf{E}] + \Pr[\overline{\mathsf{E}}] \Pr[\mathcal{A} \text{ gewinnt } | \overline{\mathsf{E}}] - 1/2 \\ &\geq \Pr[\mathsf{E}] \cdot 1 - 1/2 \\ &= (1 - f(k)) - 1/2 \\ &= 1/2 - f(k).\end{aligned}$$

Es bleibt nun noch zu zeigen, dass $g(k) := 1/2 - f(k)$ nicht vernachlässigbar ist. Man kann zeigen, dass für $k \to \infty$ jede vernachlässigbare Funktion gegen 0 geht. Es gilt aber

$$\lim_{k\to\infty} g(x) = \lim_{k\to\infty} 1/2 - f(k) = 1/2 - 0 = 1/2 \neq 0,$$

weshalb $g(k)$ nicht vernachlässigbar ist.

(b) Das folgende Pythonskript simuliert einen CRIME-Angriff auf TLS bei eingeschalteter Kompression:

```
# This python script is based on a script by Tibor Jager.

import string
import zlib
import random

# random cookie data with length 8 to 20 consisting of letters, digits, and punctuation
cookie_data = ''.join(random.choice(string.letters + string.digits + string.punctuation)
  for x in range(random.randint(8,20)))

# POST HTTP header
HEADER = ("POST / HTTP/1.1\r\n"
          "Host: host.edu\r\n"
          "Cookie: " + cookie_data + "\r\n")

# use Python's compression funcion zlib
def comp(string):
    return zlib.compress(string)

# to make it easier, the chosen plaintext oracle only outputs
#  the length of a compressed message of the form HEADER + M
def chosen_plaintext_oracle(M):
    # concatenate HEADER and M
    # use compression comp
    # return length of a compressed message
    return len(comp(HEADER+M))

# we try to extract the cookie data here
cookie_extr = ""
cnt = 0
while len(cookie_extr) < len(cookie_data):
    minimum = 2**80
    candidate = ""
    # try different candidates and compare chosen_plaintext_oracle outputs
    for x in (string.letters + string.digits + string.punctuation):
        cnt+=1
        # query oracle with chosen plain texts
        ctxt_len = chosen_plaintext_oracle("Cookie: " + cookie_extr + x)
        if ctxt_len < minimum:
           minimum = ctxt_len
           candidate = x
    cookie_extr += candidate

# print out result
print "Cookie data:           " + cookie_data
print "Extracted cookie data: " + cookie_extr
print "Queries to oracle:     " + str(cnt)
```