**Cryptography**
Übungsblatt 5

# Hackerpraktikum

FAU Erlangen-Nürnberg, Lehrstuhl für Informatik 1

### CBC Malleability (Aufgabe 1)

In dieser Aufgabe sollen Sie den bekannten Klartext einer AES-CBC verschlüsselten Nachricht manipulieren. Dazu läuft auf Host `10.0.23.61` Port 1337 ein TCP-Server, der Kommandos der folgenden Form verarbeitet um einen gegebenen Geldbetrag auf ein gegebenes Konto zu überweisen:

```
TRANSFER AMOUNT <amount> REASON <reason> DEST <dest> END
```

Um die Authentizität der Tranksaktionsanfragen sicherzustellen erwartet der Server die Anfragen AES-CBC verschlüsselt (AES-128) und anschliessend Base64 kodiert.

Sie wurden angewiesen folgende Transaktion durchzuführen:

```
TRANSFER AMOUNT $1000000 REASON Salary Jan. 2016 DEST #78384 END
```

Die dazu gültige Anfrage für den Server lautet:

```
wUHhFdm5le/fLoF/G4UOu6FGSNVtkxFA3ZIEwYombzhGF2eYUCOutHTg0h16BtYlBd5FO/XlJkQ058Ev+8hTIA==
```

1. Nutzen Sie diese Informationen um den Betrag stattdessen auf Ihr Konto #31337 zu überweisen. (2.5 P.)
2. Welches kryptographische Prinzip wurde verletzt und warum ist AES-CBC nicht geeignet um Authentizität sicherzustellen? Was würden sie stattdessen verwenden? (0.5 P.)

**Tipp:** Verwenden Sie den bereitgestellten Servercode um ihre Manipulationen lokal zu testen. Sie können dazu aus der gegebenen Transaktion eine eigene Anfrage generieren.

3 P.

### HMAC Length Extension Attack (Aufgabe 2)

Unter `http://10.0.23.61` läuft ein Webserver, auf dem man eine „Message of the Day“ setzen kann. Um das setzen beliebiger Nachrichten zu verhindern, muss man zusätzlich zur Nachricht $M$ eine HMAC basierend auf einem geheimen Schlüssel $K$ angeben. Die HMAC $H$ wird dabei wie folgt berechnet:

$$H = \text{SHA256}\,(K||M)$$

Die HMAC zur aktuellen Nachricht (`motd.txt`) lautet:

`69268ba87558295eedb751d8f4744b58bd2705ce5d09984f31927bb7fbfe9b97`

Nutzen Sie diese Informationen um eine Zeichenkette Ihrer Wahl an die aktuelle Nachricht anzuhängen, eine gültige HMAC dafür zu generieren und diese Nachricht auf dem Webserver zu platzieren. Geben Sie die erweiterte Nachricht zusammen mit Ihrer neuen HMAC auch als Textdatei ab. Warum muss Ihre Zeichnkette zwingenderweise mit dem Buchstaben "H"beginnen?

**Hinweis:** Der Server ignoriert nicht-ASCII Zeichen, die möglicherweise in Ihrer erweiterten Nachricht vorkommen.
**Tipp:** Um eine HMAC für Ihre erweiterte Nachricht zu generieren benötigen Sie neben der aktuellen Nachricht und der aktuellen HMAC auch die Länge des geheimen Schlüssels. Ermitteln Sie die Länge durch Bruteforce, indem Sie mehrere erweiterte Nachrichten mit verschiedenen Annahmen für die Länge an den Server senden.

5 P.

### RSA Small Exponent (Aufgabe 3)

Eine geheime Nachricht wurde an drei verschiedene Empfänger RSA verschlüsselt versendet. Sie haben die verschlüsselten Nachrichten (`msg1.bin`, `msg2.bin`, `msg3.bin`) abgefangen und auch die drei öffentlichen Schlüssel der Empfänger (`pk1.pem`, `pk2.pem`, `pk3.pem`) liegen Ihnen vor. Des Weiteren wissen Sie, dass RSA ohne ein Padding Scheme verwendet wurde, d.h. die verschlüsselten Daten spiegeln direkt das Ergebnis des RSA Algorithmus wieder.

1. Untersuchen Sie mit Hilfe von OpenSSL die öffentlichen Schlüssel und bestimmen Sie die verwendeten Parameter. Was fällt Ihnen auf? (1 P.)
2. Nutzen Sie Ihre Erkenntnis um die Nachricht ohne Kenntnis eines privaten Schlüssels dennoch zu entschlüsseln. (4 P.)

**Hinweis:** Verwenden Sie existierende Bibliotheken für RSA und andere mathematische Operationen. Falls Sie Floating Point Operationen durchführen müssen, achten Sie auf ausreichende Genauigkeit, zum Beispiel mit Hilfe einer Multi Precision Library.

5 P.

### ECDSA Fixed K (Aufgabe 4)

Sie haben zwei Nachrichten (`msg1.txt`, `msg2.txt`) mit dazugehörigen ECDSA-Signaturen im DER-Format (`msg1.sig`, `msg2.sig`) erhalten. Leider ist dem Besitzer des öffentlichen ECDSA Schlüssels (`vk.pem`) ein Fehler unterlaufen, denn er hat die Nachrichten trotz mangelnder Entropie signiert. Nach einem Blick auf die Verifikationsroutinen (`ecdsa-openssl-verify.sh`) haben Sie erkannt, dass als zugrundeliegender Hash der Signaturen SHA-1 verwendet wird.

1. Nutzen Sie OpenSSL um die Kurvenparameter des öffentlichen Schlüssels auszulesen. Wie lauten diese? (0.5 P.)
2. Vergleichen Sie die beiden Signaturen mit einem Hex-Editor. Welche Komponenten sind identisch und was folgt daraus? (0.5 P.)
3. Berechnen Sie nun aus beiden Signaturen und aus Informationen des öffentlichen Schlüssels den privaten ECDSA Schlüssel mit dem diese Signaturen erstellt wurden. Speichern Sie den berechneten Schlüssel wieder im PEM-Format und dokumentieren Sie Ihr Vorgehen ausführlich. (5 P.)
4. Nutzen Sie Ihren soeben berechneten privaten Schlüssel um eine Nachricht Ihrer Wahl zu signieren. Sie sollten diese Signatur nun mit dem gegebenen öffentlichen Schlüssel (`vk.pem`) verifizieren können. (1 P.)

**Tipp:** Sie können für die Bearbeitung der Aufgabe die Bibliothek *python-ecdsa* (einfach installierbar von `https://pypi.python.org/pypi/ecdsa`) nutzen. Diese unterstützt sämtliche Operationen mit den NIST-Kurven und ist kompatibel zu OpenSSL. Auch mathematische Operationen können direkt mit Hilfe von *python-ecdsa* durchgeführt werden. 7 P.

$3 + 5 + 5 + 7 =$ **20 Punkte**