Übungsaufgaben zur Vorlesung

IT-Sicherheit Grundlagen

Wolf Müller

Abgabetermin: Montag, 26.06.2023, 11:00 Uhr per Moodle
HU Berlin, Sommersemester 2023

Lab 3: Hashkollisionen (11 Punkte)

Im Rahmen ihres Online-Kurses in Cyber-Sicherheit brüsten sich Alice und Bob damit, dass sie mit ihrem erworbenen Wissen eine Möglichkeit schaffen können, Nachrichten über einen sicheren Kanal zu übertragen. Ihre Idee ist wie folgt:

Durch Anwendung des Diffie-Hellman-Schlüsselaustauschs etablieren sie ein gemeinsames Geheimnis. Dafür erzeugen beide sich je ein Schlüsselpaar und schicken den öffentlichen Schlüssel über das Nachrichtensystem der Kursplattform. Zur Authentizitätssicherung nehmen sie sich außerdem vor, über das Telefon einen Hash des Schlüssels durchzusagen. Sollte der Vergleich erfolgreich sein, benutzen sie das geheimsame Geheimnis zum Aufbau des sicheren Kanals.

Da Alice und Bob sich sicher sind, dass ihr Verfahren wasserdicht ist, fordern sie die anderen Studenten, also euch, heraus, ihre Kommunikation zu kompromittieren. Schnell ist die Idee eines Plans gefunden: Man kann sich in der benutzten Kurssoftware Accounts mit beliebigen Spitznamen erstellen, also könnte man vor dem Start des Schlüsselaustauschs als Alice eine Nachricht an Bob schicken und umgekehrt, um einen aktiven Man-in-the-middle-Angriff vorzubereiten.

Das Problem des über das Telefon verglichenen Hashs bleibt weiterhin bestehen. Es ist zum Glück allgemein bekannt, dass Alice und Bob beide eher faul sind und wir erwarten von ihnen, dass beide nach spätestens den ersten 12 durchgesagten Hexadezimalstellen die Lust verlieren.

Die Spezifikation und benutzten Kryptoprimitiven sind bekannt. Als Modul für den Diffie-Hellman-Schlüsselaustausch dient eine 2048-bit Primzahl aus dem Standard RFC 3526 mit dem Generator 2. Der gemeinsame Schlüssel wird als HEX-String (Kleinbuchstaben) mit `SHA3-224` gehasht und der resultierende Hexadezimalwert wiederum von links nach rechts vorgelesen.

Aufgabe 3.1: Theorie (3 Punkte)

Beschreiben Sie einen Angriff zur Kompromittierung des Schlüsselaustauschs, der auf Hashkollisionen basiert. Erläutern Sie zunächst, warum es Ihnen möglich ist, eine solche Kollision zu finden, auch wenn Sie nicht die Ressourcen haben, um 2^{48} Hashwerte zu berechnen. Geben Sie anschließend eine Abschätzung dafür, wie viel Zeit und wie viel Speicher Sie für Ihren Ansatz benötigen.

Aufgabe 3.2: Warm-Up (3 Punkte)

Schreiben Sie ein Programm `diffie`, das für die vorgegebenen privaten Schlüssel die korrekten Werte für die öffentlichen Schlüssel, das geteilte Geheimnis und dessen Hash berechnet. Mögliche Testwerte finden Sie in der Datei `test_keys.txt`.

Aufgabe 3.3: Kollision (5 Punkte)

Anschließend sollen Sie ein Programm schreiben, welches Ihren obigen Angriff implementiert. Hierbei sind mehrere Aspekte zu beachten. Ihr Programm soll für zwei Hexadezimalzahlen, die als Kommandozeilenargumente übergeben werden und die abgefangenen öffentlichen Schlüssel darstellen, zwei Hexadezimalzahlen ausgeben. Diese sind private Schlüssel, die in dieser Reihenfolge für die Kommunikation mit den jeweiligen Inhabern der öffentlichen Schlüsseln benutzt werden sollen.

Die Hexadezimalzahlen sind positive ganze Zahlen, die Ziffern größer als 9 werden durch Kleinbuchstaben dargestellt. (Eine eindeutige Kodierung ist hier wichtig, da die Hexadezimalziffern später als String von der Hashfunktion verarbeitet werden sollen.) Sollten Sie nur Kollisionen mit kleineren identischen Präfixen finden, geben Sie das Programm trotzdem ab und vermerken Sie das entsprechend. Die Korrektheit wird auf `gruenau2` geprüft (siehe Hinweise am Ende des Dokuments).

Aufgabe 3.4: Wettbewerb (🍓 🍓 🍓)

Zeigen Sie, was in Ihnen steckt! Um die Chance auf einen erfolgreichen Angriff zu maximieren, wird für eine zufällig generierte Eingabe gemessen, wie viele gemeinsame Ziffern Ihre Kollision für die ausgegebenen Schlüssel hat. Dafür hat Ihr Programm 30 Minuten (echte, nicht CPU-) Zeit. Um teilnahmeberechtigt zu sein, geben Sie ein Skript mit dem Namen `compete.sh` ab:

```
./compete.sh $A $B
```

Die letzten beiden Zeilen, die dieser Aufruf erzeugt, werden als Ihre Antwort für die privaten Schlüssel gewertet. Sie können andere Programme darin aufrufen, solange Sie dabei auf `gruenau2` bleiben und die Regeln der Rechnerbetriebsgruppe beachten. Gewonnen hat die längste Kollision (sofern mindestens 13 Stellen identisch sind).

Hinweise

- Die Abgabe erfolgt über Moodle.
- **Beispiel:** (mit vereinfachter Lösung, Kollision nur in 5 Hexadezimalstellen) Falls Ihr Programm oder Skript den Namen `collision` trägt. Um Ihre Resultate zu überprüfen, können Sie das vorgegebene `TestCollision.jar` benutzen.

```
$ ./collision 42 abba
56868
f3010
$ java -jar TestCollision.jar 42 abba 56868 f3010
```

  Alternativ können Sie auch das Mathematica-Script `TestCollision.m` mit Hervorhebung der Übereinstimmungen zur Überprüfung verwenden.

```
$ ./TestCollision.m 42 abba 56868 f3010
```

- **Abgabeformat:** Schreiben Sie bitten ein `bash`-Skript „`hashkollision.sh`“, welches Ihren Quelltext ggfs. kompiliert und das Binary oder Script so bereitstellt, dass sowohl `diffie` als auch `collision` in dem selben Verzeichnis danach bereit zur Ausführung sind. Packen Sie alle Ihre Quelltextdateien (Skript- oder Programmiersprache), das `bash`-Skript und eine `pdf`-Datei, die alle Ihre Antworten und Anmerkungen beinhaltet, in ein `tar`-Archiv. (Kompilierte Dateien brauchen Sie nicht mit abzugeben, da das Compilieren ihr Skript übernehmen soll.)
- **Aufrufkonvetionen für Aufgabe 3.2:** Das von Ihnen erstellte Programm wird mit `./diffie a b` aufgerufen, wobei `a` bzw. `b` für den jeweiligen privaten Schlüssel für Alice bzw. Bob in Hexadezimaldarstellung stehen. Der Aufruf soll für gegebene `a` und `b` diese und alle daraus abgeleiteten Größen im gleichen Format wie die einzelnen Beispielblöcke in der Datei `test_keys.txt` ausgeben:

```
a = 7
b = d
A = 80
B = 2000
K = 8000000000000000000000
H = e6d34422be44d87326cf2b51d2963bd2ba22a38301a410517000c4b2
```

- Sie können dafür eine Programmiersprache Ihrer Wahl verwenden. Weiterhin können Sie beliebige Bibliotheken verwenden (z.B. Cryptography, PyCryptodome, Crypto++, OpenSSL …).

- Sollten Sie Software / Bibliotheken / JARs benutzen, die auf `gruenau2` nicht für alle Nutzer installiert sind, so schließen Sie diese bitte in Ihre Abgabe mit ein oder automatisieren Sie die Schritte zur Installation mit normalen Nutzerrechten in Ihrem Bash-Skript, welches ggf. benötigte Dateien herunterlädt, compiliert oder konfiguriert.
- Starten Sie mit kurzen Bitlängen der Kollision zum Testen!
- Testen Sie Ihre Lösung nach frischer Installation in einem leeren Verzeichnis.
- Ihre abgegebenen Programme sollten unter `gruenau2` compilierbar bzw. ausführbar sein.
- Beginnen Sie bitte **frühzeitig** mit der Bearbeitung und **nutzen Sie intensiv die Übungstermine**, um potenziell auftretende Fragen (rechtzeitig) zu stellen.