htw saar

Studiengang Kommunikationsinformatik (Master)
Studiengang Praktische Informatik (Master)
Prof. Dr.–Ing. Damian Weber

Sicherheit und Kryptographie – Übung 2

Aufgabe 1 (Basisarithmetik)

Wählen Sie sich eine Programmiersprache, die mit beliebig großen Integerzahlen umgehen kann (Java, Python, C#, als C/C++–Fan die gmp–Library benutzen).

Implementieren Sie dort die effiziente Potenzierung durch wiederholtes Quadrieren. Einen Pseudocode können Sie aus der Vorlesungsmitschrift entwickeln oder unter

`https://en.wikipedia.org/wiki/Exponentiation_by_squaring`

nachlesen (Basisalgorithmus).

Übrigens kann man damit eine überraschend einfache Methode für

$$\frac{1}{a} \bmod p$$

schreiben.

Wegen

$$a^{p-1} \equiv 1 \bmod p$$

gilt

$$a^{p-2} \equiv \frac{1}{a} \bmod p.$$

Aufgabe 2 (Diffie–Hellman–Verfahren)

Implementieren Sie das Diffie–Hellman–Verfahren mit Hilfe zweier Prozesse, die das Diffie–Hellman–Protokoll über TCP/IP ausführen.

Für die Erzeugung sicherer Primzahlen finden Sie zunächst eine Primzahl q und danach prüfen Sie, ob $p = 2q + 1$ ebenfalls eine Primzahl ist. (Es ist q dann eine Sophie–Germain–Primzahl.) Falls p keine Primzahl, suchen Sie weiter nach dem nächsten q usw.

Als Primzahltest können Sie

$$a^{\frac{p-1}{2}} \equiv \pm 1 \bmod p$$

für 3 verschiedene a benutzen (siehe Vorlesung).

Nach dem Schlüsselaustausch soll der geheime Schlüssel benutzt werden, damit die Prozesse mehrere Nachrichten mit Hilfe des XTEA Blockchiffre austauschen. Die Nachrichten dürfen eine feste vorgegebene Länge besitzen.

`http://en.wikipedia.org/wiki/XTEA`

Es empfiehlt sich, für das Testen sehr kleine Primzahlen zu benutzen, etwa $q = 11$, $p = 23$.

Ihre Implementierung sollte es schaffen, mit mindestens 512–bit–Primzahlen q zurechtzukommen. Sie können die Primzahlerzeugung in einen eigenes Programm auslagern und das gefundene p den Diffie–Hellman–Prozessen als Parameter vorgeben.

Aufgabe 3 (Pollard–λ–Methode)

Implementieren Sie die Pollard–λ–Methode für einen Server und n Clients.

Gegeben sei ein dem Server bekanntes Problem zum diskreten Logarithmus

$$g^x \equiv h \bmod p.$$

Ein gestarteter Client erhält g, h, p vom Server. Er initialisiert durch zufällige Startwerte (Exponenten k, l) seine Zufallsfolge y_i und sendet beim Erreichen eines *distinguished point* die Werte y, k, l an den Server.

Ein *distinguished point* ist zu definieren (etwa alle unteren 20 Bits = 0). Ermitteln Sie hier experimentell günstige Werte betreffend Netzwerklast und Speicherplatzverbrauch.

Der Server soll Kollisionen unter den gesammelten y–Werten feststellen, die Lösung x ausrechnen und die Clients anweisen, die Rechnung zu beenden. Ein Client muss hierzu nicht aktiv kontaktiert werden. Der Server kann dem Client das Ende der Berechnung mitteilen, sobald dieser seinen nächsten *distinguished point* an den Server schickt.

Wählen Sie als Problemstellung Primzahlen p mit der Eigenschaft $q = (p-1)/2$ Primzahl und ein erzeugendes Element g mit

$$g^q \not\equiv 1 \bmod p.$$

Lösen Sie mit Hilfe der Pollard–Folge das Problem

$$(g^2)^x \equiv h^2 \bmod p$$

(ergibt $x \bmod q$) und den Fall

$$(g^q)^x \equiv h^q \bmod p$$

(ergibt $x \bmod 2$) durch Ausprobieren.

Ihr Code soll eine Spezialisierung des chinesischen Restsatzes implementieren, der die Teile

$$\begin{aligned} x &\equiv x_1 \bmod q \\ x &\equiv x_2 \bmod 2 \end{aligned}$$

zu einer Gesamtlösung zusammenfügt.