Lehrstuhl für Informatik I
— Informationsstrukturen und
wissensbasierte Systeme —
Universität Würzburg

Prof. Dr. H. Noltemeier

Würzburg, 8. Juli 1998

Übungen zur Vorlesung
Praktische Informatik II (SS98)

7. Übungsblatt

Aufgabe 24: (Programmieraufgabe)

In einem Public-Key-System wird die Authentizität der Kommunikationspartner dadurch gewährt, daß alle öffentlichen Schlüssel für alle Teilnehmer verfügbar sind. In der Praxis sind jedoch nicht alle öffentlichen Schlüssel von vornherein bekannt.

Angenommen, A (Alice) möchte mit B (Bob) kommunizieren und sichergehen, daß Bob die Nachricht M korrekt erhalten hat (und niemand außer ihm). Das folgende einfache Protokoll versucht dies zu gewährleisten:

Protokoll 1:

1. A verschlüsselt die Nachricht M an B mit dem öffentlichen Schlüssel P_B von B und fügt den eigenen öffentlichen Schlüssel als Absender an, d.h. gesendet wird $M_1 = e_{P_B}(M), P_A$.
2. B entschlüsselt den ersten Teil von M_1 mit seinem privaten Schlüssel und erhält M und P_A. Jetzt verschlüsselt B wiederum M mit dem erhaltenen öffentlichen Schlüssel P_A von A und hängt seinen eigenen öffentlichen Schlüssel P_B an: $M_2 = e_{P_A}(M), P_B$.
3. A kann nun M_2 entschlüsseln und überprüfen, ob B die Nachricht unverändert erhalten hat.

Auf den ersten Blick scheint dieses Protokoll sicher zu sein. Eine nähere Untersuchung ergibt aber Möglichkeiten für einen Eindringling C (Charles), der ebenfalls an die Nachricht M kommen und sich somit für B bzw. A ausgeben kann. Folgendes Szenario wäre denkbar (vorausgesetzt A und B halten sich an das Protokoll 1):

(1) C fängt die Nachricht M_1 ab und ersetzt den Schlüssel P_A durch seinen eigenen. B wird also $M_1' = e_{P_B}(M), P_C$ empfangen.
(2) B erhält M und antwortet mit $M_2' = e_{P_C}(M), P_B$ an C.
(3) C entschlüsselt nun M_2' mit seinem eigenen privaten Schlüssel und erhält M. Um A glauben zu machen, alles sei in Ordnung, sendet er noch $M_2 = e_{P_A}(M), P_B$ an A.

Um diesen Angriff zu vermeiden, läßt sich Protokoll 1 noch verbessern:

Protokoll 2:

1. A sendet $M_1 = e_{P_B}(e_{P_B}(M), P_A)$ an B.
2. B entschlüsselt M_1 mit seinem privaten Schlüssel und erhält M und A. Er antwortet mit $M_2 = e_{P_A}(e_{P_A}(M), P_B)$ an A.
3. A entschlüsselt M_2 und überprüft, ob B die Nachricht unverändert erhalten hat.

Bitte lösen Sie folgende Aufgaben:

a) Programmieren Sie A (Alice) als Client einer Client-Server-Anwendung in Java und führen Sie eine Authentifizierung entsprechend Protokoll 1 durch. Weitere technische Hinweise finden Sie weiter unten. (**3 Sonderpunkte**)

b) Implementieren Sie für A auch noch Protokoll 2. (**3 Sonderpunkte**)

c) Implementieren Sie einen Angreifer C für Protokoll 1. (**2 Sonderpunkte**)

d) Auch Protokoll 2 ist nicht sicher. Implementieren Sie einen Angreifer für Protokoll 2. (**2 Sonderpunkte**)

Technische Hinweise:

- Programmdateien finden Sie im Verzeichnis `~pi2/aufgabe24/`. Der Server B ist in der Klasse `AuthentiServer` enthalten und läßt sich mit `java AuthentiServer` starten. Der Quelltext ist *nicht* vorhanden. Ein Server läuft bereits auf dem Rechner „atlantis". Die Kommunikation erfolgt über den Port 1717.
- Die Kommunikation erfolgt auf Textbasis. Ein Klartext-Steuerwort teilt dem Server mit, was dieser machen soll, beispielsweise „`AUTHENTICATE_P1`" für Authentifizierungsprotokoll 1 oder „`BYE`" für das Schließen der Verbindung. Alle benutzbaren und vom Server erkennbaren Worte stehen mit Erklärung in der Datei `Authenti.java`. Als Trennzeichen zwischen Steuerworten und der nachfolgenden Kommunikation, sowie einzelner Teile einer Nachricht dient der Zeilenumbruch „`\n`".
- Die Verschlüsselung erfolgt über das RSA-Verfahren. Benutzen Sie bitte die Klasse `Cryptography` zum Ver- und Entschlüsseln. Eine verschlüsselte Nachricht besteht immer aus einer Zeile und enthält nur Ziffern und Buchstaben von A bis F.

Nützliche Klassen:

- `java.net.Socket` – zum Herstellen einer Verbindung mit dem Server.
- `java.io.PrintWriter`, `java.io.BufferedReader` – zur Kommunikation mit dem Server
- `Cryptography` – zum Ver- und Entschlüsseln von Strings mit dem RSA-Verfahren.

Abgabetermin: Die Programmieraufgabe ist am Montag, dem 27. Juli 1998, zu den CIP-Pool-Zeiten von $12^{00} - 18^{00}$ Uhr vorzuführen.

Hinweis: Maximal zwei Teilnehmer dürfen eine Programmieraufgabe bearbeiten.