Aufgabenblatt 3
IT-Security
Angewandte Informatik
WS 2018/19

Lernziele – 5 Punkte

- RSA-Verfahren

Dieses Aufgabenblatt baut auf die beiden vorherigen auf. Es wird das RSA-Verfahren realisiert und dazu passend die Langzahlbibliothek erweitert.

Aufgaben

Basierend auf den Routinen des letzten Aufgabenblatts wird nun das RSA-Verfahren realisiert. Dazu sind folgende Routinen erforderlich:

- func Keys generateRSAKeys(BigInt e, int size) – erzeugt zufällig ein size-langes Schlüsselpaar, also eine Struktur mit dem privaten und öffentlichen Schlüssel sowie p und q.
- func publicKey getPublicRSA(Keys k) – liefert den öffentlichen Schlüssel vom Schlüsselpaar k
- func secretKey getSecretRSA(Keys k) – liefert den geheimen Schlüssel vom Schlüsselpaar k, aber auch die beiden Werte für p und q
- func block encryptRSA(publicKey pk, block plain) – verschlüsselt mit dem (öffentlichen) Schlüssel den Datenblock plain
- func block decryptRSA(secretKey sk, block cipher) – entschlüsselt mit dem (geheimen) Schlüssel einen Datenblock cipher.

Dazu beachten Sie bitte folgende Hinweise:

- Bei der Erzeugung der Schlüssel müssen Bedingungen beachtet werden, die Sie prüfen müssen:
  - Die Primzahlen müssen unterschiedlich sein.
  - Die Primzahlen müssen sich in der Länge um max. 30 bit unterscheiden.
  - Das Produkt n muss der Größe size oder etwas größer entsprechen. Zum Testen sind 512 in Ordnung, richtig wird es ab 2048 bit. Das BSI empfiehlt 3072 bit.

    - Der unterste Wert für e ist 2**16+1, besser sind höhere Werte. e und Phi(n) dürfen keinen gemeinsamen Teiler haben, da sonst d nicht bestimmbar ist.
- Die Struktur Keys beinhaltet: p, q, e, d, n; publicKey: e, n; secretKey: p, q, d, n
- Bei en/decryptRSA() kann der erste Parameter vom Typ publicKey oder secretKey sein; dies können Sie per Pointer mit Typumwandlung oder besser mit einer union und einer den Typ beschreibenden Komponente realisieren, z.B. eine Bool-Variable, die als True die Public-Variante beschreibt.
- Bei jeder Anwendung des secretKey kann mit powerModPrim2() die Ent-/Verschlüsselung um den Faktor 3-4 optimiert werden, da die Primzahlen p und q bekannt sind. Leider eröffnet dies Seitenkanalangriffe, aber für diese Übung ist das erst einmal egal.
- Der Block block beinhaltet ein Array, dessen Elemente bitweise aneinander gereiht kleiner als das Produkt n ist, sowie die Angabe der Länge.
- Für die Bitkombinationen für block sind einige Werte verboten, u.a. 0, 1 und n-1. Prüfen Sie dies.

Gehen Sie bei der Entwicklung der Software immer schrittweise vor: Funktion für Funktion und immer sofort intensiv testen. Ersetzen Sie z.B. Ihre Random-Funktion gegen eine, die vorgegebene Zahlen generiert, so dass Sie die Schlüsselgenerierung nachvollziehbar testen können. Also, schreiben Sie ein Mock für Junit-Tests.

Zum Testen der RSA-Funktionen generieren Sie sich für den Block Extremwerte: alles 0, alles 1, 0.. aufsteigend bis zu einer vorgegebenen Zahl und per Zufall bzw. vorgegebene Werte. Dann wird immer ver- und dann entschlüsselt sowie das Ganze noch einmal umgekehrt. Es müssen am Ende immer dieselbe Werte heraus kommen.

Bestandteil dieses Aufgabenblatts sind noch folgende Langzahlroutinen:

- func BigInt gcd(BigInt x>0, y>0) – Euklid'scher Algorithmus. Dieser *kann* in der binären Form realisiert werden. Die langsameren Versionen sind auch in Ordnung.
- func BigIntgcd egcd(BigInt x>0, y>0) – Erweiterter Euklid'scher Algorithmus. Dieser *kann* in der binären Form realisiert werden.
- func BigInt powerModPrim2(BigInt x, y>0, p>1, q>1) – Potenzierung x^y mod p*q, wobei p und q zwei ungleiche Primzahlen sind. Hierbei wird mit dem chinesischen Restzahlensatz optimiert. Diese Routine ist für dieses Aufgabenblatt optional. Stattdessen können Sie diese Routine mit powerMod() simulieren.
- func bool gt(BigInt x, y) – liefert true, falls x > y ist

Der Returnwert vom Typ BigIntgcd ist eine Struktur bestehend aus dem ggT-Wert und den beiden Faktoren u und v der Linearkombination, deren Werte

auch negativ sein können. In diesem Falle muss das Modul noch vor dem Beenden der ggT()-Funktion addiert werden, so dass immer Werte zwischen 0 und Modul-1 geliefert werden.

Bitte beachten Sie folgendes Prinzip: **Es kommt auf Korrektheit und nicht auf Performanz an**.

Links

1. https://www.cryptool.org/de/ct1-downloads
2. https://cryptography.io/en/latest/development/test-vectors/
3. https://github.com/pyca/cryptography/tree/master/vectors/cryptography_vectors/asymmetric/RSA/pkcs-1v2-1d2-vec
4. http://cryptomanager.com/tv.html
5. http://csrc.nist.gov/groups/STM/cavp/documents/dss/186-2rsatestvectors.zip
6. https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/digital-signatures#rsavs

Abnahme

Zur Abnahme des gehören folgende Dateien:

- Testfälle mit Reports,
- Source-Code und
- Projektdateien, z.B. nbproject oder make-Dateien etc.

Alle Tests vom Dozenten müssen minimal benutzt werden; es können noch weitere Test durchgeführt werden. Die Vorführung der Lösung besteht in folgenden Ablauf: (1) Source-Code-Begutachtung, (2) Übersetzung und (3) Testlauf.

Wer mit vorgefertigten Langzahl-Bibliotheken arbeitet erhält 2 Punkte Abzug.

Auch diese Aufgabe kann per Email abgegeben werden, dann mit allen Sourcen, Übersetzungsdateien, z.B. make bzw. eclipse/netbeans-Projekte, einschließlich der Tests. Alles muss ohne weiteres auf dem Rechner des Dozenten laufen können (Java, C++, Linux, CentOS 6.10). Programmieren Sie also portabel. Im Falle von Python, Ruby oder Lua bitte noch die notwendige Laufzeitumgebung mitliefern.