Übungen zur Vorlesung Systemsicherheit

Asymmetrische Kryptographie

Tilo Müller, Reinhard Tartler, Michael Gernoth

Lehrstuhl Informatik 1 + 4

24. November 2010

Aufgabenblatt 2

Asymmetrische Kryptographie

Aufgabeblatt 2 | Oneway Functions

Sicherheit asymmetrischer Kryptographie basiert auf *Einwegfunktionen*.
Beispiele:

- **Multiplikation vs. Prim-Faktorisierung**: Die Multiplikation von Primzahlen ist einfach; die Primfaktorzerlegung ist schwer.
  - Wieviel ist $89 * 101$?
  - Was sind die Faktoren von 8927?
- **Modulare Exponentiation vs. Diskrete Logarithmen**: Die Berechnung der n-ten Potenz ist einfach; die Bestimmung des Exponenten n ist schwer.
  - Wieviel ist $2^4 \mod 11$?
  - Sei $2^n = 3 \mod 11$. Wie groß ist n?

$\rightarrow$ *Problem:*

Weder für die Primfaktorzerlegung, noch für die Berechnung diskreter Logarithmen, existieren Algorithmen die ganz ohne "Ausprobieren" auskommen.

Aufgabenblatt 2 | Asymmetric Cryptosystems

Basierend auf *Prim-Faktorisierung*:

- **RSA** (Rivest, Shamir and Adleman, 1977)
- Rabin (Michael O. Rabin, 1979)

Basierend auf *Diskreten Logarithmen*:

- **Diffie Hellman** (Diffie and Hellman, 1976)
- Elgamal (Tahir al-Dschamal, 1985)
- Eliptic Curves (Koblitz and Miller, 1985)

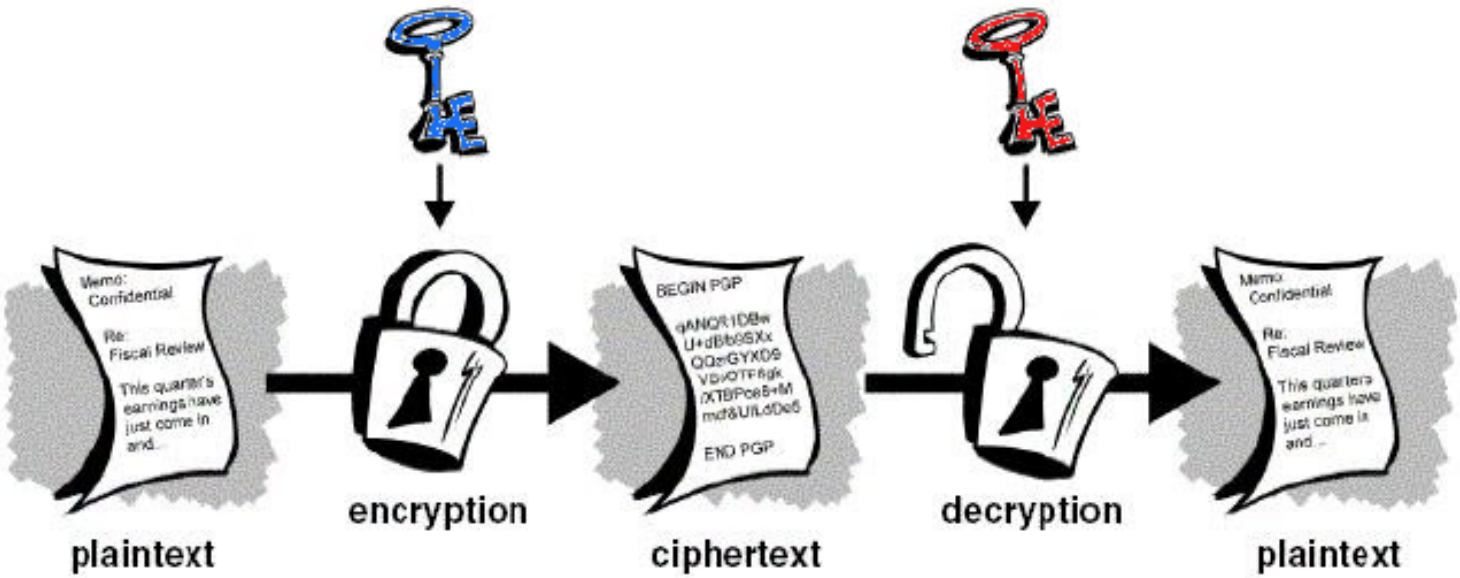
Sonstiges:

- Merkles Puzzle (Ralph Merkle, 1974)

Aufgabe 2.1
RSA

Aufgabe 2.1 | Public / private keys

- Public key E: jedem bekannt, verschlüsseln (**e**ncrypt)
- Private key D: nur dem Besitzer bekannt, entschlüsseln (**d**ecrypt)



Alice schickt Bob eine Nachricht:

- Alice: $c = E_{Bob}(m)$
- Bob: $m = D_{Bob}(c)$

Alice kann den von ihr verschlüsselten Text selbst nicht mehr entschlüsseln. D ist aus E praktisch nicht berechenbar.

Aufgabe 2.1 | RSA Applications

Email: **GPG/PGP**, **S/MIME**

Briefkasten-Metapher: Jeder kann einen Brief einwerfen (*public*), aber nur der Besitzer des Schlüssels kann ihn rausholen (*private*).

Weitere RSA Andwendungen:

- Server-Authentifizierung: Web (HTTPS, SSL/TLS)
- Client-Authentifizierung: Remote Shell (SSH)
- Key Exchange (hybride Verschlüsselung): Web, SSH, Mail, ...
- Signierung: Mail ("digitale Unterschrift"), Web (Zertifikate), ...

Aufgabe 2.1 | RSA Math

Schlüssel-Generierung:

- Wähle Primzahlen p und q
- Berechne RSA-Modulus $n = p * q$ (*Einwegfunktion*)
- Berechne $\phi(n) = (p-1) * (q-1)$
- Wähle e zu $\phi(n)$ teilerfremd
  $\rightarrow$ Public key (e, n)
- Berechne mult. Inverses d zu e bzgl. $\phi(n)$
  $\rightarrow$ Private key (d, n)

Anmerkung: n ist öffentlich; es kann aber nicht ohne weiteres auf p,q geschlossen werden (Primfaktorzerlegung), damit nicht auf $\phi(n)$ und damit nicht auf d.

Verschlüsselung (für $m < n$):

$$c = m^e \bmod n$$

Entschlüsselung:

$$m = c^d \bmod n$$

Aufgabe 2.1 | Square-and-Multiply

- Bei $x^n \bmod m$ kann der Zwischenschritt x^n sehr groß im Vergleich zum Endergebnis $\bmod m$ werden. Beispiel:

$$2^{19} \bmod 7 = 524288 \bmod 7 = 2$$

- Der naive Rechenweg führt zu Ineffizienz und (vermeidbaren) Integer-Overflows.
- $\rightarrow$ Lösung: **Square-and-Multiply** (auch: *Binary Exponentiation*):
  - Es gilt (weil $19_d = 10011_b$):

$$2^{19} = (((((2^2)^2)^2) * 2)^2) * 2$$

  - Also berechne:

$$2^{19} \bmod 7 = (((((2^2 \bmod 7)^2 \bmod 7)^2 \bmod 7) * 2)^2 \bmod 7) * 2 \bmod 7$$

Aufgabe 2.1 | Integer Overflows

overflow.c

```
int main() {
   int i;
   for (i=0; i<7; i++)
      printf(``%u'',0xfffffffd+i);
}
```

Ausgabe

```
4294967293
4294967294
4294967295
0
1
2
3
```

$\rightarrow$ Integers (32-bit, max. $0xffffffff = 4294967295$) sind zur Speicherung von großen RSA Primzahlen (einige hundert Stellen) nicht geeignet!

Aufgabe 2.1 | RSA / small numbers

Aufgabenstellung: Implementierung von RSA mit kleinen Zahlen

1. **Schlüssel-Generierung**:

```
./rsa_genkey <p> <q>
./rsa_genkey 17 19
Public  key (e,n) = (43,323)
Private key (d,n) = (67,323)
```

2. **Ver-/Entschlüsselung**:

```
./rsa_crypt enc <m> <e> <n>
./rsa_crypt enc 219 43 323
281
./rsa_crypt dec <m> <d> <n>
./rsa_crypt dec 281 67 323
219
```

3. **Schlüssel-Wiederherstellung**:

```
./rsa_crack <e> <n>
./rsa_crack 43 323
Private key (d,n) = (67,323)
```

Ende

Nächste Woche:

Aufgabe 2.2: Challenge / Response
(mit echten / großen RSA Schlüsseln)

Aufgabe 2.4: Zusatz