Computersystemsicherheit

Lösungsblatt Nr. 2

Hendrik Beckmann, Ibrahim Osman,
Patrick Elsen, Tiancheng Chen

Wintersemester 2018-2019
Technische Universität Darmstadt

Hausübung 1: Diffie-Hellman Schlüsselaustausch

Alice möchte mit Bob auf sichere Weise kommunizieren. In einer Einführungsveranstaltung zur Kryptographie erfährt Alice von dem Diffie-Hellman Schlüsselaustauschverfahren und möchte dieses dafür verwenden.

Finden Sie den vereinbarten Schlüssel.

Hier kann man ein kleines Ruby-Programm anwenden, um per Brute-Force an den geheimen Parameter x zu kommen.

```ruby
def compute_pub(g, p, sec)
  (g * sec) % p
end

def compute_key(g, p, pub, sec)
  (pub * sec) % p
end

p = 13
g = 6

pub_x = 2
pub_y = 3

sec_x = (1..p).select{|n| compute_pub(g, p,  n) == pub_x}
p sec_x
exit
```

```
18
19 k = compute_key(g, p, pub_y, sec_x)
20
21 puts "x␣is␣#{sec_x}␣and␣the␣key␣is␣#{k}."
```

Dieses Programm berechnet als geheimen Parameter $x = 5$ und den Key $k = 9$.

Alice entscheidet sich um ihren Rechenaufwand zu reduzieren eine additive Gruppe für den Diffie-Hellman-Schlüsselaustausch zu verwenden, d.h. statt $X = g^a$ verwendet sie jetzt $X = g \cdot a$. Begründen Sie, warum das keine gute Idee ist.

Der Diffie-Hellman Algorithmus besteht auf der (in annehmbarer Zeit) Unlösbarkeit des diskreten Logarithmus. Der diskrete Logarithmus ist die Umkehrung der diskreten Exponentialfunktion $b^x \mod n$. Also genau die FUnktion, die für die Schlüsselberechnung genutzt wird. Alice's Funktion ist im Vergleich sehr einfach lösbar und somit nicht sicher.

Hausübung 2: RSA

Hier kann man wieder ein kleines Ruby-Programm schreiben, um das Ergebnis zu berechnen.

```
1  n = 111791377
2  e = 3
3
4  transmitted = [106894622, 54756549, 49966544]
5
6  def encrypt(m, e, n)
7    (m ** e) % n
8  end
9
10 grades = [10, 13, 17, 20, 23, 27, 30, 33, 37, 40, 50]
11 alice = 174458
12
13 candidates = grades
14   .map{|g| [g, "#{alice}#{g}"]}
15   .map{|g, m| [g, encrypt(m.to_i, e, n)]}
16 match = candidates.select{|g, c| transmitted.include? c}.first
17
18 puts "Alice's␣ciphertext␣war␣#{match[1]}␣und␣sie␣hat␣eine␣#{match[0]}."
```

Dieses Programm gibt aus, dass Alice eine 1,7 hat, weil ihr Ciphertext 54756549 war.

Hausübung 3: RSA

Alice und Bob haben von dem RSA-Verschlüsselungsver- fahren gehört und möchten sich das jetzt gerne etwas genauer ansehen.

Gegeben ist ein RSA-Modulus $N = p \cdot q$ mit $N = 77$ und $p = 7, q = 11$ bekannt. Weiterhin ist der public key $pk = (N, e) = (77, 6)$ und Alice möchte mit diesem Schlüssel $m1 = 5$ und $m2 = 6$ verschlüsseln. Da Alice aber nicht mehr weiß wie das RSA-Verfahren funktioniert benötigt sie Ihre Hilfe. Berechnen Sie mit Hilfe des RSA-Verfahrens die Verschluüsselungen von m1 und m2.

Hier kann man ein kleines Ruby-Programm schreiben, um die Nachricht mit RSA zu verschlüsseln.

```ruby
def encrypt(m, e, n)
  (m ** e) % n
end

messages = [5, 6]
n = 77
e = 6

p messages.map{|m| encrypt(m, e, n)}
```

Dieses Programm gibt als Ergebnis $[71, 71]$ aus.

Vergleichen Sie die beiden Ciphertexte. Was fällt Ihnen aus? Warum entsteht dieses Ergebnis?

Die beiden Ciphertexte sind identisch. Das liegt wahrscheinlich an dem Exponent e, der mit p und q nicht kompatibel ist. Besser ist es, wenn man eine Primzahl als Exponent benutzt.

Bob besitzt den öffentlichen RSA-Schlüssel $(N, e) = (3127, 6)$ mit $N = pq$. Warum ist dies kein gültiger öffentlicher RSA-Schlüssel? Ändern Sie den ungültigen Anteil von Bobs öffentlichem Schlüssel minimal so ab, dass er gültig wird, und berechnen Sie anschließend den zugehörigen privaten Schlüssel d für ihn.

Für einen gültigen Schlüssel wählt man zwei Primzahlen p und q und berechnet $N = pq$. Dann wählt man ein e für das gilt :$ggT(e, \lambda(N)) = 1$. Dies ist bei Bobs' Schlüssel nicht gegeben, da der $ggT(6, 2016) = 2$ ist.

Man kann die 6 zu einer 5 machen, dessen $ggT(5, 3016)$ ist 1. Damit ist der Schlüssel gültig. Den privaten Schlüssel berechnet man mit dem erweiterten Euklidischen Algorithmus.