Grundlagen der Systemsoftware

Modul: InfB-GSS

Veranstaltung: 64-091

Utz Pöhlmann
4poehlma@informatik.uni-hamburg.de
6663579

Louis Kobras
4kobras@informatik.uni-hamburg.de
6658699

Marius Widmann
4widmann@informatik.uni-hamburg.de
6714203

6. Juli 2016

Zettel Nr. 6 (Ausgabe: 27. Juni 2016, Abgabe: 06. Juli 2016)

6.1 Zentrale Begriffe der Kryptographie

6.1.2 Schlüsselaustausch (Pflicht; 2 Punkte)

Für n Personen gibt es $\binom{n}{2}$ Paare.

symmetrisch: Bei n Personen muss jeder seinen Schlüssel an $n-1$ Personen weitergeben. Es gibt also n Schlüssel, die $n-1$ mal weitergegeben werden, also $n*(n-1) = n^2 - n$ Tauschaktionen[1]. Falls die Kommunikation in paarweise beide Richtungen stets mit dem gleichen Schlüssel stattfindet, bleiben $\frac{n^2-n}{2} = \binom{n}{2}$ Schlüsseltauschaktionen. So viele Schlüssel muss es auch geben.

asymmetrisch: Jede Person muss ein Schlüsselpaar generieren, einen `private key` und einen `public key`. Der Public Key wird per Broadcast oder als automatisierter Mailanhang verschickt, somit ensteht für jeden `private key`-Halter genau eine Aktion betreffs Schlüsselweitergabe. Dies macht bei n Personen n 'Tausch'-Aktionen, bei $2n$ generierten Schlüsseln.

6.1.3 Hybride Kryptosysteme (Pflicht; 3 Punkte)

Umstände: Hybride Kryptosysteme eignen sich bei großen Nachrichten, da symmetrische Verschlüsselung um mehrere Zehnerpotenzen schneller arbeiten als asymmetrische Verschlüsselungen. Durch die asymmetrische Verschlüsselung des vergleichsweise kurzen Keys (i.d.R. 128-256 Bit für sichere Verschlüsselungen) wird trotzdem die erhöhte Sicherheit gewährleistet, falls eine dritte Partei den übermittelten Schlüssel erhält (dieser kann durch die Asymmetrie nicht entschlüsselt werden außer vom legitimen Empfänger).

Detail-Verfahren:

1. sie verschlüsselt die Nachricht N symmetrisch mit dem von ihr erzeugtem Schlüssel S
2. S wird asymmetrisch mit Bobs public Key K_p^B verschlüsselt
3. Alice übermittelt $(K_p^B(S),\ S(N))$ an Bob
4. Bob entschlüsselt $K_p^B(S)$ mit K_s^B
5. Bob entschlüsselt $S(N)$ mit S symmetrisch

Nachricht: Folgt aus eben: $N' = \{(K_p^B(S), S(N))\}$, also die symmetrisch verschlüsselte ursprüngliche Nachricht sowie der asymmetrisch verschlüsselte Key.

6.2 Parkhaus

6.2.2 Sicherheitsanalyse (Pflicht; 4 Punkte)

Schwächen: Der links aufgedruckte Code ist für jeden Zweck jeweils immer identisch. (s. die Zahl darüber 32, 34, 36): Steht eine 32 darüber, wurde das Ticket vom Kino bearbeitet, bei 34 und bei 36 vom Händler. Der zweite Code von links ist immer identisch.

[1] Beweis siehe Anhang

Angreifermodell:

Rolle	Benutzer des Parkhauses, jedoch kein Kunde im Kino oder beim Einzelhändler
Verbreitung	kann nur sein eigenes Ticket einsehen und hebt auch keine Tickets zum Verlgeich auf oder macht Fotografien o.Ä.[1] Er probiert also nacheinander alle möglichen Barcodes durch. Ebenso kann er nicht selber Tickets editieren oder bearbeiten.
Verhalten	aktiv: liest veränderte Tickets am Automaten ein
Ressourcen	beschränkt: nicht genügend Rechenkapazität, um den Algorithmus zu knacken; Er hat außer über die Barcodescanner keinen Zugriff auf das System, insbesondere stehen ihm keine weiteren Schnittstellen zur Verfügung Wir gehen davon aus, dass die Ressourcen derart beschränkt sind, dass der Angreifer nicht in der Lage ist, durch Heuristiken, Kombinatorik, Inferenz oder Brute Force das Verfahren der Code-Generierung herauszufinden.

6.2.3 Umsetzung mit kryptographischen Techniken (Pflicht; 4 Punkte)

asymmetrische Verschlüsselung auf den Tickets:

Der Ticketdruckautomat verschlüsselt seine Daten mit einem öffentlichen (vorher ins System eingespeisten) Schlüssel S_p^T. Der Ticketbezahlautomat hat dazu den privaten Schlüssel S_s^T um die Daten wieder zu entschlüsseln. (Dies war wahrscheinlich schon bisher so.) Das System sollte Meta-Daten wie "Startdatum", "Startzeit", "Freiparkzeit", "Stundenpreis für den heutigen Tag", "allgemeiner Stundenpreis", "Ticket ist bezahlt", "Ticket ist bezahlt Datum" und "Ticket ist bezahlt Zeit" enthalten, die in nur einem Barcode verschlüsselt werden.

Die Ladenbesitzer erhalten nun jeweils beide Schlüssel und können so das Ticket einlesen und den neuen Code draufschreiben. Dies ist allerdings debattierbar, dar es die Möglichkeit für Insider-Angriffe erweitert. Insider sind allerdings von unserem Angreifermodell ohnehin nicht abgedeckt. Man kann ein Ausnutzen verhindern, indem man die internen Abläufe komplett versteckt und den Ladenbesitzern nur einen besseren 'Stempel' in die Hand drückt, der das Ticket markiert. Dann kann ein Insider zwar immer noch für Lau parken, wenn er einkaufen geht, aber er kann das System nicht weiter manipulieren, um Gewinn zu machen (was er könnte, wenn der Händler und das Kino die Ticket-Daten direkt bearbeiten könnten).

Die Daten werden dann verschlüsselt und wieder auf das Ticket geschrieben, wobei steganographisch ein Flag im Barcode versteckt werden kann, dass die Änderung legitim erfolgt ist.

Der Ticketbezahlautomat liest den im Code versteckten Flag und, sofern der Flag gesetzt ist,[2] rechnet nun den Gesamtpreis aus anhand der Daten ("Startdatum", "Startzeit", "Freiparkzeit", "Stundenpreis für den heutigen Tag" und "allgemeiner Stundenpreis") und fordert einen entsprechenden Betrag vom Kunden. Nun verschlüsselt der Ticketbezahlautomat alle vorherigen Daten ("Startdatum", "Startzeit", "Freiparkzeit", "Stundenpreis für den heutigen Tag" und "allgemeiner Stundenpreis") und seine Daten ("Ticket ist bezahlt", "Ticket ist bezahlt Datum" und "Ticket ist bezahlt Zeit") mit einem öffentlichen (vorher ins System eingespeisten) Schlüssel S_p^K. Er druckt die neuen Daten als zusätzlichen Code auf das Ticket. Die Schrankenautomat liest mit S_s^T und S_s^K (die beiden privaten Schlüssel) die beiden Datenmengen aus und öffnet die Schranke nur dann, wenn das Ticket vor weniger als 10 Minuten bezahlt wurde und die Daten in beiden entschlüsselten Datensätzen überein stimmen.

[1] Diese Einschätzung basiert darauf, dass das Angreifermodell denjenigen Angreifer darstelllen soll, gegen den das System noch geschützt ist. Sollte der Angreifer mehrere Tickets vorliegen haben, kann er herausfinden, was wir oben mit den Präfizes herausgefunden haben, und sich so den gewünschten Präfix vorne auf sein eigenes Ticket drucken. Dadurch wäre das System gebrochen. Dieser Angreifer ist vom Angreifermodell also nicht abgedeckt.

[2]
$$\text{Zustand des Flags: } \begin{cases} 1 & \text{, wenn im Laden oder im Kino editiert} \\ 0 & \text{, sofern keine Änderungen an den Daten vorgenommen wurden} \\ \emptyset & \text{, sonst (illegitime Änderung)} \end{cases}$$

6.3 Authentifizierungsprotokolle

6.3.2 Authentifikationssystem auf Basis indeterministischer symmetrischer Verschlüsselung (Pflicht; 2 Punkte)

verhindert:

- Replay-Angriffe

weiterhin möglich:

- Alle Arten von Man-in-the-Middle Angriffen (DoS, ARP-Spoofing, Sniffing...)
- Der Versuch, Daten abzufangen und zu entschlüsseln (erschwert durch verlängerte Nachricht und die Unklarheit, wo Zufallszahl aufhört / Nutzername anfängt)

6.3.3 Challenge-Response-Authentifizierung (Pflicht; 2 Punkte)

verhindert:

- Replay-Angriffe

weiterhin möglich:

- MitM
- Chosen-Plaintext-Angriffe sind weiterhin möglich, wenn der Angreifer den Verschlüsselungsalgorithmus kennt und genügend MACs mitgelesen hat um den Schlüssel zu erraten (Brute Force, Dictionary Attack)

6.5 RSA-Verfahren

6.5.2 Anwendung (Pflicht; 6 Punkte)

Folgender Python-Code berechnet die für RSA notwendigen Werte, liest die verschlüsselte Nachricht ein und speichert die entschlüsselte Nachricht in einem Textfile:

Code 1: $ python rsa.py msg.txt

```
1  from sys import 1argv  # to have access to console parameters
2
3
4  def extended_gcd(aa, bb):
5      """
6      SOURCE: http://rosettacode.org/wiki/Modular_inverse#Python
7      returns the modular multiplicative inverse of a number aa given to a
           base bb using the extended euclidean algorithm
8      :param aa: the number to invert
9      :param bb: the base
10     :return: the modular inverse to [aa]bb
11     """
12     lastremainder, remainder = abs(aa), abs(bb)
13     x, lastx, y, lasty = 0, 1, 1, 0
14     while remainder:
15         lastremainder, (quotient, remainder) = remainder,
               divmod(lastremainder, remainder)
16         x, lastx = lastx - quotient * x, x
17         y, lasty = lasty - quotient * y, y
18     return lastremainder, lastx * (-1 if aa < 0 else 1), lasty * (-1 if bb
           < 0 else 1)
19
20
```

```
21 def modinv(a, m):
22     """
23     SOURCE: http://rosettacode.org/wiki/Modular_inverse#Python
24     returns the modular inverse of a given number to a given base, if
           there is one
25     :param a: the number to invert
26     :param m: the base
27     :return: the modular inverse to [a]m, if there is one, ValueError else
28     """
29     g, x, y = extended_gcd(a, m)
30     if g != 1:
31         raise ValueError
32     return x % m
33
34
35 def getwords(file):
36     """
37     SOURCE:
           http://stackoverflow.com/questions/10264460/read-the-next-word-in-a-
           file-in-python (accepted answer)
38     This function was slightly modified compared to the source
39     """
40     with open(file) as wordfile:
41         wordgen = words(wordfile)
42         single_words = []
43         for word in wordgen:
44             if word.startswith(" "):
45                 word = word[1:]
46             if len(word) > 0 and word != '\n':
47                 single_words.append(word)
48         return single_words
49
50
51 def words(fileobj):
52     """
53     SOURCE:
           http://stackoverflow.com/questions/10264460/read-the-next-word-in-a-
           file-in-python (accepted answer)
54     This function was slightly modified compared to the source
55     """
56     for line in fileobj:
57         for word in line.split(","):
58             yield word
59
60
61 # read encrypted message into RAM (assuming the file contains nothing but
       the encrypted message)
62 words = getwords(argv[1])
63 # get encryption details
64 p = 271
65 q = 379
66 e = 47
67 phi = (p - 1) * (q - 1)
68
69 # calculate decryption exponent
70 d = modinv(e, phi)
71 # casts the results, which are 'long', as 'int'
72 res = map(lambda x: int(x),
73           # equivalent to: 'for all numbers in $words, raise them to $d
```

```
            and modulate them with $phi
74          map(lambda x: (int(x) ** d) % phi,
75              words))
76
77 # for debugging purposes, show the decrypted numbers in the console
78 print res
79
80 # store the result in a text file
81 out = open('(%s)-dec.txt' % argv[1], 'w')
82 for e in res:
83     out.write(str(e))
84     out.write(", ") # separate each number by a comma and a whitespace
85 out.close()
```

Die entschlüsselten Zahlen sehen wie folgt[1] aus (nach Code):

Code 2: (msg.txt)-dec.txt

```
1  80919, 49559, 79877, 2603, 6073, 40095, 43740, 79877, 6073, 7469, 46204,
2  46204, 88109, 67068, 49708, 85888, 49559, 71999, 49559, 2603, 6073, 71999,
3  43740, 16540, 40095, 6073, 83029, 28431, 49708, 39320, 79877, 16540, 40095,
4  79877, 6073, 81236, 98756, 79877, 37927, 79877, 16540, 6073, 94151, 43740,
5  58492, 98756, 98797, 43740, 39320, 43740, 6073, 72071, 16540, 39320, 2603,
6  79877, 43740, 83029, 79877, 2603, 37927, 28431, 40095, 79877, 49708, 49708,
7  79877, 61965, 6073, 46204, 58492, 98756, 49559, 98797, 20896, 20896, 43740,
8  79877, 49708, 79877, 61965, 6073, 1820, 85888, 43740, 16540, 29889, 28431,
9  94151, 6073, 81236, 85888, 29889, 49708, 79877, 71999, 61965, 6073, 40095,
10 43740, 79877, 6073, 43447, 96957, 16540, 88109, 22259, 46204, 43740, 58492,
11 98756, 79877, 2603, 98756, 79877, 43740, 98797, 6073, 163, 28431, 16540,
12 6073, 52488, 85888, 71999, 71999, 94151, 28431, 79877, 2603, 98797, 79877,
13 2603, 16540, 6073, 49559, 16540, 40095, 6073, 40095, 85888, 20896, 49559,
14 39320, 79877, 98756, 28431, 79877, 2603, 43740, 39320, 79877, 6073, 72071,
15 16540, 39320, 2603, 43740, 83029, 83029, 79877, 61965, 6073, 55061, 49559,
16 39320, 85888, 16540, 39320, 71999, 88109, 6073, 49559, 16540, 40095, 6073,
17 55061, 49559, 39320, 2603, 43740, 83029, 83029, 71999, 45927, 28431, 16540,
18 98797, 2603, 28431, 49708, 49708, 79877, 61965, 6073, 81236, 43740, 37927,
19 43740, 16540, 39320, 88109, 72071, 98797, 98797, 85888, 58492, 45927, 6073,
20 49559, 16540, 40095, 6073, 52488, 28431, 94151, 79877, 2603, 88109, 72071,
21 16540, 85888, 49708, 82007, 71999, 43740, 71999, 61965, 6073, 78732, 43740,
22 28431, 37927, 79877, 98797, 2603, 43740, 71999, 58492, 98756, 79877, 6073,
23 26188, 79877, 2603, 83029, 85888, 98756, 2603, 79877, 16540, 61965, 6073,
24 7469, 2603, 49559, 16540, 40095, 49708, 85888, 39320, 79877, 16540, 6073,
25 40095, 79877, 2603, 6073, 67068, 2603, 82007, 92583, 98797, 28431, 39320,
26 2603, 85888, 92583, 98756, 43740, 79877, 61965, 6073, 40095, 85888, 71999,
27 6073, 1820, 46204, 72071, 88109, 26188, 79877, 2603, 83029, 85888, 98756,
28 2603, 79877, 16540, 61965, 6073, 72071, 49559, 98797, 98756, 79877, 16540,
29 98797, 43740, 83029, 43740, 45927, 85888, 98797, 43740, 28431, 16540,
       71999,
30 92583, 2603, 28431, 98797, 28431, 45927, 28431, 49708, 49708, 79877, 6073,
31 49559, 16540, 40095, 6073, 16540, 85888, 98797, 49559, 79877, 2603, 49708,
32 43740, 58492, 98756, 6073, 85888, 49708, 49708, 79877, 6073, 85888, 16540,
33 40095, 79877, 2603, 79877, 16540, 6073, 24057, 16540, 98756, 85888, 49708,
34 98797, 79877, 61965, 6073, 40095, 43740, 79877, 6073, 94151, 43740, 2603,
35 6073, 43740, 16540, 6073, 40095, 79877, 2603, 6073, 96957, 79877, 29889,
36 49559, 16540, 39320, 6073, 49559, 16540, 40095, 6073, 40095, 79877, 2603,
37 6073, 26188, 28431, 2603, 49708, 79877, 71999, 49559, 16540, 39320, 6073,
38 29889, 79877, 98756, 85888, 16540, 40095, 79877, 49708, 98797, 6073, 98756,
39 85888, 29889, 79877, 16540, 6073, 43740, 88109, 22259,
```

[1] *Anmerkung:* Es wurden manuell Zeilenumbrüche eingefügt, die in der eigentlichen Ausgabedatei nicht enthalten waren

Anhang: Beweis zu 1.2.1

Behauptung:

$$\forall n \geq 2 : \binom{n}{2} = \frac{n^2 - n}{2}$$

Beweis durch vollständige Induktion.

Induktionsanfang:

$$\binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n \cdot (n-1)}{2} = \frac{n^2 - n}{2}$$

Sei $n = 2$:

$$\frac{2^2 - 2}{2} = \frac{4-2}{2} = \frac{2}{2} = 1 = \frac{2!}{2!0!} = \frac{2!}{2!(2-2)!} = \binom{2}{2} \quad \square$$

Induktionsannahme: Sei n beliebig, aber fest, dann gilt OBdA:

$$\binom{n}{2} = \frac{n^2 - n}{2} \Leftrightarrow \binom{n+1}{2} = \frac{(n+1)^2 - (n+1)}{2}$$

Induktionsschritt:

$$\begin{aligned}
\tbinom{n+1}{2} &= \tfrac{(n+1)!}{(n+1-2)!2!} = \tfrac{(n+1)!}{(n-1)!2!} \\
&= \tfrac{n \cdot n+1}{2} = \tfrac{n^2+n}{2} \\
&= \tfrac{n^2+n+n-n+1-1}{2} \\
&= \tfrac{n^2+2n-n+1-1}{2} \\
&= \tfrac{n^2+2n+1-n-1}{2} \\
&= \tfrac{(n+1)^2-(n+1)}{2} \quad \square
\end{aligned}$$