

RSA Fallen

Andreas Klein

20. Mai 2008

Das RSA Verfahren ist wohl eines der bekanntesten modernen Verschlüsselungsverfahren. Es ist außerordentlich simpel und daher wird es gerne in Einführungen (Schülerprojektwochen, etc.) genommen. Der Vorteil ist, dass jeder das Verfahren verstehen kann, und der Nachteil ist, dass jeder glaubt, das Verfahren verstanden zu haben. Für den Ungeübten warten bei der Implementierung eine Reihe von Fallstricken, von denen ich einige der interessanteren in den folgenden Aufgaben vorstellen möchte. Ich habe mich dabei auf den mathematischen Teil beschränkt, Implementierungsdetails wie Seitenkanalangriffe bleiben außen vor.

Die Aufgaben 1, 6, 7 und 8 sind verhältnismäßig einfach und eignen sich gut für einen Kryptographiekurs. Die Aufgaben 2, 3 und 4 bauen jeweils aufeinander auf und waren für mich der Anlass für diesen Artikel. Ihre Schwierigkeit liegt deutlich über dem Niveau eines normalen Kryptographiekurses. Aufgabe 5 ist als Wiener-Attacke bekannt.

Aufgabe 1

Anton benutzt RSA. Die Primzahlen für den öffentlichen Schlüssel hat Anton mit einem Computer Algebra System erzeugt.

Anton's Code lautete:

```
p1:=next_prime(377^220);  
p2:=next_prime(p1);  
n:= p1*p2;
```

Anton ist sehr überrascht als Berta schon nach kurzer Zeit Anton's RSA gebrochen hat. Wie hat Berta es gemacht?

Aufgabe 2

Anton möchte es beim nächsten Mal besser machen er wählt:

```
p1:=next_prime(313^244);  
p2:=next_prime(377^197);  
n:= p1*p2;
```

Berta braucht diesmal nicht sehr viel länger, um Anton's RSA zu knacken. Erkläre warum Anton's Wahl schlecht ist. Schreibe ein Programm das eine Anton-artige Zahl schnell faktorisiert.

Überprüfen Sie ihr Programm an der Faktoriesierung der folgenden Zahl.

59069332248171520688171161883195183350615044967582854299919464800
85594854093700371063753683997998857731909411581133586151161586497
40642686216913431671863462372730668331098042535928918377076912522
59835347055435658423214846329462042654732967321646282069622582858
15909253718543115193353129347079401758633333168435578062586891208
48731371424244216986302545172486945263488472258910379269283501902
05330371242522164080157948469899093371226970493041532864310767331
92974176395385334112313806795307410046986763097333000822709303790
67673998150479491439006329580969713546382026540541827685618979394
98461014370439779956199247528602498057824905655382269115431066594
08503969958550097100646164739417901076525899135682156349082492394
71610738235116089522933643874899621440000862847473471478110039607
57057018891187912823975206282868959289918652880080441466462651384
93813761695241946352167911165345860748857942649550778364639179206
45535782324763820161121603057792807052313426561132484224465935066
40204666817631544410419248118440012183279

Aufgabe 3:

Hier ein weiteres Problem, im Vergleich zu Aufgabe 2 muss man noch eine kleine zusätzliche Idee haben.

13263538599992691454338207327536684677180913323064424721097817476
82369371630580029448147127889856938144686753515456806998850502295
00435820189461201532005510125379172049188915903504684664147463695
83530293731428541385045566193418172823435911226243406050228024628
17117841415744009901475541650099472863154093765727864516060836172
57390045657170228903752490634487328178347360900344165126370563893
01384493102260587670464475286798078154274258161255293597876282214
13498101958547743750689165296571326594784147486518480859199043759
06186030491445909646106049989773874654827558242892456076668118774
89928009013640693552346318468696447902906688100289655790812805046
13875417634209946208151642499597520626579826578003058294100382635
88746022052446542555913143627326868213249992868940775419531212690
42138855725229632322427763035170190666837940213914119470857663227
00968836646599044233932617167161992610094551457436351877298171711
60302174742727565212154832579909038379383691030510925321124581516
24840338455184790877825171127311266616663632773788535630325246399
6963250905178210184267685312651731680334361322480069281043

Aufgabe 4:

Das folgende Problem basiert auf einem Fehler in einem Artikel, der RSA nur am Rande streift. (Ich hatte den Artikel vor der Veröffentlichung zur Begutachtung, in der veröffentlichten Version ist der Fehler behoben.)

Diese Aufgabe ist etwas schwerer als Aufgabe 3. (Ich habe die Aufgaben 2 und 3 entwickelt als ich nach der Lösung für dieses Problem suchte.)

34586201178041039787601387318820244721413414799352772803811244019

73224955092044702429353237831653852899308236572322787110648666243
96026528162520366869362889027847958124572048492902238930631589988
18482613361927436819233997778280771773531258289723019632876983832
12091226093774624173185088260156978955120655490200139405943091853
68543866142627334637083672548438918468553728549604911185025083035
79573898154690867854070287579231284091050661417856347622431237601
15811585790702271595966128978378655416979085147943852468787681501
37301974631020508238553705918725815721502791333002836671068519210
34742753411328100789876464417558967790656991036686872765761627509
08444563452708527467904785212450349796502307124874120148822459134
87037099190364226621719562272089742807613919578158325332361890921
67781696601196878110414271326709308070464191649369178098379415232
82561358948982532649576411015604086318734701088378495775346687820
72859228734103104199834644775463040487237495558910707470179588781
88492388101839044697736088521548202988328397688000816364073389094
41334630877093200267141833111604005645556422780165182446621858830
42717413

Aufgabe 5:

Nehmen wir an, dass Antons Primzahlen die Form $p_1 = g_1 + \epsilon_1$ und $p_2 = g_2 + \epsilon_2$ haben mit $\epsilon_i = o(\sqrt{g_i})$. Weiter nehmen wir an, dass Berta g_1 und g_2 raten kann. (Es gibt mehrere Möglichkeiten, wie Berta an g_1 und g_2 kommen kann. Eine ist, dass g_1 und g_2 glatte Zahlen sind wie in den Aufgaben 2 bis 4. Oder Anton hat p_1 und p_2 dicht beieinander gewählt, wie in Aufgabe 1, dicht heißt hier $|p_2 - p_1| \ll \sqrt{p_1}$. Oder der von Anton verwendete Zufallsgenerator könnte Schwächen haben.)

Wie kann Berta unter diesen Umständen Antons RSA knacken?

Aufgabe 6:

Anton ist wirklich vom Pech verfolgt, nun hat er ein gutes Produkt n erzeugt, aber diesmal wählt er einen zu kleinen geheimen Exponenten d . Klein bedeutet in diesem Fall $d < \sqrt{n}$, dass ist immer noch so groß, dass ausprobieren chancenlos ist.

Trotzdem knackt Berta Antons RSA in wenigen Minuten.

Aufgabe 7:

Hier ein weiteres Beispiel:

Anton und sein Freund Charlie benutzen den öffentlichen RSA-Schlüssel (e_1, n) und (e_2, n) , d.h. beide benutzen denselben Modul n . Wahrscheinlich waren Sie nach den Aufgaben 1 bis 4 froh zwei gute Primzahlen gefunden zu haben und dachten Sie könnten es sich einfach machen.

Es gelte $\gcd(e_1, e_2) = 1$. Daniel schickt Anton und Charlie dieselbe Nachricht. Ein Glück für Berta. Sie fängt die Geheimtexte, ab und kommt ganz schnell an den Klartext.

Aufgabe 8:

Anton, Charlie und Daniel haben aus der letzten Aufgabe gelernt, dass Sie verschiedene Moduln benutzen müssen. Um eine schnelle Verschlüsselung zu

gewährleisten wählen Sie den öffentlichen Exponenten $e = 3$ (Das ist an sich nicht falsch).

Eva schick allen dreien, die selbe Nachricht. Was kann Berta diesmal erreichen.

Aufgabe 9:

Anton glaubt jetzt das RSA Verfahren verstanden zu haben und setzt es an seiner Universität ein um Kosten Abrechnungen verschlüsselt zu übertragen. Eine Abrechnung besteht aus dem Datum der Rechnung (Tag, Monat und Jahr je zweistellig geschrieben), der internen Nummer der Rechnung (fünfstellig), der Nummer der Kostenstelle (dreistellig) und dem Rechnungsbetrag in Euro (sechs stellen vor dem Komma, zwei nach dem Komma). Anton schreibt diese Zahlen einfach hintereinander und verschlüsselt die so erhaltene Zahl mit RSA.

Beispiel: Am 04.07.2008 wird die Rechnung Nr. 731 von der Kostenstelle 117 verbucht. Der Betrag ist 111.33 €. Die Eingabe für den RSA Algorithmus ist dann 0404080073111700011133.

Auch dieses System ist vor Berta nicht sicher.

Zusätzlich zu den oben dargestellten Fehlern gibt es noch weitere Fallstricke bei der Implementierung von RSA. Man kann eine Zufallsquelle mit zu niedriger Entropie wählen (dies ist vor kurzem der Linux Distribution Debian passiert) oder man wird Opfer eines Timing-Attacke (ist schon einmal bei Openssl passiert).

Wie heißt es so schön:

Die Fehler sind da, sie brauchen nur noch gemacht zu werden! (Tartakower)

Lösungen

Lösung 1:

Neben anderen Problemen, die in den Aufgaben 2 bis 4 noch deutlicher ausgeführt werden, hat Antons Ansatz einen Riesenfehler. Die Primzahlen p_1 , p_2 mit $n = p_1 p_2$ dürfen beim RSA nicht nah bei einander liegen.

Berta schreibt $n = x^2 - y^2 = (x + y)(x - y)$. Bei Anton's ist $y = \frac{p_2 - p_1}{2}$ sehr klein und wird durch einfaches Ausprobieren gefunden.

Modernen Faktorisierungsmethoden wie das Zahlkörpersieb gehen übrigens von der Darstellung $n = x^2 - y^2$ aus. Es gibt eine gute Chance, das bereits eine Standardimplementierung von `Factor` Anton's n schafft.

In Aufgabe 5 werden wir sehen, dass n auch dann noch effektiv faktorisiert werden kann, wenn nur $|p_2 - p_1| < \sqrt{p_1}$ gilt.

Lösung 2:

Zunächst einmal ist gegen das Verfahren einzuwenden, dass auf diese Weise zu wenige Primzahlen erzeugt werden können. Hält man sich genau an das angegebene Beispiel sind es nur etwa 250000 mögliche Primzahlen (Basis < 1000 , Exponent < 250). Auch wenn Anton etwas komplizierte Dinge wie

```
p1:=next_prime(313^244+5555);
```

nimmt, wird man kaum eine Entropie vom mehr als 30 Bit erreichen. Man kann also nur von einem unsicheren Verfahren sprechen.

Es kommt aber noch schlimmer, denn es gibt denn LLL-Algorithmus.

Anton-artige Primzahlen können in der Form $g + \epsilon$ geschrieben werden, wobei g nur kleine Primfaktoren enthält und $\epsilon \ll g$ ist. Nimmt man zwei dieser Primzahlen $p' = g_1 + \epsilon_1$ und $p'' = g_2 + \epsilon_2$, so ist ihr Produkt

$$n = p'p'' = g_1g_2 + g_1\epsilon_2 + g_2\epsilon_1 + \epsilon_1\epsilon_2 \approx g_1g_2 .$$

Der Angriff zielt darauf ab, die glatte Zahl $g = g_1g_2$ in der Nähe von n zu rekonstruieren. Wir schreiben $g = p_1^{e_1} \cdot \dots \cdot p_k^{e_k}$ und erhalten:

$$p_1^{e_1} \cdot \dots \cdot p_k^{e_k} \approx n$$

$$e_1 \log(p_1) + \dots + e_k \log(p_k) \approx \log(n)$$

Nun kann man bereits den Zusammenhang mit Gittern und dem LLL-Algorithmus erkennen. Wir wählen eine große Zahl N (etwas kleiner als \sqrt{n}) und bauen ein Gitter mit der Basismatrix

$$\begin{pmatrix} 1 & & 0 & \lfloor \log(p_1)N \rfloor \\ & \ddots & & \vdots \\ 0 & & 1 & \lfloor \log(p_k)N \rfloor \\ 0 & \dots & 0 & \lfloor \log(n)N \rfloor \end{pmatrix}$$

Dann suchen wir eine LLL-Basis für dieses Gitter. Diese besteht aus Vektoren mit kleinem Gewicht. Der Vektor

$$(e_1, \dots, e_k, e_1 \lfloor \log(p_1)N \rfloor + \dots + \lfloor e_k \log(p_k)N \rfloor - \lfloor \log(n)N \rfloor)$$

hat kleines Gewicht und ist mit hoher Wahrscheinlichkeit der erste Vektor der LLL-Basis. (Wir müssen nur k und N richtig wählen.)

Für das angegebene Beispiel wähle ich $N = 10^{400}$ und die ersten 20 Primzahlen $p_1 = 2$ bis $p_{20} = 71$. Der LLL-Algorithmus findet das Basiselement.

$$(0, 0, 0, 0, 0, -387, 0, 0, 0, -210, 0, -177, 0, 0, 0, 0, 0, 0, 0, 491)$$

Also ist $13^{387} \cdot 29^{210} \cdot 37^{177} \approx n$.

Da $387 = 210 + 177$ spricht viel dafür, dass Anton von den glatten Zahlen $g_1 = (13 \cdot 29)^{210}$ und $g_2 = (13 \cdot 37)^{177}$ ausgegangen ist.

In der Tat gilt:

```
p1:=next_prime(481^177);
p2:=next_prime(377^210);
n:=p1*p2;
```

Lösung 3:

Zunächst starten wir wie in Aufgabe 2 und bestimmen mit dem LLL-Algorithmus eine glatte Zahl in der Nähe von n .

Wir finden $7^{427} \cdot 53^{427} \approx n$. Vieles spricht dafür, dass Anton $g_1 = 7^{a_1} \cdot 53^{b_1}$ und $g_2 = 7^{a_2} \cdot 53^{b_2}$ gewählt hat. Allerdings ist es etwas mühsam alle möglichen Werte von a_1 und b_1 zu probieren.

Wir raten, dass der ggT(g_1, g_2) zumindest $(7 \cdot 53)^5$ enthält. In diesem Fall ist $n = (g_1 + \epsilon_1)(g_2 + \epsilon_2) \bmod (13 \cdot 37)^{177} = \epsilon_1 \epsilon_2$. Da $\epsilon_1 \epsilon_2$ sehr klein ist wird uns das auffallen.

Wir überprüfen unsere Vermutung.

```
n mod (7*53)^4;
  131340
n mod (7*53)^5;
  131340
n mod (7*53)^6;
  131340
```

Es spricht also vieles dafür, dass $\epsilon_1 \epsilon_2 = 131340$ ist. Da $131340 = 2^2 \cdot 3 \cdot 5 \cdot 11 \cdot 199$ ist und sowohl ϵ_1 als auch ϵ_2 gerade sein müssen bleiben nur $2^4 = 16$ mögliche Werte für (ϵ_1, ϵ_2) . (Diese Zahl reduziert sich nochmal um die Hälfte, wenn wir die Symmetrie berücksichtigen.)

Wir raten also ϵ_1 und ϵ_2 und lösen das quadratische Gleichungssystem

$$\begin{aligned} g_1 g_2 &= g = 7^{427} \cdot 53^{427} \\ (g_1 + \epsilon_1)(g_2 + \epsilon_2) &= n \end{aligned}$$

Dies läuft auf eine einfache quadratische Gleichung für g_1 hinaus. Das Lösen geht also sehr schnell. Für $\epsilon_1 = 330$ und $\epsilon_2 = 398$ werden wir fündig. Die Lösung des Gleichungssystem liefert die ganzzahligen Lösungen $g_1 = 7^{210} \cdot 53^{216}$ und $g_2 = 7^{217} \cdot 53^{211}$.

Damit ist die Faktorisierung $n = (7^{210} \cdot 53^{216} + 330)(7^{217} \cdot 53^{211} + 398)$ gefunden.

Lösung 4:

Zunächst starten wir wieder mit dem LLL-Algorithmus, um die glatte Zahl in der Nähe von n zu finden. Das Ergebnis ist $n \approx (7 \cdot 53)^{433}$.

Beim Versuch wie in Aufgabe 3 fortzufahren wurde ich enttäuscht. Man findet keinen Wert für $\epsilon_1 \epsilon_2$.

```
n mod (7*53)^5;
  5251693771116
n mod (7*53)^6;
  2521494664775774
n mod (7*53)^7;
  709185138652986165
```

Danach versuchte ich, ob $\epsilon_1 \epsilon_2 < 0$ ist. Zunächst wieder kein Erfolg:

```
-n mod (7*53)^5;
  1776917879735
```

```
-n mod (7*53)^6;
  86120257689947
-n mod (7*53)^7;
  258239997581796326
```

Es dauerte eine Weile bis ich auf die Idee kam, wesentlich größer Produkte von $\epsilon_1\epsilon_2$ in Betracht zu ziehen:

```
-n mod (7*53)^49;
  78922592492039161826532978098784043042681682112157512
-n mod (7*53)^50;
  78922592492039161826532978098784043042681682112157512
-n mod (7*53)^51;
  78922592492039161826532978098784043042681682112157512
```

Also ist

$$\epsilon_1\epsilon_2 = -78922592492039161826532978098784043042681682112157512$$

Etwas Probieren überzeugte mich davon, dass $\text{ggT}(g_1, g_2) = (7 \cdot 53)^{212}$ ist.

```
-n mod (7*53)^212;
  78922592492039161826532978098784043042681682112157512
-n mod (7*53)^212*7;
  552458147444274132785730846691488301298771774785102584
-n mod (7*53)^212*53;
  4182897402078075576806247839235554281262129151944348136
```

Ich vermutete $g_1 = (7 \cdot 53)^{212}$ und $g_2 = (7 \cdot 53)^{221}$

Also suchen wir die Lösung von:

$$\begin{aligned} ((7 \cdot 53)^{212} + \epsilon_1)((7 \cdot 53)^{221} + \epsilon_2) &= n \\ -78922592492039161826532978098784043042681682112157512 &= \epsilon_1\epsilon_2 \end{aligned}$$

Dieses System hat die ganzzahlige Lösung:

$$\begin{aligned} \epsilon_1 &= 8883838837776377363653637188 \\ \epsilon_2 &= -8883838837377363653636274 \end{aligned}$$

Damit ist die Faktorisierung gefunden.

Lösung 5:

Bertas Ziel ist es $\epsilon_1\epsilon_2$ zu bestimmen. Dazu erstellt sie das Gitter

$$\begin{pmatrix} N & 0 & g_1 \\ 0 & N & g_2 \\ 0 & 0 & n - g_1g_2 \end{pmatrix}$$

wobei N eine Zahl in der geschätzten Größenordnung von ϵ_1, ϵ_2 ist. (Falls $g_2 > g_1$ ist, kann man ϵ_2 schätzen als $(n - g_1 g_2)/g_2 = \epsilon_2 + \epsilon_1(g_1 + \epsilon_2)/g_2$.)

Das Gitter enthält den Vektor $(N\epsilon_2, N\epsilon_1, -\epsilon_1\epsilon_2)$ und falls ϵ_1 und ϵ_2 klein genug sind wird dieser Vektor in der LLL-Basis auf tauchen. (Kleines Problem, wenn $a_1 g_1 = a_2 g_2$ für kleine a_1 und a_2 gilt, z.B. weil g_2 ein Vielfaches von g_1 ist, ist der erste Vektor der LLL-Basis $(Na_1, -Na_2, 0)$ und der zweite Vektor ist $(N\epsilon_2, N\epsilon_1, -\epsilon_1\epsilon_2) + k(Na_1, -Na_2, 0)$ für ein k .)

Auf diese Weise lernt Berta $\epsilon_1\epsilon_2$ und kann wie in Aufgabe 4 fortfahren.

Lösung 6:

Wir wissen $ed \equiv 1 \pmod{\phi(n)}$. Also gibt es ein k mit $ed = 1 + k\phi(n)$. Wegen $n \approx \phi(n)$ erhalten wir

$$ed \approx kn$$

$$\frac{e}{n} \approx \frac{k}{d}$$

Der Bruch e/n ist bekannt. Ist nun d klein im Vergleich zu n , so ist Näherung $\frac{e}{n} \approx \frac{k}{d}$, so gut dass k/d als Näherungsbruch in der Kettenbruchentwicklung von e/n auftauchen muss.

Lösung 7:

Berta bestimmt mit dem erweiterten euklidischen Algorithmus zwei ganze Zahlen a und b mit $ae_1 + be_2 = 1$. Sie kennt $c_1 \equiv m^{e_1} \pmod{n}$ und $c_2 \equiv m^{e_2} \pmod{n}$. Die Nachricht m kann sie daher als $m \equiv c_1^a c_2^b \pmod{n}$ berechnen.

Lösung 8:

Berta kennt $m^3 \pmod{n_1}, m^3 \pmod{n_2}$ und $m^3 \pmod{n_3}$ mit dem chinesischer Restsatz bestimmt sie $m^3 \pmod{n_1 n_2 n_3}$.

Da $m < \min\{n_1, n_2, n_3\}$ ist kennt sie nun m^3 . Die dritte Wurzel innerhalb von \mathbb{Z} ist schnell berechnet.

Lösung 9:

Dieses mal ist die Entropie der Nachrichtenquelle zu klein.

Nehmen wir an Berta möchte herausfinden wie viel Anton für seinen neuen PC ausgeben hat. Berta weiß, das Anton den Rechner Anfang August gekauft hat. Es kommen nur etwa 10 Tage für das Kaufdatum in Frage. Die internen Rechnungsnummern werden fortlaufend vergeben und im August ist der Zähler irgendwo zwischen 700 und 800. Antons Kostenstelle (Nr. 37) ist Berta bekannt und der Computer wird vermutlich zwischen 1000€ und 2000€ kosten. Insgesamt gibt es also $10 \cdot 100 \cdot 100000 = 10^7$ mögliche Nachrichten. Diese kann Berta alle testweise verschlüsseln und mit dem Geheimtext vergleichen. Da die 10^7 Nachrichten noch nicht einmal gleichwahrscheinlich sind hat Berta gute Chancen mit weit weniger als 10^7 Versuchen ans Ziel zu gelangen.

Ein ähnlicher Fehler wurde übrigens bei der Ausgabe der elektronisch gesicherten Personal Ausweise gemacht. Der Schlüssel hing in wesentlichen Teilen von leicht erratbaren Dingen wie Name des Ausweisbesitzers und Ausweisnummer (wurde fortlaufend vergeben) ab. Die Entropie des Schlüssel sank auf diese Weise unter 40 Bit.