

Welcome to the jungle

Kryptographie

Die Magie der asymmetrischen Verschlüsselung

Tobias Swolany aka „keksa“
kryptographie@keksa.de
<http://keksa.de>

01/24/2010

Inhaltsverzeichnis

0	Vorwort	Seite 2
1	Verschlüsselung - was zum Henker is'n das?	Seite 3
2	Symmetrische Verschlüsselung	Seite 3
2.1	Symmetrische Verschlüsselung - ROT13	Seite 4
3	Asymmetrische Verschlüsselung	Seite 5
3.1	Asymmetrische Verschlüsselung - RSA	Seite 6
4	Das Problem mit den großen Zahlen	Seite 12
5	Sicherheit von RSA	Seite 13
6	Quellcode (Anhang)	Seite 14
7	Kontakt.....	Seite 16

Vorwort

So, heute lernen wir mal etwas ordentliches :) Ich weiß, es gibt viele Paper die dieses Thema "behandeln", aber diese sind mir zu oberflächlich und langweilig. Ich will hier darauf abzielen, dass ihr danach wisst, wie ihr eine Applikation programmieren könnt, die asymmetrische Verschlüsselung verwendet. Sprich, ich werd euch ganz genau erklären was es ist und wie die Magie funktioniert.

Aber zuvor noch ein paar Bemerkungen. Dieses Thema hat sehr viel mit Mathematik zu tun, lasst euch davon nicht abschrecken :) Ich werde alles bis ins kleinste Detail und im Kindergarten-Niveau erklären. Ob du nun den erweiterten euklidischen Algorithmus herunterbeten kannst oder ob du nicht einmal weißt was Primzahlen sind, ist völlig egal :) Also hol' dir erst mal einen ordentlichen Kaffee, mach's dir bequem, leg' ein Stück Papier neben die Tastatur und nimm was zum Kritzeln in die Hand; wir werden jetzt etwas absolut geniales für unser verpfushtes Leben lernen.

Verschlüsselung - was zum Henker is'n das?

Viele verwechseln Kryptographie (^=Verschlüsselung) mit Steganographie (^=Informationen verstecken). Es gab früher ganz oft auf irgendwelchen Ich-Bin-1337-Haxor-CD's lauter Steganographie-Tools mit denen man Nachrichten in Bildern "verstecken" konnte. Man hat im Prinzip das Bild innen Hexeditor reingeschoben und Nachrichten auf NULL-Bytes geschrieben, was ja umwerfend intelligent ist, besonders wenn so ein 8x8 Pixel Bild plötzlich 5 MB groß war. Nur die letzten Idioten haben nicht gemerkt, dass in diesen vermeintlichen Bildern andere Informationen "versteckt" waren. Und wenn man da einmal drauf kam, war es auch kein Problem mehr den Text zu lesen, denn er war nicht verschlüsselt ;)

Bei der Verschlüsselung ist es wiederum genau umgekehrt. Man weiß, dass da Informationen übermittelt werden, man schafft es nur nicht diese zu lesen, weil sie unkenntlich gemacht wurden. Nur mit einer bestimmten Methode ist es möglich, diesen unleserlichen Text wieder leserlich zu machen. Und von diesen Methoden gibt es sehr viele, genau genommen existieren genau so viele Methoden wie Zahlen, nämlich unendlich. Denn man kann sich ja immer wieder eine ausdenken.

Alle Methoden lassen sich aber in 2 Kategorien einteilen:

- 1) symmetrische Verschlüsselung
- 2) asymmetrische Verschlüsselung

Um den Unterschied am besten klar zu machen, will ich auf die symmetrische Verschlüsselung eingehen, dort die wohl bekannteste Methode (ROT13) erklären und dann auf das Problem der symmetrischen Verschlüsselung hinweisen.

Symmetrische Verschlüsselung

Die symmetrische Verschlüsselung ist bei weitem einfacher als die asymmetrische. Es ist das, was sich jemand wohl ausdenken würde, wenn er sich noch nie über Kryptographie informiert hat.

Das Prinzip, das dahinter steckt, ist, dass jemand einen Text auf eine einfache Art und Weise verändert, es übermittelt und der Empfänger den Text auf die selbe Art und Weise (vielleicht nur umgekehrt) wieder lesbar macht. Z.B. der Absender tauscht A mit B aus, B mit C, C mit D und so weiter. Der Text "HALLO" würde dann zu "IBMMP". So wird dieser Text dann versandt. Der Empfänger macht dies dann rückgängig; er tauscht B mit A aus, C mit B, D mit C und so weiter. So kann er dann wieder den Text "HALLO" lesen. Dies kann man natürlich auch variieren, man kann z.B. auch jeden Buchstaben nicht mit dem nächsten, sondern mit dem zweit-nächsten Buchstaben im Alphabet austauschen, oder mit dem 13t-nächsten, womit wir zur Verschlüsselung ROT13 kommen.

Symmetrische Verschlüsselung - ROT13

Denn genau das macht ROT13 (rotate 13 = drehe 13). Das Alphabet hat 26 Zeichen. Wir schreiben 2 mal das Alphabet untereinander, nur dass wir das zweite Alphabet um die Hälfte der Zeichenanzahl (13) verschieben. Daraus bekommen wir dann:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M

Wenn wir jetzt unseren Text "HALLO" mit ROT13 verschlüsseln wollen, dann suchen wir einfach die Buchstaben in der oberen Hälfte der Tabelle und tauschen ihn mit dem Buchstaben aus der unteren Hälfte der Tabelle, also wir tauschen H mit U aus, A mit N, L mit Y und so weiter. Dann erhalten wir den Text "UNYYB". Der Empfänger macht genau das selbe und erhält dann wieder den Text "HALLO".

Eigentlich können wir ROT13 nicht als Verschlüsselung bezeichnen, denn jeder kennt es und jeder kommt drauf, wenn er sich den Text genau ansieht :) Wikipedia hat es da schön beschrieben, es ist wie wenn in der Zeitung die Lösung eines Rätsels umgekehrt gedruckt wurde, es dient nur dazu, damit man es nicht versehentlich liest :)

Es gibt aber dennoch symmetrische Verschlüsselungsverfahren die etwas aufwändiger sind. Das Problem bei symmetrischer Verschlüsselung ist

jedoch, dass die Gesprächspartner sich erst darüber einigen müssen, wie sie die Texte verschlüsseln. Und wie soll das geschehen? Über eine nicht-verschlüsselte Leitung? Oder soll ich mich mit jedem Gesprächspartner dann persönlich treffen um die Methode auszutauschen? Ich sehe es schon auf mich zukommen: 450 Millionen Menschen fahren zur Ebay-Filiale um ein unikates Verschlüsselungsverfahren zu entwickeln :)

Nee, so geht das natürlich nicht. Und das ist der Punkt an dem wir zur asymmetrischen Verschlüsselung kommen.

Asymmetrische Verschlüsselung

Sooo :) Bisher war das alles ja die selbe Floskel wie man sie in "Tuts" über Kryptographie lesen kann, mit denen Kinder einen auf intelligent machen. Jetzt geht es aber so langsam so richtig an's Eingemachte.

Was ist denn nun asymmetrische Verschlüsselung? Es ist die Lösung zu dem "Verschlüsselungsmethode ausmachen" :) Die symmetrische Verschlüsselung heißt nämlich so, weil beide Gesprächspartner die selbe Methode bzw. Rechnung durchführen (der eine eventuell nur umgekehrt). Bei der asymmetrischen Verschlüsselung ist es aber nicht so. Da schickt nämlich der Empfänger dem Absender eine Methode/Rechnung, mit der er den Text so verschlüsseln kann, dass NUR der Empfänger sie mit einer geheimen Rechnung wieder entschlüsseln kann.

Ich werde hier als Beispiel auf RSA eingehen. RSA wurde 1977 von Rivest, Shamir und Adleman am MIT entwickelt. Die Methode ist offen bekannt und konnte so von vielen Mathematikern und Informatikern untersucht werden, sie gilt als "sicher". Sie verwendet einen öffentlichen Schlüssel (zum VERschlüsseln) und einen privaten Schlüssel (zum ENTschlüsseln). Beim ver- und entschlüsseln wird die ganze Zeit die selbe Formel verwendet, und zwar:

$$\text{TEXT} \times \text{modulo } N$$

Zur Erklärung der Formel komme ich gleich :) Die Sache, auf die ich hinaus will, ist, dass bei dieser Formel je nach dem, ob man verschlüsseln oder entschlüsseln will, einfach nur die Werte einträgt. Will man verschlüsseln, so trägt man bei "TEXT" die Zahl des Buchstaben ein (für gewöhnlich geht man nach offizieller ASCII-Tabelle), bei x trägt man einen Wert des öffentlichen

Schlüssels ein und bei "N" trägt man einen Wert, der ebenfalls beim öffentlichen Schlüssel ist, ein. Will man entschlüsseln, so macht man genau das selbe, nur dass man bei "TEXT" die Zahl des verschlüsselten Buchstaben einträgt, und bei x und "N" die Werte des geheimen privaten Schlüssels, und schon hat man den Text entschlüsselt. Und den privaten Schlüssel kennt NUR der Empfänger, deswegen kann NUR der Empfänger es entschlüsseln.

So, jetzt lehn dich zurück und lass dir das auf der Zunge zergehen :) Es benötigt schon ein bisschen Hirn um zu verstehen, wie genial das ganze eigentlich ist. In der Mathematik kann man Rechnungen immer ganz einfach rückwärts gehen, indem man sie einfach auf den Kopf stellt. Z.B. $2 * 5 = 10$ kann man zurück gehen: $10 / 5 = 2$, das haben wir alle in der Grundschule gelernt. Wie zum Henker bekommt man es jetzt hin, dass man mit simplen Einmaleins einen Text verschlüsselt, den man NICHT mal eben rückwärts zurückrechnen kann? Also klar, natürlich kann man sie zurück rechnen, aber nur mit einer GANZ BESTIMMTEN Zahl, die man NICHT mit Hilfe der Zahl herausbekommt, mit der man es verschlüsselt hat. Wie zum Henker soll das denn gehen?! :D

Asymmetrische Verschlüsselung - RSA

Das erkläre ich am besten anhand des RSA-Beispiels. Das komplizierteste daran ist, wie man sich sicherich schon vorstellen kann, das Erzeugen der Schlüssel. Der öffentliche Schlüssel besteht aus 2 Zahlen, wir nennen sie hier "e" und "N". "e" ist das, was bei x eingesetzt wird und "N" ist das, was bei N eingesetzt wird. Der private Schlüssel ist genau so, nur dass anstatt "e" ein "d" verwendet wird, das ebenfalls in der Formel bei x eingesetzt wird, das "N" ist bei dem privaten und öffentlichen Schlüssel die selbe Zahl. Also, öffentlicher Schlüssel (e, N) und privater Schlüssel (d, N).

Jetzt müssen wir nur für "e", "d" und "N" passende Zahlen finden, die mit der oberen Formel eine asymmetrische Verschlüsselung ermöglichen :D

Um das Problem des nicht-zurück-rechnen-können zu lösen, nutzen wir eine sogenannte Einweg-Funktion. Wir können unsere Zahlen generieren, indem wir in den privaten Schlüssel eine Information hinterlegen, die im öffentlichen Schlüssel nicht vertreten ist. Wie gesagt, man kann alles "rückwärts rechnen", aber bei einer Einweg-Funktion ist dies nur sehr schwer möglich. Es ist wie wenn man einen Namen hat und im Telefonbuch dazu die Nummer heraussucht, das ist kein Problem, aber wenn wir nur die Nummer haben,

dann ist es sehr schwer (bzw. es dauert verdammt lange) bis man den Namen dazu findet.

Das ermöglichen wir in unserer Verschlüsselung dadurch, dass wir 2 Primzahlen miteinander multiplizieren. Primzahlen sind Zahlen, die nur durch sich selber und 1 glatt teilbar sind (z.B. 5 ist eine Primzahl, da 5/4 ungerade, 5/3 ungerade und 5/2 ungerade). Wir benötigen also 2 Primzahlen, hier sind Primzahlen von 2 bis 99991. Wir wählen als Beispiel mal 23 und 31, diese zwei Zahlen nennen wir p und q.

$$p = 23, q = 31$$

Die Einweg-Funktion, die jetzt realisiert wird, kommt durch eine sogenannte Primfaktorisation. Auf Deutsch, wenn wir 2 Primzahlen multiplizieren (z.B. $23 * 31 = 713$), dann ist es schwer herauszufinden, aus welchen zwei Primzahlen das Ergebnis (713) gemacht wurde. Das ist unsere Einweg-Funktion. Na gut, das ist jetzt kein Problem, wir schreiben uns ein Programm und dann dauert es keine Sekunde, bis wir die Zahlen 23 und 31 raus haben, aber für gewöhnlicherweise werden auch keine so kleine Primzahlen genommen, sondern eher Primzahlen die 100 bis 200 Stellen haben :) Da ist das wiederum schon sehr viel schwieriger. Ich habe jetzt nur als Beispiel zwei so kleine Zahlen genommen. Also multiplizieren wir jetzt die beiden Primzahlen, daraus entsteht schonmal unser "N", das wir für beide Schlüssel brauchen! :) Das "N" nennt man auch Modul:

$$p = 23, q = 31$$
$$N = 713 = 23 * 31$$

Jetzt müssen wir uns noch eine Zahl errechnen, die uns beim Erzeugen von e und d hilft. Und zwar brauchen wir das Ergebnis von der eulerschen Funktion "Phi", hier schmeißen wir unser N rein und merken uns das Ergebnis. Die eulersche Phi-Funktion berechnet, wie viele kleinere Zahlen es gibt, die sich dann mit der eingegebenen Zahl dividiert nicht normal schreiben lässt. So lässt sich zum Beispiel $1/3$ nicht als normale Zahl aufschreiben, weil es $0,33333333...$ mit unendlich vielen 3en ist. Hingegen lässt sich $1/2$ einfach darstellen, und zwar ist das $0,5$:) Ich hoffe das ist verständlich, so ist z.B. $\text{Phi}(10) = 4$, weil
 $10/9$ ist lässt sich nicht normal schreiben,
 $10/8$ ist lässt sich normal schreiben,
 $10/7$ ist lässt sich nicht normal schreiben,
 $10/6$ ist lässt sich normal schreiben,

10/5 ist lässt sich normal schreiben,
10/4 ist lässt sich normal schreiben,
10/3 ist lässt sich nicht normal schreiben,
10/2 ist lässt sich normal schreiben,
10/1 ist lässt sich nicht normal schreiben.

4 Zahlen lassen sich nicht normal schreiben. Aber, wir können hier in unserem Fall eine Abkürzung nehmen. Da wir wissen, dass unser N (die 713) aus den Primzahlen p und q besteht (23 und 31), können wir p und q einmal runterzählen und dann multiplizieren, daraus kriegen wir das selbe Ergebnis! Dann brauchen wir diese bekloppte eulersche Phi-Funktion nicht machen. Also $(p - 1) * (q - 1) = \text{Phi}(N)$, das rechnen wir also:

$p = 23, q = 31$ $N = 713 = 23 * 31$ $\text{Phi}(N) = 660 = 22 * 30$
--

Die Zahl 660 brauchen wir unbedingt um e und d zu errechnen. Cool, jetzt sind wir nur noch 2 Rechnungen von unseren ersten selbst gemachten privaten und öffentlichen RSA-Schlüsseln entfernt :D Aber die letzte Rechnung hat's auch nochmal so richtig in sich. Zunächst rechnen wir aber noch e aus.

"e" können wir uns aussuchen. Es gibt da nur ein paar kleine Bedingungen. Und zwar sollte e auf jeden Fall größer als 1 und kleiner als $\text{Phi}(N)$, hier also 660, sein. Außerdem sollte sie teilerfremd zu $\text{Phi}(N)$, hier also 660, sein. Was heißt teilerfremd? Teilerfremd bedeutet, dass die Zahl außer der 1 keinen gemeinsamen Teiler zu einer anderen Zahl hat.

19 und 17 sind teilerfremd, weil sie keine Zahl außer der 1 haben, die die beiden glatt teilt.

45 und 324 sind nicht teilerfremd, es gibt nämlich eine Zahl außer der 1, die beide Zahlen gleich teilt, nämlich die 9.

Hier gibt es wieder eine kleine Abkürzung, die wir nehmen können. Da die zu wählende Zahl "e" auf jeden Fall teilerfremd zu $\text{Phi}(N)$ sein sollte, muss sie eine Primzahl sein! Es geht nicht anders. Also, bevor wir jetzt herumsuchen und tausende Zahlen ausprobieren, können wir erst überprüfen, ob die Zahl eine Primzahl ist, bevor wir sie auf Teilerfremdheit checken. Noch ein kleiner Tipp, es wird bevorzugt, dass e klein gehalten wird, ein kleines e lässt sich leichter rechnen und ist keinerlei Gefahr für die Verschlüsselung.

Damit man sich jetzt nicht mit dem Taschenrechner dumm und dämlich rechnet, habe ich ein PHP-Skript geschrieben, das ein mögliches "e"

ausrechnet, das Skript liegt natürlich auch dem Download bei: <http://keksa.de/rsa.php>

Ich habe mir mal ein e ausgerechnet, und zwar 23. Damit lautet unser öffentlicher Schlüssel = (23, 713), denn der Schlüssel besteht ja aus e und N :

$p = 23, q = 31$
 $N = 713 = 23 * 31$
 $\text{Phi}(N) = 660 = 22 * 30$
 $e = 23$
public key: (23, 713)

Jetzt müssen wir nur noch den privaten Schlüssel ausrechnen :) " e " konnten wir uns selber aussuchen, das liegt daran, weil " d " auf " e " aufbaut. Wie der private Schlüssel aussieht ist also davon abhängig, wie der öffentliche Schlüssel aussieht. Und jetzt kommen wir erst mal zu einer richtig üblen Rechnung. Eigentlich ist sie ganz einfach, ich habe dazu auch ein Programm geschrieben, bei dem ihr einfach nur zwei Zahlen reinschmeissen müsst und er spuckt euch schon das Ergebnis aus (liegt mit Quellcode dem Download bei), aber falls ihr versuchen wollt es auf Papier nachzurechnen, dann solltet ihr euch zunächst vergewissern, dass ihr noch genug Kaffee in der Tasse habt ;)

Download: http://keksa.de/?dl=crypt_tools.zip (33,6 KB)

MD5-Checksum: BD199C17A285EEA385F1347C091A9273

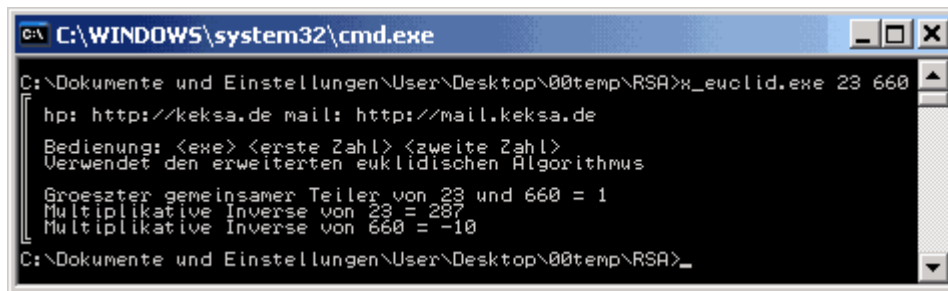
SHA1-Checksum: 57F5E58869463F4F9A40CFD2CD10293EB01CD8D1

Und zwar benötigen wir hier den erweiterten euklidischen Algorithmus. Um ungefähr 300 vor Christus gab es mal einen Mathematiker (Euklid), der den euklidischen Algorithmus erfunden hat. Damit kann man den größten gemeinsamen Teiler (auch kurz ggT) errechnen, das habe ich auch in dem PHP-Skript implementiert (Funktion "ggT"). Und dann kam da irgendwann (ich bin kein Geschichtslehrer, das interessiert mich auch nicht :D) einer her und hat es erweitert, und zwar um die sogenannten Bézout-Koeffizienten. Das sind zwei Zahlen " s " und " t ", die neben dem ggT ausgespuckt werden und die stehen in ganz besonderer Beziehung zu den Eingabezahlen und dem ggT:

$$\text{ggT}(a, b) = s * a + t * b$$

Man man man, die Welt ist bekloppt. Egal, jedenfalls sind " s " und " t " die multiplikativen Inversen von den Eingabezahlen " a " und " b ". Und genau DAS

brauchen wir! :D Um genauer zu sein, wir brauchen die multiplikative Inverse von "e" in Beziehung zu Phi(N). Sprich, "e" und Phi(N) sind die beiden Zahlen, die wir bei meinem Proggie reinhauen müssen :P



```
C:\WINDOWS\system32\cmd.exe
C:\Dokumente und Einstellungen\User\Desktop\00temp\RSA>x_euclid.exe 23 660
hp: http://keksa.de mail: http://mail.keksa.de
Bedienung: <exe> <erste Zahl> <zweite Zahl>
Verwendet den erweiterten euklidischen Algorithmus

Groeszter gemeinsamer Teiler von 23 und 660 = 1
Multiplikative Inverse von 23 = 287
Multiplikative Inverse von 660 = -10
C:\Dokumente und Einstellungen\User\Desktop\00temp\RSA>
```

"Multiplikative Inverse von 23 = 287" <- für diese Aussage werd ich in der Uni im Mathe-Tower Schläge kassieren. Aber egal, 287 ist genau die Zahl, die wir brauchen, das ist unser "d" :) Der private Schlüssel setzt sich aus d und N zusammen, also ist unser privater Schlüssel (287, 713):

```
p = 23, q = 31
N = 713 = 23 * 31
Phi(N) = 660 = 22 * 30
e = 23
public key: (23, 713)
d = 287
private key: (287, 713)
```

Amen. Öffentlicher und privater Schlüssel generiert, jetzt wollen wir auch schauen ob das funktioniert :) Also wir verschlüsseln erst mal mit dem öffentlichen Schlüssel und entschlüsseln dann mit dem privaten Schlüssel. Das machen wir mit der oben bereits erwähnten Formel:

TEXT x modulo N

An diese Stelle nochmal kurz erklärt was das komische "modulo" eigentlich bedeutet. Modulo ist eine Rechenoperation wie Plus oder Minus. Und zwar rechnet es den Rest aus einer Division zweier Zahlen aus. Um es besser zu verstehen hier wieder zwei Beispiele: 234 modulo 100 = 34, weil in 234 passt 2 mal 100 rein, was dann üblich bleibt ist 34. Noch ein Beispiel, 43 modulo 8 = 3, weil (jetzt mal anders gerechnet) 8 + 8 = 16, 16 + 8 = 24, 24 + 8 = 32, 32 + 8 = 40. Wenn wir jetzt nochmals + 8 rechnen, dann sind wir schon über 43 hinüber, also schauen wir einfach, was als Rest übrig bleibt, und das ist dann

das Ergebnis von 43 modulo 8, und zwar $43 - 40 = 3$.
Auf Taschenrechnern ist modulo meistens als "mod" abgebildet.

Wir wollen jetzt keinen richtigen Text verschlüsseln, es reicht ja schon, wenn wir einen Buchstaben verschlüsseln. Noch eine kleine aber wichtige Bemerkung: Die Zahl, die wir als "TEXT" eingeben darf keinesfalls größer als N sein! Unser N ist 713, also darf der zu verschlüsselnde Buchstabe nicht größer als 713 sein. Gehen wir mal nach der ASCII-Tabelle und verschlüsseln wir den Buchstaben "K":

Verschlüsseln:
K ist in der ASCII-Tabelle 75
Also TEXT = 75
Öffentlicher Schlüssel ist (23, 713)
Also $e = 23$ und $N = 713$
TEXT x modulo N
wird zu
75 23 modulo 713
das ergibt 489

489 ist also unser verschlüsselter Buchstabe "K". Hier noch ein Bild in welcher Reihenfolge man es in den Windows-Taschenrechner eingibt. Ich weiß, dass ich jetzt vielleicht ein wenig übertriebe, da gibt es jetzt auch keine Tricks, aber ich will das einmal über die Bühne bringen, damit ich nicht nochmals erklären muss wie das geht :D Das Eintippen ist unfassbar oft eine Fehlerquelle:



So. Jetzt schickt jemand diesen "Text" zu uns und wir müssen den natürlich wieder lesbar machen :) Dazu verwenden wir den privaten Schlüssel:

Entschlüsseln:
TEXT = 489
Privater Schlüssel ist (287, 713)
Also $d = 287$ und $N = 713$
TEXT x modulo N
wird zu
489 287 modulo 713
das ergibt 75
75 ist in der ASCII-Tabelle K

Fertig. Damit können wir jetzt ganze Texte asymmetrisch Verschlüsseln, wenn wir wollen. Zwar ist das nicht ganz sicher, weil die Primzahlen p und q viel zu klein sind, aber dennoch denke ich, dass man das als Verschlüsselt bezeichnen könnte :)

Man muss dazu noch sagen, dass die asymmetrische Verschlüsselung deutlich mehr Rechenleistung benötigt als die symmetrische (ungefähr 1000 mal so viel), deswegen wird das RSA-Verfahren meistens nur dazu verwendet, um den symmetrischen Schlüssel auszutauschen. Das nennt man dann Hybrides Verfahren.

Das Problem mit den großen Zahlen

Wie bereits erwähnt, werden für p und q gewöhnlich Primzahlen verwendet, die zwischen 100 und 200 Ziffern lang sind. Mit normalen Integers kommt man da nicht sehr weit :) Auch mit riesigen int's wie z.B. int64 ist man nicht bedient. Deswegen muss dann da ein eigenes Verfahren her. Also mal schnell ein Klasse programmieren und paar Methoden reinknallen.

Wie kann man beliebig große Zahlen im Rechner darstellen? Wie wäre es mit einem Array? `int[1000]` würde schon genügen, aber dann muss man dem Computer auch beibringen damit zu rechnen. Schrifliches Addieren und so, wie in der Grundschule :) Das hab ich selber auch schon gemacht, komplizierter wird es dann, wenn man dann modulo rechnen, oder Potenzieren will. Und irgendwann habe ich gemerkt, dass das Darstellen und Rechnen von beliebig großen Zahlen eine Aufgabe für sich ist, denn es gibt noch lauter Tricks wie z.B. dass man viel Rechenzeit sparen kann, wenn man überprüft, wie viele Stellen des Arrays nicht genutzt werden. Oder wenn man 23^7 rechnen will, dann kann der Computer das VIEL schneller, wenn er erst 23^4 rechnet, dann 23^3 und die beiden Zahlen dann miteinander multipliziert. Schon bei 20-stelligen Zahlen kann dieser Trick schon ein Unterschied von einigen Minuten und weniger als eine Sekunde bedeuten.

Da ist es empfehlenswerter, wenn man vorgefertigte Bibliotheken verwendet, davon gibt es reichlich und sie sind sehr effizient. Ich persönlich habe mich bei der Entwicklung beliebig großer Zahlen stundenlang durchgeackert, am Anfang hat es ja noch Spaß gemacht, aber dann wurde es nur noch lästig. Es ist nicht unmöglich aber doch sehr langweilig und belastend, deswegen nochmals der Tipp mit den vorgefertigten Bibliotheken.

Sicherheit von RSA

Es gibt da die typischen Angriffsmethoden auf RSA wie der Faktorisierungs-Angriff (p und q herausfinden). Diese sind alle mehrmals gut dokumentiert und scheitern alle an extrem kleinen Wahrscheinlichkeiten. Dann ist mir aber selber etwas aufgefallen, das mich erst ein paar Stunden unter Schock gesetzt hat. Ich habe dann natürlich sofort einen Professor angeschrieben, der mir das bitte erklären soll :P Der hat mir auch geantwortet, aber vorher bin ich bereits selber auf die Lösung gekommen, warum meine Idee nicht besonders gut funktionieren kann. Hier mal meine E-Mail:

Guten Tag,
[Bla, wer bin ich, was mach ich] und habe mich privat mit RSA befasst. Da ist mir eine Kleinigkeit aufgefallen, die ich mir nicht erklären konnte und hoffe, dass Sie mir da vielleicht weiterhelfen könnten.

Und zwar kann man gezielt vom öffentlichen Schlüssel den privaten Schlüssel errechnen? Das sollte doch eigentlich nicht möglich sein?

Ein Beispiel: Wir haben uns ein Schlüsselpaar (hier natürlich kleine Zahlen) generiert, das sind einmal
- öffentlicher Schlüssel (125 | 2881) und
- privater Schlüssel (377 | 2881).
Angenommen ich bin jetzt der Angreifer und ich kenne nur den öffentlichen Schlüssel (125 | 2881). Dann kann ich die eulersche Phi-Funktion auf das N anwenden, und zwar habe ich mir dazu ein Programm geschrieben, das alle Möglichkeiten durchgeht (Phi(N) wird nicht durch $(p-1)*(q-1)$ errechnet da wir diese nicht kennen, sondern es werden tatsächlich alle Zahlen $< N$ auf Teilerfremdheit geprüft). Daraus bekomme ich dann schonmal mein Phi(N): 2772. Jetzt kann ich mir mit Hilfe des erweiterten euklidischen Algorithmus ganz einfach den privaten Schlüssel errechnen, und zwar ist die multiplikative Inverse von 125 in Beziehung zu 2772 = 377. Und schon habe ich den privaten Schlüssel errechnet.

Ist das normal? Liegt da irgendwo ein Fehler vor? Verstehe ich da irgendwas nicht richtig?

Ich hoffe Sie können mir helfen.
Mit freundlichen Grüßen
[Me, myself and I]

Hm, interessanter Punkt, dachte ich mir :) Aber dann ist mir, wie gesagt, selber in den Sinn gekommen, warum das nicht funktionieren kann.

Ich habe nämlich mit kleinen Zahlen gerechnet, klar, als Beispiel. Da ist es kein Problem $\Phi(N)$ auch auf manueller Art und Weise zu bestimmen. Aber für gewöhnlich sind p und q ja 100 und 200-stellige Ziffern, also ist N dann $100 * 200 = 20.000$ Stellen lang. Und DA haben wir dann ein kleines Problemchen... :D Auf meinem Rechner würden 14-stellige Ziffern bis zum nächsten Urknall dauern. Naja, aber die gute Nachricht ist, dass man das alles nur einmal rechnen bräuchte, da man ja eine Rainbow-Tabelle für alle Zahlen anlegen könnte. Naja. Hmmh. Hehe. Science-Fiction :D Hier die Mail, die ich als Antwort vom Prof bekommen habe:

Guten Tag,
Lieber Herr [Ich],

Sie haben schon in soweit recht, dass man "grundsätzlich" aus dem öffentlichen Schlüssel den privaten Schlüssel errechnen kann. Dabei haben Sie aber übersehen, dass die (vermutete) Komplexität der dabei zu lösenden Aufgabe diese "praktisch nicht berechenbar" macht (sofern die Parameter groß genug gewählt werden).

Wenn Sie mehr darüber lernen wollen, schauen Sie am besten in einschlägige Bücher. In meinem eigenen Buch finden Sie dazu insbesondere im Abschnitt 12.9.2. , Seite 415-418, einige weitergehende Hinweise.

Herzliche Grüße
[Proffieprof]

Quellcode

So, ich glaube das war's. Mehr kann ich nicht dazu sagen. Ich schmeiß hier noch wie üblich den Quellcode rein, diesmal nur von dem C-Programm, dem erweiterten euklidischen Algorithmus, denn mit dem kann ja im Prinzip auch das ermöglichen, was das PHP-Skript macht. So far.

```

/*****
erweiterter euklidischer Algorithmus
Für neuste Version und Dokumentation besuchen
Sie bitte http://keksa.de/
*****/

#include<windows.h>
#include<stdio.h>
#include<math.h>

int main(int argc, char** argv){
    int a;
    int b;
    int q, d=0, v=0, s=0, u=1, t=1;
//    int change=0;

    printf("%c\n%c hp: http://keksa.de mail: http://mail.keksa.de\n%c \n%c "
        "Bedienung: <exe> <erste Zahl> <zweite Zahl>\n%c "
        "Verwendet den erweiterten euklidischen Algorithmus\n%c ", (char)201,
        (char)186, (char)186, (char)186, (char)186, (char)186);

    if(argc<3)
        return 0;

    a=atoi(argv[1]);
    b=atoi(argv[2]);

//    if(a<b)          /* vertauschen */
//        change=1;

    while(b>0){          /* erweiterter euklidischer Algorithmus */
        q=a/b;
        d=a-q*b; a=b; b=d;
        d=u-q*s; u=s; s=d;
        d=v-q*t; v=t; t=d;
    }

//    if(change==1){          /* vertauschen */
//        q=v;
//        v=u;
//        u=q;
//    }

```



```
d;      /* [!] d never used bla... */
printf("\n%c Groeszter gemeinsamer Teiler von %s und %s = %i",
(char)186,
      argv[1], argv[2], a);
if(a==1)
  printf("\n%c Multiplikative Inverse von %s = %i\n%c Multiplikative "
        "Inverse von %s = %i\n%c ", (char)186, argv[1], u,
        (char)186, argv[2], v, (char)200);
else
  printf("\n%c Es gibt keine multiplikative Inversen zu %s und %s\n%c ",
        (char)186, argv[1], argv[2], (char)200);
}

// keksa.de
```

Kontakt

Webpage: <http://keksa.de>

(RSS): <http://keksa.de/rss.php>

E-Mail: [Keksa <admin@keksa.de>](mailto:admin@keksa.de)

Contact: <http://keksa.de/?q=respond>

PGP-Key: 1024D/6193F777 <http://keksa.de/keksa.de.asc>

Privacy: <https://privacybox.de/keksa.msg>

Twitter: <http://twitter.com/keksaDE>

(RSS): http://twitter.com/statuses/user_timeline/89171093.rss

TorChat: (by appointment)

EOF