Rechnen mit großen natürlichen Zahlen am Beispiel der Implementation eines RSA-Kryptosystems

Einleitung

Als Wettbewerbsbeitrag für 32. Bundeswettbewerb „Jugend forscht“ 1997 im Fachbereich „Mathematik/Informatik“ implementierten wir ein RSA-Kryptosystem in einem MS Windows™-Programm. Zielsetzung war ein Programm, welches auch „Computerlaien“ ermöglicht, private e-mail-Korrespondenz sicher zu verschlüsseln. Im folgenden Artikel sollen die Grundlagen des Verfahrens sowie der mathematischen Implementation der Berechnungen näher erläutert werden.

Asymmetrische Kryptosysteme

Man unterscheidet in der Kryptologie zwei grundlegend verschiedene Typen von Verschlüsselungsverfahren. Zum einen handelt es sich hierbei um die sogenannten symmetrischen Kryptosysteme, zum anderen spricht man von asymmetrischen Kryptosystemen. Die erstgenannten Systeme zeichnen sich dadurch aus, daß beide Kommunikationspartner einen gemeinsamen Schlüssel nutzen. Bei asymmetrischen Verfahren existieren im Gegensatz dazu zwei verschiedene Schlüssel: der Sender besitzt einen Schlüssel zum Verschlüsseln ($\boldsymbol{E_T}$), der Empfänger einen anderen zum Entschlüsseln ($\boldsymbol{D_T}$). Beide Teilnehmer nutzen dieselben Algorithmen zum Ver- und Entschlüsseln (***e*** bzw. ***d***). Für jede Nachricht ***m*** in einem solchen asymmetrischen Kryptosystem gelte nun: $\boldsymbol{d}(\boldsymbol{e}(\boldsymbol{m},\boldsymbol{E_T}),\boldsymbol{D_T})=\boldsymbol{m}$.
Die Idee eines asymmetrischen Kryptosystems wurde erstmals 1976 von Whitfield Diffie und Martin Hellman geäußert [DH76].

RSA- Algorithmus

Die erste praktische Implementierung eines solchen asymmetrischen Kryptosystems lieferten 1978 die drei amerikanischen Mathematiker Ronald Rivest, Adi Shamir und Leonard Adlemann mit dem nach ihnen benannten RSA- Algorithmus [RSA78]. Nachfolgend sollen kurz die Verfahren zur Ver- und Entschlüsselung beschrieben werden:
Man bestimme zwei hinreichend „große“ Primzahlen ***p*** und ***q*** („groß“ bedeutet im Falle des RSA-Algorithmus meistens 200 stellig). Die Zahl ***n*** sei durch $\boldsymbol{n} = \boldsymbol{pq}$ definiert.Man bestimme weiterhin eine „große“ ganze Zufallszahl ***d***, wobei gelte : *ggT ((**p**-1) (**q**-1), **d**) = 1 (**d** sei zur Zahl (**p**-1) (**q**-1) teilerfremd).* Nachfolgend wird eine eindeutig bestimmte ganze Zahl ***e*** nach der Formel $\boldsymbol{ed} = 1 \pmod{(\boldsymbol{p}-1)(\boldsymbol{q}-1)}$ berechnet, wobei gelte: ***e*** £ *(**p**-1) (**q**-1).* Nun wird der öffentlichen Schlüssel („public- key“) bekanntgegeben, der aus dem Paar ganzer Zahlen (***e***,***n***) besteht. Man trenne die zu übertragende Nachricht ***M*** in gleich große Blöcke und stelle diese Blöcke als Zahl dar, wobei gelte *1£* ***M***£ ***n.*** Man verschlüssele ***M*** in das Kryptogramm ***C***, wobei hier gelte $\boldsymbol{C} = \boldsymbol{M}^{e} \pmod{\boldsymbol{n}}$

Eine Entschlüsselung dieses Kryptogramms ***C*** ist mit Hilfe des Privatschlüssels ***d*** möglich. Die entschlüsselte Nachricht berechnet sich demzufolge wie folgt: $\boldsymbol{D} = \boldsymbol{C}^{\boldsymbol{d}} \ (mod\ \boldsymbol{n})$

Grundlagen der Berechnungen

Zur Umsetzung dieser Theorie ist es offensichtlich nötig, eine modulare Exponentation der Form $\boldsymbol{a}^{\boldsymbol{b}}$ (mod ***c***) durchzuführen. Ziel unserer Überlegung war es, diese Berechnung auf Grundrechenoperationen zurückzuführen.
Bedingt durch die Rechnerarchitektur werden alle Berechnungen intern mit Dualzahlen durchgeführt. Dabei sind jeweils 16 Bits zu einem Block zusammengefaßt. Jeder Block kann somit einen Zahlenwert von 0 bis 2^{16}-1 annehmen.
Um die modulare Exponentation durchzuführen, benötigt man Algorithmen für die Multiplikation zweier Dualzahlen und für die Berechnung ***x*** mod ***y*** für zwei Dualzahlen ***x*** und ***y***. Die Berechnung der Kongruenz ***x*** mod ***y*** wird durch fortgesetztes binäres Nach-Links-Schieben (Left-Shift) von ***y*** und anschließende Subtraktion des modifizierten ***y*** von ***x*** erreicht.

Addition und Subtraktion von Dualzahlen

Diese Operationen können mit linearer Zeitkomplexität durchgeführt werden, daher ist keine größere Optimierung möglich. Bei der Berechnung werden zunächst die niedrigsten Blöcke addiert bzw. voneinander subtrahiert, der jeweilige Übertrag wird gespeichert und beim nächsthöheren Block aufaddiert. Dieses einfache Verfahren wird mit Hilfe einer Schleife solange fortgesetzt, bis alle Dualziffern addiert bzw. subtrahiert wurden. Die einzige Optimierung, die hierbei angebracht schien, war die Benutzung von Assembler-Befehlen, so daß die Rechenoperationen direkt mit den Registern des Prozessors durchgeführt werden.

Multiplikation von Dualzahlen - elementare Methode, Verfahren von Karatsuba

Die Multiplikation von zwei ***N***-stelligen Zahlen nach dem elementaren Verfahren (schriftliche Multiplikation) besitzt im allgemeinen eine Zeitkomplexität von $\mathbf{N}^2$, da jede Ziffer der ersten Zahl mit jeder Ziffer der zweiten Zahl multipliziert wird. Die Zeiten für das Nach-Links-Schieben und die Additionen können wegen der bekannten linearen Zeitkomplexität beider Operationen vernachlässigt werden.
Für die Multiplikation von großen Zahlen („groß“ bedeutet in diesem Fall eine Länge von mindestens 8 Blöcken) bietet sich allerdings das Verfahren von Karatsuba-Ofman (1963) an [Ca97]. Bekanntlich besitzt jede nicht-negative ganze Zahl p die folgende Summendarstellung:

$$\boldsymbol{p} = \boldsymbol{p_0} + \boldsymbol{p_1}\,\boldsymbol{B} + \boldsymbol{p_2}\,\boldsymbol{B^2} + \boldsymbol{p_3}\,\boldsymbol{B^3} + ... + \boldsymbol{p_{N-1}}\boldsymbol{B^{N-1}} \quad \text{mit } 0 \le \boldsymbol{p_i} \le \boldsymbol{B - 1}.$$

Dabei ist ***B*** die Basis des Zahlensystems (in unserem Fall 2^{16}) und die $\boldsymbol{p_i}$ stellen die Ziffern von ***p*** dar. Die Zahl p wird folgendermaßen in 2 Hälften geteilt:

$$p = (p_0 + p_1\,B + p_2\,B^2 + ... + p_{N/2-1}\,B^{N/2-1}) + (p_{N/2} + p_{N/2+1}\,B + p_{N/2+2}\,B^2 + ... + p_{N-1}\,B^{N/2-1})B^{N/2}$$

Die Dualzahl p_L sei der niederwertige Teil, p_H der höherwertige Teil von p:

$$p = p_L + p_H B^{N/2}.$$

Sind nun beispielsweise zwei Dualzahlen

$$p = p_L + p_H B^{N/2}$$

und

$$q = q_L + q_H B^{N/2}$$

gegeben, so läßt sich das Produkt schreiben als

$$pq = r_0 + r_1 B^{N/2} + r_2 B^N,$$

wobei für die „Koeffizienten" gilt:

$$r_0 = p_L q_L$$

$$r_1 = p_L q_H + p_H q_L = p_L b_L + p_H b_H - (p_L - p_H) (q_L - q_H)$$

$$r_2 = p_H q_H.$$

Es werden also zur Berechnung der „Koeffizienten" (die die Ziffern des Produkts darstellen) anstatt 4 Multiplikationen nur 3 benötigt. Das heißt, bei der Multiplikation von Zahlen mit 2^n Ziffern werden nur 3^n anstatt 4^n Multiplikationen benötigt, der Zeitaufwand schrumpft auf $(3/4)^n$. Das heißt, das Verfahren besitzt eine Zeitkomplexität von $O(N^{\log 3})$.

Praktisch arbeitet das Verfahren von Karatsuba-Ofman nach dem Prinzip „Teile und herrsche". Dabei wird das Problem rekursiv so in kleinere Probleme vom gleichen Typ zerlegt, daß die Zusammenfassung der Lösungen der Teilprobleme zur Lösung des Gesamtproblems einfach ist. Die Zahlen werden jeweils in 2 Hälften zerlegt, bis die Anzahl der Ziffern unter ein gewisses Limit fällt, dann wird die herkömmliche Methode benutzt, um die kleinen Zahlen zu multiplizieren.

Praktisch kann dieses Verfahren wie folgt in jeder gängigen Hochsprache implementiert werden: Die folgende Routine multipliziert die Dualzahlen p und q, speichert das Produkt in r, die Dualzahl tmp wird nur als temporärer Zwischenspeicher genutzt. Die Zahlen p und q bestehen aus 2N Blöcken, r und tmp haben die Länge 4N.

Karatsuba (r, p, q, tmp, N)

Wenn N < Limit, dann führe gewöhnliche Multiplikation durch.

Sonst:

(1) Rufe Karatsuba auf, um das Produkt $p_L q_L$ im niederwertigen Teil von r zu speichern, dabei wird der niederwertige Teil von tmp als temporärer Zwischenspeicher übergeben.

(2) Setze das dritte Viertel von r auf $|p_H - p_L|$ und das vierte Viertel von r auf $|q_H - q_L|$, beachte dabei die Vorzeichenkonstellation.

(3) Rufe Karatsuba auf, um das Produkt $|p_H - p_L||q_H - q_L|$ im niederwertigen Teil von tmp zu speichern, dabei wird der höherwertige Teil von tmp als temporärer Zwischenspeicher übergeben.

(4) Rufe Karatsuba auf, um das Produkt $p_H q_H$ in der höherwertigen Hälfte von tmp zu speichern, dabei kann der höherwertige Teil von r als Zwischenspeicher genutzt werden.

(5) Addiere (oder subtrahiere) die berechneten Teile in r, achte dabei auf eventuelle Überträge.

Modulo-Operation

Zur Bestimmung des Restes, den eine Zahl x bei Division durch eine Zahl y läßt, werden von x solange Vielfache von y abgezogen, bis das Ergebnis kleiner als y ist. Da es sich bei x und y um Dualzahlen handelt, erweist es sich als günstig, wenn y immer nur mit Zweierpotenzen multipliziert wird, da diese Operation nur aus einem binärem Schieben besteht und somit eine lineare Zeitkomplexität besitzt.

Division

Zur Bestimmung des Quotienten zweier Dualzahlen x und y wird die für den Computer optimierte „Schulmethode“ eingesetzt. Sind die Dualzahlen $x = (x_{n-1}, ..., x_0)$ und $y = (y_{n-1}, ..., y_0)$ gegeben, so läuft die Berechnung des Quotienten $q = (q_{n-1}, ..., q_0)$ wie folgt ab: In einer Schleife werden die folgenden 3 Schritte für i = n - 1, ..., 0 durchgeführt:

1. Vergleiche die Binärzahlen $x' = (x_{n-1}, ..., x_i)$ und $y = (y_{n-1}, ..., y_0)$.
2. Falls x’ kleiner als y ist, dann setze q_i auf 0.
3. Falls $x' \geq y$ ist, setze $q_i = 1$ und ersetze $(x_{n-1}, ..., x_i)$ durch die Differenz aus x’ und y.

Dieser Algorithmus enthält insgesamt n Stufen mit je einem Vergleich und einer Subtraktion.
Hinweis: Natürlich kann der Modulo - Algorithmus mit der Divisionsroutine gekoppelt werden. Allerdings ist in der Praxis meist nur entweder der Rest bei der Division oder der Quotient zweier Zahlen gesucht.

Modulare Exponentiation

Weil der RSA- Algorithmus wie oben beschrieben sehr rechenaufwendig ist, haben wir in unserem Programm Optimierungsmethoden genutzt, um die Rechenzeit zu verkleinern. Ein erster Ansatzpunkt für Optimierungen ist die modulare Exponentation, die Berechnung M^e (MOD n).
Wenn man bei der Berechnung der Potenz M e mal mit sich selbst multiplizieren würde, würde sich bei einer Begrenzung von M und e auf 512 Bits eine Zahl mit bis zu 2^{512} Binärstellen ergeben. Eine Komplettberechnung der Potenz erscheint also wenig sinnvoll. Man kann eine solche große Zahl aber vermeiden, wenn man jeder Einzelmultiplikation eine Modulo- Berechnung folgen läßt [1]. Nun ergeben sich höchstens Zahlen mit einer Länge der doppelten Stellenzahl von n (es gilt immer M < n und e < n). Um die Rechenzeit zu verkürzen, wird nun folgender Algorithmus zur modularen Exponentation angewendet, für diesen Algorithmus muß e in binärer Form vorliegen.

Man definiert Z als Anzahl der Binärstellen von e. Verwendet wird eine Hilfsvariable C.
Es sei $e = e_{z-1}, e_{z-2}, ..., e_1, e_0$ die Binärzahldarstellung von e mit $0 \leq e_i \leq 1$ und i von 0 bis Z-1.
Die folgenden Schritte werden für i = Z-1, Z-2, ..., 1, 0 durchgeführt:

1. Man berechnet $C = C^2$ MOD n
2. Falls $e_i = 1$ ist, berechnet man C = (C*M) MOD n

[1] dies ist möglich, da gilt : (A*B) MOD N = ((A MOD N) * (B MOD N)) MOD N

Mit diesem Algorithmus reduziert sich die Anzahl der modularen Multiplikationen im Mittel auf 3*Z/2, bei 512-bit Zahlen ergeben sich so im Mittel 768 modulare Multiplikationen.

Zusammenfassung, Ausblicke

Die oben beschriebenen Berechnungsverfahren wurden in unserem Programm FASTcrypt implementiert. Dieses intuitiv bedienbare e-mail- Verschlüsselungsprogramm ist vor allem für unerfahrene Computeranwender gedacht. Ein professioneller Vertrieb des Programms wird zukünftig angestrebt. Weitere Untersuchungen auf dem Gebiet der Kryptologie werden von uns im Bereich der Vesrchlüsselung von Finanzdaten bei elektronischen Einkaufsprozessen unternommen werden.

Literaturverzeichnis

[Be93] Beutelspacher, Albrecht; „Kryptologie“ ; Wiesbaden 1993

[Ca97] Casselman, Bill: „Notes on fast multiplication of large numbers“ , cass@math.ubc.ca

[DH76] Diffie, Whitfield; Hellman, Martin E.; „New Directions in Cryptography“ , in : „IEE Transactions on Information Theory“ ; VOL. IT-22 No. 6; November 1976

[Kn81] Knuth, Donald Ervin; „The Art of Computer Programming.“ , „Volume 2: Seminumerical Algoritms“ , Reading, 1981

[RSA78] Rivest, Ronald L.; Shamir, Adi; Adleman, Leonard; „A Method for Obtaining Digital Signatures and Public Key Cryptosystems“ ; in: „Communications of the ACM“ ; Volume 21, Number 2; February 1978

Patrick Reichert, Christoph Seidler
Halle (Saale)
FASTcrypt@ncc.east.de
CSeidler@cantor.east.de