In: Weck, G.; Horster, P. (Hrsg.): *Verläßliche Informationssysteme*. Proceedings der GI-Fachtagung VIS'93, DuD-Fachbeiträge 16, Vieweg Verlag, Braunschweig 1993, S. 333-352.

Der "Digital Signature Standard": Aufwand, Implementierung und Sicherheit

Dirk Fox
Schneider & Koch & Co. Datensysteme GmbH
Daimlerstraße 15, W-7500 Karlsruhe 21

Abstract

Mit der Veröffentlichung des Entwurfs für einen "Digital Signature Standard" (DSS) durch NIST im Herbst 1991 wurde erstmalig für ein kryptographisches Verfahren zur Erzeugung elektronischer Unterschriften eine Standardisierung eingeleitet. Nach einer Erläuterung des DSS-Signier- und Testalgorithmus' werden effiziente Algorithmen für eine Software-Implementierung der im DSS-Verfahren benötigten modularen Langzahl-Exponentiation betrachtet. Es folgen Vorschläge für eine schnelle Version der DSS-Algorithmen mit Vorausberechnungen. Aufwandsabschätzungen und Messungen einer Implementierung für 80x86-Mikroprozessoren des DSS- und des RSA-Verfahrens werden verglichen. Die Darstellung schließt mit Betrachtungen zur Sicherheit des DSS-Verfahrens auf der Grundlage der aktuellen Diskussion.

Einleitung

Mit ihrer wegweisenden Arbeit über asymmetrische Kryptoverfahren stießen Diffie und Hellman 1976 [13] eine intensive Forschungstätigkeit im Bereich der modernen Kryptographie an, die bis heute nicht zum Stillstand gekommen ist. Eine Schlüsselrolle spielt dabei die Entwicklung kryptographischer Verfahren zur Erzeugung digitaler Signaturen, mit denen die Authentizität und Integrität von Nachrichten auch gegenüber Dritten beweisbar wird. Die Idee ist einfach: Der Sender i einer Nachricht N berechnet mit einem nur ihm bekannten, geheimzuhaltenden Signierschlüssel G_i und einem (allgemeinen) Signieralgorithmus S eine Signatur Sig für N,

$$Sig := S(N, G_i).$$

Das Paar (N, Sig) schickt er an den gewünschten Empfänger. Dieser erhält ein Paar (N', Sig') und kann nun mit Hilfe eines öffentlich bekannten Testschlüssels $Ö_i$ des Senders i, den er bspw. einem Verzeichnis ("Telefonbuch") entnimmt, und des Testalgorithmus' T die Herkunft und Unverfälschtheit der Nachricht N' prüfen:

$$T(N', Sig', Ö_i) = true\ ?$$

Da der Schlüssel $Ö_i$ allgemein bekannt ist, ermöglicht ein solches Verfahren dem Empfänger den Nachweis gegenüber Dritten, daß die Nachricht N' tatsächlich von Sender i stammt und während der Übertragung (oder vom Empfänger selbst) nicht verfälscht wurde.

Die Sicherheit eines solchen Digitalen Signaturverfahrens beruht auf seiner Asymmetrie, d.h. der praktischen Unmöglichkeit, aus einem öffentlich bekannten Testschlüssel $Ö_i$ den zugehörigen geheimen Signierschlüssel G_i zu gewinnen. Mehr noch: Das Verfahren muß gewährleisten, daß ein *passiver*, d.h. lediglich mithörender Angreifer, aus Paaren (N, Sig) ebenfalls keine Informationen gewinnen kann, die ihm das Fälschen einer Signatur ermöglichen. Idealerweise sollte das Verfahren auch gegen *aktive* Angriffe gefeit sein, d.h.

gegen das Abhören von Signaturen zu vom Angreifer selbst gewählten und eingeschleusten Nachrichten N [23].

Digitale Signaturverfahren sind für eine Vielzahl von Anwendungen geeignet: Sie können nicht nur als Authentizitäts- und Integritätsnachweis für elektronisch übermittelte Nachrichten eingesetzt werden, sondern spielen auch eine Rolle in Schlüsselaustauschprotokollen für klassische, symmetrische Kryptoverfahren, siehe z.B. [7, 11]. Eine weitere, zur Zeit wieder intensiver diskutierte Anwendung ist ihr Einsatz als Software-Integritätstest zur Erkennung von unautorisierter Programmodifikation, beispielsweise durch Virenprogramme [12].

Das älteste bekannte Digitale Signaturverfahren ist der 1978 veröffentlichte, auf dem sogenannten "Faktorisierungsproblem" basierende RSA-Algorithmus [40]. Inzwischen wurde eine große Zahl weiterer Verfahren vorgestellt, deren Sicherheit auf unterschiedlichen, meist zahlentheoretischen Problemen beruht. Dennoch hat sich RSA (wenigstens in den USA) als "de-facto"-Industriestandard durchsetzen können. Mit dem DSS liegt nun erstmalig ein Standardentwurf für ein Digitales Signaturverfahren vor [14].

In Kapitel 1 wird zunächst die Funktionsweise des im DSS-Entwurf spezifizierten Signier- und Testalgorithmus' skizziert. Anschließend werden in Kapitel 2 Überlegungen zu deren effizienter Software-Implementierung und Aufwandsbetrachtungen angestellt. Ausführlich wird dabei auf Algorithmen zur modularen Exponentiation eingegangen. Es folgen Vorschläge für eine Implementierung des DSS-Verfahrens mit Vorausberechnungen und ein Vergleich von Aufwand und Software-Implementierung des DSS- und des RSA-Verfahrens. Die Untersuchung schließt mit einer übersichtsartigen Darstellung der aktuellen Sicherheitsdiskussion über den vorgeschlagenen Standard in Kapitel 3 und einer zusammenfassenden Bewertung in Kapitel 4.

1 Der Standardentwurf

Am 30. August 1991 wurde vom US-amerikanischen "National Institute of Standards and Technology" (NIST) ein Vorschlag für einen "Digital Signature Standard" (DSS) veröffentlicht. Das in diesem Entwurf spezifizierte Signaturverfahren leitet sich ab vom El-Gamal-Signaturverfahren [16]; es ähnelt einer von Schnorr vorgeschlagenen Variante [42]. Die Sicherheit dieses Verfahrens beruht auf der praktischen Unmöglichkeit, diskrete Logarithmen langer Zahlen (d.h. den Logarithmus einer Zahl bezüglich eines großen Modulus) in vernünftiger Zeit zu berechnen.

Das vorgeschlagene Signaturverfahren besitzt die folgenden Parameter:[1]

- Öffentliche, authentische Verfahrensparameter (p, q, g): p, $q \in Prim$[2] mit $2^{511}<p<2^{512}$, q Teiler von (p-1), $2^{159}<q<2^{160}$; $g:= h^{(p-1)/q} \bmod p$; dabei sei $0<h<p$ mit $g>1$.
- Geheimzuhaltender Signierschlüssel G_i: beliebige Ganzzahl mit $0<G_i<q$.
- Authentisch zu veröffentlichender Testschlüssel $Ö_i := g^{G_i} \bmod p$.

Weiter muß eine Hashfunktion H vereinbart werden, mit der jede zu signierenden Nachricht N zunächst auf einen Haschwert in der zulässigen Blocklänge, d.h. H(N)<q, abgebildet wird.[3]

---

1 Die Beschränkung der Größe von p wird in der überarbeiteten Fassung des Entwurfs aufgehoben. Die Länge von p kann dann frei in 64-bit-Schritten zwischen 511 und 1023 bit gewählt werden [SmBr_92].

2 *Prim* bezeichne die Menge aller Primzahlen >2.

3 H(N) wird auch "Fingerabdruck" oder "Message Digest" (MD) genannt.

Diese muß *kollisionsfrei* [10] gewählt werden, d.h. es darf praktisch nicht möglich sein, ein Nachrichtenpaar (N_1, N_2), $N_1 \neq N_2$ zu finden mit $H(N_1) = H(N_2)$.[4]

Die Hashfunktion H wird im DSS-Entwurf nicht festgelegt. Vom NIST wurde jedoch am 31. Januar 1992 ein Entwurf für einen "Secure Hash Standard" (SHS) veröffentlicht [43], der eine Hashfunktion für das DSS-Verfahren spezifiziert. Die dort vorgeschlagene Hashfunktion ähnelt dem "Message Digest Algorithm" MD4 von Ronald Rivest, RSA Inc. [36, 37]. Sie erzeugt aus einer bis zu 2^{64}-1 bit langen Nachricht einen 160 bit langen "Fingerabdruck".[5]

Der DSS-Signieralgorithmus S erzeugt zu einer gegebenen Nachricht N folgendermaßen eine Signatur $Sig = S(N, G_i)$:

- Zu der Nachricht N wird mit der Hashfunktion H ein "Fingerabdruck" H(N) bestimmt.
- Der Signierer wählt (je Signatur neu) zufällig ein k mit $0<k<q$.
- Anschließend berechnet er $r:= (g^k \bmod p) \bmod q$ und $s:= (k^{-1}\cdot(H(N) + G_i\cdot r)) \bmod q$.[6]

Das Paar Sig:= (r, s) wird als die zur Nachricht N gehörige Signatur mit N verschickt. Sig umfaßt demnach maximal 318 bit.

Der Empfänger eines Paares (N', Sig') mit Sig' = (r', s') kann nun mit dem DSS-Testalgorithmus $T = T(N', Sig', Ö_i)$ auf folgende Weise die Authentizität und Integrität der Nachricht N' prüfen:

- Zu der empfangenen Nachricht N' bestimmt er den "Fingerabdruck" H(N').
- Er berechnet das multiplikative Inverse w von s' modulo q: $w:= s'^{-1} \bmod q$.
- Dann werden zwei Hilfswerte bestimmt: $u_1:= H(N')\cdot w \bmod q$, $u_2:= r'\cdot w \bmod q$.
- Zuletzt prüft der Empfänger, ob gilt: $r' = (g^{u_1}\cdot Ö_i^{u_2} \bmod p) \bmod q$.

Wurden weder Nachricht noch Signatur bei der Übertragung verfälscht, dann gilt:

$(H(N')+G_i\cdot r')\cdot w = k$, und damit auch $(g^k \bmod p) \bmod q = (g^{(H(N')+G_i\cdot r')\cdot w} \bmod p) \bmod q$.

Scheitert der Test, dann wurde entweder die Nachricht N' oder die Unterschrift Sig' verfälscht.

2 Aufwand und Implementierung

Läßt man die Hashfunktion H außer Betracht, dann bestehen Signier- und Testfunktion des DSS-Entwurfs ausschließlich aus modularen Langzahloperationen: Multiplikationen, Quadrierungen, Modulo-Operationen und Additionen. Sowohl der Euklidsche Algorithmus zur Bestimmung von k^{-1} (Signieralgorithmus) und s'^{-1} (Testalgorithmus) als auch die modularen Exponentiationen setzen sich aus diesen Basisoperationen zusammen [2, 27, 28, 31].

Eine effiziente Implementierung des DSS-Verfahrens erfordert daher dreierlei:

- eine optimierte modulare Langzahlarithmetik, d.h. schnelle Multiplikations-, Divisions-, Quadrierungs-, Additions- und Modulooperationen für sehr große Ganzzahlen,
- einen effizient implementierten Euklidschen Algorithmus und
- einen modularen Exponentiationsalgorithmus, der mit möglichst wenigen modularen Langzahloperationen auskommt.

---

4 Genauer: Die Wahrscheinlickeit, ein solches Paar durch systematisches Verfahren zu finden, darf nicht signifikant größer sein als Raten: $P(H(N_1)=H(N_2)) = 2^{-160}$.

5 Eine genauere Betrachtung der SHS-Haschfunktion würde den Rahmen dieser Darstellung sprengen.

6 Mit k^{-1} wird das multiplikative Inverse von k modulo q bezeichnet.

Optimierte Algorithmen für modulare Langzahloperationen sind umfangreich in der Literatur beschrieben und werden an dieser Stelle nicht betrachtet; siehe dazu [28, 8, 33, 22, 6, 9, 15, 34, 30, 19]. Effiziente Implementierungen des Euklidschen Algorithmus sind in einschlägiger Literatur zu finden [2, 28, 31]; auch auf sie wird daher nicht eingegangen.

Anders bei Algorithmen für die modulare Exponentiation: Sie setzen sich aus sehr vielen aufwendigen modularen Langzahloperationen zusammen und machen über 95% des Aufwandes von Signier- und Testalgorithmus aus; Optimierungen sind an dieser Stelle besonders wirkungsvoll. Zwar finden sich Vorschläge in der Literatur [28, 9, 19], keineswegs alle sind jedoch für das DSS-Verfahren geeignet. In Abschnitt 2.1 wird daher der Aufwand unterschiedlicher Exponentiationsalgorithmen im Hinblick auf ihre Verwendung im DSS-Verfahren genauer untersucht. Aufbauend darauf werden in den Abschnitten 2.2 und 2.3 effiziente Implementierungen des DSS-Signier- und Testalgorithmus' skizziert und deren Aufwand bestimmt. Das Kapitel schließt mit einer vergleichenden Darstellung der Ausführungsgeschwindigkeiten einer effizienten Softwareimplementierung des DSS- und des RSA-Verfahrens für 80x86-Mikroprozessoren in Abschnitt 2.4.

2.1 Exponentiationsalgorithmen

Eine modulare Exponentiation $b^e \bmod p$ setzt sich zusammen aus modularen Multiplikationen und modularen Quadrierungen. Bei den üblicherweise verwendeten Algorithmen wächst der Aufwand einer modularen Multiplikation quadratisch mit der Länge der Faktoren. Das gilt auch unabhängig von dem verwendeten Reduktionsverfahren [33, 28, 29]. Multiplikationsalgorithmen, deren Aufwand nicht quadratisch in der Faktorenlänge wächst, wie beispielsweise der von Karatsuba entwickelte mit $O(n^{\log 3})$, lohnen sich erst bei Moduluslängen, die erheblich über den beim DSS-Verfahren erforderlichen liegen [2, 28, 19].

Der Aufwand einer modularen Quadrierung kann bei effizienter Implementierung auf 3/4 einer modularen Multiplikation reduziert werden [6, 9, 19]. In den folgenden Aufwandsabschätzungen wird daher der Aufwand einer modularen Multiplikation mit A_M, der einer modularen Quadrierung mit $0.75 \cdot A_M$ bezeichnet.

Die Anzahl der für eine modulare Exponentiation erforderlichen modularen Multiplikationen und Quadrierungen hängt von dem zur Auswertung des Exponenten e verwendeten Algorithmus ab. Die folgenden Verfahren kommen dafür in Frage:

2.1.1 Square-and-Multiply

Die einfachste Methode ist der bekannte "Square-and-Multiply"-Algorithmus, der für jedes der m Exponentenbits $e_1, e_2, ..., e_m$ mit $e_i \in \{0,1\}$ eine modulare Quadrierung und (Gleichverteilung der {0,1}-Werte vorausgesetzt) durchschnittlich eine halbe modulare Multiplikation erfordert [28, 12, 7, 19].

Das Verfahren läßt sich kurz folgendermaßen beschreiben:

$$b^e = (...((1^2 \cdot b^{e_1})^2 \cdot b^{e_2})^2 \cdot ...)^2 \cdot b^{e_m}.$$

Die Auswertung des Exponenten beginnt demnach mit dem höchstwertigen Bit $e_1=1$.[7] Die erste modulare Quadrierung und die erste Multiplikation können durch eine Zuweisung ersetzt

[7] Alternativ kann die Auswertung auch mit dem niederstwertigen Bit beginnen; im Algorithmus ist dann jedoch eine weitere Hilfsvariable erforderlich [Knut_81, Hors_85].

werden: $1^2 \cdot b^{e_1} = b$. Der Gesamtaufwand des Verfahrens liegt damit, Gleichverteilung der Exponentenbits vorausgesetzt, bei

Square-and-Multiply $\lceil 1.25 \cdot (m-1) \rceil A_M$ **(1)**

Die folgende Tabelle vergleicht die Anzahl der im Square-and-Multiply-Verfahren erforderlichen modularen Multiplikationen und Quadrierungen für eine Exponentiation mit den DSS- und RSA-typischen Exponentenlängen m = 160 bit und m = 512 bit (bei übereinstimmendem Modulus):

Exponentenlänge m [bit]	160	512
modulare Multiplikationen	80	256
modulare Quadrierungen	159	511
Gesamtaufwand [A_M]	199	639

Tabelle 1: Aufwand des Square-and-Multiply-Algorithmus für m = 160 und m = 512 bit.

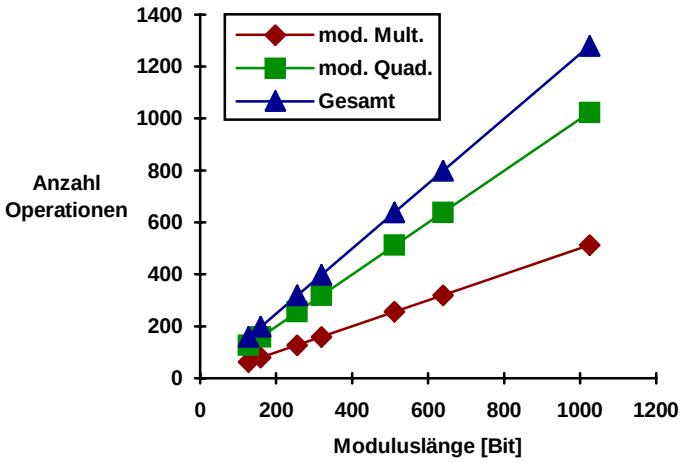

Bild 1: Anzahl der beim Square-and-Multiply-Algorithmus erforderlichen Operationen für die Moduluslängen 128, 160, 256, 320, 512, 640 und 1024 bit.

2.1.2 Bitgruppenauswertung

Eine Verallgemeinerung des Square-and-Multiply-Verfahrens reduziert durch blockweises Auslesen des Exponenten die Zahl der erforderlichen Multiplikationen erheblich. Dazu wird der Exponent beginnend mit dem höchstwertigen Bit e_1 in Bitgruppen (Nibbles) gleicher Länge r zerlegt. Dabei verbleibt ggf. ein kürzeres "Restnibble" [28, 9].[8]

[8] Der Square-and-Multiply-Algorithmus ist ein Spezialfall der Bitgruppenauswertung mit r = 1.

Seien $nib_1 := e_1...e_r$, $nib_2 := e_{r+1}...e_{2\cdot r}$, ..., $nib_i := e_{(i-1)\cdot r+1}...e_{i\cdot r}$ mit $i \in \{1, ..., (m \text{ div } r)\}$, weiter $nib^* := e_{(m \text{ div } r)\cdot r+1}...e_m$ das Restnibble, $z := 2^r$ und $z^* := 2^{(m \bmod r)}$. Dann läßt sich das Auswertungsverfahren folgendermaßen kurz beschreiben:

$$b^e = (...((1^z \cdot b^{nib_1})^z \cdot b^{nib_2})^z \cdot ...)^{z^*} \cdot b^{nib^*}.$$

Die Auswertung beginnt mit dem höchstwertigen Nibble nib_1.[9] Die Exponenten nib_i können genau 2^r unterschiedliche Werte annehmen. Die Faktoren b^j ($0 \le j \le 2^r-1$) werden vorausberechnet und in einer Tabelle vermerkt. Da $b^0 = 1$ und $b^1 = b$ bekannt sind, genügt es, eine Tabelle mit 2^r-2 Einträgen vorzusehen. Die Hälfte dieser Tabelleneinträge ($b^{2\cdot j}$, d.h. gerader Exponent) kann durch je eine Quadrierung, die andere ($b^{2\cdot j+1}$, d.h. Exponent ungerade) durch jeweils eine Multiplikation des vorausgehenden Eintrags mit der Basis b bestimmt werden. Insgesamt sind für diese Vorausberechnungen also $2^{r-1}-1$ modulare Quadrierungen und ebensoviele Multiplikationen durchzuführen. Wechselt die Basis bei jeder Exponentiation, so sind diese Vorausberechnungen jedesmal erneut erforderlich.

Ist die Tabelle gefüllt, sind nur noch $((m \text{ div } r)-1)\cdot r$ modulare Quadrierungen und $(m \text{ div } r)-1$ Multiplikationen für die $(m \text{ div } r)$ Nibbles vorzunehmen: Die ersten r Quadrierungen und die erste Multiplikation können durch eine Zuweisung ersetzt werden, da $1^z \cdot b^{nib_1} = b^{nib_1}$. Für das ggf. vergbleibende Restnibble nib^* sind weiter $(m \bmod r)$ modulare Quadrierungen und eine Multiplikation mit b^{nib^*} erforderlich.

Gleichverteilung der Exponentenbits vorausgesetzt, gilt für die Wahrscheinlichkeit, daß das Nibble nib_i den Wert 0 besitzt und daher eine modulare Multiplikation gespart werden kann:

$$P(nib_i=0) = 2^{-r}.$$

Da $(m \bmod r) < r$ ist die Wahrscheinlichkeit, daß keine Restbits übrigbleiben, das Restnibble also den Wert Null hat, etwas größer:

$$P(nib^*=0) = 2^{-(m \bmod r)}.$$

Neben der Vorausberechnung der Tabelle sind also immer m-r modulare Quadrierungen erforderlich und durchschnittlich $((m \text{ div } r)-1)\cdot(1-2^{-r})+(1-2^{-(m \bmod r)})$ Multiplikationen zu erwarten. Der Gesamtaufwand beträgt im Mittel:

$(2^{r-1}-1 + m-r)$ modulare Quadrierungen (exakt) und
$(2^{r-1} + [((m \text{ div } r)-1)\cdot(1-2^{-r})-2^{-(m \bmod r)}])$ modulare Multiplikationen (im Mittel).

Ausgedrückt in modularen Multiplikationen läßt sich der Gesamtaufwand des Verfahrens angeben mit durchschnittlich

Bitgruppenauswertung $\lceil 0.75\cdot(2^{r-1}+m-r-1) + 2^{r-1} + (m/r-1)\cdot(1-2^{-r})\rceil A_M$ **(2)**

Um den Gesamtaufwand zu optimieren, ist die Nibblelänge r in Abhängigkeit von der Exponentenlänge m einzustellen: Der Aufwand für die Vorausberechnungen darf die Einsparungen nicht übersteigen.

Daß eine Optimierung der Nibblelänge r lohnt, zeigen die folgenden Tabellen, die die Anzahl der notwendigen modularen Quadrierungen und Multiplikationen für die im DSS- und RSA-Verfahren auftretenden Exponentenlängen m = 160 bit und m = 512 bit für unterschiedliche Nibblelängen r angeben:

---

9 Beginnt man mit dem niederstwertigen Nibble, wird der Algorithmus erheblich komplizierter [Knut_81].

Nibblelänge r [bit]	2	3	4	5	6	7	8
modulare Multiplikationen	62	49	44	47	57	85	148
modulare Quadrierungen	159	160	163	170	185	216	279
Gesamtaufwand [A_M]	182	169	167	175	196	247	358

Tabelle 2: Aufwand der Bitgruppenauswertung für m = 160 bit.

Nibblelänge r [bit]	2	3	4	5	6	7	8
modulare Multiplikationen	194	152	126	114	114	135	190
modulare Quadrierungen	511	512	515	522	537	568	631
Gesamtaufwand [A_M]	578	536	512	506	517	561	663

Tabelle 3: Aufwand der Bitgruppenauswertung für m = 512 bit.

Die Formel (2) für den Gesamtaufwand kann zur Einstellung der Nibblelänge herangezogen werden: r ist optimal genau dann, wenn die Zahl der erforderlichen modularen Quadrierungen und Multiplikationen minimal wird. Daß dieser Wert für r klein sein muß, ist leicht einzusehen: Der Aufwand für die Vorausberechnung der Tabellen steigt exponentiell, die Einsparung wächst jedoch nur linear in r.

Das Minimum der Aufwandsformel läßt sich über die Nullstellen der ersten Ableitung finden. Es gilt:

$$f(r) = 0.75\cdot(2^{r-1}+m-r-1) + 2^{r-1} + (m/r-1)\cdot(1-2^{-r})$$

$$f'(r) = 7\cdot\ln(2)\cdot 2^{r-3} - \ln(2)\cdot 2^{-r} - 0.75 + m\cdot(2^{-r}/r^2 + \ln(2)\cdot 2^{-r}/r - r^{-2}$$

$$\mathbf{f'(r) = 0 \;\Rightarrow\; m = r^2\cdot(7\cdot\ln(2)\cdot 2^{r-3} - \ln(2)\cdot 2^{-r} - 0.75)/(1-2^{-r} - \ln(2)\cdot r\cdot 2^{-r}).} \quad \mathbf{(3)}$$

Mit dieser Gleichung (3) läßt sich r in Abhängigkeit von m tabellieren: Setzt man für r jeweils 1.5, 2.5 usw. ein, dann erhält man die Exponentenlänge in bit, ab der als Nibblelänge der Wert 2, 3 usw. zu wählen ist:

Exponentenlänge m [bit] ≥	6	31	107	321	904	2436	6351
Nibblelänge r [bit] =	2	3	4	5	6	7	8

Tabelle 4: Optimierung der Nibblelänge r für die Bitgruppenauswertung.

Der Vergleich der Tabellen 1, 2 und 3 zeigt, daß der Aufwand einer modularen Exponentiation bei optimaler Wahl von r gegenüber dem Square-and-Multiply-Verfahren erheblich reduziert werden kann: Bei einem 160 bit langen Exponenten sind nur noch 55% der modularen Multiplikationen erforderlich, und für m = 512 bit sinkt die Anzahl sogar unter 45%. Der Gesamtaufwand läßt sich also auf 84% bzw. 79% senken.

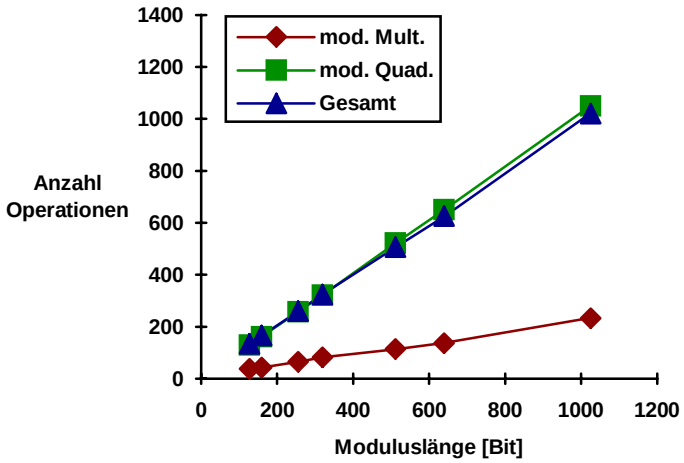

Bild 2: Anzahl der bei der Bitgruppenauswertung mit optimierter Länge r erforderlichen Operationen für die Moduluslängen 128, 160, 256, 320, 512, 640 und 1024 bit.

Ist die Basis wie beim DSS-Signieralgorithmus fest und bekannt oder sind viele Exponentiationen zu derselben Basis erforderlich, dann genügt es, die Tabelle einmalig vorauszuberechnen. Die Exponentiation setzt sich dann nur noch aus den bei der Auswertung des Exponenten anfallenden modularen Multiplikationen und Quadrierungen zusammen. Für bestimmte Anwendungen kann der Gesamtaufwand daher — je nach verfügbarem Speicherplatz, Rechenzeit und Häufigkeit der Exponentiationen — durch die Wahl größerer Nibbles weiter reduziert werden.

2.1.3 Fenstertechnik

Verwendet man statt festen Blöcken (wie bei der Bitgruppenauswertung) ein dynamisches Fenster fester Größe, das nur solche Blöcke auswählt, die mit einem 1-Bit beginnen, kann die erforderliche Tabelle halbiert und die Zahl der Multiplikationen weiter verringert werden [9, 19]. Eine solche Auswertung läßt sich realisieren, indem ein r bit breites "Fenster" in Auswertungsrichtung, d.h. beginnend beim höchstwertigen Bit e_1, über den Exponenten geschoben wird. Ist das erste Bit im Fenster ungleich Null, wird der Fensterinhalt als Nibble nib_i gewählt und das Fenster um r Bit weitergeschoben. Führende Null-Bits werden übersprungen, da sie

ohnehin keinen Einfluß auf den Faktor b^{nib_i} haben; für jedes übersprungene Bit ist jedoch vor der Multiplikation mit b^{nib_i} ein zusätzliches Mal zu quadrieren.

Bezeichne nun s_i die Zahl der vor dem Nibble nib_i übersprungenen Nullbits. Mit $s_1 = 0$, $nib_1 = e_1...e_r$, $nib_2 = e_{(r+s_2+1)}...e_{(2r+s_2)}$, $nib_3 = e_{(2r+s_2+s_3+1)}...e_{(3r+s_2+s_3)}$ usw. und $z_i := 2^{(r+s_i)}$, $z^* := 2^{rest}$ (rest sei dabei die Anzahl der verbleibenden, niederstwertigen Exponentenbits) gilt dann:

$$b^e = (...((1^{z_1} \cdot b^{nib_1})^{z_2} \cdot b^{nib_2})^{z_3} \cdot ...)^{z^*} \cdot b^{nib^*}.$$

Da alle Nibbles mit einem 1-Bit beginnen, ist nur noch die "obere Hälfte" der Tabelle vorauszuberechnen; die Fenstertechnik hat daher lediglich den halben Speicherbedarf der Bitgruppenauswertung. Mit r-1 modularen Hilfsquadrierungen erhält man den ersten, durch weitere 2^{r-1}-1 Multiplikationen alle weiteren Tabelleneinträge.[10]

Auch hier können die ersten r Quadrierungen und die erste Multiplikation durch eine Zuweisung ersetzt werden: $1^{z_1} \cdot b^{nib_1} = b^{nib_1}$. Es verbleiben exakt m-r Quadrierungen.

Der Erwartungswert für die Zahl der vor dem i-ten Nibble nib_i überspringbaren Null-Bits ist (für lange Exponenten e) näherungsweise eins (m_i bezeichnet hier die Zahl der hinter nib_{i-1} verbleibenden Exponentenbits):

$$\sum_{j=1,...,m_i} 2^{-(j+1)}.$$

Für die Abschätzung der erforderlichen Anzahl Multiplikationen kann leicht vergröbernd angenommen werden, daß (bis auf das erste Nibble) jeweils ein Exponentenbit übersprungen wird: Eine modulare Multiplikation erfolgt damit im Mittel nur alle r+1 Bits. Zu den Vorausberechnungen kommen daher weitere ((m-r) div (r+1)) Multiplikationen hinzu.

Der Restfaktor b^{nib^*} findet sich nicht in der Tabelle, da $nib^* < 2^{r-1}$. Er muß separat berechnet werden. Dazu sind (nach den Betrachtungen in Abschnitt 2.1.1) durchschnittlich (r-1)/2 modulare Multiplikationen durchzuführen.

Insgesamt sind für eine Exponentiation nach der Fenstertechnik erforderlich:

(m-1) modulare Quadrierungen (exakt) und

$(2^{r-1} - 1 + [(m-1) \text{ div } (r+1)] + \lceil (r-1)/2 \rceil)$ modulare Multiplikationen (im Mittel).

In modularen Multiplikationen läßt sich der Gesamtaufwand des Verfahrens durch folgende Formel ausdrücken:

Fenstertechnik $\lceil 0.75 \cdot (m-1) + 2^{r-1} - 1 + [(m-1) \text{ div } (r+1)] + (r-1)/2 \rceil A_M$ **(4)**

Auch hier ist die Größe des Fensters wie bei der Bitgruppenauswertung in Abhängigkeit von der Exponentenlänge zu optimieren. Die folgenden Tabellen und Graphiken zeigen die Unterschiede anschaulich für die DSS- und RSA-typischen Exponentenlängen:

---

10 Hier kann leider nicht - wie bei der Bitgruppenauswertung - die Hälfte der Multiplikationen durch schnellere Quadrierungen ersetzt werden.

Fenstergröße r [bit]	2	3	4	5	6	7	8
modulare Multiplikationen	55	43	40	43	56	88	148
modulare Quadrierungen	159	159	159	159	159	159	159
Gesamtaufwand [A_M]	175	163	160	163	176	208	268

Tabelle 5: Aufwand der Fenstertechnik für m = 160 bit.

Fenstergröße r [bit]	2	3	4	5	6	7	8
modulare Multiplikationen	172	131	110	101	105	129	187
modulare Quadrierungen	511	511	511	511	511	511	511
Gesamtaufwand [A_M]	555	514	493	484	488	512	570

Tabelle 6: Aufwand der Fenstertechik für m = 512 bit.

Die Bestimmung der optimalen Fenstergröße kann mit Gleichung (4) auf die gleiche Weise erfolgen wie bei der Bitgruppenauswertung: r ist optimal, wenn der Gesamtaufwand minimal ist. Dazu ist die Nullstelle der ersten Ableitung zu bestimmen:

$$f(r) = 0.75\cdot(m-1) + 2^{r-1} - 1 + (m-1)/(r+1) + 0.5\cdot(r-1)$$

$$f'(r) = \ln(2)\cdot 2^{r-1} - (m+1)/(r+1)^2 + 0.5$$

$$\mathbf{f'(r) = 0 \;\Rightarrow\; m = (r+1)^2\cdot(\ln(2)\cdot 2^{r-1} + 0.5) - 1.} \qquad \mathbf{(5)}$$

Nun läßt sich r in Abhängigkeit von m tabellieren, indem für r jeweils 1.5, 2.5, usw. in die Gleichung (5) eingesetzt wird. Die obere Zeile der Tabelle gibt dann die Exponentenlänge in bit an, ab der die Fenstergröße r auf den in der unteren Zeile angegebenen Wert eingestellt werden muß:

Exponentenlänge m [bit] ≥	9	30	89	252	683	1792	4568
Fenstergröße r [bit] =	2	3	4	5	6	7	8

Tabelle 7: Optimierung der Fenstergröße (Fenstertechnik).

Im Vergleich zum Square-and-Multiply-Algorithmus benötigt die Fenstertechnik bei optimaler Fenstergröße für einen 160 Bit langen Exponenten nur 50%, für einen 512 Bit langen Exponenten sogar weniger als 40% der Multiplikationen. Der Gesamtaufwand reduziert sich damit auf etwa 80% bzw. 76% des Square-and-Multiply-Algorithmus.

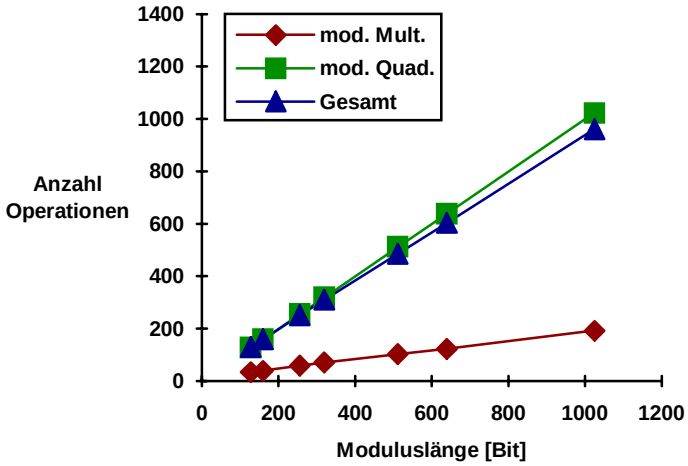

Bild 3: Anzahl der bei der Fenstertechnik und optimierter Wahl von r erforderlichen Operationen für die Moduluslängen 128, 160, 256, 320, 512, 640 und 1024 bit.

Erfolgen wie beim DSS-Signieralgorithmus viele modulare Exponentiationen mit derselben Basis bezüglich desselben Modulus oder sind Basis und Modulus lange vorher bekannt, genügt es, die Berechnung der Tabelle einmalig vorab durchzuführen. Für die eigentliche Exponentiation fallen dann nur noch die für die Auswertung des Exponenten erforderlichen Multiplikationen und Quadrierungen an. Auch hier kann der Gesamtaufwand aller Exponentiationen zur gleichen Basis durch die Wahl eines größeren Fensters weiter verringert werden (siehe Abschnitt 2.3).

2.1.4 Additionsketten

Das in Abschnitt 2.1.2 beschriebene Verfahren der Bitgruppenauswertung läßt sich verallgemeinern, indem die Bitgruppenzerlegung als Additionskette bezüglich einer festen Zweierpotenzbasis verstanden wird. Werden nun beliebige Basen zugelassen, ist es möglich, optimale Additionsketten mit minimalem Multiplikationsaufwand zu finden. Eine differenzierte Betrachtung dieses Verfahrens findet sich in [5, 41].

Es zeigt sich allerdings, daß die Einschränkung auf eine feste Zweierpotenz zu nur geringfügig vom Optimum abweichenden Zerlegungen führt [3]. Die Suche nach einer optimalen Zerlegung und die Basiskonvertierung des Exponenten erfordern zudem einen keineswegs vernachlässigbaren Aufwand. Sinnvoll ist die Verwendung von Additionsketten daher dann, wenn der Exponent fest bleibt und die optimale Zerlegung vorausberechnet werden kann. Da dies zwar beim RSA-Verfahren, nicht aber beim DSS-Verfahren möglich ist,[11] wird dieser Ansatz hier nicht weiter verfolgt.

2.2 Signieraufwand

Wie in Kapitel 1 skizziert, setzt sich der DSS-Signieralgorithmus aus einer Reihe von Operationen zusammen. Dazu zählen einmal der Aufwand für die Generierung der Zufallszahl k

[11] Bei Signier- und Testalgorithmus des DSS-Verfahrens bleibt die Basis konstant, siehe Abschnitte 2.2, 2.3.

und die Berechnung des "Fingerabdrucks" H(N) der Nachricht N mit der gewählten Hashfunktion H. Weiter werden

- eine modulare Exponentiation mit anschließender Reduktion vorgenommen (r:= (g^k mod p) mod q), 512 bzw. 160 bit Modulus, 512 bit Exponentenlänge),
- ein multiplikatives Inverses berechnet (160 bit Modulus) und
- zwei modulare Multiplikationen durchgeführt (160 bit Modulus).

Der Aufwand der Berechnung des Hashwertes ist von der Länge der Nachricht N abhängig. Da die Basisoperationen der Hashfunktion jedoch um Größenordnungen schneller arbeiten als die arithmetischen Operationen, ist der Aufwand selbst bei sehr langen Nachrichten vernachlässigbar [26].

Der für die Generierung einer Zufallszahl k (je Signatur erneut erforderliche) Aufwand kann hingegen nicht vernachlässigt werden. Er wird, da er von dem verwendeten Zufallszahlengenerator abhängt, hier mit A_G bezeichnet. Für den praktischen Einsatz bietet sich beispielsweise die Verwendung des BBS-"x^2-mod-n"-Pseudozufallszahlengenerators an, dessen Unvorhersagbarkeit äquivalent dem Faktorisieren des Modulus ist [4].**12** Er erfordert eine modulare Quadrierung für acht Pseudozufallsbits.**13** Bei der Generierung eines 160 bit langen k fallen also 20 modulare Quadrierungen an, d.h. $A_G = 15 \cdot A_M$. In einer späteren Version des DSS-Entwurfs soll ein Verfahren zur Generierung der Pseudozufallszahlen vorgeschlagen werden [44], daher wird der Aufwand A_G im folgenden nicht genauer bestimmt.

Ebenfalls nicht vernachlässigbar ist die Berechnung des multiplikativen Inversen k^{-1} mod q, die mehrere Langzahl-Divisionen erfordern. Deren Aufwand läßt sich nicht exakt in modularen Multiplikationen ausdrücken; asymptotisch liegt er bei O(log q) [31]. Er wird daher im folgenden separat angeführt und mit A_I bezeichnet.

Die Generierung von Zufallszahlen k kann vorab erfolgen, wenn sichergestellt ist, daß diese Werte geheim gehalten werden können. Dann können auch jeweils k^{-1} mod q und (g^k mod p) mod q vorausberechnet werden [26, 44].

Auf diese Weise läßt sich der aktuelle Aufwand einer Signatur auf zwei modulare Multiplikationen (160 bit Modulus) reduzieren - alle anderen Berechnungen erfolgen in der Vorphase.

Generell sollten Vorausberechnungen vor allem dann vorgesehen werden, wenn die Exponentiationszeit kritisch und die Leistung des signierenden Gerätes niedrig sind. Diese Effizienzverbesserungen lassen sich jedoch nicht in jeder Anwendung realisieren. Findet der Signiervorgang beispielsweise auf einer Smart Card statt, scheiden Vorausberechnungen wegen Speichermangels aus.

In der folgenden Darstellung wird der Aufwand des DSS-Signieralgorithmus dem Aufwand des durch Vorausberechnungen beschleunigten Signierverfahrens gegenübergestellt:**14**

---

12 Allerdings würde die Sicherheit des Signatursystems in diesem Fall sowohl auf der praktischen Unlösbarkeit des Diskreten-Logarithmus-Problems als auch auf der des Faktorisierungsproblems beruhen.

13 Das gilt für die hier betrachtete Moduluslänge von $m = 2^8 = 512$ Bit. Allgemein können nach jeder modularen Quadrierung die niederstwertigen ld(m) Bits des Ergebnisses verwendet werden.

14 Die Indizes 160 und 512 geben die Moduluslänge der bezeichneten Operation an.

DSS	Signieraufwand
	$A_G+A_{I_{160}}+484\cdot A_{M_{512}}+2\cdot A_{M_{160}}$
mit Vorausber.	$2\cdot A_{M_{160}}$

Tabelle 8: Aufwand des DSS-Signieralgorithmus' ohne und mit Vorausberechnungen.

2.3 Testaufwand

Auch beim Testen kann der Aufwand für die Berechnung des "Fingerabdrucks" H(N') vernachlässigt werden. Die Aufwandsabschätzung umfaßt daher

- die Berechnung eines multiplikativen Inversen (160 bit Modulus),
- zwei modulare Multiplikationen (160 bit Modulus),
- zwei modulare Exponentiationen ($g^{u_1}\cdot Ö_i^{u_2}$ mod p, 512 bit Modulus, 160 bit Exponentenlänge) und
- eine weitere modulare Multiplikation (160 bit Modulus).

Die beiden Exponentiationen $g^{u_1}\cdot Ö_i^{u_2}$ mod p lassen sich zusammenfassen, sofern der Square-and-Multiply-Algorithmus oder die Bitgruppenauswertung verwendet wird: Nach jedem Schritt der Exponentenauswertung können die Zwischenergebnisse miteinander multipliziert und die erforderlichen modularen Quadrierungen an dem Produkt durchgeführt durchgeführt werden. Auf diese Weise läßt sich die Zahl der erforderlichen modularen Quadrierungen halbieren [16]. Die folgende Darstellung illustriert das Verfahren für die Bitgruppenauswertung:

Sei u_1 zerlegt in Nibbles $nib_{u_1}1 := u_{11}...u_{1r}$, $nib_{u_1}2 := u_{1(r+1)}...u_{1(2\cdot r)}$, ..., $nib_{u_1}i := u_{1((i-1)\cdot r+1)}...u_{1(i\cdot r)}$ mit $i \in \{1, ..., (m \text{ div } r)\}$, und sei weiter $nib_{u_1}{}^* := u_{1((m \text{ div } r)\cdot r+1)}...u_{1m_1}$ das Restnibble. Analog sei u_2 zerlegt in die Nibbles $nib_{u_2}1$, ..., $nib_{u_2}{}^*$. Wie in Abschnitt 2.1.2 seien $z := 2^r$ und $z^* := 2^{(m \bmod r)}$. Dann gilt:

$$g^{u_1}\cdot Ö_i^{u_2} \bmod p =$$
$$[(...((1^z\cdot g^{nib_{u11}})^z\cdot g^{nib_{u12}})^z\cdot...)^{z^*}\cdot g^{nib_{u1}{}^*}\cdot(...((1^z\cdot Ö_i^{nib_{u21}})^z\cdot Ö_i^{nib_{u22}})^z\cdot...)^{z^*}\cdot Ö_i^{nib_{u2}{}^*}] \bmod p =$$
$$[(...((1^z\cdot g^{nib_{u11}}\cdot Ö_i^{nib_{u21}})^z\cdot g^{nib_{u12}}\cdot Ö_i^{nib_{u22}})^z\cdot...)^{z^*}\cdot g^{nib_{u1}{}^*}\cdot Ö_i^{nib_{u2}{}^*}] \bmod p.$$

Weiter sind die Basis g, die Basis $Ö_i$ und der Modulus p unabhängig von der signierten Nachricht und somit fest. Daher können - wie in Abschnitt 2.1.3 angemerkt - die für die Exponentiationen erforderlichen Tabellen vorausberechnet werden, sofern genügend Speicherplatz und Rechenzeit zur Verfügung stehen und signierte Nachrichten häufig getestet werden.[15] In den folgenden Aufwandsbetrachtungen werden (wie in Abschnitt 2.2) beide Methoden getrennt betrachtet:

---

15 Eine mögliche Anwendung für diesen Fall ist ein Software-Integritätstest.

DSS	Testaufwand
	$A_{I_{160}} + 2 \cdot A_{M_{160}} + 211 \cdot A_{M_{512}}$
mit Vorausber.	$A_{I_{160}} + 2 \cdot A_{M_{160}} + 154 \cdot A_{M_{512}}$

Tabelle 9: Aufwand des DSS-Testalgorithmus' ohne und mit Vorausberechnungen.

2.4 Aufwandsvergleich mit RSA

Das älteste bekannte Digitale Signaturverfahren RSA basiert auf der als praktisch unlösbares zahlentheoretisches Problem geltenden Primfaktorzerlegung langer Zahlen [40]. Da das Verfahren sich in den letzten Jahren - nicht nur in den USA - als "quasi-Standard" durchgesetzt hat, soll dessen Aufwand hier zum Vergleich herangezogen werden.

Das RSA-Signaturverfahren besitzt die folgenden Parameter:

- Einen öffentlichen, authentischen Exponenten e.
- Einen authentisch zu veröffentlichenden Testschlüssel $Ö_i = p_i \cdot q_i$, mit $p_i, q_i \in Prim$ und $p_i, q_i > 2^{250}$,$2^{511} < Ö_i < 2^{512}$.
- Den geheimzuhaltenden Signierschlüssel $G_i := e^{-1} \bmod \varphi(Ö_i)$.[16]

Auch hier muß eine Hashfunktion H vereinbart werden, mit der von jeder zu signierenden Nachricht N zunächst ein "Fingerabdruck" in der zulässigen Blocklänge, d.h. $H(N) < n_i$, berechnet wird. Die Signatur $Sig = S(N, G_i)$ einer Nachricht N wird vom Sender i dann folgendermaßen bestimmt:

$$Sig := H(N)^{G_i} \bmod Ö_i$$

Der Empfänger des Paares (N', Sig') überprüft die Signatur Sig', indem er mit $T=T(N', Sig', Ö_i)$ testet, ob gilt:

$$H(N') = Sig'^{e} \bmod Ö_i$$

Vernachlässigt man auch hier den Aufwand für die Berechnung von H(N), dann ist beim Signieren eine modulare Exponentiation (512 bit Exponenten- und Moduluslänge) durchzuführen. Dabei können die in den Abschnitten 2.1 erläuterten Exponentiationsalgorithmen verwendet werden. Da der Signierer die Faktorisierung des Modulus $Ö_i$ kennt, kann er diese durch zwei modulare Exponentiationen halber Moduluslänge (256 bit Exponenten- und Moduluslänge) und eine Anwendung des Chinesischen-Reste-Algorithmus (CRA) ersetzen [35, 25, 17, 20]. Dazu seien zunächst $G_{p_i} := e^{-1} \bmod (p_i - 1)$, $G_{q_i} := e^{-1} \bmod (q_i - 1)$. Dann gilt:

$$Sig_{p_i} := H(N)^{G_{pi}} \bmod p_i$$

$$Sig_{q_i} := H(M)^{G_{qi}} \bmod q_i$$

$$Sig = CRA(Sig_{p_i}, Sig_{q_i}, p_i, q_i).$$

Der geheimzuhaltende Schlüssel G_i wird so auf vier Komponenten erweitert: $G_i = (G_{pi}, G_{qi}, p_i, q_i)$.

---

16 $\varphi(Ö_i)$ bezeichnet die Eulersche-Phi-Funktion. Sie gibt die Anzahl der zu $Ö_i$ relativ primen Zahlen kleiner $Ö_i$ an, d.h. $\varphi(Ö_i) := (p_i - 1) \cdot (q_i - 1)$.

Der CRA erfordert etwa 1.5 modulare Multiplikationen, d.h. $A_{CRA} = 1.5 \cdot A_{M_{256}}$, damit hat der Signieralgorithmus im Mittel einen Aufwand von:

RSA-Signieraufwand $\mathbf{500 \cdot A_{M_{256}}}$ **(6)**

Der Empfänger muß beim Testen der Signatur eine volle Exponentiation (512 bit Modulus-länge, 512 bit Exponentenlänge) durchführen, da er die Primfaktoren des Modulus n nicht kennt:

RSA-Testaufwand $\mathbf{484\ A_{M_{512}}}$ **(7)**

Es wird daher vielfach vorgeschlagen, den öffentlichen Exponenten klein zu wählen, z.B. die 5. Fermatzahl $e = 2^{16}+1$ [25, 6, 29]. Bisher wurde allerdings nicht gezeigt, daß mit dieser Wahl keine Sicherheitseinbußen verbunden sind. Der Testaufwand reduziert sich in diesem Fall auf:

RSA-Testaufwand $\mathbf{(e=2^{16}+1)}$ $\mathbf{13 \cdot A_{M_{512}}}$ **(8)**

Die folgende Tabelle faßt die Ergebnisse der Aufwandsbetrachtungen in den Abschnitten 2.2, 2.3 und 2.4 übersichtsartig zusammen:

Verfahren	Signieraufwand	Testaufwand
DSS	$A_G+A_{I_{160}}+484 \cdot A_{M_{512}}+2 \cdot A_{M_{160}}$	$A_{I_{160}}+2 \cdot A_{M_{160}}+211 \cdot A_{M_{512}}$
DSS (mit Vor.)	$2 \cdot A_{M_{160}}$	$A_{I_{160}}+2 \cdot A_{M_{160}}+154 \cdot A_{M_{512}}$
RSA	$500 \cdot A_{M_{256}}$	$484 \cdot A_{M_{512}}$
RSA ($e = 2^{16}+1$)	$500 \cdot A_{M_{256}}$	$13 \cdot A_{M_{512}}$

Tabelle 10: Aufwandsvergleich zwischen den DSS- und RSA-Verfahren.

Mit einer eigenen Software-Implementierung des RSA- und des DSS-Verfahrens auf der Basis einer schnellen, in C und Maschinensprache kodierten modularen Langzahlarithmetik wurden bei Schneider & Koch für die Signier- und Testoperationen auf einem 80286-PC/AT (20 MHz) und einem 80486-PC (50 MHz) die in den Tabellen 11 und 12 zusammengefaßten Ergebnisse erzielt.
Die Gegenüberstellung zeigt, daß der Signieraufwand des DSS-Verfahrens mit Vorausberech-nungen um etwa drei Größenordnungen unter dem des RSA-Signieralgorithmus liegt. Der Signieraufwand des "reinen" DSS-Verfahrens ist jedoch etwa dreimal größer als der des "reinen" RSA-Verfahrens — ebensoviele modulare Multiplikationen bezüglich eines doppelt so langen Modulus.

Verwendet man beim RSA-Verfahren einen sehr kurzen (hier: 16 bit langen) Exponenten, liegt auch der Testaufwand erheblich unter dem des DSS-Verfahrens. Nicht aber, wenn der Exponent e zufällig gewählt wird: Dann ist der DSS-Testalgorithmus um den Faktor 2 schneller.

80286, 25 MHz	**Signieren** [sec]	**Testen** [sec]
DSS	4,04	1,86
DSS (mit Vor.)	ca. 0,002	1,40
RSA	1,24	3,95
RSA ($e = 2^{16}+1$)	1,23	0,12

Tabelle 11: Signier- und Testzeit einer effizienten Softwareimplementierung der DSS- und RSA-Verfahren auf einem 80286-PC/AT (25 MHz).

80486, 50 MHz	**Signieren** [sec]	**Testen** [sec]
DSS	1,16	0,54
DSS (mit Vor.)	ca. 0,0005	0,41
RSA	0,41	1,13
RSA ($e = 2^{16}+1$)	0,40	0,03

Tabelle 12: Signier- und Testzeit einer effizienten Softwareimplementierung der DSS- und RSA-Verfahren auf einem 80486-PC (50 MHz).[17]

17 Die Softwareimplementierung verwendet ausschließlich 16-bit-Operationen der 80x86-Mikroprozessoren. Eine zur Zeit in Entwicklung befindliche Version der Langzahlarithmetik wird die 32-bit-Operationen der Mikroprozessiren 80386 und 80486 nutzen. Dadurch ist eine Geschwindigkeitssteigerung um den Faktor 3 zu erwarten.

3 Sicherheit

Für das der Sicherheit des DSS zugrundeliegende zahlentheoretische Problem der Berechnung diskreter Logarithmen ist bis heute kein Beweis geführt worden, daß tatsächlich kein Lösungsalgorithmus existiert, der polynomialen Aufwand besitzt. Gleiches gilt für das sogenannte "Faktorisierungsproblem" (d.h. die Zerlegung sehr langer Zahlen in ihre Primfaktoren), auf dem das RSA-Verfahren beruht. Die Zahlentheoretiker sind sich recht sicher, daß beide Probleme in diesem Sinne gleich schwierig und mit hoher Wahrscheinlichkeit praktisch unlösbar sind, da sie seit vielen Jahren intensiv untersucht werden.

Dies allein garantiert jedoch noch nicht die Sicherheit des Kryptoverfahrens: Es muß weiter (möglichst bewiesenermaßen) sichergestellt sein, daß das Fälschen einer Signatur äquivalent dem Lösen des zugrundeliegenden mathematischen Problems ist - beim DSS-Verfahren der Berechnung des diskreten Logarithmus, beim RSA-Verfahren der Faktorisierung des Modulus. Auch ein solcher Beweis wurde bis heute für keines der beiden Verfahren geführt.

Im praktischen Einsatz ist für die Fälschungssicherheit des DSS-Verfahrens außerdem eine authentische Veröffentlichung der Testschlüssel und der Moduli p und q wesentlich. Können diese Parameter modifiziert werden, sind Signaturen leicht zu fälschen. Die Authentizität der verwendeten Schlüssel muß durch die Einrichtung einer Zertifizierungsinstanz und durch die Festlegung eines Zertifizierungsverfahrens sichergestellt werden (z.B. X.509, [21]).

Speziell für das DSS-Verfahren ist ein weiterer Aspekt von besonderer Bedeutung: Da hier die Zufälligkeit und die einmalige Verwendung des Parameters k notwendige Bedingung für die Sicherheit des Verfahrens sind, müssen alle Teilnehmer über einen sicheren Zufallszahlengenerator verfügen. Ein (passiver) Angreifer darf nicht in der Lage sein, aus "alten" Werten $r = ((g^k \bmod p) \bmod q)$ den Generator zu rekonstruieren. Dies kann nur durch eine echte, gleichverteilte Zufallsquelle oder einen sicher unvorhersagbaren Pseudozufallszahlengenerator garantiert werden; siehe z.B. [4, 32].

Ronald Rivest [38, 39] und Martin Hellman [24] weisen in ihrer Kritik an dem Standardentwurf DSS auf weitere Punkte hin:

- Gemeinsame öffentliche Moduli sind gefährlich, da ein erfolgreicher Angriff alle Teilnehmer trifft. Daher sollten Teilnehmer eigene Moduli wählen und veröffentlichen.
- Es existieren "trap-door"-Primzahlen, bei denen die Berechnung des diskreten Logarithmus in vernünftiger Zeit möglich ist. Sie sind ggf. schwer als solche zu erkennen.
- Die Veröffentlichung der Moduli p und q könnte es einem Angreifer erleichtern, Unterschriften zu fälschen. Eine Variante des DSS-Verfahrens, das als Moduli Produkte großer Primzahlen verwendet, besäße diese mögliche Schwäche nicht.
- Das DSS-Verfahren ist eine Modifikation des Signaturverfahrens von El Gamal. Es ist eine bisher ungenügend untersuchte Frage, ob diese Variante einfacher zu brechen ist - d.h. ohne das Problem der Berechnung des Diskreten Logarithmus zu lösen.
- Es könnte sich als problematisch erweisen, daß g in der Größenordnung von q (160 bit) statt p (512 bit) gewählt wird.

In einer Antwort auf die in der sechsmonatigen Diskussionsphase eingegangene Kritik am DSS-Entwurf kündigte NIST auf der Crypto '92 für August 1992 einen überarbeiteten Standard an, in dem Moduluslängen bis 1024 bit zugelassen sein werden [1, 44]. Die Überarbeitung ist bisher nicht veröffentlicht; vor Ende 1993 ist nach offiziellen Aussagen mit einem Abschluß der Standardisierung nicht zu rechnen.

4 Zusammenfassung

Der von NIST vorgeschlagene DSS ist als Digitales Signaturverfahren durchaus für praktische Zwecke geeignet: Sowohl die Sicherheit als auch die Geschwindigkeit genügen vielen Anwendungen, in denen die erzeugten Signaturen und die verwendeten Schlüssel eine (z.B. auf ein Jahr) begrenzte Gültigkeit besitzen. Der Aufwand der Algorithmen liegt in der Größenordnung derer des RSA-Verfahrens.

Es wurde gezeigt, daß durch Verwendung unterschiedlicher Algorithmen für die modulare Exponentiation und umfassende Vorausberechnungen der Signieraufwand des DSS-Verfahrens erheblich gesenkt werden kann. Ist genügend Speicher verfügbar, dann ist so eine sehr schnelle Software-Implementierung möglich. Laufzeiten einer effizienten Implementierung für 80x86-Mikroprozessoren in C und Maschinensprache wurden angegeben.

Eine Realisierung muß jedoch sowohl hinsichtlich des verwendeten Zufallszahlengenerators, der kollisionsfreien Hashfunktion als auch der eingesetzten Zertifizierungsverfahren zur Authentisierung der öffentlichen Testschlüssel mit Bedacht erfolgen: Ein Sicherheitsverfahren ist schließlich immer nur so sicher wie sein schwächstes Glied. In dieser Hinsicht sind kryptographische Analysen des SHS [43] und des angekündigten Pseudozufallszahlengenerators abzuwarten.

Nur eingeschränkt geeignet erscheint der DSS-Entwurf als nationaler oder gar internationaler Standard: Da das Verfahren erst wenig untersucht ist, sollte sicherheitshalber mit einer kurzen "Lebensdauer" gerechnet werden. NIST versucht diesem Problem Rechnung zu tragen: Es hat eine Überprüfung des Standards alle fünf Jahre angekündigt [44].

Benötigt man Signaturen, die über viele Jahre ihre Gültigkeit behalten, sollten etwas aufwendigere, aber dafür bewiesen sichere Verfahren gewählt werden, wie beispielsweise das GMR-Signaturverfahren mit einer Damgård-Hashfunktion, für die die Äquivalenz von Fälschung bzw. Kollisionsfindung und Faktorisierung nachgewiesen sind [23, 10, 18, 17].

Dank

Die Ergebnisse der Untersuchung unterschiedlicher Exponentiationsalgorithmen in Abschnitt 2.1 sind im wesentlichen meiner Diplomarbeit entnommen [19], die von Birgit Pfitzmann betreut wurde. Ich verdanke ihr viele Ideen, Anregungen und kritische Diskussionen. Mein Dank für konstruktive Kommentare zu diesem Papier und Hinweise auf jüngere Veröffentlichungen gilt weiter Michael Waidner, Manfred Böttger, Patrick Horster, Hubert Weiler und Ralf Aßmann sowie einigen ungenannten Gutachtern.

Literatur

[1] John A. Adam: "Cryptography = privacy?", Special Report Data Security, IEEE Spectrum, August 1992, S. 29-35.

[2] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman: "The Design and Analysis of Computer Algorithms", Addison Wesley, Massachusetts, 1974.

[3] E. Brickell, D.M. Gordon, K.S. McCurley, D. Wilson: "Fast Exponentiation with Precomputation", Eurocrypt '92, Extended Abstracts, Ungarn, 3/1992, S. 193-201.

[4] L. Blum, M. Blum, M. Schub: "A Simple Unpredictable Pseudo-Random Number Generator", SIAM J. Computing, 15/2, 1986, S. 364-383.

[5] Jurjen Bos, Mattijs Coster: "Addition Chain Heuristics", Proc. of Crypto '89, LNCS Nr.435, Springer, Berlin, 1989, S. 377-386.

[6] Dieter Bong, Christoph Ruland: "Optimized Software Implementation of the Modular Exponentiation on General Purpose Microprocessors", Computers and Security, Nr.8, 1989, S. 621-630.

[7] Gilles Brassard: "Modern Cryptology", A Tutorial, LNCS 325, Springer, 1988.

[8] Ernest F. Brickell: "A Fast Modular Multiplication Algorithm with Applications to Two Key Cryptography", Proc. of Crypto '82, S. 51-60.

[9] Paul G. Comba: "Exponential Cryptosystems on the IBM-PC", IBM Systems Journal, Bd.29, Nr.4, 1990, S. 526-538.

[10] Ivan Bjerre Damgard: "Collision free Hash Functions and Public Key Signature Systems", Eurocrypt 1987, LNCS 304, Springer, 1988, S. 203-216.

[11] Donald W. Davies, Wyn L. Price: "Security for Computer Networks", 2. Auflage, John Wiley & Sons Ltd., Chichester, 1989.

[12] Dorothy Elizabeth Robling Denning: "Cryptography and Data Security", Addison Wesley, Massachusetts, 1982.

[13] Whitfield Diffie, Martin E. Hellman: "New Directions in Cryptography", IEEE Transactions on Information Theory, Bd. IT-22, Nr.6, 1976, S. 644-654.

[14] Digital Signature Standard (DSS), Federal Information Processing Standards (FIPS) Publication XX, Draft, National Institute of Standards and Technology (NIST) 19.8.1991.

[15] Stephen R. Dussé, Burton S. Kaliski (Jr.): "A Cryptographic Library for the Motorola DSP 56000", Proc. of Crypto '90, LNCS 473, Springer, S. 230-244.

[16] Taher El Gamal: "A Public Key Cryptosystem an Signature Scheme Based on Discrete logarithms", IEEE Trans. on Inform. Theory, Bd. IT-31, Nr.4, 7/1985, S. 469-472.

[17] Dirk Fox, Birgit Pfitzmann: "Effiziente Software-Implementierung des GMR-Signatursystems", Proc. of VIS '91, Verläßliche Informationssysteme, Informatik Fachberichte Nr.271, Springer, Heidelberg, 1991, S. 329-345.

[18] Dirk Fox: "Implementierung eines sicheren digitalen Signatursystems", Studienarbeit am Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe, 5/1990.

[19] Dirk Fox: "Effiziente Softwareimplementierung asymmetrischer Kryptosysteme und der zugrundeliegenden modularen Langzahlarithmetik", Diplomarbeit am Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe, 1991.

[20] Markus Frisch: "Ein Überblick zum Thema RSA", Draft Report 91/15, E.I.S.S., Universität Karlsruhe, 1991.

[21] Walter Fumy: "Sicherheitsstandards für offene Systeme", Datenschutz und Datensicherung (DuD), 6/91, S. 288-295.

[22] J.K. Gibson: "A Generalisation of Brickell's Algorithm for Fast Modular Arithmetic", BIT 28, 1988, S. 755-764.

[23] Shafi Goldwasser, Silvio Micali, Ronald L. Rivest: "A Digital Signature Scheme Secure against Adaptive Chosen Message Attacks", SSIAM Journal on Computing, Bd.17, Nr.2, 1988, S. 281-308.

[24] Martin E. Hellman: "Response to NIST's Proposal", Communication of the ACM, Bd.35, Nr.7, 7/1992, S. 47-49.

[25] Achim Jung: "Implementing the RSA Cryptosystem", Computers and Security, Bd.6, Nr.4, 1987, S. 342-350.

[26] Burt Kaliski: "A letter to NIST commenting on the proposed Digital Signature Standard (DSS)", RSA Data Security Inc., Newsgroups: sci.crypt, 4.11.1991.

[27] Hans-Joachim Knobloch, Patrick Horster: "Eine Krypto-Toolbox für Smartcards", Datenschutz und Datensicherung (DuD), 7/92, S. 353-361.

[28] Donald Erwin Knuth: "The Art of Computer Programming", Bd.2: "Seminumerical Algorithms", 2.Auflage, Addison-Wesley, Massachusetts, 1981.

[29] Denis Laurichesse, Laurent Blain: "Optimized Implementation of RSA Cryptosystem", Computers and Security, 10/1991, S. 263-267.

[30] P. Lippitsch, R. Posch: "PC-RSA: A Cryptographic Toolkit for MS-DOS", Proc. of VIS '91, Verläßliche Informationssysteme, Informatik Fachberichte 271, Springer, Heidelberg, 1991, S. 346-354.

[31] John D. Lipson: "Elements of Algebra and Algebraic Computing", Benjamin/Cummings, Massachusetts, 1981.

[32] Silvio Micali, Claus P. Schnorr: "Efficient, Perfect Polynomial Random Number Generators", Journal of Cryptology, Bd.3, Nr.3, 1991, S. 157-172.

[33] Peter L. Montgomery: "Modular Multiplikation without Trial Division", Mathematics of Computation, Bd.44, Nr.170, 4/1985, S. 519-521.

[34] Hikary Morita: "A fast Modular Multiplication Module for Smart Cards", Proceedings of Auscrypt '90, LNCS 453, Springer, 1991, S. 406-409.

[35] J.-J. Quisquater, C. Couvreur: "Fast Deciphering Algorithm for RSA Public-Key Cryptosystem", Electronic Letters, Bd.18, Nr.21, 10/1982, S. 905-907.

[36] Ronald L. Rivest: "The MD4 Message Digest Algorithm", Advances in Cryptology, Crypto '90 Proceedings, Springer, Heidelberg, 1991, S. 303-311.

[37] Ronald L. Rivest: "The MD4 Message-Digest Algorithm", Network Working Group, MIT Laboratory for Computer Science and RSA Data Security, Inc., 4/1992.

[38] Ronald L. Rivest: "Response to NIST's Proposal", Communication of the ACM, Bd.35, Nr.7, 7/1992, S. 41-47.

[39] Ronald L. Rivest: "The Debate over the U.S. Digital Signature Standard", Special Report Data Security, IEEE Spektrum, August 1992, S. 34-35.

[40] Ronald L. Rivest, Adi Shamir, Leonard Adleman: "A Method for obtaining Digital Signatures and Public Key Cryptosystems", Communications of the ACM, Bd.21, Nr.2, 1978, S. 120-126.

[41] Jörg Sauerbrey, Andreas Dietel: "Ressource Requirements for the Application of Addition Chains in Modulo Exponentiation", Extended Abstracts, Eurocrypt '92, Ungarn, 3/1992, S. 159-167.

[42] Claus P. Schnorr: "Efficient Signature Generation by Smart Cards", Journal of Cryptology, Bd.4, Nr.3, 1991, S.161-174.

[43] Secure Hash Standard (SHS), Federal Information Processing Standards (FIPS) Publication, Draft, National Institute of Standards and Technology (NIST) 31.1.1992.

[44] Miles E. Smid, Dennis K. Branstad (NIST): "Response to Comments on the NIST proposed Digital Signature Standard", Draft, Crypto '92, 17.8.1992.