GI-Fachtagung Verläßliche Informationssysteme (VIS'91), März 1991, Darmstadt, Informatik-Fachberichte 271, Springer-Verlag, Heidelberg 1991, 329-345

Effiziente Software-Implementierung des GMR-Signatursystems

Dirk Fox, Birgit Pfitzmann
Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe (TH)
Postfach 6980, D-W7500 Karlsruhe 1

Kurzfassung

1984 wurde von Goldwasser, Micali und Rivest erstmalig ein gegen adaptive, aktive Angriffe beweisbar sicheres Signatursystem (GMR) vorgestellt. Bis dahin galt ein solches System als Paradoxon – den scheinbaren Widerspruch lösten die Autoren in ihrer Veröffentlichung.

Nach einer kurzen Einführung in die Funktionsweise von GMR arbeiten wir die schon bekannten Implementierungshinweise aus und erläutern eigene effizienzsteigernde Verbesserungen.

Diese nutzten wir für eine Implementierung auf MC680xy-Rechnern. Die von uns verwendete Langzahlarithmetik ist in Assembler geschrieben; die GMR-spezifischen Funktionen wurden im wesentlichen in Pascal entwickelt.

Wir stellen relativ genaue allgemeine Aufwandsbetrachtungen an und erläutern die Meßergebnisse unserer Implementierung. Beide Angaben werden mit Werten für das verbreitete RSA-Signatursystem verglichen. Dabei stellt sich heraus, daß GMR nicht nur praktikabel, sondern in manchen Fällen sogar effizienter ist als RSA (in reiner Form). Stellt man hohe Sicherheitsanforderungen, ist GMR demnach eine empfehlenswerte Alternative.

1 Einleitung

In zunehmendem Maße findet Kommunikation über Rechnernetze statt; insbesondere finanzielle Transaktionen werden bereits weitgehend elektronisch abgewickelt. Dabei fallen übliche, geradezu selbstverständlich gewordene Kontrollen der Echtheit und Vertrauenswürdigkeit von Nachrichten weg: die Handschrift, der bekannte Überbringer, der versiegelte Umschlag und nicht zuletzt die eigenhändige Unterschrift. Daher ist die Entwicklung technischer Verfahren notwendig, die (mit geringen „Kosten“) ein Maximum an Verläßlichkeit bieten.

Verläßlichkeit meint meist Nachweisbarkeit, in Streitfällen sogar gerichtliche Rekonstruierbarkeit von Kommunikationsabläufen. Die (effiziente) Sicherstellung von Überprüfbarkeit der Herkunft und Unverfälschbarkeit einer Nachricht spielt dabei eine zentrale Rolle, zum Beispiel bei Bankgeschäften. Im Alltag dient dazu heute die eigenhändige Unterschrift; Verfahren, die diese Authentisierung beim digitalen Nachrichtenaustausch leisten, werden daher ***digitale Signatursysteme*** genannt. Ein (konventionelles) digitales Signatursystem besteht im wesentlichen aus drei Komponenten:

1. Einem Generieralgorithmus G, der ein Schlüsselpaar $(g,ö)$ liefert. Der öffentliche Schlüssel $ö$ wird im allgemeinen in ein Verzeichnis eingetragen; g geheimgehalten.
2. Einem Signieralgorithmus S zum Unterschreiben von Nachrichten, der den geheimen Schlüssel g verwendet.
3. Einem Testalgorithmus T, der die Signatur einer Nachricht anhand des öffentlichen Schlüssels $ö$ überprüft.

Selbst bei Kenntnis aller drei Algorithmen muß ohne den geheimen Schlüssel *g* die Fälschung einer Signatur ***praktisch unmöglich***[1] sein. Erwünscht ist ein Signatursystem, das selbst bei der stärksten Angriffsform, dem ***adaptiven***, ***aktiven Angriff***,[2] das Fälschen auch nur einer einzigen (neuen) Unterschrift praktisch unmöglich macht.

Das bekannteste digitale Signatursystem – kurz ***RSA*** genannt – wurde 1978 von Rivest, Shamir und Adleman vorgestellt [RSA_78]. RSA bietet gegen aktive Angriffe allerdings keine Sicherheit [Davi_82, Denn_84]. Man versucht, diese Schwäche von RSA durch Redundanzprädikate, Hash-Funktionen o.ä. zu beheben (siehe z.B. [Jung_87]). Über die Sicherheit dieser Modifikationen ist kaum etwas bekannt.

Das von Goldwasser, Micali und Rivest entwickelte digitale Signatursystem – im folgenden nach den Autoren ***GMR*** genannt – bietet selbst gegen adaptive, aktive Angriffe ***beweisbar*** Fälschungssicherheit [GoMR_84, GoMR_88].

Da die Komplexitätstheorie bis heute keine Möglichkeit kennt, die praktische Unmöglichkeit, ein derartiges System mit polynomialem Aufwand zu brechen, *vollständig* zu beweisen, ist man auf kryptographische Annahmen angewiesen. Man versucht jedoch, mit möglichst einer einzigen, plausiblen und durch lange Erfahrung gestützten Annahme (z.B. der praktischen Unlösbarkeit eines zahlentheoretischen Problems) auszukommen.

Für die in Kapitel 3 vorgestellte spezielle Implementierung von GMR, die die praktische Unmöglichkeit des Faktorisierens großer Zahlen annimmt, wird in [GoMR_88] ***bewiesen***, daß diese kryptographische Annahme die Unfälschbarkeit des Verfahrens selbst bei adaptiven, aktiven Angriffen impliziert. RSA ist in diesem Sinne auch bei schwächeren Angriffen nicht bewiesenermaßen sicher.[3]

In Kapitel 2 wird zunächst die Funktionsweise von GMR skizziert. Kapitel 3 erläutert die wichtigsten Implementierungshinweise, die von den Autoren selbst sowie Goldreich und Levin gegeben wurden. In Kapitel 4 werden die von uns entwickelten Verbesserungen dargestellt, die das Testen der Signatur um den Faktor zwei, das Signieren bei kurzen Nachrichten um den Faktor vier, bei langen noch erheblich mehr[4] beschleunigen. Kapitel 5 und 6 enthalten die theoretischen Aufwandsbetrachtungen und Messungen, insbesondere Vergleiche mit RSA. (Für RSA werden Werte aus [BoRu_89], [Pohl_90] und eigenen Messungen herangezogen.)

2 Funktionsweise

Grundidee

RSA benötigt sogenannte ***Einwegfunktionen*** mit ***Geheimnis***[5] (trapdoor one-way functions, [DiHe_76, RSA_78]) *f*; Funktionen also, die nur mit Kenntnis eines geheimzuhaltenden Schlüssels *g* invertiert werden können:

---

[1] Unter ***praktischer Unmöglichkeit*** ist hier zu verstehen, daß eine Lösung zwar theoretisch möglich ist, aber mit realistischer Rechenleistung in vernünftiger Zeit nur eine exponentiell kleine Wahrscheinlichkeit besitzt.

[2] Dieses Modell nimmt an, daß ein Angreifer für eine polynomiale Anzahl von Nachrichten seiner Wahl Unterschriften erhält; die Wahl kann dabei *adaptiv*, d.h. in Abhängigkeit von den bereits gewählten Nachrichten und den zugehörigen Unterschriften erfolgen.

[3] Es ist z.B. denkbar, daß RSA auch ohne einen polynomialen Faktorisierungsalgorithmus gebrochen wird.

[4] Ist *l* die Länge des Modulus in Bit, etwa $l = 512$, so strebt der Faktor für lange Nachrichten näherungsweise gegen $l/1{,}2$ (bei optimalem Divisionsverfahren).

[5] Genau genommen handelt es sich hierbei um Funktionenfamilien, aus denen der Schlüssel *ö* eine Funktion $f_{ö}$ auswählt. Wir verzichten auf den Index, um die Lesbarkeit zu erhöhen. Gleiches gilt für die im folgenden einzuführenden klauenfreien Permutationenpaare.

Der Sender unterschreibt die Nachricht m mit $Sig := f^{-1}(g, m)$; der Empfänger testet $f(Sig)=m$. (Der zur Bestimmung von f notwendige Schlüssel $\ddot{o}$ steht z.B. in einem öffentlichen Verzeichnis.)

Bei GMR wird nicht mehr die Nachricht als Funktionswert verwendet, sondern eine zufällig gewählte, nur einmal verwendbare ***Referenz Ref***. Die Nachricht legt dafür die Wahl der Funktion f fest – jeder Nachricht m ist genau eine Funktion f_m zugeordnet.

Dadurch wird erreicht, daß ein aktiver Angreifer seine Unterschrift jedesmal zu einer anderen Referenz erhält. Ein adaptives „Variieren über m“ (wie beim adaptiven, aktiven Angriff) wird hier vereitelt, da zugleich eine Variation über *Ref* stattfindet, die vom Angreifer nicht beeinflußt werden kann. Die Referenz *Ref* ist daher authentisiert bekanntzugeben.

Dieses Vorgehen hat zwei weitere positive (Neben-) Effekte: So existiert f_m für beliebig lange Nachrichten m, es ist also keine Aufteilung in Blöcke oder der Einsatz einer Hashfunktion notwendig. Außerdem ist die Möglichkeit eines „Replay“-Angriffs (bei gleichem Empfänger) ausgeschlossen, da dieser die Referenz wiedererkennen und so den Angriff aufdecken würde.

Kryptographische Annahme

Notwendig für die Sicherheit dieses Verfahrens ist die Existenz ***klauenfreier Permutationenpaare*** mit Geheimnis (claw-free permutation pairs); sie haben folgende Eigenschaft (genaue Definition in [GoMR_88]):

Es ist praktisch unmöglich, ohne Kenntnis des Geheimnisses g zum klauenfreien Permutationenpaar (F_0, F_1) ein Paar (x, y) zu finden mit $F_0(x)=F_1(y)$.

Diese kryptographische Existenzannahme ist schärfer als die Forderung von Einwegfunktionen mit Geheimnis: Mit einem klauenfreien Permutationenpaar liegen zugleich zwei Einwegpermutationen mit Geheimnis vor.

In [GoMR_88] wird bewiesen, daß die Klauenfreiheit der Permutationenpaare die praktische Unmöglichkeit impliziert, das Verfahren mit größerer Wahrscheinlichkeit als Raten zu brechen.[6]

Signier- und Testfunktion

Gibt es klauenfreie Permutationenpaare (F_0, F_1), so läßt sich zu jeder Nachricht m eine Funktion f_m konstruieren, indem F_0 und F_1 bitweise in Abhängigkeit von m komponiert werden.

Beispiel: m=100101, $f_{100101}(x) := F_1(F_0(F_0(F_1(F_0(F_1(x))))))$.

Mit Kenntnis des Geheimnisses g lassen sich die Urbilder $F_0^{-1}(x)$ und $F_1^{-1}(x)$ berechnen und damit auch $f_m^{-1}(Ref)$: Der Sender der Nachricht m signiert mit $f_m^{-1}(Ref) =: Sig$. Die Auswertungsrichtung von m keht sich dabei um:

Beispiel: m=100101, $f_{100101}^{-1}(x) := F_1^{-1}(F_0^{-1}(F_1^{-1}(F_0^{-1}(F_0^{-1}(F_1^{-1}(x))))))$.

Der Empfänger entnimmt dem öffentlichen Verzeichnis den Schlüssel $\ddot{o}$, erhält damit F_0, F_1 und *Ref* und testet $f_m(Sig) = Ref$.

Präfixfreie Abbildung

Die Komposition von F_0 und F_1 birgt jedoch eine Gefahr: Der „zurückrechnende“ Empfänger, der die Signatur mit f_m testet, erhält mit jeder Anwendung von F_0 und F_1 wieder eine gültige Unterschrift zu der um jeweils das letzte Bit verkürzten Nachricht.

---

[6] Ein entsprechender Beweis konnte, wie erwähnt, für RSA bisher nicht geführt werden.

Beispiel: $F_1(Sig_{100101}) = F_1(f_{100101}^{-1}(Ref)) = f_{10010}^{-1}(Ref) = Sig_{10010}$.

Dies läßt sich verhindern, indem die Form einer „gültigen Nachricht“ durch eine ***präfixfreie Abbildung*** *präf(•)* festgelegt wird: Kein Präfix einer so abgebildeten Nachricht darf wiederum eine präfixfreie Abbildung einer anderen Nachricht sein. Die ursprüngliche Nachricht erhält so einen „Gültigkeitsvermerk“, der bei Verlust des (oder der) letzten Bits automatisch erlischt. Unterschrieben wird nun *präf*(*m*) mit $f_{präf(m)}^{-1}(Ref)$[7].

Authentisierung der Referenzen

Die zufälligen Referenzen *Ref* sind, wie schon erwähnt, natürlich ihrerseits wieder zu authentisieren – andernfalls könnte ein Angreifer zu einer Nachricht *m* seiner Wahl eine Unterschrift *Sig* zufällig wählen und von dort mittels F_0 und F_1 die Referenz $Ref := f_{präf(m)}(Sig)$ berechnen.

Dies könnte dadurch geschehen, daß eine bestimmte Anzahl von Referenzen bekanntgegeben und Signatur für Signatur „verbraucht“ werden. Dieses Vorgehen hat den Nachteil, daß die öffentliche Referenzenliste sehr lang wird.[8] Goldwasser, Micali und Rivest schlagen daher folgendes Verfahren vor:

Nur eine Referenz r_ε wird veröffentlicht. 2^b für Nachrichtensignaturen (im weiteren kurz ***N-Signaturen*** genannt) vorgesehene Referenzen R_0, R_1, ... werden nun an die Blätter eines Binärbaumes der Tiefe *b* angehängt; jeder Knoten des Baumes enthält wiederum eine Referenz r_j (diese „Hilfsreferenzen“ dienen der Authentisierung der Referenzen R_i). Der Baum wird im folgenden ***Referenzenbaum*** genannt (Abbildung 2.1).

Für jede Signatur wird nun ein ***Signaturkopf*** berechnet, der die aktuelle Referenz R_i bezüglich des öffentlichen r_ε authentisiert. Dies geschieht folgendermaßen:

Beginnend mit der (öffentlichen) Referenz r_ε, dem Wurzelknoten des Referenzenbaumes also, werden alle Hilfsreferenzen r_j auf dem Pfad von r_ε nach R_i benutzt, um ihre beiden Nachfolgeknoten r_{j0} und r_{j1} zu signieren (Abbildung 2.2): r_{j0} und r_{j1} werden dazu (mit Trennzeichen) konkateniert, als „Nachricht“ interpretiert und präfixfrei abgebildet. Dann wird die Signatur $KSig := f_{präf(r_{j0}|r_{j1})}^{-1}(r_j)$ gebildet (diese Knotensignaturen werden im folgenden kurz ***K-Signaturen*** genannt). Schließlich wird die angehängte Referenz R_i mit $RSig := f_{präf(R_i)}^{-1}(r_i)$ authentisiert; diese Referenzsignatur wird im folgenden als ***R-Signatur*** bezeichnet.[9]

Beim Testen ist also nicht nur die N-Signatur zu überprüfen, sondern auch die *b* K-Signaturen zu sämtlichen Knoten r_j des Pfades durch den Referenzenbaum und die R-Signatur der Referenz R_i. Der dadurch entstehende zusätzliche Aufwand fällt jedoch, wie die Aufwandsabschätzungen in Kapitel 5 zeigen, bei längeren Nachrichten kaum ins Gewicht.

---

[7] Es ist anschaulich klar, daß es nun praktisch unmöglich ist, f-Klauen zu finden: Jede f-Klaue $f_{m_1}(Sig_1)=f_{m_2}(Sig_2)$ enthält eine F-Klaue an der ersten Stelle, an der sich m_1 und m_2 unterscheiden; z.B. folgt aus $f_{100101}(Sig_1)=f_{101110}(Sig_2)$, daß $F_0(f_{101}(Sig_1))=F_1(f_{110}(Sig_2))$. Umgekehrt gilt, daß nie zwei Signaturen (unterschiedlicher Nachrichten m_1, m_2) zur selben Referenz erfolgen dürfen, da $(f_{m_1}^{-1}(Ref), f_{m_2}^{-1}(Ref))$ eine f-Klaue bilden: $f_{m_1}(f_{m_1}^{-1}(Ref))=f_{m_2}(f_{m_2}^{-1}(Ref))=Ref$. Eine solche würde das Faktorisieren ohne Kenntnis des Geheimnisses ermöglichen.

[8] Für diesen Fall ist die Sicherheit von GMR außerdem nicht bewiesen; wir sehen auch nicht, wie sie bewiesen werden könnte.

[9] Für K- und R-Signaturen ist ein anderes Schlüsselpaar $(g,ö)$ zu verwenden als für die N-Signaturen; dies wird im Sicherheitsbeweis benötigt [GoMR_88]. Im folgenden wird diese Unterscheidung aus Übersichtlichkeitsgründen meist nicht erwähnt, z.B. schreiben wir einfach *p* und *q*, obwohl es für N-Signaturen andere sind als für K- und R-Signaturen. Welche gemeint sind, ist aus dem Kontext klar.

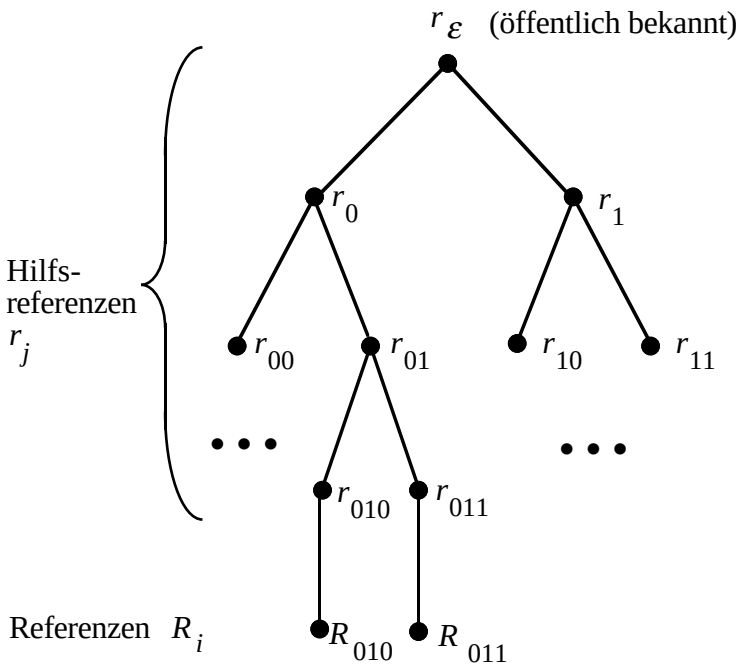

Abbildung 2.1 Referenzenbaum

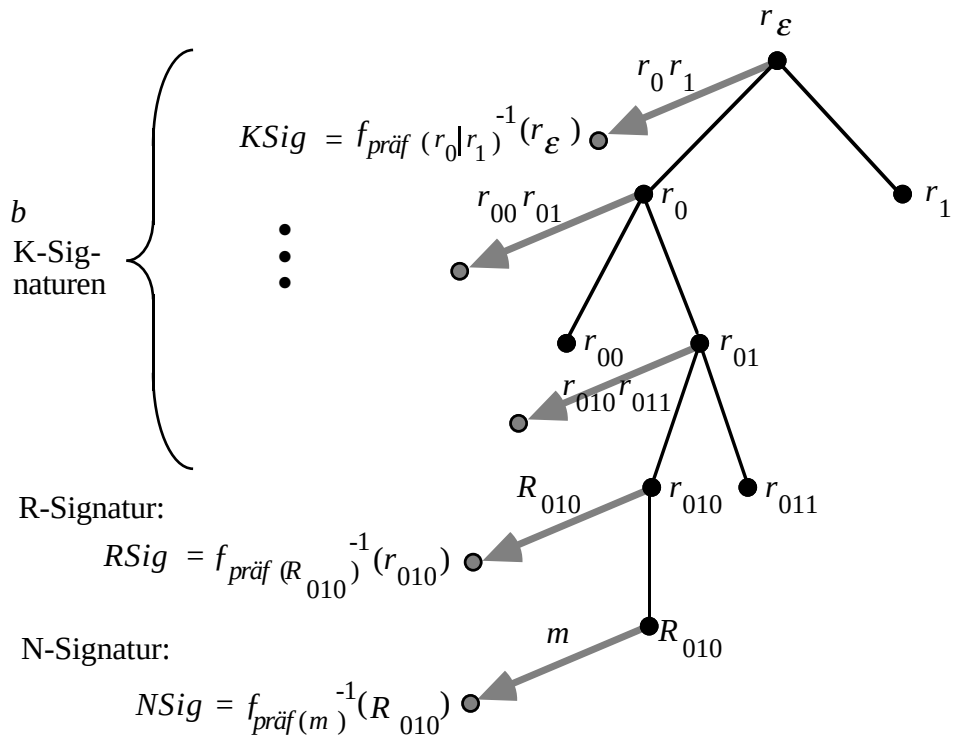

Abbildung 2.2 Signatur von m bezüglich Referenz R_2

3 Implementierung

Klauenfreie Permutationenpaare

Als klauenfreie Permutationenpaare werden von Goldwasser, Micali und Rivest die folgenden Funktionen vorgeschlagen:[10]

$$F_0(x) := \begin{cases} x^2 \bmod n, \text{ falls } (x^2 \bmod n) < n/2 \\ -x^2 \bmod n \text{ sonst.} \end{cases}$$

$$F_1(x) := \begin{cases} 4 \cdot x^2 \bmod n, \text{ falls } (4 \cdot x^2 \bmod n) < n/2 \\ -4 \cdot x^2 \bmod n \text{ sonst.} \end{cases}$$

Der Modulus n ist dabei speziell zu wählen: n muß eine ***Blum-GMR-Zahl*** sein, d.h. folgende Eigenschaften besitzen: $n=p \cdot q$ mit p,q prim und $p \equiv 3$, $q \equiv 7 \bmod 8$.[11] Der Definitionsbereich von F_0 und F_1 ist $D_n=\{x| \left(\frac{x}{n}\right)=1, x < n/2\}$[12].

Die Klauenfreiheit dieser speziellen Permutationenpaare beruht auf der kryptographischen Annahme, daß das Faktorisieren großer Blum-Zahlen praktisch unmöglich ist, d.h. alle Algorithmen zur Lösung dieses Problems in vernünftiger Zeit selbst mit größter realistischer Rechenleistung keine wesentlich größere Erfolgswahrscheinlichkeit besitzen als Raten.

Für diese Wahl von F_0 und F_1 ist ***bewiesen*** [GoMR_88], daß die praktische Unlösbarkeit des Faktorisierungsproblems die Klauenfreiheit der Permutationenpaare und, darauf aufbauend, die Fälschungssicherheit des gesamten Systems GMR impliziert.

Der öffentliche Schlüssel $\ddot{o}$ enthält n (genauer: zwei Werte n_f und n_g, siehe [9]) und r_ε, der geheime Schlüssel g besteht aus den Primfaktoren p und q von n: $\ddot{o}=(n,r_\varepsilon)$, $g=(p,q)$. Der geheime Schlüssel ermöglicht die Umkehrung von F_0 und F_1 durch modulares Wurzelziehen:

> Sei zunächst x aus D_n und $y=x^2 \bmod n$. Berechne $x_p:=y^{(p+1)/4} \bmod p$, $x_q:=y^{(q+1)/4} \bmod q$ und $x_n:=\text{CRA}(x_p,x_q)$. Dann ist x_n quadratischer Rest, und es gilt: $x=x_n$, falls $x_n<n/2$ und $x=-x_n \bmod n$ sonst.[13] Falls $y=-x^2 \bmod n$, ist dieselbe Rechnung mit $-y$ durchzuführen.[14]

Für F_1 ist zusätzlich einmal modulo n durch 4 zu dividieren.

---

[10] Das Zeichen „<“ auf Zahlen modulo n ist auf die Standarddarstellung durch $0,\ldots,n-1$ bezogen.

[11] Von ***Blum-Zahlen*** n [Blum_82] wird gefordert, daß $n=p \cdot q$ mit $p \equiv 3$ und $q \equiv 3 \bmod 4$. Diese Eigenschaft besitzt etwa die Hälfte aller Primzahlen. Jede Blum-GMR-Zahl ist eine Blum-Zahl; die geforderten Blum-GMR-Eigenschaften besitzt jeweils ein Viertel aller Primzahlen.

[12] $\left(\frac{x}{n}\right)$ bezeichnet das ***Jakobi-Symbol*** von x bezüglich n (s. z.B. [Hors_85]).

[13] Man beachte, daß bei Blum-Zahlen $p+1$ und $q+1$ durch 4 teilbar sind. Der Nachweis gelingt leicht mit Hilfe des ***Euler-Kriteriums***: Ist x quadratischer Rest mod n, dann gilt $x^{(p-1)/2}=\left(\frac{x}{p}\right)=1 \bmod p$, also $y^{(p+1)/4}=x^{2(p+1)/4}=x^{(p+1)/2}=x \cdot x^{(p-1)/2}=x \bmod p$. Gleiches gilt modulo q, also ist $x_n=x$. Ist x aus D_n kein quadratischer Rest, so ist $-x$ einer, und es gilt auch $y=-x^2$. Somit folgt wie oben $x_n=-x$. Da zudem $x<n/2$, also $x_n=-x \geq n/2$, folgt die Behauptung.

[14] Ist $y=F_0(x)$, aber nicht $y=x^2$, so ist $y=-x^2$ (die Fälle können z.B. daran unterschieden werden, ob $\left(\frac{y}{p}\right)=+1$ oder $=-1$). Dies heißt $-y=x^2$, man erhält also x aus $-y$ so wie oben aus y.

Wahl der Referenzen

Die Referenzen *Ref* müssen aus D_n gewählt werden. Dies kann geschehen, indem *Ref* zufällig < $n/2$ gewählt und anschließend das Jakobi-Symbol berechnet wird.[15] Die „Trefferwahrscheinlichkeit“ liegt in diesem Fall bei 1/2. Oder aber der Wert des Jakobi-Symbols wird erzwungen: Eine zufällig gewählte Zahl *z* wird (modulo *n*) quadriert; ist das Ergebnis z^2 nicht kleiner als $n/2$, so wird $Ref := n - z$ verwendet, anderenfalls $Ref := z$.

Präfixfreie Abbildung

Goldwasser, Rivest und Micali schlagen (exemplarisch) folgende präfixfreie Abbildung vor: Jede 1 der Nachricht wird durch 11, jede 0 durch 00 ersetzt; das Ende der Nachricht wird mit 01 gekennzeichnet. Werden zwei Referenzen r_0 und r_1 unterschrieben, so kennzeichnet die Bitfolge 10 das Ende der ersten und den Beginn der zweiten Referenz. Diese Abbildung verdoppelt die Nachrichtenlänge, die für den Berechnungsaufwand entscheidend ist; hier konnten wir Verbesserungen vorschlagen (Kapitel 4).[16]

Authentisierung der Referenzen

Der Unterzeichner kann für jede Signatur aus der jeweils vorigen den gemeinsamen Anfangsabschnitt des Pfades im Referenzenbaum mit allen zugehörigen, bereits berechneten K-Signaturen in den neuen Signaturkopf übernehmen. Es empfiehlt sich daher, nicht nur die weiterhin benötigten Hilfsreferenzen, sondern den gesamten Signaturkopf der vorherigen Signatur im „Gedächtnis“ zu behalten.

Da die Berechnung der Umkehrfunktionen (F_0^{-1}, F_1^{-1}) aufwendiger ist als das Rechnen mit F_0 und F_1, sollte weitestmöglich darauf verzichtet werden. Es zeigt sich, daß tatsächlich nur einmal je Signatur eine Authentisierung mit f^{-1}, also F_0^{-1} und F_1^{-1} erfolgen muß:

Bei jeder neuen Signatur kann der Signaturkopf zunächst „von unten“ mittels F_0 und F_1 berechnet werden: Die N-Signatur *NSig* für die Nachricht *m* wird zufällig gewählt und die Referenz als $R_i := f_{präf(m)}(NSig)$ berechnet.[17] Anschließend wird die R-Signatur *RSig* zufällig gewählt und r_i berechnet usw., bis ein „Verbindungsknoten“ erreicht ist, an dem der übernommene Pfadteil mit dem soeben neu berechneten verknüpft werden muß. Hier ist ein einziges Mal mit F_0^{-1} und F_1^{-1} zu rechnen (Abbildung 3.1).

---

[15] Diese Berechnung kann effizient mit einem rekursiven Algorithmus erfolgen; Hinweise dazu finden sich z.B. in [Hors_85].

[16] Die gleiche präfixfreie Abbildung wird auch von Damgård [Damg_87] verwendet; auch dort läßt sich unsere Abbildung einsetzen.

[17] Da F_0 und F_1 Permutationen sind, garantiert die Zufälligkeit von *NSig* auch die von R_i.

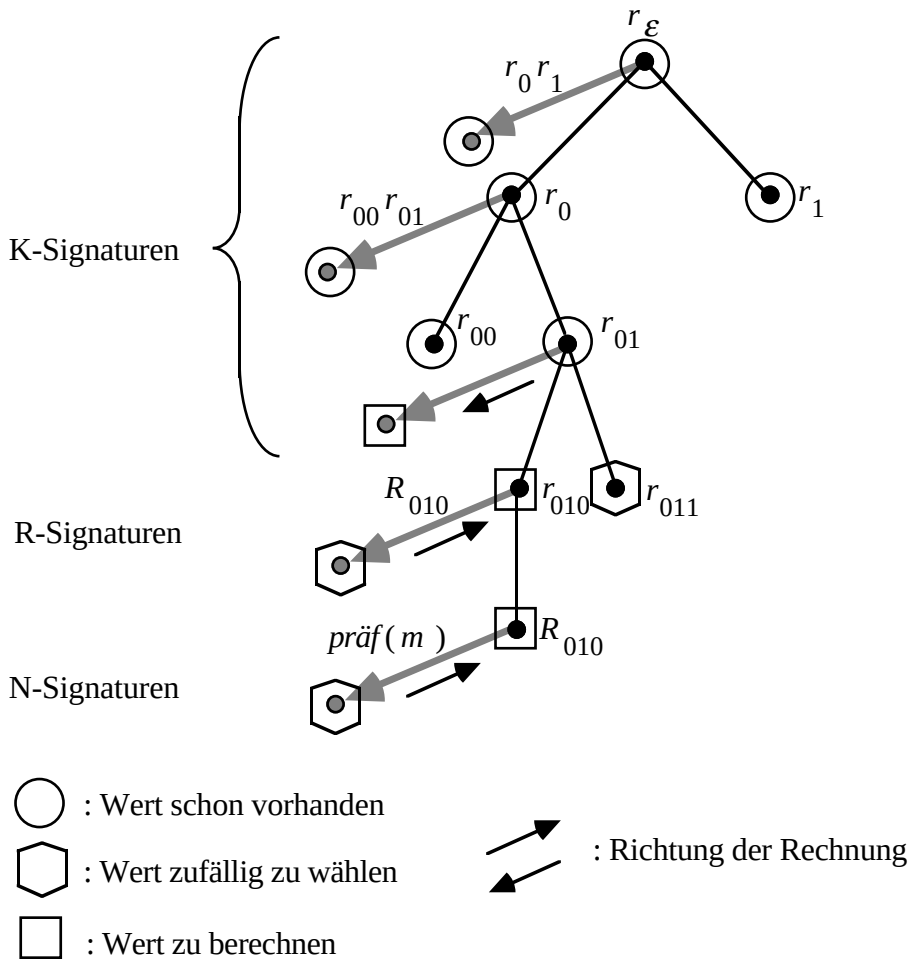

Abbildung 3.1 Rechnen „von oben“ und „von unten“

Berechnung von $f_w^{-1}(Ref)$

Von Goldreich wird in [Gold_86] ein Verfahren vorgeschlagen, das die (je Signatur einmalige) Berechnung von f_w^{-1} deutlich beschleunigt. Anstatt bitweise F_0^{-1} und F_1^{-1} anzuwenden, kann $f_w^{-1}(Ref)$ effizienter folgendermaßen in einem Schritt bestimmt werden:

Zunächst kann man zeigen, daß (bis auf das Vorzeichen)

$$Ref = f_w(Sig) = Sig^{2^{|w|}} \cdot 4^{rev(w)} \bmod n,$$

wobei die Funktion *rev* das bitweise Umdrehen einer Nachricht bezeichnet,[18]

also $$Sig = \sqrt[2^{|w|}]{Ref}_n \cdot \left(\left(\sqrt[2^{|w|}]{4}_n \right)^{-1} \right)^{rev(w)}. \qquad \text{(i)}$$

Dabei muß geprüft werden, ob *Ref* quadratischer Rest mod *n* ist – ist das nicht der Fall, ist die Rechnung mit *-Ref* mod *n* durchzuführen (siehe Abschnitt „klauenfreie Permutationenpaare“). Die Berechnung wird wieder mod *p* und *q* einzeln vorgenommen und die Ergebnisse mittels des CRA zusammengefügt. Modulo *p* wird eine *y*-fache Quadratwurzel folgendermaßen bestimmt:

$$\sqrt[2^y]{x}_p = x^{((p+1)/4)^y} = x^{((p+1)/4)^y \bmod (p-1)} \bmod p.$$

---

[18] Die Notation $\sqrt[2^y]{x}_n$ mod *n* bezeichnet die *y*-malige Berechnung der „Quadratwurzel“ modulo *n* aus *x* in $\mathbb{Z}_n^*$ und |*w*| die Länge der Nachricht *w* in bits.

Man berechnet also zunächst

$$expp := ((p+1)/4)^{|w|} \bmod (p-1), \qquad \text{(ii)}$$

anschließend $zwp := \sqrt[2^{|w|}]{Ref}_p = Ref^{expp}$

sowie $cp := \left(\sqrt[2^{|w|}]{4}_p\right)^{-1} = (4^{expp})^{-1}$, (iii)

und zuletzt $Sig_p := zwp \bullet cp^{rev(w)}$.

Analog wird Sig_q bestimmt. Schließlich errechnet man das Ergebnis als CRA(Sig_p, Sig_q).

Gedächtnisfreie Version

Leonid Levin schlägt vor, durch Nutzung der in [GoGM_86] vorgestellten Pseudozufallsfunktionen („polyrandom collections“) die Speicherung der jeweils letzten Signatur einzusparen. Dabei werden alle Referenzen nicht mehr zufällig gewählt, sondern als Funktionswert ihrer Position im Referenzenbaum bestimmt.[19]

Eine weitere Idee von Goldreich erlaubt, sogar die Speicherung der Nummer der letzten Signatur einzusparen; die neue Nummer wird statt dessen als Pseudozufallsfunktionswert der Nachricht bestimmt. Zur Kollisionsvermeidung muß dann aber der Referenzenbaum wesentlich größer als sonst gewählt werden.

4 Eigene Verbesserungen

Präfixfreie Abbildung

Wie im vorigen Abschnitt angedeutet, ist eine erhebliche Beschleunigung des Verfahrens durch eine effizientere präfixfreie Abbildung möglich. In unserer Implementierung haben wir die drei folgenden Verbesserungen vorgenommen:

1. Da alle Referenzen r_j an den Knoten des Referenzenbaumes, die im Signaturkopf zu authentisieren sind, etwa die gleiche Länge haben, kann hier eine ***feste Länge*** angenommen werden (kürzere Referenzen werden durch führende Nullen auf die korrekte Länge gebracht). Als Länge wird sinnvollerweise gerade der ***Sicherheitsparameter l*** gewählt, der die Länge des Modulus *n* (in bit) festlegt. Eine zusätzliche präfixfreie Abbildung erübrigt sich also.
2. Die Nachricht *m* läßt sich durch Voranhängen eines ***Längenprädikates***, eines Feldes fester Länge also, das die Länge der Nachricht enthält, präfixfrei abbilden. Dadurch wächst ihre Länge lediglich um ld(|*m*|).[20]
3. Das Umdrehen der Nachrichten *w* für das Goldreich-Verfahren zur schnellen Berechnung von f_w^{-1} kostet (unnötige) Rechenzeit. Auch die Auswertung der Nachricht zur Berechnung von f_w von hinten nach vorne ist störend, wenn man sukzessiv enstehende oder eintreffende Daten unmittelbar signieren will.[21] Dies läßt sich ändern, wenn man die präfixfreie Abbildung so

---

[19] Eine Skizze des Sicherheitsbeweises findet sich in [GoMR_88].

[20] Der Vorzug des GMR-Signatursystems, Nachrichten beliebiger Länge unterschreiben zu können, geht dabei nicht verloren, wenn z.B. über ein „Erweiterungsbit“ das Einfügen weiterer Längenfelder ermöglicht wird. Konkret kann man dazu das Vorzeichenbit wählen.

[21] Auch die folgenden Ideen würden das Umdrehen der Nachrichten erforderlich machen. Alternativ lassen sich Exponentiationen und Divisionen mit umgedrehten Nachrichten direkt programmieren; dies senkt aber entweder die Effizienz oder die Portabilität der Implementierung (je nachdem, ob diese Änderungen in Maschinensprache realisiert werden).

definiert, daß sie die Nachricht (virtuell) umdreht: Nun kann man die ursprüngliche Nachricht von vorn nach hinten auswerten. (Das Längenprädikat ist dann an die Nachricht anzuhängen.)

Diese Maßnahmen kommen sowohl dem Signier- als auch dem Testvorgang zugute: Der Aufwand wird jeweils näherungsweise um den Faktor zwei reduziert.

Man beachte, daß die Festlegung der präfixfreien Abbildung, im Unterschied zu den folgenden Effizienzverbesserungen, in allen Implementierungen gleich gewählt werden muß, damit die erzeugten Signaturen wechselseitig kompatibel (testbar) bleiben.

Testen

Zunächst kann generell beim Rechnen mit F_0 und F_1 auf den Test „$< n/2$“ verzichtet werden, da im Ergebnis lediglich ein Vorzeichenfehler auftritt und die nächste Operation wieder eine Quadrierung ist. Es genügt ein einziger Test nach der letzten Anwendung von F_0 und F_1. Auch die Anwendung von F_1 läßt sich noch ein wenig beschleunigen: Anstatt erst zu quadrieren und das Ergebnis mit 4 zu multiplizieren ($4x^2$), wird zuerst verdoppelt und anschließend quadriert ($(2x)^2$).[22]

Signieren

a) N-Signatur:

Im allgemeinen wird der Aufwand zum Signieren der eigentlichen Nachricht deutlich über dem Zusatzaufwand für die Generierung des Signaturkopfes liegen (Kapitel 5). Deshalb fallen Effizienzsteigerungen hier besonders stark ins Gewicht.

Die Nachricht selbst kann immer „von unten nach oben“ signiert werden; *NSig* wird zufällig gewählt und von dort mittels F_0 und F_1 die Referenz $R_i := f_{präf(m)}(NSig)$ berechnet.

Da der Unterzeichner den geheimen Schlüssel (und damit die Primfaktoren p und q von n) kennt, kann diese Berechnung mit Hilfe des CRA deutlich beschleunigt werden, analog [QuCo_82] für RSA. So kann man zunächst mit F_0 und F_1 modulo p und modulo q „hochrechnen“ und anschließend die Teilergebnisse mit Hilfe des CRA zusammenfügen. Modulo p und modulo q werden nur Zahlen der Länge $l/2$ multipliziert; da der Aufwand einer Multiplikation (in unserer Größenordnung) noch fast quadratisch mit der Länge wächst, läßt sich auf diese Weise fast die halbe Rechenzeit einsparen.

Eine Modifikation des Verfahrens von Goldreich führt zu einer weiteren erheblichen Geschwindigkeitssteigerung: Die iterativen Aufrufe von F_0 und F_1 modulo p bzw. modulo q lassen sich durch folgende Rechnung ersetzen (analog für q; die Teilergebnisse werden mithilfe des CRA zusamengefügt, *len* bezeichnet Länge der präfixfrei abgebildeten Nachricht):

$$R_{i,p} := f_{präf(m)}(NSig) = NSig^{2^{len} \bmod (p-1)} \cdot 4^{präf(m) \bmod (p-1)} \bmod p \qquad \text{(iv)}$$

Die Reduzierung von $präf(m)$ modulo $(p-1)$ bzw. $(q-1)$ vor der Exponentiation beschleunigt die Rechnung um den bereits erwähnten Faktor (siehe [4]).

b) R-Signatur:

Jede zweite R-Signatur erfolgt gerade an einem Verbindungsknoten und muß daher von oben berechnet werden. Dabei wird das Goldreich-Verfahren (i) angewendet. Dies läßt sich hier deutlich beschleunigen, da die Länge von R_i durch unsere Wahl der präfixfreien Abbildung festliegt: Alle Zwischenwerte, die nur von

---

[22] Der Aufwand für das Testen des Signaturkopfes läßt sich um cirka 5% senken, wenn die Referenzen statt an einen Binär- an einen Ternärbaum angehängt werden. Die Repräsentation wird jedoch deutlich komplexer, und der Verwaltungsaufwand steigt. Insbesondere kommt diese Verbesserung lediglich dem Testen des Signaturkopfes und nicht dem (bei langen Nachrichten deutlich aufwendigeren) Signaturrumpf zugute.

der Länge $l := |R_i|$ abhängen, lassen sich beim Generieren der Schlüssel vorausberechnen und als Konstante mitführen. Im einzelnen sind dies die Werte aus (ii) und (iii):

$$exppl := ((p+1)/4)^l \bmod (p-1)$$

und

$$cpl := \left(\sqrt[2^l]{4}\right)^{-1} \bmod (p-1).$$

Während des Signierens ist also bei R-Signaturen nur noch

$$\begin{aligned} RSig_p &:= r_i^{\,exppl} \cdot cpl^{\,R_i} \\ RSig_q &:= r_i^{\,exqql} \cdot cql^{\,R_i} \\ RSig &:= \mathrm{CRA}(RSig_p, RSig_q) \end{aligned} \qquad \text{(v)}$$

zu bestimmen. Dabei werden die Exponenten der zweiten Faktoren vorher modulo p-1 reduziert, um möglichst kurze Exponentiationen zu erhalten.

Ist die R-Signatur kein Verbindungsknoten, kann *RSig* zufällig gewählt und modulo p und modulo q „hochgerechnet“ werden. Die Teilergebnisse $r_{i,p}$ und $r_{i,q}$ werden mittels CRA zu $r_i := f_{R_i}(RSig)$ zusammengefügt.

c) K-Signaturen:

Erfolgt eine K-Signatur gerade an einem Verbindungsknoten, kommt wieder das Goldreich-Verfahren (i) zur Anwendung, diesmal allerdings mit Exponenten der Gesamtlänge 2l bit, da hier die Konkatenation zweier Nachfolgeknoten r_{j0} und r_{j1} zu signieren ist. Auch hier lassen sich die entsprechenden Konstanten bei der Schlüsselgenerierung vorausberechnen. Gemäß (ii) und (iii) sind dies

$$expp2l := ((p+1)/4)^{2l} \bmod (p-1)$$

und

$$cp2l := \left(\sqrt[2^{2l}]{4}\right)^{-1} \bmod (p-1).$$

Während des Signieren ist

$$\begin{aligned} KSig_p &:= r_j^{\,expp2l} \cdot cp2l^{\,r_{j0}r_{j1}} \\ KSig_q &:= r_j^{\,exqq2l} \cdot cq2l^{\,r_{j0}r_{j1}} \\ KSig &:= \mathrm{CRA}(KSig_p, KSig_q) \end{aligned} \qquad \text{(vi)}$$

zu bestimmen.

Alle übrigen K-Signaturen lassen sich mit der bei N-Signaturen beschriebenen Modifikation des Goldreich-Verfahrens effizient „von unten“ berechnen: Zunächst werden *KSig* und r_{j1} zufällig gewählt. Analog Formel (iv) wird r_j berechnet durch

$$r_{j,p} = f_{r_{j0}r_{j1}}(KSig) = KSig^{2^{2l} \bmod (p-1)} \cdot 4^{r_{j0}r_{j1} \bmod (p-1)} \bmod p, \qquad \text{(vii)}$$

die analoge Formel modulo q und eine Anwendung des CRA.[23] Die Konstante $2^{2l} \bmod (p-1)$ kann bereits beim Generieren der Schlüssel vorausberechnet werden.

Wahl der Referenzen

Zur Wahl der Referenzen (bzw. der Signaturen, siehe vorausgegangener Abschnitt) verwenden wir den BBS-Pseudozufallsbitfolgengenerator [BlBS_86, VaVa_84]. Ausgehend von einer (echt zufälligen) Quelle

[23] Aus den Aufwandsabschätzungen kann man entnehmen, daß sich die Anwendung von Formel (iv) bei K-Signaturen schon lohnt, bei R-Signaturen jedoch noch nicht.

der Länge *l* bit wird bezüglich eines *l* bit langen Modulus *n'* quadriert. Von dem Ergebnis werden jeweils die letzten ld(*l*) Bit in die Pseudozufallszahl übernommen. Es ist bewiesen, daß dieser Pseudozufallsbitfolgengenerator alle statistischen Tests besteht.[24] Die so gewonnene Pseudozufallszahl *z* ist anschließend noch einmal modulo *n* zu quadrieren, um $(\frac{z}{n})=1$ zu erzwingen; gewählt wird *z*, falls $z<n/2$, anderenfalls *-z* mod *n*. Merkt man sich, ob *z* oder *-z* quadratischer Rest ist, kann man diesen Test bei der Berechnung von $f_w^{-1}(Ref)$ sparen (vgl. Kapitel 3).

Keine gedächtnisfreie Version

Da es bei einer Softwareimplementierung in erster Linie um eine Reduzierung des Rechenaufwandes des Verfahrens geht, verzichteten wir auf die Implementierung einer gedächtnisfreien Version: Zum einen können darin keine Authentisierungen aus der vorhergehenden Signatur übernommen werden; dadurch steigt der Berechnungsaufwand für den Signaturkopf durchschnittlich um den Faktor *b*. Zum anderen ist auch das Berechnen eines Funktionswertes einer Pseudozufallsfunktion aufwendiger als einfaches pseudozufälliges Generieren. Erst recht lohnt es sich nicht, die Nummer der Signatur einzusparen.

Besondere Anwendungen

In Sonderfällen kann man das Verfahren weiter beschleunigen:[25]

Wenn zwischen dem Signieren einzelner Nachrichten im allgemeinen etwas Zeit vergeht, eine Nachricht aber, sobald sie vorliegt, so schnell wie möglich unterschrieben werden soll, kann man in der Zwischenzeit die von der Nachricht unabhängigen Teile der nächsten Signatur vorwegberechnen. Insbesondere kann man schon alle zufälligen neuen Referenzen und Signaturen wählen und sämtliche K-Signaturen berechnen. Im Extremfall kann man (wenn genügend Speicherplatz vorhanden ist) den ganzen Referenzenbaum vorweg authentisieren.

Das Testen kann man beschleunigen, wenn ein Empfänger mehrere Signaturen desselben Signierers erhält. Er speichert dazu ebenfalls die jeweils letzte Signatur und testet vom Signaturkopf nur den Teil, der sich geändert hat. Dies lohnt sich insbesondere, wenn er mehrere Signaturen direkt hintereinander erhält (etwa in einem mehrschrittigen Protokoll). Im allgemeinen Fall können die Signierer dies unterstützen, indem sie die Signaturen nicht von vorn nach hinten „verbrauchen“, sondern bestimmten Empfängern bestimmte Teile des Baumes zuordnen. Dann muß im Mittel, wie beim Signieren, auch nur eine K-Signatur pro Gesamtsignatur getestet werden.

5 Aufwandsabschätzungen

Als Einheit für die folgenden Aufwandsabschätzungen verwenden wir die Aufwände $A_{l/2}$ und A_l einer modulare Multiplikationen bzgl. Moduli der Länge $\frac{l}{2}$ bzw. *l*-bit, im folgenden kurz $\frac{l}{2}$-Multiplikationen bzw. *l*-Multiplikationen genannt. (Real kommen noch einige Additionen und Verwaltungsoperationen hinzu.) Bei der Exponentiation wird die Leistung unserer Langzahlarithmetik zugrundegelegt; für einen Exponenten der Länge *l* werden etwa $\frac{3}{2}l$ Multiplikationen benötigt.[26]

---

[24] Die Sicherheit von GMR wird durch diese Wahl nicht beeinträchtigt, wenn man die Referenzenbaumbreite polynomial begrenzt. Die Beweisskizze für die von Levin vorgeschlagenen Pseudozufallsfunktionen in [GoMR_88] läßt sich auf den BBS-Generator übertragen.

[25] Diese speziellen Verbesserungen haben wir nicht implementiert.

[26] Dieser Wert läßt sich noch deutlich verbessern: [BoRu_89] erreichen einen Faktor von 1,25; in [Knut_81] und [BoCo_89] finden sich weitere effizienzsteigernde Hinweise. Diese Verbesserungen kommen RSA in allen Teilen zugute, GMR nur da, wo exponentiert statt mit F_0 und F_1 gerechnet wird.

Für die Reduktion langer Zahlen gehen wir vom Wert aus [Knut_81] aus: Die Division einer Zahl aus x Blöcken der Länge $\frac{l}{2}$ durch eine Zahl der Länge $\frac{l}{2}$ entspricht etwa 1,2•(x-1) nichtmodularen Multiplikationen der Länge $\frac{l}{2}$, also näherungsweise 1/2•(x-1) (modularen) $\frac{l}{2}$-Multiplikationen.

Signieren bei GMR

Die 2^{b+1}-1 K-Signaturen und die 2^b R-Signaturen des Referenzenbaumes sind jeweils genau einmal zu berechnen. Damit sind pro Gesamtsignatur neben der N-Signatur durchschnittlich zwei zusätzliche Authentisierungen im Referenzenbaum notwendig: eine K-Signatur zweier Nachfolgereferenzen r_{j0} und r_{j1} und eine R-Signatur einer Referenz R_i. Der zusätzlich erforderliche mittlere Signieraufwand ist also unabhängig von der Referenzenbaumtiefe b!

Der Rechenaufwand mit allen beschriebenen Verbesserungen beträgt im einzelnen:

a) N-Signatur:

Zunächst wird *NSig* mithilfe des BBS-Generators gewählt, was etwa $\frac{l}{\mathrm{ld}(l)}$ l-Multiplikationen erfordert, und das Ergebnis wird modulo n quadriert, um ein Jakobi-Symbol 1 zu erzwingen. Gemäß Formel (iv) wird nun 2^{len} mod (p-1) berechnet und $präf(m)$ modulo p-1 reduziert. Dies entspricht $\frac{3}{2}$•ld(len) bzw. etwa $\frac{1}{2}\bullet\frac{len}{l/2} = \frac{len}{l}\frac{l}{2}$-Multiplikationen. Gleiches erfolgt modulo q. Dann erfolgen 4 Exponentiationen, bei denen alle Parameter die Länge $\frac{l}{2}$ haben, also jeweils $\frac{3}{2}\bullet\frac{l}{2}\ \frac{l}{2}$-Multiplikationen. Es folgen 2 Multiplikationen der Teilergebnisse und der CRA, der wiederum 1,5 $\frac{l}{2}$-Multiplikationen enthält.
Gesamtaufwand: $\mathbf{(\frac{l}{ld(l)}+1)\bullet A_l + (\frac{3}{2}\bullet ld(len) + \frac{len}{l} + 3l+3{,}5)\bullet A_{l/2}}$.

b) R-Signatur:

- Von oben: Bei jeder zweiten Signatur ist die R-Signatur gerade der Verbindungsknoten (alle K-Signaturen können von der vorherigen Signatur übernommen werden). In diesem Fall wird das durch Vorausberechnungen verbesserte Goldreich-Verfahren angewendet (Formel (v)): Zunächst wird R_i (der Länge $\frac{l}{2}$) modulo (p-1) und (q-1) reduziert, was etwa einer $\frac{l}{2}$-Multiplikation entspricht. Dann erfolgen 4 Exponentiationen, bei denen alle Parameter die Länge $\frac{l}{2}$ haben, also jeweils $\frac{3}{2}\bullet\frac{l}{2}\ \frac{l}{2}$-Multiplikationen. Es folgen 2 Multiplikationen der Teilergebnisse und der CRA, der wieder 1,5 $\frac{l}{2}$-Multiplikationen enthält.
  Gesamtaufwand: $\mathbf{(3l+4{,}5)\bullet A_{l/2}}$.
- Von unten: Zur Wahl von *RSig* sind (wie für *NSig*) $\frac{l}{\mathrm{ld}(l)}$+1 l-Multiplikationen notwendig. Anschließend wird modulo p und modulo q „hochgerechnet“ (Länge des Exponenten: l Bit, entspricht jeweils l $\frac{l}{2}$-Multiplikationen) und die Teilergebnisse mit dem CRA zusammengefügt.
  Gesamtaufwand: $\mathbf{(\frac{l}{ld(l)}+1)\bullet A_l + (2l+1{,}5)\bullet A_{l/2}}$

c) K-Signaturen:

- Von oben: Erfolgt die K-Signatur an einem Verbindungsknoten, so wird nur die Referenz r_{j1} mit dem BBS-Generator gewählt ($\frac{l}{\mathrm{ld}(l)}$+1 l-Multiplikationen). Dann kommt wie bei R-Signaturen das Goldreich-Verfahren mit Vorausberechnungen zum Tragen (Formel (vi)): Zunächst wird $r_{j0}r_{j1}$ (der Länge 2l) modulo (p-1) und (q-1) reduziert, was etwa drei $\frac{l}{2}$-Multiplikationen entspricht. Dann erfolgen 4 Exponentiationen, bei denen alle Parameter die Länge $\frac{l}{2}$ haben, also jeweils $\frac{3}{2}\bullet\frac{l}{2}\ \frac{l}{2}$-Multiplikationen. Es folgen 2 Multiplikationen der Teilergebnisse und der CRA.
  Gesamtaufwand: $\mathbf{(\frac{l}{ld(l)}+1)\bullet A_l + (3l+6{,}5)\bullet A_{l/2}}$.
- Von unten: Die Signatur *KSig* und die Referenz r_{j1} werden mit dem BBS-Generator gewählt. Dazu sind 2•$\frac{l}{\mathrm{ld}(l)}$+2 l-Multiplikationen erforderlich. Gemäß Formel (vii) ist wieder $r_{j0}r_{j1}$ modulo p-1 und q-1 zu reduzieren (etwa drei $\frac{l}{2}$-Multiplikationen), dann erfolgen 4 Exponentiationen, 2 Multiplikationen und der CRA.
  Gesamtaufwand: $\mathbf{(2\bullet\frac{l}{ld(l)}+2)\bullet A_l + (3l+6{,}5)\bullet A_{l/2}}$.

Man sieht, daß zwischen Hinauf- und Herunterrechnen bei K-Signaturen nach allen Verbesserungen kaum noch Unterschiede bestehen.

Wie erwähnt, sind je Signatur durchschnittlich je eine K-, R- und N-Signatur zu berechnen, wobei der Verbindungsknoten abwechselnd bei der R- und einer K-Signatur ist. Der mittlere Signieraufwand beträgt demnach näherungsweise:

Mittlerer Signieraufwand:

$$(3 \bullet \frac{l}{\mathrm{ld}(l)} + 3) \bullet A_l + (\frac{3}{2} \bullet \mathrm{ld}(len) + \frac{len}{l} + 8{,}5l + 13) \bullet A_{l/2}$$

Testen bei GMR

Im allgemeinen Fall muß angenommen werden, daß der Empfänger keinen Teil der empfangenen Signatur bereits von älteren Signaturen kennt. Daher sind die N-Signatur der Nachricht, b K-Signaturen zweier Nachfolgereferenzen r_{j0} und r_{j1} sowie die R-Signatur zu überprüfen. Der zusätzliche Testaufwand steigt also linear mit der Baumtiefe (logarithmisch mit der Anzahl der zu unterzeichnenden Nachrichten).

Beim Überprüfen der gesamten Signatur muß mit l-Multiplikationen von unten nach oben gerechnet werden, da die Faktorisierung von n (der geheime Schlüssel g) dem Empfänger nicht bekannt ist.

Für die Überprüfung der K-Signaturen zweier Nachfolgereferenzen r_{j0} und r_{j1} sind $2l$ l-Multiplikationen erforderlich; der Test der R-Signatur von R_i benötigt l und die Überprüfung der N-Signatur schließlich len+ld(len) l-Multiplikationen. Der gesamte Überprüfungsaufwand liegt demnach je Signatur bei:

Testaufwand: $((2b+1) \bullet l + (len + \mathrm{ld}(len)) \bullet A_l$

Signieren bei (reinem) RSA

Wir nehmen an, daß die Multiplikativität von RSA durch ein Redundanzprädikat umgangen wird, das die Nachricht m um ca. ld(len) Bit verlängert. Dann liegt der Signieraufwand pro Nachrichtenblock bei zweimal (bezüglich p und q) $\frac{3}{2} \bullet \frac{l}{2} \frac{l}{2}$-Multiplikationen, plus 1,5 für den CRA, und die Nachricht enthält $\frac{len+\mathrm{ld}(len)}{l}+1$ Blöcke.

Signieraufwand: $(\frac{3}{2}l+1{,}5) \bullet (\frac{len+\mathrm{ld}(len)}{l}+1) \bullet A_{l/2}$

Testen bei (reinem) RSA

Auch bei RSA kann beim Testen nicht auf den CRA zurückgegriffen werden. Der Aufwand liegt also etwas über dem Signieraufwand: $\frac{len+\mathrm{ld}(len)}{l}+1$ Blöcke mit je $\frac{3}{2}l$ l- Multiplikationen.

Testaufwand: $\frac{3}{2}l \bullet (\frac{len+\mathrm{ld}(len)}{l}+1) \bullet A_l$

Signaturlänge

Eine GMR-Signatur enthält b K-Signaturen *KSig* und $2b$ Hilfsreferenzen r_j, eine R-Signatur *RSig*, die authentisierte Referenz R_i und die N-Signatur *NSig*, umfaßt also $3b$+3 Langzahlen (Länge l in bit). Zum Testen genügt es jedoch, lediglich eine Hilfsreferenz je K-Signatur zu kennen – die andere liefert der Testalgorithmus als Zwischenergebnis; gleiches gilt für die Referenz R_i. Die Länge der zu versendenden Signatur läßt sich also auf $\mathbf{2(b+1) \bullet l}$ bit verringern (Einsparung: 1/3). Die Signaturlänge ist unabhängig von der Nachrichtenlänge.

Die Länge einer RSA-Signatur ist identisch der Länge der Nachricht plus Redundanzprädikat: ***len*+ld(*len*)** bit.

Vergleich

Insgesamt ließ sich durch die vorgeschlagene präfixfreie Abbildung der unsprüngliche Testaufwand um den Faktor 2 reduzieren; den Signieraufwand konnten wir für lange Nachrichten um einen Faktor nahe der Moduluslänge l vermindern.

Damit die Formeln übersichtlicher werden, wird der Aufwand noch etwas gröber abgeschätzt: Konstante und logarithmische Terme werden vernachlässigt, und es wird $A_l = 4A_{l/2}$ angenommen (was bei l=512 in unserer Arithmetik auf dem MC68020 noch zutrifft).

Multiplikationen [$A_{l/2}$] allgemein	**GMR**	**RSA**
Signieren	$\frac{len}{l} + (8{,}5 + \frac{12}{\mathrm{ld}(l)}) \bullet l$	$\frac{3}{2}\, len$
Testen	$4((2b+1) \bullet l + len)$	$6\, len$

Für sehr lange Nachrichten nähern sich diese Werte den folgenden:

Multiplikationen [$A_{l/2}$] für sehr lange Nachrichten	**GMR**	**RSA**
Signieren	$\rightarrow \frac{len}{l}$	$\frac{3}{2}\, len$
Testen	$\rightarrow 4\, len$	$6\, len$

Der Vergleich zeigt, daß GMR für lange Nachrichten effizienter ist als reines RSA, für kurze aufwendiger. Beim Testen verringert sich allerdings der Unterschied, wenn man die Exponentiation den Vorschlägen in [Knut_81] und [BoCo_89] entsprechend verbessert.

RSA wird allerdings nur selten in reiner Form angewendet. Oft wird die Größe des öffentlichen Exponenten verringert, was den Testaufwand drastisch senkt, z.B. [Jung_87]. Es ist unbewiesen, ob das System dann noch so sicher wie reines RSA ist (vgl. z.B. [WiSc_79]). Sowohl Signieren als auch Testen werden meist durch Hashen der Nachricht vor dem Unterschreiben beschleunigt; auch wird die Länge der Unterschriften dadurch auf die Blocklänge l verkürzt. Die Sicherheit bleibt jedoch nur erhalten, wenn es für einen Angreifer praktisch unmöglich ist, mit einer signifikant höheren Wahrscheinlichkeit als Raten zwei Nachrichten m, m' zu finden mit $h(m)=h(m')$.[27] Für Hashfunktionen, die effizienter sind als GMR, kann dies bisher nicht bewiesen werden.

6 Messungen

RSA und GMR wurden von uns – ausgehend von einer LISP-Implementierung [Fox_90] – in Pascal auf Macintosh-Rechnern programmiert. Dabei nutzten wir eine MC680xy-Assembler-Langzahlarithmetik mit Pascal-Schnittstelle, die an unserem Institut entwickelt wurde [Aßma_89]. Wir ergänzten sie durch eine

[27] Diese Eigenschaft einer Hashfunktion heißt ***Kollisionsfreiheit*** [Damg_87].

Prozedur, die – um die Vergleichbarkeit mit RSA zu erhalten – das „Hochrechnen“ in GMR ebenso wie die Exponentiation in Maschinensprache erledigt.[28]

Als Beispiel wählten wir eine (pseudozufällige) 1 KByte (d.h. 8192 bit) lange Nachricht; diese Größe entspricht etwa dem Umfang eines (kürzeren) DIN-A-4-Briefes. Moduluslänge ist in unserem Beispiel l=512 bit; wir wählten diese Länge, um einen Vergleich mit veröffentlichten Werten zu ermöglichen.[29] Im Beispiel halten sich der Aufwand von RSA (in reiner Form) und GMR etwa die Waage.

GMR

	Multiplikationen [$A_{l/2}$], grob Moduluslänge l=512 bit Baumtiefe b=10 (1024 Signaturen) Nachrichtenlänge *len*=8192 bit	**Zeit** [s] Mac Plus MC68000 7,83 MHz	 Mac II MC68020 15,7 MHz	 Mac IIfx MC68030 40 MHz
N-Signatur (hoch)	$4\frac{l}{\mathrm{ld}(l)} + 3l + \frac{len}{l}$	48,2	6,5	2,15
R-Signatur (hoch)	$4\frac{l}{\mathrm{ld}(l)} + 2l$	13,7	1,25	0,4
R-Signatur (runter)	$3l$	14,5	1,3	0,45
K-Signatur (hoch)	$8\frac{l}{\mathrm{ld}(l)} + 3l$	23,4	2,25	0,75
K-Signatur (runter)	$3l$	15,2	1,4	0,45
Generierung einer zufälligen Referenz	$4\frac{l}{\mathrm{ld}(l)}$	3,7	0,35	0,1
Signieren (Mittel)	$\frac{len}{l} + (8{,}5 + \frac{12}{\mathrm{ld}(l)})\bullet l$	81,6	9,6	3,2
Testen	$4\,((2b+1)\bullet l + len)$	608,9	47,3	16,55

Die Länge der GMR-Signatur beträgt im Beispiel 11264 bit (1408 Byte).

Die Messung zeigt, daß unsere RSA-Implementierung etwas langsamer ist als die in [BoRu_89] veröffentlichte; diese Abweichung dürfte auf effizientere Exponentiations- und Divisionsalgorithmen zurückzuführen sein. Die Meßergebnisse stimmen fast mit den in [Pohl_90] vorgestellten Werten überein.

RSA

	Multiplikationen [$A_{l/2}$], grob Moduluslänge l=512 bit Nachrichtenlänge *len*=8192 bit	**Zeit** [s] Mac Plus MC68000 7,83 MHz	 Mac II MC68020 15,7 MHz	 Mac IIfx MC68030 40 MHz
Signieren	$\frac{3}{2}\,len$	124,95	11,85	4,15
Testen	$6\,len$	401,6	38,25	13,30

Die Länge der RSA-Signatur beträgt im Beispiel 8192 bit (1024 Byte).

[28] Eine Implementierung auf Intel 80x86-Rechnern ist derzeit in Arbeit.

[29] Aus Sicherheitsgründen sollte heute üblicherweise eine Moduluslänge von 1024 bit gewählt werden.

Dank

Wir danken vor allem *Ralf Aßmann*, der den größten Teil der verwendeten Langzahlarihmetik programmierte und sicherstellte, daß die Rechner rechneten. *Gerrit Bleumer* leistete bei der Implementierung wertvolle Unterstützung. Für hilfreiche Diskussionen danken wir außerdem *Manfred Böttger*, *Jörg Lukat*, *Andreas Pfitzmann* und *Michael Waidner*.

Literatur

Aßma_89 Ralf Aßmann: "Assembler-Implementierung von modularer Langzahlarithmetik", Studienarbeit, Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe 1989.

Blum_82 Manuel Blum: "Coin flipping by telephone – a protocol for solving impossible problems", *Proc. IEEE Spring CompCon*, San Francisco 1982, S.133-137.

BlBS_86 Lenore Blum, Manuel Blum, Michael Shub: "A Simple Unpredictable Pseudo-Random Number Generator", *SIAM Journal on Computing* Bd.15, 1986, S.364-383.

BoCo_89 Jurjen Bos, Matthijs Coster: "Addition Chain Heuristics", Crypto '89, LNCS 435, Springer-Verlag, Heidelberg 1990, S.400-407.

BoRu_89 Dieter Bong, Christoph Ruland: "Optimized Software Implementations of the Modular Exponentiation on General Purpose Microprocessors", *Computers & Securit*y, Bd.8, Nr.7, 1989, S.621-630.

Damg_87 Ivan Bjerre Damgård: "Collision Free Hash Functions and Public Key Signature Schemes", Eurocrypt 1987, *Lecture Notes in Computer Science* 304, Springer, Berlin 1988, S.203-216.

Davi_82 George I. Davida: "Chosen Signature Cryptanalysis of the RSA (MIT) Public Key Cryptosystem", Technical Report TR-CS-82-2, University of Wisconsin, 10/1982.

Denn_84 Dorothy E.R. Denning: "Digital Signatures with RSA and Other Pulic-Key Cryptosystems", *Communications of the ACM*, Bd.27, Nr.4, 1984, S.388-392.

DiHe_76 Whitfield Diffie, Martin E. Hellmann: "New Directions in Cryptography", *IEEE Trans. on Inform. Theory*, Bd.IT-22, Nr.6, 1976, S.644-654.

Fox_90 Dirk Fox: "Implementierung eines sicheren digitalen Signatursystems", Studienarbeit, Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe 1990.

GoGM_84 Oded Goldreich, Shafi Goldwasser, Silvio Micali: "How to Construct Random Functions", *Proc. of 25th Symposium on FOCS* 1984, 1984, S.464-479.

GoGM_86 Oded Goldreich, Shafi Goldwasser, Silvio Micali: "How to Construct Random Functions", *Journal of the ACM*, Bd.33, Nr.4, 1986, S.792-807.

Gold_86 Oded Goldreich: "Two Remarks Concerning the Goldwasser-Micali-Rivest Signature Scheme", Crypto 1986, *Lecture Notes in Computer Science* 263, Springer, Berlin 1987, S.104-110.

GoMR_84 Shafi Goldwasser, Silvio Micali, Ronald L. Rivest: "A 'Paradoxical' Solution to the Signature Problem", 25th Symposium on FOCS 1984, *IEEE Computer Society* 1984, S.441-448.

GoMR_88 Shafi Goldwasser, Silvio Micali, Ronald L. Rivest: "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks", *SIAM Journal on Computing*, Bd.17, Nr.2 1988, S.281-308.

Hors_85 Patrick Horster: "Kryptologie", *Reihe Informatik* Nr. 47, Bibliographisches Institut, Mannheim 1985.

Jung_87 Achim Jung: "Implementing the RSA Cryptosystem", *Computers & Security*, Nr.6, 1987, S.342-350.

Knut_81 Donald E. Knuth: "The Art of Computer Programming", Bd.2, "Seminumerical Algorithms", 2.Auflage, *Addison-Wesley*, Massachusetts 1981.

Pohl_90 Norbert Pohlmann: "Das RSA-Verfahren und dessen Anwendung", DuD Datenschutz und Datensicherung, Nr.1, 1990, S.14-22.

QuCo_82 Jean-Jaques Quisquater, C. Couvreur: "Fast Decipherment Algorithm for RSA Public-Key Cryptosystem", *Electronics Letters*, Bd.18, Nr.21, 1982, S.905-907.

RSA_78 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.

VaVa_84 Umesh V. Vazirani, Vijai V. Vazirani: "Efficient and Secure Pseudo-random Number Generation", *Crypto '84*, LNCS 196, Springer-Verlag Berlin, 1985, S.193-202.

WiSc_79 Hugh C. Williams, B. Schmid: "Some Remarks concerning the MIT Public-Key Cryptosystem", *BIT*, Bd.19, 1979, S.525-538.