RSA-Kryptosystem

RSA (**Rivest–Shamir–Adleman**) ist ein asymmetrisches kryptographisches Verfahren, das sowohl zum Verschlüsseln als auch zum digitalen Signieren verwendet werden kann.[1] Es verwendet ein Schlüsselpaar, bestehend aus einem privaten Schlüssel, der zum Entschlüsseln oder Signieren von Daten verwendet wird, und einem öffentlichen Schlüssel, mit dem man verschlüsselt oder Signaturen prüft. Der private Schlüssel wird geheim gehalten und kann nicht mit realistischem Aufwand aus dem öffentlichen Schlüssel berechnet werden.

Inhaltsverzeichnis

Geschichte

Nachdem Whitfield Diffie und Martin Hellman im Jahr 1976 eine Theorie zur Public-Key-Kryptografie veröffentlicht hatten,[2] versuchten die drei Mathematiker Ronald L. Rivest, Adi Shamir und Leonard Adleman am MIT, die Annahmen von Diffie und Hellman zu widerlegen. Nachdem sie den Beweis bei verschiedenen Verfahren durchführen konnten, stießen sie schließlich auf eines, bei dem sie keinerlei Angriffspunkte fanden. Hieraus entstand 1977 *RSA*, das erste veröffentlichte asymmetrische Verschlüsselungsverfahren. Der Name RSA steht für die Anfangsbuchstaben ihrer Familiennamen. Da Adleman seinen Anteil als gering einschätzte und anfangs gar nicht als Autor genannt werden wollte, kam es zur nicht-alphabetischen Reihenfolge der Autoren und damit zur Abkürzung RSA[3].

Bereits Anfang der 1970er Jahre war im britischen GCHQ von Ellis, Cocks und Williamson ein ähnliches asymmetrisches Verfahren entwickelt worden, welches aber keine große praktische Bedeutung erlangte und aus Geheimhaltungsgründen nicht wissenschaftlich publiziert wurde.[4] RSA konnte daher 1983 zum Patent angemeldet werden,[5] obgleich es nicht das erste Verfahren dieser Art war. Das Patent erlosch am 21. September 2000.

Verfahren

Das Verfahren ist mit dem Rabin-Verschlüsselungsverfahren verwandt. Da es deterministisch arbeitet, ist es unter Umständen für bestimmte Angriffe anfällig. In der Praxis wird RSA daher mit dem Optimal Asymmetric Encryption Padding kombiniert.

Einwegfunktionen

→ *Hauptartikel: Einwegfunktion*

Funktionen, bei denen eine Richtung leicht, die andere (Umkehrfunktion) schwierig zu berechnen ist, bezeichnet man als *Einwegfunktionen* (engl. *one-way function*). Beispielsweise ist nach aktuellem Wissensstand die Faktorisierung einer großen Zahl, also ihre Zerlegung in ihre Primfaktoren, sehr aufwändig, während das Erzeugen einer Zahl durch Multiplikation von Primzahlen relativ einfach und schnell möglich ist. Spezielle Einwegfunktionen sind Falltürfunktionen (engl. *trapdoor one-way function*), die mit Hilfe einer Zusatzinformation auch rückwärts leicht zu berechnen sind.

Die Verschlüsselung und die Signatur mit RSA basieren auf einer Einwegpermutation mit Falltür (engl. *trapdoor one-way permutation*, kurz *TOWP*), einer Falltürfunktion, die gleichzeitig bijektiv, also eine Permutation, ist. Die Einwegeigenschaft begründet, warum die Entschlüsselung (bzw. das Signieren) ohne den geheimen Schlüssel (die Falltür) schwierig ist.

Erzeugung des öffentlichen und privaten Schlüssels

Die Schlüssel bestehen aus den drei Zahlen . und . Man nennt den *RSA-Modul*, . den *Verschlüsselungsexponenten* und . den *Entschlüsselungsexponenten*. Das Zahlenpaar . bildet den öffentlichen Schlüssel (public key) und das Paar . den privaten Schlüssel (private key). Diese Zahlen werden durch das folgende Verfahren erzeugt:

1. Wähle zufällig und stochastisch unabhängig zwei Primzahlen . Diese sollen die gleiche Größenordnung haben, aber nicht zu dicht beieinander liegen, sodass der folgende Rahmen ungefähr eingehalten wird: .[6] (In der Praxis erzeugt man dazu solange Zahlen der gewünschten Länge und führt mit diesen anschließend einen Primzahltest durch, bis man zwei Primzahlen gefunden hat.)
2. Berechne den RSA-Modul
3. Berechne die Eulersche φ-Funktion von
4. Wähle eine zu teilerfremde Zahl , für die gilt .
5. Berechne den Entschlüsselungsexponenten als multiplikativ Inverses von bezüglich des Moduls , was mit dem erweiterten euklidischen Algorithmus erfolgen kann. Es soll also die Kongruenz gelten:

Die Zahlen , und werden nicht mehr benötigt und können nach der Schlüsselerstellung gelöscht werden. Es ist jedoch relativ einfach, diese Werte aus , und zu rekonstruieren. und müssen geheim gehalten werden.

Da die Primzahltests inzwischen ausreichend schnell sind, wählt man heutzutage zuerst einen kleinen Exponenten mit und verwirft bei der Erzeugung die Primzahlen , für die nicht teilerfremd zu sind. Die Wahl eines kleiner als die Fermat-Zahl kann zu Angriffsmöglichkeiten führen, etwa in Form des von Johan Håstad publizierten „Broadcast“-Angriffs, bei dem der Versand einer Nachricht an mehrere Empfänger zu einer Dechiffrierung über den chinesischen Restsatz führen kann.[7]

Wenn weniger als ein Viertel der Bits des RSA-Moduls hat, kann – sofern nicht bestimmte Zusatzbedingungen erfüllt sind – mit einem auf Kettenbrüchen aufbauenden Verfahren effizient ermittelt werden.[8] Bei der Wahl eines Exponenten kleiner als ist diese Möglichkeit jedoch ausgeschlossen.

Beispiel

1. Man wählt den Exponenten .
2. Man wählt und für die beiden Primzahlen. Die Zahlen und sind teilerfremd zum Exponenten .
3. Der RSA-Modul ist . Damit bilden und den öffentlichen Schlüssel.
4. Die eulersche φ-Funktion hat den Wert .
5. Berechnung der Inversen zu modulo :
   Es gilt: ,
   im konkreten Beispiel: .
   Mit dem erweiterten euklidischen Algorithmus berechnet man nun die Faktoren und , und somit ist der private Schlüssel, während nicht weiter benötigt wird.

Verschlüsseln von Nachrichten

Um eine Nachricht zu verschlüsseln, verwendet der Absender die Formel

.

und erhält so aus der Nachricht den Geheimtext . Die Zahl muss dabei kleiner sein als der RSA-Modul .

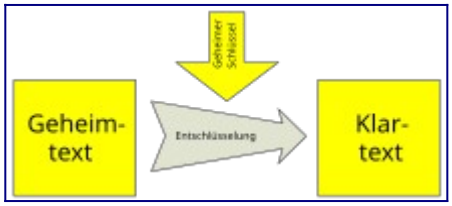
Beispiel

Es soll die Zahl 7 verschlüsselt werden. Der Sender benutzt den veröffentlichten Schlüssel des Empfängers , und rechnet

.

Das Chiffrat ist also .

Entschlüsseln von Nachrichten



Der Geheimtext kann durch modulare Exponentiation wieder zum Klartext entschlüsselt werden. Der Empfänger benutzt die Formel

.

mit dem nur ihm bekannten Wert sowie .

Beispiel

Für gegebenes wird berechnet

.

Der Klartext ist also .

Signieren von Nachrichten

Um eine Nachricht zu signieren, wird vom Sender auf die Nachricht die RSA-Funktion mit dem eigenen privaten Schlüssel angewendet. Zum Prüfen wendet der Empfänger auf die Signatur mit Hilfe des öffentlichen Schlüssels des Senders die Umkehrfunktion an und vergleicht diese mit der zusätzlich übermittelten unverschlüsselten Nachricht . Wenn beide übereinstimmen, ist die Signatur gültig und der Empfänger kann sicher sein, dass derjenige, der das Dokument signiert hat, auch den privaten Schlüssel besitzt und dass niemand seit der Signierung das Dokument geändert hat. Es wird also die Integrität und Authentizität garantiert, vorausgesetzt, der private Schlüssel ist wirklich geheim geblieben. Aufgrund der Homomorphieeigenschaft von RSA ist dieses Signaturverfahren

jedoch ungeeignet. Liegen zwei Signaturen ., . vor, so kann ein Angreifer daraus durch Multiplizieren die Signatur der Nachricht . berechnen. Sogar aus nur einer Signatur . kann ein Angreifer beliebig viele Nachrichten-Signatur-Paare erzeugen: . ist ein solches Paar für beliebige .

Dieses Problem kann umgangen werden, indem nicht die Nachricht selbst signiert wird. Stattdessen wird mit einer zusätzlich zum Signaturverfahren spezifizierten kollisionsresistenten Hashfunktion der Hash-Wert . der Nachricht . berechnet. Dieser wird mit dem privaten Schlüssel signiert, um die eigentliche Signatur zu erhalten. Der Empfänger kann die so erhaltene Signatur mit dem öffentlichen Schlüssel verifizieren und erhält dabei einen Wert .. Diesen vergleicht er mit dem von ihm selbst gebildeten Hashwert . der ihm vorliegenden Nachricht . Wenn beide Werte . übereinstimmen, kann mit hoher Wahrscheinlichkeit davon ausgegangen werden, dass die Nachricht fehlerfrei übertragen wurde und nicht gefälscht ist. Auch diese Modifikation erfüllt allerdings nicht die modernen Sicherheitsanforderungen, daher werden Verfahren wie RSA-PSS verwendet, um mit RSA zu signieren.

Siehe auch: Elektronische Unterschrift und Full-Domain-Hash

RSA mit dem Chinesischen Restsatz

Mit Hilfe des Chinesischen Restsatzes können Nachrichten effizienter entschlüsselt oder signiert werden. Weil der Modul sehr groß ist, sind auch die im Rechner verwendeten Bitdarstellungen der Zahlen sehr lang. Der Chinesische Restsatz erlaubt es, die Berechnungen statt in einer Gruppe der Größe gleichzeitig in den zwei kleineren Gruppen der Größe . und . auszuführen und das Ergebnis danach wieder zusammenzusetzen. Da hier die Zahlen wesentlich kleiner sind, ist diese Berechnung insgesamt schneller. Diese Variante wird nach der englischen Bezeichnung des Chinesischen Restsatzes CRT (Chinese remainder theorem) auch CRT-RSA genannt.

Der private Schlüssel besteht dann im Gegensatz zu dem, was im Rest dieses Artikels angenommen wird, aus folgenden Komponenten:

- , der *RSA-Modul*,
- , der *Entschlüsselungsexponent*,
- , die *erste Primzahl*,
- , die *zweite Primzahl* (ungleich p),
- ., häufig *dmp1* genannt,
- ., häufig *dmq1* genannt und
- ., häufig *iqmp* genannt.

Eine Nachricht . wird dann wie folgt signiert:

- .
- .
- .

Aus der letzten Gleichung sieht man sofort, dass . und .. Die Signatur stimmt also sowohl . als auch . mit . überein, daher ist nach dem Chinesischen Restsatz .. (Bemerkung: Die Identität . sieht man so: Modulo p gilt .. Die letzte Identität folgt aus dem kleinen fermatschen Satz. Analog erhält man ..)

RSA ist kein Primzahltest

Wenn Primzahlen sind, funktioniert das RSA-Verfahren. Umgekehrt kann aber aus dem funktionierenden RSA-Verfahren nicht geschlossen werden, dass der Modul das Produkt zweier Primzahlen und ist, denn mit Carmichael-Zahlen funktioniert das Verfahren auch.

Sicherheit

Public-Key-Verschlüsselungs-Verfahren wie RSA werden in der Praxis immer als hybride Verfahren in Verbindung mit symmetrischen Verfahren verwendet. Bei der Analyse der Sicherheit im praktischen Einsatz müssen die Sicherheit des Public-Key-Verfahrens und die praktische Sicherheit des Gesamtsystems betrachtet werden. Angriffe auf das RSA-Verfahren erfolgen oft über Seitenkanäle. Das Gesamtsystem kann unsicher sein, wenn nur eine Komponente, beispielsweise ein Computer, kompromittiert wurde.

Beziehung zwischen RSA und dem Faktorisierungsproblem

Bei der Kryptoanalyse des RSA-Verfahrens unterscheidet man zwischen zwei Problemen:

- RSA-Problem ( ): Gegeben sind der öffentliche Schlüssel sowie der Geheimtext . Gesucht wird der Klartext , wobei gilt:

  Das Problem liegt hier in der Schwierigkeit, -te Wurzeln modulo zu ziehen, was zur Bestimmung der Nachricht notwendig ist.

- RSA-Schlüsselproblem ( ): Gegeben ist der öffentliche Schlüssel . Gesucht wird der geheime Schlüssel , wobei gilt:

  Das Problem liegt hier in der Schwierigkeit, die Eulersche φ-Funktion von *ohne* Kenntnis der Faktoren *p* und *q* zu berechnen.

Folgende Beziehungen zwischen den RSA-Problemen und , dem Faktorisierungsproblem, sind bekannt:

.

Die Beziehung ist trivial, denn wenn man den privaten Schlüssel hat, kann man damit wie oben jeden beliebigen Geheimtext entschlüsseln. Ob die Umkehrung gilt, ist zurzeit unbekannt.

Auch die Beziehung ist trivial, denn wenn man faktorisiert hat, kann man damit leicht berechnen, und dann mit dem euklidischen Algorithmus zu gegebenem öffentlichen Schlüssel den zugehörigen privaten Schlüssel effizient berechnen, wie in der Schlüsselerzeugung.

Für die Beziehung ist schon lange ein *probabilistischer* Polynomialzeitalgorithmus bekannt. Vor kurzem wurde gezeigt, dass sich diese Reduktion im balancierten RSA (d. h. und haben gleiche Bitlänge) auch *deterministisch* durchführen lässt. Der Beweis verwendet das Coppersmith-Verfahren zur Bestimmung von Nullstellen eines irreduziblen bivariaten Polynoms mit ganzzahligen Koeffizienten, welches sich auf eine Gitterbasenreduktion zurückführen lässt.

Da alle gängigen Implementierungen balanciertes RSA verwenden, ist in der Praxis das Brechen des geheimen Schlüssels nur mit der Kenntnis des öffentlichen Schlüssels genau so schwer wie das Faktorisieren von . Wegen ist im Fall der zusätzlichen Kenntnis eines Geheimtexts die Schwierigkeit des Faktorisierungsproblems von zentralem Interesse.

Schwierigkeit des Faktorisierungsproblems

Man möchte . für sehr große Primzahlen . und . faktorisieren. Diese Primfaktorzerlegung ist für große Zahlen mit den heute bekannten Verfahren praktisch nicht durchführbar. Es ist aber nicht bewiesen, dass es sich bei der Primfaktorzerlegung um ein prinzipiell schwieriges Problem handelt.

Mit dem Quadratischen Sieb wurde 1994 die Zahl RSA-129 mit 129 Dezimalstellen in 8 Monaten von ca. 600 Freiwilligen faktorisiert. Mit der Methode des Zahlkörpersiebs wurde im Jahr 2005 von Wissenschaftlern der Friedrich-Wilhelms-Universität Bonn die im Rahmen der RSA Factoring Challenge von RSA Laboratories vorgegebene 200-stellige Dezimalzahl RSA-200 in ihre zwei großen Primfaktoren zerlegt[9]. Die ersten RSA-Zahlen bis RSA-500 wurden entsprechend der Anzahl der Dezimalstellen benannt, weitere RSA-Zahlen nach der Anzahl der Binärstellen. Die Faktorisierung begann Ende 2003 und dauerte bis Mai 2005. Unter anderem kam ein Rechnerverbund von 80 handelsüblichen Rechnern an der Universität Bonn zum Einsatz. Im November 2005 zahlten RSA Laboratories für die Faktorisierung von RSA-640, einer Zahl mit 640 Bits bzw. 193 Dezimalstellen, eine Prämie von 20.000 US-Dollar.[10] Obwohl mittlerweile für das Faktorisieren der RSA-Challenge-Zahlen keine Prämien mehr gezahlt werden, wurde im Dezember 2009 die Zahl RSA-768 faktorisiert.[11]

Für die Faktorisierung von RSA-1024 (309 Dezimalstellen) oder gar RSA-2048 (617 Dezimalstellen) waren 100.000 $ bzw. 200.000 $ ausgelobt; die RSA Laboratories haben im Mai 2007 die RSA Factoring Challenge beendet, nachdem die o. g. Wissenschaftler der Universität Bonn im selben Monat eine 1039-Bit Mersennezahl (312 Dezimalstellen) faktorisiert hatten.[12] Aufgrund der ungleichen Stellenzahl der Faktoren war das aber wesentlich leichter, als eine RSA-Zahl gleicher Länge zu faktorisieren. Die wachsende Rechenleistung moderner Computer stellt für die kurzfristige Sicherheit von RSA im Wesentlichen kein Problem dar, zumal diese Entwicklung vorhersehbar ist: Der Nutzer kann bei der Erzeugung seines Schlüssels darauf achten, dass der während der Dauer der beabsichtigten Verwendung nicht faktorisiert werden kann. Nicht vorhersehbare Entwicklungen wie die Entdeckung deutlich schnellerer Algorithmen oder gar Schaffung eines leistungsfähigen Quantencomputers, der die Faktorisierung von Zahlen durch Verwendung des Shor-Algorithmus effizient durchführen könnte, bergen zumindest für die mittel- und langfristige Sicherheit der verschlüsselten Daten gewisse Risiken.

Zum konkreten Sicherheitsniveau bestimmter Schlüssellängen gibt es unterschiedliche Aussagen.[13] Laut Bundesnetzagentur sind für RSA-basierte Signaturen bis Ende 2020 Schlüssel mit einer Mindestlänge von 1976 Bit geeignet (Empfehlung 2048 Bit). Für Signaturverfahren nach den Anforderungen aus § 17 Abs. 1 bis 3 SigG, „für die die besten bekannten Angriffe auf dem Problem der Faktorisierung großer Zahlen oder auf dem Problem der Berechnung diskreter Logarithmen in endlichen Körpern beruhen (RSA und DSA), werden Schlüssellängen von mindestens 3 000 Bit verpflichtend werden“, um perspektivisch mindestens ein Sicherheitsniveau von 120 Bit zu etablieren.[6]

Schwierigkeit des RSA-Problems

In einigen Spezialfällen kann man das RSA-Verfahren brechen, ohne das Faktorisierungsproblem gelöst zu haben. Der Angriff von Wiener bei balanciertem RSA löst das RSA-Schlüsselproblem effizient unter der Annahme, dass der private Schlüssel nur eine geringe Bitlänge aufweist, genauer . . Wiener verwendete dabei die Tatsache, dass unter der Abschätzung für . der Bruch . (für eine ganze

Zahl) in der Kettenbruchentwicklung von auftaucht. Die Schranke wurde mit Mitteln der Gitterbasenreduktion auf verbessert.

Auch das RSA-Problem kann unter einigen Annahmen effizient ohne Faktorisieren gelöst werden. Der Angriff von Håstad ermittelt einen Klartext, der mit kleinem Verschlüsselungsexponenten (etwa) für mehrere Empfänger vor dem Verschlüsseln speziell aufbereitet wurde, etwa wenn die Nummer des Empfängers in den Klartext codiert wurde. Dieser Angriff verwendet die Coppersmith-Methode, um kleine Nullstellen eines Polynoms in einer Unbestimmten zu berechnen, welche wiederum auf Gitterbasenreduktion beruht.

Angriffe gegen das unmodifizierte RSA-Verfahren („Textbook-RSA“)

RSA ist in der oben beschriebenen Version, die auch als „Textbook-RSA“ bekannt ist, weder als Verschlüsselungs- noch als Signaturverfahren geeignet, da es in beiden Fällen auf gravierende Weise unsicher ist und als Signaturverfahren auch keine langen Nachrichten signieren kann.

Die RSA-Verschlüsselung ist deterministisch. Das erlaubt es einem Angreifer, einen Klartext zu raten, ihn mit dem öffentlichen Schlüssel zu verschlüsseln und dann mit einem Chiffrat zu vergleichen. Dies kann insbesondere bei sehr kurzen Nachrichten wie “Ja” und „Nein“ sehr praktikabel und verheerend sein. Hieraus folgt, dass unmodifiziertes RSA nicht IND-CPA-sicher ist, heute eine absolute Minimalanforderung an Verschlüsselungsverfahren.

Wenn der Klartext und der Verschlüsselungsexponent so klein sind, dass sogar ist, dann kann ein Angreifer die -te Wurzel aus ziehen und das Chiffrat auf diese Weise entschlüsseln. Wurzelziehen ist nur modulo einer großen Zahl schwierig, aber in diesem Fall kann als ganze Zahl betrachtet werden.

Wenn dieselbe Nachricht zu mehreren Empfängern geschickt wird, die zwar alle unterschiedliche (und teilerfremde) Moduli benutzen, aber als öffentlichen Schlüssel den gleichen Exponenten , dann kann aus Nachrichten mit dem Chinesischen Restsatz berechnet werden. Weil (nach Voraussetzung ist für alle), kann diese Zahl wieder als in den ganzen Zahlen liegend aufgefasst werden und Wurzelziehen ist dort einfach. Dieser Angriff wird nach seinem Entdecker Johan Håstad als „Håstads Angriff“ bezeichnet.[14]

Da die RSA-Funktion multiplikativ ist (d. h. gilt), kann man aus jedem Chiffrat ein weiteres gültiges Chiffrat erzeugen. Wenn man den Besitzer des zugehörigen geheimen Schlüssels davon überzeugen kann, diese Zahl zu entschlüsseln oder zu signieren, kann man aus dem Ergebnis leicht gewinnen.

Dieselbe Eigenschaft erlaubt auch einen Angriff auf das Signaturverfahren. Aus bekannten Klartext-Signaturpaaren lassen sich weitere gültige Signaturen

zu Nachrichten

berechnen.

Padding

Um solche Angriffe zu verhindern, werden bei RSA-Verschlüsselung und RSA-Signatur sogenannte Padding-Verfahren eingesetzt. Standards für Padding-Verfahren für RSA werden z. B. in PKCS#1 oder ISO 9796 definiert. Diese nutzen aus, dass die Länge des Klartextes bzw. Hash-Wertes deutlich

kleiner als die Länge von ist, und fügen dem Klartext bzw. dem Hash-Wert vor der Verschlüsselung oder Signatur eine Zeichenfolge R mit vorgegebener Struktur an, die unter mehreren möglichen zufällig gewählt wird und dadurch das Chiffrat randomisiert. Es wird also die RSA-Funktion nicht auf die Nachricht oder auf den Hash-Wert angewendet, sondern auf den Klartext (bzw. seinem Hashwert) mit angehängtem . In der Regel enthält eine Angabe über die Länge der Nachricht oder des Hash-Wertes oder eine eindeutige Zeichenfolge, die den Beginn von kennzeichnet. Dies erleichtert nicht nur die Dekodierung, sondern erschwert auch Angriffe. Padding-Verfahren können für die Berechnung von auch Zufallszahlen und Hashfunktionen verwenden. Einige moderne Paddingverfahren – beispielsweise das Optimal Asymmetric Encryption Padding (OAEP) oder das Probabilistic Signature Scheme (PSS) – verwenden kryptographische Hashfunktionen, um den Klartext vor der Verschlüsselung weiter zu randomisieren, und sind unter idealisierenden Annahmen an die verwendete Hashfunktion beweisbar sicher unter der RSA-Annahme.[15][16]

Chosen-Ciphertext-Angriff

Daniel Bleichenbacher stellte 1998 einen Angriff auf die in PKCS#1 v1 spezifizierte RSA-Verschlüsselung vor. Dabei nutzte er aus, dass PKCS#1 v1 ein Nachrichtenformat vorgibt und einige Implementierungen nach dem Entschlüsseln Fehlermeldungen ausgeben, falls dieses Format nicht eingehalten wurde. Um den Angriff gegen ein Chiffrat durchzuführen, wählt man eine Zahl und berechnet daraus ein neues Chiffrat . Bei dem Nachrichtenformat sind die ersten zwei Bytes 00 und 02, wenn also keine Fehlermeldung kommt, weiß man, dass sowohl bei der ursprünglichen Nachricht als auch bei der neuen Nachricht die ersten beiden Bytes 00 02 sind. Mehrfache Wiederholung mit geschickt gewählten erlauben es, nach und nach den gesamten Klartext aufzudecken.[17] RSA nach PKCS#1 ab Version 2 ist immun gegen diesen Angriff.

Sicherheit hybrider Verfahren

RSA wird aus Effizienzgründen in der Regel in Hybridverfahren mit symmetrischen Verfahren kombiniert. Zur hybriden Verschlüsselung wird zufällig ein Sitzungsschlüssel für ein symmetrisches Verschlüsselungsverfahren generiert, der dann per RSA verschlüsselt und zusammen mit der Nachricht übertragen wird. Zum Signieren wird nicht die gesamte Nachricht, sondern nur ein Hash-Wert signiert.

Für die Sicherheit von RSA sind Primzahlen mit mehreren hundert Dezimalstellen (mindestens 2048 Bit) erforderlich. Damit können symmetrische Schlüssel jeder üblichen Länge verschlüsselt werden. Gängige Verfahren zur symmetrischen Verschlüsselung basieren beispielsweise auf der Blockchiffre AES mit einer Schlüssellänge von 128, 192 oder maximal 256 Bit.

Eine sichere Hashfunktion wie SHA-2 erzeugt Hashwerte mit einer Länge von 224 bis 512 Bit. Damit lassen sich Signaturverfahren mittels RSA realisieren, die nur einen Signaturschritt benötigen.

Die Sicherheit des Gesamtsystems hängt sowohl im Fall der Verschlüsselung als auch der Signatur von der Sicherheit beider verwendeter Verfahren ab. Da bei RSA für ein ähnliches Sicherheitsniveau wie beim symmetrischen Verfahren deutlich längere Schlüssel nötig sind, wird die Sicherheit des Hybridverfahrens meistens von der Sicherheit des Public-Key-Verfahrens bestimmt.

Vollständiges Beispiel

Anmerkung

- RSA direkt auf Texte anzuwenden, birgt erhebliche Risiken. RSA wird deshalb, anders als im Beispiel, in der Praxis praktisch nur in Kombination mit anderen Verfahren verwendet. (Siehe: Hybride Verschlüsselung und Abschnitt Angriffe gegen das unmodifizierte RSA-Verfahren.)
- Um das Beispiel übersichtlich zu halten, wurden relativ kleine Primzahlen verwendet. Zur sicheren Verschlüsselung werden typischerweise mindestens 600-stellige empfohlen.[18]

Vorarbeiten

Die oben genannten Schritte sollen nun an einem vollständigen Beispiel erläutert werden. Um einen Text zu verschlüsseln, müssen zunächst Buchstaben in Zahlen umgewandelt werden. Dazu verwendet man in der Praxis zum Beispiel den ASCII-Code. Hier sei willkürlich die folgende Zuordnung gewählt:

```
Leerzeichen=00 A=01 B=02 C=03 usw.
```

Darüber hinaus sei angenommen, dass jeweils drei Zeichen zu einer Zahl zusammengefasst werden. Die Buchstabenfolge AXT wird also zu 012420. Die kleinste zu verschlüsselnde Zahl ist dann 000000 (drei Leerzeichen), die größte 262626 (ZZZ). Der Modul muss also größer als 262626 sein.

```
Klartext:   W  I  K   I  P  E   D  I  A
Kodierung: 23 09 11  09 16 05  04 09 01
```

Schlüsselerzeugung

Zunächst werden geheim zwei Primzahlen gewählt, beispielsweise und . Damit ergibt sich:

-
-
- (zufällig, teilerfremd zu)
- (das multiplikative Inverse zu mit Hilfe des erweiterten euklidischen Algorithmus)

Öffentlicher Schlüssel:

- und

Privater Schlüssel:

- und

Verschlüsselung

$C_n = K_n^e \bmod N \quad \text{für } n=1,2,3(,...)$

$C_1 = 230911^{1721} \bmod 263713 = 001715$

$C_2 = 091605^{1721} \bmod 263713 = 184304$

$C_3 = 040901^{1721} \bmod 263713 = 219983$

Entschlüsselung

$K_n = C_n^d \bmod N \quad \text{für } n=1,2,3(,...)$
$K_1 = 001715^{1373} \bmod 263713 = 230911$
$K_2 = 184304^{1373} \bmod 263713 = 091605$
$K_3 = 219983^{1373} \bmod 263713 = 040901$

Signatur

$C_n = K_n^d \bmod N$
$C_1 = 230911^{1373} \bmod 263713 = 219611$
$C_2 = 091605^{1373} \bmod 263713 = 121243$
$C_3 = 040901^{1373} \bmod 263713 = 138570$

Verifikation

$K_n = C_n^e \bmod N$
$K_1 = 219611^{1721} \bmod 263713 = 230911$
$K_2 = 121243^{1721} \bmod 263713 = 091605$
$K_3 = 138570^{1721} \bmod 263713 = 040901$

Die Berechnung der modularen Exponentiation kann durch binäre Exponentiation (Square-and-multiply) beschleunigt werden.

Dabei wendet man nach jedem Rechenschritt auf die zu handhabenden Zahlen die Modulo-Operation „mod“ an, um die Zwischenergebnisse möglichst klein zu halten. Aus dem Klartext „7“ erhalten wir somit den Geheimtext „2“.

Programmierung

Das folgende Programm in der Programmiersprache C++ zeigt die Implementierung des RSA-Verfahrens mit , und , was den privaten Schlüssel ergibt. Das Programm gibt den verschlüsselten und den entschlüsselten Text (in diesem Beispiel *Werde Mitglied bei Wikipedia!*) auf der Konsole aus.

```
#include <iostream>
using namespace std;

// berechnet aus dem öffentlichen Schlüssel e den privaten Schlüssel (das
multiplikative Inverse von e modulo phi)
unsigned getPrivateKey(unsigned e, unsigned phi)
{
        unsigned a = phi, b = e;
        unsigned d = 0, u = 1;
        while (b != 0)
        {
                unsigned q = a / b;
                unsigned x = b; // Variable zum Zwischenspeichern
                b = a - q * b;
                a = x;
                x = u;
                u = d - q * u;
                d = x;
        }
        if (a > 1) {
```

```
                cout << "Fehler: e und phi nicht teilerfremd" << endl;
                exit(1);
        }
        return d;
}

// Diese Funktion berechnet die Potenz a ^ b modulo n
unsigned modularPower(unsigned a, unsigned b, unsigned n)
{
        unsigned res = (b & 1) ? a : 1;
        b >>= 1;
        while (b)
        {
                a = a * a % n;
                if (b & 1) res = res * a % n;
                b >>= 1;
        }
        return res;
}

// Diese Funktion ver- oder entschlüsselt den Klartext 'text' elementweise mit
dem Schlüssel 'key'
void RSA(unsigned *text, int len, unsigned n, unsigned key)
{
        for (int i = 0; i < len; i++) // for-Schleife, die die Zeichen des
Textes durchläuft
        {
                unsigned c = modularPower(text[i], key, n); // ver- bzw.
entschlüsselt text[i]
                text[i] = c;
        }
}

int main()
{
        unsigned p = 223; // die Primzahlen p und q
        unsigned q = 127;
        unsigned n = p * q;
        unsigned e = 121; // Der öffentliche Schlüssel
        unsigned phi = (p - 1) * (q - 1);
        unsigned d = getPrivateKey(e, phi); // Erzeugt den privaten Schlüssel

        const char *klartext = "Werde Mitglied bei Wikipedia!";
        unsigned text[100];
        int len = 0;
        while (klartext[len])
                text[len] = klartext[len], ++len;

        RSA(text, len, n, e); // Verschlüsseln
        // Ausgabe des verschlüsselten Texts auf der Konsole:
        for (int i=0 ; i < len ; ++i)
                cout << ' ' << text[i];
        cout << '\n';

        RSA(text, len, n, d); // Entschlüsseln
        // Ausgabe des Dechiffrats:
        for (int i=0 ; i < len ; ++i)
                cout << (char)text[i];
        cout << '\n';
}
```

Anwendung

Hybride Verfahren

→ *Hauptartikel: Hybride Verschlüsselung*

RSA ist im Vergleich zu Verfahren wie 3DES und AES mindestens um den Faktor 100 langsamer. In der Praxis wird RSA daher meist nur zum Austausch eines Schlüssels für die symmetrische Verschlüsselung benutzt. Für die Verschlüsselung der Daten werden dann symmetrische Verfahren eingesetzt. Damit sind die Vorteile beider Systeme vereint: einfacher Schlüsselaustausch und effiziente Verschlüsselung.

Anwendungsgebiete

- Internet- und Telefonie-Infrastruktur: X.509-Zertifikate
- Übertragungs-Protokolle: IPsec, TLS, SSH, WASTE
- E-Mail-Verschlüsselung: OpenPGP, S/MIME
- Authentifizierung französischer Telefonkarten
- Kartenzahlung: EMV
- RFID-Chip auf dem deutschen Reisepass
- Electronic Banking: HBCI

Literatur

- Johannes Buchmann: *Einführung in die Kryptographie*. Springer-Verlag, Berlin 1999, ISBN 3-540-66059-3.
- *Der Dialog der Schwestern*. In: *c't*. Nr. 25, 1999 (Liegt auch dem E-Learning-Programm CrypTool bei).
- Alexander May: *Computing the RSA Secret Key is Deterministic Polynomial Time Equivalent to Factoring*. In: *Advances in Cryptology (Crypto 2004), Lecture Notes in Computer Science*. Band 3152. Springer Verlag, 2004, S. 213–219.
- Dan Boneh: *Twenty Years of Attacks on the RSA Cryptosystem*. In: *Notices of the American Mathematical Society (AMS)*. Band 46, Nr. 2, 1999, S. 203–213.
- Alfred Menezes, Paul C. van Oorschot, Scott A. Vanstone: *Handbook of Applied Cryptography*. CRC Press, 1996, ISBN 978-0-8493-8523-0 (archive.org).
- Thomas H. Cormen, Charles E. Leiserson, Charles E. Leiserson, Clifford Stein: *Introduction to Algorithms*. 2nd Auflage. MIT Press and McGraw-Hill, 2001, ISBN 978-0-262-03293-3, S. 881–887.

Weblinks

Wikibooks: Beweisarchiv: Korrektheit des RSA-Kryptosystems – Lern- und Lehrmaterialien

- Erklärung von RSA vom Fachbereich Mathematik der Universität Wuppertal
- Einfache Erklärung von RSA mit Online-Beispielen von Hans-G. Mekelburg
- *Arbeit über RSA mit Schwerpunkt in der Informatik.* Archiviert vom Original (nicht mehr online verfügbar) am 18. August 2012; abgerufen am 18. August 2012.

- RSA-Codebeispiel in Scheme vom Massachusetts Institute of Technology
- Beispiel zur RSA-Verschlüsselung vorgerechnet von Joachim Mohr
- Interaktive Präsentation des RSA-Verfahrens von CrypTool
- Whitfield Diffie and Martin E. Hellman: New Directions in Cryptography, IEEE Transactions on Information Theory, Vol. IT-22, No. 6, November 1976
- Video: *RSA: Einführung*. Christian Spannagel 2012, zur Verfügung gestellt von der Technischen Informationsbibliothek (TIB), doi:10.5446/19815.
- Video: *RSA: Konstruktion der Schlüssel*. Christian Spannagel 2012, zur Verfügung gestellt von der Technischen Informationsbibliothek (TIB), doi:10.5446/19816.
- Video: *RSA: Ver- und Entschlüsselung*. Christian Spannagel 2012, zur Verfügung gestellt von der Technischen Informationsbibliothek (TIB), doi:10.5446/19817.
- Video: *RSA: Beispiel Teil 1*. Christian Spannagel 2012, zur Verfügung gestellt von der Technischen Informationsbibliothek (TIB), doi:10.5446/19813.
- Video: *RSA: Beispiel Teil 2*. Christian Spannagel 2012, zur Verfügung gestellt von der Technischen Informationsbibliothek (TIB), doi:10.5446/19814.

Einzelnachweise

- R.L. Rivest, A. Shamir, and L. Adleman: *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. (mit.edu [PDF]).
- Whitfield Diffie, Martin E. Hellman: *New Directions in Cryptography*. (stanford.edu [PDF]).
- Gina Kolata: *Leonard Adleman; Hitting the High Spots Of Computer Theory*. In: *The New York Times*. 13. Dezember 1994, ISSN 0362-4331 (nytimes.com).
- C. C. Cocks: *A note on 'non-secret encryption'*. 1973 (Memento vom 27. Februar 2008 im *Internet Archive*)
- Patent US4405829: *Cryptographic communications system and method.* Veröffentlicht am 20. September 1983, Anmelder: Massachusetts Institute of Technology, Erfinder: Ronald L. Rivest, Adi Shamir, Leonard Adleman.
- Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen: Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung (Übersicht über geeignete Algorithmen) vom 21. Januar 2014 ()
- D. Boneh: *Twenty Years of Attacks on the RSA Cryptosystem*. In: *Notes of the AMS*. Band 46, Nr. 2, Februar 1999, S. 203–213 (PDF).
- MJ Wiener: *Cryptanalysis of short RSA secret exponents*. In: *IEEE Transactions on information theory*. Band 36, Nr. 3, Mai 1990, S. 553–558, doi:10.1109/18.54902.
- RSA Labs: *RSA-200 is factored!* (Memento vom 18. November 2007 im *Internet Archive*)
- MathWorld: *RSA-640 Factored*
- RSA Labs: *RSA-768 is factored!*
- Archivierte Kopie (Memento vom 22. Februar 2015 im *Internet Archive*)
- http://www.keylength.com/en/compare/
- Johan Håstad: *Solving Simultaneous Modular Equations of Low Degree*. In: *SIAM Journal on Computing*. Band 17, Nr. 2, 1988, S. 336–341 (Solving Simultaneous Modular Equations of Low Degree).
- What is OAEP? (englisch)
- What is PSS/PSS-R? (englisch)

- Daniel Bleichenbacher: *Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1*. In: *CRYPTO '98*. 1998, S. 1–12, doi:10.1007/BFb0055716.

18. *Kryptographische Verfahren: Empfehlungen und Schlüssellängen.* (PDF 1.4 (830kiB)) In: *BSI*. Bundesamt für Sicherheit in der Informationstechnik, 10. Februar 2014, S. 15, 28, archiviert vom Original (nicht mehr online verfügbar) am 22. Februar 2014; abgerufen am 13. Juli 2014 (Tabelle 1.2 und Tabelle 3.1 empfehlen Schlüssellängen von 2000Bit für RSA).

-

Kategorie:

- Asymmetrisches Verschlüsselungsverfahren