„RSA: Implementation and Security“ (Sommersemester 2014)

Thomas Stadler

Lehrstuhl für Technische Informatik

Zusammenfassung Diese Aufbereitung des Kapitel [1] aus dem Buch [2] legt sein Augenmerk auf die Realisierung der RSA Verschüsselungstechnik. Dabei werden sowohl die algorithmische als auch die hardwarespezifische Implementierung und deren Komplexität beleuchtet.

1 Einleitung

Der Wunsch nach gesicherter Übermittlung von Nachrichten ist wohl so alt wie die Kommunikation selbst. In der Geschichte etablierten sich mannigfache Möglichkeiten Texte zu Ver- und Entschlüsseln. Die Grundlage jeder Verschlüsselung ist in der Mathematik und der Berechnung eines Geeigneten Schlüssels zu finden. Je komplexer die Berechnung, desto schwieriger ist es, die Nachricht wieder zu dechiffrieren. Der große Sprung in der Versicherungstechnik gelang am Anfang des zwanzigsten Jahrhunderts mit der Computertechnologie, die es ermöglichte, komplexe Aufgaben an eine Maschine weiterzugeben. Dabei ist die Art und Weise der Verschlüsselung noch immer die selbe. Ein Klartext wird durch ein bestimmtes Schema in einen nicht lesbaren Text übersetzt und durch ein festgelegtes Schema wieder zurück-gelesen. Der Trick dabei ist recht simpel: Die beiden Kommunikationspartner brauchen einen Schlüssel für die jeweilige Übersetzung. Seit dem Siegeszug der Computer Mitte des zwanzigsten Jahrhunderts sind diese Schlüssel Zahlen, die in die Übersetzungs-Algorithmen eingespeist werden.

2 Schlüsseltypen

Man unterscheidet bei der Verschlüsselung zwei Kategorien: Die symmetrische und die asymmetrische Verschlüsselung. Der Unterschied der Typen liegt in den verwendeten Schlüssel.

2.1 Symmetrische Verschlüsselung

Bei dieser Chiffrierung wird ein Schlüssel erstellt, der sowohl zur Ver- als auch zur Entschlüsselung verwendet wird. Der Algorithmus, der verwendet wird um Klartext zu chiffrieren, benötigt also den gleichen Schlüssel um den Chiffretext wieder in einen Klartext zu übersetzen. Ein großer Nachteil dieser Technik ist also der

Schlüssel selbst. Dieser muss gesichert den Kommunikationspartnern übermittelt werden. Sollte der Schlüssel nach außen gelangen, ist die Verschlüsselung nichtig und kann jederzeit gelesen werden.

2.2 Asymmetrische Verschlüsselung

Bei diesem Typ werden zwei voneinander verschiedene Schlüssel erzeugt, ein öffentlicher und ein privater Schlüssel. Mit dem öffentlichen Schlüssel kann jeder einen Text für den Inhaber des privaten Schlüssels chiffrieren. Dabei ist sichergestellt, dass der verschlüsselte Text nur vom Kommunikationspartner entziffert werden kann. Selbst der Inhaber des öffentlichen Schlüssels kann den Chiffretext mit diesem nicht mehr entschlüsseln. Trotzdem muss der private Schlüssel stets geheimgehalten werden, denn nur mit ihm können Nachrichten entschlüsselt werden. Durch die Verwendung eines gesonderten Schlüssels stellt diese Art ein Plus an Sicherheit dar, da nicht mehr der eine Schlüssel aus dem symmetrischen Verfahren offen verwendet werden muss, um Texte zu chiffrieren. Ein weiterer Aspekt der Sicherheit ist, dass der private Schlüssel nicht aus dem öffentlichen berechnet werden kann. Sollte also der öffentliche Schlüssel in fremde Hände gelangen, ist es weiterhin nicht möglich, Chiffretexte zu entschlüsseln. Der im Nachfolgenden genauer erklärte RSA Algorithmus zählt zu den asymmetrischen Verfahren.

3 RSA

Der RSA Algorithmus ist sehr bekannt und weit verbreitet. Doch trotz seines Bekanntheitsgrades gilt diese Verschlüsselung als sehr sicher. Ein Grund für die Verbreitung des Verfahrens ist wohl die einfache mathematische Herleitung des Algorithmus. In diesem Kapitel wird auf die Herleitung und die hardwarespezifische Umsetzung eingegangen.

3.1 mathematische Herleitung

Der erste Schritt der Berechnung ist die Wahl von zwei voneinander verschiedenen, großen und zufälligen Primzahlen, im Weiteren werden diese als P und Q bezeichnet. Die beiden Primzahlen werden verwendet, um zwei weitere Zahlen für die Rechnung zu ermitteln, $N = P \cdot Q$ und $\psi = (P-1) \cdot (Q-1)$. Als nächstes wird eine Zahl E ermittelt, für die gilt: $1 < E < \psi$ und E ist teilerfremd zu ψ. Durch E und ψ wird eine weitere Zahl berechnet, für die gilt: $D \cdot E = 1 mod \psi$. Nach der Ermittlung aller benötigten Zahlen, N, E und D, können die Schlüssel erzeugt werden. Dabei werden die Zahlen N und E für den öffentlichen Schlüssel verwendet, N und D für den privaten.
Um nun einen Klartext in einen Chiffretext zu überführen, muss die Nachricht in eine natürliche Zahl I umgewandelt werden. Unter Verwendung von E und N kann die Umrechnung wie folgt ausgeführt werden:
$C = I^E mod N$

Das Dechiffrieren funktioniert ähnlich der Verschlüsselung. Mit den Zahlen N und D des privaten Schlüssels wird die Nachricht folgendermaßen zurückgerechnet:
$I = C^D mod N$

Interessanterweise gestaltet sich die mathematische Herleitung des Algorithmus sehr einfach und die Ver- und Entschlüsselung können mit jeweils einem mathematischen Ausdruck ermittelt werden. Um die Schwierigkeiten der Implementation des RSA zu verstehen, müssen die Herleitungsschritte genauer betrachtet werden.

Sicherheit der Herleitung Schon der erste Schritt der Herleitung entpuppt sich als der vielleicht Schwierigste und zugleich Wichtigste. Aus zwei großen, voneinander verschiedenen und zufälligen Primzahlen wird durch Multiplikation die Zahl N erzeugt, die sowohl Teil des privaten als auch des öffentlichen Schlüssels ist. Sollte der öffentliche Schlüssel eines Kommunikationspartners bekannt werden, wäre es für einen Angreifer ein Leichtes, alle anderen Zahlen zu ermitteln und den Algorithmus zu knacken, sofern er die beiden Primzahlen errechnen kann. Doch hier zeigt sich eine Barriere, die Teil des Fundamentalsatzes der Arithmetik ist: Es existiert kein Verfahren, das die Primfaktorzerlegung einer Zahl in polynomialer Zeit errechnen kann.
Jedoch kann man mit modernen Computern alle Möglichkeiten der Faktorisierung durchprobieren. Um dieser einfachen Methode entgegenzuwirken, müssen die Zahlen P und Q sehr groß gewählt werden. Damit kann sichergestellt werden, dass es unrealistisch ist, dass ein Brute-Force-Anschlag in abwartbarer Zeit zum Erfolg führt. Zur gegenwärtiger Zeit ist eine Größenordnung von 1024 oder 2048 Bit für die Primzahlen gängig.

Komplexität der Herleitung Der Vorteil, der sich für die Sicherheit aus der Wahl der großen Primzahlen ergibt, ist jedoch ein Nachteil für die Komplexität des Algorithmus, da die Zahlen dadurch sehr schwer zu bestimmen sind. Dabei stellt jede Forderung an die Zahlen ein Plus an Sicherheit auf Kosten der Komplexität dar.
Um einer Brute-Force-Attacke entgegenzuwirken, müssen die Zahlen sehr groß sein. Zudem müssen sie voneinander verschieden sein, da ein Angreifer sonst ein Quadrat erkennen und damit die Zeit der Primfaktorzerlegung signifikant senken könnte. Die letzte Anforderung ist dabei die wohl Wichtigste und zugleich am schwierigsten zu Erzeugende. Die Zahlen müssen prim sein, da sonst das Produkt der beiden mehr Teiler hat als die Zahlen selbst hat.
Das Erzeugen von Primzahlen gestaltet sich jedoch sehr schwer. In einer Größenordnung 2^{512} ist eine von 200 Zahlen prim. Dennoch kann ein Zahlengenerator bei ausreichender Zeit eine Primzahl finden. Doch der gefundene Wert muss noch validiert werden. Der Standardprozess für die Prüfung ist es, die vermeintliche Primzahl P durch alle Zahlen des Intervalls $[1;\sqrt{P}]$ zu teilen. Trotzdem ist die

Berechnung deterministisch und in polynomialer Zeit lösbar.
Ein weiteres grundsätzliches Problem sind die beiden Produkte für N und ψ mit den Multiplikatoren P und Q. Kosten und Zeit eines Systems, das den RSA verwendet, werden direkt von der Wahl des Multiplikationssystems beeinflusst. Absolut sicher ist der RSA Algoryithmus jedoch nicht, da er bei ausreichender Zeit geknackt werden kann. Werden die Primzahlen jedoch so hoch angesetzt, dass es unrealistisch wird den Algorithmus in polynomialer Zeit zu knacken, so ist diese Zeit zugleich das Herzstück der Sicherheit dieser Verschlüsselung.

3.2 Hardware-Implementierung

An der Hardware-Implementierung zeigt sich die Schwierigkeit der Multiplikation von großen Zahlen. Wie oben erwähnt, ist die Grundlage des Algorithmus die Multiplikation von zwei mindestens 512 BIt großen Zahlen. Im Folgenden wird auf verschiedene Möglichkeiten eingegangen, diese Problematik zu behandeln und in ein RSA Verschlüsselungssystem zu integrieren.

Kombinatorischer Multiplizierer Für die Multiplikation zweier n-bit-Zahlen werden n^2 Volladdierer benötigt. Diese werden zu einer n x n großen Matrix zusammengeschlossen, die das Ergebnis der Multiplikation nach dem selben Prinzip wie bei der Multiplikation von Hand ermitteln. Jede Reihe des Kombinatorischen Multiplizierers berechnet dabei ein Teilergebnis, die wiederum am Ende zusammenaddiert werden. Diese Methode berechnet das Ergebnis in konstanter Zeit. Die Anzahl der Volladdierer, die jedes Bit durchlaufen muss, ist jedoch sehr hoch und wächst exponentiell mit der Größe der Primzahlen P und Q. So wird die konstante Zeit unausführbar lange. In der Praxis kommt der Kombinatorische Multiplizierer wie hier dargestellt aufgrund der Zeit und Kosten sehr selten zum Einsatz.

bit serieller Multiplizierer Diese Methode verwendet zwei Eingaben und eine Ausgabe sowie zusätzlich ein Taktsignal. Hierbei wird dem Zyklus jeweils ein Bit der Eingaben zugeführt. Typischerweise vom least-significant Bit bis zum most-significant BIt. Während jedem Takt wird ein Teilergebnis zu einer Gesamtsumme addiert und das least-significant Bit der Gesamtsumme wird in die Ausgabe verlagert.
Ein Beispiel Multiplizierer könnte wie folgt aufgebaut sein: Eine Zelle besteht aus einem 5-3 Komprimierungs Modul, welches 5 gleich gewichtete Eingaben auf 3 Ausgaben verdichtet und einigen Flip-Flops. Um mit diesem Aufbau zwei n-Bit Zahlen zu Multiplizierer benötigt man n solcher Zellen. Das System errechnet das Ergebnis in 2n+1 Takten.
Der größte Vorteil gegenüber dem Kombinatorischen Multiplizierer ist die Verringerung der benötigten Fläche. Brauchte die erste Variante noch einen Platz von n^2 Volladdierern, kann das System auf eine lineare Fläche unter Beachtung der Operandenlänge verkleinert werden.

Hybrid seriell/parallel Unter Zuhilfenahme eines fest zugeordneten traditionellen Sammler kann ein hybrid aus seriellem und parallelem Multiplizierer aufgebaut werden. Dabei wird ein Operand seriell und einr parallel bereitgestellt. Der Parallele muss dabei über die gesamte Dauer konstant bleiben. Der serielle Operand wir dagegen Bitweise vom least-significant Bit zum most-significant Bit übergeben. Dabei wird jedes Bit untersucht. Ist es eine 1, so wird dem mit Null initialisierten Sammler der parallele Operand hinzu addiert. Bei einer Null wird keine Addition ausgeführt. Nach diesem Schritt wird das untersuchte Bit aus dem Multiplikationszyklus ausgelagert. Dieser Prozess wird für alle 2n+1 Bit des Produkts ausgeführt.
Dieses System hat einen weit weniger komplexen Aufbau als die beiden zuvor Besprochenen. Der Entwurf kann auf eine lineare Fläche unter Beachtung der Operandenlänge abgebildet werden.

3.3 Sicherheitsanalyse

Nach der Tiefenanalyse der Hardware-Implementierung ist es an der Zeit, die Anforderungen an ein Kryptosystem zu prüfen. Im letzten Kapitel ist klar geworden, wie umfangreich und komplex die Berechnungen der Multiplikationen zu Stande kommen. Dennoch muss sichergestellt sein, dass immer das richtige Ergebnis ermittelt wird. Denn bei einem so komplexen Bauteil können sich kleine Fehler einschleichen, die dazu führen können, dass der Algorithmus nicht mehr funktioniert. Das würde heißen, dass der Chiffretext nicht mehr decodiert werden kann. Weiter müssen die gewünschten Längen der Schlüssel so gewählt werden, dass sie einen ausreichenden Schutz bieten. Ferner muss ein solches System robust sein, was Angriffe angeht, seien es bereits bekannte Methoden oder auch nur theoretische Überlegungen.
Kernstück der Sicherheit stellt die Lösbarkeit des Problems der Primfaktorzerlegung dar. Daher ist jede Anforderung 'groß', 'zufällig', 'voneinander verschieden' und 'prim' ein plus an Sicherheit.
Seit der Publizierung des Algorithmus 1979 wuchs die Größe der Zahlen stetig an. Geschuldet ist das den modernen Prozessoren und deren Rechengeschwindigkeit. Daher müssen die Zahlen so groß gewählt werden, dass eine Brute-Force-Attacke in polynomialer Zeit unwahrscheinlich ist. Momentan gelten Zahlen im Bereich von 2048 oder 4096 als sicher.
Wenn in der Zufälligkeit eine Unausgewogenheit herrscht, kann ein Angreifer das Zahlenfeld eingrenzen und damit die Zeit, die er brauchen würde die Primfaktorisierung durchzuführen, signifikant verkürzen.
Sollten die beiden Zahlen gleich sein, so kann man aus dem Produkt eine Quadratzahl erkennen und die Faktorisierung sofort durchführen. Auch wenn der Abstand der Zahlen zu gering oder viel zu groß ist, wird die Verschlüsselung anfällig für andere Arten die Faktorisierung zu knacken.
Die letzte Anforderung, dass die Zahlen prim sein müssen, ist wie bereits angesprochen die Wichtigste und zugleich die am schwierigsten zu Kontrollierende. Denn wenn die Zahlen nicht prim wären, so würden mehrere Faktorisierungen existieren. Das würde die Komplexität des Algorithmus enorm vermindern.

Ein weiterer Ansatz für Angreifer ist die Zahl E, die im Kapitel der Herleitung als $1 < E < \psi$ teilerfremd zu ψ definiert wurde. In der Entschlüsselung $C = I^E mod N$
ist E ein teil des privaten Schlüssels und damit sehr interessant für Angriffe. Um solche Attacken auszuschließen wird für die Zahl E in herkömmlichen RSA Systemen die Fermat-Zahl $2^{16} + 1$ verwendet
Bis heute gilt es als praktisch unmöglich die RSA Verschlüsselung auf algorithmischem Wege zu knacken. Für eine sichere Verschlüsselung genügt es nicht, nur algorithmisch stark zu sein, auch die Implementierung muss sicher sein. Neue Arten der Angriffe richten sich gezielt auf bestimmte Implementierungs-Arten.
So gesehen ist der Algorithmus sicher, wenn Angreifer eine verschlüsselte Nachrichten abfangen und versuchen, diese zu entschlüsseln. Doch falls ein Angreifer zudem Zugang zu der physikalischen Schicht des Systems hat, eröffnet ihm das neue Möglichkeiten, vor allem durch die Anwendung sogenannter 'side-channel Attacken'. Diese Angriffe richten sich auf ungewollte Informationslecks, vor allem auf die zeitliche Aktivität und den Stromverbrauch der Hardware. Die Techniken, die diese Channels analysieren, haben ähnliche Konzepte und brauchen nicht zwingend Systemzugriff.
Angenommen ein Angreifer weiß, welches Verfahren der Hardware Implementierung angewandt wurde, in Annahme den serielI/parallel Multiplizierer. Dieses System errechnet den Chiffretext durch die Quadrierung des Normaltextes und die anschließende Subtraktion des Modulus von N. Dieser Prozess wird E - 1 mal wiederholt. Eventuell wurde das System so entworfen, dass das zuvor errechnete Produkt Teil der parallelen Multiplikation ist, während der Normaltext der Operand der seriellen Multiplikation ist. Daraus folgt, dass für die Berechnung des Chiffretextes C der Normaltext N E mal seriell in den Muliplizierer gespeist wird. Hier wird jedes Bit des seriellen Operanden untersucht und dem Speicher hinzu addiert. Wenn das aktuelle Bit 1 ist, wird die Addition durchgeführt, falls es 0 ist, wird keine Addition durchgeführt. Vom algorithmischen Standpunkt aus ist diese Entscheidung irrelevant, auf die Zeit- und Strom-Channels hat dies jedoch Auswirkungen. So hat etwa die Durchführung der Addition eine längere Zeit oder einen höheren Stromverbrauch zur Folge. In jedem Fall kann ein Angreifer diese Abweichungen messen. Solche Informationen lassen Rückschlüsse auf die geheime Berechnung zu. Einfache Änderungen im System könnten solche Attacken zu Nichte machen. So könnte zum Beispiel immer eine Addition durchgeführt, das Ergebnis gegebenen Falls aber verworfen werden.
Der hybrid Multiplizierer ist jedoch nicht das einzige System, dass solche Informationen Preis gibt. Der kombinatorische Multiplizierer mit seiner Größe macht ihn anfällig für side-channel Attacken. Hier können durch die hohen Durchlaufzahlen und der daraus resultierenden Temperatur Rückschlüsse gezogen werden, welche Zeilen und Spalten beteiligt waren.
Vollständige Sicherheit kann also nur erreicht werden, wenn sowohl auf die korrekte Ausführung des Algorithmus achtgegeben wird, als auch auf eine Implementierung, die kaum Rückschlüsse auf die Berechnung gibt.

3.4 Fazit

Seit seiner Entwicklung 1979 hat sich am Prinzip der RSA Verschlüsselung nichts geändert. Mit dem Fortschritt der Technik ist der RSA Standartteil der sicheren Kommunikation geworden. Aktuelle Rechner können die Schlüsselpaare der asymmetrischen Verschlüsselung nativ berechnen. Doch so leicht es auch ist, den Algorithmus mathematisch zu bestimmen, so beängstigend ist es ihn hardwarespezifisch umzusetzen. Wie gezeigt wurde ist die mathematische Herleitung einfach zu verstehen, doch durch die anforderung der Operanten, groß, voneinander verschieden, zufällig und prim gewinnt die Methodik an Schwierigkeit. Vor allem beim Versuch ohne genaue Kenntnisse der Operatoren diese zu ermitteln. Doch eben diese Faktoren, die die Sicherheit gewährleisten, sind hardwarespezifisch sehr aufwendig zu realisieren. Sowohl im Hinblick auf die Berechnungszeit und der Kosten des Systems müssen vor der Implementierung auch mögliche Angriffe durchdacht werden, welche auf das Überwachen der Hardwareressourcen ausgelegt sind.
Der RSA Algorithmus verändert sich ständig durch kleine Änderungen weiter. Wie zum Beispiel die Verwendung einer Fermat Primzahl als Verschlüsselungs-Exponenten. Andere Änderungen wie den Ausschluss der side-channel Attacken bedürfen etwaiger Änderungen, die eventuell großen Einfluss auf die Komplexität, den Platz oder den Stromverbrauch des Systems haben. Wie bei jeder anderen Verschlüsselung ist das Ziel der Ausschluss derer, die sie knacken wollen um die sichere Kommunikation zwischen zwei Partnern zu gewährleisten.

Literatur

1. Tuzzio, N., Tehranipoor, M.: RSA: Implementation and security. In Tehranipoor, M., Wang, C., eds.: Introduction to Hardware Security and Trust. Springer (2012) 51–64
2. Tehranipoor, M., Wang, C.: Introduction to Hardware Security and Trust. Springer (2012)