Cryptography and its implementation

Simon Braitsch
Studiengang Softwaretechnik
Universität Stuttgart
Email: braitssn@studi.informatik.uni-stuttgart.de

***Abstract*—In einer Zeit, in der Kryptowährungen, Blockchain und Abhörskandale in aller Munde sind, wird das Feld der Kryptographie auch für Laien zunehmend bedeutend. Oft gibt es, gerade für solche Personen, wenige Quellen, die wichtige Elemente dieses Bereichs einfach, kompakt und verständlich erklären. Diese Arbeit soll deshalb auch dazu dienen, sowohl Softwareentwicklern als auch Laien zwei der wichtigsten Verfahren der Kryptographie näherzubringen, sowie diese bezüglich Sicherheit und Performanz zu vergleichen. Bei den vorgestellten Verfahren handelt es sich um RSA und AES. Anschließend werden die Verfahren insbesondere auf Anfälligkeiten für Side-Channel-Attacks untersucht. Es wird zu dem Schluss gelangt, dass die gängigen Verfahren und ihre zugrunde liegenden Algorithmen prinzipiell sehr sicher sind. Notwendige Sicherheitsmaßnamen zur effektiven Verhinderung von Side-Channel-Attacks werden von allen etablierten Kryptobibliotheken umgesetzt. Schwachstellen, die darüber hinausgehen, liegen im Verantwortungsbereich der CPU-Hersteller.**

I. Introduction

Seit im Juni 2013 die ersten, von Edward Snowden geleakten, Informationen zur umfassenden Abhörung durch US-Geheimdienste an die Öffentlichkeit gelangten, haben die Themen Sicherheit und Privatsphäre im Internet stark an Relevanz gewonnen [2]. Berichte über die erfolglosen Versuche des FBI, die Verschlüsselung eines iPhones zu knacken [3], oder die vermehrte Verbreitung von sogenannter Ransomware, schädlicher Software, die die Festplatte betroffener Nutzer verschlüsselt und Lösegeld fordert [4], trugen ihren Teil dazu bei. Endgültig in den Köpfen der Menschen angekommen ist die Kryptographie mit dem rasanten Aufstieg der Kryptowährungen, allen voran Bitcoin. Das Ver- und Entschlüsseln von Daten ist damit nicht mehr nur ein Thema für Softwareexperten. Oft wird das Interesse an der Thematik, gerade bei Laien auf dem Gebiet, allerdings gebremst. Übermäßig kompliziert formulierte und auf technischem Wissen basierende Artikel stellen eine große Einstiegshürde dar. Diese Arbeit versucht daher, die Themen in ihrer Komplexität auf ein Minimum zu reduzieren, ohne dabei jedoch an Korrektheit einzubüßen. Sie soll als Einstieg in das große Feld der Kryptographie dienen und eine erste Vorstellung vermitteln. Aus diesem Grund beschränkt sich die Anzahl der vorgestellten Verfahren auf zwei der wohl bekanntesten Algorithmen. Namentlich RSA und AES. RSA ist Lehrstoff an den meisten Universitäten, als Verfahren recht simpel und leicht zu verstehen, ohne Vorwissen im Bereich der Informatik zu besitzen. Daher wird es als erstes vorgestellt. Die Enscheidung für AES fiel auf Grund seiner Bedeutung. Denn AES, kurz für *Advanced Encryption Standard*, ist, wie der Name verrät, die Standard-Verschlüsselung, die bspw. auch die US-Regierung für ihre Dokumente verwendet [6].

A. Einführende Begriffserläuterungen

Bevor die einzelnen Verfahren näher betrachtet werden, sollten zunächst einige der Begrifflichkeiten der Kryptographie erklärt werden.

1) Schlüssel: Als Schlüssel bezeichnet man in der Kryptographie einen Text, numerisch oder alphanumerisch, der als Parameter des Verfahrens die Ausgabe bestimmt. Der Schlüssel wird dabei auf den Klartext, also unverschlüsselten Text, bzw. auf den verschlüsselten Text, bei einer Entschlüsselung, angewandt. Das einfachste Beispiel eines kryptographischen Schlüssels ist der Schlüssel bei einem Caesar-Chiffre [7]. Lautet der Schlüssel hier bspw. 7, dann bedeutet das, dass jeder Buchstabe im Eingabetext ersetzt wird durch den Buchstaben 7 Stellen danach im Alphabet. So wird aus dem Wort "Apfel" das Wort "Hwmls". Um daraus wieder den Klartext zu erhalten, wird ein neuer Schlüssel aus dem alten generiert, indem man den Ausgangsschlüssel von der Länge des verwendeten Alphabets abzieht. Aus 7 wird damit, bei einer Alphabetlänge von 26, 19. Bei Anwendung dieses Schlüssels auf "Hwmls" erhält man dann wieder das Wort "Apfel".

2) Symmetrische Verschlüsselung: Symmetrische Verschlüsselungsverfahren zeichnen sich durch die Eigenschaft aus, dass, sowohl für Ver- als auch Entschlüsselung, der selbe Schlüssel benutzt wird. Bei einem, auf diese Weise verschlüsselten, Austausch benötigen also beide Parteien den genutzten Schlüssel. AES und Twofish sind Beispiele für ein solches Verfahren.

3) Asymmetrische Verschlüsselung: Dem gegenüber stehen die asymmetrischen Verfahren. Hier kommen Schlüsselpaare, bestehend auf einem öffentlichen und einem privaten Schlüssel, zum Einsatz. Zur Verschlüsselung wird hier der öffentliche Schlüssel verwendet, zur Entschlüsselung der private. Solche Verfahren sind daher von Natur aus komplexer und deutlich langsamer als symmetrische Verschlüsselungen [8]. Ein Beispiel für eine asymmetrische Verschlüsselung ist RSA.

B. Benötigtes Wissen über Zahlensysteme

Um kryptographische Algorithmen verstehen zu können, kommt man um die Nutzung des Binär- und Hexadezimalsystems nicht umher. An dieser Stelle daher eine sehr kurze Einführung.

1) Binär und Hexadezimal: Jede Stelle in unserem dezimalen Zahlensystem steht für eine Potenz zur Basis 10 und kann maximal einen Wert von Basis - 1 annehmen, also 9 im Dezimalsystem. Eine dreistellige Zahl, bspw. 114, besteht aus $1 * 10^2 + 1 * 10^1 + 4 * 10^0$. In den Systemen Binär und Hexadezimal, ändert sich die Basis zu 2 bzw. 16. Aus 114 wird dann im Binärsystem 1110010, zusammengesetzt aus $1*2^6+1*2^5+1*2^4+1*2^1$. Im Hexadezimalsystem wird aus 114 die Zahl 72, zusammengesetzt aus $7*16^1+2*16^0$. Stellen mit einem Wert größer als 9 werden im Hexadezimalsystem durch Buchstaben dargestellt. Von A für 10 bis F für 15.

2) XOR: Während Schlüssel, der Übersicht wegen, meist in Hex-Werten angegeben werden, finden Rechenoperationen auf der Binärrepräsentation statt. Die wichtigste Operation für vorgestellte Verfahren wird XOR, ein exklusives Oder, sein. Dabei werden zwei Binärzahlen stellenweise miteinander verglichen. Das Ergebnis basiert auf den Werten beider Zahlen an den gleichen Stellen. Beispiel:

$$11010110 \oplus 10001010 = 01011100 \tag{1}$$

Das Ergebnis ist an einer Stelle genau dann 1, wenn exakt eine der beiden Ausgangszahlen an dieser Stelle eine 1 hatte. Ansonsten ist das Ergebnis 0.

II. RSA

RSA ist ein asymmetrisches Verschlüsselungsverfahren. Es kommen also ein privater Schlüssel für die Entschlüsselung und ein öffentlicher Schlüssel für die Verschlüsselung zum Einsatz. Es wurde 1977 entwickelt, nachdem ein Jahr zuvor die Forderung nach einer neuen Art der Verschlüsselung aufkam, um Probleme symmetrischer Verfahren (große Anzahl an Schlüsseln, Kenntnis des Verschlüsselungs-Schlüssels ermöglicht Entschlüsselung) zu beheben. Bei der Durchführung des Verfahrens spielt der Modulo-Operator eine größere Rolle. Hierbei handelt es sich um den Rest bei einer Teilung. So ist $5 \pmod 3 = 2$ und $5 \pmod 8 = 5$. RSA ist in seiner Funktionsweise vergleichsweise einfach zu verstehen und daher gut geeignet, um z.B. Studenten die Kryptographie näherzubringen. Die einzelnen Schritte des Verfahrens können weitestgehend in simple Multiplikationen und Divisionen zerteilt werden und sind daher wenig abstrakt. Aus diesem Grund wird RSA auch so oft an Universitäten gelehrt.

A. Ablauf

1) Schlüsselerzeugung: Zunächst gilt es zwei sehr große Primzahlen zu finden (Größenordnung 10^{200}). Diese beiden Zahlen, im Folgenden p und q, multipliziert man anschließend und erhält das Produkt n. Nun soll $\varphi(n)$ gebildet werden. Bei $\varphi(n)$ handelt es sich um die eulersche $\varphi - Funktion$, die die Anzahl der, zu n teilerfremden, Zahlen angibt, die kleiner als n sind. Berechnet wird sie mit:

$$\varphi(n) = (p-1) * (q-1) \tag{2}$$

Anschließend wird der öffentliche Schlüssel e so gewählt, dass er zu $\varphi(n)$ teilerfremd ist und die größten gemeinsamen Teiler $ggT(e-1, p-1)$ und $ggT(e-1, q-1)$ möglichst klein sind. Teilerfremdheit zu $\varphi(n)$ erreicht man entweder über eine Primeigenschaft von e mit $e > \varphi(n)$ oder, indem man ein e mit $e < \varphi(n)$ wählt, für das gilt, dass $ggT(e, \varphi(n)) = 1$. Der private Schlüssel d ergibt sich dann über die Formel:

$$(e * d) \pmod{\varphi(n)} = 1 \tag{3}$$

Der öffentliche Schlüssel e und das Produkt n der Zahlen p und q wird nun jedem zur Verfügung gestellt, da mit Hilfe dieser die Verschlüsselung stattfindet. Die Zahlen p und q nach Verwendung gelöscht [9].

2) Verschlüsselung: Die Verschlüsselung eines Textes t findet unter Benutzung des Zahlenpaars (n, e) aus dem vorherigen Schritt statt. Da das RSA-Verfahren aber, anders als bspw. das zuvor erwähnte Caesar-Chiffre, keinen Text, sondern nur Zahlen verschlüsseln kann, muss t zunächst in eine Zahlenfolge transformiert werden. Dazu wird in der Regel der ASCII-Code verwendet, der Buchstaben und Symbolen einen Zahlenwert zuweist. Um nun den umgewandelten Text t zu verschlüsseln, wird er mit dem öffentlichen Schlüssel potenziert und anschließend modulo n genommen. Also:

$$t' = t^e \pmod n \tag{4}$$

3) Entschlüsselung: Zur Entschlüsselung wird das Zahlenpaar (n, d) benötigt. Sie entspricht praktisch der Verschlüsselung, allerdings wird der zu entschlüsselnde Text nicht mit dem öffentlichen, sondern dem privaten Schlüssel d potenziert [10][11]. Also:

$$t = t'^d \pmod n \tag{5}$$

4) Anwendung am Beispiel: Der Einfachheit wegen werden in diesem Beispiel sehr kleine Zahlen verwendet. Bei richtigem RSA werden für gewöhnlich Schlüssellängen zwischen 2048 und 4096 Bit verwendet [12]. Sei $p = 5$ und $q = 11$. Es folgt $n = p * q = 55$. Dann ist $\varphi(n) = (5-1) * (11-1) = 40$. Als öffentlicher Schlüssel wird 3 gewählt. Da $ggT(3, 40) = 1$ ist die 3 als Schlüssel geeignet. Der private Schlüssel berechnet sich dann wie folgt:

$$(3 * d) \pmod{40} = 1 \tag{6}$$

$$3 * 27 = 81 \tag{7}$$

$$81 \pmod{40} = 1 \tag{8}$$

$$d = 27 \tag{9}$$

Unser zu verschlüsselnder Text t sei die Zahl 53. Es ist hierbei wichtig, dass $t < n$, sonst funktioniert das Verfahren nicht. Es folgt:

$$t' = 53^3 \pmod{55} = 47 \tag{10}$$

$$t = 5^{27} \pmod{55} = 53 \tag{11}$$

Ver- und Entschlüsselung waren demnach erfolgreich.

B. Sicherheit des Algorithmus

RSA ist gegenüber herkömmlichen Angriffsvektoren sehr sicher, solange die Parameter p und q geschickt gewählt werden. So werden zur Berechnung eines privaten Schlüssels bei einer Länge von 2048 Bit mit einem Brute-Force-Ansatz (Durchprobieren aller Möglichkeiten) bereits Zeiträume im Bereich von Milliarden von Jahren benötigt [13]. Ein Aufbrechen der Verschlüsselung auf diese Weise ist also unwahrscheinlich. Ebenfalls leidet RSA nicht unter dem Problem, dass es auf Grund wachsender Berechnungsleistungen schwächer wird. Sollte der Fall eintreten, dass bspw. durch Entwicklung funktionstüchtiger Quantencomputer, unsere Rechenleistung enorm ansteigt, wird einfach die Schlüssellänge der RSA-Implementierung erhöht. RSA ist demnach leicht skalierbar über die Wahl der Primfaktoren bzw. der Schlüssellänge unter Erhöhung der Berechnungszeiten.

C. Eigenschaften von RSA-Implementierungen

Während der Algorithmus an sich keine wirklichen Schwachstellen hat, gilt das nicht für die Wahl der Parameter, die für die Verschlüsselung genutzt werden. Bei Implementierungen von RSA wird daher vor allem darauf geachtet, dass p und q nicht nah beieinander liegen. Sonst ist es mittels der Faktorisierungsmethode von Fermat möglich, diese Primfaktoren relativ einfach zu berechnen [14][15]. Ebenfalls sollten der zu verschlüsselnde Text t und der öffentliche Schlüssel e ausreichend groß gewählt werden. Gilt $t^e < n$, dann sinkt die Schwierigkeit des Wurzelziehens für t', da die Modulo-Operation der Verschlüsselung hinfällig wird und Wurzelziehen nur modulo einer sehr großen Zahl wirklich schwer ist. Dem Angreifer wäre es also möglich aus t' die e-te Wurzel zu ziehen und damit t zu berechnen. Um zu Verhindern, dass t zu klein ist, wird deshalb in RSA-Implementierungen, die die PKCS (Public-Key Cryptography Standards) erfüllen, Padding angewandt. Dabei wird dem Text t eine Zeichenkette s angehängt, deren Struktur vorgegeben ist und unter mehreren möglichen zufällig gewählt wird. Das RSA-Verfahren wird dann also nicht auf t angewandt sondern auf ts. Somit wird der resultierende, verschlüsselte Text randomisiert und Angriffsvektoren, die die Kürze von t ausnutzen, ausgehebelt [16].

D. Performanz

Performanz ist die große Schwäche von RSA, verglichen mit weiteren Verschlüsselungen. Als asymmetrisches Verschlüsselungsverfahren gibt es erhöhte Anforderungen an die Schlüssellänge, da die Menge möglicher Werte, die für die Berechnung des privaten Schlüssels genutzt werden können, nur eine kleine Untermenge aller möglichen Kombinationen von Werten ist. Ein Schlüssel mit vergleichbarer Länge zu Schlüsseln symmetrischer Verfahren, z.B. AES, wäre deutlich unsicherer, da der Suchraum für Angreifer kleiner ist [17]. Aus diesem Grund wird RSA auch nicht zur Verschlüsselung größerer Datenmengen verwendet. Stattdessen kommt es häufig zum Einsatz, wenn Schlüssel symmetrischer Verfahren an einen Austauschpartner geschickt werden müssen. Diese Schlüssel werden dann mit RSA verschlüsselt. Ein Beispiel hierfür ist SSH [18].

III. AES

AES (Advanced Encryption Standard), oft auch Rijndael (nach den Entwicklern des Algorithmus) genannt, ist ein weit verbreitetes, rundenbasiertes, symmetrisches Verschlüsselungsverfahren. Es ist seit 2000 als Nachfolger von DES der Verschlüsselungsstandard. Es handelt sich um ein sogenanntes Blockchiffre mit einer Blocklänge von 128 Bit. Das heißt, der zu verschlüsselnde Text wird in Blöcke der Größe 128 Bit, mit 4*4 Einträgen der Größe 1 Byte, unterteilt. Da es ein rundenbasierter Algorithmus ist, wird der Ausgangsschlüssel solange erweitert, bis es für jede Runde einen 128 Bit Rundenschlüssel gibt. Diese Rundenschlüssel werden anschließend nacheinander, und in Kombination mit weiteren Operationen, auf die Blöcke angewandt. AES ist in 3 Varianten verfügbar, die sich in der Länge des Schlüssels und der Rundenanzahl unterscheiden. AES-128 mit 10 Runden, AES-192 mit 12 Runden und AES-256 mit 14 Runden [19].

A. Ablauf

Bevor die eigentliche Verschlüsselung beginnen kann, muss zunächst der initial zufällig bestimmte Schlüssel expandiert werden. Diesen Vorgang nennt man Schlüsselexpansion. Darauf folgt eine Vorrunde, gefolgt von den Rundenoperationen byteSub, shiftRow, mixColumn und keyAddition. Abschließend gibt es eine Schlussrunde, nach deren Durchführung der verschlüsselte Text ausgegeben wird.

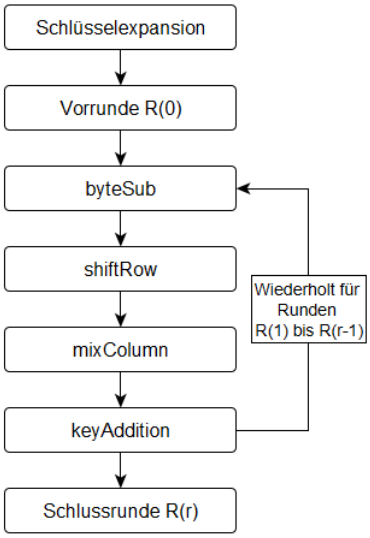

Fig. 1: Ablauf einer AES-Verschlüsselung

1) Schlüsselexpansion: Ziel ist es, den Schlüssel auf eine Länge von $B * (R + 1)$ zu erweitern. Dabei ist B die Blocklänge von 128 Bit und R die Anzahl der Runden, basierend auf der Wahl der AES-Variante. Nur so kann der Schlüssel anschließend in 128 Bit lange Abschnitte unterteilt werden, die als Rundenschlüssel fungieren. Hierzu wird jeder

Rundenschlüssel rekursiv aus dem vorherigen berechnet. Das heißt, der erste Rundenschlüssel wird aus dem Hauptschlüssel berechnet. Dabei kommen eine S-Box (Substitutions-Box) und eine Rundenkonstante zum Einsatz. Die Rundenkonstante $rcon$ wird aus einer vorberechneten rcon-Tabelle entnommen und ändert sich für jede Runde. Damit wird jede neue Rundenschlüsselberechnung leicht verändert und erhöht die Sicherheit des Verfahrens [20]. Eine S-Box ist ebenfalls eine Tabelle, in der für jedes Eingabe-Byte ein unterschiedliches Ausgabe-Byte definiert ist. Das Byte wird also durch ein anderes substituiert. Um nun aus dem vorherigen Schlüssel den nächsten Rundenschlüssel zu berechnen, wird wie folgt vorgegangen: Das erste Byte einer Zeile des Schlüssels wird $\oplus$-verknüpft mit dem, durch die S-Box substituierten, Byte der letzten Spalte, drei Zeilen zuvor. Das Ergebnis dieser Operation wird wiederum $\oplus$-verknüpft mit der aktuellen Rundenkonstante. Im Fall des Hauptschlüssels also $rcon(0)$. Alle folgenden Bytes in dieser Zeile werden berechnet, indem das Byte des vorhandenen Schlüssels $\oplus$-verknüpft wird mit dem zuletzt berechneten Byte des nächsten Rundenschlüssels[21]. Dieser Vorgang wird wiederholt, bis der Hauptschlüssel S die Länge $B * (R + 1)$ erreicht hat.

2) Vorrunde: Die Vorrunde ist lediglich eine $\oplus$-Verknüpfung von Hauptschlüssel und dem zu verschlüsselnden Textblock. Dies entspricht der keyAddition-Operation.

3) byteSub: In diesem Schritt wird jedes Byte des Eingabeblocks durch die S-Box substituiert.

4) shiftRow: Die Zeilen 2 - 4 des Blocks werden jeweils um $(Zeilennummer - 1)$ Stellen nach links verschoben.

5) mixColumn: Hier werden die Spalten des Blocks jeweils mit der folgenden Matrix multipliziert.

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \tag{12}$$

6) keyAddition: Der Block wird $\oplus$-verknüpft mit dem aktuellen Rundenschlüssel.

7) Schlussrunde: Die Schlussrunde entspricht einem weiteren Durchlauf der Rundenoperationen; der Schritt mixColumn wird jedoch nicht ausgeführt.

8) Anwendung am Beispiel: Da für die komplette Anwendung von AES viele größere Parameter, wie S-Box und Rundenschlüssel, benötigt werden, beschränkt sich das folgende Beispiel auf die Berechnung neuer Bytes eines Rundenschlüssels. Dazu wird der Inhalt von Table 1 als Hauptschlüssel verwendet.
Um nun das erste neue Byte des nächsten Rundenschlüssels zu berechnen, werden die Bytes in Zeile 1, Spalte 1 und

TABLE I: Beispiel eines Hauptschlüssels, Zellenwerte in Hexadezimal

A9	B5	FF	4
8E	8F	5A	B8
3D	6F	A5	33
9E	17	9C	2E

in Zeile 2, Spalte 4 benötigt. Byte A ist also A9 und Byte B ist B8. Byte B muss jetzt durch die S-Box substituiert werden. Das Ergebnis der Substitution sei an dieser Stelle 6C. Anschließend werden A9 und 6C $\oplus$-verknüpft. Dafür werden beide Bytes in ihre Binärrepräsentation umgewandelt. Aus A9 wird so 10101001, aus 6C wird 01101100.

$$10101001 \oplus 01101100 = 11000101 \tag{13}$$

Anschließend wird das Ergebnis dieser Berechnung mit der Rundenkonstante $\oplus$-verknüpft. Da dies die Berechnung des ersten Rundenschlüssels ist, setzen wir $rcon(0) = 1$ bzw. in Binär 00000001.

$$11000101 \oplus 00000001 = 11000100 \tag{14}$$

Dieses Ergebnis entspricht dem Wert C4 in Hexadezimal und kann an die erste Stelle des Rundenschlüssels (Table 2) eingetragen werden.

TABLE II: Rekursiv aus Table 1 berechneter Rundenschlüssel

C4	71	8E	8A

Um die weiteren Stellen des Rundenschlüssels in dieser Zeile zu berechnen, werden jetzt die zuvor berechneten Werte genutzt. Den Wert für die zweite Spalte in Zeile 1 von Table 2 erhält man durch eine $\oplus$-Verknüpfung von B5 aus Table 1, Zeile 1, Spalte 2 und dem eben berechneten Wert C4 aus Table 2:

$$10110101 \oplus 11000100 = 01110001 \tag{15}$$

Das entspricht in Hexadezimal 71. Auf diese Weise lässt sich die komplette Schlüsselexpansion durchführen. Lediglich bei der Variante AES-256 wird zusätzlich jedes vierte Byte noch durch die S-Box substituiert.

B. Sicherheit des Algorithmus

AES ist, wie es sein Status als Standard vermuten lässt, ein sehr sicherer Algorithmus. Zwar gibt es Angriffsvektoren, die, verglichen mit der Berechnungsdauer eines simplen Brute-Force-Ansatzes, um bis zu ein Vierfaches schneller sind, jedoch sind auch diese nicht praktikabel. So benötigt der sogenannte Biclique-Angriff, der 2011 vorgestellt wurde, zwischen 2^{126} Schritten bei AES-128 und 2^{254} Schritten bei AES-256 [22]. Auf einem Rechner, der pro Sekunde 1.000.000.000 mögliche Kombinationen überprüfen kann, würde dies bereits eine

unvorstellbare Anzahl an Jahren benötigen. Und das bei AES-128. Es lässt sich also behaupten, dass das Verschlüsselungsverfahren von AES sicher ist.

C. Eigenschaften von AES-Implementierungen

Zunächst muss bei einer Implementierung von AES, ähnlich wie bei RSA, die Länge des zu verschlüsselnden Textes beachtet werden. Dies liegt an der strikten Blocklänge von AES. Sollte der Eingabetext diesen Ansprüchen nicht genügen, ist seine Länge also kein Vielfaches von 128, muss Padding stattfinden. Dabei wird der Text solange mit Null-Bytes aufgefüllt, bis er die notwendige Länge erreicht und verschlüsselt werden kann [23]. Es existieren zudem verschiedene Modi, zwischen welchen man bei der Durchführung von AES wählen kann, die zum Teil gravierende Unterschiede bezüglich Sicherheit aufweisen. Diese Modi sind nicht einzigartig für AES selbst und finden bei allen Blockchiffres Verwendung. So wird im ECB-Modus (Electronic Code Book) jeder Block separat und unabhängig verschlüsselt. Dies hat zur Folge, dass identische Klartext-Blöcke auch in ihrer verschlüsselten Form identisch sind. So können Muster und Strukturen des Klartextes in der Verschlüsselung sichtbar werden und die Integrität dieser beeinträchtigen. In anderen Modi, wie z.B. dem CBC-Modus (Cipher Block Chaining), existiert dieses Problem nicht, da die Verschlüsselung eines Blocks von allen zuvor verschlüsselten Blöcken abhängig ist [24].

D. Performanz

AES ist eines der schnellsten, wenn nicht sogar das schnellste, Verschlüsselungsverfahren. Dies liegt jedoch nicht nur an dem Algorithmus selbst, der zusammen mit Twofish der performanteste Kandidat bei der Auswahl der AES-Finalisten war [25]. Die teils enormen Unterschiede bei der Berechnungsdauer von AES, verglichen mit anderen Verfahren, resultieren vor allem aus den AES-NI (Intel Advanced Encryption Standard New Instructions). Dabei handelt es sich um eine Gruppe von Hardware-Instruktionen, die von CPU-Herstellern in neueren Prozessoren in der Regel implementiert werden. Diese Befehle, z.b. AESENC und AESDEC, erlauben die Ausführung von AES-Abläufen. Dadurch, dass diese direkt auf der Hardware implementiert sind, profitiert man von starken Leistungsverbesserungen. So erreichte man bei einem Direktvergleich zweier Prozessoren (Intel Core i5-661 und Intel Core i7-870), von denen nur der i5 die AES-NI implementierte, die bis zu dreifache Geschwindigkeit bei einem AES-Verschlüsselungsbenchmark [27]. Es ist also nicht überraschend, dass die Performanz von AES andere Verfahren in den Schatten stellt, zumindest bei ähnlichem Sicherheitslevel der Verschlüsselung. Besonders deutlich wird dies in Benchmarks der Verschlüsselungs-Software TrueCrypt. Dort wurden AES (mit AES-NI Implementierung), Twofish, Serpent und Kombinationen der drei Verfahren verglichen. Mit deutlichem Abstand schneidet AES mit einer Durchsatzrate von 1.6 GB/s am besten ab, gefolgt von Twofish mit 267 MB/s und AES-Twofish mit 230 MB/s. Serpent, eine besonders robuste Verschlüsselung, erreichte nur 142 MB/s [28].

IV. Anfälligkeiten für Side-Channel-Attacks

In diesem Abschnitt soll die generelle Anfälligkeit der vorgestellten Verfahren für Side-Channel-Attacks untersucht werden und wie dagegen vorgegangen werden kann. Zu diesem Zwecke folgt zunächst eine kurze Einführung in das Thema.

A. Side-Channel-Attacks

Side-Channel-Attacks, zu deutsch Seitenkanal-Angriffe, stellen eine Sonderform von Angriffsvektoren aus dem Bereich der Kryptoanalyse dar. Während herkömmliche Angriffe versuchen, Schwächen in den implementierten Algorithmen und der dahinterstehenden Logik zu finden und auszunutzen, zielen Side-Channel-Attacks auf Schwächen in der konkreten Implementierung der Verfahren ab. In diesem Kontext sind Seitenkanäle Informationsströme des beobachteten Gerätes bei der Ausführung eines kryptographischen Verfahrens. Für einen solchen Angriff möglicherweise relevante Informationen sind bspw. Rechenzeit, Speichernutzung, Stromverbrauch und sogar elektromagnetische Abstrahlung oder Schall [29]. Bei der sogenannten Fault Attack werden sogar gezielt Fehler in den Prozess geschleust und anschließend ausgenutzt. Es gibt zwei Kategorien solcher Fehler. Zum einen Berechnungsfehler, bspw. ausgelöst durch gezielte Spannungserhöhungen. Zum anderen Fehlerhafte Eingabedaten, die ein spezielles Verhalten im Programm auslöst [30].

B. RSA und Side-Channel-Attacks

Bei der Durchführung einer RSA Ver- bzw. Entschlüsselung gibt es vor allem zwei Schritte, die anfällig für Side-Channel-Attacks sind. Zum einen die Generierung der Primfaktoren p und q, zum anderen die modulare Exponentiation t' = $t^e \pmod n$. Bei der Generierung der Primfaktoren ist es Angreifern theoretisch möglich, über Messungen des Stromverbrauchs während des Prozesses Rückschlüsse über die gewählten bzw. gefundenen Faktoren zu ziehen. Dies ist möglich, wenn beim Findungsprozess der Primfaktoren deterministische Verfahren genutzt werden. Dabei werden, ausgehend von einer zufällig ausgewählten Ausgangszahl, in Inkrementen Primzahltests durchgeführt. Aus Performanzgründen sind dies in der Regel absolut korrekten Tests. Stattdessen wird ein probabilistisches Verfahren angewandt, um zu erkennen, ob eine Zahl aller Wahrscheinlichkeit nach prim ist. Bei diesen Tests wird, bis zu einer festgelegten Grenze, die Teilbarkeit der Testzahl mit kleineren Faktoren überprüft. Sobald eine Teilbarkeit festgestellt wird, bricht der Primtest ab. Je nach getesteter Zahl ist damit der Energieverbrauch eines Tests sehr variabel, abhängig davon wie schnell ein Teiler gefunden wird [31]. Eine einfache Methode, um solche Angriffe abzuwehren ist es, den Test immer komplett durchlaufen zu lassen. Legt man also fest, dass jede Zahl mit allen ungeraden Zahlen bis 100 auf Teilbarkeit getestet werden soll, und findet bereits nach 3 Tests einen Teiler, dann führt man trotzdem alle restlichen Tests durch. Der Energieverbrauch sollte damit für alle Primzahltests in etwa identisch sein. Andere Varianten derartige Angriffe zu entschärfen, wären bspw. das zufällige Hinzufügen weiterer

Teiler-Tests oder die Auswahl des nächsten Teiler-Tests zu randomisieren [32]. Im Gegensatz zur ersten Methode wird dabei also nicht der Energieverbrauch konstant gehalten, sondern randomisiert, sodass auf dessen Basis keine Rückschlüsse auf die Primfaktoren gezogen werden kann.
Anders als bei der Generierung der Primfaktoren ist beim Schritt der modularen Exponentiation nicht der Stromverbrauch der Angriffsvektor, sondern die Berechnungszeit. Indem der Angreifer eigens gewählte Texte zur Verschlüsselung an die RSA-Implementierung gibt, ist es ihm möglich, über studieren der benötigten Rechenzeiten, den privaten Schlüssel, der genutzt wird, zu errechnen. Dies ist möglich, da die Dauer der Exponentiation mit Länge und Inhalt des zu verschlüsselnden Textes korreliert [33]. Solche Angriffe, bei denen der Angreifer ihm bekannte Texte nutzen kann, werden Chosen-Plaintext-Attacks genannt [34].
Eine Methode, um derartige Angriffe zu verhindern, wurde bereits vorgestellt, da sie auch für die generelle Sicherheit einer RSA-Implementierung wichtig ist. Indem man den übermittelten Text mit Padding transformiert, wird die Abhängigkeit von Text und Berechnungszeit stark verwischt. Dem Angreifer ist es dann nicht mehr möglich mit Sicherheit zu sagen, wie sein Eingabetext die Berechnungsdauer beeinflusst hat. Eine Padding-Technik, die dafür verwendet werden kann ist bspw. OAEP (Optimal Asymmetric Encryption Padding). Diese bietet im Vergleich mit dem Padding des Public Key Cryptography Standards (PKCS) noch mehr Sicherheiten [35]. Der Vorgang, durch den die Eingabe des Angreifers (oder auch des normalen Nutzers) in eine Form gebracht wird, die er selbst nicht mehr kennt, ist eine Variante des sogenannten Blindings [36].

C. AES und Side-Channel-Attacks

Klassisches AES besitzt durchaus eine grundsätzliche Anfälligkeit für Side-Channel-Attacks. Dies liegt vor allem an der typischen Implementierung des Verfahrens. Ein größerer Teil des Algorithmus sieht Nachschlagen von Substitutionen in der S-Box-Tabelle vor. Damit die Performanz des Verfahrens deutlich verbessert werden kann, bietet es sich an die S-Box als Lookup-Table zu implementieren, in dem die zu substituierenden Werte nachgeschlagen werden. Dadurch werden häufig genutzte Werte/Buchstaben in den Cache gezwungen und sind dann deutlich schneller abzurufen, als sie das bei dynamischer Berechnung bei Bedarf wären. Dies bedeutet eine substanzielle Performanzverbesserung.
Problematisch wird das allerdings im Hinblick auf Timing-Angriffe; Side-Channel-Attacks, die benötigte Berechnungszeiten ausnutzen, um Rückschlüsse auf Parameter der Verschlüsselung zu ziehen. Lookup-Tables sind für solche Angriffe besonders anfällig. Nachschlagezeiten für Einträge in der S-Box korrelieren mit dem verwendeten Index, die Dauer der AES-Berechnung korreliert wiederum mit der Dauer der Abrufe von Werten der S-Box. Indem man nun das Verhalten und die Antwortzeiten einer solchen AES-Implementierung studiert, ist es möglich den kompletten verwendeten Schlüssel über Korrelationen abzuleiten, wie Daniel Bernstein zeigen konnte [37]. Moderne Implementierungen von AES in populären Kryptobibliotheken enthalten allerdings bereits schnelle Software-Lösungen, um Timing-Attacks und weitere Side-Channel-Attacks erfolgreich zu unterbinden. Zusätzlich besitzen auch die AES-NI integrierten Schutz gegen Side-Channel-Attacks. So werden bei Verwendung der AES-NI Befehle keine Lookup Tables benötigt und Latenzen, Ver-/Entschlüsselung, Schlüsselexpansion und Speicherzugriffe sind unabhängig von den Eingabedaten. Es ist somit nicht möglich Korrelationen zwischen Eingabedaten und Ausgabedaten aufzustellen [38]. Da praktisch alle modernen CPUs die AES-NI implementieren, welche von den verfügbaren Krypto-Softwarebibliotheken genutzt werden, ist die Integrität von AES auch im Hinblick auf Side-Channel-Attacks ausreichend gesichert. Angriffe, wie sie unter anderem von Osvik et al. 2005 [39] vorgestellt wurden, stellen damit keine relevante Gefahr dar, da die eigens vorgeschlagenen Gegenmaßnahmen vollständig von AES-NI bereitgestellt werden.

V. Meltdown und Spectre

Die Sicherheit von kryptographischen Verfahren muss immer auch im Kontext des gesamten Systems betrachtet werden, auf welchem sie implementiert und angewandt werden. Dies ist bspw. analog zur Sicherheit der Verschlüsselungs-Modi von Blockchiffres, die zuvor erwähnt wurden. Der Cipher-Block-Chaining Modus (CBC) ist prinzipiell sicher. Er ist aber nur so sicher wie der zugrunde liegende Block-Chiffre. So ist CBC auf Basis von AES sicher, aber nicht auf Basis von bspw. einer Vernam Verschlüsselung [41]. Deutlich wurde das um die Jahreswende 2017/2018 herum, als die Existenz fataler Sicherheitslücken in den CPU-Architekturen von Intel und, in geringerem Ausmaße, AMD öffentlich wurden [5]. Die veröffentlichten Lücken bekamen die Namen Meltdown und Spectre und zeigen, wie auf erschreckend einfache Art und Weise die komplette Integrität des Systems zerlegt werden kann bzw. konnte, da die meisten der initial bekannten Lücken inzwischen gefixt wurde. Ob behoben oder nicht, beweist die bloße Existenz solcher Lücken allerdings, dass sichere Verfahren alleine nicht ausreichen, um Daten zu schützen.

A. Angegriffene CPU-Eigenschaften

Sowohl Meltdown als auch Spectre nutzen spezielle CPU-Eigenschaften aus, um Speicher auszulesen auf den die ausführenden Prozesse keinen Zugriff haben sollten. Besonders wichtig ist hierbei die sogenannte spekulative Ausführung. Sie ermöglicht es Befehle vorzeitig auszuführen, bspw. während für einen vorherigen Befehl noch auf eine Speicherauslesung gewartet wird, und somit Rechenzeit sinnvoll zu füllen. Diese Funktion ist also zur Leistungs- und Durchsatzsteigerung des Prozessors vorhanden und sorgt teilweise für dramatische Geschwindigkeits-Verbesserungen. Es ist natürlich möglich, dass ein solcher vorzeitig ausgeführter Befehl nicht erlaubt ist. Bspw. wenn dieser Befehl auf geschützten Speicher zugreifen will. In diesen Fällen wird die CPU eine Exception werfen und der Befehl wird abgebrochen und rückgängig gemacht.[42]

Eine weitere Eigenschaft, die ausgenutzt wird, ist das Mapping des gesamten Kernel-Speicherraums in die Nutzer-Prozesse [43]. Dies ist in der Regel kein Problem, da Speicheradressen Zugriffs-Einstellungen haben, wird aber bei Meltdown und Spectre zum Verhängnis.

B. Wie Meltdown funktioniert(e)

Zur Erklärung, wie die spekulative Ausführung ausgenutzt werden kann, um Speicher auszulesen, wird die Funktionsweise von Meltdown vorgestellt. Zwar wurde Meltdown relativ schnell gepatcht, aber der Ablauf ist wesentlich einfacher und besser zu erklären.

TABLE III: Verwendete Register

(a) Speicher-Register des angreifenden Prozesses

0	0	0	0	0

(b) Geschütztes Speicher-Register

3	1	2	4	2

1) Schritt 1: Zugriff auf verbotenen Speicher: Zunächst einmal ist es wichtig, dass kein Element unseres Speicher-Registers (Table III a) im Cache liegt. Dazu führt man einen "Flush" durch. Dies ist der erste Schritt zur Durchführung eines "Flush & Reload" (s. Schritt 3). Dabei handelt es sich um eine Cache-Timing-Attacke [44]. Wenn der Cache geleert ist, kann mit dem eigentlichen Ablauf von Meltdown angefangen werden. Dazu wird zunächst versucht von einer Speicher-Adresse auszulesen, auf die wir keinen Zugriff haben. Inbesondere interessant ist hierbei Kernel-Speicher, dessen Adress-Raum in unseren angreifenden Prozess gemapt ist. Nehmen wir also an wir versuchen den Wert des ersten Elements des geschützten Registers zu lesen (Table III b). Anschließend schreiben wir den gelesenen Wert in unser eigenes Register an dem Index, der dem gelesenen Wert entspricht. Wir würden hier also 3 lesen und an Index 3 in unser Register schreiben. Unser Register sieht dann kurzweilig so aus:

TABLE IV: Register nach Schritt 1

0	0	0	3	0

2) Schritt 2: Wert aus eigenem Register Lesen: Nun versuchen wir den eben geschriebenen Wert, also 3, wieder aus unserem Register zu lesen. Dazu greifen wir also auf unser Register an Index 3 zu. Hierdurch sollte diese Speicheradresse in den Cache gezwungen werden, um für mögliche zukünftige Zugriffe schnellere Geschwindigkeiten zu erreichen. Hierbei kommt es allerdings zu einer Wettlaufsituation. Diese startet, sobald wir in Schritt 1 auf den verbotenen Speicher zugreifen. Die CPU wird dabei irgendwann merken, dass die nötigen Berechtigungen fehlen und eine Exception werfen. Es ist jedoch möglich, dass durch die spekulative Ausführung der CPU bereits nachfolgende Befehle ausgeführt wurden. Konkret möchten wir, dass das Schreiben des gelesenen Werts und das anschließende Lesen aus unserem Register ausgeführt wurden, bevor die Exception ankommt. Ist dies nicht der Fall, dann muss der Vorgang wieder von Schritt 1 begonnen werden. Ging die Wettlaufsituation jedoch zu unseren Gunsten aus, können wir zu Schritt 3.

3) Schritt 3: Flush & Reload: Die CPU hat jetzt erkannt, dass wir auf Speicher zugreifen wollten, den wir nicht lesen dürfen. Sie wird daher alle Befehle rückgängig machen, die unberechtigt ausgeführt wurden. Das heißt unser Register sieht jetzt wieder aus wie vor Beginn des Angriffs. Was die CPU jedoch nicht zurücksetzt ist die Cache-Line zu unserem Register-Element, in das wir zuvor den Wert gespeichert hatten, auf den wir nicht zugreifen durften. Wir starten nun den zweiten Teil von "Flush & Reload", also das Reloaden. Dafür führen wir Lesebefehle auf allen Elementen unseres Registers durch und messen die benötigte Zeit. Bei einem erfolgreichen Angriff wird an einer Stelle eine deutlich geringere Zeit zum Lesen benötigt werden. Dies ist das Element, welches wir in den Cache zwingen konnten. Wenn wir jetzt also feststellen, dass wir an Index 3 eine deutlich geringere Zugriffszeit benötigen, dann wissen wir, dass an der angegriffenen Speicheradresse der Wert 3 gespeichert war. Auf diese Art und Weise lässt sich schlussendlich der komplette Kernel-Speicher auslesen. [45] Vor einem solchen Angriff ist dann logischerweise auch kein privater RSA-Schlüssel oder AES-Schlüssel sicher, der im RAM liegt.

VI. Conclusion

RSA und AES sind die vermutlich bekanntesten Verschlüsselungsverfahren die es gibt. Dementsprechend ist es keine Überraschung, dass beide in ihren zugrunde liegenden Algorithmen keine wirklichen Schwächen aufweisen, die ihre Sicherheit gefährden würde. Wäre dies der Fall, dann wären sie sicherlich nicht so populär geworden. Eine grundsätzliche Anfälligkeit für Side-Channel-Attacks verschiedener Arten lässt sich aber auch bei RSA und AES nicht absprechen. Eine Fülle an kryptographischen Bibliotheken, mit sorgfältig gewählten Implementierungen der einzelnen Verfahren, sorgt jedoch auch hier für ausreichend Sicherheit. Zahlreiche der verfügbaren Bibliotheken sind FIPS 140-2 validiert [40] und bieten somit gegen gängige Side-Channel-Attacks Absicherungen in Form der zum Teil zuvor erwähnten Gegenmaßnahmen wie Padding und Randomisierungen. Leider kann auch die beste Implementierung nicht alle ausnutzbaren Schwachstellen beheben. Sicherheitslücken wie Meltdown und Spectre lassen sich nicht über Softwaremaßnahmen seitens der Entwickler von kryptographischen Bibliotheken mitigieren. Hier sind die Hersteller der CPUs gefragt, denn jede Verschlüsselung ist nur so sicher, wie das System auf dem sie angewandt wird.

REFERENCES

[1] Paul Szoldra, 2016, *This is everything Edward Snowden revealed in one year of unprecedented top-secret leaks*, (verfügbar auf https://www.businessinsider.de/snowden-leaks-timeline-2016-9, Zugriff: 09.06.2018)

[2] Sam Bocetta, 2017, *Millions of People Are Taking an Interest in Cryptography*, (verfügbar auf https://tutanota.com/blog/posts/interest-cryptography, Zugriff: 09.06.2018)

[3] Chris Smith, 2016, *The FBI can't hack the iPhone 6s*, (verfügbar auf http://bgr.com/2016/04/07/iphone-6s-fbi-hack/, Zugriff: 09.06.2018)

[4] Kaspersky, *The Rise of Ransomware*, (verfügbar auf https://usa.kaspersky.com/resource-center/threats/ransomware-threats-an-in-depth-guide, Zugriff: 09.06.2018)

[5] Chris Davies, 2018, *Google reveals CPU security flaw Meltdown and Spectre details*, (verfügbar auf https://www.slashgear.com/google-reveals-cpu-security-flaw-meltdown-and-spectre-details-03513512/, Zugriff: 09.06.2018)

[6] East-Tec, *Use the U.S. Government approved algorithm for storing classified information*, (verfügbar auf https://www.east-tec.com/kb/what-is-the-aes-standard/, Zugriff: 09.06.2018)

[7] Kryptowissen, 2013, *Caesar Verschlüsselung / Caesar Chiffre*, (verfügbar auf https://www.kryptowissen.de/caesar-chiffre.html, Zugriff: 09.06.2018)

[8] Alan Hughes, *The Disadvantages of Asymmetric Encryption*, (verfügbar auf https://www.techwalla.com/articles/the-disadvantages-of-asymmetric-encryption, Zugriff: 09.06.2018)

[9] M. Busse, M. Schmitt, J. Steeg, 1999 *Der RSA-Algorithmus*, (verfügbar auf https://www.zum.de/Faecher/Inf/RP/infschul/kr_rsa.html, Zugriff: 11.06.2018)

[10] Hartmut Härtl, 2006, *Asymmetrische Verschlüsselung über so genannte „Public-Key-Verfahren“*, (verfügbar auf http://www.oszhdl.be.schule.de/gymnasium/faecher/informatik/krypto/rsa.htm, Zugriff: 11.06.2018)

[11] Uni Tuebingen, 2006, *Beispiel zur RSA-Verschlüsselung* , (verfügbar auf http://dm.inf.uni-tuebingen.de/lehre/kryptoVL/ws0607/Beispiel-RSA, Zugriff: 11.06.2018)

[12] Jürgen Schmidt, 2017 *Sichere Schlüssellänge bei GPG*, (verfügbar auf https://www.heise.de/ct/hotline/Sichere-Schluessellaenge-bei-GPG-3609103.html, Zugriff: 11.06.2018)

[13] Digicert, *Check our Numbers*, (verfügbar auf https://www.digicert.com/TimeTravel/math.htm, Zugriff: 11.06.2018)

[14] Mark Dickinson, 2012, *RSA - bitlength of p and q*, (verfügbar auf https://stackoverflow.com/questions/12192116/rsa-bitlength-of-p-and-q, Zugriff: 11.06.2018)

[15] Wikipedia, *Faktorisierungsmethode von Fermat*, (verfügbar auf https://de.wikipedia.org/wiki/Faktorisierungsmethode_von_Fermat, Zugriff: 12.06.2018)

[16] Wikipedia, *RSA-Kryptosystem*, (verfügbar auf https://de.wikipedia.org/wiki/RSA-Kryptosystem#Padding, Zugriff: 12.06.2018)

[17] StackExchange, *Why does the recommended key size between symmetric and asymmetric encryption differ greatly?*, (verfügbar auf https://crypto.stackexchange.com/questions/6236/why-does-the-recommended-key-size-between-symmetric-and-asymmetric-encryption-di, Zugriff: 12.06.2018)

[18] Linode, 2018, *Use Public Key Authentication with SSH*, (verfügbar auf https://www.linode.com/docs/security/authentication/use-public-key-authentication-with-ssh/, Zugriff: 12.06.2018)

[19] Wikipedia, *Advanced Encryption Standard*, (verfügbar auf https://de.wikipedia.org/wiki/Advanced_Encryption_Standard, Zugriff: 14.06.2018)

[20] Stackexchange, 2017, *What is the importance of Rcon in Rjindael's key expansion from a security prespective?*, (verfügbar auf https://crypto.stackexchange.com/questions/42784/what-is-the-importance-of-rcon-in-rjindaels-key-expansion-from-a-security-presp/42798, Zugriff: 14.06.2018)

[21] Heinrich Klocke, *Advanced Encryption Standard (AES)Rijndael – Algorithmus*, (verfügbar auf http://www.gm.fh-koeln.de/ hk/lehre/ala/ws0506/Praktikum/Projekt/C_gelb/Dokumentation_AES.pdf, Zugriff: 14.06.2018)

[22] Daniel Bachfeld, 2011, *Erster Kratzer für Kryptoalgorithmus AES*, (verfügbar auf https://www.heise.de/security/meldung/Erster-Kratzer-fuer-Kryptoalgorithmus-AES-1324532.html, Zugriff: 14.06.2018)

[23] Wikipedia, *AES Implementations*, (verfügbar auf https://en.wikipedia.org/wiki/AES_implementations, Zugriff: 14.06.2018)

[24] Datalocker, *ECB versus CBC Mode AES encryption*, (verfügbar auf https://datalocker.com/what-is-the-difference-between-ecb-mode-versus-cbc-mode-aes-encryption/, Zugriff: 14.06.2018)

[25] Bruce Schneier, 2000, *A Performance Comparison of the Five AES Finalists*, (verfügbar auf https://www.schneier.com/academic/paperfiles/paper-aes-comparison.pdf, Zugriff: 17.06.2018)

[26] Intel, *Intel® Data-Protection-Technik mit AES-NI und Secure Key*, (verfügbar auf https://www.intel.de/content/www/de/de/architecture-and-technology/advanced-encryption-standard–aes-/data-protection-aes-general-technology.html, Zugriff: 17.06.2018)

[27] P. Schmid, A. Roos, 2010, *AES-NI Performance Analyzed*, (verfügbar auf https://www.tomshardware.com/reviews/clarkdale-aes-ni-encryption,2538-5.html, Zugriff: 17.06.2018)

[28] robo47, 2010, *AES-NI Performance Analyzed*, (verfügbar auf http://www.robo47.net/blog/200-Truecrypt-7.0-Linux-AES-NI-Benchmark-with-i7-620M-on-Dell-Latitude-E6510, Zugriff: 17.06.2018)

[29] Wikipedia, *Seitenkanalattacke*, (verfügbar auf https://de.wikipedia.org/wiki/Seitenkanalattacke, Zugriff: 20.06.2018)

[30] YongBin Zhou, DengGuo Feng, *Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing*, (verfügbar auf https://csrc.nist.gov/csrc/media/events/physical-security-testing-workshop/documents/papers/physecpaper19.pdf, Zugriff: 20.06.2018)

[31] Thomas Finke, Max Gebhardt, Werner Schindler, 2009, *A New Side-Channel Attack on RSA Prime Generation*, Cryptographic Hardware and Embedded Systems – CHES 2009, volume 5747 of Lecture Notes in Computer Science, pages 141–155.

[32] Aurélie Bauer, Eliane Jaulmes, Victor Lomné, Emmanuel Prouff, Thomas Roche, 2014, /emphSide-Channel Attack Against RSA Key Generation Algorithms, (verfügbar auf https://www.ssi.gouv.fr/uploads/IMG/pdf/CHES2014_Side_Channel_Attack_against_RSA_Key_Generation_Algorithms.pdf, Zugriff: 13.07.2018)

[33] Amuthan Arjunan, Praveena Narayanan, and Kaviarasan Ramu, 2013, *Securing RSA Algorithm against Timing Attack*, (verfügbar auf http://www.ccis2k.org/iajit/PDF/vol.13,%20no.4/7478.pdf , Zugriff: 13.07.2018)

[34] Wikipedia, *Chosen-plaintext Attack*, (verfügbar auf https://simple.wikipedia.org/wiki/Chosen-plaintext_attack, Zugriff: 13.07.2018)

[35] Crypto Stackexchange, 2017, *RSA-OAEP vs RSA-PKCS*, (verfügbar auf https://crypto.stackexchange.com/questions/47436/how-much-safer-is-rsa-oaep-compared-to-rsa-with-pkcs1-v1-5-padding, Zugriff: 13.07.2018)

[36] Wikipedia, *Blinding*, (verfügbar auf https://en.wikipedia.org/wiki/Blinding_(cryptography), Zugriff: 13.07.2018)

[37] Daniel Bernstein, 2005, *Cache-timing attacks on AES*, (verfügbar auf https://cr.yp.to/antiforgery/cachetiming-20050414.pdf, Zugriff: 20.06.2018)

[38] Crypto Stackexchange, 2017, *Does AES-NI offer better side-channel protection compared to AES in software?*, (verfügbar auf https://crypto.stackexchange.com/questions/43563/does-aes-ni-offer-better-side-channel-protection-compared-to-aes-in-software, Zugriff: 20.06.2018)

[39] D. A. Osvik, Adi Shamir, Eran Tromer, 2005, *Cache Attacks and Countermeasures: the Case of AES*, (verfügbar auf: https://www.cs.tau.ac.il/ tromer/papers/cache.pdf, Zugriff: 20.06.2018)

[40] NIST, 2001, *FIPS 140-2-Standard und Selbstverschlüsselung*, (verfügbar auf: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf, Zugriff: 14.07.2018)

[41] Prof. Dr. Ralf Küsters, 2017, *Grundlagen der Informationssicherheit: Symmetric Encryption*, (Foliensatz "Symmetric Encryption" der Vorlesung "Grundlagen der Informationssicherheit", Folie 313, Zugriff: 14.07.2018)

[42] Joel Hruska, 2018, *What is Speculative Execution?*, (verfügbar auf: https://www.extremetech.com/computing/261792-what-is-speculative-execution, Zugriff: 14.07.2018)

[43] Stack Overflow, 2010, *https://stackoverflow.com/questions/2445242/what-does-the-kernel-virtual-memory-of-each-process-contain*, (verfügbar auf: https://stackoverflow.com/questions/2445242/what-does-the-kernel-virtual-memory-of-each-process-contain, Zugriff: 14.07.2018)

[44] Yuval Yarom, Katrina Falkner, 2013, *What is Speculative Execution?*, (verfügbar auf: https://eprint.iacr.org/2013/448.pdf, Zugriff: 14.07.2018)

[45] Matt Klein, 2018, *Meltdown and Spectre, explained*, (verfügbar auf: https://medium.com/@mattklein123/meltdown-spectre-explained-6bc8634cc0c2, Zugriff: 14.07.2018)