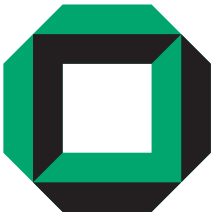
Seminar

Praktische Kryptoanalyse



Universität Karlsruhe (TH)
Fakultät für Informatik
Institut für Kryptographie und Sicherheit
Europäisches Institut für Systemsicherheit

J. Müller-Quade
A. Sobreira de Almeida
D. Kraschewski

Wintersemester 2009/10

Europäisches Institut für Systemsicherheit
Institut für Kryptographie und Sicherheit
Fakultät für Informatik
Universität Karlsruhe (TH)
Am Fasanengarten 5
76128 Karlsruhe

Inhaltsverzeichnis

Kapitel 1

Fault Induction & Bug Using

Dominik Bruhn

1.1 Motivation

Angenommen folgendes fiktive und vereinfachte Szenario:
Die Botschaft eines „Schurkenstaates“ in den USA erhält zwei Arten von Sendungen per Post: Zum einen werden der Botschaft Pakete mit harmlosen Akten zugestellt. Jeder Aktenordner wiegt 3 kg. In einer Sendung können beliebig viele Aktenordner enthalten sein. Zum anderen erhält die Botschaft regelmäßig Pakete mit Bombenzündern zu je 5 kg zugeschickt. Auch hier können beliebig viele in einem Paket enthalten sein.
Den Behörden am Zoll ist das Problem bekannt. Da aber die Pakete sich äußerlich nicht unterscheiden und aus diplomatischen Gründen nicht geöffnet werden dürfen, sind die Agenten dazu übergegangen die Pakete zu wiegen: Pakete die ein Vielfaches von 5kg enthalten werden aus dem Verkehr gezogen, Pakete die ein Vielfaches von 3 kg wiegen werden an die Botschaft ausgeliefert. Dieses Vorgehen stößt jedoch bei einem 15 kg schweren Paket an seine Grenzen: Wie können die Agenten trotzdem noch sicherstellen dass keine Bombenzünder an die Botschaft geliefert werden? Eine einfache Lösung ist: Das Paket wird so schlecht behandelt, dass eventuell enthaltene Bombenzünder auf jeden Fall defekt sind. Danach wird das Paket an die Botschaft ausgeliefert. Diese Beschädigung wird anders als das Öffnen der Pakete keinen Verdacht auf die Agenten lenken. Die Botschaft wird nun Ersatz aus dem Ausland anfordern. Ist der Inhalt der neuen Sendung eindeutig zu identifizieren wird wie oben beschrieben verfahren, ist dagegen keine Eindeutige Entscheidung auf Grund des Gewichts möglich wird die Sendung wiederum beschädigt.
Dieses einfache Beispiel zeigt, wie durch das absichtliche Einbringen von Fehlern (die Pakete bzw. deren Inhalt war nachher fehlerhaft) Rückschlüsse auf eigentlich geheime Informationen (der Inhalt der Pakete) gezogen werden kann. (Frei nach [Nac05]).

1.2 Einleitung

Diese Arbeit zeigt Szenarios auf, unter denen sich Prozessoren, die kryptografische Berechnungen durchführen, nicht mehr wie spezifiziert verhalten. So ändern zum Beispiel bestimmte Register, Speicherstellen oder Befehle ihren Inhalt oder zeigen veränderte Auswirkungen. Zu Beginn dieser Ausarbeitung werden diese Szenarios mit Beispielen beschrieben, im Anschluss wird auf Basis der mathematischen Grundlagen erklärt, wieso diese einfachen Fehler dazu genutzt werden können ein Angriff auf ein Kryptosystem durchzuführen. In einem letzten Kapitel werden mögliche Gegenmaßnahmen für die verschiedenen Szenarios und Algorithmen beschrieben.

1.3 Fault Using

Unter *Fault Using* bzw. *Fault Attacks* verstehen Boneh, Demillo und Lipton [BDL97] eine von ihnen entdeckte Klasse von Fehlern: Ein Mikrocontroller oder Prozessor wird außergewöhnlichen Bedingungen, wie zum Beispiel Strahlung, Hitze oder Eingabesignale außerhalb der Spezifikation ausgesetzt. Unter diesen Einflüssen ergeben manche Operationen der Recheneinheit ein verändertes und damit falsches Ergebnis. Diese Fehler lassen sich nun ausnützen. Die Vorbedingung für diesen Angriff ist jedoch physikalischer Zugang zu dem anzugreifenden Gerät. Dieses muss sich also in den Händen des Angreifers befinden, um es den oben beschriebenen außergewöhnlichen Bedingungen auszusetzen. Während dies bei Angriffen auf Smartcards, wie sie im PayTV-Bereich verwendet werden, kein Problem darstellt, macht es Angriffe auf Personal Computer oder entfernte Systeme über ein Netzwerk wesentlich schwieriger.

1.3.1 Geschichtliches

In den 70er Jahren entdeckten Chiphersteller, dass bestimmte Materialien, die sie in den Chipgehäusen verwendeten, zu Rechenfehlern führten. Dies wurde von Wissenschaftlern ([MW78]) auf die in den Verpackungsmaterialen enthaltenen schwachen α-Strahler zurückgeführt. Diese Strahlen in den Gehäusen führten dazu, dass einige Bits spontan ihren Wert änderten. Obwohl laut Studien diese strahlenden Elemente nur im Verhältnis 1 zu 1 Million in den Gehäusen vorkommen, reicht diese geringe Konzentration aus, das Verhalten der Chips zu verändern.
Flugzeug- und Raumfahrthersteller fanden ([ZL80]) mit der sogenannten kosmischen Strahlung eine weitere Fehlerquelle. Aufgrund der Erdatmosphäre sind diese Strahlen auf dem Erdboden sehr schwach, nehmen aber mit zunehmender Höhe zu. Das heißt, dass Chips in Flugzeugen und vor allem in Raketen wesentlich stärker kosmischer Strahlung ausgesetzt sind, als auf dem Boden. Außerdem nahmen und nehmen die Speichergrößen immer weiter zu, was rein statistisch die Chance auf einen Fehler vergrößert.
Diese Erkenntnisse führten sowohl zur Erforschung dieser Fehler, als auch zur Entwicklung von Gegenmaßnahmen.

1.3.2 Manipulationsmöglichkeiten

Es lassen sich nach [AK98] sogenannte *invasive Angriffe* von sogenannten *nichtinvasiven* Angriffen unterscheiden. Erstere setzten zumeist aufwendige, teure und schwierig zu erlangende Ausrüstung voraus, um damit direkt das Verhalten des Chips auf der Transistor-Ebene zu beeinflussen. Dazu muss häufig das Gehäuse des Chips zerlegt werden und dauerhafte (daher der Name) Veränderungen sind meist unumgänglich. Zweitere basieren auf dem Verändern der äußeren Parameter wie Spannung oder Frequenz des Zeitgebers. Hierzu muss der Chip nicht zwangsläufig verändert werden.
Die erzeugten Fehler lassen sich nach [BECN+06] unterscheiden, in *vorübergehende Fehler* (*Provisional Faults*) und in *zerstörende Fehler* (*Destructive Faults*). In die erste Kategorie fällt das Umschalten eines oder mehrerer Bits durch einen Fehler. In die zweite Kategorie fallen Fehler, die entstehen, weil Transistoren (zum Beispiel durch einen Kurzschluss) dazu gebracht wurden ständig zu leiten und nicht mehr zu schalten und somit dauerhaft beschädigt sind.
Im Folgenden werden einige Angriffe aufgeführt und erklärt, wobei auf Grund der großen Anzahl nur einige Aspekte und Möglichkeiten diskutiert werden können.

1.3.2.1 Versorgungsspannung

Der Hersteller jedes Mikrochips gibt einen bestimmten Spannungsbereich (zum Beispiel im Falle von Smartcards zwischen 4,5V und 5,5V) und eine Standard-Spannung (z.B. 5V) vor. Innerhalb dieses Bereiches verhält sich die Recheneinheit wie in den Unterlagen spezifiziert. Sobald jedoch eine Spannung oberhalb oder unterhalb dieses Bereichs angelegt wird, kann es zu verschiedenen Fehlern kommen: Operationen können falsche Ergebnisse liefern oder Instruktionen können ganz ausgelassen werden. Das Ziel des Angriffs ist, genau im richtigen Moment die Versorgungsspannung kurzzeitig zu erhöhen (sogenannter *Spike* oder *Glitch*) um einen einzigen, bestimmten und vorher festgelegten Befehl zu verändern (siehe 1.2). Um den genau richtigen Moment abzupassen, ist eine ausführliche Analyse des Gerätes und ein genaues Timing notwendig. Zur Durchführung des Angriffs muss der Chip nicht verändert werden, da einfach der am Spannungseingang anliegende Wert verändert wird. Häufiges Ziel eines solchen Angriffs sind Sicherheitsüberprüfungen am Endes eines Algorithmus, die verhindern sollen, dass fehlerhafte Daten zurückgegeben werden. Auch können Sprunganweisungen ausgelassen werden, so dass etwa nicht nur das gewünschte Ergebnis sondern weitere geheime Informationen ausgegeben werden.
In dem Artikel [KQ07] führen Kim und Quisquater einen Angriff auf ein RSA-System vor: Ihnen gelang es mit vergleichsweise einfacher Ausrüstung (siehe Abb. 1.1) bestimmte Phasen der Berechnung in der RSA-Verschlüsselung zu überspringen und damit die Sicherheit des Ergebnisses zu beeinträchtigen (siehe Abb. 1.2).

1.3.2.2 Externer Taktgeber

Fast alle moderne Mikroprozessoren (und auch die meisten Smartcards) benötigen einen externen Taktgeber. Sie generieren ihr Taktsignal also nicht selbst, sondern verlassen sich auf einen von außen angeschlossenen Taktgenerator. Im einfachsten Fall hat der Chip eine eigene Leitung auf der er ein passend getaktetes Rechtecksignal erwartet. Wie schon bei der Versorgungsspannung, gibt hier der Hersteller einen bestimmten Bereich vor, in

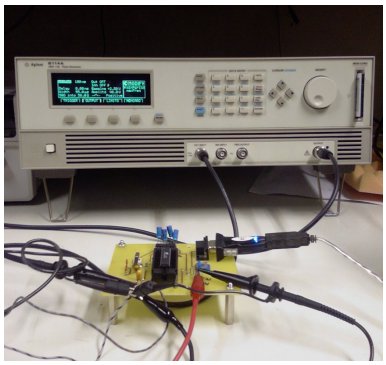


Abbildung 1.1: Versuchsaufbau in [KQ07]

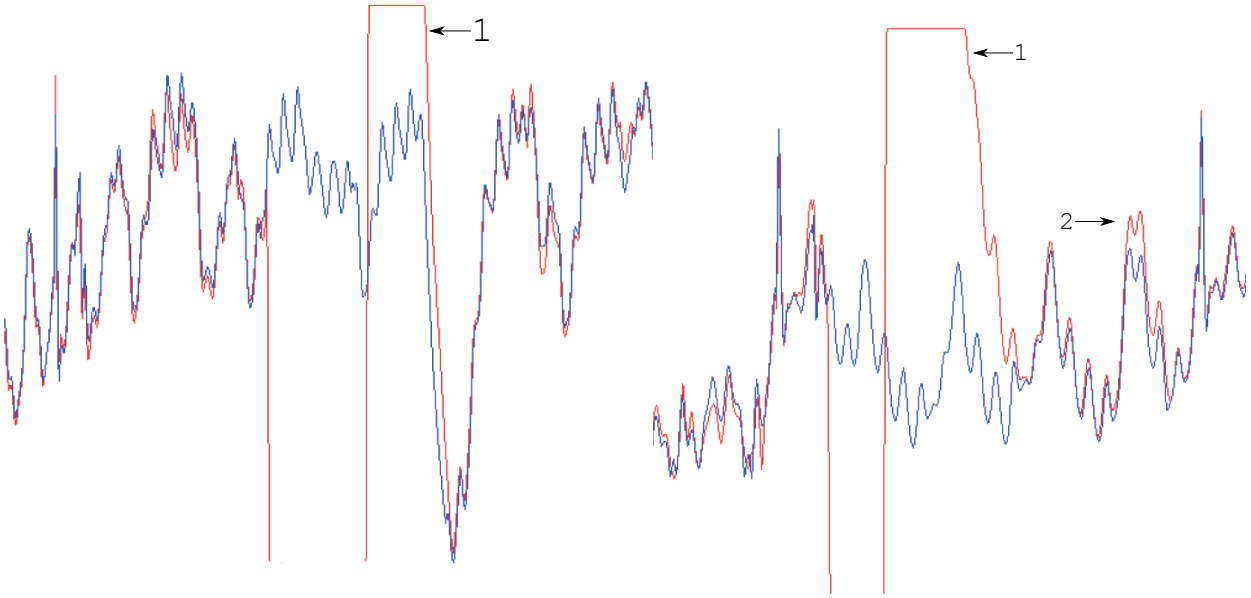


Abbildung 1.2: Stromverbrauch, bei normaler Ausführung (blau) und bei einem Angriff durch veränderte Versorgungsspannung (rot). Links: Ein Befehl wird übersprungen (siehe 1). Rechts: Ein Befehl wird übersprungen (siehe 1) und es wurden Daten verändert. Dies ist am veränderten Stromverbrauch im weiteren Verlauf zu erkennen. [Nac05]

dem sich die Frequenz bewegen darf. Erhöht man die Frequenz für einen Zyklus über die Toleranz hinaus, führt das dazu, dass bestimmte Befehle ein falsches Ergebnis liefern oder übersprungen werden [BECN+06]. Auch hier sind keine Veränderungen am Chip notwendig, da dieser ein von außen manipulierbares Taktsignal erwartet.

1.3.2.3 Licht

Während Manipulationen an der Stromversorgung und am Taktsignal an bereits vorhandenen Leitungen vorgenommen werden können, basieren die nun folgenden Angriffe auf dem Verändern von äußeren Bedingungen. Auf Grund des sogenannten *Photoelektrischen Effekts* sind elektrische Schaltkreise durch Photonen und damit Licht beeinflussbar (siehe Abb. 1.5). Sobald Photonen mit ausreichender Energie auf ein Elektron treffen wird dieses aus seiner Bindung gelöst (siehe hierzu [Wik09b]).
In ihrer Arbeit [SA03] beschreiben Skorobogatov und Anderson eine einfache und billige Möglichkeit das Verhalten eines Prozessors zu manipulieren. Hierzu wurde der SRAM, also der Arbeitsspeicher, eines Mikrocontrollers starkem Licht ausgesetzt, was dazu führte, dass einzelne Speicherzellen ihren Inhalt veränderten. Da dabei nur die Verpackung des Chips zerstört werden musste, der Chip selbst aber nicht verändert wurde, nannten sie diesen Angriff in Anlehnung an die oben beschriebene Kategorisierung *semi-invasiv*.

Sie nahmen ein einfaches Foto-Blitzlicht und montierten es auf ein Mikroskop (siehe Abb. 1.3). Um den Strahl des Lichtes weiter einzuschränken, wurde eine einfache Blende aus Alufolie benutzt.
Der verwendete Mikrocontroller wurde von seinem Gehäuse befreit (Abb. 1.4) und so programmiert, dass sich sein Speicherinhalt über eine serielle Verbindung auslesen und überschreiben ließ. Hierdurch wurde es später möglich, Veränderungen am Speicher festzustellen.
Durch Experimente gelang es den Forschern herauszufinden, welche Position auf dem SRAM-Bereich des Mikrocontrollers belichtet werden musste, damit sich genau ein bestimmtes Bit im Speicher veränderte. Als Ergebnis der Arbeit stellte sich heraus, dass es mit sehr einfachen Mitteln möglich ist einzelne Werte im SRAM gezielt zu verändern.
Im Weiteren werden die Auswirkungen dieser kleinen Modifikation auf die Sicherheit von Kryptosystemen erklärt.

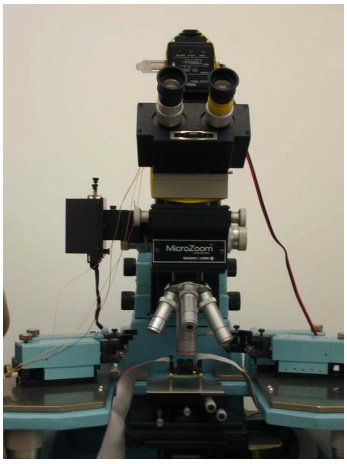


Abbildung 1.3: Mikroskop mit montiertem Blitzlicht und angeschlossenem Chip [SA03]

Abbildung 1.4: Mikrocontroller unverändert (oben) und mit entferntem Gehäuse (unten) [SA03]

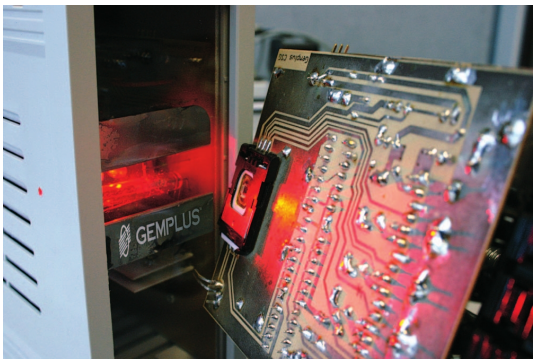


Abbildung 1.5: Blitzlampe im Testeinsatz bei einem Smartcard-Hersteller [Nac05]

1.3.2.4 Laser

Ähnlich wie Licht, können auch Laserstrahlen verwendet werden, um Fehler in Rechenwerken zu induzieren. Der Vorteil der Laserstrahlung liegt in der Genauigkeit: Laser lassen sich wesentlich einfacher auf einen bestimmten Punkt konzentrieren. Nur dort wird die hohe Energie entladen und es werden keine weiteren Speicherzellen oder Schaltkreise beeinflusst. Die hierfür benötigte Ausrüstung (siehe Abb. 1.6) ist jedoch teuer. Auch hier muss der Chip innerhalb des Gehäuses freigelegt werden, damit der Laser direkt auf die Schaltbahnen gerichtet werden kann.

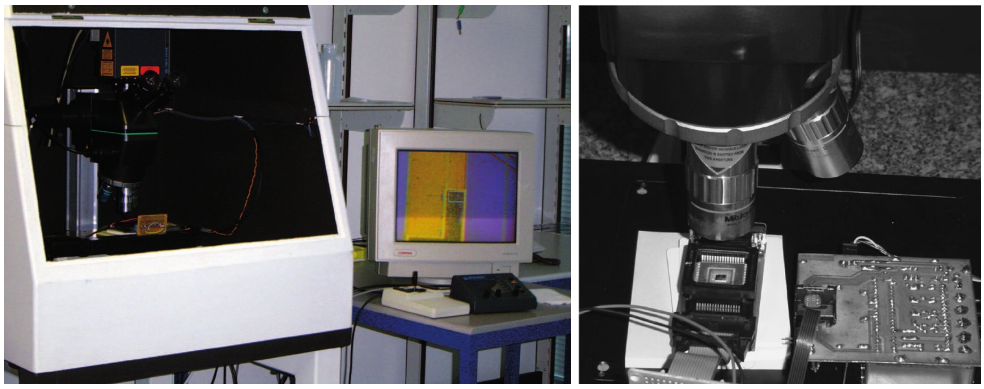


Abbildung 1.6: Ausrüstung, um per Laser Fehler zu induzieren [BECN+06], [Nac05]

1.3.2.5 Temperatur

Auch für die Temperatur (wie für die Versorgungsspannung) gibt es seitens der Hersteller einen spezifizierten Operationsbereich. Durch erhöhte, über diesen Bereich hinausgehende, Temperaturen lassen sich zwei Fehler ([BECN+06]) erzeugen: Es kann sich der Inhalt der Speicherzellen eines Mikroprozessors unter Hitze verändern. Dabei ist allerdings das Verhalten selten vorhersehbar, dass heißt, welche Werte sich nachher im Speicher finden lassen, lässt sich kaum steuern. Ein weiterer wichtiger Fehlertyp beruht auf der Tatsache, dass viele nichtflüchtige Speicher (*NVM*) sich bei bestimmten Temperaturen zwar noch auslesen, jedoch nicht beschreiben lassen oder umgekehrt. Hält man also einen Chip auf einer bestimmten Temperatur, kann verhindert werden dass dieser Werte in seinen Speicher schreibt oder aus diesem liest.

1.3.2.6 Memory Remanence

Nicht direkt im Zusammenhang mit dem Verändern der äußeren Umstände steht ein ungewöhnlicher Angriff: In [AK98] wird eine Eigenheit von Speicherchips beschrieben, die zu einem Sicherheitsproblem führt: Im Allgemeinen wird angenommen, dass der Arbeitsspeicher sofort nach dem Abschalten der Stromversorgung seinen kompletten Inhalt verliert. Das stellte sich als unwahr heraus: Unter normalen äußeren Umständen, behält ein vom Strom getrennter Speicherchip einen hohen Prozentsatz seines Inhaltes noch nach 30 Sekunden (siehe Abb. 1.7). Durch das Kühlen des Chips, zum Beispiel mit einfachem Eis-Spray, können sogar über 30 Minuten überbrückt werden, in denen kaum Informationen verloren gehen. Diese zurückgebliebenen Informationen werden als *Memory Remanence* bezeichnet.
Der Angriff ermöglicht es aus dem Arbeitsspeicher geheime Informationen, wie private Schlüssel, zu extrahieren. Zum Beispiel werden Geldautomaten automatisch vom Stromnetz getrennt sobald das Gehäuse geöffnet wird. Dies dient dem Schutz der im Arbeitsspeicher abgelegten geheimen Informationen und Schlüssel. Gelingt es dem Angreifer schnell nach dem Öffnen des Gehäuses den Arbeitsspeicher auszulesen kann er Teile dieser Schlüssel auslesen.

1.4 Bug Using

Unter *Bug Using* verstehen Biham, Carmeli und Shamir [BCS08] im Gegensatz zum *Fault Using* einen Angriff, der nicht auf dem Verändern der physikalischen Bedingungen basiert. Stattdessen wird hierbei ein bereits im Prozessor vorhandener Rechenfehler ausgenutzt. Dies macht die Fehler wesentlich gefährlicher: Während bei den oben beschriebenen Fehlern der Angreifer, um einen Fehler produzieren zu können, stets physikalischen Zugang zum Gerät oder Chip braucht, kann er nun aus beliebiger Distanz, zum Beispiel über das Internet, und ohne direkte Modifikationen handeln. Auch ist es einfacher sehr viele Geräte gleichzeitig anzugreifen, was auf Grund des Aufwandes bei den vorher beschriebenen Angriffen unmöglich war. Außerdem sind die Rechenfehler, da sie im Design der CPUs verankert sind, deterministisch, sie lassen sich also reproduzieren und vorhersagen.
Die entscheidende Frage ist: Woher kommen die Fehler in den CPUs? In [BCS08] werden mehrere Möglichkeiten diskutiert.

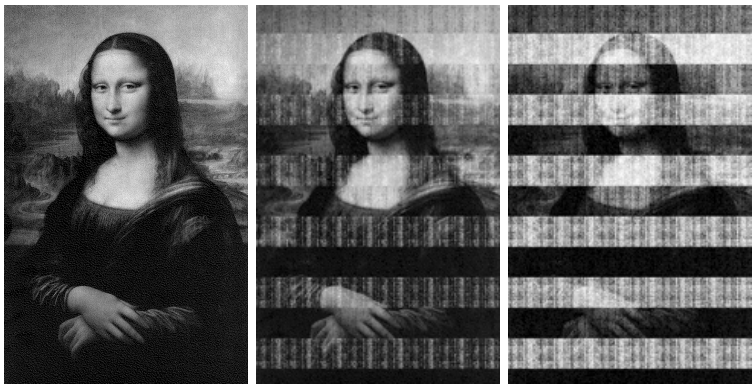


Abbildung 1.7: Ein im Hauptspeicher abgelegtes Bild 5, 30 und 60 Sekunden nach dem Abschalten der Stromversorgung. [HSH+08]

1.4.1 Unbeabsichtigte Fehler

Im Zuge der zunehmenden Komplexität von CPUs wird es wahrscheinlicher, dass sich aus Versehen Fehler im Design einschleichen. Weil die Bau- und Schaltpläne der Prozessoren geheim sind, ist es auch nicht ohne weiteres möglich, die Fehlerfreiheit des Designs zu überprüfen oder zu widerlegen. Schon für eine einfache 64 bit Multiplikation gibt es nach [BCS08] 2^{128} mögliche Einzelmultiplikationen. Sie alle auf einem Prozessor nacheinander auf Fehler zu überprüfen, ist auf Grund der benötigten Zeit, ein fast unmögliches Vorhaben. Es muss sogar davon ausgegangen werden, dass viele dieser Multiplikationen niemals von eine CPU ausgeführt werden.
Selbst wenn man davon ausgeht, dass die großen CPU-Hersteller auf diese Probleme achten und ihre Designs vor der Produktion überprüfen oder beweisen, verbleiben noch viele kleinere und mittlere Hersteller, die diesem Problem eventuell nicht die nötige Aufmerksamkeit zukommen lassen. Zusätzlich beschränkt sich das Problem nicht auf Desktop-CPUs: Auch Mobile Geräte wie PDAs oder Handys benötigen heutzutage Prozessoren um Daten zu verschlüsseln. Gerade in diesen Geräten kommen eventuell wenig getestete Prozessoren zum Einsatz.

1.4.2 Beabsichtige Fehler

Im Gegensatz zu diesen „unabsichtlichen“ Fehlern besteht die Möglichkeit, dass Rechenfehler absichtlich in den Prozessoren platziert werden. Das amerikanische Verteidigungsministerium (*DOD*) [Def05] sieht in seiner Analyse aus dem Jahr 2005 eine wachsende Gefährdung. So werden in den USA heute kaum noch Chips hergestellt. Stattdessen werden diese aus anderen Ländern, wie zum Beispiel China importiert. Auf Grund der Komplexität und der Masse (das DOD kaufte allein im Jahr 2004 Chips im Wert von 3,6 Milliarden US-Dollar) der Chips, ist es kaum möglich, zu überprüfen, ob durch eine ausländische Macht, sei es ein Staat oder ein einzelner Konzern, absichtlich Fehler eingebracht wurden. Ein zusätzliches Problem entsteht, weil sich das Verhalten moderner Prozessoren nach der Produktion noch durch das Verändern der Firmware oder des Microcodes verfälschen lässt. Dies macht Manipulationen für jeden Lieferanten und Verkäufer möglich und kann ohne den Einfluss und das Wissen des Herstellers geschehen.
Es sind jedoch laut [BCS08] bis heute keine Fälle von absichtlichen Manipulationen bekannt geworden.

1.5 Mögliche Angriffe

Es werden im folgenden nun Auswirkungen der oben beschriebenen Fehler beschrieben.

1.5.1 RSA Erklärung

RSA (nach den Erfindern Rivest, Shamir und Adleman) ist ein asymmetrisches Kryptosystem. Es kann einerseits zum Verschlüsseln von Nachrichten, andererseits auch zum Erstellen einer digitalen Signatur verwendet werden. Jeder Teilnehmer an der Kommunikation erstellt sich ein Schlüsselpaar, bestehend aus einem öffentlichen Schlüssel, den er jedem mitteilen kann, und einem privaten Schlüssel, der auf jeden Fall geheim bleiben

muss. Die Erstellung des Schlüsselpaars wird hier nicht beschrieben, sie findet sich zum Beispiel in [JSP08].
Es ist $N = p * q$ für p, q prim mit $p \neq q$. Sei außerdem e und d geeignet gewählt, dann ist (d, N) der private Schlüssel und (e, N) der öffentliche Schlüssel. Das N ist hierbei in beiden Fällen gleich. Die Zerlegung in die Faktoren p und q ist jedoch geheim und damit nur dem Schlüsselbesitzer bekannt.
Soll nun eine Nachricht K mit einem bekannten öffentlichen Schlüssel (e, N) verschlüsselt werden, so berechnet der Absender der Nachricht $C \equiv K^e \ mod \ N$.
Der Empfänger und Besitzer des privaten Schlüssels (d, N) kann nun die so verschlüsselte Nachricht C wieder dekodieren: Er berechnet $K \equiv C^d \ mod \ N$.
Die Sicherheit von RSA basiert vor allem darauf, dass es extrem aufwendig ist, N in die beiden Primfaktoren zu zerlegen.

1.5.2 Chinesischer Restsatz

Der Chinesische Restsatz (englisch *Chinese Remainder Theorem* oder kurz *CRT*) stammt aus dem Bereich der Zahlentheorie. Er wird häufig verwendet, um die Entschlüsselung und die Signierung mit RSA zu beschleunigen ([BDL97], [BCS08]). Hier wird nur die spezielle Fassung (aus [JSP08]) für zwei simultane Kongruenzen, wie sie für die RSA Implementierung benötigt wird, beschrieben. Eine allgemeine Version findet sich zum Beispiel in [Wik09a]).
Es seien für eine unbekannte Zahl x zwei Kongruenzen a modulo p und b modulo q bekannt, wobei p und q teilerfremd sind:

$$\begin{aligned} x &\equiv a \ mod \ p \\ x &\equiv b \ mod \ q \\ a, b, p, q \ bekannt; \ ggT(p, q) &= 1 \end{aligned}$$

dann lässt sich $x \ mod \ n$ mit $n = p * q$ wie folgt bestimmen: Finde zwei beliebige ganze Zahlen k und l, so dass gilt:

$$1 = k * p + l * q$$

Setzte nun

$$\begin{aligned} \alpha &= k * p \\ \beta &= l * q \end{aligned}$$

Dann ist $x \equiv \alpha * b + \beta * a \ mod \ n$. x löst auch die oben genannten Kongruenzen (Beweis siehe [JSP08]).

1.5.3 Angriffe auf RSA

Ob und wie RSA angegriffen werden kann, hängt stark von der Implementierung ab. Während die mathematischen Grundlagen eindeutig sind, ist die Realisation der einzelnen Operationen, wie das Potenzieren, eine Frage der Implementierung. Deshalb werden im Weiteren verschiedene Vorgehensweisen unterschieden. Vor allem auf kleinen Geräten oder Smartcards spielt auch die Geschwindigkeit der Implementierung eine Rolle. Es muss daher oft zwischen Sicherheit und Geschwindigkeit abgewogen werden.

1.5.3.1 RSA mit CRT

Der Chinesische Restsatz (*CRT*) kann den RSA-Algorithmus um das Vierfache beschleunigen ([JSP08]). Das spielt vor allem auf kleinen eingebetteten Geräten und Chipkarten eine Rolle, da hier die zur Verfügung stehende Rechenkapazität begrenzt ist. Der Einsatz des CRT setzt jedoch voraus, dass die Primfaktorzerlegung p und q von N des Schlüssels bekannt ist. Da diese Informationen geheim gehalten werden müssen, kann der CRT nur für die Entschlüsselung und die Signierung, nicht jedoch für die Verschlüsselung verwendet werden. Zumeist werden die Primfaktoren zusammen mit dem privaten Schlüssel so auf dem Gerät abgelegt, dass sie nicht ausgelesen werden können.
Für die Entschlüsselung einer Nachricht C mit dem privaten Schlüssel (d, N) muss gelten (siehe 1.5.1) $K \equiv C^d \ mod \ N$. Diese Operation ist sehr aufwendig, da sowohl d als auch N sehr groß sind, und es bei der naiven

Implementierung, zuerst C^d zu berechnen, schnell zu einem Überlauf kommt.
Hier kann der CRT helfen: Es wird zuerst $K_1 = C^d \ mod \ p$ und $K_2 = C^d \ mod \ q$ berechnet. Dann finden sich wie oben beschrieben α und β, so dass $K = \beta * K_1 + \alpha * K_2$. Das getrennte Potenzieren benötigt zwar ungefähr die gleiche Anzahl an Multiplikationen, es halbieren sich aber die Größen der Zahlen. Dies führt dazu, dass diese Implementierung ungefähr um den Faktor 4 schneller ist ([BCS08]).

1.5.3.2 Angriff auf RSA mit CRT

Die Beschleunigung durch den Einsatz von CRT erzeugt jedoch ein Sicherheitsproblem: In [BCS08] wird ein sehr einfacher Angriff auf RSA beschrieben, der sich einsetzten lässt, falls CRT zur Implementierung verwendet wird und der Prozessor inkorrekte Rechenergebnisse liefert:
Die Idee ist eine Nachricht zweimal signieren zu lassen. Sollte bei einer der Berechnungen ein Rechenfehler auftreten, so kann N faktorisiert werden und somit erhält der Angreifer den privaten Schlüssel, mit dem er nun selbst Nachrichten signieren und entschlüsseln kann. Sei K eine Nachricht, die der Angreifer frei wählt. Ihm sei auch der öffentliche Schlüssel (e, N) des Gerätes bekannt. Es sei nun C die korrekte Signatur und $\hat{C}$ die durch einen oder mehrere Rechenfehler entstandene Signatur. Um diese Signaturen intern zu errechnen, führt das Gerät mit dem CRT, wie oben beschrieben, die Berechnungen durch:

$$C = \beta * K_1 + \alpha * K_2 \ mod \ N$$
$$\hat{C} = \beta * \hat{K_1} + \alpha * \hat{K_2} \ mod \ N$$

Es wird angenommen es käme nur bei der Berechnung von $\hat{K_1}$ oder $\hat{K_2}$ zu einem Rechenfehler, nicht jedoch bei beiden. Ohne Beschränkung der Allgemeinheit kann weiter angenommen werden, es käme zu einem Fehler bei der Berechnung von $\hat{K_1}$. Es ist also $K_1 \neq \hat{K_1}$ und $K = \hat{K_2}$. Es gilt dann weiter:

$$\begin{aligned} ggT(C - \hat{C}, N) = ggT(\beta * K_1 + \alpha * K_2 - (\beta * \hat{K_1} + \alpha * K_2), N) = \\ ggT(\beta * K_1 + \alpha * K_2 - \beta * \hat{K_1} - \alpha * K_2, N) = \\ ggT(\beta * K_1 - \beta * \hat{K_1}, N) = ggT(\beta * (K_1 - \hat{K_1}), N) = \\ ggT(l * q * (K_1 - \hat{K_1}), p * q) = q \end{aligned}$$

Somit ist q, einer der beiden Primfaktoren von N, durch die Operation $ggT(C - \hat{C}, N)$ zu berechnen, woraus direkt mit $p = N/q$ auch der andere Faktor folgt. Somit ist es gelungen N zu faktorisieren. Damit ist das gesamte System unsicher geworden. Wer im Besitz der Primfaktoren ist, kann den privaten Schlüssel errechnen und eigene Nachrichten signieren oder dekodieren. Der hier beschriebene Angriff ist sehr allgemein gehalten: Es werden keine Anforderungen an die Art des Fehlers gestellt: Es kann sich um jede der oben besprochenen Manipulationen handeln. Es ist außerdem unerheblich, welche Rechenfehler passieren und wie oft diese auftreten, solange sie nur bei der Berechnung von entweder $\hat{K_1}$ oder $\hat{K_2}$ passieren. Ähnliche Angriffe lassen sich auf andere Kryptosysteme anwenden, die den CRT einsetzen ([BCS08]).
In [BCS08] wird ein weiterer Angriff auf RSA mit dem CRT beschrieben: Angenommen es ist bekannt, dass für ein bestimmtes Paar (a, b) die Multiplikation falsch berechnet wird, das Ergebnis also $\neq a * b$ ist. Durch die geschickte Wahl eines zu signierenden Textes kann der private Schlüssel erlangt werden. Hierzu werden die beiden Faktoren a und b im zu signierenden Text so verwendet, dass es beim Potenzieren zur Multiplikation $a*b$ kommen muss. Da diese Multiplikation aber nach Annahme fehlerhaft ist, ist auch das Ergebnis der Potenzierung falsch. Es kann nun wie oben vorgegangen werden, um den privaten Schlüssel zu erhalten. Dieser Angriff spielt vor allem im Falle des *Bug Usings* eine Rolle, da hier eben von der Existenz einer solchen fehlerhaften Multiplikation ausgegangen wird.

1.5.4 Algorithmen mit LTOR/RTOL

Da Implementierungen von RSA die auf dem CRT basieren sehr einfach gebrochen werden können, gibt es alternative Berechnungen für die Funktion $f(x) = x^s \ mod \ N$. Nachfolgend werden zwei häufig verwendete, schnelle Implementierungen ([BCS08], [BDL97]) dieser Funktion beschrieben. Diese benötigen zusätzlich die Binärdarstellung von s: $s = s_{n-1}s_{n-2}...s_1s_0$ mit $\forall 0 \leq i < n : s_i \in \{0, 1\}$:

- Berechnung nach LTOR (*Left To Right*)
  $z \leftarrow 1$
  for $k = log(N)$ **down to** 0
  if $s_k = 1$ **then** $z \leftarrow z^2 * x \ mod \ N$
  else $z \leftarrow z^2 \ mod \ N$
  return z
- Berechnung nach RTOL (*Right to Left*)
  $y \leftarrow x$
  $z \leftarrow 1$
  for k=0 **to** $log(n)$
  if $s_k = 1$ **then** $z \leftarrow z * y \ mod \ N$
  $y \leftarrow y^2 \ mod \ N$

In [BDL97] wird ein Angriff auf eine auf RTOL basierende RSA-Implementierung erklärt: Anforderung ist hierbei allerdings, dass alle Operationen korrekt ablaufen, es also zu keinen Rechenfehlern innerhalb von Befehlen kommt. Stattdessen geht der Angriff von defekten oder manipulierten Registern aus: Mit einer sehr kleinen Wahrscheinlichkeit ändern einige wenige Bits eines Registers spontan ihren Wert. Dabei muss die Wahrscheinlichkeit für diese Änderung so gering sein, dass sie sehr selten während der Berechnung des Algorithmus auftritt. Im Falle von diesen, *register faults* genannten Fehler, konnten die Autoren folgendes zeigen:
Angenommen, es treten bei jeder Ausführung des RTOL Algorithmus $(n/m)log2n$ Fehler auf (für jedes $1 \leq m \leq N$). Es kann nun durch das Ausführen von $\mathcal{O}((2^m n^3 log^2 n)/m^3)$ RSA-Verschlüsselungsoperationen der geheime private Schlüssel mit einer Wahrscheinlichkeit von 0,5 gefunden werden. Treten also im speziellen genau $N = p * q$ Fehler auf $(m = log(N))$, dann benötigt man genau $\mathcal{O}(n^3)$ RSA-Operationen (Beweis siehe [BCS08]).

In [BCS08] wird ein Angriff auf RSA mit LTOR beschrieben: Wieder ist ein fehlerhaftes Faktorenpaar a, b einer Multiplikation bekannt. Die Idee hierbei ist, jedes einzelne Bit des privaten Schlüssels einzeln zu erhalten. Hierzu wird für jedes Bit so lange nach einem C gesucht, bis C einerseits b enthält, und andererseits a in $C^{d'} \ mod \ N$ enthalten ist. Dabei ist d' der schon bekannte Teil des privaten Schlüssels. Ein so gewählter Text C wird auf dem fehlerhaften Chip mit RSA entschlüsselt, dies ergibt K. Der entschlüsselte Text wird nun wieder verschlüsselt in $\hat{C}$ gespeichert. Dies ist möglich, da der öffentliche Schlüssel bekannt ist. Ist nun $\hat{C} = C$, so kann daraus geschlossen werden, dass das aktuell untersuchte Bit des privaten Schlüssels 0 sein muss. Falls die Texte ungleich sind, ist das Bit 1.
Nach diesem Schema kann vom höchsten zum niederwertigsten Bit vorgegangen werden. Um einen passenden Klartext zu finden, benötigt man ungefähr 2^{27} Operationen. Dies wird für jedes Bit des Schlüssels durchgeführt. Bei einem 1024 bit $(= 2^{10}$ bit) langen Schlüssel benötigt man also $2^{10} * 2^{27} = 2^{37}$ Operationen. Einen Angriff, bei dem der zu verschlüsselnde Text bei jedem Durchlauf neu angepasst werden muss, nennt man *Adaptive Chosen Ciphertext Attack*. Es gibt auch einen Angriff ([BCS08]) auf RSA-Implementierungen die auf LTOR basieren, bei denen zu Beginn einmalig 2^{39} Klartexte ausgewählt werden. Diese Klartexte müssen alle den Faktor b der fehlerhaften Multiplikation enthalten. Hierdurch lässt sich der Aufwand für den Angriff senken, jedoch wird das Vorgehen, um das jeweilige Bit zu bestimmten, komplizierter. Der Gesamtaufwand für diesen Angriff beträgt 2^{29}. Diese Angriffe werden *Chosen Ciphertext Attack* genannt, da hier der oder die Klartexte vorher feststehen und nicht verändert werden.

Es lassen sich auf ähnliche Art andere Krypto-Systeme angreifen, falls sie RTOL oder LTOR zur Potenzierung einsetzen. So lässt sich auch aus symmetrische Kryptosystemen, also solche, bei denen der gleiche Schlüssel zum Ver- und zum Entschlüsseln verwendet wird, der geheime Schlüssel extrahieren. In [BCS08] wird ein Angriff auf die *Pohlig-Hellman-Chiffre* vorgestellt der eingesetzt werden kann, falls diese entweder mit RTOL oder mit LTOR implementiert wird. Bei dieser Chiffre handelt es sich um eine symmetrische Verschlüsselung, die ebenfalls auf der Potenzierung modulo N basiert. Es lassen sich für beide Implementierungen sowohl *Adaptive Chose Ciphertext* als auch *Chosen Ciphertext* Angriffe finden.

1.5.4.1 RSA mit OAEP

Bei OAEP (*Optimal Asymmentric Encryption Padding* handelt es sich um eine Methode den Klartext, vor der Verschlüsselung, so zu transformieren, dass das Ergebnis der Verschlüsselung sich nicht mehr einfach vorhersagen lässt. Hierzu wird dem Klartext eine bestimmte Anzahl an zusätzlichen zufälligen Zeichen hinzugefügt und zusätzlich eine Einwegfunktion auf Teile dieses verlängerten Textes angewendet. Erst dann wird der Text mit

einem frei wählbaren Algorithmus verschlüsselt.
Die zusätzliche Sicherheit basiert darauf, dass ein zufällig gewählter verschlüsselter Text mit hoher Wahrscheinlichkeit nicht den Anforderungen der OAEP-Dekodierung genügt und deshalb die gesamte Entschlüsselung abgebrochen wird. Es fällt also wesentlich schwerer, einen verschlüsselten Text zu raten, um dann Schlüsse auf den Klartext zu ziehen.
OAEP wird häufig in Kombination mit RSA verwendet. Dabei ist der grobe Ablauf wie folgt:
$C = RSACrypt(OAEPEncode(K))$ um einen gegebenen Klartext K zu verschlüsseln und
$K = OAEPDecode(RSADecrypt(C))$ um den verschlüsselten Text C zu entschlüsseln. Jedoch kann OAEP mit jedem Verschlüsselungsalgorithmus verwendet werden, hierzu werden einfach die Funktionen *RSACrypt* und *RSADecrypt* durch die passenden Alternativen ausgetauscht.
OAEP macht die oben beschriebenen Angriffe schwieriger: Es kann nicht einfach ein Text C gewählt werden der b enthält, da nach der OAEP-Enkodierung das a nicht mehr im Text vorkommt. Es muss also ein zufälliger Text gefunden werden, der nach der Anwendung der Enkodierung b enthält damit ein Angriff ausgeführt werden kann. Allerdings beobachten [BCS08], dass die Wahrscheinlichkeit, dass eine bestimmte vergleichsweise kurze Zahlenfolge im enkodierten Text irgendwo vorkommt, relativ hoch ist.
Soll eine mit OAEP gesicherte RSA-Implementierung, die LTOR benutzt, angegriffen werden, so muss ein C gefunden werden, so dass b in $C' = OAEPEncode(C)$ enthalten ist und zusätzlich $C^{d'}$ gilt. Die Wahrscheinlichkeit, dass für ein zufällig gewähltes C diese beiden Bedingungen gelten, liegt bei 2^{-54}. Hierdurch steigt die Komplexität des Angriffs von 2^{27} ohne OAEP, auf 2^{64} mit dem Schutz durch OAEP.

1.5.5 DES

Während bis jetzt nur Angriffe auf RSA diskutiert wurden, lassen sich mit den oben beschriebenen Fehlern auch andere Verschlüsselungen angreifen. DES (*Data Encryption Standard*) ist im Gegensatz zu RSA eine symmetrische Blockverschlüsselung. Dabei wird jeder Block in 16 Runden nach einem bestimmten Schema mit dem geheimen Schlüssel ''verwürfelt''. Nach jeder Runde wird das Ergebnis mit den vorhergehenden mit XOR verknüpft. Der Schlüssel ist 64 bit lang, wobei davon 8 bit für Prüfsummen benutzt werden, es stehen also effektiv nur 56 bit zur Verfügung.
In [AK98] wird ein einfacher Angriff beschrieben: Angenommen, es wäre möglich, einzelne Befehle zu überspringen oder auszulassen. Man deaktiviert nun die zwei letzten der 16 Runden. Dies kann geschehen, indem die XOR-Operationen übersprungen werden. Aus dem Unterschied zwischen dem falsch und dem korrekt verschlüsselten Text lassen sich zwei oder manchmal drei sogenannte S-Boxen ermitteln. Aus diesen wiederum können nun bis zu 5 bit des Schlüssels berechnet werden. So lassen sich mit 6 gut gewählten Eingabetexten bis zu 30 bit des Schlüssels errechnen. Durch die Reduktion des Suchraumes ist nun ein Brute-Force-Angriff schnell durchzuführen.
Die Schwierigkeit ist vor allem die XOR-Operationen zu deaktivieren. Da jedoch viele Smartcard-Hersteller ihren Karten ein Development-Kit mit Standard-Implementierungen verschiedener Algorithmen, unter anderem auch DES, beilegen, lässt sich die Position der XOR-Instruktionen genau rekonstruieren. Dies gilt vor allem, da diese Entwicklungskits gut dokumentiert und nicht geheim sind. Selbst wenn der Quelltext der DES-Implementierung nicht vorliegt, kann immer noch durch einfaches Ausprobieren herausgefunden werden, welche Operationen die zu deaktivierenden XOR-Operationen sind. Der Suchraum für die Operationen ist auf Grund der einfachen Implementierung von DES sehr klein.

Ein weiterer Angriff ([AK98]) nutzt aus, dass der Schlüssel von DES eine Prüfsumme enthält. Es ist davon auszugehen, dass die Implementierungen zu Beginn einer Verschlüsselung prüfen, ob die Prüfsumme korrekt ist. Falls die Prüfsumme nicht korrekt ist, wird die Funktion einen Fehler ausgeben. Angenommen der Schlüssel liegt in einem Speichermodul, zum Beispiel einem EEPROM, auf das von außen nicht lesend zugegriffen werden kann. Es besteht jedoch die Möglichkeit einzelne Bits in diesem Speichermodul zu setzten (zum Beispiel wie in 1.3.2.3 beschrieben). Man setzt nun ein Bit des Schlüssels auf "1" und führt eine beliebige Verschlüsselung aus: Zeigt das Modul einen Prüfsummen-Fehler an, so ist dieses Bit im Schlüssel eine "0" gewesen. Falls die Verschlüsselung ohne Fehler durchläuft, muss es sich um eine "1" gehandelt haben. Wiederholt man dieses Vorgehen für alle Bits, erhält man den kompletten Schlüssel, auch ohne die Implementierung des DES-Algorithmus zu kennen.

1.5.6 Elliptic Curve Cryptography

Im Gegensatz zu RSA, dessen Sicherheit darauf basiert, dass das Faktorisieren einer Zahl schwierig ist, basiert *Elliptic Curve Cryptography* (*ECC*) auf der Schwierigkeit, den diskreten Logarithmus in der Gruppe der Punkte einer elliptischen Kurve zu berechnen. Die oben genannten Angriffe lassen sich jedoch auch auf dieses Kryptosystem übertragen ([BCS08]), da auch hier große Zahlen multipliziert werden und damit bei bekannten Multiplikationsfehlern Rückschlüsse auf einzelne Bits des privaten Schlüssels gezogen werden können.

1.5.7 Code-Angriffe

Die bisher beschriebenen Angriffe galten einzelnen Algorithmen. Aber auch auf Instruktionsebene lassen sich die Fehler ausnützen: Durch das geschickte Auslassen von Befehlen, wie es, wie oben beschrieben, möglich ist, lassen sich im Speicher abgelegte geheime Informationen, wie private Schlüssel, auslesen.
Angenommen nach dem Dekodieren mit einem geheimen, im Speicher abgelegten Schlüssel, soll das Ergebnis ausgegeben werden. Hierzu wird folgender Code (aus [AK98]) verwendet:

```
1   b = answer_address
2   a = answer_length
3   if (a == 0) goto 8
4   transmit(*b)
5   b = b + 1
6   a = a - 1
7   goto 3
8   ....
```

Die Idee ist nun, bestimmte Instruktionen auszulassen. Gelingt es entweder die Abbruch-Bedingung in Zeile 3 oder das Verringern der Schleifenvariable in Zeile 6 zu überspringen, so wird, anstatt des Ergebnisses, der komplette Speicher übermittelt. Dadurch wird auch der geheime private Schlüssel öffentlich.
Mit ähnlichen Modifikationen, bei denen anstatt mehrerer Durchläufe nur ein einziger Transformationsdurchlauf ausgeführt wird, lassen sich Verschlüsselungsoperationen ganz überspringen oder stark vereinfachen. Alle diese Angriffe setzen jedoch die Kenntnis des Quellcodes, oder zumindest der Position der Operation, die übersprungen werden soll, voraus. Diese Informationen können auf verschiedene Arten gewonnen werden: Falls der Programmierer Standard-Implementierungen verwendet hat, können diese betrachtet werden, ansonsten muss das Programm einer Codeanalyse unterzogen werden.

1.6 Gegenmaßnahmen

Die beschriebenen Angriffe lassen sich auf verschiedenen Ebenen und damit von verschiedenen Personenkreisen oder Organisationen verhindern: Während Angriffe auf Hardware-Ebene durch den Hersteller des Chips unterbunden werden können, lassen sich Angriffe auf Kryptosysteme, vor allem durch Modifikationen, durch Verschlüsselungsexperten abwehren.

1.6.1 Auf Hardware-Ebene

Hardware-Hersteller sind dazu übergegangen, sicherheitsrelevante Chips, wie Smartcards, mit aufwendigen Schutzfunktionen auszustatten. Die Gehäuse werden so mit den Schaltkreisen verbunden, dass jedes Zerlegen der Chips diese automatisch zerstört. Außerdem wird der eigentliche Chip von einem Metallgeflecht umhüllt. Sobald dieses zerstört, beschädigt oder unterbrochen wird, deaktiviert sich der Chip.
Eine weitere Möglichkeit sind aktive Sensoren: Ein Lichtsensor kann den Angriff, wie er in 1.3.2.3 beschrieben wird, erkennen und bei zu starken Lichteinfall den Chip temporär oder dauerhaft abschalten, so dass keine Rechenfehler auftreten können. Ähnliche Sensoren helfen auch gegen überhöhte Temperaturwerte (siehe 1.3.2.5), Spannungsschwankungen (siehe 1.3.2.1) und Modifikationen an der Taktfrequenz (siehe 1.3.2.2).
Das Hauptproblem dieser Schutzmechanismen ist die mangelnde Effizienz: Angriffe werden erschwert, aber nicht verhindert: Bevor der eigentliche Angriff stattfinden kann, muss zuerst der Schutzmechanismus ausgehebelt werden.
In [BECN+06] beschreiben die Autoren verschiedene Möglichkeiten, durch das redundante Auslegen bestimmter Bauteile (wie zum Beispiel der arithmetischen Einheiten), die Sicherheit zu erhöhen:

- *Simple Duplication with Comparison (SDC)* und *Multiple Duplication with Comparison (MDC)*: Ein Hardwareblock wird zweimal (*SDC*) oder mehr als zweimal (*MDC*) dupliziert und jede Rechenoperation wird von allen Blöcken gleichzeitig durchgeführt. Nach der Operation werden die Ergebnisse der einzelnen Blöcke verglichen. Sind diese nicht identisch, bestehen zwei Handlungsalternativen: Es kann der komplette Prozessor zurückgesetzt werden, da von einem Angriff auszugehen ist. Es kann aber auch bei *MDC* ein Mehrheitsentscheider eingebracht werden. Hierbei handelt es sich um ein Bauteil, dass das häufigste Ergebnis als Ausgabe annimmt und die Ausführung fortsetzt. Liefern zum Beispiel 2 Blöcke a und 1 Block b als Rückgabe, so wird a als Endergebnis angenommen.
  Eine solche Verdopplung verhindert nur fokussierte Angriffe, also Angriffe auf einzelne Teilbereiche des Chips. Wird dagegen der gesamte Chip, inklusive der beiden Blöcke, gleichmäßig angegriffen, werden alle Blöcke die gleichen falschen Ergebnisse liefern und der Fehler wird nicht erkannt.

- *Simple Duplication with Complementary Redundancy (SDCR)*: Einem Hardwareblock wird ein weiterer Block zur Seite gestellt, der genau die inverse Operation durchführt. Dieser Block erhält die invertierte Gesamteingabe als Berechnungsgrundlage. Nach der gleichzeitigen Ausführung müssen wie bei *SDC* die Ergebnisse identisch sein. Da es schwierig ist, zwei Fehler mit gegenteiligem Effekt zu erzeugen, schützt dies auch wirksam gegen großflächige Angriffe.

- *Simple Time Redundancy with Comparison (STRC)* und *Multiple Time Redundancy with Comparison (MTRC)*: Jeder Block wird zweimal (STRC) oder mehrmals (MTRC) hintereinander mit den gleichen Eingaben versorgt. Das Ergebnis der Durchläufe wird danach verglichen. Danach kann wie bei *MDC* auch entweder das häufigste Ergebnis (*MTRC*) genutzt oder der Chip gesperrt werden. Es gelten die gleichen Einschränkungen, wie bei *MTRC* in Bezug auf fokussierte Angriffe.

- *Re-computing with Swapped/Shifted Operands*: Bei beiden Vorgehensweisen wird jede Operation auf einem Block zweimal durchgeführt. Einmal mit den unveränderten Eingaben und einmal mit umgekehrter Byte-Reihenfolge (*Endianness*), im Falle von *Swapped Operands* oder mit um einen konstanten Wert verschobenen Bytes, im Falle von *Shifted Operands*. Nach der Ausführung werden diese Operationen wieder rückgängig gemacht und das Ergebnis verglichen.

Da *SDC*, *MDC* und *SDCR* darauf basieren, dass Blöcke vervielfacht werden, erhöhen sie die Fläche des Chips und damit die Produktionskosten. Außerdem verkompliziert sich das Layout der Chips. Bei *STRC* und *MTRC* erhöht sich dagegen der Schaltungsaufwand kaum, dafür nimmt die Ausführungszeit zu.

In [SA03] schlagen die Autoren vor, für jedes Bit nicht eine Leitung, sondern zwei Leitungen zu verwenden (sogenannte *dual-rail logic*). Es soll dann "11" dem ursprünglichen "1" und "00" dem ursprünglichen "0" entsprechen. Sollten im Zuge von Übertragungen oder Berechnungen andere Werte als "00" oder "11" auftreten, so kann von einem Angriff ausgegangen werden und der Chip kann neu gestartet oder gesperrt werden.

1.6.2 Auf Software-Ebene

Während Hardwaremodifikationen zumeist teuer sind und eine Änderung am Chipdesign benötigen, ist es zumeist simpler die Implementierungen abzuwandeln ohne die Hardware zu verändern. Eine einfache Lösung ist es, Programmabschnitte einfach zweimal auszuführen und das Ergebnis im Anschluss zu vergleichen. Viele der beschriebenen Angriffe lassen sich nicht genau genug steuern, um zweimal wirklich das genau gleiche Ergebnis zu erhalten. Deshalb werden sich, im Falle einer Manipulation, die Ergebnisse der zweiten Ausführung von der ersten Ausführung unterscheiden.
Auch können Programme so erweitert werden, dass sie nach der eigentlichen Ausführung die Ergebnisse auf Plausibilität und Sicherheit überprüfen. So kann zum Beispiel geprüft werden, ob der RSA-Schlüssel eine ausreichende Komplexität hat. Eine andere Möglichkeit ist es, die inverse Operation auszuführen, um das Ergebnis zu überprüfen: Soll etwa eine Nachricht mit RSA verschlüsselt werden, so nimmt man das (eventuell manipulierte) verschlüsselte Ergebnis und entschlüsselt dieses wieder. Die entschlüsselte und die Ursprungsnachricht sollten nun identisch sein, ansonsten kann von einem Fehler oder eine Manipulation ausgegangen werden.
Alle diese Vorgehensweisen habe den Nachteil, die Ausführungszeit zu erhöhen. Das spielt vor allem auf eingebetteten Systemen und Smartcards eine Rolle, da deren Rechenkapazität begrenzt ist.

1.6.3 Auf Algorithmen-Ebene

Da der RSA-Algorithmus, falls er mit dem CRT implementiert wird, mit nur einer einzigen Berechnung gebrochen werden kann, sollte der CRT nicht verwendet werden. Stattdessen bieten sich Algorithmen wie *LTOR* oder *RTOL* an, die zwar auch keine hundertprozentige Sicherheit bieten, den Aufwand für einen Angriff jedoch stark erhöhen ([BCS08]).
Wie oben gezeigt wurde, bietet auch OAEP einen starken Schutz und erhöht den Aufwand für einen Angriff. Der Einsatz von OAEP ist nicht auf RSA beschränkt, weshalb sich auch andere Kryptosysteme, wie zum Beispiel DES, hiermit verstärken lassen.

1.7 Fazit

Für die gängigen Angriffe existieren praktikable Gegenmaßnahmen. Jedoch muss einerseits davon ausgegangen werden, dass weitere noch nicht bekannte Angriffe existieren, andererseits führen die Schutzfunktionen selbst oft zu neuen Sicherheitsproblemen. Nicht nur die Wahl der Krypto-Systems, sondern auch die Wahl der Implementierung hat Einfluss auf die Sicherheit des Gesamtprodukts, hieraus ergeben sich Arbeitsbereiche für Industrie und Forschung.

Literaturverzeichnis

[AK98] ANDERSON, Ross ; KUHN, Markus: Low cost attacks on tamper resistant devices. In: *Security Protocols* Bd. 1361, Springer Berlin / Heidelberg, 1998 (Lecture Notes in Computer Science). – ISBN 978–3–540–64040–0, 125–136

[BCS08] BIHAM, Eli ; CARMELI, Yaniv ; SHAMIR, Adi: Bug Attacks. In: *Advances in Cryptology CRYPTO 2008* Bd. 5157, Springer Berlin / Heidelberg, 2008 (Lecture Notes in Computer Science). – ISBN 978–3–540–85173–8, 221–240

[BDL97] BONEH, Dan ; DEMILLO, Richard A. ; LIPTON, Richard J.: On the Importance of Checking Cryptographic Protocols for Faults. In: *Advances in Cryptology EUROCRYPT 97* Bd. 1233, Springer Berlin / Heidelberg, 1997 (Lecture Notes in Computer Science). – ISBN 978–3–540–62975–7, 37–51

[BECN+06] BAR-EL, H. ; CHOUKRI, H. ; NACCACHE, D. ; TUNSTALL, M. ; WHELAN, C.: The Sorcerer's Apprentice Guide to Fault Attacks. In: *Proceedings of the IEEE* 94 (2006), Feb., Nr. 2, S. 370–382. http://dx.doi.org/10.1109/JPROC.2005.862424. – DOI 10.1109/JPROC.2005.862424. – ISSN 0018–9219

[Def05] DEFENSE SCIENCE BOARD TASK FORCE: *High Performance Microchip Supply*. http://www.acq.osd.mil/dsb/reports/2005-02-HPMS_Report_Final.pdf. Version: Feb. 2005. – [Online; Stand 9. November 2009]

[HSH+08] HALDERMAN, J. A. ; SCHOEN, Seth D. ; HENINGER, Nadia ; CLARKSON, William ; PAUL, William ; CALANDRINO, Joseph A. ; FELDMAN, Ariel J. ; APPELBAUM, Jacob ; FELTEN, Edward W.: Lest We Remember: Cold Boot Attacks on Encryption Keys. In: *Proc. 17th USENIX Security Symposium (Sec 08)*, 2008

[JSP08] JOACHIM SWOBODA, Stephan S. ; PRAMATEFTAKIS, Michael: *Kryptographie und IT-Sicherheit: Grundlagen und Anwendungen*. Vieweg+Teubner, 2008. – ISBN 978–3–8348–0248–4

[KQ07] KIM, Chong H. ; QUISQUATER, Jean-Jacques: Fault Attacks for CRT Based RSA: New Attacks, New Results, and New Countermeasures. In: *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems* Bd. 4462, Springer Berlin / Heidelberg, 2007 (Lecture Notes in Computer Science). – ISBN 978–3–540–72353–0, 215–228

[MW78] MAY, Timothy C. ; WOODS, Murray H.: A New Physical Mechanism for Soft Errors in Dynamic Memories. In: *Reliability Physics Symposium, 1978. 16th Annual*, 1978. – ISSN 0735–0791, S. 33–40

[Nac05] NACCACHE, D.: Finding faults [data security]. In: *Security & Privacy, IEEE* 3 (2005), Sept.-Oct., Nr. 5, S. 61–65. http://dx.doi.org/10.1109/MSP.2005.122. – DOI 10.1109/MSP.2005.122. – ISSN 1540–7993

[SA03] SKOROBOGATOV, Sergei P. ; ANDERSON, Ross J.: Optical Fault Induction Attacks. In: *Cryptographic Hardware and Embedded Systems - CHES 2002* Bd. 2523, Springer Berlin / Heidelberg, 2003 (Lecture Notes in Computer Science). – ISBN 978–3–540–00409–7, 31–48

[Wik09a] WIKIPEDIA: *Chinesischer Restsatz — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Chinesischer_Restsatz&oldid=65358343. Version: 2009. – [Online; Stand 21. November 2009]

[Wik09b] WIKIPEDIA: *Photoelektrischer Effekt — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Photoelektrischer_Effekt&oldid=66189179. Version: 2009. – [Online; Stand 8. November 2009]

[ZL80] ZIEGLER, J. ; LANFORD, W.: The effect of sea level cosmic rays on electronic devices. In: *Solid-State Circuits Conference. Digest of Technical Papers. 1980 IEEE International* Bd. XXIII, 1980, S. 70–71