Nicht triviale Sicherheitsprobleme im E- und M-Commerce

cand. ing. Andreas Ahrens *Dipl.-Ing. Hagen Ploog*

Universität Rostock
Institut für Angewandte Mikroelektronik und Datentechnik
Richard-Wagner-Str. 31, 18119 Rostock, Tel. (0381) 498 35 37
Hagen.Ploog@uni-rostock.de

Abstract. Der Aspekt eines rein softwarebasierten Angriffs auf einen Verschlüsselungsalgorithmus findet heute bereits bei dessen Entwicklung Beachtung. Häufig nicht beachtet wird die Angriffsanfälligkeit des Algorithmus, sobald dieser in Hardware implementiert wird. Ist ein Angreifer in der Lage, Messungen an der ihm vorliegenden Hardware vorzunehmen, besteht die Möglichkeit mit Hilfe von Grundkenntnissen über das Verschlüsselungsverfahren den unbekannten Schlüssel zu ermitteln. Die sogenannte "Timing-Attack" bietet die Möglichkeit über Zeitmessungen den geheimen Schlüssel in relativ kurzer Zeit zu berechnen. Chipkarten stellen eine im E- und M-Commerce, aber auch im privaten Bereich typische Form zur Realisierung von Sicherheit dar. Da der Angriff nicht-invasiv erfolgt, ist ein Attackieren als solches nicht zu erkennen. Am Beispiel des RSA-Algorithmus wurde die Timing-Attack als C-Programm implementiert und die Leistungsfähigkeit des Angriffs mit verschiedenen Test demonstriert.

Einführung

Public-Key-Kryptographie zeichnet sich dadurch aus, dass unterschiedliche Schlüssel zum Ver- und Entschlüsseln benutzt werden. Dabei wird großer Aufwand bei der Implementierung in Chipkarten betrieben, damit der private Schlüssel nicht in Erfahrung gebracht werden kann. Anstatt nun den Algorithmus als solchen zu attackieren, kann man mit Hilfe der sogenannten Timing-Attack einen Angriff auf eine Chipkarten-Implementierung ausführen**.**

1. Kryptographie

1.1. Konzepte der modernen Kryptographie

Für das Verständnis der Kryptographie, sowie der damit verbundenen Terminologie ist es zunächst notwendig einige Begriffe näher zu erläutern. Kryptographie beschäftigt sich mit den Möglichkeiten eine sichere Kommunikation über einen unsicheren Kanal zu ermöglichen. Darunter versteht man die Möglichkeit eines Angreifers Eve die Kommunikation, also das Senden einer Nachricht zwischen zwei Kommunikationspartnern Alice und Bob, in irgendeiner Art und Weise zu belauschen. Für eine sichere Kommunikation zwischen den beiden müssen zumindest folgende Kriterien beachtet werden:

Integrität:

Dieses Kriterium befasst sich mit dem unautorisierten Ändern von Daten. Um die Datenintegrität sicherzustellen, müssen die Kommunikationspartner in der Lage sein, Manipulationen (Einfügen, Löschen, Ersetzen) erkennen zu können.

Authentizität:

Ist eng verwandt mit Identifizierung. Dieses Kriterium ist sowohl für die Kommunikationspartner als auch für die auszutauschenden Daten anzuwenden.

Unleugbarkeit:

Soll verhindern, dass Kommunikationspartner getätigte Handlungen, wie zum Beispiel die Unterzeichnung eines Vertrages, abstreiten können.

Vertraulichkeit:

Dient dazu, den Inhalt von Nachrichten nur autorisierten Kommunikationspartnern zugänglich zu machen. Dies kann zum Beispiel mittels mathematischer Algorithmen erreicht werden.

1.2. Kryptoanalytische Attacken

Passive Attacken:

der Angreifer kann nur den Datenfluss mitverfolgen, hierbei wird also die Vertraulichkeit der Daten verletzt.

Aktive Attacken:

der Angreifer hat auch die Möglichkeit, Daten zu verändern, hierbei werden dann auch die Integrität und die Authentizität verletzt.

Außerdem können die verschiedenen Methoden der Kryptoanalyse anhand der von ihnen benötigten Daten klassifiziert werden:

Ciphertext only attack (Angriff mit der Chiffre):

der Angreifer kennt nur den Schlüsseltext. Eve muss dann einige Annahmen über den Klartext treffen können. Das wäre zum Beispiel die Sprache, in der er verfasst wurde. Ziel ist es, den Klartext oder den Schlüssel k zu finden.

Known plaintext attack (Angriff mit bekanntem Klartext):

der Angreifer kennt sowohl den Klartext, als auch den Schlüsseltext. Das Ziel ist es, k zu ermitteln. Ist der Schlüsselraum klein, kann k durch einfaches Durchprobieren aller Möglichkeiten gefunden werden. Man spricht dann auch von "exhaustive search attacks".

Chosen plaintext (ciphertext) attack (Angriff mit gewähltem Klartext)

die wohl stärkste Attacke geht von der Annahme aus, dass der Angreifer nicht nur den Klartext kennt, sondern diesen auch nach Belieben wählen kann.

2. Kocher Attacken

Bei Kryptosystemen in Hardware - Applikationen, dürfen nicht nur hardwareunabhängige Attacken betrachtet werden. Selbst scheinbar sichere Verschlüsselungsalgorithmen werden anfällig für einen Angriff, sobald Messungen an der zum Algorithmus gehörenden Hardware möglich sind. Damit in Verbindung stehen zwei Angriffe: Die (differentielle) "Power Analysis" und die "Timing-Attack", die beide auf Paul Kocher [1] zurückgehen. Beiden Angriffen liegt die Annahme zugrunde, dass es dem Angreifer möglich ist, das Kryptosystem während des Verschlüsselungsvorgangs zu beobachten.

2.1. Timing-Attacken

Die Idee der Timing-Attacke basiert auf folgender Überlegung: Die Berechnungszeiten zur Verschlüsselung der Eingangsdaten hängen von dem geheimen Schlüssel, also dem Ziel des Angreifers ab. Besteht die Möglichkeit diese Zeiten zu messen, sind unter Anwendung statistischer Methoden Rückschlüsse auf den Schlüssel möglich [1]. Hat der Angreifer auch noch Wissen über den Aufbau des Verschlüsselungsverfahrens, dann kann eine statistische Analyse den geheimen Parameter herausfiltern.

2.2. Angriff auf RSA

Allgemein formuliert arbeitet ein Verschlüsselungsalgorithmus A, indem er mit einer Nachricht m als Input eine Reihe von Rechenschritten (Signatur genannt) ausführt. Dabei wird ein geheimer Schlüssel k verwendet.

Verwendete Bezeichnungen:

M, Menge von Nachrichten m

K, Menge von Schlüsseln k

S, Menge von Signaturen

A: M x $K \rightarrow S\,(m,k)$, die Signatur abhängig von m und dem geheimen Schlüssel k

$B=\{0,1\}$

$T : M\,x\,K \rightarrow \Re : m \rightarrow t=T\,(m,k)$, die Zeit um $A(m,k)$ zu berechnen

$O: M \rightarrow B : m \rightarrow O(m)$, Orakel, basierend auf dem Wissen über den Algorithmus $A(m,k)$ und dessen Implementierung

2.2.1. Modulo – Exponentierung

Die Modulo-Exponentierung (RSA) ist das am meisten verwendete Public-Key-Verfahren. Dabei wird eine Nachricht m unter Einbeziehung des öffentlichen Schlüssels E und des Moduls N zum verschlüsselten Text C , also:

$$C = m^{E} \bmod N .$$

Beim Dekodieren wird statt des öffentlichen der private Schlüssel D benutzt; es ist $m = C^{D} \bmod N$. Beiden Fällen liegt die gleiche mathematische Operation zu Grunde.

Allgemein gilt : $m = B^{K} \; mod \; N$. Die Modulo-Exponentierung lässt sich durch wiederholte Ausführung der Modulo-Multiplikationen ("square and multiply") darstellen:

```
Input:    m,K,N
Output: m^K mod N
1.   Y = m
2.   For i = n-1 DOWNTO 0 DO
3.     Y=Y² mod N
4.     IF ( k_i == 1) Y=Y • m mod N
5.   END FOR
6.   RETURN Y
```

Algorithmus 1: Square and Multiply

Wobei m eine Nachricht aus einer Menge von Nachrichten und k_i der Wert des Schlüssels an der entsprechenden Bitposition ist Die Berechnungszeiten zur Verschlüsselung der Eingangsdaten sind damit vom gesuchten Schlüssel abhängig. Besteht die Möglichkeit diese Zeit zu messen, sind Rückschlüsse auf den Schlüssel möglich. Dabei wird davon ausgegangen, dass die Berechnungszeit für die Modulo-Multiplikation annähernd konstant ist und nur um die Zeit einer innerhalb der Multiplikation eventuell auszuführenden Korrekturaddition variiert. Diese, als Reduktion

bezeichnete Addition, ist nötig, da das Ergebnis nach dem Reduktionsalgorithmus in Abhängigkeit des Verfahrens entweder im Bereich]-*N* ... *N*[bzw. [*0* ... *2N*[liegt.

2.2.2. Angriff auf die Quadrierung

In [1] und [2] wird der Multiplikationsschritt von *Algorithmus 1* angegriffen. Die Nachrichtenmengen werden in zwei Untermengen aufgeteilt, abhängig davon ob eine Reduktion während des Multiplikationsschrittes stattfinden würde (Menge *M1*) oder nicht (Menge *M2*). Ist das Schlüsselbit an dieser Position gesetzt, werden die Zeiten in *M1* die zusätzliche Reduktionszeit sowie die restliche Rechenzeit beinhalten. Ist das Schlüsselbit null, dann fehlt die Reduktionszeit, da in diesem Fall auch die Multiplikation nicht ausgeführt wird. *M2* wird dagegen in beiden Fällen die Reduktionszeit nicht enthalten. Die Schwierigkeit dieses Verfahrens besteht darin, zu entscheiden ob die Aufteilung in zwei Untermengen zufällig war, also die Zeiten für die Menge *M1* im Mittel ungefähr gleich denen von *M2* sind (Schlüsselbit = 0). Im anderen Fall sollte *M1* wesentlich größere Zeiten aufweisen als *M2*. Problematisch ist es zu entscheiden, was ist "gleich" und was "wesentlich größer".

Eine Attackierung der Quadrierung bietet den Vorteil, dass nicht mehr entschieden werden muss ob die Aufteilung sinnvoll ist, sondern welche signifikanter ist. Um nun das i-te Bit des Schlüssels zu berechnen, vorausgesetzt die vorherigen i-1 Bits sind bekannt, werden die i-1 Schritte des "Square and Multiply" und die Quadrierung für Bit i ausgeführt. Angenommen k_i ist 1, dann sind die nächsten beiden Operationen die Multiplikation für Bit i und die Quadrierung für Bit i+1. Die Multiplikation wird ausgeführt und für die Quadrierung entschieden, ob eine Reduktion stattfinden würde oder nicht. Dies wird für jede Nachricht durchgeführt und die gesamte Nachrichtenmenge in zwei Untermengen aufgeteilt.

Menge *M1*: alle Nachrichten bei denen eine Reduktion,

Menge *M2*: alle Nachrichten bei denen keine Reduktion nötig wäre.

Angenommen k_i ist 0, dann würde keine Multiplikation folgen und der nächste Rechenschritt wäre die Quadrierung. Wieder werden die Nachrichten aufgeteilt in *M3* und *M4*, abhängig davon ob eine Reduktion nötig wäre oder nicht.

Natürlich macht nur eine dieser Aufteilungen Sinn, deshalb werden diese verglichen. Wenn der Zeitunterschied zwischen *M1* und *M2* größer als der zwischen *M3* und *M4* ist, wird entschieden, dass k_i =1 ist, sonst ist k_i = 0. Getestet wird auf die Mittelwerte dieser Mengen, da in jeder Menge zwar garantiert die zur aktuellen Bitstelle korrespondierenden Reduktions- und Quadrierungszeiten enthalten sind, aber selbstverständlich auch die Zeiten für die restlichen Bitstellen. Diese restlichen Zeiten sind abhängig von der gesendeten unverschlüsselten Nachricht, denn davon abhängig ist es, ob bei gesetztem Bit in der Quadrierung eine Reduktion stattfindet oder nicht. Im Mittelwert sollten sich also diese nicht zur aktuellen Position gehörenden Reduktionszeiten nahezu aufheben.

Oder formal:

Für den Angriff auf Bit i wird der Algorithmus *A(m,k)* in *L(m,k)* und *R(m,k)* aufgeteilt, wobei *L(m,k)*, die Berechnung mit Reduktion und R(m,k) die restlichen Berechnungen darstellt.

- berechnen von : $m_{temp} = (m^k)^2$ mit $k=k_{n-1} \ldots k_2, k_1$
- definieren von 2 Orakeln:

$$O_1{:}m \rightarrow \begin{cases} 1\text{, wenn } (\text{mtemp} \cdot \text{m})^2 \text{ mit Reduktion ausgeführt} \\ 0\text{, wenn } (\text{mtemp} \cdot \text{m})^2 \text{ ohne Reduktion ausgeführt} \end{cases}$$

$$O_2{:}m \rightarrow \begin{cases} 1\text{, wenn } (\text{mtemp})^2 \text{ mit Reduktion ausgeführt} \\ 0\text{, wenn } (\text{mtemp})^2 \text{ ohne Reduktion ausgeführt} \end{cases}$$

definieren der Untermengen:

$M1 = \{ m \in M: O_1(m)=1\}$

$M2 = \{ m \in M: O_1(m)=0\}$

$M3 = \{ m \in M: O_2(m)=1\}$

$M4 = \{ m \in M: O_2(m)=0\}$

$F_k : M_k \rightarrow \Re : m \rightarrow F_k(m)=T(m)$ für $1 \leq k \leq 4$

- wenn k_i=1, dann :

$F1=T^R+T^L$

$F2= T^R$

$F3=F4$

-und damit : $\overline{F1} > \overline{F2}, \overline{F3} = \overline{F4}$

- wenn k_i=0, dann :

$F1=F2$

$F3= T^R+T^L$

$F4 =T^R$

-und damit : $\overline{F1} = \overline{F2}, \overline{F3} > \overline{F4}$

Nur das letzte Bit des Schlüssels kann auf diese Weise nicht berechnet werden, denn vom Schritt für k_0 aus gibt es keine folgende Quadrierung. Bit k_0 muss also geraten oder mittels einer Berechnung ermittelt werden. Zum Beispiel ist ein Test möglich, ob mit k_0 = 1 die ursprüngliche kodierte Nachricht mit der selbstberechneten übereinstimmt .In diesem Fall sollte der berechnete Schlüssel korrekt sein.

2.2.3. Tests

Das beschriebene Angriffsverfahren wurde als C-Programm implementiert und getestet.

Signifikanztest:

Eine Schwierigkeit der Implementierung stellte der Test auf Gleichheit der gefundenen Mittelwerte dar. In einer ersten Variante des Programms wurden dafür die beiden Mengen von einander subtrahiert und anschließend der Betrag der Subtraktion darauf getestet einen bestimmten Wert nicht zu überschreiten. Diese Zahl wurde in Versuchsreihen mit mehreren Schlüsseln ermittelt. Ein Wert von 0.45 zeigte gute Ergebnisse. Mit einer zweiten Methode, welche die Tatsache ausnutzt, dass immer eine der Aufteilungen signifikanter ist (*Diagramm* 1), konnten die Ergebnisse teilweise noch verbessert werden. Das heißt, sowohl die Anzahl der benötigten Nachrichten für das Erkennen eines Schlüssels sowie die damit verbundene Berechnungsdauer konnten weiter gesenkt werden (*Diagramm 2* und 3).

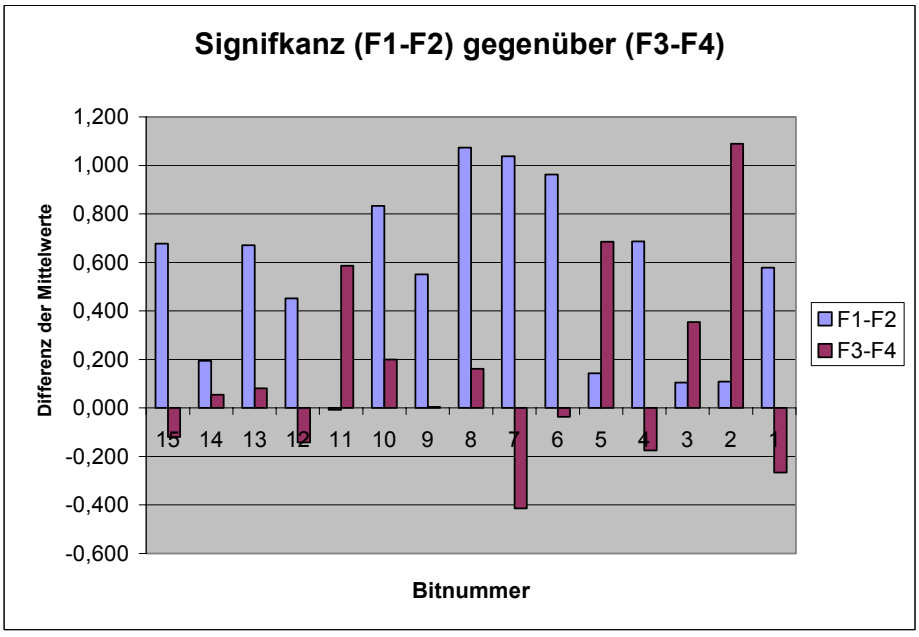

Diagramm 1

In den Diagrammen 2 und 3 ist für je fünf Schlüssel die für eine richtige Erkennung notwendige Anzahl der Nachrichten dargestellt. Zu erkennen ist, dass mit der zweiten Methode genauso viel bzw. oft noch weniger Nachrichten benötigt werden.

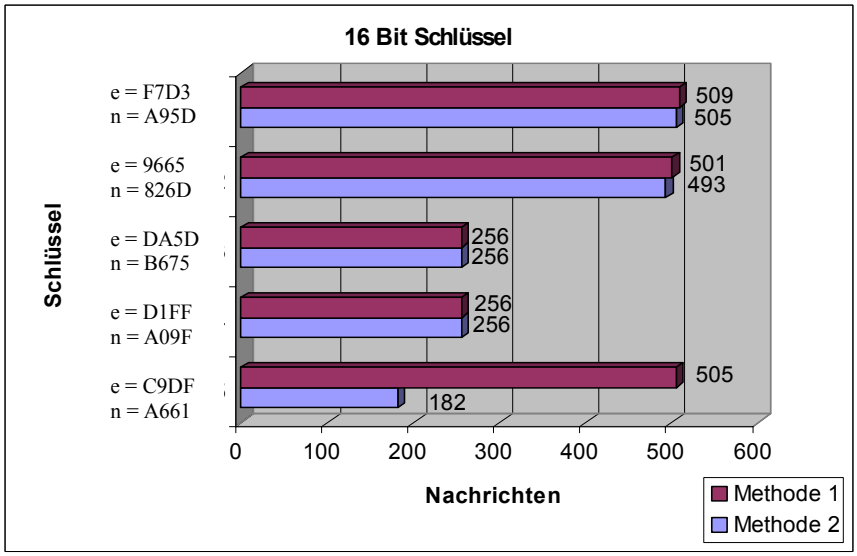

Diagramm 2

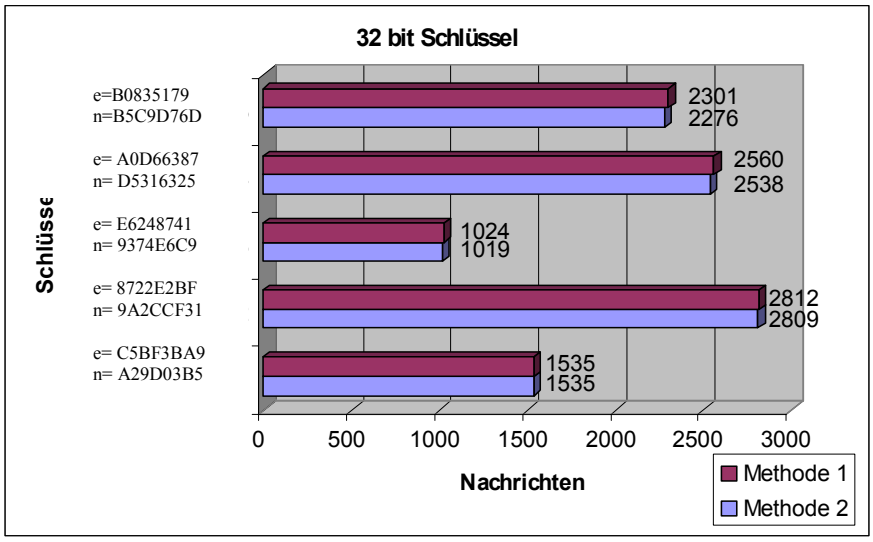

Diagramm 3

- Nachrichtenverteilung:

In allen bisherigen Tests wurden die Nachrichten gleichmäßig über den zur Verfügung stehenden Zahlenraum verteilt, in der Praxis ergeben sich, z.B. durch Protokolle, gewisse Einschränkungen. Deshalb wäre es interessant zu wissen, ob es auch bei ungleicher Verteilung möglich ist den Schlüssel zu ermitteln. Wie im Diagramm 4 zu erkennen funktioniert das Verfahren auch dann und teilweise sogar ohne die Nachrichtenanzahl zu erhöhen. Die Achse Prozent ist so zu verstehen, dass die Nachrichten im Bereich von 0 bis x Prozent gleichmäßig verteilt wurden. Selbst unterhalb 10% des theoretisch möglichen "Nachrichtenraums" ist es also möglich Schlüssel ohne Erhöhung der Nachrichtenmenge zu erkennen.

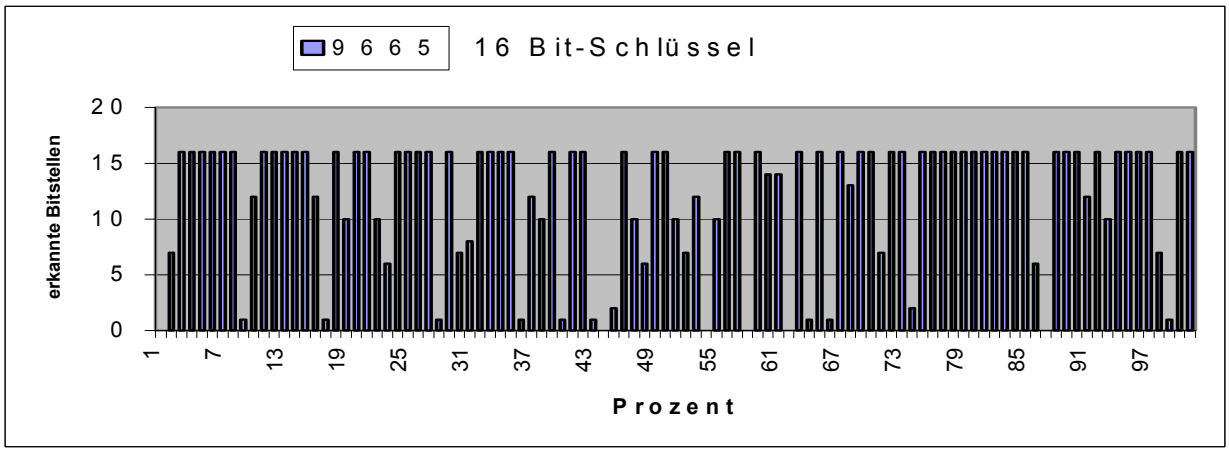

Diagramm 4

- Verhältnis Multiplikationszeit zu Reduktionszeit:

Das verwendete Verfahren basiert auf dem durch den Reduktionsschritt hervorgerufenen Zeitunterschied. Bei einer Implementierung des Verfahrens ist jedoch das Verhältnis zwischen Multiplikationszeit Tmul und eventuellem zusätzlichen Reduktionsschritt Tred dem Angreifer im allgemeinen nicht bekannt. Tatsächlich verhindert das angewandte Verfahren mit dieser Unkenntnis eventuell verbundene Schwierigkeiten. Durch die Subtraktionen der Mengen F1-F2 sowie F3-F4 ergeben sich als Ergebnis immer nur die Unterschiede in den Mengen, so das sich für verschiedene Tmul/Tred – Verhältnisse immer konstante Resultate ergeben (siehe Diagramme 5 –7). Die Differenz bleibt also konstant, die Kenntnis von Tmul / Tred ist deshalb nicht erforderlich.

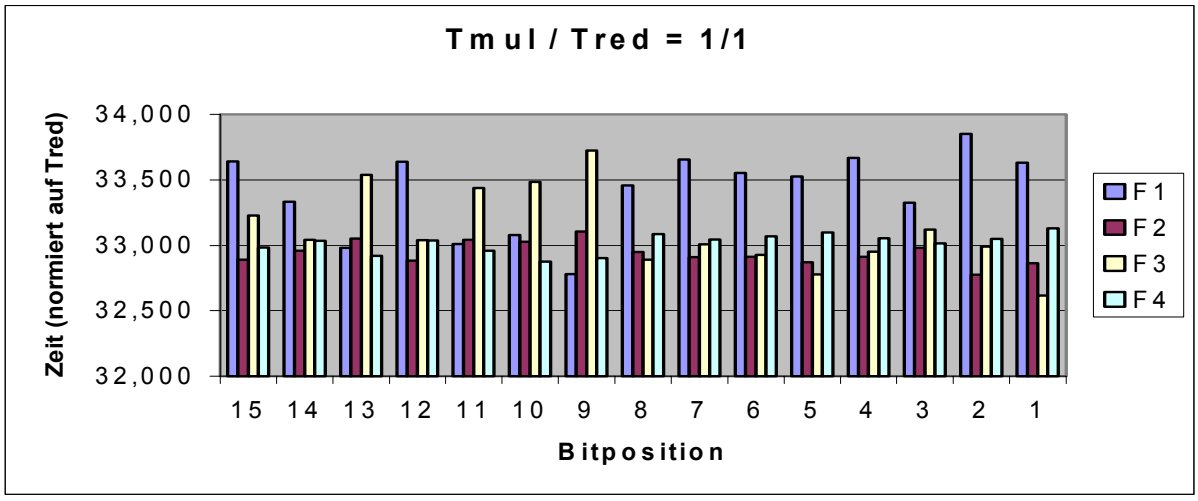

Diagramm 5

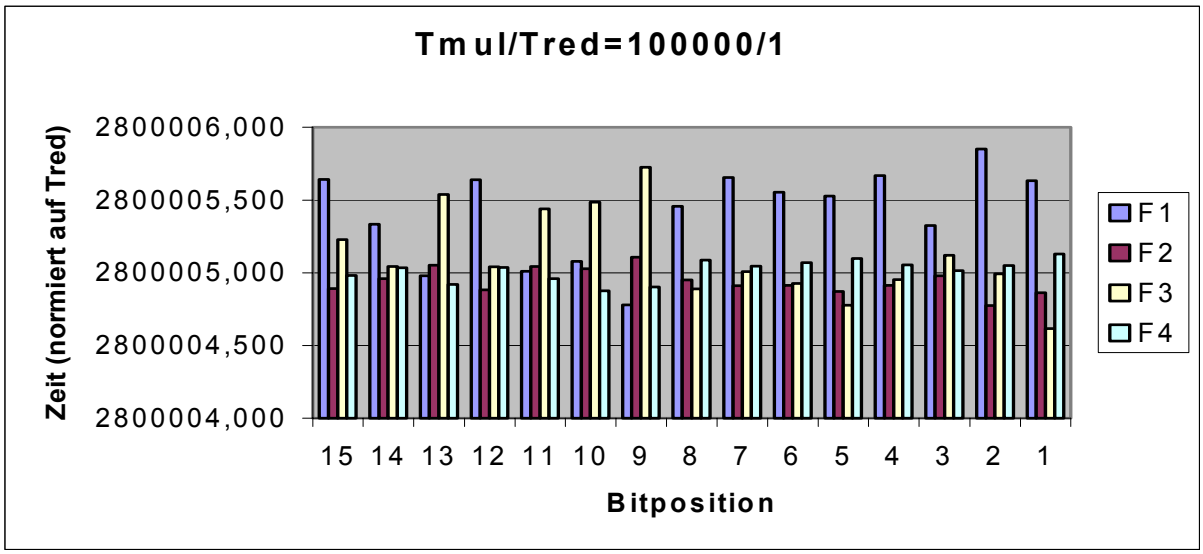

Diagramm 6

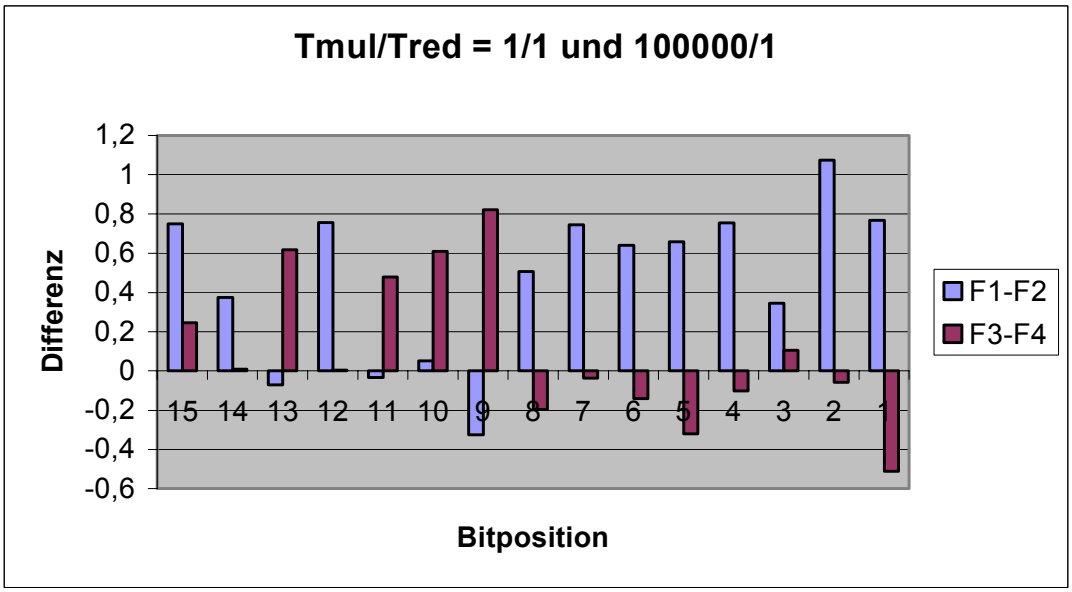

Diagramm 7

2.2.4. Gegenmaßnahmen

Eine offensichtliche Möglichkeit der Abwehr von Timing Attacken ist es, alle Operationen exakt gleich lang zu gestalten. In der Praxis ist das bei Softwareimplementierungen kaum möglich, denn verschiedene Betriebssysteme sind zu berücksichtigen, verschiedene Compiler optimieren Programmcode unterschiedlich, oder RAM Zugriffszeiten sind unterschiedlich.

Bei Hardwareimplementierung, wie auf Smartcards, ist dies sicher schwierig, aber auf jeden Fall würden dadurch die Algorithmen langsamer ablaufen, da jede Optimierung auch eine Angriffsmöglichkeit darstellt.

Eine andere Variante wären zufällige Pausen in die Abarbeitung des Algorithmus einzubauen. Dadurch könnte eine genaue Zeitmessung verhindert werden, und der Angreifer wäre dazu gezwungen, mehr Messungen vorzunehmen.

Tests mit zufällig eingestreuten Operationen zeigten, dass es durchaus noch möglich ist den richtigen Schlüssel zu finden. Dies zu überprüfen war aber nur möglich weil im Programm der gesuchte Schlüssel ja eigentlich schon bekannt war. In der Praxis wäre die einzige Möglichkeit einigermaßen sicher zu sein, den richtigen Schlüssel gefunden zu haben, die Werteanzahl zu steigern. Zeigt dann das Verfahren mehrmals dasselbe Ergebnis, sollte es sich um den richtigen Schlüssel handeln. Das Einstreuen von zufälligen Operationen verhindert jedoch diese Möglichkeit. In dem gezeigten Beispiel wurde die Reduktionszeit bei jedem hundersten Wert im Bereich von 0 bis 10 Prozent der Multiplikationszeit variiert.

Ohne zufällige Operationen funktioniert die Erkennung ab einer Mindestanzahl von Werten mit jeder Steigerung der Wertezahl, mit eingestreuten Rechenschritten tritt dies nicht auf. (Die Achse Übereinstimmung in Bitstellen ist als Übereinstimmung von höchsten Bit aus gesehen zu verstehen, die Zählung wurde mit der ersten Abweichung abgebrochen).

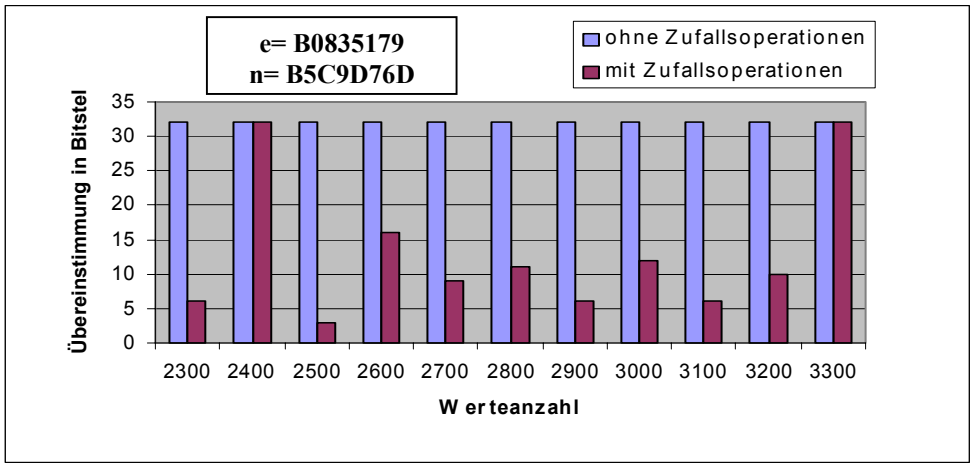

Diagramm 8

Ein Ansatz, zufällige Operationen einzusetzen, ergibt sich aus den Gesetzen der modularen Mathematik. Es gilt:

$$A \bmod N = (A + z \bullet N) \bmod N = (A + 2^x \bullet N) \bmod N$$

Es müssen lediglich Vielfache des Moduls N addiert werden. Leicht zu implementieren sind Vielfache zur Basis 2, die sich durch Linksschieben des Moduls erzeugen lassen. Dadurch wird das Ergebnis der Berechnung nicht verändert. Die Zeit hingegen, die für die Signatur benötigt wird, variiert und lässt somit keine Schlussfolgerungen mehr auf den Schlüssel zu. Um die Sicherheit zu erhöhen, erscheint es sinnvoll zufällig ausgewählte Vielfache zu addieren, was z.B. durch rückgekoppelte Schieberegister erfolgen kann.

3. Zusammenfassung

Es lässt sich also demonstrieren, dass mit einem relativ einfachen Angriff der geheime Schlüssel in Erfahrung gebracht werden kann, ohne dabei die Hardware der Chipkarte zu zerstören. Bei der Implementierung von Public-Key-Kryptosystemen muss deshalb auch dieser Aspekt immer Berücksichtigung finden.

Literatur

[1] P.C. Kocher,
"Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems",
Advances in Cryptology - CRYPTO '96, LNCS 1109, 1996

[2] Dehm et al,
"A practical implementation of the timing attack",
Crypto Group Technical Report Series CG,1998

[3] T.S. Messerges,
"Investigations of power analysis attacks on smartcards",
USENIX Workshop on Smartcard Technology, 1999

[4] Elisabeth Oswald,
"Analyse der Anwendung von DPA auf DES Bausteine"
Graz, 1999

[5] Andreas Ahrens,
"Sicherheit von Public-Key Implementierungen auf Smart Cards gegen Timing-Angriffe"
Kleiner Beleg, Institut MD , FB Elektrotechnik und Informationstechnik , Universität Rostock, 2001