# Implementierung Elliptischer Kurven auf Chipkarten

Detlef Hühnlein
secunet Security Networks GmbH
Mergenthalerallee 79-81
D-65760 Eschborn
huehnlein@secunet.de

## Zusammenfassung

Kryptosysteme auf Basis Elliptische Kurven über endlichen Körpern erfreuen sich wachsender Beliebtheit. So sieht beispielsweise das BSI im Anforderungskatalog für das Signaturgesetz u.a. auch die Verwendung Elliptischer Kurven vor. In dieser Arbeit soll die Implementierung Elliptischer Kurven über *GF(p)* auf Chipkarten diskutiert werden. Neben der offensichtlich möglichen Implementierung auf *Karten mit kryptographischem Coprozessor*, werden auch andere Ansätze betrachtet. So wird auch die Möglichkeit der Implementierung auf *Karten ohne kryptographischem Coprozessor* und *Java-Karten* beleuchtet.

# 1 Einleitung

Kryptosysteme auf Basis Elliptischer Kurven über endlichen Körpern, wie in **[26]** und **[16]** vorgeschlagen und momentan in **[14]** standardisiert, bieten gegenüber den heute gebräuchlicheren Verschlüsselungs- und Signatursystemen, wie RSA **[34]** , ElGamal **[6]** und DSA **[30]**, den Vorteil, daß die zu verwendenden Parameter bei vergleichbarer Sicherheit deutlich kleiner gewählt werden dürfen und deshalb eine effizientere Implementierung ermöglichen.

Typischerweise hat heute z.B. ein RSA-Modulus eine Länge von 768 oder gar 1024 Bit. Man erwartet, daß die Faktorisierung (siehe z.B. **[2]** und **[5]**) von 512 Bit Moduli in wenigen Jahren mehr oder minder 'problemlos' möglich sein wird. Auch das Berechnen diskreter Logarithmen in *GF(p)** (siehe z.B. **[10]**, **[36]** ) wird für 512 Bit Primzahlen *p* in wenigen Jahren leicht möglich sein. In beiden Fällen arbeitet man mit dem sog. Zahlkörpersieb **[20]**. Der Algorithmus besitzt eine subexponentielle Laufzeit

$$O\left(\exp(1.9223\,(\log m)^{1/3}(\log\log m)^{2/3}\right)$$

wobei *m* der Modulus *n* bzw. die Primzahl *p* ist. Deshalb schlägt das BSI in **[3]** für Signaturen mit einer Lebensdauer von sechs Jahren die Verwendung von mindestens 1024 Bit Moduli bzw. Primzahlen vor.

Dagegen ist heute *kein Algorithmus subexponentieller Laufzeit* bekannt, der diskrete Logarithmen in der Punktegruppe Elliptischer Kurven berechnet. Die besten Algorithmen hierfür (d.h. Shank's Baby-Step-Giant-Step, Pollard-Rho und Pohlig-Hellman, **[25]**, S. 103 ff.) besitzen exponentielle Laufzeit. Deshalb schlägt das BSI für Signaturen die Verwendung

Elliptischer Kurven über *GF(q)* vor, wobei $q$ entweder eine Primzahl mit $\log_2 q \geq 160$ oder $q=2^m$
mit $m \geq 160$ ist. Das bedeutet, daß mit einer 160-Bit Körperarithmetik gearbeitet werden kann, was eine vergleichsweise effiziente Implementierung ermöglicht. Die Arithmetik für Körper der Charakteristik 2 kann in Hardware sehr effizient implementiert werden. Eine reine Software-Implementierung dieser Arithmetik (siehe **[21]**) erreicht eine ähnliche Effizienz, wie sie von gewöhnlichen 'Langzahlarithmetiken' her bekannt ist durch den speicherplatzintensiven Einsatz von vorausberechneten Tabellen bestimmter Elemente, z.B. eines Teilkörpers, wie in **[44]** und **[11]**. Ob bei einer Software-Implementierung auf Chipkarten auf diese speicherplatzintensiven Tabellen verzichtet werden kann muß näher untersucht werden.

In dieser Arbeit wollen wir in Kapitel 2 die nötigen *Grundlagen Elliptischer Kurven* wiederholen. In Kapitel 3 werden wir grob auf verschiedene *Karten-Arten* eingehen, die möglicherweise für die Implementierung von Kryptosystemen auf Basis Elliptischer Kurven in Frage kommen. Schließlich wollen wir in Kapitel 4 *implementierungstechnische Aspekte* beleuchten und die Implementierung mit den heute weiter verbreiteten Verfahren RSA und DSA vergleichen.

# 2 Elliptische Kurven

In dieser Arbeit wollen wir uns auf Kurven über *GF(p)* beschränken. Für die Diskussion von Kurven über $GF(2^n)$ sei auf **[23]** verwiesen.

Eine *Elliptische Kurve E(K)* über einem Körper $K$ ($char(K) \neq 2,3$) ist definiert als die Menge aller Punkte $P=(x,y)$ der Ebene $K \times K$, die die Gleichung

$$F(x,y): y^2 = x^3 + ax + b \qquad (1)$$

erfüllen und kein Punkt ein singulärer Punkt ist, d.h.

$$4a^3 + 27b^2 \neq 0 \qquad (2)$$

Die Forderung (2) stellt übrigens sicher, daß keine Knoten (Abbildung 1[1]) oder Spitzen (Abbildung 2) auftreten. Es läßt sich nämlich zeigen (z.B. **[13]**, S. 77ff), daß die

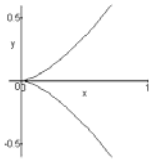


*Abbildung 2: $E_2: y^2=x^3$*

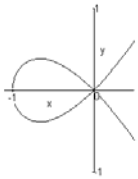


*Abbildung 1: $E_1: y^2=x^3+x^2$*

Punktegruppe einer singulären Kurve über
einem endlichen Körper *GF(p)* bei einem Knoten isomorph zur multiplikativen Gruppe *GF(p)**, bzw. bei einer Spitze sogar isomorph zur additiven (!) Gruppe $GF(p)^+$ des zugrundeliegenden Körpers ist.

[1] Aus Gründen der Anschaulichkeit sind alle abgebildeten Kurven über den reellen Zahlen definiert.

Zusätzlich zu den so definierten Punkten nimmt man den Punkt $O = (\infty,\infty)$ im Unendlichen zur Elliptischen Kurve hinzu.

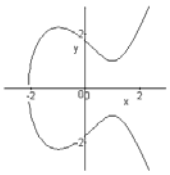


*Abbildung 3:$E_3$:$y^2=x^3-3x+3$*

Beispiele für Elliptische Kurven über den reellen Zahlen sehen wir im folgenden (Abbildung 3 und Abbildung 4). Der Punkt $O=(\infty,\infty)$ im Unendlichen wäre hier anschaulich die unendliche Verlängerung der rechten Kurvenäste.

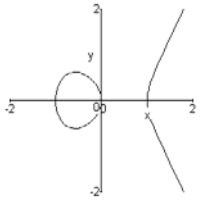


*Abbildung 4 :$E_4$: $y^2=x^3-x$*

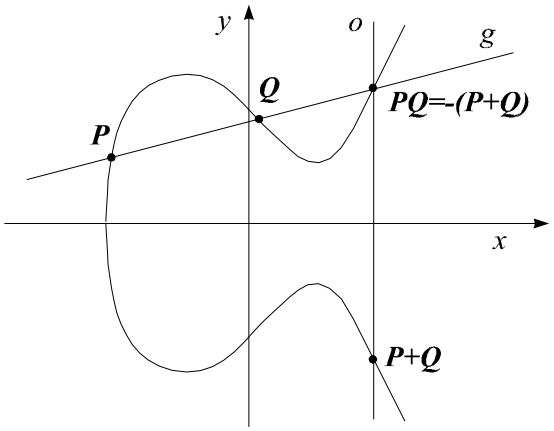



*Abbildung 5 : Die Gruppenverknüpfung '+'*

Da sich auf diesen Punkten eine Gruppenverknüpfung definieren läßt, sind Elliptischen Kurven für die praktische Kryptographie interessant. Die als 'Sehnen-Tangenten-Regel' bekannte Gruppenver-knüpfung sei an Abbildung 5 veranschaulicht.

$P+Q$ berechnet sich als der an der $x$-Achse gespiegelte weitere Schnittpunkt $PQ$ der Gerade $g$ (Sehne) durch $P$ und $Q$. Eine Punktverdopplung $2P$ wird durch anlegen der Tangente an $P$ erreicht. Das neutrale Element der Gruppe ist der Punkt $O=(\infty,\infty)$ im Unendlichen. Daraus ist klar, daß Invertieren einer Spiegelung an der $x$-Achse entspricht. Der Nachweis, daß es sich in der Tat um eine Gruppe handelt kann rein geometrisch geführt werden. Die folgenden Formeln für die Gruppenverknüpfung entstehen durch Berechnung der Steigung $\lambda$ der Geraden $g$.

Es sei $E:y^2=x^3+ax+b$ eine Elliptische Kurve über einem Körper mit $char(K)\neq 2,3$ und $P,Q,R \in E$, $R(x_3,y_3):=P(x_1,y_1)+Q(x_2,y_2)$, $P,Q\neq O$ und $P\neq -Q$. Dann erhalten wir für

$$x_3 = \lambda^2 - x_1 - x_2$$

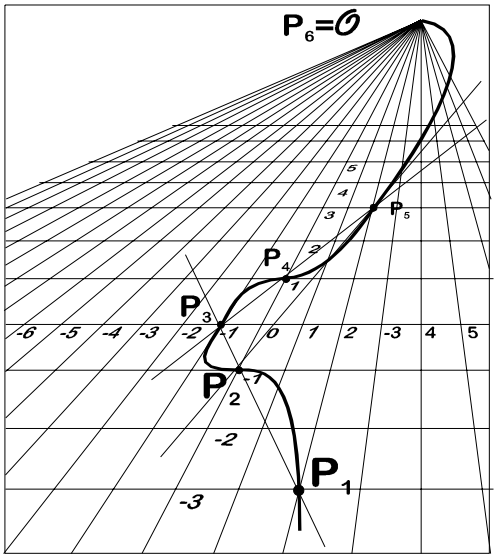
$$y_3 = -y_1 + \lambda(x_1 - x_3)$$

$$wobei \qquad \lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} \; f\ddot{u}r P \neq Q \\ \dfrac{3x_1^{\,2} + a}{2y_1} \; f\ddot{u}r P = Q \end{cases} \qquad (3)$$

Daraus sehen wir, daß die Addition zweier Punkte mit *drei Multiplikationen* und *einer Inversion (mod p)* möglich ist. Das Verdoppeln eines Punktes benötigt *vier Multiplikationen* und wiederum *eine Inversion (mod p)*. Hier sei angemerkt, daß die Addition *(mod p)* und die Multiplikation mit kleinen Konstanten vernachlässigt wurde.

Für das Inverse *-P* des Punktes $P(x_1, y_1)$ erhält man einfach durch Negieren der y-Koordinate; d.h.
$-P(x_1,-y_1)$. Addition des neutralen Elementes *O* ändert nichts; d.h. $P+O=P$.

Diese Formeln sind sowohl für Kurven über den reellen Zahlen, als auch für Kurven über großen Primkörpern gültig. Der 'Übergang' von der reellen Zahlen zu *GF(p)* und die Rolle des Punktes im Unendlichen sei an folgender Abbildung 6 veranschaulicht.




In dieser Abbildung sehen wir eine Elliptische Kurve $E_5: y^2=x^3+1$ über den reellen Zahlen (durchgezogene Linie). Das 'nette' an dieser Kurve ist nun, daß die ganzzahligen Kurvenpunkte genau die Punkte der Kurve über dem, für kryptographische Anwendungen zugegebenermaßen etwas kleingeratenen, endlichen Körper *GF(5)* sind. Die Kurve über *GF(5)* hat die sechs Punkte $P_1(2,-3)$, $P_2(0,-1)$, $P_3(-1,0)$, $P_4(0,1)$, $P_5(2,3)$ und $P_6(\infty,\infty)=O$. Es ist sofort zu sehen, daß $P_1=-P_5$, $P_2=-P_4$ und $P_3=-P_3$. Außerdem ist die Rolle des neutralen Elementes *O* gut ersichtlich. Schließlich sieht man, daß $2P_1=P_2$, $P_2+P_1=P_3$, $P_3+P_1=P_4$, $P_4+P_1=P_5$ und $P_5+P_1=P_6=O$. $P_1$ hat also Ordnung 5 und ist ein primitives Element der Punktegruppe.

*Abbildung 6: $E_5: y^2=x^3+1$ in der Ebene*

Für den allgemeinen Fall einer Elliptischen Kurve über *GF(p)* ist mit dem *Satz von Hasse* (siehe z.B. **[13]**, S. 243) bekannt, daß für die Anzahl der Punkte *N* gilt

$$N = p + 1 - t, \quad mit \quad |t| \leq 2\sqrt{p}\,. \qquad (4)$$

In der Arbeit von Waterhouse **[43]** wurde Hasses Resultat etwas präzisiert. Er zeigt u.a., daß alle (natürlich ganzzahligen) Werte *t, ggT(t,p)=1* in diesem Intervall auch auftreten. Außerdem gibt es sog. supersingulären Kurven mit Spur *t=0*. Eine Elliptische Kurve ist nach **[40]**, Theorem 4.1, Seite 140 genau dann supersingulär, wenn ihre definierende Gleichung die Form

$$y^2 = x^3 + ax, \quad mit \quad p \equiv 3 \bmod 4 \qquad (5)$$

oder

$$y^2 = x^3 + b, \quad mit \quad p \equiv 2 \bmod 4 \qquad (6)$$

hat. Die Tatsache, daß die Anzahl der Punkte ($N=p+1$) bekannt ist, verschaffte den supersinglulären Elliptischen Kurven zumindest in den frühen Jahren ihre Beliebtheit.

Allerdings wurde in **[24]** gezeigt, daß man Elliptische Kurven über *GF(p)* mit Hilfe der sog. Weil-Paarung in einen zugehörigen Erweiterungskörper $GF(p^k)$ einbetten kann. Ausgenutzt wird bei dieser Einbettung nur, daß die Punkteanzahl der Gruppe *N* die Gruppenordnung von $GF(p^k)$ teilt, also $N \mid (p^k - 1)$. Die Berechnung diskreter Logarithmen erfolgt nach dieser Transformation nicht in der Punktegruppe sondern in der multiplikativen Gruppe $GF(p^k)^*$ mit subexponentiellen Algorithmen, wie dem Zahlkörpersieb **[36]**. Nun ist dieser Angriff nicht das Ende aller Kryptosysteme auf Basis Elliptischer Kurven, da der Grad der Körpererweiterung *k* im allgemeinen sehr groß ist. Allerdings ist klar, daß supersinguläre Kurven über *GF(p)* nicht mehr verwendet werden dürfen, da diese Kurven bereits in den Erweiterungskörper $GF(p^2)$ eingebettet werden können, da $p^2-1=(p+1)(p-1)$.

Kürzlich wurde von N. Smart gezeigt, daß auch Kurven mit Spur $t=1$ nicht mehr verwendet werden dürfen, da bei diesen Kurven die Gruppenordnung gleich *p* ist und diskrete Logarithmen mittels *p*-adischer Logarithmen in polynomieller Zeit berechnet werden können.

Welche Kurven für kryptographische Zwecke geeignet sind und wie man solche erhält wurde in **[28]** ausführlich diskutiert.

# 3 Chipkarten

In diesem Kapitel wollen wir etwas näher auf Chipkarten eingehen, die für die Implementierung Elliptischer Kurven potentiell in Frage kommen. Eine umfassende Spezifikation von Chipkarten mit Kontakten ist mit der Normenreihe ISO7816 **[15]** gegeben. Dort sind z.B. die elektrischen und mechanischen Eigenschaften, sowie die Struktur von Datenelementen festgelegt. Da Ziel der Implementierung in jedem Fall sein soll, daß die kryptographischen Protokolle selbst auf der Karte ablaufen, kommen natürlich keine sog. Speicherkarten, wie z.B. Karten mit Chips der Siemens SLE44XY-Reihe in Frage. Diese Karten werden z.B. für Telefon- und Krankenversicherungskarten eingesetzt. Die grundsätzliche Spezifizierung eines Chipkartenbetriebsystems ist in ISO7816-4 geschehen. Eines der ersten praktischen Betriebsysteme war das von Giesecke & Devrient und der GMD entwickelte STARCOS, das recht ausführlich z.B. in **[33]** beschrieben ist. Allerdings sieht STARCOS aus historischen Gründen (noch) nicht die Verwendung von Public-Key-Algorithmen vor. Ein anderes Chipkarten-Betriebssystem, das auch die Verwendung von Public-Key-Algorithmen unterstützt ist z.B. das von der Telesec entwickelte TCOS Betriebssystem **[42]**. Als Industriestandard für Chipkartenleser darf das B1-Konzept **[41]** angesehen werden. In dieser Arbeit wollen wir uns nun auf die verschieden Prozessortypen für eine mögliche Implementierung Elliptischer Kurven konzentrieren. Die meisten verfügbaren Prozessoren für Chipkarten (siehe **[29]**) basieren im Kern entweder auf den CISC-Intel-Chip 8051 oder den RISC-Chip 68HCO5 der Firma Motorola. Wir teilen die verfügbaren Prozessor-Karten in drei Kategorien ein: ***Chipkarten mit kryptographischem Coprozessor***, ***Chipkarten ohne*** einen solchen ***Coprozessor*** und ***Java-Interpreter-Karten***.

## 3.1 Chipkarten mit Coprozessor

Bei diesen Karten steht zusätzlich zur normalen CPU ein arithmetischer Coprozessor zu Verfügung, der die Berechnung der modularen Multiplikation bzw. Exponentiation übernimmt. Da selbst die Implementierung von 512-Bit RSA nicht mit akzeptabler Performance ohne diese Coprozessoren möglich ist, sind alle heute verfügbare RSA-Karten mit einem solchen Coprozessor ausgestattet. In der folgenden Tabelle wollen wir die wichtigsten Daten der gängigsten Prozessoren auflisten. Für eine umfassendere Darstellung sei an **[29]** und Herstellerdatenblätter, wie **[39]**, **[27]** oder **[31]** verwiesen.

| Hersteller | Kern | Name | ROM in kByte | RAM in Byte | EEPROM in kByte | Max. Takt in MHz |
|---|---|---|---|---|---|---|
| **Motorola** | 68HC05 | SC29 | 13 | 512 | 4 | 5 |
| **Motorola** | 68HC05 | SC49 | 20 | 896 | 4 | 5 |
| **Motorola** | 68HC05 | SC50 | 20 | 896 | 8 | 5 |
| **Philips** | 8051 | P83C852 | 6 | 256 | 2 | 6 |
| **Philips** | 8051 | P83C855 | 20 | 512 | 2 | 8 |
| **Philips** | 8051 | P83C858 | 20 | 640 | 8 | 8 |
| **Philips** | 8051 | P83W858 | 20 | 672 | 8 | 8 |
| **Siemens** | 8051 | SLE44C200 | 9 | 256 | 9 | 5 |
| **Siemens** | 8051 | SL44CR80S | 14 | 706 | 8 | 5 |
| **SGS** | 68HC05 | ST16CF54 | 16 | 352 | 16 | 5 |
| **SGS** | 68HC05 | ST16KL74 | 20 | 608 | 20 | 5 |

*Tabelle 1 :Karten mit Coprozessor*

## 3.2 Chipkarten ohne Coprozessor

Da die Karten mit ohne Coprozessoren im allgemeinen etwa für den halben Preis zu haben sind, wie Karten mit Coprozessoren, wäre eine mögliche Implementierung von Elliptischen Kurven auf gewöhnlichen Karten natürlich besonders interessant. Auch hier sei für eine umfassendere Darstellung auf **[29]**, **[27]**, **[31]** und **[39]** verwiesen.

| Hersteller | Kern | Name | ROM in kByte | RAM in Byte | EEPROM in kByte | Max. Takt in MHz |
|---|---|---|---|---|---|---|
| **Motorola** | 68HC05 | SC01 | 1,6 | 39 | 1 | 4 |
| **Motorola** | 68HC05 | SC03 | 2 | 52 | 2 | 4 |
| **Motorola** | 68HC05 | SC11 | 6 | 128 | 8 | 4 |
| **Motorola** | 68HC05 | SC21 | 6 | 128 | 3 | 5 |
| **Motorola** | 68HC05 | SC24 | 3 | 128 | 1 | 5 |
| **Motorola** | 68HC05 | SC26 | 6 | 224 | 1 | 5 |
| **Motorola** | 68HC05 | SC27 | 16 | 240 | 3 | 5 |
| **Motorola** | 68HC05 | SC28 | 12,8 | 240 | 8 | 5 |
| **SGS** | 68HC05 | ST1821 | 2 | 44 | 1 | 5 |
| **SGS** | 8048 | ST1834 | 4 | 76 | 3 | 5 |
| **SGS** | 8048 | ST16612 | 6 | 224 | 2 | 5 |
| **SGS** | 68HC05 | ST16F48 | 6 | 224 | 3 | 5 |
| **SGS** | 68HC05 | ST16CF54 | 16 | 512 | 8 | 5 |
| **SGS** | 68HC05 | ST16301 | 3 | 160 | 1 | 5 |
| **Philips** | 8051 | P83C864 | 16 | 256 | 4 | 8 |
| **Philips** | 8051 | P83W864 | 20 | 256 | 4 | 8 |
| **Philips** | 8051 | P83C868 | 20 | 384 | 8 | 8 |
| **Philips** | 8051 | P83W868 | 20 | 384 | 8 | 8 |
| **Philips** | 8051 | P83W8616 | 20 | 384 | 16 | 8 |
| **Siemens** | 8051 | SLE44C10S | 7 | 256 | 1 | 5 |
| **Siemens** | 8051 | SL44C20S | 7 | 256 | 2 | 5 |
| **Siemens** | 8051 | SLE44C40S | 7 | 256 | 4 | 5 |
| **Siemens** | 8051 | SL44C42S | 15 | 256 | 4 | 5 |
| **Siemens** | 8051 | SLE44C80S | 15 | 256 | 8 | 5 |
| **Siemens** | 8051 | SLE44C160S | 15 | 256 | 16 | 5 |

*Tabelle 2 : Karten ohne Coprezessor*

## 3.3 Java™-Karten

Im Gegensatz zu den obengenannten Karten, die mit dem jeweiligen 8051- bzw. 68HC05 Assemblercode programmiert werden müssen, sind seit April 1997 erste Java-Interpreter-Karten verfügbar **[38]**. Das Konzept der Java-Karte ist in Abbildung 7 ersichtlich. Der Unterschied zu konventionellen Karten ist die Interpreter-Schicht. Java ist bekanntlich eine von Sun Inc. entwickelte objektorientierte Programmiersprache, die einen großen Teil ihrer Popularität dem Internet verdankt. Dort können sog. Java-Applets durch den Web-Browser heruntergeladen und auf dem lokalen Rechner ausgeführt werden. Den dadurch entstehenden Sicherheits-Risiken wurden durch das Design der Java-API, d.h. dem stark eingeschränkten ‚Java-Sandkasten', bereits Rechnung getragen. Im Fall der Java-Karte können dann diese Applets mit entsprechenden Sicherheitsvorkehrungen auch auf die Karte geladen und dort ausgeführt werden. Für Entwickler ist Java auch besonders interessant, da sich durch den objektorientierten Ansatz der Quellcode gut wiederverwenden läßt. Für unsere Zwecke würde das bedeuten, daß eine Java-Implementierung Elliptischer Kurven auf der Karte, sowie z.B. auf einem PC ablaufen kann. Die oben gemachte Unterscheidung zwischen den Prozessor-Typen wäre nicht nötig, da die Java-Implementierung prozessorunabhängig ist. Hier sind allerdings ein paar Wehrmutstropfen angebracht:

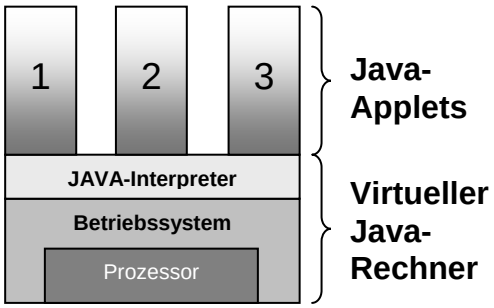



*Abbildung 7: Konzept der Java-Karte*

- Es ist bisher nur eine Java-Karte verfügbar - die ***'Cyberflex'***-Karte der Firma Schlumberger **[38]**. Diese Karte besitzt einen 8-Bit-Motorola 68HC05-SC49 Prozessor mit Coprozessor. Allerdings wird dieser Coprozessor ***nicht*** genutzt. In der aktuellen Version der Java-Card-API wird bisher nur 8-Bit Arithmetik unterstützt.
- Es gibt (noch) keine Langzahl-Arithmetik, die als Applet genutzt werden kann
- Durch das Interpreterkonzept sind arithmetische Operationen etwa ***100 mal langsamer***, als die gleichen Operationen auf dem gleichen Chip in Assembler.

Für die Zukunft, d.h. nach Erweiterung der Java-Card-API und bei Verfügbarkeit entsprechend leistungsstarker (16 oder gar 32-Bit) Chips, jedoch dürfte das Konzept der Java-Karte auch für die Implementierung von rechenintensiven kryptographischen Algorithmen sehr interessant werden.

# 4 Implementierung Elliptischer Kurven

In diesem Abschnitt wollen wir etwas näher auf die implementierungstechnischen Aspekte Elliptischer Kurven auf Chipkarten eingehen. Grundsätzlich existieren drei Möglichkeiten die Implementierung Elliptischer Kurven zu optimieren:

***1. Optimierung der Körperarithmetik***

***2. Varianten für Punktmultiplikation***

Die zeitaufwendigste Operation in einem kryptographischem DL-Protokoll ist die Berechnung eines Punktvielfachen *nP*. Ein Überblick über die verschieden Verfahren findet man z.B. in **[12]**. Die naive Methode diese Punktmultiplikation durchzuführen ist mit dem

sog. Square&Multiply-Verfahren (z.B. **[17]**) gegeben. Der Aufwand für die Berechnung von *nP*, also die Länge der Additionskette beträgt

$$l(n) = \lambda(n) + \gamma(n) - 1 \qquad (7)$$

Gruppenoperationen, wobei $\lambda(n) = \lfloor \log_2 n \rfloor$ die Länge und $\gamma(n)$ das Gewicht des Faktors. Unterstellt man Gleichverteilung für die Faktoren, so benötigen wir für 160 Bit Faktoren durchschnittlich etwa ***239 Gruppenoperationen***. Allerdings kann dieser Aufwand durch eine geschicktere Auswertung des Faktors reduziert werden. Während der Aufwand für eine *m*-Bit-Fenstermethode (im Durchschnitt, bei Gleichverteilung der Faktoren) nur etwas geringer[2] ist

$$l_m(n) = 2^{m-1} + \frac{2^m - 1}{2^m}\left\lceil \frac{\lfloor \log_2 n \rfloor}{m} \right\rceil + \lambda(n), \qquad (8)$$

läßt sich der Aufwand durch Precomputation Varianten deutlich reduzieren. Bei der für 160 Bit Faktoren optimalen Fenstergröße *m*=4 entspricht das durchschnittlich etwa ***206 Gruppenoperationen***. Da der verwendete permanente Speicherplatz im EEPROM der Karte u.U. stark eingeschränkt ist, kommen keine 'Intensiv'-Precomputation-Varianten für die 'Punktmultiplikation' wie z.B. **[1]** zum Einsatz. Allerdings stellen Precomputation, bei denen lediglich ein weiterer Kurvenpunkt in der Karte gespeichert werden muß einen guten Kompromiß zwischen Laufzeit und Speicherplatzbedarf dar. Aus **[12]** wissen wir, daß mit der *'(2 × 1) Additionskette mit Online Precomputation'* eine sehr effiziente Variante gegeben ist, bei der lediglich ein weiterer Kurvenpunkt im EEPROM der Karte gespeichert werden muß. Diese Methode verbindet die permanente Precomputation aus **[1]** und **[22]** mit der Online-Precomputation der Festervariante aus **[17]**. Der Aufwand für dieses Verfahren beträgt (wieder bei Gleichverteilung der möglichen Faktoren) durchschnittlich

$$l_{(2\times1)}(x) = \left\lceil \frac{\lambda(x)}{2} \right\rceil + \frac{2^4 - 1}{2^4}\left\lceil \frac{\lambda(x)}{4} \right\rceil + 10 \qquad (9)$$

Gruppenoperationen. In unserem Fall sind das durchschnittlich etwa ***127.5 Gruppenoperationen***.

### *3. Punktdarstellung (affin - projektiv)*

Die Frage, welche Punktdarstellung für die jeweilige Punktmultiplikationsvariante zu bevorzugen ist, wurde in **[12]** ausgiebig erörtert. Da die affine Punktdarstellung in jedem Fall die Implementierung des **E**rweiterten **E**uklidischen **A**lgorithmus auf der Karte voraussetzt und selbst bei relativ effizienter Implementierung des EEA der projektiven Punktdarstellung unterlegen ist, wird die projektive Punktdarstellung verwendet.

Die ***projektive Ebene*** $\mathbb{P}^2(K)$ über einem Körper *K* ist die Menge aller Äquivalenzklassen der Tripel $(A_1, A_2, A_3)$, $A_i \in K$ bezüglich der Äquivalenzrelation $(X_1, X_2, X_3) \sim (Y_1, Y_2, Y_3) :\Leftrightarrow \exists\, \lambda \in K \setminus \{0\}: X_i = \lambda Y_i$, ohne der Äquivalenzklasse des Tripels (0,0,0). Der Wechsel von affiner zu projektiver Darstellung geschieht durch Anhängen der dritten Koordinate $A_3=1$. Der umgekehrte Weg kann durch eine Inversion und zwei Multiplikationen erreicht werden. Der Punkt im Unendlichen, was in der projektiven Ebene dem Horizont entspricht hat übrigens die Koordinaten *O*(0,1,0).

Das projektive Pendant zu (1), d.h. die definierende Gleichung der elliptischen Kurve in der projektiven Ebene ist nun

$$E(X,Y,Z): Y^2Z = X^3 + aXZ^2 + bZ^3 \qquad (10)$$

[2] Der Aufwand (8) ist für die statische Auswertung eines *m*-Bit-Fensters und läßt sich durch eine dynamische Auswertung weiter geringfügig verbessern.

Es sei wieder $P(X_1, Y_1, Z)$, $Q(X_2, Y_2, Z)$ und $R(X_3, Y_3, Z_3)$, $P,Q,R \in E$, $P,Q \neq O(0,1,0)$ und $P \neq -Q$. Dann ist $R:=P+Q$
für $P \neq Q$

$$
\begin{aligned}
&X_3 = -su \\
&Y_3 = t(u + s^2 X_1) - s^3 Y_1 \\
&Z_3 = s^3 Z \\
&\textit{wobei} \\
&u = s^2 (X_1 + X_2) - t^2 Z \\
&\textit{und} \\
&s = X_2 - X_1, \quad t = Y_2 - Y_1
\end{aligned}
\qquad (11)
$$

und für $P=Q$

$$
\begin{aligned}
&X_3 = 2sh \\
&Y_3 = w(4f - h) - 8e^2 \\
&Z_3 = 8s^3 \\
&\textit{mit} \\
&s = Y_1 Z \\
&w = 3X_1^2 + aZ^2 \\
&e = Y_1 s \\
&f = X_1 e \\
&h = w^2 - 8f
\end{aligned}
\qquad (12)
$$

Mit diesen beiden Formeln aus **[18]** benötigen wir für die Punktaddition mit (11) zehn Körpermultiplikationen, falls die beiden Punkte bereits die gleichen Z-Koordinaten besitzen. Im allgemeinen muß jedoch vor Anwendung von (11) kreuzweise multipliziert werden (fünf Multiplikationen), womit wir insgesamt bei ***15 Körpermultiplikationen*** angelangt wären. Für das Verdoppeln eines Punktes benötigen wir mit (12) ***zehn Körpermultiplikationen***.[3]

## 4.1 Implementierung auf Karten mit Coprozessor

Die naheliegendste Plattform für die Implementierung Elliptischer Kurven auf Smartcards sind natürlich Chips mit integrierten kryptographischen Coprozessoren. Diese Coprozessoren sind je nach konkreter Ausprägung in der Lage Modulomultiplikationen und -exponentiationen mit einer Modul-Länge bis zu 1024 Bit auszuführen. Das bedeutet, daß die Implementierung Elliptischer Kurven über $GF(p)$ mit 160 Bit $p$ problemlos möglich ist, da die Modulo-Arithmetik des Coprozessors verwendet werden kann. Für die Kurvenarithmetik müssten lediglich die folgenden Routinen (allerdings im jeweiligen Assembler) implementiert werden:

- PADD$(P_1,P_2)$ Punktaddition mit (11)
- PDBL$(P)$ Punktverdopplung mit (12)
- PSUBNORM$(P_1,P_2)$ Punktsubnormalisierung, d.h. kreuzweise Multiplikation
  $P_1(x_1,y_1,z_1) \rightarrow P_1(x_1\, x_2, y_1\, y_2, z_1\, z_2)$

[3] Es sei noch einmal angemerkt, daß wir Additionen, Subtraktionen sowie die Multiplikation mit kleinen Konstanten (z.B. $a$ in den Kurvengleichungen (1) und (10) ) bei dieser Betrachtung vernachlässigen können.

$P_2(x_2,y_2,z_2) \rightarrow P_2(x_2\, x_1\, ,\, y_2\, y_1\, ,\, z_2\, z_1)$

- PNORM$(P_1)$ Punktnormalisierung, d.h. Transformation in affine Koordinaten, durch eine Inversion und zwei Multiplikationen
- PEQUAL$(P_1,P_2)$ Gleichheitstest
- PMUL$(P_1,P_2)$ Punktmultiplikation mit einer der oben geschilderten Varianten

Zur Performanceabschätzung und dem Vergleich der EC-DSA-Signatur mit DSA- und RSA-Signaturen wollen wir davon ausgehen, daß die RSA- und DSA Exponentiation mit einer 5-Bit-Fenstermethode durchgeführt wird, das Gewicht des Exponenten die Hälfte der Bitlänge beträgt und daß die Modulo-Multiplikation quadratische Komplexität besitzt. Das bedeutet, daß (nach (8)) bei der gewöhnlichen Signatur (ohne CRT) durchschnittlich 1239 modulare 1024-Bit Multiplikationen nötig sind. Bei Anwendung des chinesischen Restsatzes (CRT) **[32]** benötigen wir rund 2·628+2=1258 modulare 512 Bit Multiplikationen. Bei quadratischer Komplexität der Modulomultiplikation entspricht das etwa $1258\cdot(512/1024)^2 \approx 315$ modularer 1024-Bit Multiplikationen. Im Vergleich dazu benötigen wir für eine DSA-Signatur nur etwa 207+20=227 1024 Bit Multiplikationen, wobei der Aufwand für die Berechnung von $s=(k^{-1}(h(m)+xr) \bmod q$ mit 20 1024 Bit Multiplikationen veranschlagt wurde. Für eine EC-DSA Signatur benötigen wir mit der Speicherung eines zusätzlichen Punktes nach (9) durchschnittlich 81 Punktverdoppelungen und rund 47 Punktmultiplikationen. Das heißt insgesamt benötigen wir dazu 81·10+47·15+22=1537 160 Bit Multiplikationen. Auch hier wurde die Transformation in affine Koordinaten (1 Inversion und 2 Multiplikationen) mit 20+2=22 Multiplikationen 'verbucht'. Wenn wir quadratische Komplexität der Modulo-Multiplikation unterstellen, so entspricht das nur etwa $1537\cdot(160/1024)^2 \approx 38$ modularer 1024 Bit Multiplikationen. Die Ergebnisse sind in Tabelle 3 zusammengefaßt. Wir unterstellen, daß im EEPROM der Karte die folgenden Parameter gespeichert sind:

- RSA (ohne CRT) $d,n$
- RSA (mit CRT) $p,q,d_p\,,d_q\,,invq$
- DSA $p,q,g,x$
- EC-DSA $a,b,p,q,x,P_1(x_1\,,y_1),\ P_2(x_2\,,y_2)$

Außerdem wird hier unterstellt, daß der Kurvenparameter $a$ und $x_1$ als 'Shortinteger' gewählt sind und deshalb nur 4 Byte Speicher benötigen. Die letzten beiden Spalten sind die zu erwartenden Laufzeiten auf den Prozessoren Philips 83C852 und Siemens 44C200 (siehe **[29]**). Die mit * gekennzeichneten Werte sind extrapoliert.

| Algorithmus | Modul | EEPROM für Parameter (Byte) | mod. Mult. # (Bit) | ≈ mod. Mult (1024 Bit) | Philips 83C852 in ms | Siemens 44C200 in ms |
|---|---|---|---|---|---|---|
| **RSA ohne CRT** | 1024 | 256 | 1239 (1024) | 1239 | 8493* | 640* |
| **RSA mit CRT** | 1024 | 448 | 1258 (512) | 315 | 2160* | 163* |
| **RSA ohne CRT** | 768 | 192 | 934 (768) | 525 | 3600 | 271 |
| **RSA mit CRT** | 768 | 336 | 952 (384) | 134 | 919* | 69* |
| **DSA-Sign** | 1024 | 296 | 227 (1024) | 227 | 1557* | 117* |
| **DSA-Sign** | 768 | 232 | 227 (768) | 127 | 817* | 65* |
| **DSA-Verify** | 1024 | s.o. | 22 (160) + 414 (1024) | 415 | 2845* | 214* |
| **DSA-Verify** | 768 | s.o. | 22 (160) + 414 (768) | 233 | 1598* | 120* |
| **EC-DSA-Sign** | 160 | 145 | 1537 (160) | 38 | 260* | 19* |
| **EC-DSA-Verify** | 160 | s.o. | 3864 (160) | 94 | 645* | 49* |

*Tabelle 3: Signatur-Algorithmen mit Coprozessor im Vergleich*

An dieser Stelle sei noch einmal darauf hingewiesen, daß diese Abschätzungen äußerst grob sind. So wurde beispielsweise der Aufwand für die Berechnung des Hashwertes einfach vernachlässigt. Außerdem wurde unterstellt, daß der Aufwand für die Modulo-Multiplikation quadratische Komplexität besitzt, was für bestimmte Chips nicht genau zutrifft. Die Ergebnisse dieser Tabelle können daher nur 'ein Gefühl' für die zu erwartende Laufzeit vermitteln. Allerdings ist bereits deutlich ersichtlich, daß die Implementierung Elliptischer Kurven selbst auf (leicht angestaubten) Chips, wie dem Philips 83C852[4] mit ***sehr guter Performance*** möglich ist.

Zu beachten ist schließlich ein im Vergleich zu RSA ein etwas ***erhöhter (EEP)ROM-Bedarf*** für die Kurvenarithmetik und ggf. die zusätzliche Implementierung des EEA zur Inversion mod $p$.

## 4.2 Implementierung auf Karten ohne Coprozessor

Da die Leistungsfähigkeit des Coprozessors (z.B. modulare 1024 Bit-Multiplikation) bei der Implementierung ohnehin nicht völlig ausgenutzt wird, liegt der Gedanke nahe, ganz auf diesen zu verzichten. Karten ohne Coprozessor sind etwa für den ***halben Preis*** zu haben, als vergleichbare Karten mit einer Kryptoeinheit. Das bedeutet aber, daß neben der Kurvenarithmetik, wie im vorherigen Kapitel erläutert, eine möglichst effiziente Körperarithmetik im entsprechenden 8051- bzw. 68HC05-Assembler zu implementieren ist, was wiederum etwas ***mehr (EEP)ROM*** beansprucht. Für diese Karten gibt es verschiedene Möglichkeiten der Implementierung, die jeweils weiterer Forschung bedürfen:

- ***GF(p)-Artihmetik***, wobei $p>2^{160}$
- ***Polynomarithmetik über GF(2)***, *d.h.* $GF(2^m)$-Arithmetik, wobei $m>160$
- ***Polynomarithmetik über GF(p)***, wobei $p<256$, d.h. $GF(p^k)$-Arithmetik mit $k>20$

Daß eine solche Implementierung möglich ist, hat die Firma Schlumberger **[4]** bereits gezeigt. Es wurde ein Signatursystem auf Basis Elliptischer Kurven auf der sog. ***Multiflex-Karte ohne Koprozessor implementiert***. Der Chip auf dieser Karte ist ein 68HC05-SC28. Aus Tabelle 2 sehen wir, daß 12,8 kByte ROM, 240 Byte RAM und 8 kByte EEPROM zu verfügung stehen. Davon wurden ca. 90 Byte RAM und rund 4 kByte EEPROM für die Implementierung des Signatursystems benötigt. Die Erstellung der Signatur dauert ca. 600 ms, was für durchaus vergleichbar zu entsprechenden RSA-Implementierungen auf Karten mit Coprozessor ist.

## 4.3 Implementierung auf Java™-Karten

Da durch das Java-Interpreter-Konzept eine mögliche Implementierung etwa 100 mal langsamer ist, als eine Assembler-Programmierung auf dem gleichen Chip, ist die Implementierung Elliptischer Kurven auf diesen Karten ***nicht mit akzeptabler Performance möglich***. Mit einer Änderung der Java-Card-API, d.h. die Möglichkeit den vorhandenen Co-Prozessor durch Java-Kommandos anzusteuern, oder leistungsfähigeren 16- oder 32-Bit-Chips könnte diese Möglichkeit in einigen Jahren sehrwohl interessant werden.

[4] Z.B. 1024 Bit RSA-Operationen sind mit diesem Chip nicht nur zu langsam, sondern durch das relativ kleine (EEP)ROM (2kB bzw. 6kB) sogar unmöglich.

# 5 Zusammenfassung

In dieser Arbeit wiederholten wir recht ausführlich die nötigen Grundlagen Elliptischer Kurven über Primkörpern und diskutierten Ansätze die Implementierung möglichst effizient zu gestalten. Außerdem gaben wir einen Überblick über die zur Zeit gebräuchlichen Prozessor- und Kartentypen, die potentiell für die Implentierung Elliptischer Kurven in Frage kommen und schätzten grob die zu erwartende Laufzeit auf Karten mit kryptographischem Coprozessor ab. Dies zeigte, daß selbst Chips, die heute (z.B. für RSA) nicht mehr tauglich sind, als Plattform für die Implementierung Elliptischer Kurven dienen können. Das bedeutet, daß die gleiche Sicherheit mit ***preisgünstigeren Chipkarten*** erreicht werden kann. Es ist sogar möglich ganz ***auf den Coprozessor verzichten***. Welche der möglichen Alternativen zu bevorzugen ist muß weiter untersucht werden. Ein weiterer offener Punkt für die Zukunft ist die Integration Elliptischer Kurven in Anwendungsstandards (wie z.B. HBCI). Der Schritt des BSI, Elliptische Kurven für Signaturen im Sinne des Signatur-Gesetzes vorzuschlagen sollte nicht der letzte in diese Richtung bleiben.

## Literatur:


**[1]** E. Brickell, D. Gordon, K. McCurley, D. Wilson: „Fast Exponentiation with Precomputation“, Proceedings of EUROCRYPT ’92, LNCS 658, Springer Verlag, Berlin 1993, SS. 200-207

**[2]** J. Buchmann, J. Loho, J. Zayer: „An implementation of the general number field sieve“, Advances in Cryptology Crypto (1993), Lecture Notes in Computer Science, 773, SS. 159-165

**[3]** Bundesamt für Sicherheit in der Informationstechnik (BSI): "Maßnahmenkatalog zur digitalen Signatur, Teil 6.1. - KryptoalgorithmenV", 1997,
via http://www.bsi.bund.de/aktuell/index.htm

**[4]** Schlumberger: „Elliptic Curve Digital Signatures on Smart Cards without Coprocessor“, Mai 1997, via
http://www.schlumberger.com

**[5]** B. Dodson and A.K. Lenstra: "NFS with four large primes: An explosive experiment", Proceedings of Crypto '95, Springer-Verlag, 1995, SS. 372-385

**[6]** T. ElGamal: „A Public Key Cryptosystem and a Signature Scheme based on discrete Logarithms“, Proceedings of CRYPTO ‘84, Springer, Berlin 1985, SS. 10-18

**[7]** D. Fox, A. Röhm: "Effiziente Digitale Signatursysteme auf der Basis elliptischer Kurven", Tagungsband "Digitale Signaturen", DuD Fachbeiträge, Vieweg, 1996, SS. 201-220

**[8]** W. Fulton: "Algebraic Curves", Benjamin, 1969

**[9]** S. Goldwasser, J. Kilian: „Almost all primes can be quickly certified“, Proceedings of the 18th Annual ACM Symposium on Theory of Computing, 1986, SS. 316-329

**[10]** D. Gordon: "Discrete Logarithms in GF(p) Using the Number Field Sieve", Siam Journal on Discrete Mathematics 6, 1993, SS. 124-138

**[11]** J. Guajardo, C. Paar: "Efficient Algorithms for Elliptic Curve Cryptosystems", erscheint in den Proceedings der CRYPTO '97, 1997

**[12]** D. Hühnlein: "Effiziente Exponentiation und optimale Punktdarstellung für Signatursysteme auf Basis elliptischer Kurven", Tagungsband "Digitale Signaturen", DuD Fachbeiträge, Vieweg, 1996, SS. 221-235

**[13]** D. Husemöller: „Elliptic Curves“, Graduate Texts in Mathematic - 111, Springer Verlag, Berlin, 1986, ISBN 3-540-96371-5

**[14]** IEEE: "IEEE P1363 Working Draft", z.B. über ftp://stdsbbs.ieee.org/pub/p1363/predrafts

**[15]** ISO 7816: "Identification cards - Integrated circuit(s) card with contacts",
"Part 1: Physical characteristics", 1987
"Part 2: Dimensions and location of the contacts", 1988
"Part 3: Electronic signals and transmission protocol", 1989
"Part 3-Amd 1: Amendment 1: Protocol type T=1, asynchronous half duplex block transmission protocoll", 1992
"Part 3-Amd 2: Amendment 2: Protocol type selection", 1994
"Part 4: Inter-industry commands for interchange ", 1995
"Part 5: Numbering system and registration procedure for application identifiers", 1994
"Part 5-Amd 1: Registration of identifiers", 1995
"Part 6: Inter-industry data elements", 1995
"Part 7: Enhanced inter-industry commands", 1995
"Part 8: Inter-industry security architecture", 1995

**[16]** N. Koblitz: "Elliptic Curve Cryptosystems", Math. Comp., vol. 48, 1987, 203-209

**[17]** D. E. Knuth: „The Art of Computer Programming-Vol.2 Seminumerical Algorithms“, 2nd Ed., Addison-Wesley, Massachusets 1981

**[18]** K. Koyama, Y. Tsuruoka: „A signed binary window method for fast computing over elliptic curves“, Proceedings of CRYPTO 1992, LNCS 740, Springer-Verlag, Berlin, 1993, SS. 345-357

**[19]** G-J. Lay und H.G. Zimmer: „Constructing elliptic curves with given group order over large finite fields“, in L. Adleman (Ed.), ANTS-I (1994), Lecture Notes in Computer Science - 877, Berlin, Springer Verlag

**[20]** A.K. Lenstra, H.W. Lenstra: "The Development of the Number Field Sieve (LNM 1554), Springer, 1993

**[21]** LiDIA-Group: "Library for computational number theory", http://www.informatik.th-darmstadt.de/TI/LiDIA/Welcome.html

**[22]** C.H.Lim, P.J.Lee: „More Flexible Exponentiation with Precomputation“, Pre-Proceedings of CRYPTO 1994, Springer Verlag, Berlin 1994, SS. 95-107

**[23]** A.J. Menezes: „Elliptic Curve Public Key Cryptosystems“, Kluwer Academic Press, Dordrecht 1993, ISBN 0-7923-9368-6

**[24]** A.J. Menezes, T. Okamoto und S.A. Vanstone: "Reducing elliptic curve logarithms to logarithms in finite fields", Proceedings of 23rd Annual ACM Symposium on Theory of Computing (STOC), 1991, SS. 80-89

**[25]** A.J. Menezes, P.C. van Oorschot, S.A. Vanstone: „Handbook of Applied Cryptography“, CRC Press, 1996, ISBN 0-8493-8523-7

**[26]** V. Miller: "Use of Elliptic Curves in Cryptology", Advances in Cryptology: Proceedings of Crypto '85, LNCS 218, Springer, 1986

**[27]** Motorola: "M68HC05SC Family - At a Glance"

http://mot-sps.com/csic/SMARTCRD/sctable.htm

**[28]** V. Müller, S. Paulus: „On the generation of cryptographical strong elliptic curves“, eingereicht zur Eurocrypt 1998, preprint via http://www.informatik.th-darmstadt.de/TI/reports

**[29]** D. Naccache: "Arithmetic Co-processors: State of the Art", Hand-out zu Vortrag auf der Eurocrypt '95, Saint-Malo, 1995

**[30]** National Institute of Standards and Technology (NIST): Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186 (FIPS-186), 19th May, 1994

**[31]** Philips: "Integrated Circuits and Modules for CHIP CARDS", Produktblatt, 1997

**[32]** J.J. Quisquater, C. Couvreur: "Fast Decipherment Algorithm for RSA Public-Key Cryptosystem", Electronic Letters, vol. 18, no. 21, Oct 1982, SS. 905-907

**[33]** W. Rankl, W. Effing: "Handbuch der Chipkarten", Hanser-Verlag, München, 1996, ISBN 3-446-18893-2

**[34]** R. Rivest, A. Shamir, L. Adleman: „A method for obtaining Digital Signatures and Public-Key-Cryptosystems", Communications of the ACM, v.21,n.2, Feb 1978, SS. 120-126

**[35]** J. Sauerbrey, A. Dietel: „Resource Requirement for the Application of Addition Chains in Modulo Exponentiation“, Proceedings of Eurocrypt 1992, Lecture Notes in Computer Science - 658, Springer Verlag, Berlin 1993, SS. 174-182

**[36]** O. Schirokauer, D. Weber, T. Denny: "Discrete Logarithms and the Effectiveness of the Index Calculus Method", Proceedings of ANTS II, Springer, 1996, SS. 337-362

**[37]** C.P. Schnorr: „Efficient Identification and Signatures for Smart-Cards“, Proceedings of CRYPTO ‘89, Springer Verlag, Berlin 1990, SS. 237-252

**[38]** Schlumberger: " First-Ever Java-Based Smart Card Demonstrated by Schlumberger", Pressemitteilung, 02. April, 1997 http://www.slb.com/ir/news/et-java0497.html

**[39]** Siemens: "Security Controller IC'S", http://www.sci.siemens.com/htdocs/catalog/ICs/smartcard/scics.html

**[40]** J. Silverman: "The Arithmetic of Elliptic Curves", Graduate Texts in Mathematics - 106, Berlin, Springer-Verlag, 1986

**[41]** Telesec: "B1- Das Konzept für ein universelles Chipkartenzugangsgerät", http://www.telesec.de/b1.htm

**[42]** Telesec: "TCOS-Telesec Chipcard-Operating System", http://www.telesec.de/tcos.htm

**[43]** E. Waterhouse: "Abelian Varieties over Finite Fields", Ann. Sci. Ecole Norm. Sup., vol. 2, 1969, SS. 521-560

**[44]** E. de Win, A. Bosselaers, S. Vandenberghe, P. de Cersem J. Vandewalle: "A Fast Software Implementation for Arithmetic Operations in $GF(2^n)$", Proceedings of Asiacrypt '96, Springer, 1996