KRITERIEN ZUR OPTIMALEN AUSWAHL VON ELLIPTIC CURVE CRYPTOGRAPHY ALS HARD- ODER SOFTWARELÖSUNG

Mathias Schmalisch, Hagen Ploog und Dirk Timmermann

Institut für Angewandte Mikroelektronik und Datentechnik, Universität Rostock
Richard-Wagner-Str-31, 18119 Rostock
Tel.: +49 (381) 498 35 36
Fax: +49 (381) 498 36 01
Email: mathias.schmalisch@uni-rostock.de

Kurzfassung: Bei der Entwicklung von Geräten, die zur Verschlüsselung von Daten verwendet werden, steht man meist vor der Wahl das Verschlüsselungsverfahren entweder in Hardware oder in Software zu implementieren. Hat man sich für eine Variante entschieden, muss man sich für einen Algorithmus entscheiden. Gerade bei den asymmetrischen Verschlüsselungsverfahren gibt es verschiedene Algorithmen, mit denen das Verfahren berechnet werden kann. Dieser Beitrag stellt die verschiedenen Algorithmen für die Elliptic Curve Cryptography (ECC) gegenüber und gibt eine Entscheidungshilfe für eine Hard- bzw. Softwareimplementation.

1 EINFÜHRUNG

In der modernen Kryptographie werden zwei Arten von Verfahren unterschieden. Das sind die symmetrischen und asymmetrischen Verfahren, wobei jede Art ihre Vor- und Nachteile hat. Bei den symmetrischen Verfahren, auch Private-Key-Verfahren genannt, gibt es nur einen Schlüssel. Dieser wird sowohl zu Ver- als auch zum Entschlüsseln benötigt. Daher ist es wichtig, eine sichere Möglichkeit zu finden, um den Schlüssel an beide Kommunikationsteilnehmer zu verteilen. Im Gegensatz dazu kommen bei den asymmetrischen Verfahren, auch Public-Key-Verfahren genannt, zwei Schlüssel zum Einsatz. Ein Schlüssel ist der öffentliche und der andere der private Schlüssel. Der öffentliche Schlüssel kann an alle Nutzer verteilt werden, ohne die Sicherheit zu gefährden. Wenn nun jemand Informationen austauschen möchte, verschlüsselt er diese mit dem öffentlichen Schlüssel des Empfängers. Diese verschlüsselte Information kann dann nur mit dem privaten Schlüssel des Empfängers wieder entschlüsselt werden. Dadurch ist der Schlüsselaustausch bei den asymmetrischen Verfahren unproblematischer als bei den symmetrischen Verfahren. Die asymmetrischen Verfahren besitzen den Nachteil, dass sie etwa 100 bis 1000 mal langsamer als vergleichbare symmetrische Verfahren sind. Sie werden in der Regel nur zur Authentifizierung und zum Schlüsselaustausch für den symmetrischen Schlüssel eingesetzt.

Der erwähnte Geschwindigkeitsunterschied ist von der benötigten Schlüssellänge und der Anzahl mathematischer Operationen, die das entsprechende Verfahren benötigt, abhängig. In einer aktuellen Untersuchung [1] wurden verschiedene Verfahren hinsichtlich ihrer Sicherheit miteinander verglichen. Mit Hilfe dieser Untersuchung wurde die Abbildung 1 erstellt, in dieser Abbildung wird das RSA-Verfahren [2] dem ECC-Verfahren gegenübergestellt. Dabei wird die Sicherheit über das Jahr angezeigt, ab dem das Verfahren mit der entsprechenden Schlüssellänge wahrscheinlich gebrochen werden kann. In der

Abbildung erscheint das ECC-Verfahren zwei mal, "ECC" und "ECC+". Dabei wird beim "ECC" davon ausgegangen, dass durch Kryptoanalyse keine weiteren Möglichkeiten gefunden werden, um das Verfahren schneller zu brechen. Da das ECC-Verfahren aber noch relativ neu ist, sollte davon ausgegangen werde, dass die Kryptoanalyse noch einige Möglichkeiten aufdeckt, mit der das ECC-Verfahren schneller gebrochen werden kann. In diesem Fall gilt dann der "ECC+" Graph. Weiterhin ist in Abbildung 1 sehr gut zu sehen, dass die Schlüssellänge beim RSA-Verfahren wesentlich stärker steigt als beim ECC-Verfahren.

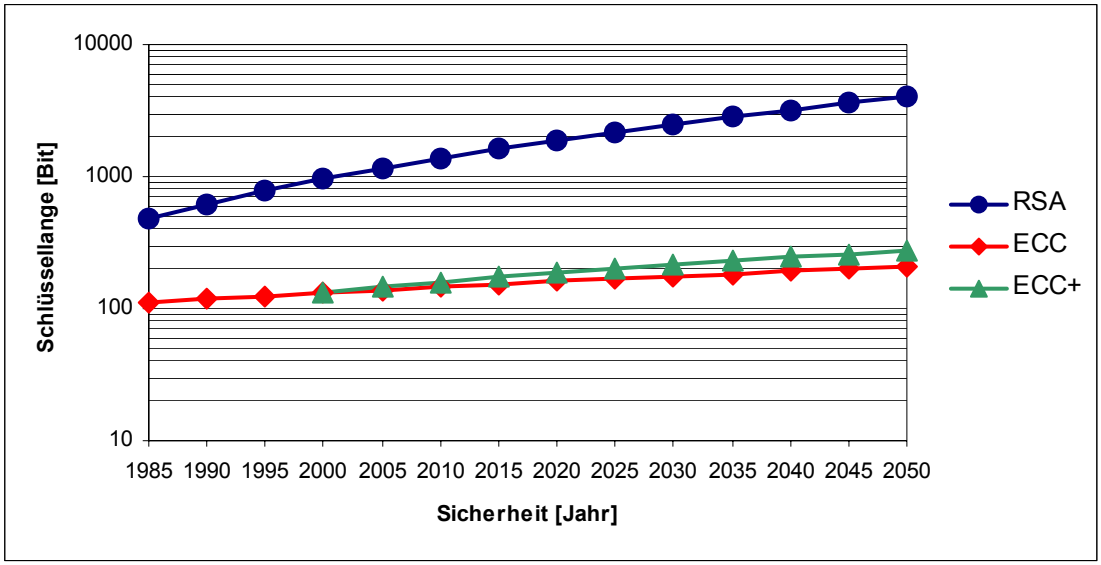

Abbildung 1: Vergleich der Schlüssellänge von RSA und ECC

2 ELLIPTISCHE KURVEN

Im Jahre 1986 wurden von Neal Koblitz [3] und Victor Miller [4] erstmals Möglichkeiten vorgestellt, wie elliptische Kurven in der Kryptographie eingesetzt werden können. Seit dieser Zeit hat sich die sogenannte Elliptic Curve Cryptography einen bedeutenden Platz in der Public Key Kryptographie erobert.

Elliptische Kurven werden auch als kubische Kurven bezeichnet. Dabei handelt es sich um eine Menge von Punkten (x, y) in der affinen Ebene, deren Koordinaten eine bestimmte Gleichung erfüllen. Die allgemeine Weierstrass-Normalform in der affinen Ebene für eine elliptische Kurve E über den Körper K lautet:

$$E / K : y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6 \quad y, x, a_i \in K \tag{1}$$

Diese Form lässt sich unter einigen Voraussetzungen wesentlich vereinfachen. Dazu ist es notwendig zu wissen, auf welchen Körper die elliptische Kurve abgebildet wird. In der Kryptographie werden die elliptischen Kurven auf endliche Körper GF(q), auch Galois Felder genannt, abgebildet. Dabei ist q eine Primzahlpotenz $q = p^m$. Für kryptographische Verfahren werden die Sonderfälle $m = 1$ bzw. $p = 2$ eingesetzt. Wobei bei GF(2^m) noch zwischen Polynom- und Normalbasis unterschieden wird, siehe Abbildung 2.

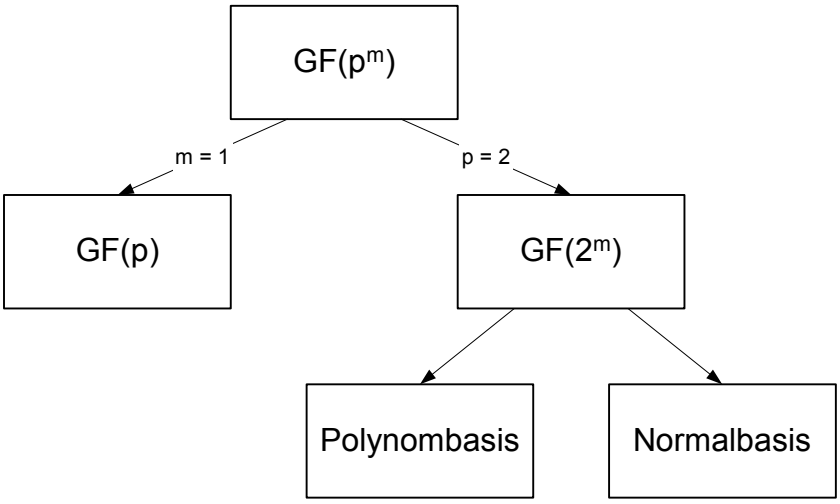

Abbildung 2: Endliche Körper für elliptische Kurven

Für die Implementation des ECC-Verfahrens in Hardware eignen sich die elliptischen Kurven über den endlichen Körper von GF(2^m) am besten. Für diesen Fall lässt sich die allgemeine Weierstrass-Normalform auf folgende Gleichung vereinfachen, siehe dazu auch [5]:

$$E / K : y^2 + xy = x^3 + ax^2 + b \quad y, x, a, b \in K \tag{2}$$

Alle elliptischen Kurven, welche Gleichung (2) erfüllen, dürfen nicht singulär sein. Ansonsten sind die Punktoperationen nicht auf der elliptischen Kurve definiert. Um die Singularität zu bestimmen, wird die Diskriminante Δ berechnet. Wenn $\Delta \neq 0$ ist, dann ist die entsprechende elliptische Kurve nicht singulär. Die Diskriminante für Gleichung (2) ist sehr einfach zu berechnen:

$$\Delta = b \tag{3}$$

Somit darf *b* nicht null werden, damit die elliptische Kurve nach Gleichung (2) nicht singulär wird.

2.1 Punktoperationen

Die Punkte auf einer solchen elliptischen Kurve habe einige besondere Eigenschaften. Wenn eine Gerade über die elliptische Kurve gelegt wird, dann schneidet diese Gerade die Kurve in drei Punkten, wie in Abbildung 3. Die Summe dieser Punkte ergibt null:

$$P_1 + P_2 + P_3 = 0 \tag{4}$$

Somit kann dann aus zwei bekannten Punkten der dritte berechnet werden:

$$\begin{aligned} P_1 + P_2 &= -P_3 \quad \text{mit } P_3 = -Q \\ P_1 + P_2 &= Q \end{aligned} \tag{5}$$

Um den dritten Punkt *Q* zu berechnen, wird durch die beiden bekannten Punkte P_1 und P_2 eine Gerade *L* gelegt. Die Gleichung einer Geraden sieht wie folgt aus:

$$L : y = \lambda x + \beta \tag{6}$$

Mit Hilfe der Steigung λ und Verschiebung β kann dann der dritte Punkt auf der Geraden $-Q$ und mit der Punktnegation Q berechnet werden.

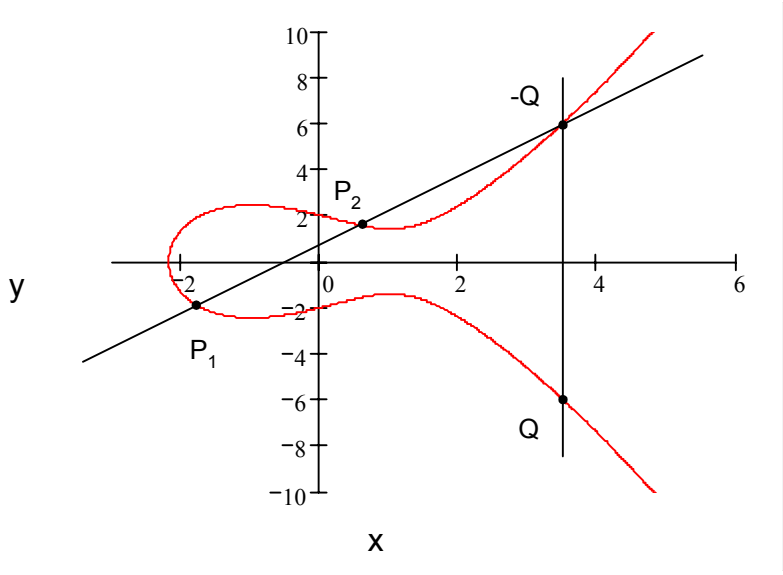

Abbildung 3: Beispiel einer elliptischen Kurven

Um die Steigung λ der Geraden zu bestimmen, sind zwei Fälle zu betrachten:

1. $P_1 \neq P_2$, mit $x_1 \neq x_2$. In diesem Fall ist λ die Steigung der Sekante durch die Punkte P_1 und P_2. Daher kann hier das Verfahren für die Sekantensteigung angewendet werden.

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2} \tag{7}$$

2. $P_1 = P_2$, mit $y_1 \neq 0$. Dann ist λ die Steigung der Tangente durch den Punkt P_1. Die Steigung der Tangente entspricht der ersten Ableitung von der elliptische Kurve E im Punkt P_1, so erhält man für λ:

$$\lambda = \frac{x1 + y1}{x1} \tag{8}$$

Wie hier gezeigt wurde, hängt die Steigung λ davon ab, ob zwei unterschiedliche Punkte miteinander addiert werden, oder ob eine Punktverdopplung stattfindet, indem ein Punkt mit sich selbst addiert wird.

Aus der Steigung kann dann die Verschiebung und Punktnegation berechnet werden. Zusammengefasst in einer Gleichung können so die Koordinaten von Q berechnet werden:

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + a + x_1 + x_2 \\ y_3 &= \lambda x_3 + x_3 + x_1^{\,2} \end{aligned} \tag{9}$$

Im Fall der Punktverdopplung kann die Gleichung (9) für x_3 durch einsetzten der Gleichung (8) auf folgende Gleichung vereinfacht werden:

$$x_3 = x_1^2 + \frac{b}{x_1^2} \tag{10}$$

2.2 Projektive Ebene

Die Punktoperationen im vorangegangenen Abschnitt sind nur für elliptische Kurven in der affinen Ebene geeignet. Es ist aber auch möglich, die Operationen in der projektiven Ebene auszuführen. Dazu werden die (x, y)-Koordinaten in (X, Y, Z)-Koordinaten transformiert:

$$\begin{aligned} \text{in projektive Koordinaten: } & X = x, Y = y, Z = 1 \\ \text{in affine Koordinaten: } & x = \frac{X}{Z}, y = \frac{Y}{Z} \end{aligned} \tag{11}$$

Die allgemeine Weierstrass-Normalform aus Gleichung (1) hat in der projektiven Ebene folgende Form:

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 \tag{12}$$

Für die Punktaddition und Punktverdopplung sind daher drei Koordinaten zu berechnen. Auf die Formeln soll hier nicht weiter eingegangen werden, sie sind in [6] nachzulesen. Durch die zusätzliche Koordinate werden die Berechnungen etwas aufwendiger als bei den Punktoperationen in der affinen Ebene. Dafür wird hier aber die Division, welche für die Berechnung der Steigung benötigt wird, eingespart. Die Division wird nur zur Rückkonvertierung der projektiven Koordinaten in affine Koordinaten benötigt.

Damit hat auch die Berechnung über die projektiven Koordinaten ihre Daseinsberechtigung, denn die Division im endlichen Körper lässt sich nur über die Berechnung des modular Inversen lösen. Aber gerade diese Berechnung ist mit großem Aufwand verbunden. Da die Berechnung des modular Inversen bei der Wahl der Ebene, über welche die Punktoperationen ausgeführt werden, eine große Bedeutung beizumessen ist, wird sie in Abschnitt 3 eingehender behandelt.

2.3 Skalarmultiplikation *k * P*

Mit Hilfe der Punktaddition und Punktverdopplung kann eine skalare Zahl *k* mit einem Punkt *P* auf der elliptischen Kurve berechnet werden. Mit dieser Skalarmultiplikation kann dann das eigentliche kryptographische Verfahren implementiert werden. Das einfachste, auf elliptische Kurven basierende Verfahren, ist der Diffie-Hellmann Schlüsselaustausch [6]. Um die Skalarmultiplikation zu berechnen gibt es unterschiedliche Algorithmen, z.B. den "Double and Add" Algorithmus:

Algorithmus 1: Double and Add
INPUT: Integer k > 0 und Punkt P
OUTPUT: Q = k * P
$k \leftarrow (k_{n-1}, ..., k_1, k_0)_2$
$Q \leftarrow P$
for i from (n – 2) downto 0 do
    $Q \leftarrow 2 * Q$
    if $k_i = 1$ then
        $Q \leftarrow Q + P$
return Q

Ein weiterer Algorithmus zur Berechnung der Skalarmultiplikation stammt von Peter Montgomery und wird in [7] und [8] näher beschrieben und hier noch einmal vorgestellt:

Algorithmus 2: Montgomery
INPUT: Integer k > 0 und Punkt P
OUTPUT: Q = k * P
$k \leftarrow (k_{n-1}, ..., k_1, k_0)_2$
$P_1 \leftarrow P$, $P_2 \leftarrow 2 * P$
for i from (n – 2) downto 0 do
    if $k_i = 1$ then
        $P_1 \leftarrow P_1 + P_2$; $P_2 \leftarrow 2 * P_2$
    else
        $P_1 \leftarrow 2 * P_1$; $P_2 \leftarrow P_1 + P_2$
return ($Q = P_1$)

Dieser Algorithmus hat den Vorteil, dass er immer die selbe Anzahl an Punktoperation benötigt, egal wie das Hamminggewicht (Anzahl der Einsen in *k*) von *k* aussieht. Wohingegen der "Double and Add" Algorithmus abhängig vom Hamminggewicht ist. Ein weitere Vorteil besteht darin, dass beim Montgomery-Algorithmus die Berechnung der *y*-Koordinate eingespart werden kann. Das liegt daran, dass immer zwei benachbarte Punkte berechnet werden. Denn Peter Montgomery hat festgestellt, dass sich aus den *x*-Koordinaten zweier benachbarter Punkte die *y*-Koordinaten berechnen lassen.

Damit gibt es zwei Algorithmen die zur Berechnung der Skalarmultiplikation geeignet sind. Des weiteren sind beide Algorithmen sowohl über die affine Ebene als auch über die projektive Ebene anwendbar. Somit ergeben sich vier Algorithmen die zur Berechnung jeweils unterschiedlich viele Operationen im endlichen Körper benötigen.

3 BERECHNUNG DES MODULAR INVERSEN

Wie schon erwähnt, ist die Berechnung des modular Inversen im endlichen Körper die aufwendigste Operation. Daher hängt die Wahl des geeigneten Algorithmus zum großen Teil von Aufwand für die Berechung des modular Inversen ab. Um das modular Inverse zu berechnen, sind zwei Möglichkeiten bekannt, die nachfolgend näher beschrieben werden.

3.1 Erweiterter Euklidischer Algorithmus

Die erste Möglichkeit für die Berechnung des modular Inversen ist der Erweiterte Euklidische Algorithmus. Mit dem einfachen Euklidischen Algorithmus kann der größte gemeinsame Teiler *d* zweier Zahlen *a* und *b* berechnet werden. Mit dem Erweiterten Euklidischen Algorithmus können zusätzlich noch die beiden Faktoren *x* und *y* berechnet werden, so dass:

$$ax + by = d \tag{13}$$

Wenn nun eine der beiden Zahlen *a* oder *b* eine Primzahl ist und die andere Zahl kleiner als diese Primzahl ist, so ergibt sich für den größten gemeinsamen Teiler immer ggT(*a, b*) = 1. Somit wird der Faktor *x* für die Primzahl *a* so berechnet, dass:

$$\begin{aligned} & ax \equiv 0 \bmod a \\ \Rightarrow\ & by \equiv 1 \bmod a \\ \Rightarrow\ & y = b^{-1} \end{aligned} \tag{14}$$

Um also das modular Inverse zu erhalten, kann der Erweiterte Euklidische Algorithmus verwendet werden, wie er im folgenden dargestellt ist:

Algorithmus 3: Erweiterter Euclidischer Algorithmus
INPUT: Modulo M, Zahl B
OUTPUT: B^{-1} mod M
S ← M; V ← 0
R ← B; U ← 1
repeat
    Q ← ⌊ S / R ⌋
    tmp ← S – Q * R; S ← R; R ← tmp
    tmp ← V – Q * U; V ← U; U ← tmp
until (R = 0)
return (B^{-1} = V)

Der Erweiterte Euklidische Algorithmus eignet sich besonders gut, um in Hardware implementiert zu werden. Das am besten geeignete Verfahren wurde von Brunner, Curiger und Hofstetter entwickelt und ist in [9] nachzulesen. Dieses Verfahren benötigt *m* Takte bei einer *m* Bit breiten Zahl und hat dabei den geringsten Flächenbedarf, allerdings ist sie nur für den endlichen Körper GF(2^m) auf Polynombasis geeignet. Außerdem lässt sich diese Hardwarelösung sehr gut parallelisieren und kann damit auch in weniger als *m* Takten berechnet werden.

Bei einer seriellen Hardwarelösung ist somit die Berechnung des modular Inversen genauso schnell, wie die Berechnung der Multiplikation. Durch die Parallelisierbarkeit des Verfahrens kann es auch bei einer parallelisierten Hardwarelösung eingesetzt werden.

3.2 Fermat'sches Theorem

Neben dem Erweiterten Euklidischen Algorithmus gibt es noch eine weitere Möglichkeit das modular Inverse zu berechnen. Diese Möglichkeit stellt das Fermat'sche Theorem dar. Das Fermatsche Theorem besagt, wenn *n* eine Primzahl ist, dann gilt:

$$a^n \bmod n = a \tag{15}$$

Wenn auf beiden Seite durch a^2 dividiert wird, erhält man die Formel für das modular Inverse:

$$a^{n-2} \bmod n = a^{-1} \tag{16}$$

Die Berechnung des modular Inversen kann somit durch Multiplikationen und Quadrierungen berechnet werden. Daher eignet sich diese Methode nur, wenn das modular Inverse selten

benötigt wird. Was der Fall ist, wenn die Skalarmultiplikation über die projektive Ebene berechnet wird. Denn dabei wird das modular Inverse nur zur Rücktransformation in affine Koordinaten benötigt.

Das Verfahren nach dem Fermat'schen Theorem mit den wenigsten Multiplikationen und Quadrierungen wurden von Itoh und Tsujii entwickelt [10]. Für eine m Bit breite Zahl werden die folgende Anzahl Multiplikationen und Quadrierungen benötigt:

$$\begin{aligned} &\text{Multiplikationen:} && \lfloor \log_2(m-1) \rfloor + H_w(m-1) - 1 \\ &\text{Quadrierungen:} && m-1 \end{aligned} \tag{17}$$

H_w stellt wieder das Hamminggewicht von (m-1) dar.

4 VERGLEICH DER ALGORITHMEN

Nachdem nun die verschiedenen Methoden zur Berechnung des modular Inversen vorgestellt wurden, soll jetzt eine Gegenüberstellung der verschiedenen Algorithmen erfolgen. Dazu sind in Tabelle 1 die beiden Algorithmen für die Skalarmultiplikation, der "Double and Add" Algorithmus und der Montgomery Algorithmus, jeweils über die affine und projektive Ebene dargestellt. Zu diesen Algorithmen sind jeweils die Anzahl an Operationen (Additionen, Multiplikationen, Quadrierungen und Invertierungen) im endlichen Körper GF(2^m) für die Punktaddition, die Punktverdopplung und die Skalarmultiplikation angegeben.

Algorithmus		Affine Ebene P(x, y)		Projektive Ebene P(X, Y, Z)	
		Double and Add	Montgomery	Double and Add	Montgomery
$P_1 + P_2$	+	8	3	7	2
	*	2	1	15	4
	x^2	2	1	7	1
	x^{-1}	1	1	0	0
$2 * P$	+	4	1	4	1
	*	3	1	5	2
	x^2	2	1	7	4
	x^{-1}	1	1	0	0
$k * P$	+	$8h + 4m$	$4m + 6$	$7h + 4m$	$3m + 7$
	*	$2h + 3m$	$2m + 4$	$15h + 5m + 3$	$6m + 10$
	x^2	$2h + 2m$	$2m + 2$	$7h + 7m + 1$	$5m + 3$
	x^{-1}	$h + m$	$2m + 1$	1	1

$m = \log_2 k$ (Bitbreite von k); $h = m/2$ im Durchschnitt (Hamminggewicht von k)

Tabelle 1: Vergleich der verschiedenen Algorithmen

In der vorangegangenen Tabelle ist sehr gut zu erkenne, dass der "Double and Add" Algorithmus abhängig vom Hamminggewicht von k ist. Des weiteren benötigten die Algorithmen über die projektive Ebene nur eine einzige Invertierung.

5 ERGEBNISSE

Aus dem Vergleich der verschiedenen Algorithmen ergibt sich dann folgendes Ergebnis:

- Für eine Softwareimplementierung eignet sich der Montgomery Algorithmus über die projektive Ebene unter Verwendung des Verfahrens von Itoh und Tsujii [10] für die Berechnung des modular Inversen am besten.

- Bei einer Hardwareimplementierung ist der Montgomery Algorithmus über die affine Ebene die beste Wahl, wobei das modular Inverse nach dem Verfahren von Brunner, Curiger und Hofstetter [9] berechnet wird.

Um dies noch einmal zu verdeutlichen, sei hier ein Beispiel für eine serielle Hardwarelösung bei einer Bitbreite von 167 Bit aufgeführt. Die Invertierung für die affine Ebene wird nach dem Verfahren von Brunner, Curiger und Hofstetter und über die projektive Ebene nach dem Verfahren von Itoh und Tsujii berechnet.

Anzahl Takte für die Operationen im endlichen Körper von $GF(2^m)$ mit m=167, serielle Hardwarelösung:

- Addition: 1 Takt
- Multiplikation: m Takte
- Quadrierung: m/2 Takte
- Invertierung: m Takte (affine Ebene – Brunner, Curiger, Hofstetter [9])
  $(\lfloor \log_2(m-1) \rfloor + H_w(m-1))*m + (m-1)*(m/2)$ (hier: 15614 Takte)
  (projektive Ebene – Itoh, Tsujii [10])

Algorithmus	Affine Ebene P(x, y)		Projektive Ebene P(X, Y, Z)	
	Double and Add	Montgomery	Double and Add	Montgomery
Anzahl Takte	197148	141289	514908	255518
Prozentual	77 %	55 %	202 %	100 %

Tabelle 2: Anzahl Takte für eine serielle Hardwarelösung

Wie in Tabelle 2 sehr gut zu erkennen ist, benötigt der Montgomery-Algorithmus über die affine Ebene die wenigsten Takte, womit die vorhergehende Aussage über die Wahl der Algorithmen bestätigt wird. In der Tabelle wurde der Montgomery-Algorithmus über die projektive Ebene als Referenz herangezogen, weil die aktuell schnellste Hardwarelösung [11] diesen Algorithmus verwendet. Bei diesem seriellen Beispiel könnte also durch die Wahl eines geeigneteren Algorithmus eine Geschwindigkeitssteigerung von etwa 45% erzielt werden. Orlando und Paar haben allerdings eine teilweise parallelisierte Hardware verwendet. Bei gleicher Parallelisierungsstufe könnte durch die Wahl des Montgomery-Algorithmus über die affine Ebene immerhin noch eine Geschwindigkeitssteigerung von 38% erzielt werden.

6 ZUSAMMENFASSUNG

Nach einer kurzen Einführung in die Funktionsweise der Elliptic Curve Cryptography wurden die verschiedenen Algorithmen für die Berechnung im endlichem Körper GF(2^m) aufgeführt. Dabei hat sich herauskristallisiert, dass die Wahl des Algorithmus von der Berechnung des modular Inversen abhängt. Daher wurden die Möglichkeiten für die Berechnung des modular Inversen eingehender untersucht und Lösungsvorschläge aufgezeigt. Mit diesem Wissen ist dann eine optimale Algorithmenwahl möglich. Anhand eines Beispiels wurde die Wahl noch einmal veranschaulicht.

Dieser Beitrag zeigt die verschiedenen Algorithmen für die Elliptic Curve Cryptography über einen endlichen Körper von GF(2^m) auf und verglicht ihre Performance. Damit stellt dieser Beitrag ein Hilfsmittel für die Entwickler von sicherheitsrelevanter Applikationen dar, wenn sie sich dafür entschieden haben, die ECC einzusetzen.

LITERATUR

[1] A. K. Lenstra, E. R. Verheul, "Selecting Cryptographic Key Sizes," 3rd workshop on Elliptic Curve Cryptography (ECC '99), 1999.

[2] R. L. Rivest, A. Shamir, L. Adleman, "A Method of obtaining digital signature and public key cryptosystems," Comm. of ACM, Vol. 21, No. 2, pp. 120-146,1978.

[3] N. Koblitz, "Elliptic curve cryptosystems," Mathematics of Computation, vol. 48, no. 177, pp. 203-209, 1987.

[4] V. Miller, "Use of elliptic curves in cryptography," Advances in Cryptology CRYPTO '85, LNCS 218, pp. 417-426, Springer Verlag, 1986.

[5] A. Menezes, "Elliptic Curve Public Key Cryptosystems," Kluwer Academic Publishers, 1993.

[6] IEEE P1363: "Standard Specifications for Public Key Cryptography," Draft Version 13, 1999.

[7] P. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization," Mathematics of Computation, vol. 48, pp. 243-264, 1987.

[8] J. Lopez, R. Dahab, "Fast Multiplication on Elliptic Curves over GF(2^m) without Precomputation," Workshop on Cryptographic Hardware and Embedded Systems (CHES '99), LNCS 1717, Springer Verlag, 1999.

[9] H. Brunner, A. Curiger, M. Hofstetter, "On Computing Multiplicative Inverses in GF(2^m)," IEEE Transactions on Computation, vol. 42, pp. 1010-1015, Aug. 1993.

[10] T. Itoh, S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses in GF(2^m) using normal bases," Information on Computation, vol. 78, pp. 171-177, 1988.

[11] G. Orlando, C. Paar, "A High-Performance Reconfigurable Elliptic Curve Processor for GF(2^m)," Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000), Springer Verlag, LNCS 1965, 2000.