ElGamal-Signaturen mit elliptischen Kurven

Diplomarbeit

von

Oliver Ehli

nach einem Thema von
Prof. Dr. Johannes Buchmann
Fachgebiet Theoretische Informatik
Fachbereich Informatik
Technische Universität Darmstadt

18. Juni 2002

Hiermit erkläre ich an Eides statt, daß ich die vorliegende Arbeit selbstständig und ausschließlich mit den angegebenen Hilfsmitteln angefertigt habe.

Darmstadt, den 18. Juni 2002

Oliver Ehli

Zuerst möchte ich mich bei Prof. Buchmann für das interessante Thema dieser Diplomarbeit bedanken.

Bei meinem Betreuer Bodo Möller bedanke ich mich für die vielen Anregungen und Hinweise auf Ungenauigkeiten; ich habe in dieser Zeit viel gelernt.

Für das Korrekturlesen der verschiedenen Stadien dieser Arbeit möchte ich mich bei meinem Bruder Torsten, bei meiner Mutter Renate und bei Florian Nowotny bedanken.

Ganz besonders möchte ich mich bei meiner Verlobten Susanne Loose bedanken. Ohne ihre Unterstützung, während dieser Arbeit und auch während der gesamten Diplomprüfungsphase, wäre ich nicht, wo ich jetzt bin.

Oliver Ehli

Inhaltsverzeichnis

1 Einleitung **1**

2 Digitale Signaturen **3**
2.1 Übersicht . . . 3
2.1.1 RSA-Signaturverfahren . . . 3
2.1.2 ElGamal-Signaturverfahren . . . 4
2.2 Hashfunktionen . . . 4
2.3 Angriffe . . . 5
2.4 Beweisbare Sicherheit . . . 6
2.4.1 Generic-Group-Modell . . . 7
2.4.2 Random-Oracle-Modell . . . 8

3 Komplexitätstheorie **11**
3.1 Probabilistische Turingmaschinen . . . 11
3.1.1 Münzwurfmaschinen . . . 12
3.1.2 Interaktive Turingmaschinen . . . 13
3.2 Interaktive Beweissysteme . . . 15
3.2.1 Vergleich von Wahrscheinlichkeitsverteilungen . . . 17
3.2.2 Zero-Knowledge-Beweise . . . 18

4 Elliptische Kurven **25**
4.1 Einführung . . . 25
4.2 Verallgemeinerung und andere Gruppen . . . 27
4.3 Elliptische Kurven über endliche Körper . . . 31
4.3.1 Elliptische Kurven über $\mathbb{F}_p$ . . . 32
4.3.2 Elliptische Kurven über $\mathbb{F}_{2^m}$ . . . 33
4.4 ECDLP . . . 37
4.4.1 Angriffe auf elliptische Kurven . . . 37
4.4.2 Schwache Kurven . . . 39

5 ElGamal-Signaturen **41**
5.1 ElGamal Signatur . . . 41
5.2 Verallgemeinerung für beliebige Gruppen . . . 41
5.2.1 Schlüsselerzeugung . . . 41
5.2.2 Signaturerzeugung . . . 42
5.2.3 Signaturverifikation . . . 42
5.3 Meta-ElGamal Signatur . . . 43
5.4 ElGamal Signaturen mit elliptischen Kurven . . . 44
5.5 Sicherheit und Effizienz . . . 46

6 ElGamal-Signaturen und Random Oracles **47**
6.1 Oracle Replay Attack . . . 47
6.2 ElGamal-Signatur Typ EG-I Nr.1 . . . 49
6.2.1 No-Message-Attack . . . 49
6.2.2 Adaptively-Chosen-Message-Attack . . . 50
6.3 ElGamal-Signatur Typ EG-II Nr. 3 . . . 52
6.3.1 No-Message-Attack . . . 52
6.3.2 Adaptively-Chosen-Message-Attack . . . 52

6.4 Bemerkungen . . . 54
6.4.1 Meta-ElGamal-Signatur . . . 54
6.4.2 Hashfunktionen und Random Oracles . . . 55
6.4.3 Die Funktion h_2 und ECDSA . . . 57

7 EC-ElGamal und SEC **61**
7.1 Übersicht . . . 61
7.2 Struktur der Standards for Efficient Cryptography . . . 61
7.3 EC-ElGamal . . . 62
7.4 Datentypen . . . 63
7.4.1 Bit-Strings und Octet-Strings . . . 63
7.4.2 Integers und Octet-Strings . . . 64
7.4.3 Körperelemente und Octet-Strings . . . 65
7.4.4 EC-Punkte und Octet-Strings . . . 66
7.5 Kryptographische Komponenten . . . 68
7.5.1 EC-Domain-Parameter . . . 68
7.5.2 EC-Domain-Parameter Generierung . . . 69
7.5.3 EC-Domain-Parameter Validierung . . . 71
7.5.4 Generierung des Schlüsselpaares . . . 72
7.5.5 Validierung des öffentlichen Schlüssels . . . 73
7.6 EC-ElGamal in SEC . . . 74
7.6.1 Konfiguration . . . 74
7.6.2 Schlüsselaustausch . . . 74
7.6.3 Erzeugen einer Signatur . . . 75
7.6.4 Verifizieren einer Signatur . . . 76
7.7 Bemerkungen . . . 77
7.8 ASN.1 Syntax . . . 77
7.8.1 Syntax für endliche Körper . . . 78
7.8.2 Syntax für EC-Domain-Parameter . . . 80
7.8.3 Syntax für den öffentlichen Schlüssel . . . 81
7.8.4 Syntax für den privaten Schlüssel . . . 82
7.8.5 Syntax für die EC-ElGamal Signatur . . . 82

8 Zusammenfassung **85**

A SEC2: EC-Domain-Parameter **87**
A.1 Elliptische Kurven über $\mathbb{F}_p$ . . . 87
A.2 Elliptische Kurven über $\mathbb{F}_{2^m}$ . . . 91
A.3 ASN.1 Syntax . . . 96

Literatur **97**

1 Einleitung

In der vorliegenden Arbeit werden ElGamal-Signaturverfahren über elliptische Kurven besprochen und deren Sicherheit im Random-Oracle-Modell betrachtet. Diese Verfahren stellen eine Alternative zu dem bekannten und standardisierten ECDSA dar. Anschliessend wird eines der vorgestellten Verfahren in den bereits standardisierten Rahmen der Kryptographie mit elliptischen Kurven integriert.

Zunächst werden die wichtigsten Begriffe und Verfahren bzw. Konzepte vorgestellt, welche die Grundlage zur dann folgenden Analyse bilden. Aufgrund des damit einhergehenden Umfangs kann dies nur in Ausschnitten geschehen, weswegen an den entsprechenden Stellen auf Hintergrundliteratur verwiesen wird. Es handelt sich also eher um eine Wiederholung oder Zusammenfassung, die auch auf die Angabe von Beweisen verzichtet. Im Folgenden wird der Inhalt der einzelnen Kapitel kurz vorgestellt.

In Kapitel 2 (*Digitale Signaturen*) wird kurz auf unterschiedliche Konzepte von Signaturverfahren eingegangen, grundlegende Begriffe eingeführt, sowie Ansätze, die Sicherheit der Verfahren zu untersuchen, vorgestellt.

Dabei werden die klassischen RSA und ElGamal Signaturverfahren betrachtet, sowie die Notwendigkeit der Verwendung von Hashfunktionen erläutert. Anschliessend werden Angriffe gegen die Verfahren besprochen und kategorisiert. Schliesslich werden zwei Modelle, das Generic-Group-Modell und das Random-Oracle-Modell, vorgestellt, mit denen sich Aussagen über die Sicherheit von Signaturverfahren treffen lassen. Diesen Modellen werden Signaturverfahren gegenübergestellt, deren Sicherheit direkt beweisbar ist.

Kapitel 3 (*Komplexitätstheorie*) behandelt die benötigten Begriffe und Grundlagen, auf denen das Random-Oracle-Modell aufbaut.

Es werden unter anderem probabilistische Turingmaschinen und interaktive Beweissysteme vorgestellt sowie die Zero-Knowledge-Eigenschaft besprochen. Der Zusammenhang zwischen Zero-Knowledge-Beweisen und Signaturverfahren sowie die Möglichkeit zu untersuchen, ob ein Beweissystem die Zero-Knowledge-Eigenschaft besitzt, bildet den Kern der später vorgenommenen Analysen im Random-Oracle-Modell.

Kapitel 4 (*Elliptische Kurven*) stellt einen kleinen Ausschnitt der Theorie und Arithmetik der elliptischen Kurven dar.

Elliptische Kurven werden dazu klassisch mit komplexen Zahlen eingeführt und dann später auf endliche Körper übertragen. An dieser Stelle wird auch in äusserst kompakter Form auf die Arithmetik der verwendeten endlichen Körper eingegangen. Anschliessend wird das Problem des diskreten Logarithmus für elliptische Kurven und Algorithmen diesen zu berechnen vorgestellt. Abschliessend werden für kryptographische Zwecke ungeeignete Kurven beschrieben.

Im Kapitel 5 (*ElGamal Signaturen*) wird genauer auf das Signaturverfahren von Ther ElGamal eingegangen, Varianten vorgestellt und diese schliesslich auf elliptische Kurven übertragen.

In Kapitel 6 (*ElGamal Signaturen im Random-Oracle-Modell*) wird zuerst das Random-Oracle-Modell detaillierter erläutert. Schliesslich werden zwei Va-

rianten des ElGamal-Signaturverfahrens mit Pointchevals Forking Lemma untersucht. Danach wird kurz auf den Zusammenhang mit ECDSA eingegangen, dessen Sicherheit sich nicht im Random-Oracle-Modell beweisen lässt.

Das Kapitel 7 (*EC-ElGamal in SEC*) gibt zunächst eine kompakte Übersicht einiger existierender Standards zur Kryptographie mit elliptischen Kurven. Anschließend werden die einzelnen benötigten Komponenten, die in den Standards for efficient Cryptography (SEC) festgelegt wurden, wiedergegeben. Danach folgen die Verfahren zur Umsetzung des ElGamal Signaturverfahrens über elliptische Kurven. Zum Schluss wird die Beschreibung sämtlicher Komponenten in ASN.1 Syntax besprochen.

Die *Zusammenfassung* wiederholt die Ergebnisse und der Anhang enthält schliesslich die in SEC2 standardisierten elliptischen Kurven und ihre Syntax.

2 Digitale Signaturen

Die Problematik, eine digitale Äquivalenz zu klassischen Unterschriften zu ermöglichen, wurde erst mit der Entwicklung asymetrischer, kryptographischen Verfahren lösbar, deren Prinzip 1975 in [DH76] von Whitfield Diffie und Martin Hellman vorgestellt wurden und die Grundlage der gesamten Public-Key-Kryptographie bildet. Im Gegensatz zu den bis dato ausschließlich verwendeten symmetrischen Verschlüsselungsverfahren, die nur einen Schlüssel verwenden, werden nun zwei Schlüssel verwendet, von denen einer jedem öffentlich zugänglich ist und der deswegen *öffentlicher Schlüssel* (Public Key) heißt, während der andere geheim gehalten und *privater Schlüssel* (Private Key) genannt wird. Damit kann nun jeder, der den öffentlichen Schlüssel einer Person kennt, mit diesem eine Nachricht verschlüsseln, die nur der Besitzer des privaten Schlüssels entschlüsseln kann. Damit fällt zum einen das Problem des Schlüsselaustauschs weg. Zum anderen kann aus einem solchen Verschlüsselungsverfahren auch ein Signaturverfahren konstruiert werden, bei dem der private Schlüssel zur Signaturerzeugung und der öffentliche Schlüssel zur Signaturverifikation verwendet wird. Ist es nun praktisch unmöglich den privaten Schlüssel aus dem öffentlichen Schlüssel und den signierten Nachrichten zu berechnen oder aber anderweitig korrekt signierte Nachrichten zu erzeugen, so ist damit die Authentizität der Nachricht und ebenso die Identität des Absenders festgelegt.

In den folgenden Abschnitten werden sowohl Signaturverfahren, als auch Definitionen und Verfahren vorgestellt, welche die Sicherheit des privaten Schlüssels und die Resistenz eines Signaturverfahrens gegenüber der Fälschung von Nachrichten beschreiben.

2.1 Übersicht

Eines der angesprochenen Attribute ist die Sicherheit des privaten Schlüssels. Typischerweise wird in Public-Key-Verfahren ein privater Schlüssel zufällig gewählt und daraus der öffentliche Schlüssel berechnet. Die Umkehrung, also den privaten Schlüssel aus dem öffentlichen zu bestimmen, muss dann praktisch unmöglich sein, weswegen in diesem Fall auch von einer *Einweg-Funktion* gesprochen wird. Die heute verwendeten Verfahren basieren auf bisher ungelösten mathematischen Problemen, von denen wiederum die Faktorisierung großer Zahlen das bekannteste ist. Zwei Signaturverfahren und die Probleme, auf denen diese basieren, werden nun in ihrer ursprünglichen Form vorgestellt.

2.1.1 RSA-Signaturverfahren

Das erste, und immer noch populärste, Public-Key-Verfahren ist das, nach seinen Entwicklern Rivest, Shamir und Adleman benannte und bereits 1977 vorgestellte, RSA-Verfahren [RSA78], das wie folgt funktioniert. Zuerst werden zwei große Primzahlen p und q zufällig gewählt und daraus das *RSA-Modul* $n = pq$ berechnet. Dann werden zwei zufällige Zahlen d und e aus dem Breich $1 < d, e < (p-1)(q-1)$ gewählt, welche beide prim zu $(p-1)(q-1)$ sein und die Bedingung $de \equiv 1 \bmod (p-1)(q-1)$ erfüllen müssen. Der private Schlüssel ist d und der öffentliche Schlüssel (n, e). In der Praxis wird für den Exponnenten e ein fester Wert, etwa $e = 2^{16} + 1$ gewählt.

Der private Schlüssel ist also einfach zu bestimmen, wenn die Faktorisierung von n bekannt ist. Umgekehrt kann auch gezeigt werden, dass sich mit bekanntem d, e und n die Primfaktoren p und q bestimmen lassen. Da die Primfaktoren und der private Schlüssel geheim gehalten werden, muss daher n faktorisiert werden um d zu erhalten, was für ein entsprechend großes n, etwa $|n| = \lfloor \log_2 n \rfloor + 1 \geq 1024$, momentan nicht praktisch möglich ist.

Die Signatur s einer Nachricht $m < n$ wird mit $s = m^d \bmod n$ berechnet. Sie kann verifiziert werden, indem $m = s^e \bmod n$ berechnet wird, weil die Bedingung $de \equiv 1 \bmod (p-1)(q-1)$ erfüllt ist, und deswegeen $m \equiv m^{de} \bmod n$ gilt.

2.1.2 ElGamal-Signaturverfahren

Im Jahre 1984 wurde in [ElG85] ein weiteres Verfahren von Taher ElGamal vorgestellt, dessen Sicherheit auf der Schwierigkeit, den diskreten Logarithmus in Primzahlkörpern zu berechnen, basiert. Mit einer großen Primzahl p und einer Primitivwurzel $g \bmod p$ ist der private Schlüssel a eine zufällig gewählte Zahl aus der Menge $\{1, \ldots, p-2\}$, mit welcher die Zahl y durch $y = g^a \bmod p$ berechnet wird. Der öffentliche Schlüssel besteht dann aus dem Tripel (p, g, y). Um die Signatur einer Nachricht $m < p$ berechnen zu können, wird für jede Signatur eine neue zufällig gewählte Zahl $k \in \{1, \ldots, p-2\}$ benötigt. Dieses k muss die Bedingung $\gcd(k, p-1) = 1$ erfüllen, so dass $k^{-1} \bmod (p-1)$ existiert. Mit diesem k wird das Element $r = g^k \bmod p$ berechnet, das einen Teil der Signatur (r, s) einer Nachricht $m < p-1$ darstellt. Der zweite Bestandteil wird mit $s = k^{-1}(m - ar) \bmod (p-1)$ berechnet. Eine Signatur ist gültig, wenn die Bedingung $1 < r < p-1$ und die Kongruenz $y^r r^s \equiv g^m \bmod p$ erfüllt sind.

2.2 Hashfunktionen

In der Praxis werden die vorgestellten Signaturverfahren nicht in dieser Form verwendet, denn es ist einfach möglich, Signaturen zu fälschen. Für das ElGamal-Signaturverfahren werden dazu zwei zufällige Zahlen $v, w \in \mathbb{Z}^*_{p-1}$ benötigt, wobei $\gcd(w, p-1)$ erfüllt sein muss. Mit diesen wird ein $r = g^v y^w \bmod p$ berechnet, woraus sich die Signatur $s = -rw^{-1} \bmod (p-1)$ der Nachricht $m = sv \bmod (p-1)$ ergibt.

Um diese Form der Fälschung zu verhindern, werden kryptographische Hashfunktionen verwendet und nicht die Nachricht selbst signiert. (Kryptographische) Hashfunktionen sind Abbildungen der Form $h : \Sigma^* \to \Sigma^n$, welche also Strings beliebiger Länge auf Strings fester Länge abbilden. Eine derart definierte Funktion besitzt offensichtlich Kollisionen, was bedeutet, dass es eine Nachricht m und eine weitere Nachricht m' gibt, für welche die Hashfunktion einen identischen Haswert erzeugt, so dass also die Bedingung $h(m) = h(m')$ erfüllt ist. Ist es praktisch unmöglich zu einer gegebenen Nachricht m die Nachricht m' zu finden, für die diese Bedingung erfüllt ist, so heißt die Hashfunktion *schwach kollisionsresistent* (second-preimage resistant). Ist es dagegen praktisch unmöglich irgendein Paar (m, m') mit $m \neq m'$zu finden, für das $h(m) = h(m')$ gilt, heißt die Hashfunktion *stark kollisionsresistent*. Lediglich die Hashfunktionen, welche als stark kollisionsresistent angenommen werden, sind für Signaturverfahren geeignet, denn bei diesen kann davon ausgegangen werden, dass tatsächlich die Nachricht m signiert wurde. Es ist dann üblich, das Attribut

„stark“ zu vernachlässigen und stattdessen einfach von Kollisionsresistenz zu sprechen. Eine weitere Bedingung die an Hashfunktionen gestellt wird ist, dass es praktisch unmöglich sein sollte, zu einem gegebenen Hashwert x eine Nachricht m zu finden, für die $h(m) = x$ erfüllt ist. Es kann gezeigt werden, dass jede kollisionsresistente Funktion diese Bedingung erfüllt.

In der Praxis werden zur Zeit zwei standardisierte Hasfunktionen, die beide Ausgaben der Länge 160 Bit erzeugen, als kollisionsresistent betrachtet und für Signaturverfahren verwendet: SHA1 (Secure Hash Algorithm) [SHA1] und RIPEMD-160 [DBP96, ISO98]. Die Hasfunktion MD5 (Message Digest 5), welche noch häufig in Verbindung mit RSA verwendet wird, und insbesondere der Vorgänger MD4 können nicht mehr als sicher betrachtet werden.[Dob98, Dob96, dBB93]

Neue Hashfunktionen, SHA-256 und SHA-512, wurden bereits entwickelt und befinden sich in der Endphase der Standardisierung[SHA2]. Eine gute Übersicht über Hashfunktionen, die allerdings nicht die aktuellen Funktionen enthält, ist mit [BSP95] gegeben.

2.3 Angriffe

Die beschriebenen Signaturverfahren müssen Angriffen auf verschiedenen Ebenen widerstehen können. Zuerst muss die verwendete Hashfunktion, wie bereits erwähnt, kollisionsresistent und nicht invertierbar sein. Außerdem wäre jedes der beschriebenen Verfahren auch unsicher, wenn das zugrundeliegende mathematische Problem plötzlich doch lösbar ist. Die Vorausetzung bei einer Sicherheitsanalyse asymetrischer Verschlüsselungs- oder Signaturverfahren ist daher, dass einerseits die Hashfunktionen die gewünschten Eigenschaften haben und andererseits das zugrundeliegende mathematische Problem tatsächlich schwer ist. Schließlich kann auch das eigentliche Signaturverfahren schlecht konstruiert sein, so dass es deutlich einfacher ist, in dem Verfahren Signaturen zu fälschen oder den privaten Schlüssel zu bestimmen, als das zugrundeliegende Problem, etwa den diskreten Logarithmus, zu lösen.

Die möglichen Resultate eines Angriffs auf ein Signaturverfahren können formal beschrieben werden. Da verschiedene Resultate aber unterschiedlich schwerwiegend sind, gibt es dementsprechend unterschiedliche Begriffe für die Resistenz eines Signaturverfahrens gegen einen Angriff mit eben diesem Resultat.

Das schwerwiegenste Resultat einer Attacke auf ein Signaturverfahren ist, wenn es dem Angreifer gelingt, den privaten Schlüssel zu bestimmen. In diesem Fall ist das Verfahren offensichtlich unbrauchbar. Dieser Erfolg heißt *total break*.

Das Verfahren ist ebenfalls unbrauchbar, wenn eine Attacke einen Algorithmus hervorbringt, der das Signieren jeder beliebig gewählten Nachricht ermöglicht. Dies wird als *universelle Fälschung* (universal forgery) bezeichnet.

Wenn nun zwar nicht jede beliebige, dafür aber frei wählbare Nachrichten signiert werden können, die einer bestimmten Form genügen, ist für das Verfahren die *selektive Fälschung* (selective forgery) möglich. Das Verfahren ist dann nur noch bedingt verwendbar.

Das minimale[1] Resultat einer erfolgreichen Attacke, ist lediglich eine Signatur einer weiteren Nachricht zu erzeugen, die der Angreifer nicht gezielt auswählen kann, was im vorangegangenen Abschnitt anhand des ursprünglichen ElGamal-Signaturverfahrens demonstriert wurde. Ein solches Resultat heißt *existentielle Fälschung* (existential forgery). Die derart erzeugte Nachricht erfüllt zwar möglicherweise keine vorgegebene oder erwartete Struktur (beispielsweise natürliche Sprache), aber es ist nicht mehr eindeutig sicher, dass eine Signatur wirklich für eine spezielle Nachricht erzeugt wurde.

Auch die Angreifer lassen sich kategorisieren. So ist ein Angreifer, dem außer dem öffentlichen Schlüssel noch eine Sammlung von Signaturen und den dazugehörenden Nachrichten zur Verfügung steht, als mächtiger oder gefährlicher anzusehen, als ein Angreifer, der ausschließlich den öffentlichen Schlüssel besitzt. Das Ziel eines Angreifers ist eines der aufgelisteten Resultate und es wird natürlich in allen Fällen vorausgesetzt, dass der Angreifer das verwendete Verfahren kennt.

Einem *No-Message-Angreifer* steht für seinen Angriff lediglich der öffentliche Schlüssel zur Verfügung.

Einem *Known-Message-Angreifer* steht zusätzlich zum öffentlichen Schlüssel noch eine Anzahl von Nachricht/Signaturpaaren zur Verfügung, die dieser allerdings nicht auswählen konnte.

Ein *Chosen-Message-Angreifer* darf die Liste der Nachricht/Signaturpaare auswählen. Es wird dabei noch zwischen einem *Generic-Chosen-Message-Angreifer* und einem *Oriented-Chosen-Message-Angreifer* unterschieden, wobei der letztere seine Liste in Abhängigkeit des öffentlichen Schlüssels wählen darf. Ersterer muss dieses tun, bevor er den öffentlichen Schlüssel erhält.

Schließlich darf ein *Adaptively-Chosen-Message-Angreifer* jedes seiner Nachricht/Signaturpaare sowohl in Abhängigkeit des öffentlichen Schlüssels als auch basierend auf der Kenntnis der zuvor gewählten und erhaltenen Paare aussuchen.

Im Allgemeinen heißt ein Verfahren *sicher*, wenn selbst einem Adaptively-Chosen-Message-Angreifer die existentielle Fälschung praktisch unmöglich ist. Dies entspricht dem stärksten vorstellbaren Angreifer, der das minimale Ziel nicht erreichen kann.

2.4 Beweisbare Sicherheit

Um Aussagen über die Sicherheit kryptographischer Protokolle und Algorithmen treffen zu können, wird in der Regel ein Verfahren zuerst vorgeschlagen und anschließend getestet. Ein aktuelles Beispiel dafür ist mit *AES* (Advanced Encryption Standard) [AES01, DRj01] gegeben, dessen Entwicklung mit einem

[1] Formal kann auch die Möglichkeit weitere gültige Signaturen einer signierten Nachricht zu erzeugen als (erfolgreicher) Angriff betrachtet werden, was aber üblicherweise nicht getan wird.

„Call for Algorithms“ im Januar 1997 begann. Es kristallisierten sich, insbesondere im Zusammenhang mit *DES (Data Encryption Standard)* [DES99], verschiedene Formen der Kryptoanalyse heraus, sowie Verfahren, die Resistenz gegen die resultierenden Attacken zu beweisen [Vau98], um ungeeignete Algorithmen von vorneherein ausschließen zu können. Beispiele für Verfahren, die nicht öffentlich untersucht wurden und Probleme und Schwächen besitzen, die während der Entwicklung nicht entdeckt wurden, sind RC4 und das darauf basierenden WEP (Wireless equals Privacy) [FMS01].

In der Public-Key-Kryptographie wird den Verfahren ein bekanntes, schwer lösbares mathematisches Problem zugrunde gelegt. Attacken sind in diesem Fall sowohl gegen das eigentliche Protokoll als auch gegen das mathematische Problem möglich. Wie bei den symetrischen Verfahren haben sich verschiedene Angriffe etabliert, und es existieren Methoden, um den möglichen Erfolg dieser Angriffe zu untersuchen. Zwei für die vorliegende Arbeit bedeutsame Modelle werden nun kurz angesprochen. Beide Modelle können außerdem kombiniert werden. Auf diese Weise wird etwa in [ScJ98] die Sicherheit eines Signatur- und eines Verschlüsselungsverfahrens bewiesen, die beide auf dem Problem des diskreten Logarithmus basieren.

Resultate in diesen Modellen sind jedoch keineswegs vollständige Beweise für die uneingeschränkte Sicherheit eines untersuchten Protokolls, sondern lediglich Sicherheitsargumente für den Kern der Verfahren, und sollten auch also solche aufgefasst werden. Es können zudem weitere Schwächen existieren, wie unter anderem [Ble96, Vau96, NSh00] verdeutlichen.

Neben diesen Reduktionsbeweisen existieren auch Verfahren, deren Sicherheit ‘klassisch’, relativ zu dem zugrunde liegenden mathematischen Problem, bewiesen werden kann. Ein auf RSA basierendes Signaturverfahren dieser Kategorie, das bedingt praktisch verwendbar ist, wird in [DwN94] vorgestellt und heißt Dwork-Naor-Signaturverfahren. In diesem Verfahren wird ein Baum mit bereits erzeugten Signaturen verwaltet, sowie öffentliche Listen von zuvor festgelegten Primzahlen und Zufallszahlen zum Erzeugen von Signaturen verwendet. Für das Verfahren mit diesen Statusinformationen wird dann die Sicherheit bewiesen.

Weitere auf RSA basierende Verfahren, die allerdings ohne Statusinformationen auskommen, werden in [GHR99] und [CrS99] besprochen. In beiden Arbeiten wird die Sicherheit unter der *Strong RSA Assumption*, welche in [BaP97] definiert wurde, bewiesen. In [GHR99] werden aber zusätzliche Annahmen über die verwendeten Hashfunktionen gemacht, die letztlich dazu führen, dass diese lediglich Primzahlen als Ausgabe liefern dürfen.

2.4.1 Generic-Group-Modell

Im *Generic-Group-Modell* wird der zentrale Begriff einer *idealen Gruppe* definiert. Für eine Attacke auf ein kryptographisches System erhält ein Angreifer nicht die binäre Repräsentation der Elemente dieser Gruppe, sondern darf lediglich Gruppenoperationen durch ein *Group Oracle* ausführen lassen. Mit dieser Vorgabe werden nun *generische Algorithmen* untersucht, welche ein identisches Problem, etwa den diskreten Logarithmus, in beliebigen Gruppen lösen sollen. Zu dieser Klasse von Algorithmen gehören der Pohlig-Hellmann- und der Pollard-ρ-Algorithmus, nicht jedoch die Index-Calculus-Methode (Eine Beschreibung dieser Verfahren ist in [Buc00, MOV97] zu finden). In [Sho97] wird

eine untere Grenze für die Berechnung des diskreten Logarithmus mit generischen Algorithmen gefunden und bewiesen. Methoden, die diese Grenze unterschreiten, müssen daher zusätzliche Informationen über die Repräsentation der Gruppenelemente verwenden. Derartige Algorithmen sind für die üblicherweise in der Kryptographie verwendeten elliptischen Kurven bisher nicht bekannt und das effizienteste Verfahren ist momentan der parallelisierte Pollard-ρ-Algorithmus [vOW94, WiZ98].

Die Sicherheit von ECDSA [JMV99, P1363, X9.62] wurde mit Hilfe des Generic-Group-Modells in [Bro00] bewiesen, wobei dieses Resultat aber nicht auf den ursprünglichen DSA übertragen werden kann. Die Autoren argumentieren, dass ein Angreifer, der Signaturen fälschen kann, die verwendete Hashfunktion attackiert haben muss, also entweder diese invertiert oder Kollisionen gefunden hat. Wird nun aber ein effizienter, nicht generischer Algorithmus für die verwendete Gruppe gefunden, so muss lediglich diese nun unsichere Gruppe durch eine sichere ersetzt werden, für die (noch) ausschließlich generische Algorithmen existieren. Die Sicherheit oder Korrektheit des Protokolls ist davon unabhängig.

In [Fis00] wurde jedoch gezeigt, dass diese Argumentation einige Probleme mit sich bringen kann, was zunächst an einem einfachen Beispiel beschrieben und dannach an mehreren Protokollen demonstriert wird. Wird beispielsweise beim Signieren eine Hashfunktion verwendet und dieser neben der Nachricht ein Gruppenelement übergeben, so wird implizit davon ausgegangen, dass die Hashfunktion unabhängig von der verwendeten Gruppe arbeitet. Die Autoren konstruieren ein Beispiel, bei dem diese Bedingung nicht erfüllt ist, und das untersuchte Signaturverfahren in Abhängigkeit der verwendeten Gruppe eine als sicher angenommene und eine weitere total unsichere Variante ergeben, obwohl beide Fälle im Generic-Modell als sicher bewiesen werden können. Ein weiteres wichtiges Problem besteht in der Tatsache, dass die binäre Repräsentation durchaus wichtig ist, auch wenn damit nicht der diskrete Logarithmus lösbar ist. So wird die Sicherheit von Verschlüsselungsverfahren auch dadurch gezeigt, dass Verschlüsselungen verschiedener Nachrichten nicht effizient von Zufallsstrings unterscheidbar sind, was aber mit einer „schlechten“ Repräsentation der Gruppenelemente nicht unbedingt erfüllt sein muss. Das Problem der Unterscheidung ist somit unabhängig von dem Problem des diskreten Logarithmus, und die Übertragung eines Beweises im Generic-Modell für generischen Algorithmen auf Verfahren, die neben dem diskreten Logarithmus auf weiteren Annahmen/Problemen basieren, ist problematisch.

2.4.2 Random-Oracle-Modell

Im Random-Oracle-Modell werden alle teilnehmenden Parteien durch interaktive probabilistische Polynomzeit-Turingmaschinen repräsentiert, die Orakel-Zugriff auf eine zufällig gewählte Zufallsfunktion, ein *Random Oracle*, haben. Beweise in diesem Modell basieren auf der Annahme, dass mit dieser Zufallsfunktion geeignete Hashfunktionen repräsentiert werden können. Das Protokoll soll daher unabhängig von den Eigenschaften einer bestimmten Funktion auf die Äquivalenz mit dem verwendeten mathematischen Problem, etwa dem diskreten Logarithmus, hin analysiert werden: Wer Signaturen fälschen kann, kann mit vergleichbarem Aufwand das dem Protokoll zugrundeliegende Problem lösen, von dem aber angenommen wird, dass es nicht effizient, also nicht in polynomi-

eller Zeit, lösbar ist. Das Vorgehen zum Entwurf von kryptographischen Protokollen im Random-Oracle-Modell wird in [BeR93] vorgestellt. Ist die Sicherheit im Random-Oracle-Modell gezeigt, wird für eine Implemetierung das Random-Oracle durch eine Hashfunktion, und damit jede Anfrage an das Oracle durch die Auswertung der Hashfunktion, ersetzt. Hier entsteht das Problem, dass ein Angreifer nicht unbedingt die Hashfunktion auswerten muss, sondern eventuell, aufgrund der ihm zur Verfügung stehenden Spezifikation der Funktion, deren spezielle Eigenschaften ausnutzen kann.

Mit dieser Methode wurden beispielsweise Varianten der RSA-Signatur entworfen und ihre Sicherheit bewiesen [BeR96]. Es ist jedoch möglich, im Random-Oracle-Modell beweisbar sichere Protokolle zu entwerfen, die jedoch für sämtliche Hashfunktionen unsicher sind, was in [CGH98] gezeigt wird. Der Beweis basiert auf den strukturellen Eigenschaften der Random Oracles. Diese liefern für jede Eingabe einen zufälligen Wert als Antwort, aber für zwei identische Anfragen wird derselbe Wert ausgegeben. Da die Antworten zufällig sind, folgt daraus, dass keine Korrelation zwischen Eingabe und Ausgabe besteht, unabhängig davon, wieviele Random Oracles betrachtet werden. Die Autoren beweisen nun, dass diese Aussage auf voll spezifizierte Familien von Funktionen als Instanziierung des Random Oracles nicht zutreffen kann. Sie konstruieren danach Verfahren, deren Sicherheit von dieser Korrelationsbedingung abhängt, und die damit für keine Hashfunktion sicher sind.

3 Komplexitätstheorie

In diesem Kapitel werden einige Begriffe der Komplexitätstheorie erläutert, die für die theoretische Betrachtung von Verschlüsselungs- und Signaturverfahren bedeutend sind. Das Modell einer Turingmaschine und auch die wichtigsten Komplexitätsklassen werden dabei als bekannt vorausgesetzt.

3.1 Probabilistische Turingmaschinen

Die Definition einer probabilistischen Turingmaschine baut auf der Definition einer *deterministischen Turingmaschine* auf. Diese wird durch ein Quadrupel DTM $= (K, \Sigma, \delta, k_0)$ beschrieben, wobei hier exemplarisch der Fall einer Ein-Band-Turingmaschine betrachtet wird. Dann ist K eine endliche Menge von *Zuständen* der endlichen Steuerung der DTM mit dem speziellen *Startzustand* $k_0 \in K$ und dem *Alphabet* Σ, welches die Menge der *Bandsymbole* repräsentiert. Diese sei für die vorliegende Arbeit als $\Sigma = \{0, 1, \#\}$ gegeben, wobei $\#$ das Leerzeichen ist und damit die Menge der *Eingabesymbole* als $\Sigma_0 = \Sigma \backslash \{\#\}$ definiert ist. Die *Übergangsfunktion* δ ist definiert als $K \times \Sigma \to (K \cup \{h\}) \times (\Sigma \cup \{L, R\})$, mit h dem *Haltezustand* und L, R den Aktionen "Kopf-nach-links-" und "Kopf-nach-rechts-bewegen". Eine *Konfiguration* einer DTM ist dann durch den aktuellen Zustand k, den Bandinhalt links der momentanen Kopfposition u, den Bandinhalt rechts der momentanen Kopfposition v und dem Zeichen $\underline{c}$ unter dem Kopf eindeutig bestimmt, während aus der Übergangsfunktion und der aktuellen Konfiguration $\alpha = (k, u\underline{c}v)$ eindeutig die nächste Konfiguration $\alpha' = (k', u'\underline{c}'v')$ der DTM folgt, solange $k \neq h$ erfüllt ist.

Ersetzt man nun die eindeutige Übergangsfunktion δ durch eine Funktion der *Übergangswahrscheinlichkeit* $d : K \times \Sigma \times (K \cup \{h\}) \times (\Sigma \cup \{L, R\}) \to [0, 1]$, erhält man eine *probabilistische Turingmaschine*. Die Funktion d erfüllt dabei die Bedingung

$$\forall k \in K, \forall a \in \Sigma : \sum_{a' \in \Sigma \cup \{L,R\}} \sum_{k' \in K \cup \{h\}} d(k, a, k', a') = 1,$$

wobei die definierten Übergangsregeln in der Menge $\Delta \subseteq K \times \Sigma \times (K \cup \{h\}) \times (\Sigma \cup \{L, R\})$ zusammengefasst sind. Es ist nun nicht mehr eindeutig festgelegt, welcher Zustand auf den aktuellen Zustand k folgt, vielmehr tritt jeder mögliche Zustand $k' \in K \cup \{h\}$, verbunden mit jeder Aktion a', mit einer durch d gegebenen Wahrscheinlichkeit ein. Anstatt die Definition einer probabilistischen Turingmaschine aus der Definition einer deterministischen Turingmaschine zu konstruieren, kann letztere auch als Spezialfall der ersteren aufgefasst werden, für den $d(k, a, k', a') = 1$ gilt, falls $\delta(k, a) = (k', a')$ erfüllt ist und damit $d(k, a, k', a') = 0$ folgt, wenn $\delta(k, a) \neq (k', a')$ gilt.

Beispiel 3.1 *Sei die Ein-Band-Turingmaschine MW definiert durch die Zustandsmenge $K = \{k_0, q\}$ und die Übergangsregeln $(k_0, a, q, i), (q, i, h, R)$ und $(q, \#, h, R)$ mit $i \in \{0, 1\}$, $a \in \Sigma$. Ferner sei $d(k_0, a, q, i) = \frac{1}{2}$, $d(q, i, h, R) = 1$ und $d(q, \#, h, R) = 1$. Dann simuliert die probabilistische Turingmaschine MW den Wurf einer Münze.*

3.1.1 Münzwurfmaschinen

Allgemein werden probabilistische Turingmaschinen als *Münzwurfmaschinen* bezeichnet, wenn $d(k, a, k', a') \subseteq \{0, \frac{1}{2}, 1\}$ erfüllt ist, womit ein beliebiger Zustand k dieser Maschinen höchstens zwei Folgezustände hat. Eine n-Band-Münzwurfmaschine kann auch als eine deterministische $n + 1$-Band-Turingmaschine aufgefasst werden, wobei das zusätzliche Band ω eine zufällige Folge von Elementen aus $\{0, 1\}$ enthält und die Münzwurfmaschine jede ihrer Entscheidungen aus dem Element ableitet, welches sich zu diesem Zeitpunkt auf dem Band ω unter dem Kopf befindet. Dazu tritt die Turingmaschine formal zwischen zwei aufeinanderfolgenden Zuständen mit der Wahrscheinlichkeit 1 in einen *Münzwurfzustand* ein und liest vom Zufallsband, während alle anderen Köpfe unbewegt bleiben. Es dürfen keine zwei Münzwurfzustände hintereinander folgen. Ein Folgezustand tritt dann mit der Wahrscheinlichkeit $\frac{1}{2}$ ein. Das einseitig unbeschränkte Band ω darf außerdem nur gelesen werden, wobei nach jedem Lesevorgang der Kopf nach rechts bewegt wird. Die übrigen n Bänder der Münzwurfmaschine dürfen beidseitig unbeschränkt sein. Probabilistische Turingmaschinen könnten ebenso als Orakel-Turingmaschinen aufgefasst werden. Dazu wäre dann das Band nicht zu Beginn der Berechnung vollständig initialisiert, sondern würde nach jeder Anfrage an das Orakel den Inhalt 0 oder 1 enthalten. Die Münzwurfzustände wären dann die Anfragen an das Orakel. Allgemein können Orakel von deterministischen Turingmaschinen verwendet werden, um Probleme, die nicht in polynomieller Zeit lösbar sind, etwa das Enthaltensein einer Eingabe x in einer Sprache L, von der eigentlichen Berechnung der Maschine zu isolieren. Eine Maschine $\mathcal{M}$ mit Orakel A, repräsentiert durch ein zusätzliches Band, wird dann durch $\mathcal{M}^A$ dargestellt. Sie hat beispielsweise drei spezielle Zustände $k_?, k_j$ und k_n, wobei $k_?$ die Frage „Ist $x \in L$?“ repräsentiert, während die Zustände k_j und k_n die jeweiligen „ja“ und „nein“ Zustände, abhängig vom Bandinhalt, der Antwort des Orakels sind.

In dieser Arbeit werden von nun an lediglich Münzwurfmaschinen verwendet, daher:

Definition 3.1 *Eine probabilistische Turingmaschine ist eine Turingmaschine mit einem Zufallsband $\omega \in \{0, 1\}^*$, die für jede Konfiguration maxmimal zwei Folgekonfigurationen besitzt. Entscheidungen dieser Turingmachine basieren auf dem Inhalt des Zufallsbandes, so dass eine gültige Folgekonfiguration für die momentane Konfiguration ausschließlich mit einer in $\{\frac{1}{2}, 1\}$ enthaltenen Wahrscheinlichkeit eintreten kann. Das Zufallsband wird als zweites Eingabeband aufgefasst und die Münzwurfzustände werden nicht mitgezählt.*

Eine *Berechnung* einer probabilistischen Turingmaschine ist eine Sequenz $\alpha_0 \dots \alpha_m$ von Konfigurationen, wobei m als die *Laufzeit* der Turingmaschine bezeichnet wird. Die Sequenz muss gültig sein: Für alle $i, 0 \leq i \leq m - 1$, muss die Konfiguration α_{i+1} mit einer von Null verschiedenen Wahrscheinlichkeit auf die Konfiguration α_i folgen. Die Wahrscheinlichkeit wird durch die Funktion p bestimmt:

$$p(\alpha, \alpha') = \begin{cases} d(k, a, k', a') & \text{wenn } (k, a, k', a') \in \Delta \\ 0 & \text{sonst.} \end{cases}$$

Die Wahrscheinlichkeit, mit der die Konfiguration α nach n Rechenschritten in

die Konfiguration α' übergeht, ist zu

$$p^{(0)}(\alpha, \alpha') = \begin{cases} 1 & \text{wenn } \alpha = \alpha' \\ 0 & \text{sonst.} \end{cases}$$

und

$$p^{(n)}(\alpha, \alpha') = \sum_{\tau} p^{(n-1)}(\alpha, \tau)\ p(\tau, \alpha')$$

definiert. Damit gilt für die Wahrscheinlichkeit, eine Haltekonfiguration β nach n Rechenschritten mit der Eingabe x zu erreichen

$$P_n(x) = \sum_{\beta \in H} p^{(n)}((k_0, x\underline{\#}), \beta),$$

wobei H die Menge aller *Haltekonfigurationen* $(h, y\underline{\#})$ für ein Ergebnis $y \in \Sigma^*$ und $(k_0, x\underline{\#})$ eine *Startkonfiguration* bezeichnet.

Sei nun eine probabilistische Turingmaschine $\mathcal{M}$ gegeben, die für alle Eingaben $x \in \Sigma^*$ nach endlich vielen Schritten einen Haltezustand erreicht, dann ist die *erwartete Laufzeit* von $\mathcal{M}$ für die Eingabe x durch den Grenzwert der Reihe

$$E_{\mathcal{M}}(x) = \sum_{n=0}^{\infty} nP_n(x)$$

gegeben. Ist diese erwartete Laufzeit für sämtliche $x \in \Sigma^*$ bestimmt und existiert zusätzlich eine Funktion $T(|x|)$, die $E_{\mathcal{M}}(x) \leq T(|x|)$ für sämtliche $x \in \Sigma^*$ erfüllt, dann ist die erwartete Laufzeit *durch T beschränkt*. Ist ferner T ein Polynom, so hat $\mathcal{M}$ eine *polynomiell beschränkte erwartete Laufzeit* oder eine *polynomielle erwartete Laufzeit*.

Ist zusätzlich für alle $x \in \Sigma^*$ und für alle $n > T(|x|)$ die Wahrscheinlichkeit $P_n(x) = 0$, dann besitzt $\mathcal{M}$ eine *durch T beschränkte Laufzeit*. Ist T ein Polynom wird von *polynomieller Laufzeit* gesprochen.

3.1.2 Interaktive Turingmaschinen

Um Kommunikation zwischen probabilistischen Turingmaschinen beschreiben zu können, werden *interaktive Turingmaschinen* definiert. Dazu sei eine interaktive Turingmaschine eine $n+2$-Band probabilistische Turingmaschine, wobei das $(n+1)$ste Band das *Sendeband* und das $(n+2)$te Band das *Empfangsband* repräsentiert. Zusammen heißen diese Bänder *Kommunikationsbänder*. Für diese Bänder sind einige Einschränkungen gültig, so darf etwa der Kopf auf diesen Bändern ausschließlich nach rechts bewegt werden. Das Empfangsband darf nur gelesen werden, während das Sendeband nur beschrieben werden darf. Die Menge der Übergangsregeln sei nun $\Delta \subseteq K \times \Sigma^{n+2} \times (K \cup \{h\}) \times (\Sigma \cup \{L, R\})^{n+2}$ und eine Übergangsregel durch (k, a, k', a') mit $a = (a_1, a_2, ..., a_{n+2}) \in \Sigma^{n+2}$, $a' = (a'_1, a'_2, ..., a'_{n+2}) \in (\Sigma \cup \{L, R\})^{n+2}$ und $k, k' \in K$ definiert. Mit den beschriebenen Einschränkungen der Kommunikationsbänder folgt damit formal $a'_{n+i} \in \Sigma \cup \{R\},\ i = 1, 2$ und für das Sendeband $a'_{n+1} \in \Sigma \Rightarrow a_{n+1} = \#$, während für das Empfangsband $a'_{n+2} \neq R \Rightarrow a_{n+2} = a'_{n+2}$ erfüllt sein muss.

Eine interaktive Turingmaschine muss das Ende einer Nachricht markieren und erkennen können. Dazu wird das Alphabet Σ um das Symbol $\square$ erweitert. Sie darf außerdem nicht gleichzeitig lesen und schreiben und muss ferner nach

dem Senden einer Nachricht auf den Empfang einer neuen Nachricht warten und diese vollständig und verzögerungsfrei (ein Zeichen pro Schritt) gelesen haben, bevor sie erneut mit dem Senden beginnt. Dies wird durch die beiden Bedingungen $a_{n+2} \neq \square, \# \Rightarrow a'_{n+2} = R$ und $a_{n+2} \neq \square \Rightarrow a_{n+1} = a'_{n+1}$ formalisiert. Berechnungen kann die Maschine zwischen dem Ende des Lesevorgangs und dem Beginn des Sendevorgangs durchführen, indem sie die erhaltene Nachricht auf ein weiteres Band kopiert und mit dieser Kopie arbeitet.

Aus zwei interaktiven Turingmaschinen $\mathcal{I}_1, \mathcal{I}_2$ kann ein *interaktives System* $S(\mathcal{I}_1, \mathcal{I}_2)$ konstruiert werden, wobei die beiden Maschinen ihre Kommunikationsbänder teilen und dabei das Sendeband von $\mathcal{I}_1$ dem Empfangsband von $\mathcal{I}_2$ und umgekehrt entspricht. Diese Bänder besitzen dann zwei Köpfe, um unterschiedliche Kopfpositionen der beiden Maschinen zu ermöglichen. Diese Eigenschaft wird durch eine erweiterte Notation der Konfiguration, hier beispielhaft für ein Band, beschrieben:

$$\alpha = (k, u \underline{\overset{1}{c}} v \underline{\overset{2}{e}} w).$$

Ein weiteres Band, bei welchem lediglich die erste Zelle verwendet wird und das beide Turingmaschinen gemeinsam nutzen, synchronisiert die Kommunikation der Maschinen. Jede der Maschinen invertiert nach abgeschlossenem Sendevorgang den Inhalt der Zelle und ist danach inaktiv, während die jeweils andere Maschine nun mit dem Lesen beginnt. In der Startkonfiguration enthält die Zelle das Zeichen 0 und $\mathcal{I}_1$ beginnt. Hierfür muss allerdings das Alphabet um die Aktion $N =$ „Stehenbleiben ohne Schreiben“ erweitert werden, um Kollisionen („Races“) auf dem Synchronisationsband zu vermeiden, die dadurch entstehen, dass die Köpfe beider Maschinen sich auf der selben Position befinden und Lesen mathematisch durch Schreiben des sich zuvor an dieser Stelle befindenden Zeichens definiert ist. Invertierte nun eine Maschine den Inhalt um die andere Maschine zu aktivieren, wäre der Inhalt der Zelle undefiniert, denn die jeweils andere Maschine würde versuchen, den ursprünglichen Inhalt zu schreiben.

Das interaktive System $S(\mathcal{I}_1, \mathcal{I}_2)$ ist dann ebenfalls eine probabilistische Turingmaschine $\mathcal{S}$ mit $(n_1 + n_2 + 3)$-Bändern, welche die Kommunikation der beiden Maschinen $\mathcal{I}_1, \mathcal{I}_2$ simuliert. Die endliche Steuerung der Maschine $\mathcal{S}$ hat die Zustandsmenge $K = (K_1 \cup \{h_1\}) \times (K_2 \cup \{h_2\}) \backslash \{(h_1, h_2)\}$, den Startzustand $(k_{0,1}, k_{0,2}) \in K$ und den Haltezustand (h_1, h_2). Übergangsregeln sind nun von der Form (k, a, k', a') mit

$$a = (a_{1,1}, a_{2,1}, ..., a_{n_1,1}, a_{1,2}, a_{2,2}, ..., a_{n_2,2}, A_1, A_2, A_3)$$

$$a' = (a'_{1,1}, a'_{2,1}, ..., a'_{n_1,1}, a'_{1,2}, a'_{2,2}, ..., a'_{n_2,2}, A'_1, A'_2, A'_3)$$

für $a_{i,j} \in \Sigma,\ a'_{l,j} \in \Sigma \cup \{L, R\}$ mit den Indizes $1 \leq i \leq n_j + 3$ sowie $j \in \{1, 2\}$ und $1 \leq l \leq n_j$. Für die Kopfpaare der gemeinsam genutzten Bänder

$$A_1 = (a_{n_1+1,1}, a_{n_2+2,2}),\ A_2 = (a_{n_2+1,1}, a_{n_1+2,2}),\ A_3 = (a_{n_1+3,1}, a_{n_2+3,2})$$

$$A'_1 = (a'_{n_1+1,1}, a'_{n_2+2,2}),\ A'_2 = (a'_{n_2+1,1}, a'_{n_1+2,2}),\ A'_3 = (a'_{n_1+3,1}, a'_{n_2+3,2})$$

gilt $A_1, A_2 \in \Sigma \times \Sigma$ und $A'_1, A'_2 \in \Sigma \cup \{R\} \times \Sigma \cup \{R\}$, sowie $A_3 \in \Sigma_0$ und mit der im letzten Absatz definierten Aktion N ist $A'_3 \in \Sigma_0 \cup \{N\}$. Mit dem Synchronisationsband $(n_j + 3)$ müssen die Übergangsregeln folgenden Bedingungen

genügen:

$$\begin{aligned} a'_{n_j+1,j} \neq \ & \square & \Rightarrow a'_{n_j+3,j} = a_{n_j+3,j} \\ a_{n_j+1,j} \neq \square \wedge a'_{n_j+1,j} = \ & \square & \Rightarrow a'_{n_j+3,j} = \neg a_{n_j+3,j} \\ a_{n_1+3,1} = \ & 1 & \Rightarrow a'_{i,1} = a_{i,1},\ a'_{n_1+3,1} = N \\ a_{n_2+3,2} = \ & 0 & \Rightarrow a'_{i,2} = a_{i,2},\ a'_{n_2+3,2} = N \end{aligned}$$

Zu Beginn einer Berechnung der Maschine $\mathcal{S} = S(\mathcal{I}_1, \mathcal{I}_2)$ ist der Inhalt der Bänder für die Eingabe $(x_1, x_2) \in \Sigma_0^* \times \Sigma_0^*$ definiert als

$$(x_1\underline{\#}, \underline{\#}, ..., x_2\underline{\#}, \underline{\#}, ..., \overset{1,2}{\underline{\#}}, \overset{1,2}{\underline{\square}}, \overset{1,2}{\underline{0}}).$$

Nach der Berechnung, die wie zuvor durch eine Sequenz von Konfigurationen definiert ist, enthält das Sendeband von $\mathcal{I}_j$ eine Zeichenkette der Form $a_{1_j}\square a_{2_j}\square \ldots a_{m_j}\square$, jedes a_{i_j} ist eine *Nachricht* von $\mathcal{I}_j$. Eine *Interaktion* ist damit ein Sequenz von Konfigurationen, wobei die erste Konfiguration einer Interaktion durch das Schreiben des ersten Zeichens der Nachricht a_{i_1} und die letzte Konfiguration einer Interaktion durch das Lesen des Symbols $\square$ am Ende der Nachricht a_{i_2} definiert ist. Hat nun $\mathcal{I}_2$ genau m_2 Nachrichten geschrieben, so hat $\mathcal{I}_1$ entweder $m_1 = m_2$ oder $m_1 = m_2 + 1$ Nachrichten erzeugt. Die *Anzahl der Interaktionen* wird definiert als m_1.

3.2 Interaktive Beweissysteme

Da die Wahrscheinlichkeit für Folgekonfigurationen einer probabilistischen Turingmaschine immer $\frac{1}{2}$ ist, wenn diese aufgrund eines Münzwurfes eintreten, können sehr einfach Wahrscheinlichkeitsverteilungen für die Menge der Berechnungen und die Menge der Ausgaben bestimmt werden. Sei nun eine probabilistische Turingmaschine gegeben, welche für eine Berechnung mit der Eingabe x exakt $T(|x|)$ Münzwürfe benötigt. Dann ist die Wahrscheinlichkeit $\Pr[\mathcal{M}(x) = y]$, dass $\mathcal{M}$ bei der Eingabe von x mit dem Zufallsband ω das Ergebnis y liefert bestimmt als:

$$\Pr[\mathcal{M}(x) = y] \ = \sum_{\substack{\omega \in \{0,1\}^{T(|x|)} \\ \mathcal{M}(x,\omega) = y}} \frac{1}{2^{T(|x|)}}.$$

Mit dieser Vorausetzung wird die Komplexitätsklasse *BPP* (Bounded-Probability Polynomial-Time) als eine Klasse von Sprachen definiert, für die eine probabilistische Turingmaschine $\mathcal{M}$ existiert, die mit hoher Wahrscheinlichkeit eine Eingabe x in polynomieller Zeit akzeptiert, falls diese in einer Sprache $L \subseteq \Sigma_0^*$ enthalten ist:

$$x \in L \Rightarrow \Pr[\mathcal{M}(x) = 1] \geq \frac{2}{3},$$

$$x \notin L \Rightarrow \Pr[\mathcal{M}(x) = 1] < \frac{1}{3}.$$

Die Wahl der Konstanten $\frac{2}{3}$ ist dabei willkürlich und kann durch jeden Term $\frac{1}{2} + \epsilon$ für $0 < \epsilon < \frac{1}{2}$ ersetzt werden. Die Maschine $\mathcal{M}$ macht offenbar Fehler, gibt

aber meistens das richtige Ergebnis aus. Der Fehler kann beliebig verkleinert werden, indem $\mathcal{M}$ so oft aufgerufen wird, bis die Wahrscheinlichkeit für ein gewähltes ϵ und die Mehrzahl der Antworten von $\mathcal{M}$ erfüllt ist. Die Klasse der Sprachen ändert sich dabei nicht.

Damit wird nun ein interaktives System zweier probabilistischer Turingmaschinen konstruiert, mit dem dann bewiesen werden kann, dass eine für beide Maschinen identische Eingabe in einer gegebenen Sprache enthalten ist. Eine der beiden Maschinen übernimmt dabei die Rolle des *Beweisers*, während die andere Maschine, der *Verifizierer* $\mathcal{V}$, die Behauptung des Beweisers $\mathcal{P}$ überprüft. Beide Maschinen haben einen *akzeptierenden* und einen *ablehnenden Haltezustand*. Zusätzlich ist die Laufzeit des Verifizierers polynomiell beschränkt, während dem Beweiser unbeschränkte Laufzeit zugestanden wird. Die Anzahl der *Runden* in diesem Beweis entspricht der Anzahl der zwischen dem Beweiser und dem Verifizierer ausgetauschten Nachrichten. Die erste (leere) Nachricht, die lediglich dazu dient den Beweiser zu starten, wird nicht mitgezählt. Eine Sprache, für die ein solcher *interaktiver Beweis* oder *interaktives Beweissystem* existiert, heißt *interaktiv beweisbar*, und die Klasse dieser Sprachen wird als *IP* bezeichnet. Formal ist eine Sprache L interaktiv beweisbar, wenn ein interaktives System $I(\mathcal{P},\mathcal{V})(x) = S(\mathcal{P},\mathcal{V})(x,x)$ existiert, für das die folgenden Bedingungen gelten:

1. *Vollständigkeit:* $\forall x \in L : \Pr[I(\mathcal{P},\mathcal{V})(x) = 1] \geq \frac{2}{3}$.
2. *Korrektheit:* Für alle interaktive Turingmaschinen $\mathcal{P}'$ mit unbeschränkter Laufzeit gilt: $\forall x \notin L : \Pr[I(\mathcal{P}',\mathcal{V})(x) = 1] < \frac{1}{3}$.

Mit dieser Definition folgt sofort $BPP \subseteq IP$.

Die Vollständigkeitsbedingung garantiert einem Beweiser, dass er den Verifizierer mit einer Wahrscheinlichkeit von mindestens $\frac{2}{3}$ überzeugen kann, falls $x \in L$ ist, während die Korrektheitsbedingung garantiert, dass kein „mogelnder“ Beweiser den Verifizierer dazu bringen kann, einen Beweis mit einer Wahrscheinlichkeit größer als $\frac{1}{3}$ zu akzeptieren, wenn $x \notin L$ ist.

Beispiel 3.2 *Sei ISO die Menge aller isomorphen Graphen-Paare (G_0, G_1) mit gerichteten Kanten, wobei ein Graph $G_i = (V_i, E_i)$ durch eine Menge der Knoten V_i und eine Menge der Kanten $E_i \subseteq V_i \times V_i$ definiert ist. Zwei Graphen sind isomorph zueinander, wenn die Bedingung $|V_0| = |V_1|$ erfüllt ist und eine Permutation π aller Kanten $(v,w) \in E_0$ existiert, so dass $(\pi(v), \pi(w)) \in E_1$ erfüllt ist. Die Notation sei dann $\pi(G_0) = G_1$ und $G_0 \cong G_1$ und es lässt sich der folgende, interaktive Beweis für Graphenisomorphismen konstruieren.*

Verifizierer	**Kommunikation**	**Beweiser**
liest Eingabe (G_0, G_1)		
	$\rightarrow \square \rightarrow$	
		liest Eingabe (G_0, G_1)
		berechnet π
		mit $\pi(G_0) = G_1$
	$\leftarrow \pi\square \leftarrow$	
akzeptiert den Beweis, wenn $\pi(G_0) = G_1$		

Die Fehlerwahrscheinlichkeit bei diesem Beweissystem beträgt 0. *Sowohl die Vollständigkeitsbedingung als auch die Korrektheitsbedingung sind offensichtlich erfüllt.*

In diesem Beispiel hat der Beweiser eine Runde benötigt, um den Verifizierer zu überzeugen, dass $(G_0, G_1) \in ISO$ erfüllt ist.

Aus der Definition von BPP ist sofort ersichtlich, dass ein Verifizier alleine $x \in BPP$ verifizieren kann, weswegen in diesem Fall null Runden benötigt werden. Da ein Verifizierer polynomielle Laufzeit besitzt, kann es nicht mehr als polynomiell, in Abhängigkeit der Länge der Eingabe, viele Runden geben. Jeder interaktive Beweis ist daher in einer Komplexitätsklasse $IP(r(|x|))$ für ein Polynom $r(|x|)$ enthalten, wenn höchstens $r(|x|)$ Runden benötigt werden. Also ist $BPP = IP(0)$ und $NP \subseteq IP(1)$. Mit $poly$, der Menge aller ganzzahligen Polynome, ist dann $IP = IP(poly)$.

3.2.1 Vergleich von Wahrscheinlichkeitsverteilungen

An dieser Stelle werden nun einige Begriffe definiert, die auch und insbesondere im nächsten Abschnitt benötigt werden.

Für eine abzählbare Menge M heißt $\delta : 2^M \to [0,1]$ eine *Wahrscheinlichkeitsverteilung* über M wenn $\delta(M) = 1$ und wenn $\forall A \subseteq M$ die Bedingung $\delta(A) \geq 0$ gilt, sowie $\forall A_1, A_2 \subseteq M$, für die $A_1 \cap A_2 = \emptyset$ ist, die Gleichung $\delta(A_1 \cup A_2) = \delta(A_1) + \delta(A_2)$ erfüllt ist. Eine *Verteilung* δ könnte so beispielsweise die Wahrscheinlichkeit $\delta_x(Y)$, mit der eine probabilistische Turingmaschine ein Ergebnis $y \in Y$ für die Eingabe x liefert, beschreiben.

Seien nun für eine Sprache $L \subseteq \Sigma_0^*$ für jedes $x \in L$ zwei Verteilungen δ_x und δ'_x über Σ_0^* gegeben und seien dann $\delta = \{\delta_x\}_{x \in L}$ und $\delta' = \{\delta'_x\}_{x \in L}$ *Familien von Verteilungen*, so sind diese

- *perfekt ununterscheidbar*, wenn $\delta_x = \delta'_x$ für alle bis auf endlich viele $x \in L$ erfüllt ist,
- *statistisch ununterscheidbar*, wenn für alle Polynome p gilt:

$$\lim_{|x| \to \infty} p(|x|) \sum_{y \in \Sigma_0^*} |\delta_x(y) - \delta'_x(y)| = 0$$

  Der *statistische Unterschied* Δ_{S} wird definiert als:

$$\Delta_{\mathrm{S}}(\delta_x, \delta'_x) = \sum_{y \in \Sigma_0^*} |\delta_x(y) - \delta'_x(y)|$$

- *algorithmisch ununterscheidbar*, wenn für alle Polynome p und alle probabilistischen Turingmaschinen $\mathcal{D}$ mit polynomieller Laufzeit gilt:

$$\lim_{|x| \to \infty} p(|x|) \sum_{y \in \Sigma_0^{|x|}} \left| \Pr_{y \in \delta_x}[\mathcal{D}(y) = 1] - \Pr_{y \in \delta'_x}[\mathcal{D}(y) = 1] \right| = 0$$

  Der *algorithmische Unterschied* Δ_{A} ist dann:

$$\Delta_{\mathrm{A}}(\delta_x, \delta'_x) = \sum_{y \in \Sigma_0^{|x|}} \left| \Pr_{y \in \delta_x}[\mathcal{D}(y) = 1] - \Pr_{y \in \delta'_x}[\mathcal{D}(y) = 1] \right|$$

Gilt für jedes Polynom p und alle hinreichend langen $x \in L$ die Bedingung $\Delta_S < p(|x|)^{-1}$, so heißt der statistische Unterschied *vernachlässigbar*. Genauso ist der algorithmische Unterschied vernachlässigbar, wenn $\Delta_A < p(|x|)^{-1}$ erfüllt ist.

Diese Kriterien bilden eine Hierachie; so sind alle statistisch ununterscheidbaren Familien von Verteilungen auch algorithmisch ununterscheidbar und alle perfekt ununterscheidbaren Familien sind statistisch ununterscheidbar.

3.2.2 Zero-Knowledge-Beweise

In dem in Beispiel 3.2 beschriebenen interaktiven Beweissystem sollte der Beweiser den Verifizierer überzeugen, dass $(G_0, G_1) \in ISO$ ist. Dabei hat er nicht nur die Information, dass diese Bedingung erfüllt ist, preisgegeben, sondern zusätzlich den dazugehörigen Isomorphismus. Er hat damit mehr Information kommuniziert als nötig wäre, um den Verifizierer zu überzeugen. Angenommen das Beweissystem wäre (in welcher Form auch immer) Bestandteil eines Authentisierungsprotokolls, und zusätzlich gäbe es einen Angreifer mit polynomiell beschränkter Laufzeit. Dann ist dieser nicht in der Lage, Graphenisomorphismen zu berechnen, denn dieses Problem scheint in $NP \backslash P$ enthalten zu sein. Er gewinnt jedoch, indem er die Kommunikation zwischen $\mathcal{P}$ und $\mathcal{V}$ belauscht, zusätzliches *Wissen*, welches er dann für einem Angriff verwenden könnte. Formal wird beispielsweise bei allen Beweissystemen für die Sprachen $L \in BPP$ kein zusätzliches Wissen preisgegeben, denn es findet keine Kommunikation statt.

Der Begriff *Wissenskomplexität* (Knowledge-Complexity), und ein mathematisches Maß dafür, wurde in [GMR85] definiert, um das bei einer Kommunikation in der übertragenen Information enthaltene Wissen kategorisieren zu können. Sei $I(\mathcal{P}, \mathcal{V})$ ein interaktives Beweissystem für die Sprache L und der Beweiser $\mathcal{P}$ habe am Ende des Protokolls den Verifizierer $\mathcal{V}$ überzeugt, dass $x \in L$ ist. Neben diesem Ergebnis existiert dann auch ein Transkript der Kommunikation, dessen Inhalt für das System I mit einer durch die Verteilung δ_I bestimmten Wahrscheinlichkeit auftritt. Sei nun eine Maschine S gegeben, welche derartige Transkripte nach einer Verteilung δ_S erzeugt, ohne den Beweiser $\mathcal{P}$ (und damit ohne das Geheimnis des Beweisers) zu verwenden. Das übertragene Wissen wird dann über den Unterschied der beiden Verteilungen δ_S und δ_I definiert. Beispielsweise werden bei einem String aus Zufallsbits der Länge n zwar n Bit Information übertragen, aber diese Zufallsbits können von einer weiteren Turingmaschine mit einer ununterscheidbaren Verteilung erzeugt werden. In diesem Fall wird also kein zusätzliches Wissen übertragen. Besitzen die Transkripte nun die Länge n und die Verteilungen δ_S und δ_I höchstens den Unterschied $\Delta = \frac{1}{2^{f(n)}}$ für eine monoton steigende Funktion f, dann hat L die Wissenskomplexität $f(n)$ und man schreibt $L \subseteq KC(f(n))$. Also ist trivialerweise $BPP \subseteq KC(0)$. Alle interaktiven Beweissysteme, die in $KC(0)$ enthalten sind, heißen *Zero-Knowledge-Beweise*. Mit den im letzten Abschnitt definierten Unterschieden von Verteilungen existieren daher algorithmische, statistische und perfekte Zero-Knowledge Beweise. Für diese Beweissysteme existieren immer *Simulatoren*, die für jede Maschine $\mathcal{V}'$, mit der der Beweiser $\mathcal{P}$ interagiert, eine Kommunikation mit einer ununterscheidbaren Verteilung erzeugen kann.

Für das Beispiel 3.2 etwa, ist die Länge der Beschreibung der Permutation π abhängig von der Anzahl der Knoten und daher werden in diesem Fall $f(|V_0|)$ Bit an Wissen übertragen. Im folgenden wird das Beispiel modifiziert, so dass

kein zusätzliches Wissen preisgegeben wird.

Beispiel 3.3 *Es gelten die Definitionen aus Beeispiel 3.2. Dann ist das folgende interaktive Beweissystem $I(\mathcal{P}, \mathcal{V})$ für ISO ein Zero-Knowledge-Beweis, basierend auf der Tatsache, dass ein Graph H nur dann zu zwei Graphen G_0und G_1 isomorph ist, wenn auch $G_0 \cong G_1$ erfüllt ist. Die Vollständigkeitsbedingung ist offensichtlich erfüllt, denn $\mathcal{P}$ ist für zwei isomorphe Graphen (G_0, G_1) per Definition in der Lage die Isomorphismen π und π_1 zu berechnen. Sind nun die beiden Graphen nicht isomorph, dann ist die Wahrscheinlichkeit dafür, dass der Verifizierer nach n Runden akzeptiert $(\frac{1}{2})^n$, und das ist für $n \geq 2$ kleiner als $\frac{1}{3}$, weswegen auch die Korrektheitsbedingung erfüllt ist.*

Verifizierer	**Kommunikation**	**Beweiser**
liest Eingabe (G_0, G_1)		
	$\rightarrow \square \rightarrow$	
		liest Eingabe (G_0, G_1)
		berechnet π mit $\pi(G_0) = G_1$
$n = \lvert V_0 \rvert$ Durchläufe über i:		
		wählt zufällig eine Permutation $\pi_{0,i}$
		berechnet $H_i = \pi_{0,i}(G_0)$ und $\pi_{1,i}$: $H_i = \pi_{1,i}(G_1)$
	$\leftarrow H_i \square \leftarrow$	
wählt zufälliges $r_i \in \{0, 1\}$		
	$\rightarrow r_i \square \rightarrow$	
	$\leftarrow \pi_{r,i} \square \leftarrow$	
wenn $i < n$:	$\rightarrow \square \rightarrow$	
Ende der Schleife		
akzeptiert, wenn $\forall i : \pi_{r,i}(G_{r,i}) = H_i$		

Um nun zu zeigen, dass dieses Beweissystem in $KC(0)$ enthalten ist, wird eine probabilistische Turingmaschine $\mathcal{M}$ mit polynomieller Laufzeit konstruiert, welche eine Ausgabe mit einer Verteilung erzeugt, die identisch zu der Verteilung der Ausgabe des Beweissystems $I(\mathcal{P}, \mathcal{V}')$ für einen beliebigen Verifizierer $\mathcal{V}'$ ist. Wie bereits beschrieben, wird die Maschine $\mathcal{M}$ *Simulator* genannt, da sie die Kommunikation des Beweisers und des Verifizierers simuliert. Kann die Simulation nun nicht von einer reellen Kommunikation unterschieden werden, so wird in der Kommunikation kein zusätzliches Wissen übertragen.

Der Simulator $\mathcal{M}$ arbeitet nun wie folgt: Zuerst simuliert er den Beweiser und wählt ein zufälliges $k \in \{0, 1\}$, eine zufällige Permutation π und berechnet

$H = \pi(G_k)$. Dannach speichert er den Zustand des Verifizierers und startet ihn. Der Verifizierer schreibt dann $r\Box$ mit $r \in \{0, 1\}$ auf das Empfangsband des simulierten Beweisers. Ist $k = r$, so gibt $\mathcal{M}$ die Sequenz $H\Box\pi\Box$ aus, andernfalls setzt der Simulator den Verifizierer auf den zuletzt gespeicherten Zustand zurück und versucht es erneut. Dieses Vorgehen wird insgesamt n mal wiederholt. Danach enthält das Empfangsband des Verifizierers, sowohl für das „reale" Beweissystem als auch für den Simulator, eine Sequenz zufälliger Graphen und Isomorphismen der Form $H_1\Box\pi_1\Box H_2\Box\pi_2\Box \ldots H\Box_n\pi_n\Box$, und es bleibt zu zeigen, dass die Verteilungen der beiden Fälle ununterscheidbar ist. Das ist trivial für den beschriebenen Simulator, denn dieser wird, nach beliebig vielen Versuchen, in jedem Durchlauf einen Fall $k = r$ erreichen und dann eine gültige Ausgabe erzeugen für welche $\Pr[\mathcal{M}(x) = y] = \Pr[I(\mathcal{P}, \mathcal{V}')(x) = y]$ erfüllt ist. Dann heißt das Beweissystem *perfekt Zero Knowledge*. Damit hat jedoch der Simulator keine polynomielle Laufzeit, sondern lediglich eine polynomiell beschränkte erwartete Laufzeit. Um der ursprünglichen Definition zu folgen, muss also die Laufzeit polynomiell beschränkt werden, indem die innere Schleife maximal m mal durchlaufen werden darf. Für die Wahrscheinlichkeit, dass in m Durchläufen $r = k$ nicht erfüllt ist, gilt $p' = (\frac{1}{2})^m$. Dann ist also die Wahrscheinlichkeit dafür, dass in einer der n Runden keine gültige Ausgabe erzeugt wird $p = 1 - (1 - p')^n$. Mit der Bernoullischen Ungleichung $(1 + x)^n \geq 1 + nx$ folgt daraus $p \leq 1 - (1 - np') \leq \frac{n}{2^m}$.

Sei Z die Menge der ungültigen Ausgaben und $\bar{Z}$ die Menge der gültigen Ausgaben. Dann ist mit $m = |x| + 1$

$$\Delta_S = \lim_{|x|\to\infty} q(|x|) \sum_{y\in\Sigma^*} |\Pr[\mathcal{M}(x) = y] - \Pr[I(\mathcal{P}, \mathcal{V}')(x) = y]| \; =$$

$$\lim_{|x|\to\infty} q(|x|) \left(\Pr[\mathcal{M}(x) \in Z] + \sum_{y\in\bar{Z}} |\Pr[\mathcal{M}(x) = y] - \Pr[I(\mathcal{P}, \mathcal{V}')(x) = y]| \right)$$

Weil aber $\Pr[\mathcal{M}(x) = y] = (1 - p) \Pr[I(\mathcal{P}, \mathcal{V}')(x) = y]$ gilt, ergibt sich daraus

$$\Delta_S = \lim_{|x|\to\infty} q(|x|) \left(\Pr[\mathcal{M}(x) \in Z] + \sum_{y\in\bar{Z}} p \cdot \Pr[I(\mathcal{P}, \mathcal{V}')(x) = y] \right)$$

Zusätzlich ist $\Pr[\mathcal{M}(x) \in Z] = p \leq n2^{-(|x|+1)}$, und damit folgt schliesslich

$$\Delta_S = \lim_{|x|\to\infty} q(|x|) \, (p + p) \; = 0$$

Für einen Simulator $\mathcal{M}$ mit polynomiell beschränkter Laufzeit ist also die Verteilung der Ausgabe statistisch ununterscheidbar von der Verteilung der Ausgabe des interaktiven Systems $I(\mathcal{P}, \mathcal{V}')$, das Beweissystem heißt somit *statistisch Zero Knowledge*.

Obwohl bisher bereits von Zero-Knowledge-Beweisen gesprochen wurden, wird streng genommen immer noch Information preisgegeben, nämlich dass die Bedingung $x \in L$ erfüllt ist. In [FFS86] wurde das Prinzip der Beweise modifiziert, so dass nun nicht das Enthaltensein einer Eingabe in einer Sprache bewiesen wird, sondern dass ein Beweiser ein Geheimnis kennt und den Verifizierer davon überzeugt, ohne irgendeine Information über dieses Geheimnis

preiszugeben. Für das bisherige Beispiel würde das bedeuten, dass der Beweiser nun nicht $G_0 \cong G_1$ beweist, sondern lediglich seine Kenntnis über eine Beziehung zwischen den beiden Graphen, so dass der Verifizierer selbst über eben diese Beziehung keine Information erhält. Dieses Wissen wird als geheime, lediglich dem Beweiser zugängliche Eingabe, in Form eines zusätzlichen Eingabebandes $I(\mathcal{P}(s), \mathcal{V})$ repräsentiert. Da nun, anders als bisher, dieses Wissen nicht mehr vom Beweiser berechnet werden muss, kann damit auch der Beweiser eingeschränkt werden, so dass ihm nun ebenfalls nur polynomielle Rechenzeit zugestanden wird.

Um dies zu formalisieren, sei $R \subseteq \{0,1\}^* \times \{0,1\}^*$ eine *Relation*, mit der Lösungsmenge $R(x) = \{s : (x,s) \in R\}$ und sei außerdem $L_R = \{x : R(x) \neq \emptyset\}$. Eine Relation R heißt *polynomiell beschränkt*, wenn ein Polynom p existiert, das für alle $(x,s) \in R$ die Bedingung $|s| \leq p(|x|)$ erfüllt.

Die Zugehörigkeitsbedingungen für $KC(0)$ bleiben unverändert, jedoch die Vollständigkeits- und Korrektheitsbedingungen werden damit zu:

1. *Vollständigkeit:* $\forall (x,s) \in R : \Pr[I(\mathcal{P}(s), \mathcal{V})(x) = 1] \geq \frac{2}{3}$.
2. *Korrektheit:* Es gibt eine Maschine $\mathcal{M}$, welche alle interaktiven Turingmaschinen $\mathcal{P}'$, die eine polynomieller Laufzeit besitzen, das Geheimnis nicht kennen und $\Pr[I(\mathcal{P}'(y), \mathcal{V})(x) = 1] > \frac{1}{3}$ für $x \in L_R,\ y \in \Sigma^*$ erfüllen, mit unverändertem Zufallsband verwenden kann, so dass $\Pr[S(\mathcal{P}'(y), \mathcal{M})(x) \in R] \geq \frac{2}{3}$ gilt. Falls also $\mathcal{P}'$ den Verifizierer überzeugen konnte, extrahiert $\mathcal{M}$ aus diesem ein s, dass mit großer Wahrscheinlichkeit $(x,s) \in R$ erfüllt.

In der Regel bestehen Protokolle, die auf solchen Beweisen basieren, aus mindestens einer Runde, wobei in jeder Runde drei Nachrichten zwischen dem Verifizierer und dem Beweiser gesendet werden. Die erste sendet der Beweiser und diese wird *Witness* genannt. Dieser Witness wird aus einem zufällig gewählten und geheimen *Commitment* berechnet. Damit wird die Zufälligkeit und Unabhängigkeit der mehrfachen Ausführung des Protokolls garantiert und die Menge der Fragen, die der Beweiser behauptet beantworten zu können, definiert. Aus dieser Menge wählt nun der Verifizierer eine zufällige Frage, welche *Challenge* genannt wird, aus und sendet diese an den Beweiser, der darauf mit einer *Response* reagiert, welche wiederum vom Verifizierer auf ihre Korrektheit überprüft wird. Im folgenden wird nun ein solches praktisches Protokoll vorgestellt, das seinen Ursprung in [FFS86] hat.

Beispiel 3.4 (Feige-Fiat-Shamir-Identifikationsprotokoll) *In diesem Beispiel überzeugt ein Beweiser den Verifizierer von seiner Kenntnis über Quadratwurzeln modulo einer zusammengesetzten Zahl $n = pq$. Dieses Modul ist das Produkt zweier großen Primzahlen p, q, die beide kongruent zu* 3 mod 4 *sind. Die Sicherheit des Protokolls beruht auf der Schwierigkeit, n zu faktorisieren. Die Zahl n ist öffentlich bekannt, ebenso die Zahlen k und t, welche die Sicherheitsparameter des Beweissystems darstellen. Ein Beweiser $\mathcal{P}$ wählt nun k zufällige Zahlen $s_1, s_2, ..., s_k$ derart, dass alle im Bereich $1 \leq s_j \leq n-1$ liegen und prim zu n sind, sowie k zufällige Bits $b_1, b_2, \ldots, b_k$. Mit diesen berechnet er $v_j = (-1)^{b_j} s_j^{-2} \mod n$ für alle j im Bereich $1 \leq j \leq k$. Das Geheimnis eines Beweisers sind dabei die k Zahlen $(s_1, \ldots, s_k)$, und auch die b_i können geheimgehalten werden. Die k Zahlen $(v_1, \ldots, v_k)$ müssen dem Verifizierer bekannt sein. Dann läuft der Beweis in der folgenden Art und Weise ab:*

Verifizierer	**Kommunikation**	**Beweiser**
liest Eingabe $(v_1, \ldots, v_k)$		
	$\rightarrow \square \rightarrow$	
		liest Eingabe $(s_1, \ldots, s_k)$
		liest Eingabe $(v_1, \ldots, v_k)$
	t Runden über i:	
		wählt zufällig ein r_i, aus $1 \leq r_i \leq n-1$
		berechnet mit Zufallsbit b_i $x_i = (-1)^{b_i} r_i^2 \bmod n$
	$\leftarrow x_i \square \leftarrow$	
wählt k Zufallsbits $(e_1, \ldots, e_k)$		
	$\rightarrow (e_1, \ldots, e_k) \square \rightarrow$	
		berechnet $y_i = r_i \prod_{j=1}^{k} s_j^{e_j} \bmod n$
	$\leftarrow y_i \square \leftarrow$	
berechnet $z_i = y_i^2 \prod_{j=1}^{k} v_j^{e_j} \bmod n$		
wenn $i < t$:	$\rightarrow \square \rightarrow$	
	Ende der Schleife	
akzeptiert, wenn $\forall i : z_i \neq 0 \wedge z_i = \pm x_i$		

Die Vollständigkeitsbedingung ist erfüllt, denn es gilt

$$y_i^2 \prod_{j=1}^{k} v_j^{e_j} = (r_i \prod_{j=1}^{k} s_j^{e_j})^2 \prod_{j=1}^{k} v_j^{e_j} = r_i^2 \prod_{j=1}^{k} (s_j^2 v_j^{e_j}) = \pm r_i^2 = \pm x_i \bmod n.$$

Für den Beweis zur Korrektheitsbedingung sei jedoch auf [FFS86] verwiesen. Ähnlich zu Beispiel 3.3 muss ein Simulator konstruiert werden, um die Zero-Knowledge Eigenschaft zu beweisen. Dieser muss ebenfalls die Challenge des Verifizierers raten und hat dabei mit einer Wahrscheinlichkeit von $\frac{1}{2^k}$ in jeder Runde Erfolg. Ebenso wie zuvor wird die Erfolgswahrscheinlichkeit durch Wiederholung erhöht. Hat er passende Bits $(e_1, e_2, \ldots, e_k)$ gefunden, so kann er für das vorbereitete $x_i = \pm r / (\prod_{j=1}^{k} v_j) \bmod n$ auf die Challenge mit $y_i = r_i$ antworten.

Das vorgestellte Identifikationsprotokoll, und ebenso jeder andere auf ei-

ner Witness-Challenge-Response-Sequenz basierende Zero-Knowledge-Beweis, kann in ein Signaturverfahren umgewandelt werden. Dazu muss die Challenge $(e_1, e_2, \ldots, e_k)$, und damit der Verifizierer, durch die Anwendung einer Hashfunktion auf die Nachricht und den Witness mit $e = h(m, x) = h(m\|x)$ ersetzt werden, womit das folgende Signaturverfahren entsteht.

Beispiel 3.5 (Feige-Fiat-Shamir-Signatur)

1. Schlüsselerzeugung:
   Zuerst erzeugt Ally zwei unterschiedliche Primzahlen p und q und berechnet $n = pq$. Sie wählt danach eine Zahl k und erzeugt k zufällige Zahlen $s_1, s_2, \ldots, s_k \in \mathbb{Z}_n^*$. Mit diesen berechnet sie dann für $1 \leq i \leq k$ die Zahlen $v_i = s_i^{-2} \bmod n$. Allys öffentlicher Schlüssel ist $((v_1, v_2, \ldots, v_k), n)$ und ihr privater Schlüssel ist $(s_1, s_2, \ldots, s_k)$.

2. Signaturerzeugung:
   Um nun die Nachricht m zu signieren, wählt Ally ein zufälliges $r \in \mathbb{Z}_n^*$ und berechnet $x = r^2 \bmod n$. Dann erzeugt sie k Bits $(e_1, e_2, \ldots, e_k)$ aus dem Hashwert $e = h(m\|x)$. Daher sollte $|e| = k$ erfüllt sein. Sie berechnet anschließend $s = r \prod_{i=1}^{k} s_i^{e_i} \bmod n$ und erhält die Signatur (e, s).

3. Signaturverifikation:
   Um Allys Signatur (e, s) der Nachricht m zu verifizieren, berechnet Billy mit Allys authentischem, öffentlichen Schlüssel $((v_1, v_2, \ldots, v_k), n)$ zuerst eine Zahl $y = s \prod_{i=1}^{k} v_i^{e_i} \bmod n$ und danach den Wert $e' = h(m\|y)$. Er akzeptiert die Signatur nur, wenn $e = e'$ erfüllt ist.

Von besonderer Bedeutung für die vorliegende Arbeit ist die Korrektheitsbedingung der Zero-Knowledge-Beweise, welche besagt, dass das Geheimnis eines Beweisers mit fixiertem Zufallsband extrahiert werden kann. Dies wird nun für das Feige-Fiat-Shamir-Protokoll gezeigt. Wenn nun also die Münzwürfe des Beweisers für das Protokoll fixiert sind, so wird dieser bei jedem Ablauf des Protokolls in jeder Runde identische r und b wählen. Die Challenges $(e_1, \ldots, e_k)$ und $(e'_1, \ldots, e'_k)$ des Verifizierers sind dabei in zwei auffeinanderfolgenden Abläufen in der gleichen Runde höchstens mit einer Wahrscheinlichkeit von 2^{-k} identisch. Für zwei solche nicht-identischen Challenges ist $x_i = \pm y_i^2 \prod_{e_j=1} v_j \bmod n$ und $x'_i = \pm y'^2_i \prod_{e'_j=1} v_j \bmod n$. Werden die Challenges nun aber so gewählt, dass nur ein einzelnes Bit in beiden unterschiedlich ist, während alle anderen Bits gleich sind, dann kann das entsprechende s_i berechnet werden, wenn $e_i \neq e'_i$ ist. Denn es gilt $x_i = (-1)^b r^2 \bmod n$ und ebenso $x'_i = (-1)^{b'} r'^2 \bmod n$, aber aufgrund der fixierten Zufallsbänder ist $r = r'$ und $b = b'$ und deswegen auch $x_i = x'_i$. Damit folgt wegen $e_j = e'_j$ für $j \neq i$ aus $\pm y_i^2 \prod_{e_j=1} v_j = \pm y'^2_i \prod_{e'_j=1} v_j \bmod n$ direkt $(y'/y)^2 = \pm v_i^{-1} \bmod n$. Aufgrund spezieller Eigenschaften[2] von n ist deswegen $s_i = y'/y \bmod n$. Diese Idee lässt sich nun ebenfalls auf das Signaturverfahren übertragen und bildet die Grundlage der damit verbundenen Sicherheitsanalysen im Random-Oracle-Modell in Kapitel 6.

[2] Die Zahl n ist eine *Blum-Zahl* und deswegen hat entweder v_i^{-1} oder $-v_i^{-1}$ eine Quadratwurzel modulo n, aber nicht beide. Vgl. [MOV97]

Eine umfassende Abhandlung mit sämtlichen Beweisen der in diesem Kapitel angesprochenen Themen kann, neben den schon erwähnten speziellen Quellen, [Gol99, BB93, Bie98] entnommen werden.

4 Elliptische Kurven

4.1 Einführung

Elliptische Kurven verdanken ihren Namen dem Versuch, den Umfang von Ellipsen zu berechnen. Dieser ist $U = 4aE(\epsilon)$ mit dem *elliptischen Integral* $E(\epsilon) = \int_0^{\pi/2} \sqrt{1 - \epsilon^2 \sin^2 \varphi}\ d\varphi$. Das Integral ist eines einer Familie von Integralen der allgemeinen Form $\int R(z, \sqrt{p(z)})\ dz$, wobei R eine beliebige rationale Funktion bezeichnet und p ein Polynom, das im Falle der elliptischen Integrale vom Grad drei oder vier ist.[Teu96]

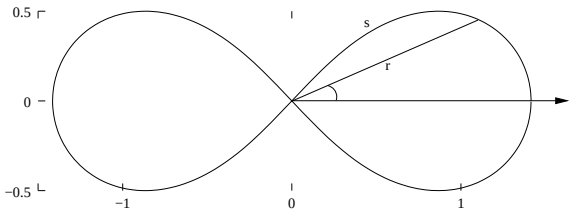

Abbildung 4.1: Lemniskate $(x^2 + y^2)^2 - 2a^2(x^2 - y^2) = 0, a = 1$

Gauß beschäftigte sich 1796 mit der Leminiskate (Abb. 4.1) mit der Frage, wie aus der durch das elliptische Integral $s(r) = \int_0^r d\varrho\big(\sqrt{1-\varrho^4}\big)^{-1}$ bestimmten Bogenlänge s der Abstand r eines Punktes zum Koordinatenursprung berechnet werden kann. Er entwickelte dabei die *lemniskatische Sinus-* und *Kosinusfunktion*, die im Gegensatz zu den trigonometrischen Funktionen doppeltperiodisch sind, mit einer reellen Periode ω_1 und einer imaginären Periode ω_2 im komplexen Zahlenraum $\mathbb{C}$. Meromorphe Funktionen mit diesen Eigenschaften heißen *elliptische Funktionen* und sind definiert durch $f(z + \omega_j) = f(z)$ für alle $z \in \mathbb{C}$ und $j \in \{1, 2\}$.

Elliptische Funktionen spannen durch ihre Periodizität ein Gitter in $\mathbb{C}$ auf, welches durch $\Gamma := \{n\omega_1 + m\omega_2\}$ mit $n, m \in \mathbb{Z}$ definiert und eine additive Gruppe ist. Alle elliptischen Funktionen können durch rationale Kombinationen der Weierstraßschen $\wp$-Funktion und ihrer Ableitung $\wp'$

$$\begin{aligned} \wp(z) &:= \frac{1}{z^2} + \sum_{g \in \Gamma, g \neq 0} \left(\frac{1}{(z-g)^2} - \frac{1}{g^2} \right) \\ \wp'(z) &= -2 \sum_{g \in \Gamma} \frac{1}{(z-g)^3} \end{aligned}$$

konstruiert werden, sie bilden daher den Körper $\mathbb{C}(\Gamma) = \mathbb{C}(\wp(z), \wp'(z))$ [Sil85].

Die Funktionen $\wp(z)$ und $\wp'(z)$ erfüllen außerdem die Differentialgleichung

$$\wp'^2(z) = 4\wp^3(z) - g_2\wp(z) - g_3 \tag{4.1}$$

mit den Konstanten g_2, g_3 definiert durch $e_1 := \wp(\frac{\omega_1}{2}), e_2 := \wp(\frac{\omega_1+\omega_2}{2})$, sowie $e_3 := \wp(\frac{\omega_2}{2})$ als $g_2 := -4(e_1e_2 + e_1e_3 + e_2e_3)$ und $g_3 := 4e_1e_2e_3$. Also sind die Konstanten g_2, g_3 abhängig von dem zugrundeliegenden Gitter Γ.

Wird nun die Abbildung $\phi_\Gamma : z \mapsto (\wp(z), \wp'(z))$ definiert, so geht diese aufgrund der Pole von $\wp(z)$ und $\wp'(z)$ für $z = 0$ von $\mathbb{C}$ nach $\bar{\mathbb{C}} \times \bar{\mathbb{C}}$ mit $\bar{\mathbb{C}} = \mathbb{C} \cup \{\infty\}$. Durch die vom Gitter Γ vorgegebene Periodizität erhält man auf kanonische Weise eine Abbildung $\mathbb{C}/\Gamma \to \bar{\mathbb{C}} \times \bar{\mathbb{C}}$, deren Bild wegen (4.1) lediglich diejenigen $(x, y) \in \bar{\mathbb{C}} \times \bar{\mathbb{C}}$ enthält, welche die Gleichung

$$y^2 \;=\; 4x^3 - g_2 x - g_3 \tag{4.2}$$

erfüllen. Diese Menge, zuzüglich des durch die Pole erforderlichen speziellen Punktes $\mathcal{O} = \phi(0) = (\infty, \infty)$, wird mit $E(\mathbb{C})$ bezeichnet und ist eine elliptische Kurve über $\mathbb{C}$. Gleichungen der Form $y^2 = x^3 + ax + b$ werden als *reduzierte Weierstraß-Normalform* einer elliptischen Kurve bezeichnet.

Mit der im Folgenden beschriebenen Addition auf $E(\mathbb{C})$ wird die bijektive Abbildung $\phi_\Gamma : \mathbb{C}/\Gamma \to E(\mathbb{C}) : z \mapsto (\wp(z), \wp'(z))$ ein Isomorphismus und $E(\mathbb{C})$ eine additive Gruppe. Eine Methode der Addition besteht darin, die Urbilder zweier Punkte P_1, P_2 zu bestimmen, sie in $\mathbb{C}/\Gamma$ zu addieren und das Ergebnis nach $E(\mathbb{C})$ abzubilden. Das neutrale Element der Gruppe $(0 + \Gamma)$ entspricht dem durch die Pole definierten Punkt im Unendlichen. Das Inverse $-P$ eines Punktes P kann ebenfalls über dessen Urbild mit dem Inversen $-z+\Gamma$ von $z+\Gamma$ bestimmt werden. Also ist $-P = (\wp(-z), \wp'(-z))$. Nun ist $\wp$ gerade und somit $\wp(z) = \wp(-z)$, während $\wp'$ ungerade ist, woraus für das Inverse des Punktes $P = (x, y)$ direkt $-P = (x, -y)$ folgt. Anschaulich erhält man also das Inverse eines Punktes durch Spiegelung an der x-Achse.

Um direkt in elliptischen Kurven zu rechnen, wird das Additionstheorem der $\wp$ Funktion benötigt[3]:

$$\wp(z + w) \;=\; -\wp(z) - \wp(w) + \frac{1}{4}\left(\frac{\wp'(z) - \wp'(w)}{\wp(z) - \wp(w)}\right)^2 \tag{4.3}$$

Damit erhält man für einen Punkt $P_3 = P_1 + P_2$ mit $P_i = (x_i, y_i), i = 1, 2, 3$

$$x_3 = -x_1 - x_2 + \frac{1}{4}\left(\frac{y_1 - y_2}{x_1 - x_2}\right)^2.$$

Hierfür gibt es eine anschaulich geometrische Interpretation. Die Punkte $\phi(z)$ und $\phi(w)$ liegen beide auf der Kurve $y^2 \;=\; 4x^3 - g_2 x - g_3$. Zieht man durch diese eine Gerade, so findet man einen dritten Schnittpunkt[4] mit den Koordinaten $(\wp(z + w), -\wp'(z + w)) = \phi(-z - w)$. Die Summe $\phi(z + w)$ ergibt sich durch Spiegelung dieses Punktes an der x-Achse. Es ergibt sich y_3', die y-Koordinate des ungespiegelten Punktes, indem x_3 in eine Geradengleichung $y = mx + c$ eingesetzt wird:

$$y_3' = \frac{y_1 - y_2}{x_1 - x_2} x_3 + c.$$

[3] In der Literatur wird i.d.R. umgekehrt argumentiert: Die Gruppenoperation wird geometrisch definiert und daraus das Additionstheorem hergeleitet. Historisch wurde die Gleichung jedoch 1834 von Jacobi aus dem Eulerschen Additionstheorem für elliptische Integrale abgeleitet. [Teu96] Kapitel 3.8.1.3. Der Beweis für die Korrektheit der geometrischen Analogie steht in [Lem99].

[4] Dieses Verfahren ist auch als *Diophantische Sekantenmethode* bekannt.

Zusätzlich ist aber auch $y_1 = \frac{y_2-y_1}{x_2-x_1}x_1 + c$ und es ergibt sich durch Auflösen beider Gleichungen nach c und Gleichsetzen

$$y_3' = y_1 - \frac{y_1 - y_2}{x_1 - x_2}(x_1 - x_3).$$

Die abschließende Spiegelung führt dann zu

$$y_3 = -y_1 + \frac{y_1 - y_2}{x_1 - x_2}(x_1 - x_3).$$

Mit $P_1 = -P_2$ folgt $P_3 = \mathcal{O}$, wie einfaches Einsetzen verdeutlicht. Der dritte Schnittpunkt mit der Geraden ist hier $\mathcal{O}$ ubnd liegt im Unendlichen.

Im Falle der Verdopplung ($P_3 = P_1 + P_2$ mit $P_1 = P_2$) wird der eingeklammerte Term von (4.3) mit $z = w$ und $z + w = 2z$ zu einem Ausdruck der Form $\frac{0}{0}$. Daher folgt mit der Regel von l'Hospital:

$$\wp(2z) \;=\; -2\wp(z) + \frac{1}{4}\left(\frac{\wp''(z)}{\wp'(z)}\right)^2.$$

Die Gerade ist nun eine Tangente an P_1, welche die Kurve in einem einzigen weiteren Punkt schneidet, der schließlich ebenfalls an der x-Achse gespiegelt werden muss, um $P_3 = 2P_1$ zu erhalten. Damit gilt

$$x_3 \;=\; -2x_1 + \frac{1}{4}\left(\frac{y_1'}{y_1}\right)^2.$$

Das totale Differential von (4.2) ist $y' = \frac{1}{2}(12x^2 - g_2)$ und es folgt

$$x_3 \;=\; -2x_1 + \frac{1}{4}\left(\frac{12x_1^2 - g_2}{2y_1}\right)^2.$$

Erneutes Einsetzen in Geradengleichungen wie bei der Addition zuvor und eine nachfolgende Spiegelung an der x-Achse führt schließlich zu

$$y_3 \;=\; \frac{12x_1^2 - g_2}{2y_1}(x_1 - x_3) - y_1.$$

Näheres und eine vollständigere Betrachtung inklusive der hier fehlenden Beweise findet sich in [Lem99], Kapitel 1 und [Sil85], Kapitel 3 und auch in [Teu96], an mehreren Stellen in Kapitel 1.14.17 ff. und Kapitel 3.8.

4.2 Verallgemeinerung und andere Gruppen

Elliptische Kurven und die beschriebenen Gruppenoperationen können auf sämtliche Körper übertragen werden. Eine besondere Position bei der Übertragung nimmt das neutrale Element $\mathcal{O}$ ein, dessen Herleitung jedoch sehr einfach ist, wenn in projektiven Räumen gerechnet wird. Dazu wird die elliptische Kurve zuerst als eine *homogene Form* geschrieben, beispielsweise wäre im Falle der reduzierten Weierstraß-Gleichung das homogene Polynom über X, Y und Z mit

$$Y^2Z \;=\; X^3 + aXZ^2 + bZ^3 \tag{4.4}$$

gegeben, womit jedes Monom den Grad drei hat. Nun gibt es für diese Gleichung, die eine ebene Kurve beschreibt, nicht mehr eindeutige Lösungen (x, y, z), sondern eine Menge äquivalenter homogener Koordinaten (tx, ty, tz) und es gilt $f(tx, ty, tz) = t^3 f(x, y, z)$. Das wird mit der Abbildung $\mathbb{A}^2(K) \mapsto \mathbb{P}^2(K) : (x, y) \mapsto (X, Y, 1)$ und ihrer Umkehrung $(X, Y, Z) \mapsto (X/Z, Y/Z)$ für $Z \neq 0$ deutlich. Damit entsprechen die Punkte im projektiven Raum $\mathbb{P}^2$ Geraden im affinen Raum $\mathbb{A}^3$, die durch den Koordinatenursprung verlaufen. Den zuvor ausgeschlossenen Sonderfall bilden hierbei die Punkte $(X, Y, 0)$, welche als *Gerade im Unendlichen* bezeichnet werden.

Die Lösungsmenge von (4.4) besteht aus allen $(x, y, z) \in \mathbb{P}^2(K)$, für die $F(x, y, z) = 0$. Damit existieren zwei Klassen von Lösungen: $z \neq 0$ wofür mit $z = 1$ die ursprünglichen Lösungen der Kurve $y^2 = x^3 + ax + b$ in affinen Koordinaten gegeben sind, und $z = 0$, womit $x = 0$ und y beliebig folgt. Letzterer ist der Schnittpunkt $(0, 1, 0)$ der Kurve mit der Geraden im Unendlichen, welcher das neutrale Element $\mathcal{O}$ bildet. Alle Punkte bilden zusammen eine elliptische Kurve $E(K)$ über K.

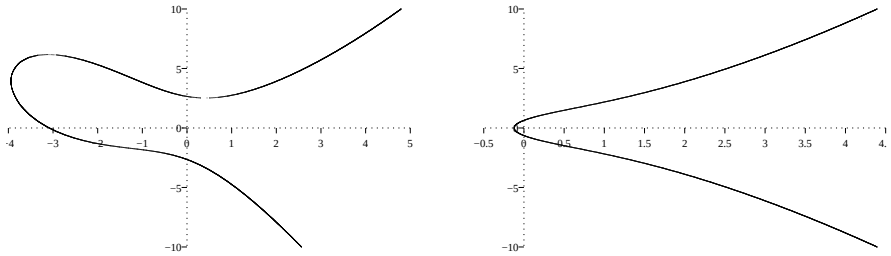

Abbildung 4.2: Links: $f(x, y) = y^2 + 2xy - x^3 - 3x^2 - 2x - 7$ (allgemeine Form), rechts: $f(x, y) = y^2 - x^3 - 3\frac{1}{3}x - 0.\overline{407}$ (reduzierte Form)

Zusätzlich zur reduzierten gibt es noch eine *lange Weierstraß-Gleichung* oder *allgemeine Weierstraß-Normalform* , welche in affinen Koordinaten gegeben ist durch

$$y^2 + a_1xy + a_3y \;=\; x^3 + a_2x^2 + a_4x + a_6. \tag{4.5}$$

Dazu werden üblicherweise die folgenden Konstanten definiert, um nachfolgende Gleichungen vereinfacht darstellen zu können.

$$\begin{aligned} b_2 &= a_1^2 + 4a_2 \\ b_4 &= a_1a_3 + 2a_4 \\ b_6 &= a_3^2 + 4a_6 \\ b_8 &= a_1^2a_6 - a_1a_3a_4 + 4a_2a_6 + a_2a_3^2 - a_4^2 \\ 4b_8 &= b_2b_6 - b_4^2 \\ c_4 &= b_2^2 - 24b_4 \\ c_6 &= -b_2^3 + 36b_2b_4 - 216b_6 \end{aligned} \tag{4.6}$$

Die reduzierte Normalform geht aus der allgemeinen Normalform durch eine *zulässige Transformation* der Variablen hervor, hierzu muss jedoch für die Charakteristik des verwendeten Körpers $\mathrm{char}(K) \neq 2, 3$ gelten. Die Charakteristik ist als die Zahl m definiert, für die in einem Körper K mit dem Einselement e die Gleichung $me = 0$ unter der Bedingung $ne \neq 0, n = 1, 2...m-1$ erfüllt ist. Existiert keine solche Zahl m, wie etwa in $\mathbb{R}$, so hat der Körper die Charakteristik $\mathrm{char}(K) = 0$, ansonsten ist m immer eine Primzahl.

Als zulässig werden Transformationen bezeichnet, die der Form $x' = u^2x + r$ und $y' = u^3y + su^2x + t$ für $r, s, t, u \in K$ und $u \neq 0$ genügen. Diese Substitution ist ein Isomorphismus, der eine elliptische Kurve E in eine elliptische Kurve E' abbildet. In Abbildung 4.2 ist eine elliptische Kurve in der allgemeinen und der reudzierten Form dargestellt.

Gilt nun $\mathrm{char}(K) \neq 2$, so folgt mit $y' = y + \frac{1}{2}(a_1x + a_3)$ aus (4.5)

$$y'^2 = x^3 + \frac{b_2}{4}x^2 + \frac{b_4}{2}x + \frac{b_6}{4}.$$

Gilt zusätzlich $\mathrm{char}(K) \neq 3$, folgt mit $x' = x + b_2/12$

$$y'^2 = x'^3 - \frac{c_4}{48}x' - \frac{c_6}{864},$$

was üblicherweise als

$$y^2 = x^3 + ax + b \tag{4.7}$$

aufgefasst wird.

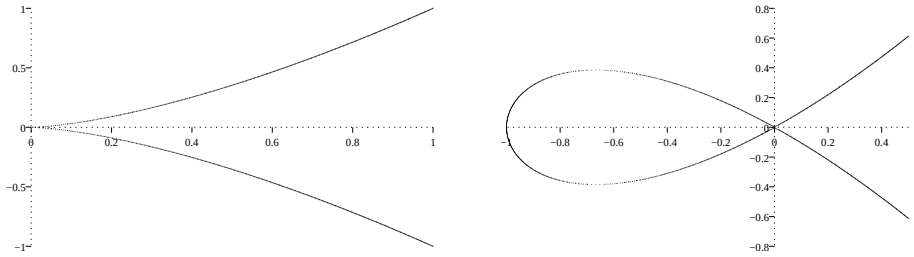

Abbildung 4.3: Links: $f(x, y) = y^2 - x^3$, rechts: $f(x, y) = y^2 - x^3 - x^2$

Ebenfalls mit den Koeffizienten aus (4.6) wird die Diskriminante von (4.5) definiert als

$$\Delta = -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6.$$

und für den Fall $\mathrm{char}(K) \neq 2, 3$ gilt $2^6 3^3 \Delta = 1728\Delta = c_4^3 + c_6^2$. Für die reduzierte Form (4.7) ist $\Delta = -16(4a^3 + 27b^2)$. Ist für eine elliptische Kurve $\Delta = 0$, so heißt sie singulär. Eine singuläre Kurve besitzt mindestens einem Punkt $P = (x, y, z)$, für den

$$\frac{\partial}{\partial x}f(x,y,z) = \frac{\partial}{\partial y}f(x,y,z) = \frac{\partial}{\partial z}f(x,y,z) = 0$$

erfüllt ist. Existiert kein solcher Punkt, heißt die Kurve *glatt*. Abbildung 4.3 zeigt zwei singuläre elliptische Kurven. Die linke ist in der langen Weierstraß-Normalform gegeben und es ist $c_4 = 0$, während für die rechte $c_4 \neq 0$ erfüllt ist. Jedoch lassen sich sämtliche singulären elliptischen Kurven in eine Kurve abbilden, die einer Gleichung der Form $y^2 = x^3 + kx^2$ genügen.

Wie bereits in Kapitel 4.1 gezeigt, bilden die Punkte einer elliptischen Kurve eine additive Gruppe mit dem neutralen Element $\mathcal{O}$. Für das Inverse eines Punktes $P = (x, y)$ gilt $-P = (x, -y)$. Für $P_3 = P_1 + P_2$ müssen drei Fälle unterschieden werden. Der einfachste ist für $P_1 = -P_2$ gegeben, wofür $P_3 = \mathcal{O}$ folgt. Die folgenden Gleichungen erhält man durch Schneiden von Geraden mit der Kurve, wobei P_1, P_2 und P_3' die Schnittpunkte der ersten Gerade mit der Kurve repräsentieren, und die Summe P_3 der drei Punkte durch den Schnitt einer weiteren Gerade durch $\mathcal{O}, P_3'$ und P_3 gegeben ist, was einer Spiegelung von P_3' an der x-Achse entspricht. Abbildung 4.4 zeigt die Verdopplung eines Punktes und die Addition zweier nicht identischer Punkte.

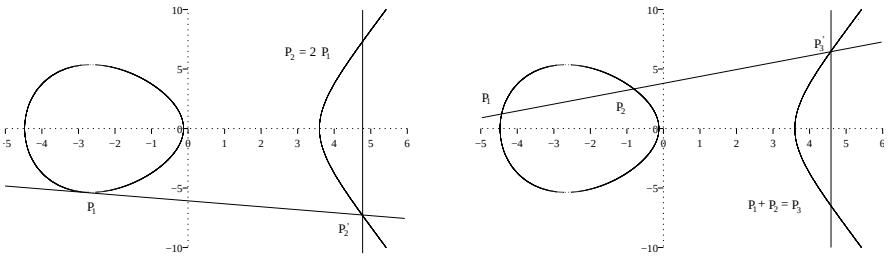

Abbildung 4.4: Addition auf elliptischen Kurven mittels Geraden.

1. $P_1 \neq P_2$:
   Für $(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$ gilt bei elliptischen Kurven in der allgemeinen Weierstraß-Normalform:

$$\begin{aligned} x_3 &= -x_1 - x_2 - a_2 + a_1\left(\frac{y_2 - y_1}{x_2 - x_1}\right) + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 \\ y_3 &= -y_1 - (x_3 - x_1)\left(\frac{y_2 - y_1}{x_2 - x_1}\right) - a_1 x_3 - a_3 \end{aligned} \tag{4.8}$$

   Für die reduzierte Weierstraß-Normalform haben diese Rechenvorschriften die folgende Form:

$$\begin{aligned} x_3 &= -x_1 - x_2 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 \\ y_3 &= -y_1 - (x_3 - x_1)\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \end{aligned} \tag{4.9}$$

2. $P_1 = P_2$:
   Für $(x_3, y_3) = 2(x_1, y_1)$ gilt bei elliptischen Kurven in der allgemeinen Weierstraß-Normalform:

$$\begin{aligned} x_3 &= -2x_1 - a_2 + a_1 \left(\frac{3x_1 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 - a_1x_1} \right) \\ &\quad + \left(\frac{3x_1 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 - a_1x_1} \right)^2 \\ y_3 &= -y_1 - (x_1 - x_3) \left(\frac{3x_1 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 - a_1x_1} \right) - a_1x_3 - a_3 \end{aligned} \tag{4.10}$$

   Für die reduzierte Weierstraß-Normalform gilt stattdessen:

$$\begin{aligned} x_3 &= -2x_1 + \left(\frac{3x_1^2 + a}{2y_1} \right)^2 \\ y_3 &= -y_1 - (x_3 - x_1) \left(\frac{3x_1^2 + a}{2y_1} \right) \end{aligned} \tag{4.11}$$

Als Einführung in die Theorie der elliptischen Kurven sind [Lem99] und [Ham00] sehr zu empfehlen, während [Sil85] eine ausgesprochen umfangreiche Besprechung enthält. In allen, aber insbesondere in letzterem, finden sich auch alle Beweise zum oben Beschriebenen.

4.3 Elliptische Kurven über endliche Körper

Für kryptographische Anwendungen sind elliptische Kurven über Körper mit einer Charakteristik $\mathrm{char}(K) \neq 0$ von Interesse, speziell $\mathbb{F}_p$ mit $\mathrm{char}(\mathbb{F}_p) = p$ für eine große Primzahl p und $\mathbb{F}_{2^m}$ mit $\mathrm{char}(\mathbb{F}_{2^m}) = 2$. Beide Klassen von Körper lassen sich einheitlich durch $\mathbb{F}_q$ mit $q = p^m$ bezeichnen, wobei p eine Primzahl repräsentiert und $m = 1$ für $p \neq 2$.

Eine wichtige Größe stellt dabei die Ordnung der resultierenden Kurve $E(\mathbb{F}_q)$ dar, die aber nicht ad hoc angegeben oder einfach berechnet werden kann, sondern mittels Punktezählalgorithmen bestimmt wird oder bereits eine Vorgabe bei der Erzeugung von elliptischen Kurven ist. Jedoch kann die Punktezahl $\#E(\mathbb{F}_q)$ und damit die Ordnung auf das sogenannte *Hasse-Intervall* eingeschränkt werden, welches durch

$$\#E(\mathbb{F}_q) = q + 1 - t \tag{4.12}$$

für $|t| \leq 2\sqrt{q}$ bestimmt ist. Die Größe t heißt *Frobenius-Spur* und ist ein wichtiges Kriterium bei der Auswahl von Kurven, die für kryptographische Zwecke geeignet sind. Darauf wird zusammen mit weiteren Bedingungen und Anforderungen noch genauer eingegangen. In den beiden folgenden Abschnitten wird die Arithmetik sowohl in $\mathbb{F}_q$ als auch in $E(\mathbb{F}_q)$ beschrieben. Weitere Informationen über endliche Körper finden sich etwa in [Buc00] und [Ham00], während sich Näheres über elliptische Kurven mit diesen Körpern und deren Ordnung unter anderem in [BSS99, Ros99, Ham00] und auch in [Sil85] findet.

4.3.1 Elliptische Kurven über $\mathbb{F}_p$

Die Menge $\{0, 1, ..., p-1\}$ bildet für eine Primzahl p zusammen mit den Operationen $+$ und $\cdot$ den Körper $(\mathbb{F}_p, +, \cdot)$. Die Operationen heißen *Addition modulo* p respektive *Multiplikation modulo* p und sind definiert als $c = a+b$ und $d = a \cdot b$ mit $c, d \in \{0, 1, ..., p-1\}$, so dass $a + b = c + ip$ und $a \cdot b = d + jp$ für $i, j \in \mathbb{N}$ erfüllt ist. Man schreibt dafür $c = a+b \bmod p$ und $d = a \cdot b \bmod p$. Im folgenden werden in der Notation eines Körpers die Operationen $(+, \cdot)$ vorausgesetzt und nicht explizit angegeben.

In diesem Zusammenhang ist auch der Begriff der *Kongruenz* wichtig: a ist kongruent $b \bmod p$, falls $p|(b-a)$. Man schreibt dafür $a \equiv b \bmod p$. Das *multiplikative Inverse* ist damit ein c, das $c \cdot a \equiv 1 \bmod p$ erfüllt. Diese Inversen werden mit Hilfe des erweiterten euklidschen Algorithmus berechnet, der beispielsweise in [Buc00] beschrieben wird.

Ein solcher Körper besitzt mindestens eine *Primitivwurzel*. Primitivwurzeln sind diejenigen Elemente, für die $a = g^m \bmod p$ für alle $m \in \mathbb{F}_p \backslash \{0\}$ ebenfalls sämtliche Werte aus $\mathbb{F}_p \backslash \{0\}$ annimmt. Formal erzeugt jedes Element eine Untergruppe einer Ordnung n von $\mathbb{F}_p$, wobei für eine Primitivwurzel $n = p - 1$ gilt. Somit ist die in diesem Fall erzeugte Untergruppe identisch mit der multiplikativen Gruppe von $\mathbb{F}_p$. Falls ein Element keine Primitivwurzel ist, enthält die Untergruppe lediglich eine Teilmenge von $\mathbb{F}_p$, deren Ordnung $n < p - 1$ ist.

Wie bereits erwähnt, können für den Fall $p \neq 2, 3$ die elliptischen Kurven in reduzierter Form (4.7) verwendet werden, für welche die vereinfachten Gruppenoperationen (4.9) gelten. Die Addition und Multiplikation werden nun lediglich durch die entsprechenden modulo-Operationen in $\mathbb{F}_p$ ersetzt, was anhand des folgenden Beispiels demonstriert werden soll.

Beispiel 4.1 *Sei $p = 29$ und damit $\mathbb{F}_{29} = \{0, 1, \ldots, 28\}$ und eine elliptische Kurve in reduzierter Weierstraß-Normalform durch die Parameter $a = 1$ und $b = 6$ als $y^2 = x^3 + x + 6$ gegeben. Für diese Kurve gilt $\Delta = -16(4a^3 + 27b^2) = -16(4 + 27 \cdot 36) = -15616 \equiv 15 \bmod 29$ und mit (4.12) erhält man für die Punktezahl der elliptischen Kurve das Intervall $\#E(\mathbb{F}_{29}) \in [20, 40]$. Tatsächlich besitzt diese Kurve neben dem Punkt $\mathcal{O}$ die folgenden 37 Punkte:*

$$(0,8)\ (0,21)\ (2,4)\ (2,25)\ (4,4)\ (4,25)\ (8,2)\ (8,27)\ (16,0)\ (3,6)\ (3,23)$$
$$(6,5)\ (6,24)\ (12,8)\ (12,21)\ (14,3)\ (14,26)\ (28,2)\ (28,27)\ (27,5)\ (27,24)$$
$$(25,5)\ (25,24)\ (26,11)\ (26,18)\ (23,4)\ (23,25)\ (17,8)\ (17,21)\ (5,7)\ (5,22)$$
$$(10,1)\ (10,28)\ (20,14)\ (20,15)\ (22,2)\ (22,27),$$

womit also $\#E(\mathbb{F}_{29}) = 38 = p + 1 - t = 29 + 1 + 8$, und damit $t = -8$ folgt.

Werden nun die Punkte $P_1 = (8, 27)$ und $P_2 = (17, 21)$ addiert, so erhält man als Resultat den Punkt $P_3 = (27, 5)$, wie die folgende Rechnung zeigt:

$$\begin{aligned} x_3 &= -x_1 - x_2 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 = -8 - 17 + \left(\frac{21 - 27}{17 - 8}\right)^2 \\ &= 4 + (23 \cdot 13)^2 = 89405 \equiv \mathbf{27 \bmod 29} \end{aligned}$$

$$\begin{aligned} y_3 &= -y_1 - (x_3 - x_1)\left(\frac{y_2 - y_1}{x_2 - x_1}\right) = -27 - (27 - 8)\left(\frac{21 - 27}{17 - 8}\right) \\ &= 2 - 19 \cdot 9 = -169 \equiv \mathbf{5 \bmod 29} \end{aligned}$$

Soll das Resultat dieser Addition verdoppelt werden, so ist $P_3 = 2P_1$ *mit* $P_1 = (27, 5)$ *der Punkt* $P_3 = (17, 8)$*, denn mit (4.11) wird die folgende Rechnung ausgeführt:*

$$\begin{aligned} x_3 &= -2x_1 + \left(\frac{3x_1^2 + a}{2y_1}\right)^2 = -2 \cdot 27 + \left(\frac{3 \cdot (27)^2 + 1}{2 \cdot 5}\right)^2 \\ &= 4 + (13 \cdot 3)^2 = 1525 \equiv \mathbf{17 \bmod 29} \\ y_3 &= -y_1 - (x_3 - x_1)\left(\frac{3x_1^2 + a}{2y_1}\right) = -5 - (17 - 27)\left(\frac{3 \cdot (27)^2 + 1}{2 \cdot 5}\right) \\ &= 24 - 19 \cdot 10 = -166 \equiv \mathbf{8 \bmod 29} \end{aligned}$$

4.3.2 Elliptische Kurven über $\mathbb{F}_{2^m}$

Wie aus dem letzten Abschnitt abgeleitet werden kann, besteht der Körper $\mathbb{F}_2$ aus der Menge $\{0, 1\}$. Definiert man nun eine Körpererweiterung[5] über diesen Körper, die auch als m-dimensionaler Vektorraum aufgefasst werden kann, erhält man den Körper $\mathbb{F}_{2^m}$, dessen multiplikative Gruppe die Ordnung $2^m - 1$ besitzt. Die Komponenten eines Elements eines solchen Körpers, das einem Vektor mit m Dimensionen entspricht, können lediglich die Werte 0 und 1 annehmen.

Es gibt mehrere mögliche Basen für diese Konstruktion, von denen eine im folgenden beschrieben wird. Allen gemein ist die Addition, da diese komponentenweise erfolgt und hier, mit den binären Operanden, der (schnelleren) *XOR*-Verknüpfung entspricht. Die Multiplikation (und damit auch deren neutrales Element) ist jedoch von der verwendeten Basis abhängig.

Allgemein können Elemente also in der Form $a = \sum_{i=0}^{m-1} a_i \xi_i$ dargestellt werden, wobei $a_i \in \{0, 1\}$ ist, und $(\xi_1, \ldots, \xi_m)$ eine Basis in m Dimensionen repräsentiert. Zwischen beliebigen Basen identischer Dimensionalität existiert immer eine bijektive Transformation der Elemente.

Beispiel 4.2 *Für den Fall* $m = 4$ *besteht der Körper* $\mathbb{F}_{2^4}$ *aus den Elementen:*

(0001)	(0011)	(0101)	(0111)	(1001)	(1011)	(1101)	(1111)
(0010)	(0100)	(0110)	(1000)	(1010)	(1100)	(1110)	(0000)

Das Resultat der Addition (1011) + (1001) ist dann das Element (0010). Das neutrale Element der Addition ist mit (0000) gegeben. Interessanterweise ist die Addition in $\mathbb{F}_2$ identisch zu der Substraktion, so ist zum Beispiel $1+1 = 0 \bmod 2$ und ebenfalls $1 - 1 = 0 \bmod 2$.

[5] Diese Konstruktionen, die man allgemein mit p^n, $n = 1, 2, \ldots$ für einen Körper der primen Charakteristik p erhält, heißen *Galois Field*. Der zuvor beschriebenen Körper $\mathbb{F}_p$ findet sich hier als Sonderfall $n = 1$ wieder.

Im folgenden wird die Polynombasis beschrieben. Für Informationen über alternative Basen sei auf [BSS99, Ros99] verwiesen.

In dieser Basis sind alle Elemente durch Polynome repräsentiert, wobei die zuvor beschriebenen Komponenten eines Vektors die Koefizienten der Monome darstellen. Der maximale Grad eines Elementes in $\mathbb{F}_{2^m}$ ist $m-1$. Wie bereits im letzten Abschnitt, so ist auch hier die Multiplikation eine Modulo-Operation. Demzufolge wird eine Möglichkeit der Reduktion benötigt um das Ergebnis einer Operation in die endliche Menge der Elemente abzubilden. Diese Reduktion erfolgt mit Hilfe eines Polynoms der Form $r(x) = x^m + r_{m-1}x^{m-1} + \ldots + r_0$, mit $r_i \in \{0,1\}$ vom Grad m,das zusätzlich *irreduzibel* sein muss, also nicht als das Produkt nicht-konstanter Polynome von Graden kleiner m darstellbar sein darf. Ein zur Reduktion geeignetes irreduzibles Polynom vom Grad 4 ist $r(x) = x^4 + x + 1$.

Beispiel 4.3 *Der Körper $\mathbb{F}_{2^4}$ wird zur Polynomabasis wie folgt dargestellt:*

$1(0001)$	$x^2+1(0101)$	$x^3+1(1001)$	$x^3+x^2+1(1101)$
$x(0010)$	$x^2+x(0110)$	$x^3+x(1010)$	$x^3+x^2+x(1110)$
$x+1(0011)$	$x^2+x+1(0111)$	$x^3+x^2(1100)$	$x^3+x^2+x+1(1111)$
$x^2(0100)$	$x^3(1000)$	$x^3+x+1(1011)$	$0(0000)$

Das Produkt c zweier Elemente aus $a, b \in \mathbb{F}_{2^m}$ zur Polynombasis ist dann definiert als $a(x) \cdot b(x) = c(x) + i(x) \cdot r(x)$ für ein Polynom $i(x)$. Ausgeschrieben gilt für die Multiplikation

$$a(x) \cdot b(x) = \Big(\sum_{i=0}^{m-1} a_i x^i\Big)\Big(\sum_{j=0}^{m-1} b_j x^j\Big) = \sum_{i=0}^{m-1} a_i x^i \Big(\sum_{j=0}^{m-1} b_j x^j\Big).$$

Das Produkt ist also die Summe aller Monome des Polynoms $a(x)$, einzeln multipliziert mit dem vollständigen Polynom $b(x)$, also $a_n b_{m-1} x^{m-1+n} + \ldots + a_n b_1 x^{n+1} + a_n b_j x^n, 0 \leq n < m$. Der Grad jedes einzelnen Monoms von $b(x)$ wird demnach um n erhöht, was in $\mathbb{F}_{2^m}$ einer binären *Left-Shift*-Operation der Komponenten eines Vektors entspricht. Anschließend werden die einzelnen Resultate addiert. So ist beispielsweise $(0101) \cdot (0011) = (x^2+1) \cdot (x+1) = x^2(x+1) + (x+1) = x^3 + x^2 + x + 1$, denn (0011) wird wegen dem ersten Monom von Grad 2 um zwei Stellen verschoben wird und es ergibt sich (1100). Das zweite Monom ist vom Grad 0 und es erfolgt daher kein Shift. (0001) ist also das neutrale Element der Multiplikation. Aus den Einzelresultaten folgt die Summe $(1100) + (0011) = (1111)$. Das resultierende Polynom ist in diesem Fall vom Grad $m-1 = 3$, und bildet deswegen den gesamten Rest bei der Reduktion mit $r(x)$, welches von Grad $m = 4$ ist.

Das Polynom $c(x)$ kann durch die Division nach der Schulmethode bestimmt werden, was in dem folgenden Beispiel demonstriert wird.

Beispiel 4.4 *Multiplikation der Elemente* (1100) *und* (1011) *ergibt das Polynom* $x^6+x^5+x^4+x^2$ *und Division durch* x^4+x+1 *liefert den Rest* x^3+x^2+1*:*

$$
\begin{array}{cccccccccc}
x^6 & +x^5 & +x^4 & & +x^2 & & & : & x^4+x+1 & = & x^2+x+1 \\
x^6 & & & +x^3 & +x^2 & & & & & & \\
\hline
 & x^5 & +x^4 & +x^3 & & & & & & & \\
 & x^5 & & & +x^2 & +x & & & & & \\
\hline
 & & x^4 & +x^3 & +x^2 & +x & & & & & \\
 & & x^4 & & & +x & +1 & & & & \\
\hline
 & & & x^3 & +x^2 & & +1 & & \Rightarrow c(x) & = & (\mathbf{1101})
\end{array}
$$

Damit ist also $(1101) = (1100) \cdot (1011) \bmod r(x)$.

Für die Reduktion existieren schnellere Verfahren, die alle abhängig von der Form der Reduktionspolynome sind. Für sämtliche bisher betrachteten m existieren irreduzible Trinome oder Pentanome. Die Wahl dieser minimierten Reduktionspolynome ist jedoch keineswegs zwingend, und für andere Polynome existieren entsprechend auch andere Reduktionsverfahren. Näheres dazu, mit weiteren Hinweisen auf entsprechende Literatur, kann in [BSS99] nachgelesen werden.

Das Inverse eines Elements wird, wie das für den Körper $\mathbb{F}_p$ der Fall ist, mit Hilfe des erweiterten euklidschen Algorithmus berechnet, wobei lediglich das alternierende Vorzeichen hier nicht benötigt wird. Ziel dieses Verfahrens ist es, zu gegebenen teilerfremden Polynomen $a(x)$ und $r(x)$ zwei Polynome p_n und s_n zu finden, für welche die Bedingung $\gcd(a(x), r(x)) = 1 = p_n a(x) + s_n r(x)$ erfüllt ist. Diese Polynome werden aus je einer Folge bestimmt, deren Elemente $p_{k+1} = q_k p_k + p_{k-1}$ und $s_{k+1} = q_k s_k + s_{k-1}$ aus $q_{k+1} = r_k / r_{k-1}$ mit $r_{k+1} = r_{k-1} \bmod r_k$ für $1 \leq k \leq n$ folgen. Die Berechnung beginnt mit $p_0 = 1$, $s_0 = 0$ sowie $p_1 = 0$ und $s_1 = 1$. Zusätzlich wird $r_0 = r(x)$ und $r_1 = a(x)$ gesetzt. Folgt aus der k-ten Iteration $r_{k+1} = 0$, so bricht der Algorithmus mit dem Ergebnis p_k ab.

Beispiel 4.5 *Es wird das Inverse von* (1100) *in* $\mathbb{F}_{2^4}$ *mit* $r(x) = x^4 + x + 1$ *gesucht.*

k	0	1	2	3	4	5
r_k	$r(x)$	1100	0111	0010	0001	0000
q_k		0011	0010	0011	0010	
p_k	0000	0001	0011	0111	**1010**	$r(x)$
s_k	0001	0000	0001	0010	0111	1100

Das Inverse von (1100) *ist also* (1010)*. Tatsächlich lässt sich verifizieren, dass* $(1100) \cdot (1010) = (0001) \bmod r(x)$ *ist.*

Wie bereits in Abschnitt 4.2 erwähnt, müssen für Körper der Charakteristik 2 elliptische Kurven in der langen Weierstraß-Normalform verwendet werden. In der Praxis werden dabei Kurven verwendet, für die in Gleichung (4.5) die Parameter $a_1 = 1$ und $a_3 = a_4 = 0$ gesetzt wurden, so dass auch hier, wie bei der reduzierten Normalform, nur zwei Parameter benötigt werden um die Kurve zu beschreiben. Eine elliptische Kurve über einen Körper $\mathbb{F}_{2^m}$ is somit durch

$$y^2 + xy \;=\; x^3 + ax^2 + b$$

vollständig spezifiziert, wobei $a, b \in \mathbb{F}_{2^m}$ und $b \neq 0$ erfüllt sein muss. Weil außerdem Substraktion und Addition identisch sind, wird aus den Gleichungen (4.8) für die Addition $(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$ zweier ungleicher Punkte

$$x_3 = x_1 + x_2 + a + \frac{y_2 + y_1}{x_2 + x_1} + \left(\frac{y_2 + y_1}{x_2 + x_1}\right)^2$$

und

$$y_3 = x_3 + y_1 + (x_3 + x_1)\left(\frac{y_2 + y_1}{x_2 + x_1}\right),$$

sowie für die Punktverdopplung $(x_3, y_3) = 2(x_1, y_1)$ aus den Gleichungen 4.10

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

und

$$y_3 = x_1^2 + x_3 + x_3\left(x_1 + \frac{y_1}{x_1}\right).$$

Auch hier ist $P + \mathcal{O} = P$ und $P + (-P) = \mathcal{O}$, wobei für $P = (x, y)$ das Inverse mit $-P = (x, x + y)$ gegeben ist.

Beispiel 4.6 *Sei über den Körper $\mathbb{F}_{2^4}$ eine elliptische Kurve mit den Parametern $a = (0011)$ und $b = (0001)$ als $y^2 + xy = x^3 + (0011)x^2 + (0001)$ gegeben. Mit dem Reduktionspolynom $r(x) = x^4 + x + 1$ hat diese Kurve die folgenden $\#E(\mathbb{F}_{2^4}) = 16$ Punkte:*

(0000, 0001)	(0001, 1100)	(0001, 1101)	(0110, 1000)
(0110, 1110)	(0111, 0010)	(0111, 0101)	(1000, 0101)
(1000, 1101)	(1010, 0111)	(1010, 1101)	(1100, 0101)
(1100, 1001)	(1111, 0000)	(1111, 1111)	$\mathcal{O}$

Dann ist $(1010,\ 1101) + (0110,\ 1000) = (1000,\ 1101)$*, denn es ist*

$$\begin{aligned} x_3 &= (1010) + (0110) + (0011) + \frac{(1000) + (1101)}{(0110) + (1010)} + \left(\frac{(1000) + (1101)}{(0110) + (1010)}\right)^2 \\ &= (1111) + (0100) + (0100)^2 = (1011) + (0011) = (\mathbf{1000}), \\ y_3 &= (1000) + (1101) + ((1000) + 1010))\left(\frac{(1000) + (1101)}{(0110) + (1010)}\right) \\ &= (0101) + (0010)(0100) = (0101) + (1000) = (\mathbf{1101}). \end{aligned}$$

Die Summe $(x, y) = 2(x_3, y_3) = 2(1000,\ 1101) = (0110, 1110)$ *folgt aus*

$$\begin{aligned} x &= (1000)^2 + \frac{(0001)}{(1000)^2} = (1100) + \frac{(0001)}{(1100)} = (1100) + (1010) = (\mathbf{0110}), \\ y &= (1100) + (0110) + (0110)\left((1000) + \frac{(1101)}{(1000)}\right) = \\ &\quad (1010) + (0110)(1111) = (1010) + (0100) = (\mathbf{1110}). \end{aligned}$$

4.4 ECDLP

Mit den beschriebenen Gruppenoperationen Verdopplung und Addition lässt sich eine skalare Multiplikation $mP_1 = \sum_1^m P_1 = P_2$ für die Punkte einer elliptischen Kurve definieren. Diese Operation ist, bei geeigneter Implementation, einfach und effizient durchzuführen. Die umgekehrte Frage nach dem diskreten Logarithmus, also ein m zu finden, so dass $P_2 = mP_1$ gilt, ist wiederum sehr schwer zu lösen, worauf die Sicherheit der kryptographischen Systeme basiert, die elliptische Kurven über endlichen Körpern verwenden. Der naive Ansatz besteht darin, alle Punkte $P' = iP_1$ für $1 \leq i < n$ für die Gruppenordnung n zu berechnen, bis P_2 gefunden wurde. Dieses Vorgehen erfordert daher maximal n Additionen. Schnellere Verfahren können in zwei Gruppen aufgeteilt werden. Zum einen gibt es elliptische Kurven, deren Eigenschaften subexponentielle Algorithmen ermöglichen und die daher für die Verwendung in kryptographischen Protokollen vermieden werden; zum anderen gibt es Kurven, für die keine solche Möglichkeit bekannt ist. Im folgenden werden zunächst die Verfahren und ihre Laufzeit beschrieben, welche für sämtliche elliptischen Kurven den diskreten Logarithmus berechnen können. Dannach wird kurz auf die sogenannten schwachen Kurven eingegangen und Verfahren vorgestellt, die für jene schneller zum Erfolg führen.

4.4.1 Angriffe auf elliptische Kurven

Die in diesem Abschnitt beschriebenen Verfahren gehören der Kategorie der generischen Algorithmen an, mit welchen der diskrete Logarithmus in sämtlichen Gruppen berechnet werden kann. Als erstes wird das Verfahren von Pohlig-Hellmann vorgestellt, welches die Berechung des diskreten Logarithmus einer Gruppe mit Primzahlpotenzordnung auf die Berechnung in Untergruppen mit primer Ordung reduziert. Sei dazu die Ordnung einer elliptischen Kurve mit $n = \#E(\mathbb{F}_q) = \prod_{p_i|n} p_i^{e_i}$ gegeben und gebe es ein $m \in \mathbb{Z}_n$, welches $P_2 = mP_1$ erfüllt. Indem für jeden Primteiler p_i der Gruppenordnung ein $m_{i,0} = m \bmod p_i$ aus

$$P_{2,0} = \big(np_i^{-(e_i-1)}\big)P_2 = m_{i,0}\big(np_i^{-(e_i-1)}\big)P_1$$

berechnet wird und dann aus $m_{i,0}, \ldots, m_{i,k-1}$ (auch für $k = 1$!) der Koeffizient $m_{i,k} = m \bmod p_i^k$ mit

$$P_{2,k} = P_2 - \big(\sum_{j=0}^{k-1} m_{i,j}p_i^j\big)P_1,$$

durch Lösen des diskreten Logarithmus

$$np_i^{-(e-k)}P_{2,k} = m_{i,k}np_i^{-(e_i-1)}P_1$$

bestimmt werden, kann dieses m mit dem Chinesischen Restsatz aus der simultanen Kongruenz $m \equiv m_i \bmod p_i^{e_i}$ für $m_i = \sum_{j=0}^{e-1} m_{i,j}p_i^j$ ermittelt werden. Für jeden Teiler $p_i^{e_i}$ der Gruppenordnung müssen also lediglich e_i diskrete Logarithmen in einer Gruppe der Ordnung p_i berechnet werden. Aus diesem Grund sollte die Gruppenordnung entweder prim sein, oder aber zumindest einen großen Primfaktor enthalten. Die Ordnung der in der Praxis verwendeten Kurven ist tatsächlich $n = hp$ für eine große Primzahl p und $h \in \{1, 2, 4\}$, wobei jedoch in der Untergruppe der Ordnung p gerechnet wird.

Um den diskreten Logarithmus in einer Gruppe mit primer Ordnung zu berechnen, kann zwischen verschiedenen Verfahren gewählt werden, von denen aber der *Pollard-ρ-Algorithmus* am effizientesten ist, auch weil sich dieses Verfahren parallelisieren lässt.[vOW94, Tes98] Dieser Ansatz wird im folgenden für elliptische Kurven beschrieben, kann jedoch für jede endliche Gruppe verwendet werden.

Die Aufgabe ist die Berechnung einer Zahl m, so dass $mA = B$ für zwei Punkte A und B einer elliptischen Kurve erfüllt ist. Das Ziel des Pollard-ρ-Algorithmus ist es, eine Gleichung der Form $P_1 = a_1A + b_1B = a_2A + b_2B = P_2$ zu finden, mit der sich dann aus $m \equiv (a_1 - a_2)(b2 - b1)^{-1} \bmod n$ der diskrete Logarithmus berechnen lässt, falls $a_1 \neq a_2$ und $b_1 \neq b_2$ ist. Dazu werden parallel berechenbare Folgen von Punkten erzeugt, bis zwei identische Punkte gefunden werden. Die Funktion, mit welcher die Punkte dieser Folge bestimmt werden, wird zu Beginn der Berechnung probabilistisch festgelegt. Zuerst wird eine Zahl s benötigt, welche die Anzahl der möglichen Folgepunkte festlegt, sowie eine Funktion $f : E(\mathbb{F}_q) \to \mathbb{Z}_s$, welche aus dem aktuellen Punkt einen der möglichen Folgepunkte auswählt.Dann müssen s Zufallszahlenpaare $a_i, b_i,\ i \in \mathbb{Z}_s$ erzeugt werden, mit denen eine Liste von Punkten der Form $M_i = a_iA + b_iB$ festgelegt wird. Mit dieser Liste wird schliesslich die Funktion $F : P \mapsto P + M_{f(P)}$ festgelegt, und es ergibt sich eine Folge $P_k = F(P_{k-1})$ für einen beliebig gewählten Startpunkt P_0. Da nun aber für eine einmal festgelegte Funktion F jeder Punkt aus einem anderen folgt, müssen nicht sämtliche berechneten Punkte (und deren Koeffizienten a_k, b_k) gespeichert werden, sondern nur diejenigen, welche einer definierten, frei wählbaren Bedingung entsprechen. Auf diese Weise kann der benötigte Speicherplatz, auf Kosten der Laufzeit, entsprechend geringer gehalten werden. Wird nun in einer dieser Folgen ein Punkt erreicht, welcher zuvor in einer beliebigen der n Folgen berechnet, aber nicht gespeichert wurde, dann sind von da an sämtliche Punkte dieser zwei Folgen bereits berechnet worden. Somit wird, entsprechend der gewählten Bedingung, auch tatsächlich ein bereits gespeicherter Punkt erreicht.

Beispiel 4.7 *Sei eine elliptische Kurve über den Körper $\mathbb{F}_{163}$ mit den Parametern $a = 2$ und $b = 4$ gegeben. Diese Kurve hat die Ordnung $\#E(\mathbb{F}_{163}) = 181$, und es soll der diskrete Logarithmus $mA = B$ für die Punkte $B = (127, 100)$ und $A = (7, 19)$, einem Erzeuger der Kurve, gefunden werden. Weil die Gruppe sehr klein ist, wird $s = 4$ gesetzt, und die folgenden zufälligen Multiplikatoren generiert.*

$$M_0 : 2A + 4B = (122, 90) \qquad M_1 : 9A + 5B = (148, 48)$$
$$M_2 : 5A + 10B = (41, 59) \qquad M_3 : 9A + 1B = (112, 111)$$

Die Funktion f ist dann definiert als $f : E(\mathbb{F}_{163}) \to \mathbb{Z}_4$, und wird realisiert durch $f(x_P, y_P) = x_P \bmod 4$, für einen Punkt $P = (x_P, y_P) \in E(\mathbb{F}_{163})$. Der Algorithmus startet mit $P_0 = A$ und $Q_0 = B$. Wenn diejenigen Punkte gespeichert

werden, deren x-Koordinate ungerade ist, so ergibt das den folgenden Ablauf:

k	P_k	Q_k
1	$21A + 23B = (137, 44)$	$9A + 2B = (39, 118)$
2	$35A + 38B = (21, 4)$	$\mathbf{30}A + \mathbf{27}B = (\mathbf{87}, \mathbf{145})$
3	$49A + 53B = (21, 159)$	$44A + 38B = (37, 53)$
4	$58A + 58B = (145, 2)$	$53A + 43B = (109, 78)$
5	$67A + 63B = (87, 18)$	$64A + 52B = (47, 116)$
6	$\mathbf{90}A + \mathbf{92}B = (\mathbf{87}, \mathbf{145})$	$75A + 57B = (115, 112)$

Es ist also $P_6 = Q_2$, weswegen aus $90A + 92B = 30A + 27B$ für den diskreten Logarithmus $116^{-1} \cdot 60A = 142 \cdot 60A = 13A = B$, also $m = 13$, folgt.

Der Pollard-ρ-Algorithmus hat eine erwartete Laufzeit von $O(\sqrt{p\pi/2})$ für eine Kurve der Ordnung p, weswegen für das Beispiel nach etwa 17 Schritten ein Ergebnis erwartet werden kann. In [vOW94] wird gezeigt, dass der Algorithmus mit r Prozessoren implementiert werden kann, so dass für die erwartete Laufzeit $O(\sqrt{\pi p}/(2r))$ gilt. Aufgrund der zufällig festgelegten Punkte, die in jeder Iteration verwendet werden, heißen derartige Methoden *Random Walks*. Es gibt ein alternatives Verfahren, den Pollard-λ-Algorithmus, der eine identische erwartete Laufzeit besitzt, in der Praxis zwar etwas langsamer arbeitet, jedoch geeignet ist, um den diskreten Logarithmus schneller zu berechnen, falls dieser in einem bekannten kleinen Intervall liegt.[BSS99] Allerdings wurde die Laufzeit des Verfahrens in [GLV98] speziell für anomale elliptische Kurven über $\mathbb{F}_{2^m}$ nochmals verbessert.

Näheres zu diesen (und weiteren) Verfahren, auch für Primzahlkörper, kann [Buc00, BSS99, MOV97] entnommen werden.

4.4.2 Schwache Kurven

Die Erläuterung der den in diesem Abschnitt beschriebenen Attacken zugrundeliegenden Eigenschaften von elliptischen Kurven sprengt den Rahmen der vorliegenden Arbeit, weswegen sich die Übersicht lediglich auf die Beschreibung von Merkmalen der betroffenen Kurven und die Idee des resultierenden Verfahrens beschränkt.

Elliptische Kurven über $\mathbb{F}_{2^m}$ mit zusammengesetztem m

Im Jahr 1999 stellte Silverman in [Jac00] ein *Xedni-Calculus* benanntes Verfahren vor, das einen völlig neuen Ansatz einer möglichen Attacke gegen elliptische Kurven über Primkörper großer Charakteristik beschrieb. Für dieses Verfahren wurde jedoch in [SZSH98] gezeigt, dass es in der Praxis nicht erfolgreich durchführbar ist. In [GaS99, GHS00] wurde von Galbraith, Smart, Hess und Gaudry, und unabhängig davon auch von anderen Autoren, ein diesem Verfahren nicht unähnlicher Ansatz für Kurven über $\mathbb{F}_{2^m}$ mit zusammengesetztem m beschrieben, das wiederum in [GHS00] für alle primen m in dem Intervall $m \in [160 - 600]$ als nicht erfolgreich analysiert wurde. Anderserseits wurde gezeigt, dass für den Fall $m = 155$ eine kleine Zahl von Kurven möglicherweise anfällig für diese Attacke sind, und der diskrete Logarithmus dann auf einen

diskreten Logarithmus hyperelliptischer Kurven reduziert werden kann, der für diese speziellen Fälle wahrscheinlich in subexponentieller Zeit lösbar ist. Aus diesem Grund werden alle Körper $\mathbb{F}_{2^m}$ mit einem zusammengesetzten m für die Verwendung in kryptographischen Protokollen mit elliptischen Kurven ausgeschlossen.

Anomale Kurven über $\mathbb{F}_p$

Anomale Kurven besitzen die Frobeniusspur $t = 1$, woraus folgt, dass diese Kurven eine Ordnung $\#E(\mathbb{F}_p) = p$ besitzen. Dies ermöglicht für elliptische Kurven über Primzahlkörper großer Charakteristik die Lösung des diskreten Logarithmus in linearer Zeit, bezogen auf der Anzahl der Gruppenoperationen, was unter anderem in [Sma99] demonstriert wird. Dazu wird das ECDLP von der Kurve $E(\mathbb{F}_p)$ auf den diskreten Logarithmus der identischen Kurve über den Körper $\mathbb{Q}_p$ reduziert.

Supersinguläre Kurven und die MOV-Bedingung

Als *supersingulär* werden elliptische Kurven über einen Körper $\mathbb{F}_q$ bezeichnet, wenn $q = p^m$ für eine Primzahl p ist, die Frobeniusspur $t^2 \in \{0, q, 2q, 3q, 4q\}$ erfüllt und $p|t$ gilt. Für diese Kurven ist es möglich, das Problem des diskreten Logarithmus in elliptischen Kurven auf den diskreten Logarithmus für endliche Körper zu reuzieren. Dieser Angriff heisst *MOV-Reduktion*, nach den Initialen der Autoren Menezes, Okamoto und Vanstone [MOV91]. Dieser Angriff funktioniert auch bei elliptischen Kurven, welche die folgende Beindung erfüllen. Gibt es ein k, so dass für die Ordnung $n = \#E(\mathbb{F}_q)$ einer elliptischen Kurve $n|q^k - 1$ gilt, kann der diskrete Logarithmus auch dann in subexponnentieller Zeit gelöst werden. Dazu wird eine Körpererweiterung definiert, und das Problem in $\mathbb{F}_{q^k}$ berechnet, was in [SZSH98] gezeigt wird. Von praktischem Interesse sind dabei aber lediglich diejenigen Erweiterungen, für die $k \leq 20$ erfüllt ist, weswegen überprüft werden muss, ob für ein $i \in \{1, \ldots, 20\}$ die Ordnung n einer elliptischen Kurve die Bedingung $n|q^i - 1$ erfüllt [X9.62]. Elliptische Kurven, die diese Bedingung (*MOV-Condition*) erfüllen, dürfen nicht verwendet werden.

5 ElGamal-Signaturen

In diesem Teil wird zuerst die ursprüngliche ElGamal-Signatur mit Hashfunktion vorgestellt. Danach folgt die verallgemeinerte Variante, die unabhängig von verwendeten Gruppen beschrieben und anhand von Beispielen mit einem endlichen Körper primer Charakteristik demonstriert wird. Anschließend wird das Meta-ElGamal-Schema vorgestellt, welches das Signaturverfahren für unterschiedliche Permutationen der Signaturerzeugung und Verifikation verallgemeinert. Darauf basierend werden zwei Varianten für die Verwendung mit elliptischen Kurven vorgestellt. Zum Abschluß wird kurz auf die Sicherheit und Effizienz der Verfahrens eingegangen.

5.1 ElGamal Signatur

Das von Taher ElGamal ursprünglich vorgeschlagene Signaturverfahren wurde bereits in Kapitel 2 vorgestellt und die Notwendigkeit einer Hashfunktion begründet. Das resultierende Verfahren benötigt dann eine große Primzahl p und einen Erzeuger g des Körpers $\mathbb{Z}_p$, sowie eine Hashfunktion $h : \{0,1\}^* \to \mathbb{Z}_{p-1}$. Die Signatur (r, s) einer Nachricht m besteht dann aus der Zahl $r = g^k \bmod p$, für ein zufällig gewähltes $k \in \mathbb{Z}^*_{p-1}$ und der Lösung s der Kongruenz $h(m) \equiv ar + ks \bmod p-1$. Die Zahl $a \in \mathbb{Z}^*_{p-1}$ ist dabei der private Schlüssel, während der öffentliche aus diesem mit $y = g^a \bmod p$ berechnet wird. Eine Signatur (r, s) ist gültig für eine Nachricht m, wenn die Kongruenz $g^{h(m)} \equiv y^r r^s \bmod p$ erfüllt ist.

In diesem Verfahren wird also in zwei Gruppen, nämlich $\mathbb{Z}_p$ und $\mathbb{Z}_{p-1}$ gerechnet. Dabei wird die Signaturgleichung in $\mathbb{Z}_{p-1}$ gelöst, während die Verifikation in $\mathbb{Z}_p$ stattfindet. Diese Eigenschaft bildet die Grundlage für die im folgenden beschriebene Verallgemeinerung.

5.2 Verallgemeinerung für beliebige Gruppen

Das Attribut “generisch” bezeichnet hier die Möglichkeit, das Signaturverfahren auf beliebige, endliche, abelsche Gruppen übertragen zu können, wobei jedoch potentielle Gruppen einerseits effizient zu implementierende Gruppenoperationen besitzen sollten, andererseits muss das Problem des diskreten Logarithmus in der Gruppe schwer zu lösen sein, um insgesamt ein möglichst schnelles, aber sicheres Verfahren zu erhalten.

Das Verfahren benötigt zwei optional identische kryptographische Hashfunktionen $h_1, h_2 : \{0,1\}^* \to \mathbb{Z}_n$ und eine Funktion $f : \mathbb{G} \to \{0,1\}^*$, welche Elemente der Gruppe $\mathbb{G}$ mit Ordnung n in Bitstrings abbildet, so dass $h_2(f(a))$ für alle $a \in \mathbb{G}$ definiert ist. Im folgenden wird die Gruppe $\mathbb{G}$ multiplikativ geschrieben, und e sei das neutrale Element in $\mathbb{G}$. Die Anwendung der Funktion f auf a wird dabei vorausgesetzt, so dass statt $h_2(f(a))$ nur $h_2(a)$ geschrieben wird.

5.2.1 Schlüsselerzeugung

Um ein Schlüsselpaar zu erzeugen, wird eine zyklische Gruppe $\mathbb{G}$ der Ordnung n mit einem Erzeuger g benötigt. Der private Schlüssel besteht aus einer zufällig gewählten Zahl $a \in \{1, \ldots, n-1\}$. Damit wird nun ein Gruppenelement $y \in \mathbb{G}$ mit $y = g^a$ berechnet. Der öffentliche Schlüssel besteht aus dem Tupel (g, y).

5.2.2 Signaturerzeugung

Um eine Nachricht $m \in \{0,1\}^*$ zu signieren, wählt Ally zufällig eine *in* $\mathbb{Z}_n$ *invertierbare* Zahl $k \in \{1,\ldots,n-1\}$ und berechnet das Gruppenelement $r = g^k$. Benötigt werden zusätzlich das Inverse k^{-1} von k in $\mathbb{Z}_n$ und die Hashwerte $h_2(r)$ und $h_1(m)$. Mit diesen erhält Ally ein $s \in \mathbb{Z}_n$ durch

$$s \;=\; k^{-1}\big(h_1(m) - ah_2(r)\big) \mod n. \tag{5.1}$$

Die Signatur der Nachricht m besteht aus dem Tupel (r,s).

5.2.3 Signaturverifikation

Um nun die Signatur (r,s) der Nachricht m zu verifizieren, muss Billy zunächst Allys öffentlicher Schlüssel vorliegen und dessen Authentizität sichergestellt sein. Danach berechnet Billy $h_1(m)$ und $h_2(r)$, und die Gruppenelemente v_1 und v_2 mit

$$\begin{aligned} v_1 &= y^{h_2(r)} r^s \\ v_2 &= g^{h_1(m)} \end{aligned} \tag{5.2}$$

und akzeptiert die Signatur nur, wenn $r \neq e$ und $v_1 = v_2$ ist.
Die Verifikationsbedingung folgt direkt aus (5.1), denn es ist

$$\begin{aligned} ks &\equiv h_1(m) - ah_2(r) \bmod n \\ h_1(m) &\equiv ah_2(r) + ks \bmod n \end{aligned}$$

und damit auch

$$g^{h_1(m)} \;=\; g^{ah_2(r)+ks}.$$

Da aber $r = g^k$ und $y = g^a$ gilt, folgt

$$g^{h_1(m)} \;=\; y^{h_2(r)} r^s,$$

und somit ist $v_2 = v_1$.

Beispiel 5.1 *Sei* $\mathbb{G} = \mathbb{F}_{29} = \mathbb{Z}_{29}$ *mit Ordnung* $n = 28$ *und dem Erzeuger* $g = 2$. *Mit dem zufällig gewählten* $a = 23$, *ihrem privaten Schlüssel, berechnet Ally*

$$y = g^a = 2^{23} \equiv 10 \bmod 29$$

. Ihr öffentlicher Schlüssel ist also $(g,y) = (2,10)$. *Für eine beliebige Nachricht* m *sei* $h_1(m) = 9$. *Ally wählt* $k = 5$, *mit dem Inversen* $k^{-1} = 17 \bmod 28$ *und erhält*

$$r = g^k = 2^5 \equiv 3 \bmod 29.$$

Für dieses r *sei* $h_2(r) = 13$. *Damit berechnet sie*

$$s = 17(9 - 23 \cdot 13) \bmod 28 = 26.$$

Die Signatur der Nachricht m *ist also* $(r,s) = (3,26)$.

Billy möchte nun diese Signatur verifizieren. Er berechnet zuerst $h_1(m) = 9$ *und* $h_2(3) = 13$ *und danach*

$$v_1 = 10^{13} \cdot 3^{26} = 26 \cdot 13 \equiv 19 \bmod 29$$

und schließlich

$$v_2 = 2^9 \equiv 19 \bmod 29.$$

Er akzeptiert die Signatur, weil $v_1 = v_2$ *ist.*

5.3 Meta-ElGamal Signatur

In der Literatur wurden etliche Varianten des ElGamal-Signaturverfahrens besprochen und in [HPN94] zu einem Gesamtschema, welches mehr als 13000 Varianten enthält, zusammengefasst. Dieses Schema basiert zwar auf der ursprünglichen Instanziierung mit Primzahlkörpern, jedoch können die meisten Verallgemeinerungen auch in eine Beschreibung für generische Gruppen übernommen werden.

Es wurde in Abschnitt 5.2.3 gezeigt, dass die Signaturgleichung direkt in die Verifikationsgleichung überführt werden kann. Es liegt also nahe, diverse Permutationen der Parameter a, k, s beziehungsweise r, y, g, sowie der Hashwerte $h_1(m)$ und $h_2(r)$ in beiden Gleichungen zu untersuchen. Zusätzlich sind Kombinationen einzelner Parameter mit einer in beiden Argumenten invertierbaren Operation $o : (\mathbb{Z}_n \backslash \{0\}) \times (\mathbb{Z}_n \backslash \{0\}) \to \mathbb{Z}_n$ möglich. Außerdem ist die Frage interessant, ob nicht h_2 durch eine Funktion ersetzt werden kann, an die schwächere Bedingungen gestellt werden, ohne dass die Sicherheit des Verfahrens beeinträchtigt wird. Um diese Möglichkeit offen zu halten, und zusätzlich die Notation zu vereinfachen, sei im folgenden $\bar{r} := h_2(r)$, ohne dass h_2 näher spezifiziert wird. Für die Analyse in Kapitel 6 muss die Berechnung des Hashwerts der Nachricht um den Punkt R erweitert werden, so dass $h(m, R)$ anstatt $h(m)$ berechnet wird. Im Folgenden wird dafür das Symbol $\hbar = h(m, R)$ verwendet.

Zuerst werden die angesprochenen Permutationen betrachtet. Sei (A, B, C) eine Permutierung von $\{\hbar, \bar{r}, s\}$, so kann die Signaturgleichung allgemein als

$$\pm A \equiv \pm Ba \pm Ck \bmod n$$

formuliert werden, wobei die Vorzeichen so gewählt werden sollten, dass keine Multiplikation mit -1 erforderlich ist, um s zu berechnen. Damit ergeben sich folgende Signatur- und Verifikationsgleichungen, wenn die Vorzeichenpermutationen vernachlässigt werden:

Nr.	A	B	C	Signatur	Verifikation
1	$\hbar$	$\bar{r}$	s	$\hbar \equiv \bar{r}a + sk$	$g^{\hbar} = y^{\bar{r}} r^s$
2	$\hbar$	s	$\bar{r}$	$\hbar \equiv sa + \bar{r}k$	$g^{\hbar} = y^s r^{\bar{r}}$
3	s	$\bar{r}$	$\hbar$	$s \equiv \bar{r}a + \hbar k$	$g^s = y^{\bar{r}} r^{\hbar}$
4	s	$\hbar$	$\bar{r}$	$s \equiv \hbar a + \bar{r}k$	$g^s = y^{\hbar} r^{\bar{r}}$
5	$\bar{r}$	s	$\hbar$	$\bar{r} \equiv sa + \hbar k$	$g^{\bar{r}} = y^s r^{\hbar}$
6	$\bar{r}$	$\hbar$	s	$\bar{r} \equiv \hbar a + sk$	$g^{\bar{r}} = y^{\hbar} r^s$

Um die angesprochene Möglichkeit zur Kombination von Parametern beschreiben, muss zunächst die erwähnte Operation spezifiziert werden. Diese sei kommutativ, so dass $o(x,y) = o(y,x),\ x,y \in \mathbb{Z}_n$ gilt, sowie invertierbar auf $(\mathbb{Z}_n\backslash\{0\}) \times (\mathbb{Z}_n\backslash\{0\})$ in beiden Argumenten, so dass $o^{-1}(o(x,y),x) = y$ und $o^{-1}(o(x,y),y) = x$ für alle $x,y \in \mathbb{Z}_n\backslash\{0\}$ erfüllt ist. Diese Bedingungen werden beispielsweise von der Addition und der Multiplikation erfüllt. Mit zwei Operationen o_1 und o_2 lassen sich dann die folgenden Typen von Signatur- und Verifikationsgleichungen definieren, wobei für jeden dieser Typen die bereits beschriebenen sechs Permutationen der Parameter möglich sind:

Typ	$(\pm A, \pm B, \pm C)$-Permutation		
EG-I	$\hbar$	$\bar{r}$	s
EG-II	$o_1(\hbar, \bar{r})$	s	1
EG-III	$o_1(\hbar, \bar{r})$	$o_2(\hbar, s)$	1
EG-IV	$o_1(\hbar, s)$	$o_2(\bar{r}, s)$	1
EG-V	$o_1(\hbar, \bar{r})$	$o_2(\bar{r}, s)$	1

Die eingangs vorgestellte generische ElGamal-Signatur entspricht damit in dieser Notation dem Typ EG-I Nr.1. Für die Signatur EG-II Nr.3 hätten die Signatur- und Verifikationsgleichungen folgende Form:

$$\begin{aligned} &\text{Signatur:} && s \equiv k - o_1(\hbar, \bar{r})a \bmod n \\ &\text{Verifikation:} && r = g^s y^{o_1(\hbar,\bar{r})} \end{aligned}$$

Es sind diese beiden Varianten mit der Multiplikation als Operation o_1, die in Kapitel 6 untersucht werden.

5.4 ElGamal Signaturen mit elliptischen Kurven

Um die soeben beschriebenen Verfahren auf die additive Gruppe einer elliptischen Kurve zu übertragen, müssen lediglich die Gruppenoperationen angepasst werden, also Multiplikation durch Addition und Potenzieren durch Multiplizieren mit Skalaren ersetzt werden. Zur zusätzlichen Verdeutlichung des Unterschiedes zwischen Punkten auf elliptischen Kurven und Zahlen in $\mathbb{Z}_n^*$ werden Letztere wie bisher durch kleine Buchstaben, Erstere aber mit großen Buchstaben dargestellt. Damit ergibt sich folgende Notation, wenn eine elliptische Kurve allgemein als $E(\mathbb{F}_q)$ mit $q = p$ für einen Primkörper mit $p \gg 3$ oder einen Polynomkörper $q = 2^m$ bezeichnet wird. Weiterhin sei n die Ordnung der von G erzeugten Untergruppe.

1. Schlüsselerzeugung: Mit dem privaten Schlüssel, einer zufälligen Zahl $a \in \mathbb{Z}_n^*$, und dem Erzeuger $G \in E(\mathbb{F}_q)$ ist der öffentliche Schlüssel $Y = aG$. Die Schlüsselerzeugung ist für sämtliche Varianten der ElGamal Signatur identisch.

2. Signaturerzeugung:

   - Typ EG-I Nr.1: Die Signatur (R, s) einer Nachricht m ist definiert durch $s = k^{-1}(\hbar - a\bar{r}) \bmod n$ für einen Punkt $R = kG$ mit $k \in_R \mathbb{Z}_n^*$, so dass $\bar{r} \neq 0$.

- Typ EG-II Nr.3: Die Signatur (R, s) der Nachricht m ist hier für den Punkt $R = kG$ mit $\bar{r} \neq 0$ durch $s = k - \hbar\bar{r}a \bmod n$ mit $k \in_R \mathbb{Z}_n^*$ gegeben.

3. Signaturverifikation:

   - Typ EG-I Nr.1: Um eine Signatur (R, s) für eine Nachricht m und den öffentlichen Schlüssel Y zu verifizieren, werden zwei Punkte auf der Kurve $E(\mathbb{F}_q)$ durch $V_1 = \bar{r}Y + sR$ und $V_2 = \hbar G$ berechnet. Die Signatur ist gültig, wenn $V_1 = V_2$ erfüllt ist.
   - Typ EG-II Nr.3: Um die Signatur (R, s) der Nachricht m mit dem öffentlichen Schlüssel Y zu verifizieren, wird der Punkt V auf $E(\mathbb{F}_q)$ mit $V = \hbar\bar{r}Y + sG$ berechnet. Die Signatur ist gültig, wenn $V = R$ erfüllt ist.

Für die Variante EG-I Nr.1 darf der Signaturwert s niemals die Bedingung $s = 0$ erfüllen, weil damit der private Schlüssel trivial zu berechnen wäre. Aus $s = 0 \equiv k^{-1}(h(m, R) - a\bar{r}) \bmod n$ folgt dann nämlich $a = h(m, R)\bar{r}^{-1} \bmod n$. Falls also ein solches s berechnet wird, sollte die Signaturerzeugung mit einem neuen k wiederholt werden. Ebenso darf der Verifizierer Signaturen mit $s = 0$ nicht akzeptieren.

In beiden Varianten muss $\bar{r} = 0$ ausgeschlossen werden, weil eine derart berechnete Signatur unabhängig vom geheimen Schlüssel wäre.

Beispiel 5.2 (Typ EG-I Nr.1) *Sei nun mit*

$$E(\mathbb{F}_{29}) : y^2 = x^3 + x + 6$$

die elliptische Kurve in kurzer Weierstraß-Normalform mit den Parametern $a = 1$, $b = 6$ *mit* $\#E(\mathbb{F}_{29}) = 38$ *Punkten (die Ordnung der Kurve) über das endliche Feld* $\mathbb{F}_{29}$ *der Ordnung* 28 *aus Beispiel 4.1 gegeben. Sei weiterhin* $G = (2, 4)$ *ein Punkt und Erzeuger der Kurve. Der private Schlüssel von Ally sei nun* $a = 5$, *womit sie*

$$Y = aG = 5(2, 4) = (4, 25)$$

berechnet. Allys öffentlicher Schlüssel ist also $(G, Y) = ((2, 4), (4, 25))$. *Sie will nun die Nachricht* m *signieren, deren Hashwert* $\hbar = h_1(m, R) = 19$ *sei. Mit einem zufällig gewählten* $k = 23$ *und dessen Inversem* $k^{-1} = 29 \bmod 38$ *bestimmt sie den Punkt*

$$R = kG = 23(2, 4) = (14, 26)$$

, für den $\bar{r} = 17$ *gelte. Es ergibt sich* $s = 29(19 - 5 \cdot 17) \bmod 38 = 12$ *und damit die Signatur* $(R, s) = ((14, 26), 12)$.

Mit den Hashwerten $\hbar = 19$ *und* $\bar{r} = 17$ *erhält Billy die beiden Punkte*

$$V_1 = \bar{r}Y + sR = 19(4, 25) + 12(14, 26) = (16, 0)$$

und

$$V_2 = \hbar G = 19(2, 4) = (16, 0)$$

. Weil $V_1 = V_2$ *ist, akzeptiert er die Signatur.*

5.5 Sicherheit und Effizienz

Die Effizienz der unterschiedlichen Varianten des ElGamal-Signaturverfahrens ist eng mit der verwendeten Gruppe verknüpft. Werden beispielsweise elliptische Kurven verwendet, so ist die Ordnung der Gruppe $\mathbb{Z}_n$ eher klein und Invertierungen von Elementen und andere Operationen fallen daher nicht sonderlich ins Gewicht, während die skalare Multiplikation der Punkte auf der Kurve den höchsten Aufwand erfordert.

Wird aber ein Körper $\mathbb{Z}_p$ verwendet, sind es gerade diese Invertierungen, die eine Variante „teurer“ machen als eine andere. Für diese Gruppen ist daher die Variante EG-II Nr.3 als besonders effizient zu betrachten, was zwar ebenso für elliptische Kurven zutrifft, aber dort kann die Verifikation beispielsweise für die Variante EG-I Nr.1 dadurch beschleunigt werden, dass beide Seiten der Verifikationsgleichung mit s^{-1} multipliziert werden. Daher müssen für $s^{-1}h_2(r)Y + R = s^{-1}h_1(m, R)G$ lediglich zwei Punktmultiplikationen durchgeführt werden. Der resultierende Vorteil ist allerdings bei den in der Praxis verwendeten optimierten Implementierungen der Multiplikation eher gering.

In allen Fällen kann die Signaturerzeugung beschleunigt werden, indem das Element $R = kG$ vorberechnet wird, weswegen aber R und k, bzw. k^{-1}, auch garantiert sicher gespeichert werden müssen.

Ebenso interessant ist eine vergleichende Analyse der Rechenzeiten, die in der Praxis erforderlich sind, um Signaturen zu erzeugen oder zu verifizieren. Für einen solchen, beispielhaften Vergleich von ECDSA mit RSA sei dazu auf [Sil00] verwiesen.

Die Signaturgröße hängt von der verwendeten Gruppe und ihrer Ordnung ab. Für ElGamal Signaturverfahren die eine elliptische Kurve $E(\mathbb{F}_q)$ verwenden, besteht eine Signatur (R, s) somit aus $|n| + |q|$ Bits. Da der Aufwand zur Berechnung des diskrete Logarithmus in einem Primzahlkörper mit $|p| = 1024$ Bits und einer elliptischen Kurven mit einer Ordnung der Länge $|n| = 160$ vergleichbar ist, sind diese Signaturen also extrem kompakt.

Die Sicherheit des privaten Schlüssels basiert darauf, dass für eine Signatur zwei Unbekannte, a und k, in der Signaturgleichung verwendet werden, weswegen damit die Sicherheit des ElGamal-Signaturverfahrens auf dem diskreten Logarithmus $\log_G Y$ basiert.

Für eine universelle Fälschung muss bei der Variante EG-I Nr.1 der diskrete Logarithmus $\log_R(\hbar G - \bar{r}Y)$ berechnet werden. Für die Variante EG-II Nr.3 gilt stattdessen $\log_G(R - o_1(\hbar, \bar{r})Y)$.

Um eine existentielle Fälschung für die Variante EG-I Nr.1 durchzuführen, wählt ein Angreifer zufällige $u, w \in \mathbb{Z}_n^*$, mit denen er zuerst $R = uG - wY$ berechnet und dann $s = \bar{r}w^{-1}$, sowie $\hbar = us$ bestimmt. Mit denselben u, w setzt der Angreifer dagegen für die Variante EG-II Nr.3 zuerst $s = u$, und berechnet daraufhin $R = sG - wY$ und $\hbar = w\bar{r}^{-1}$. Mit diesem Ansatz ist also die existentielle Fälschung für das ElGamal-Signaturverfahren möglich, wenn es einem Angreifer gelingt, ein m zu finden, für welches $\hbar = h(m, R)$ gilt.

6 ElGamal-Signaturen und Random Oracles

In diesem Kapitel werden die soeben vorgestellten Signaturverfahren im Random-Oracle-Modell auf ihre Sicherheit gegenüber No-Message- und Adaptively-Chosen-Message-Angriffe untersucht. Zuerst wird jedoch der Rahmen vorgestellt, in dem diese Untersuchung stattfindet.

6.1 Oracle Replay Attack

Wie bereits angesprochen, sind Beweise im Random-Oracle-Modell Reduktionen, welche die Äquivalenz zwischen dem Problem Signaturen zu fälschen und der Lösung des dem Verfahren zugrundeliegenden mathematischen Problems zeigen. Dazu werden Hashfunktionen durch Random Oracles ersetzt. In [PoS00] wird nun eine Reduktion für generische Signaturverfahren vorgestellt, welche die Grundlage für die Untersuchung der EC-ElGamal Signatur bilden soll und daher an dieser Stelle beschrieben wird.

Ein generische Signaturverfahren ist ein Tripel $(\mathcal{G}, \Sigma, V)$, bestehend aus einem probabilistischen Schlüsselerzeugungs-Algorithmus $\mathcal{G}$ mit Sicherheitsparameter t, der ein Schlüsselpaar K_s, K_p liefert, einem möglicherweise probabilistischem Signieralgorithmus Σ, der für eine Nachricht m mit dem Schlüsselpaar K_s, K_p eine Signatur σ erzeugt, und einem Verifizier-Algorithmus V, welcher überprüft, ob eine Signatur σ der Nachricht m für den öffentlichen Schlüssel K_p gültig ist. Der Signieralgorithmus muss die folgenden Bedingungen erfüllen. Er muss Tripel $\sigma = (\sigma_1, h, \sigma_2)$ produzieren, wobei σ_1 ein zufälliges Element einer großen Menge ist, h der Hashwert[6] von (m, σ_1) und der Signaturwert σ_2 ausschließlich von m, σ_1 und h abhängt. Die Bedingungen, nach denen die Signatur mit Σ erzeugt werden muss, werden von allen Signaturverfahren erfüllt, die äquivalent zu einem Drei-Runden-Zero-Knowledge-Beweis sind. Dabei entspricht der Witness des Beweisers dem Element σ_1 und die zufällig gewählte Challenge des Verifizierers dem Hashwert von (m, σ_1). Der erwähnte Sicherheitsparameter t, definiert sowohl die Länge in Bit der Ausgabe des Random Oracle als auch die Länge der Schlüssel. Für jeden Teil der Signatur (σ_1, h, σ_2) und damit für die Ausgabe des Random Oracle existieren dann 2^t Möglichkeiten. Der erwartete Aufwand, den ein Angreifer zum Brechen des Verfahrens benötigt, ist also bei optimaler Sicherheit exponentiell in t.

Mit diesen Voraussetzungen werden nun effiziente No-Message- und Adaptively-Chosen-Message-Angreifer, repräsentiert durch probabilistische Turingmaschinen, angenommen, die in polynomieller Zeit T mit einem Random Oracle eine gültige Signatur (σ_1, h, σ_2) produzieren können. In [PoS00] wird gezeigt, dass sie ebenfalls in polynomieller Zeit mit dem selben Zufallsband zwei gültige Signaturen der selben Nachricht für zwei unterschiedliche Random Oracles erzeugen können, indem polynomiell viele Wiederholungen (Replays) dieser Attacke mit anderen Random Oracles durchgeführt werden. Dann existieren zwei gültige Signaturen (σ_1, h, σ_2) und $(\sigma_1', h', \sigma_2')$, so dass $\sigma_1 = \sigma_1'$ gilt, aber $h \neq h'$ und $\sigma_2 \neq \sigma_2'$ erfüllt ist. Mit diesen zwei Signaturen soll dann der private

[6] Die explizite Angabe des Hashwerts ist für die in dieser Arbeit betrachteten Verfahren weder nötig noch sinnvoll, da er bei der Verifikation erneut erzeugt werden muss. Sie werden trotzdem beibehalten, um der Notation in [PoS00] zu entsprechen und die Antworten verschiedener Random Oracles bezeichnen und verwenden zu können.

Schlüssel K_s zu berechnen und somit das dem Signaturverfahren zugrunde liegende mathematische Problem ebenfalls lösbar sein.

Für die Adaptively-Chosen-Message-Attacke wird der reale Signierer durch einen Simulator, der nicht im Besitz des privaten Schlüssels ist, ersetzt. Es muss also bewiesen werden, dass der Signierer im Random-Oracle-Modell tatsächlich simuliert werden kann, weswegen insbesondere die Signaturverfahren interessant sind, die zu einem Zero-Knowledge-Beweis äquivalent sind, denn dann kann der Verifizierer keine Information über das Geheimnis des Beweisers (den privaten Schlüssel) erfahren. Dazu werden Verteilungen über das Auftreten der Signaturen für den realen Signierer und den Simulator verglichen. Sind diese ununterscheidbar, so kann der Simulator nicht von dem realen Signierer anhand der erzeugten Signaturen unterschieden werden.

Die Beweise zu den folgenden Theoremen sind in [PoS00] nachzulesen und werden hier nicht wiedergegeben.

Theorem 6.1 (Entspricht Theorem 10 (Forking Lemma) in [PoS00])
Sei $(\mathcal{G}, \Sigma, V)$ ein generisches Signaturverfahren mit Sicherheitsparameter t. Sei $\mathcal{A}$ eine probabilistische Turingmaschine, deren Eingabe lediglich aus öffentlich zugänglichen Daten besteht und welche $Q > 0$ Fragen an das Random Oracle richten darf. Angenommen $\mathcal{A}$ produziert in der Zeit T mit einer Wahrscheinlichkeit $\varepsilon \geq 7Q/2^t$ eine Nachricht m und eine dafür gültige Signatur (σ_1, h, σ_2), dann existiert eine andere Maschine, die $\mathcal{A}$ verwendet und zwei Signaturen (σ_1, h, σ_2) und $(\sigma_1, h', \sigma_2')$ der selben Nachricht in der erwarteten Zeit $84480QT/\varepsilon$ produziert, so dass $h \neq h'$ erfüllt ist.

Theorem 6.2 (Entspricht Theorem 13 (Forking Lemma) in [PoS00])
Sei $\mathcal{A}$ eine probabilistische Turingmaschine, deren Eingabe lediglich aus öffentlich zugänglichen Daten besteht und welche Q Fragen an das Random Oracle und R Fragen an den Signierer richten darf. Angenommen $\mathcal{A}$ produziert in der Zeit T mit einer Wahrscheinlichkeit $\varepsilon \geq 10(R+1)(R+Q)/2^t$ eine Nachricht m und eine dafür gültige Signatur (σ_1, h, σ_2). Wenn die Tripel (σ_1, h, σ_2) ohne Kenntnis des privaten Schlüssels des Signierers mit einer nicht unterscheidbaren Verteilung simuliert werden können, dann existiert eine andere Maschine, die $\mathcal{A}$ verwendet, und die Interaktionen mit dem Signierer durch die Simulation ersetzt und damit zwei gültige Signaturen (σ_1, h, σ_2) und $(\sigma_1, h', \sigma_2')$ der Nachricht m in einer erwarteten Zeit $120686QT/\varepsilon$ produziert, so dass $h \neq h'$ erfüllt ist.

Die in 5.4 beschriebenen ElGamal-Signaturverfahren verwenden zwei Hashfunktionen. Von diesen wird in den folgenden Analysen h_1 als Random Oracle modelliert, und h_2 als kollisionsresistent angenommen. Zusätzilch darf h_2 nicht den Wert 0 als Ergebnis liefern. Damit werden die Signatur- und Verifikationsgleichung beispielsweise des Typ EG-I Nr.1, mit $\bar{r} = h_2(R)$ und $h_1 = h$ zu:

$$\begin{array}{ll} \text{Signaturerzeugung :} & s = k^{-1}(h(m,R) - a\bar{r}) \bmod n \\ \text{Signaturverifikation :} & h(m,R)G = \bar{r}Y + sR \end{array} \tag{6.1}$$

Die Eigenschaften von Random Oracles wurden mehrfach erwähnt und werden an dieser Stelle formuliert. Ein Random Oracle ist eine Zufallsfunktion

$h : \{0,1\}^* \to \{0,1\}^t$ für $t = \lceil \log n \rceil$. Für die Wahrscheinlichkeit, dass h ein bestimmtes Ergebnis liefert, gilt $\Pr[h(x) = y] = 2^{-t}$. Damit ist die Wahrscheinlichkeit, dass $h(x) = h(x')$ ist, wenn $x \neq x'$ gilt, $\Pr[h(x) = h(x')|x \neq x'] = 2^{-t}$. Selbstverständlich gilt aber $\Pr[h(x) = h(x')|x = x'] = 1$.

In den nächsten beiden Abschnitten werden zwei Varianten des ElGamal Signaturverfahren, Typ EG-I Nr.1 und EG-II Nr.3, im Random-Oracle-Modell untersucht. Signaturerzeugung und -verifikation beider Varianten haben (mit $\bar{r} = h_2(R)$) dann folgende Form:

$$\begin{array}{ll} \text{EG} - \text{I Nr.1} & \\ \text{Signaturerzeugung :} & s = k^{-1}(h(m,R) - a\bar{r}) \bmod n, \quad s \neq 0 \\ \text{Signaturverifikation :} & h(m,R)G = \bar{r}Y + sR \end{array} \tag{6.2}$$

$$\begin{array}{ll} \text{EG} - \text{II Nr.3} & \\ \text{Signaturerzeugung :} & s = k - h(m,R)\bar{r}a \bmod n \\ \text{Signaturverifikation :} & R = sG + h(m,R)\bar{r}Y \end{array} \tag{6.3}$$

6.2 ElGamal-Signatur Typ EG-I Nr.1

6.2.1 No-Message-Attack

Theorem 6.3 *Wenn ein No-Message Angreifer $\mathcal{A}$ in der durch T begrenzten Zeit erfolgreich mit einer Wahrscheinlichkeit $\varepsilon \geq 7Q/n$ die existentielle Fälschung für die generische ElGamal-Signatur EG-I Nr.1 durchführen kann und dabei maximal Q Anfragen an das Random Oracle richtet, dann kann der diskrete Logarithmus in der verwendeten Gruppe $\mathbb{G}$ mit der primen Ordnung n in der erwarteten Zeit $84480QT/\varepsilon$ gelöst werden.*

Beweis. Mit dem Forking-Lemma (Theorem 6.1) erhält man zwei Signaturen $(R, h(m,R), s)$ und $(R, h'(m,R), s')$ mit $s \neq s'$, $h(m,R) \neq h'(m,R)$. Aus (6.2) ergibt sich daher $h(m,R)G = \bar{r}Y + sR$ und $h'(m,R)G = \bar{r}Y + s'R$ und daraus folgt für $s, s' \in \mathbb{Z}_n^*$

$$\begin{aligned} R &= h(m,R)(s^{-1} \bmod n)G - \bar{r}(s^{-1} \bmod n)Y \\ R &= h'(m,R)(s'^{-1} \bmod n)G - \bar{r}(s'^{-1} \bmod n)Y. \end{aligned}$$

Das führt wiederum zu

$$\bar{r}(s'^{-1} - s^{-1})Y = (h'(m,R)s'^{-1} - h(m,R)s^{-1})G$$

und damit zu

$$Y = \left(h'(m,R)s'^{-1} - h(m,R)s^{-1}\right)\bar{r}^{-1}\left(s'^{-1} - s^{-1}\right)^{-1} G.$$

Der diskrete Logarithmus von y zur Basis g in der Gruppe $\mathbb{G}$ ist daher:

$$\log_G Y \equiv \frac{h'(m,R)s'^{-1} - h(m,R)s^{-1}}{\bar{r}\left(s'^{-1} - s^{-1}\right)} \bmod n.$$

Vereinfacht wird daraus schließlich

$$\log_G Y \equiv \frac{h'(m,R)s - h(m,R)s'}{\bar{r}(s-s')} \bmod n. \tag{6.4}$$

Damit dieses Resultat definiert ist, muss das Element $(s-s')^{-1}$ in $\mathbb{Z}_n$ existieren. Da n prim ist und bis auf die 0 alle Elemente in $\mathbb{Z}_n$ invertierbar sind, muss also $s \neq s'$ erfüllt sein. Das gilt für ein feststehendes R und die ebenfalls feststehende Nachricht m wegen $h(m,R) \neq h'(m,R)$ immer. □

Sei nun $n = \prod_{p|n} p^{e(p)}$ zusammengesetzt, so gibt es für jeden Primteiler p von n exakt $\frac{n-p}{p}$ nicht invertierbare Elemente in Z_n, die Vielfachen von p, die in der Restklasse $\{p + \mathbb{Z}/n\mathbb{Z}\}$ zusammengefasst sind. Die Menge der invertierbaren Elemente bildet die Einheitengruppe $\mathbb{Z}_n^*$, deren Ordnung durch die Eulersche φ-Funktion mit $\varphi(n) = n\prod_{p|n}\frac{p-1}{p}$ gegeben ist. Daher muss für Gleichung (6.4) die Bedingung $\bar{r}(s-s') \in \mathbb{Z}_n^*$ erfüllt sein, um den diskreten Logarithmus $\log_G Y$ in $\mathbb{Z}_n$ berechnen zu können. Da in der Praxis verwendeten elliptischen Kurven prime (Unter-)Gruppenordnung besitzen, werden diese Überlegungen nicht weiterverfolgt.

6.2.2 Adaptively-Chosen-Message-Attack

Theorem 6.4 *Sei $\mathcal{A}$ nun ein Angreifer, der mit einer Wahrscheinlichkeit ε erfolgreich in durch T begrenzter Zeit die existentielle Fälschung für die generische ElGamal-Signatur EG-I Nr.1 durchführt. Er darf dabei R beliebige Anfragen, die er in Abhängigkeit von bereits erhaltenen Antworten wählen darf, an das Signier-Oracle und Q Anfragen an das Random Oracle stellen. Sei die Erfolgswahrscheinlichkeit durch $\varepsilon \geq 10(R+1)(R+Q)/n$ gegeben, dann kann das Problem des diskreten Logarithmus in der verwendeten Gruppe $\mathbb{G}$ der primen Ordnung n in der erwarteten Zeit $120686QT/\varepsilon$ gelöst werden.*

Der folgende Beweis verwendet Theorem 6.2 und basiert darauf, dass simulierte Oracle-Antworten nicht wirklich in einem Zusammenhang mit einer tatsächlichen Nachricht stehen müssen. Der Angreifer akzeptiert die berechneten Antworten als korrekt für eine Nachricht, weswegen diese Nachricht auch nicht als Anfrage an das Random Oracle gestellt werden darf. Die Wahrscheinlichkeit für das Auftreten dieser Kollisionen ist jedoch sehr gering, wie in [PoS00] gezeigt wird. Das bedeutet, dass durch den Simulator die Adaptively-Chosen-Message-Attack auf eine No-Message-Attack reduziert wird.

Beweis. Es muss also lediglich gezeigt werden, dass ein realer Signierer simuliert werden kann, ohne dass dessen privater Schlüssel benötigt wird. Dazu werden zwei Verteilungen definiert, welche Aussagen darüber erlauben, mit welcher Wahrscheinlichkeit eine bestimmte Signatur für einen realen Signierer und für die Simulation auftritt. Für den realen Signierer hängt eine Signatur von seinem privaten Schlüssel a, einem zufällig gewählten invertierbaren $k \in \mathbb{Z}_n$, welches das Gruppenelement $R = kG$ erzeugt, und der Antwort $h(m,R) \in \mathbb{Z}_n$ des Random Oracles ab. Damit gilt für die Signatur $(R, h(m,R), s)$ die Verteilung

$$\delta = \left\{ (R, h(m,R), s) \;\middle|\; \begin{array}{c} k \in_R \mathbb{Z}_n^* \\ h(m,R) \in_R \mathbb{Z}_n \\ R = kG \\ s = k^{-1}(h(m,R) - a\bar{r}) \bmod n \\ s \neq 0 \end{array} \right\}. \tag{6.5}$$

Um den Signierer ohne den privaten Schlüssel simulieren zu können, wird die Verifikationsbedingung (6.2) verwendet. Der Simulator $\mathcal{S}$ wählt dazu zwei zufällige Elemente $k \in \mathbb{Z}_n$ und $l \in \mathbb{Z}_n^*$ und berechnet $R = kG - lY$. Falls $R = E$, dem neutralen Element in $\mathbb{G}$, müssen zwei neue k, l gewählt werden. Mit diesem R und $\bar{r} = h_2(R)$, sowie $s = \bar{r}l^{-1} \bmod n$ berechnet er $h(m,R) = ks \bmod n$ und liefert als Ergebnis die Signatur $(R, h(m,R), s)$, für die dann folgende Verteilung gilt:

$$\delta' = \left\{ \big(R, h(m,R), s\big) \;\middle|\; \begin{array}{c} k \in_R \mathbb{Z}_n \\ l \in_R \mathbb{Z}_n^* \\ R = kG - lY \\ s = \bar{r}l^{-1} \bmod n \\ h(m,R) = ks \bmod n \\ R \neq E \end{array} \right\}. \tag{6.6}$$

Nach beiden Verteilungen werden nur Tripel $(R, h(m,R), s) = (\Gamma, \lambda, \beta)$ mit $\Gamma \in \mathbb{G}\backslash\{E\}$ und $\lambda, \beta \in \mathbb{Z}_n$ erzeugt, welche $\lambda G = h_2(\Gamma)Y + \beta\Gamma$ erfüllen. Die Wahrscheinlichkeit für das Auftreten eines dieser Tripel wird im Folgenden untersucht. Für das Auftreten einer der Kombinationen, die diese Bedingung nicht erfüllen, beträgt die Wahrscheinlichkeit also 0.

Unter der Verteilung (6.5) hängt der Wert R von dem zufällig gewählten k ab, also ist $\Pr_\delta[R = kG = \Gamma] = (n-1)^{-1}$. Der Signaturwert s hängt nun sowohl von R als auch von der Orakelantwort ab. Für ein feststehendes $R = \tilde{k}G$ ($\tilde{k} \in \mathbb{Z}_n^*$) ist damit $\Pr_\delta[s = k^{-1}(h(m,R) - a\bar{r}) = \beta \mid k = \tilde{k}, s \neq 0] = \frac{1}{n-1}$. Mit

$$\Pr_\delta[(R, h(m,R), s) = (\Gamma, \lambda, \beta)] = \Pr_\delta[R = \Gamma] \cdot \Pr_\delta[s = \beta \mid k = \tilde{k}, s \neq 0]$$

folgt für die Wahrscheinlichkeit, dass eine bestimmte Signatur auftritt:

$$\Pr_\delta\big[(R, h(m,R), s) = (\Gamma, \lambda, \beta)\big] = \Pr_\delta\left[\begin{array}{c} kG = \Gamma \\ h(m,R) = \lambda \\ k^{-1}(\lambda - ah_2(\Gamma)) = \beta \end{array}\right] = \frac{1}{(n-1)^2}.$$

Um nun die Verteilung (6.6) damit vergleichen zu können, muss nun ebenfalls die Wahrscheinlichkeit, für welche $(R, h(m,R), s) = (\Gamma, \lambda, \beta)$ ist, berechnet werden. Nun hängt R von k und l ab und die Orakelantwort $h(m,R)$ wird aus k und s berechnet, wobei aber s auf $\bar{r}$ und l basiert. Somit können diese Wahrscheinlichkeiten zwar nicht direkt angegeben werden, allerdings können sie als bedingte Wahrscheinlichkeiten in Abhängigkeit von jeweils einer der Zufallszahlen berechnet werden. Weil die Wahrscheinlichkeiten dann unabhängig sind, ist für $\tilde{k} \in \mathbb{Z}_n$ und $\tilde{l} \in \mathbb{Z}_n^*$

$$\Pr_{\delta'}\big[(R, h(m,R), s) = (\Gamma, \lambda, \beta) \mid R \neq E\big] = \Pr_{\delta'}[R = \Gamma \mid l = \tilde{l}, R \neq E] \cdot \Pr_{\delta'}[s = \beta \mid k = \tilde{k}].$$

Unter der Bedingung $k = \tilde{k}$ hängt s jetzt nur noch von l ab und somit ist $\Pr_{\delta'}[s = \beta \mid k = \tilde{k}] = (n-1)^{-1}$ und mit $\Pr_{\delta'}[R = \Gamma \mid l = \tilde{l}, R \neq E] = (n-1)^{-1}$ folgt

$$\Pr_{\delta'}\left[(R, h(m,R), s) = (\Gamma, \lambda, \beta)\right] = \Pr_{\delta'}\left[\begin{array}{c} kG - lY = \Gamma \\ h_2(\Gamma)l^{-1} = \beta \\ k\beta = \lambda \end{array} \middle| \; R \neq E \right] = \frac{1}{(n-1)^2}.$$

Damit ist nun $\delta = \delta'$, die Verteilungen sind perfekt ununterscheidbar, und das Ergebnis folgt direkt aus Theorem 6.2 und dem Beweis zu Theorem 6.3. □

6.3 ElGamal-Signatur Typ EG-II Nr. 3

6.3.1 No-Message-Attack

Theorem 6.5 *Wenn ein No-Message-Angreifer $\mathcal{A}$ in durch T begrenzter Zeit erfolgreich mit einer Wahrscheinlichkeit $\varepsilon \geq 7Q/n$ die existentielle Fälschung für die generische ElGamal-Signatur EG-II Nr.3 durchführen kann und dabei maximal Q Anfragen an das Random Oracle richtet, dann kann der diskrete Logarithmus in der verwendeten Gruppe $\mathbb{G}$ mit der primen Ordnung n in der erwarteten Zeit $84480QT/\varepsilon$ gelöst werden.*

Beweis. Wie im letzten Abschnitt erhält man zwei Signaturen $(R, h(m,R), s)$ und $(R, h'(m,R), s')$ mit $s \neq s'$, $h(m,R) \neq h'(m,R)$. Aus (6.3) folgt dann $R = sG - h(m,R)\bar{r}Y$ und für die zweite Signatur $R = s'G - h'(m,R)\bar{r}Y$. Mit diesen ergibt sich

$$sG - h(m,R)\bar{r}Y = s'G - h'(m,R)\bar{r}Y,$$

und damit

$$Y = \bar{r}^{-1}(h(m,R) - h'(m,R))^{-1}(s - s')G.$$

Der diskrete Logarithmus ist also

$$\log_G Y \equiv \frac{s - s'}{\bar{r}\,(h(m,R) - h'(m,R))} \bmod n. \tag{6.7}$$

Dieser ist nicht definiert, wenn das Element $h'(m,R) - h(m,R) = 0$ ist, was nie eintritt, weil $h(m,R) \neq h'(m,R)$ ist. □

6.3.2 Adaptively-Chosen-Message-Attack

Theorem 6.6 *Sei $\mathcal{A}$ nun ein Angreifer, der mit Wahrscheinlichkeit ε erfolgreich in durch T begrenzter Zeit die existentielle Fälschung für die generische ElGamal-Signatur EG-II Nr.3 durchführt. Er darf dabei R beliebige Anfragen, die er in Abhängigkeit von bereits erhaltenen Antworten wählen darf, an das Signier-Oracle und Q Anfragen an das Random Oracle stellen. Sei die Erfolgswahrscheinlichkeit durch $\varepsilon \geq 10(R+1)(R+Q)/n$ gegeben, dann kann das Problem des diskreten Logarithmus in der verwendeten Gruppe $\mathbb{G}$ der primen Ordnung n in der erwarteten Zeit $120686QT/\varepsilon$ gelöst werden.*

Beweis. Wie zuvor werden zwei Verteilungen für den Signierer und den Simulator definiert. Für den realen Signierer hängt eine Signatur auch in dieser Variante von a, dem zufällig gewählten, invertierbaren $k \in \mathbb{Z}_n$, und der Antwort $h(m,R) \in \mathbb{Z}_n$ des Random Oracles ab. Damit gilt für die Signatur $(R, h(m,R), s)$ die Verteilung

$$\delta = \left\{ (R, h(m,R), s) \;\middle|\; \begin{array}{c} k \in_R \mathbb{Z}_n^* \\ h(m,R) \in_R \mathbb{Z}_n \\ R = kG \\ s = k - a\bar{r}h(m,R) \bmod n \end{array} \right\}. \tag{6.8}$$

Der Simulator arbeitet ähnlich dem im letzten Abschnitt beschriebenen. $\mathcal{S}$ wählt nun zwei zufällige Elemente $k, l \in \mathbb{Z}_n$ und setzt $s = k$. Dann berechnet er $R = sG + lY$. Falls $R = E$, müssen hier ebenfalls zwei neue k, l gewählt werden. Mit $\bar{r} = h_2(R)$ erhält $\mathcal{S}$ dann $h(m,R) = l\bar{r}^{-1} \bmod n$ und liefert die Signatur $(R, h(m,R), s)$ als Ergebnis. Für die Signatur des Simulators gilt damit die folgende Verteilung:

$$\delta' = \left\{ (R, h(m,R), s) \;\middle|\; \begin{array}{c} k \in_R \mathbb{Z}_n \\ l \in_R \mathbb{Z}_n \\ s = k \\ R = sG + lY \\ h(m,R) = l\bar{r}^{-1} \bmod n \\ R \neq E \end{array} \right\}. \tag{6.9}$$

Nach beiden Verteilungen werden nur diejenigen Tripel (Γ, λ, β) erzeugt, welche die Bedingung $\Gamma = \beta G + h_2(\Gamma)\lambda Y$ erfüllen, also beträgt die Wahrscheinlichkeit für das Auftreten einer der Kombinationen, die diese Bedingung nicht erfüllen 0.

Wie zuvor, gilt für den realen Signierer $\Pr_\delta[kG = \Gamma] = (n-1)^{-1}$ und mit $\tilde{k} \in \mathbb{Z}_n^*$ ist $\Pr_\delta[s = k - a\bar{r}h(m,R) = \beta \mid k = \tilde{k}] = n^{-1}$. Somit folgt für die Wahrscheinlichkeit, dass $(R, h(m,R), s) = (\Gamma, \lambda, \beta)$ ist, unter Verteilung 6.8:

$$\Pr_\delta\left[(R, h(m,R), s) = (\Gamma, \lambda, \beta)\right] = \Pr_\delta\left[\begin{array}{c} kG = \Gamma \\ h(m,R) = \lambda \\ s = k - a\bar{r}h(m,R) = \beta \end{array}\right] = \frac{1}{n(n-1)}$$

Unter Verteilung (6.9) muss die Wahrscheinlichkeit, dass $(R, s) = (\gamma, \beta)$ ist, wie im letzten Abschnitt über k und l berechnet werden. In diesem Fall ist $s = k$ während R von k und l abhängt und $h(m,R)$ aus l und $\bar{r}$ berechnet wird. Somit muss die Wahrscheinlichkeit, dass ein bestimmtes R auftritt, in Abhängigkeit der Signatur s, und damit der Zufallszahl k, betrachtet werden. Also ist $\Pr_{\delta'}\left[(R, h(m,R), s) = (\Gamma, \lambda, \beta)\right] = \Pr_{\delta'}[R = \Gamma \mid k = \tilde{k}] \cdot \Pr_{\delta'}[s = \beta]$ für $\tilde{k} \in \mathbb{Z}_n$. Mit$\Pr_{\delta'}[s = \beta] = n^{-1}$ und $\Pr_{\delta'}[R = \Gamma \mid k = \tilde{k}, R \neq E] = (n-1)^{-1}$ folgt

$$\Pr_{\delta'}\left[(R, h(m,R), s) = (\Gamma, \lambda, \beta)\right] = \Pr_{\delta'}\left[\begin{array}{c} kG + lY = \Gamma \\ k = \beta \\ h_2(\Gamma)l^{-1} = \lambda \end{array} \;\middle|\; R \neq E\right] = \frac{1}{n(n-1)}.$$

Damit sind δ und δ' auch für diese Variante perfekt ununterscheidbar. □

6.4 Bemerkungen

Im folgenden wird das Vorgehen zur Untersuchung weiterer Varianten des ElGamal-Signaturverfahrens vorgestellt, auf die Implementierung der Random Oracles mittels Hashfunktionen eingegangen und abschliessend der standardisierte ECDSA betrachtet.

6.4.1 Meta-ElGamal-Signatur

Die übrigen Typen der ElGamal-Signatur und deren Permutationen können mit demselben Schema (analog zu Abschnitt 6.2 und 6.3) untersucht werden.

1. No-Message-Attack: Zuerst müssen zwei Varianten der Verifikationsgleichung $\pm aG \equiv \pm bR \pm cY$ mit $h(m,R) \neq h'(m,R)$ und $s \neq s'$ aufgestellt und diese danach gleichgesetzt und nach Y aufgelöst werden.

2. Adaptively-Chosen-Message-Attack: Aus der in Kapitel 5.5 vorgestellten Two-Parameter-Forgery kann ein Simulator für jede Variante der Signaturgleichung und jede Permutation konstruiert werden, indem zuerst ein $R = kG - lY$ für zwei zufällig gewählte k und l berechnet wird, und dann die Gleichungen $kC \equiv A \bmod n$ und $kC \equiv B \bmod n$ nach s und $h(m,R)$ gelöst werden.

Es gibt zwei weitere Varianten der Signatur- und Verifikationsgleichung, die bisher nicht erwähnt wurden und in [HPN94] als unsicher (dort für Hashwerte der Form $h(m)$, statt hier $h(m,R)$) ausgeklammert wurden. Durch die modifizierte Verwendung der Hashfunktion ist eine der beiden Varianten nun sicher. Beide Aussagen, die Sicherheit der einen Variante und die Unsicherheit der anderen Variante, lassen sich im Random-Oracle-Modell nachvollziehen.

1. rs-Variante:

$$\text{Signatur}: \quad h(m,R) = a\bar{r}s + k \bmod n$$

$$\text{Verifikation}: \quad h(m,R)G = \bar{r}sY + R$$

2. ms-Variante:

$$\text{Signatur}: \quad h(m,R)s = a\bar{r} + k \bmod n$$

$$\text{Verifikation}: \quad h(m,R)sG = \bar{r}Y + R$$

Für beide Varianten lässt sich die Äquivalenz zum diskreten Logarithmus für einen No-Message-Angreifer zeigen:

rs-Variante:

$$\log_G Y = \bar{r}\frac{s - s'}{h(m,R) - h'(m,R)} \bmod n$$

ms-Variante:

$$\log_G Y = \frac{\bar{r}}{h(m,R)s - h'(m,R)s'} \bmod n$$

Wird nun ein Simulator für die Adaptively-Chosen-Message-Attacke konstruiert, so ergibt dies das folgende Vorgehen:

rs-Variante	ms-Variante
$v, w \in \mathbb{Z}_n$	$v, w \in \mathbb{Z}_n$
$R = vG + wY$	$R = vG + wY$
$h(m, R) = v$	$s = vh(m, R)^{-1} \bmod n$
$s = w\bar{r}^{-1} \bmod n$	$w = \bar{r} \stackrel{?}{=} h_2(R)$

Die Simulation für die rs-Variante ist durch die modifizierte Verwendung der Hashfunktion erfolgreich. Würde jedoch, wie in Abschnitt 5.3 beschrieben, lediglich $h(m)$ verwendet, so wäre die rs-Variante für einen No-Message-Angreifer universell fälschbar, indem dieser $v = h(m)$ für eine beliebige Nachricht m setzt, denn v ist in diesem Fall unabhängig von R. Dadurch, dass $v = h(m, R)$ gelten muss, ist eben diese Voraussetzung nicht erfüllt. (Jedoch ist das Forking-Lemma auf die ursprüngliche Definition mit $h(m)$ nicht anwendbar.)

Im Gegensatz dazu kann die ms-Variante, zumindest auf die beschriebene Weise, nicht simuliert werden, denn es ist äußerst unwahrscheinlich, dass das zufällig gewählte w tatsächlich die Bedingung $w = \bar{r}$ erfüllt. Ist aber eine gültige Signatur (R, s) der Nachricht m bekannt, kann daraus eine weitere gültige Signatur erzeugt werden, indem aus der Bedingung $h(m, R)s = v$ für eine Nachricht m' und deren Hashwert $h(m', R)$ die Signatur $s' = h(m', R)^{-1}v$ berechnet wird. Damit ist also in der ms-Variante die universelle Fälschung für einen Known-Message-Angreifer möglich.

6.4.2 Hashfunktionen und Random Oracles

In Abschnitt 3.2.2 wurde die Transformation von Zero-Knowledge-Identifikationsverfahren in Signaturverfahren besprochen. In [ABN02] wurde inzwischen gezeigt, dass Signaturverfahren, die zu einem ZK-Identifikationsverfahren äquivalent sind, nur dann im Random-Oracle-Modell beweisbar sind, wenn das Identifikationsverfahren beweisbar sicher gegenüber passiven Attacken ist. Die Ausgabe des Random Oracles für die zu signierende Nachricht entspricht dabei der zufällig gewählten Challenge des Verifizierers.

Es wurde bereits in Abschnit 2.4.2 erwähnt, dass Random Oracles in der Praxis durch Hashfunktionen implementiert werden, sich das aber mathematisch nicht rechtfertigen lässt. Im folgenden werden nun zwei Eigenschaften von Random Oracles und mögliche Implementationen betrachtet, sowie Gefahren aufgezeigt, die durch die Verwendung speziell konstruierter Hashfunktionen entstehen können.

Oracle Hashing

In [Can00] wird die Eigenschaft der Random Oracles betrachtet, zufällige und von der Eingabe unabhängige Ausgaben zu erzeugen und mehrere Konstruktionen vorgeschlagen, welche als *Oracle Hashing* bezeichnet werden. Mit diesen Konstruktionen lassen sich Random Oracles nachvollziehbar in einer Implementation eines idealen Modells ersetzen, falls die der Sicherheit zugrundeliegende Annahme darin besteht, dass eine Antwort des Random Oracles keinerlei Aussage über die gestellte Frage ermöglicht. Zwei dieser Konstruktionen basieren auf Hashfunktionen und benötigen ein Zufalls-String x, dessen Länge der Länge der Ausgabe der verwendeten Hashfunktion entspricht. Damit wird in der ersten

Variante $H(m) = x, h(x\|m)$ als Hashwert berechnet. In der zweiten Variante wird stattdessen $H(m, x) = x, h(x\|h(m))$ berechnet. Aufgrund des verwendeten Zufalls-Strings und weil eine Hashfunktion deterministisch ist, ist auch der erzeugte Hashwert ein Zufalls-String und damit liefert H für eine identische Anfrage stets ein neues Ergebnis. Eine Verifikation eines solchen Hashwertes ist dennoch einfach möglich, da der verwendete Zufalls-String ebenfalls ausgegeben wird.

Full Domain Hashing

Ein weiteres Problem ist die Länge der Ausgabe der Hashfunktion. Die momentan verwendeten Hashfunktionen erzeugen eine 160-Bit-Ausgabe. Als Näherung für Random Oracles ist das zwar ausreichend für elliptische Kurven mit einer Ordnung von etwa 2^{160}, jedoch sind auch Kurven spezifiziert, die eine Ordnung von bis zu 2^{571} besitzen. Random Oracles sind aber so definiert worden, dass sämtliche Elemente in $\mathbb{Z}_n$ (*Full Domain Hash*) erzeugt werden können und nicht nur alle Elemente in $\mathbb{Z}_{160}$. Es ist jedoch einfach möglich, eine Hashfunktion zu konstruieren, die entsprechend größere Hashwerte erzeugt. Um eine Möglichkeit an einem Beispiel zu demonstrieren, sei eine Nachricht m gegeben, deren Hashwert mit einer Länge $|H(m)| = 480$ Bit berechnet werden soll. Mit einer Hashfunktion werden drei Hashwerte $h_i = h(i\|m),\ i = 1, 2, 3$ berechnet und dann zu $H(m) = h_1\|h_2\|h_3$ zusammengesetzt. Die Länge der erzeugten Hashwerte ist jedoch stets ein Vielfaches von 160, und es muss eine Möglichkeit geben diese auf ein beliebige Länge zu reduzieren, beispielsweise durch die Wahl der ersten oder letzten $|n|$ Bits. Durch die mehrfache Verwendung der Nachricht m entstünden dabei keine zusätzlichen Kollisionen. Eine Kombination von Oracle Hashing und Full Domain Hashiing könnte dann durch $H(m, \underline{x}) = \underline{x}, h\big(x_1\|m\big) \parallel h\big(x_2\|m\big) \parallel \cdots \bmod n$ realisiert werden, wobei $\underline{x} = (x_1, x_2, \ldots)$ mit $x_i \in_R \{0,1\}^{160}$ für $i = 1, 2, \ldots$ ist.

Adversarial Hashing

Es besteht die Gefahr, dass Hashfunktionen mit speziellen Kollisionen konstruiert werden, welche in einer Analyse, die Funktionen mit zufälligen Argumenten testet, mit hoher Wahrscheinlichkeit nicht entdeckt werden können. Dieses Prinzip wird *adversarial Hashing* genannt und [MiR98] beschrieben. Dem Konstrukteur der Funktion ist es damit möglich Kollisionen für Nachrichten, die eine spezielle vorgegebene Struktur besitzen, auszunutzen und Signaturen unabhängig vom Schlüsselpaar zu erzeugen. Mit einem zufälligen zusätzlichen Argument werden diese Angriffe nicht gänzlich ausgeschaltet, weil lediglich auf Seiten des Signierers das zusätzliche Argument wirklich zufällig ist. Für einen Verifizierer ist es durch die Signatur vorgegeben. Die in [MiR98] beschriebene Attacke basiert darauf, Signaturen unabhängig von dem verwendeten Schlüsselpaar erzeugen zu können, im Falle der ElGamal Signatur EG-I Nr.1 besteht eine triviale Möglichkeit darin, je einen zufällig gewählten Punkte verschiedener populärer elliptische Kurven derart zu fixieren, dass die Hashfunktionen für diese den Wert 0 liefert. Damit wäre die Signatur unabhängig vom öffentlichen Schlüssel Y, und ein Angreifer könnte sogar beliebige Nachrichten signieren. Die Autoren schlagen vor, in sämtlichen Anwendungen einer Hashfunktion den öffentlichen Schlüssel als zusätzliches Argument zu verwenden, denn dieser ist

zufällig gewählt, nachdem die Hashfunktion konstruiert wurde, und es ist somit extrem unwahrscheinlich, eine der fixierten Kollisionen während einer Signaturerzeugung zu verwenden.

Bewertung

Um nun die Art der Verwendung einer Hashfunktion zu fixieren, müssen mehrere Aspekte berücksichtigt weren. Neben den angesprochenen Problemen sind auch die Konsequenzen der vorgeschlagenen Lösungen hinsichtlich dem Aufwand der Signaturen und die Verwendbarkeit des Verfahrens mit zukünftigen Hashfunktionen entscheidend.

Adversarial Hashing kann vermieden werden, indem nicht vorgegebene elliptische Kurven, sondern zufällig neu erzeugte Kurven verwendet werden. Diese sind dem Kronstrukteur der Hashfunktion nicht bekannt.

Durch die Implementierung des Oracle Hashing wird die erzeugte Signatur entsprechend größer, da der Zufallsstring mit der Signatur übertragen werden muss. Es ist aber möglich, dass in Zukunft entweder Hashfunktionen entworfen werden, die den Eigenschaften des Random Oracle eher entsprechen, oder aber Beweisverfahren entwickelt werden, welche die Annahme eines Random Oracles überflüssig machen. In beiden Fällen wäre der zusätzliche Aufwand obsolet.

Gleiches gilt für die Länge des Hashwertes. Hasfunktionen mit einer Ausgabelänge von mehr als 160 Bit werden entwickelt ([SHA2]), so dass auch hier die Implementierung auf der Ebene des Signaturverfahrens eher hinderlich ist.

Es erscheint wichtig, die Hashfunktion als ein Modul zu verstehen, welches den aktuellen Anforderungen entsprechend aus den aktuell verfügbaren Funktionen gewählt wird. Um das Signaturverfahren möglichst flexibel und erweiterungsfähig zu halten, wird daher $\mathfrak{h} = h(m, R)$ als Instanziierung der Random Oracles vorgeschlagen.

6.4.3 Die Funktion h_2 und ECDSA

Die in Kapitel 5 vorgestellten ElGamal-Signaturverfahren wurden ursprünglich für Primzahlkörper $\mathbb{Z}_p$ vorgeschlagen und es wurden im Laufe der Zeit mehrere Varianten für eine Funktion h_2, wie sie in Gleichung (5.1) verwendet wird, vorgeschlagen. In der ursprünglichen Definition wurde für h_2 die Identitätsfunktion gewählt und die Signatur s in der Gruppe $\mathbb{Z}_{p-1}$, also $\bmod p-1$, berechnet. Wenn p eine Primzahl ist, ist $p-1$ zumindest durch den Faktor 2 dividierbar und es ergeben sich die in Abschnitt 6.3 beschriebenen Probleme im Random-Oracle-Modell. Später wurde vorgeschlagen die Signatur in einer primen Untergruppe $q : q|p-1$ zu berechnen[Sch91]. Für h_2 kann in diesem Fall ebenfalls die Identitätsfunktion verwendet werden, und die Verifikation findet weiterhin in $\mathbb{Z}_p$ statt. Jedoch kann zusätzlich die Verifikation in der Untergruppe $\mathbb{Z}_q$ vorgenommen werden, indem beide Seiten der Verifikationsgleichung modulo q reduziert werden [DSS91], so dass

$$r \bmod q = \left(g^{h(m)s^{-1}} y^{-(r \bmod q)s^{-1}} \bmod p\right) \bmod q$$

zu prüfen ist, weswegen also in diesem Fall $h_2(r) = r \bmod q$ ist. Schließlich wurde vorgeschlagen, h_2 durch eine Hashfunktion h zu realisieren, welche aber

ebenfalls auf beiden Seiten der Verifikationsgleichung angewandt werden kann, etwa

$$h(r) = h\left(g^{h(m)s^{-1}} y^{-h(r)s^{-1}} \bmod p\right)$$

für die Variante EG1, Nr 1.

Die zugrundeliegende Problematik, und das gilt ebenso für die ElGamal-Signatur mit elliptischen Kurven, ist die unterschiedliche binäre Länge der Elemente in beiden Gruppen, denn obwohl in der von G erzeugten Untergruppe q Elemente enthalten sind, sind einige davon größer als q, und entsprechend ist auch deren binäre Länge größer als die binäre Länge von q. So wird beispiesweise von $G = 7$ von der Gruppe $\mathbb{Z}_{29}$ mit Ordnung 28 die Untergruppe $\{1, 7, 20, 24, 23, 16, 25\}$ erzeugt. Alle Elemente in $\mathbb{Z}_7$ können mit 3 Bit dargestellt werden, die von 7 erzeugte Untergruppe enthält jedoch Elemente, die (falls die Darstellung nicht entsprechend angepasst wir) mit 5 Bit dargestellt werden müssen. Werden nun alle diese Elemente modulo 7 reduziert, so ergibt das $\{1, 0, 6, 3, 2, 2, 4\}$. Es können also Kollisionen entstehen, aber andererseits sind die so erzeugten Signaturen in Abhängigkeit der verwendeten Untergruppe kürzer.

Bei elliptischen Kurven stellt sich dieser Sachverhalt etwas anders dar. Ein Punkt $R = (x_R, y_R)$ auf einer elliptischen Kurve $E(\mathbb{F}_p)$ besteht aus zwei Elementen des Körpers $\mathbb{F}_p = \mathbb{Z}_p$. Die von einem Punkt $G \in E(\mathbb{F}_p)$ erzeugt Untergruppe der Ordnung $n = \#E(\mathbb{F}_p)/h$ enthält also möglicherweise Elemente, bei denen für eine Koordinate die Bedingung $|x_R| > |n|$ erfüllt ist. Dies hängt sowohl von der Frobeniusspur, als auch vom Kofaktor ab. Beim ECDSA [JMV99] wird nun die Funktion h_2 mit $\bar{r} = h_2(R) = x_R \bmod n$ implementiert. ECDSA ist eine Variante der ElGamal-Signatur EG-I. Nr.1, und die Signatur- und Verifikationsgleichungen haben die folgende Form.

$$\text{Signatur}: \quad s = k^{-1}(h(m) + a\bar{r}) \bmod n$$

$$\text{Verifikation}: \quad \bar{r} = f\left(h(m)s^{-1}G + \bar{r}s^{-1}Y\right)$$

Nun ist aber die Funktion h_2 nicht invertierbar, denn es existiert zu jedem Punkt der elliptischen Kurve ein Inverses mit identischer x-Koordinate. Die Funktion h_2 bildet also diese auf dasselbe $\bar{r}$ ab und es können weitere Punkte existieren, deren x-Koordinate auf dieses $\bar{r}$ abgegbildet werden. Alle Punkt mit identisch reduzierter x-Koordinate sind in der Restklasse $x_{R'} \in \{\bar{r} + \mathbb{Z}/\mathbb{Z}_n\}$ mit der Bedingung $x_{R'} < p$ zusammengefasst. Umgekehrt ist allerdings nicht jedes $\bar{r}$ dieser Menge die x-Koordinate eines Punktes auf der gegebenen elliptischen Kurve.

Beispiel 6.1 *Sei nochmals die elliptische Kurve über $\mathbb{F}_{29}$ aus Beispiel 4.1 mit $a = 1, b = 6$ und $\#E(\mathbb{F}_{29}) = 38$ gegeben. Der Punkt $G = (3, 6)$ erzeugt eine Untergruppe der Ordnung $n = 19$, womit also für den Kofaktor $h = 2$ folgt. Diese Untergruppe enthält inklusive $\mathcal{O}$ die Punkte:*

$1G = (3, 6)$	$2G = (22, 27)$	$3G = (0, 8)$	$4G = (20, 15)$	$5G = (12, 8)$
$6G = (23, 25)$	$7G = (25, 5)$	$8G = (17, 21)$	$9G = (10, 1)$	$10G = (10, 28)$
$11G = (17, 8)$	$12G = (25, 24)$	$13G = (23, 4)$	$14G = (12, 21)$	$15G = (20, 14)$
$16G = (0, 21)$	$17G = (22, 2)$	$18G = (3, 23)$	$19G = \mathcal{O}$	

Die Menge aller x-Koordinaten ist mit $\{3, 22, 0, 20, 12, 23, 25, 17, 10\}$ gegeben und es fällt auf, dass einige Werte dieser Menge größer als 19 sind, weswegen

die Menge der modulo 19 *reduzierten Elemente aus* $\{3, 3, 0, 1, 12, 4, 6, 17, 10\}$ *besteht. Die Punkte mit den* x*-Koordinaten* 3 *und* 22 *werden also auf einen identischen Wert abgebildet.*

Mit dem privaten Schlüssel $a = 5$*, und dem zufällig gewählten* $k = 17$ *und dem Punkt* $17G = (22, 2)$ *ergibt sich für den Hashwert* $h = 10$ *der Nachricht* m *der Signaturwert*

$$s = 17^{-1}(10 + 3 \cdot 5) \equiv 16 \bmod 19.$$

Die Verifikation mit $\bar{r} = 3$ *und* $s^{-1} = 6$ *ergibt*

$$(22, 2) = 10 \cdot 6G + 3 \cdot 6Y = 3(3, 6) + 18(12, 8) = (0, 8) + (12, 12) = (22, 2),$$

und deswegen $22 \bmod 19 = 3 = \bar{r}$*. Jedoch existieren weitere Nachrichten/Signaturpaare, welche erfolgreich mit dem identischen* $\bar{r}$ *verifiziert werden können. Für eines dieser Paare existiert ein* u*, mit dem* $s' = us$ *und* $h' = uh$ *gilt. So folgt aus* $u = 5$ *für den Hashwert der Nachricht* $h' = 12$*, für den Signaturwert* $s' = 4$ *und die Verifikation ergibt mit* $s'^{-1} = 5$

$$(22, 2) \neq 12 \cdot 5G + 3 \cdot 5Y = 3(3, 6) + 15(12, 8) = (0, 8) + (3, 23) = (22, 27),$$

und ebenso folgt für $u = 6$*,* $h' = 3$ *und* $s = 1$

$$(22, 2) \neq 3 \cdot 1G + 3 \cdot 1Y = 3(3, 6) + 3(12, 8) = (0, 8) + (20, 14) = (3, 23),$$

sowie $u = 12$*,* $h' = 6$ *und* $s = 2$

$$(22, 2) \neq 6 \cdot 10G + 3 \cdot 10Y = 3(3, 6) + 11(12, 8) = (0, 8) + (22, 2) = (3, 6).$$

Da nun aber $-(22, 2) = (22, 27)$ *ist, existieren zwei Signaturen* s *und* $-s'$ *unterschiedlicher Nachrichten für ein identisches* k*, womit sich mit Gleichung 6.4 der private Schlüssel berechnen lässt:*

$$a = -\frac{h's + hs'}{\bar{r}(s + s')} = -\frac{12 \cdot 16 + 10 \cdot 4}{3(16 + 4)} \equiv 5 \bmod 19$$

Für Kollisionen mit Punkten unterschiedlicher x*-Koordinate, ist dem Angreifer nicht unbedingt bekannt, in welchem Zusammenhang die Punkte stehen. In diesem Beispiel gibt es solche Kollisionen von* $(22, 2)$ *mit* G *und* $-G$ *und um das auszunutzen, müsste der Angreifer zusätzlich* k *bestimmen.*

Das Beispiel verdeutlicht insbesondere den Zusammenhang zwischen u und dem privaten Schlüssel a, und tatsächlich ist wegen

$$a = -\frac{h's + hs'}{\bar{r}(s + s')} = -\frac{uhs + uhs}{\bar{r}(s + us)} \equiv -\frac{2uh}{\bar{r}(u + 1)} \bmod n$$

und daher

$$\frac{u + 1}{u} \equiv -\frac{2h}{\bar{r}a} \bmod n, \quad u^{-1} \equiv -\frac{2h}{\bar{r}a} - 1 \bmod n$$

das Problem, u zu bestimmen, identisch mit der Berechnung des privaten Schlüssels, und somit existiert keine praktische Möglichkeit, die Kollisionen der Funktion $h_2(R) = x_R \bmod n$ auf direktem Weg auszunutzen.

Eine allgemeinere Betrachtung würde $s' = us$ und $h' = vh$ als Ansatz wählen, weswegen in obiger Gleichung in der Form

$$\frac{u+1}{u+v} \equiv -\frac{h}{\bar{r}a} \bmod n$$

also drei Unbekannte enthalten wären.

Jedoch ist nicht zwangsläufig der private Schlüssel das Ziel des Angreifers, weswegen an dieser Stelle kurz auf den Zusammenhang zwischen Kollisionen und der Erzeugung weiterer Signaturen eingegangen wird. Mit einer gültigen Signatur $(\bar{r}, s)$ gilt für ECDSA $R = s^{-1}hG + s^{-1}\bar{r}Y$, mit $R = kG$. Dann existiert für den Hashwert h' einer frei wählbaren Nachricht und einen Punkt R', mit $x_{R'} \bmod n = \bar{r}$ eine weitere gültige Signatur $R' = s'^{-1}h'G + s'^{-1}\bar{r}Y$ und damit zwei Gleichungen $\bar{r}Y = sR - hG$ und $\bar{r}Y = s'R' - h'G$, woraus sich nach Gleichsetzen $sR - hG = s'R' - h'G$ ergibt. Auflösen nach s' führt dann zu $s'R' = sR - hG + h'G$ und damit ist

$$s' = \log_{R'}(sR + (h - h')G).$$

Somit können die Kollisionen auch nicht direkt zum Erzeugen weiterer Signatur/Nachrichtenpaare verwendet werden. Mit $u = 1$ folgt, dass eine Signatur $(\bar{r}, s)$ für die Hashwerte von mindestens zwei Nachrichten verifizierbar ist. Mit $R' = -R$ und $s' = s$ gilt $2sR = (h(m) - h(m'))G$. Diese zweite Nachricht kann aber wegen $2ks \equiv h(m) - h(m') \bmod n$ nicht direkt berechnet werden. Abschließend ist noch zu bemerken, dass auch umgekehrt für jede Nachricht zwei gültige Signaturen $(\bar{r}, s)$ und $(\bar{r}, -s)$ existieren.

Die Sicherheit von ECDSA lässt sich jedoch nicht mit dem Forking Lemma im Random-Oracle-Modell zeigen, denn die Definition des generischen Signaturverfahrens, wie es auf Seite 47 beschrieben wurde, verlangt, dass der Parameter σ_1, der in diesem Fall mit $\bar{r} \in \mathbb{Z}_n$ gegeben ist, ein zufälliges Element in einer großen Menge ist. Diese Bedingung wird im ECDSA nicht erfüllt, da einerseits Elemente aufgrund der beschriebenen Kollisionen mit unterschiedlicher Wahrscheinlichkeit auftreten und andererseits nicht unbedingt jedes Element in $\mathbb{Z}_n$ erzeugt wird. Insgesamt kann somit keine Wahrscheinlichkeit für das Auftreten eines bestimmten Elementes angegeben werden. An dieser Stelle sei auf [NyS00] verwiesen. Dort wird die Verteilung von ECDSA-Signaturen modulo q näher betrachtet.

7 EC-ElGamal und SEC

In diesem Kapitel wird das generische ElGamal-Signaturverfahren für elliptische Kurven auf der Basis der *Standards for Efficient Cryptography* (SEC) spezifiziert. Dazu wird zuerst eine kurze Übersicht der existierenden Standards und deren Zusammenhänge dargestellt. Anschließend wird die Struktur der SEC erläutert und danach werden, entsprechend dieser Struktur, die benötigten Komponenten vorgestellt und das Signaturverfahren integriert.

7.1 Übersicht

Das auf elliptischen Kurven basierende Signaturverfahren ECDSA wurde in den Standards FIPS 186-2, ISO/IEC 14888-3, IEEE P1363, ANSI X9.62 und in SEC1 spezifiziert. Dabei referenziert FIPS 186-2 die Spezifikation von ANSI X9.62 und empfiehlt die Verwendung einer von 15 vorgegebenen elliptische Kurven. Das Ziel beider Spezifikation war einen möglichst hohen Sicherheitsstandard zu erreichen, indem einerseits eine exakt formulierte Schrittfolge für die einzelnen Prozesse wie Schlüsselerzeugung oder Signaturerzeugung/verifkation und andererseits die Kriterien für die Wahl einer elliptischen Kurve vorgegeben wurden. Während FIPS 186-2 neben ECDSA auch DSA und RSA behandelt, sind in ANSI X9.62 keine weiteren Algorithmen spezifiziert und für ECDSA auch keine speziellen elliptischen Kurven empfohlen. Allerdings entsprechen alle in FIPS 186-2 vorgesehenen Kurven den Vorgaben in ANSI X9.62. Die ECDSA Spezifikation von SEC entspricht im wesentlichen ebenso ANSI X9.62, lässt jedoch Charakteristik 2-Körper zu, deren Ordnung 160-Bit unterschreitet. Die in FIPS 186-2 empfohlenen elliptische Kurven sind auch in SEC enthalten. Zusätzlich zu ECDSA enthält SEC1 Spezifikationen für ECIES (Elliptic Curve Integrated Encryption Scheme) sowie für zwei auf elliptischen Kurven basierende Schlüsselaustauschverfahren, das EC-Diffie-Hellman Scheme und das EC-MQV Scheme (Menezes-Qu-Vanstone), welche alle im (noch nicht verabschiedeten) Standard ANSI X9.63 behandelt werden. Die Zielsetzung von SEC ist eine möglichst große Interoperabilität dadurch zu erreichen, dass Optionen, die nicht in sämtlichen Standards vorhanden sind, vermieden werden. Das betrifft neben der Hashfunktion RIPEMD-160 auch Parameter elliptischer Kurven und die Normal Basis für elliptische Kurven über Körper der Charakteristik 2. Im Gegensatz zu dieser Intention versteht sich der Standard IEEE P1363 eher als Referenz einer Vielzahl von verschiedenen Techniken, die auf Faktorisierung oder auf diskreten Logarithmen, sowohl für Primzahlkörper als auch für elliptische Kurven, basieren, ohne beispielsweise minimale Schlüsselgrößen zu verlangen.

7.2 Struktur der Standards for Efficient Cryptography

Die SEC bestehen aus zwei separaten Dokumenten, von denen SEC1 die Spezifikation der einzelnen Komponenten und SEC2 die Parameter der empfohlenen elliptischen Kurven enthält. Die spezifizierten Komponenten in SEC1 ummfassen die verwendeten Datentypen, welche hier in Abschnitt 7.4 vorgestellt werden sowie die benötigten kryptograpischen Primitive, zu denen neben der Hashfunktion private und öffentliche Schlüssel und Parameter der elliptischen Kurven gehören. Diese werden hier in Abschnitt 7.5 behandelt.

Die in SEC1 spezifizierten darauf basierenden Signatur-, Verschlüsselungs- und Schlüsselaustauschverfahren werden hier nicht wiedergegeben. Stattdessen wird das ElGamal Signaturverfahren in der Variante EG-I Nr. 1 in Abschnitt 7.3 kurz zusammengefasst und dann in Kapitel 7.6 spezifiziert.

Um erzeugte Signaturen oder Schlüssel und die dazugehörigen Parameter speichern und übertragen zu können, wird eine verbindliche Beschreibung benötigt. Als Basis dafür dient die in den ITU-T X.680, X.681, X.682 und X.683 [X.68?] Standards definierte Sprache *Abstract Syntax Notation One* (ASN.1), welche auch als ISO/IEC Standard veröffentlicht wurde und zur Beschreibung von Datenstrukturen verwendet wird. ASN.1 definiert mehrere Basistypen, wie Integer, Booleans und diverse Stringtypen, aber auch spezielle Typen wie UTC-Datum, sowie Möglichkeiten der Wertzuweisung und Referenzierung der Basistypen in zusammengesetzten Strukturen. ASN.1 heisst abstrakt, weil über die tatsächliche Codierung der Information zum Zweck der Übertragung keinerlei Vorgaben gemacht werden. Einige Möglichkeiten werden in den ITU-T Standards X.690 und X.691 [X.69?] spezifiziert. In der Praxis ist aber insbesondere DER (Distinguished Encoding Rules) relevant. Die Verwendung von ASN.1 für die zuvor spezifizierten Komponenten wird in Abschnitt 7.8 behandelt, ohne jedoch, aufgrund des Umfangs, näher auf die Sprache selbst einzugehen.

Im Anhang A werden die in SEC2 empfohlenen elliptischen Kurven aufgelistet.

7.3 EC-ElGamal

Es wird im Folgenden die ElGamal Variante EG-I Nr.1 betrachtet, und daher an dieser Stelle das Verfahren zusammengefasst. Will man SEC1 möglichst weitgehend folgen, steht als Hashfunktion lediglich SHA-1 zur Verfügung. Welche elliptischen Kurven vermieden werden müssen, wurde in Abschnitt 4.4.2 gesagt.

1. Schlüsselerzeugung: Der private Schlüssel ist eine Zahl $a \in_R \mathbb{Z}_n^*$. Mit dem Erzeuger $G \in E(\mathbb{F}_q)$ ist der öffentliche Schlüssel $Y = aG$.

2. Signaturerzeugung: Die Signatur (R, s) einer Nachricht m ist

$$s = k^{-1} \left(h(R\|m) - ah(R)\right) \mod n$$

   für einen Punkt $R = kG$ mit $k \in_R \mathbb{Z}_n^*$.

3. Signaturverifikation: Eine Signatur (R, s) für eine Nachricht m und den öffentlichen Schlüssel Y wird verifiziert, indem zwei Punkte auf der Kurve $E(\mathbb{F}_q)$ durch

$$X_1 = h(R)Y + sR \text{ und } X_2 = h(R\|m)G$$

   berechnet werden. Die Signatur ist gültig, wenn $X_1 = X_2$ erfüllt ist.

7.4 Datentypen

In SEC1 werden fünf verschiedene Datentypen vereinbart, die von den verschiedenen Komponenten verwendet werden. In diesem Abschnitt werden nun diese Datentypen und die dazughörenden Regeln der Konvertierung zwischen den einzelnen Typen beschrieben. Die Typen lassen sich nach der Art ihrer Verwendung in zwei Kategorieen unterteilen. Zum einen werden Typen benötigt, die algebraische Elemente repräsentieren, also Integer, FieldElement und EllipticCurvePoint, zum anderen muss die Nachricht und das Argument für die Hashfunktion repräsentiert werden wofür die Typen BitString und OctetSting verwendet werden.

7.4.1 Bit-Strings und Octet-Strings

Algorithmus 1 *BitString* $\rightarrow$ *OctetString*

Eingabe: Bit String B, bestehend aus $blen$ Bits

Ausgabe: Octet String M bestehend aus $mlen = \lceil blen/8 \rceil$ Octets (Bytes)

Der übergebene Bit String $B_0B_1 \ldots B_{blen-1}$ wird zuerst auf eine durch acht teilbare Länge gebracht, indem von links her die $blen$ mod 8 fehlenden Bits jeweils mit 0 aufgefüllt werden. Das Resultat wird dann in 8-Bit-Blöcke aufgeteilt und bildet den Octet String $M = M_0M_1 \ldots M_{mlen-1}$. Dazu sollte wie folgt vorgegangen werden:

1. Für i in dem Intervall $0 < i \leq mlen - 1$ setze

$$M_i \;=\; B_{blen-8-8(mlen-1-i)}B_{blen-7-8(mlen-1-i)} \cdots B_{blen-1-8(mlen-1-i)}$$

2. Für M_0 setze die linken $8(mlen) - blen$ Bits auf 0 und setze die übrigen $8 - (8(mlen) - blen)$ Bits aus $B_0B_1 \ldots B_{8-8(mlen)+blen-1}$ zusammen.

3. Gib M aus.

Algorithmus 2 *OctetString* $\rightarrow$ *BitString*

Eingabe: Octet String M, bestehend aus $mlen$ Octets

Ausgabe: Bit String B, bestehend aus $blen = 8(mlen)$ Bits

Da ein Octet aus acht Bits besteht, ist bei der Konvertierung lediglich das Folgende zu tun.

1. Für i in dem Intervall $0 \leq i \leq mlen - 1$ setze

$$B_{8i}B_{8i+1} \ldots B_{8i+7} = M_i.$$

2. Gib B aus.

7.4.2 Integers und Octet-Strings

Algorithmus 3 *Integer → OctetString*

Eingabe: eine positive Integer x und die gewünschte Anzahl $mlen$ der Octets; dazu muss $2^{8(mlen)} > x$ erfüllt sein

Ausgabe: Octet String M bestehend aus $mlen$ Octets

Die übergebene Integer x wird hier als Bit String interpretiert und in einzelne 'Ziffern' eines Zahlensystems zur Basis $2^8 = 256$ derart aufgeteilt, dass damit $x = x_{mlen-1}2^{8(mlen-1)} + x_{mlen-2}2^{8(mlen-2)} + \cdots x_1 2^8 + x_0$ gilt. Das Resultat ist dann der Octet String $M = M_0 M_1 \dots M_{mlen-1}$. Die Konvertierung sollte dabei wie folgt durch geführt werden.

1. Für i in dem Intervall $0 < i \leq mlen - 1$ setze
$$M_i \;=\; x_{mlen-1-i}$$
2. Gib M aus.

Algorithmus 4 *OctetString → Integer*

Eingabe: Octet String M bestehend aus $mlen$ Octets

Ausgabe: eine Integer x

Der übergebene Octet String $M = M_0 M_1 \dots M_{mlen-1}$ wird als Zahl zur Basis 256 aufgefasst und deren Konvertierung zur Integer x wie folgt durchgeführt werden.

1. Jedes M_i ist eine Zahl im Intervall $[0, 255]$. Berechne x mit:
$$x \;=\; \sum_{i=0}^{mlen-1} 2^{8(mlen-1-i)} M_i$$
2. Gib x aus.

7.4.3 Körperelemente und Octet-Strings

Algorithmus 5 *FieldElement* → *OctetString*

Eingabe: Ein Element a des endlichen Körpers $\mathbb{F}_q$

Ausgabe: Octet String M bestehend aus $mlen = \lceil \log_2 q/8 \rceil$ Octets

Das Element a ist entweder eine Integer des Körpers $\mathbb{F}_p$ oder ein Polynom des Körpers $\mathbb{F}_{2^m}$, dessen Koeffizienten als Bit String aufgefasst werden. Dabei wird der Koeffizient des Monoms mit dem höchsten Grad als das erste Bit von links definiert. Ein Elemente a wird dann wie folgt in einen Octet String $M = M_0 M_1 \dots M_{mlen-1}$ konvertiert:

1. Falls $q = p$ eine ungerade Primzahl ist, dann ist a eine Integer im Intervall $[0, p-1]$. In diesem Fall konvertiere a zu M mit Algorithmus 3 und gib das Ergebnis M aus.
2. Gilt dagegen $q = 2^m$, dann ist $a = a_{m-1}x^{m-1} + \cdots a_1 x + a_0$ ein binäres Polynom. Dann wird a nach M wie folgt konvertiert:
   2.1. Für i in dem Intervall $0 < i \leq mlen - 1$ setze
   $$M_i \;=\; a_{7+8(mlen-1-i)} a_{6+8(mlen-1-i)} \dots a_{8(mlen-1-i)}.$$
   2.2. Für M_0 setze die linken $8(mlen) - m$ Bits auf 0 und setze die übrigen $8 - (8(mlen) - m)$ Bits aus $a_0 a_1 \dots a_{8-8(mlen)+1}$ zusammen.
   2.3. Gib M aus.

Algorithmus 6 *OctetString* → *FieldElement*

Eingabe: einen Hinweis auf den verwendeten Körper $\mathbb{F}_q$ und ein Octet String M bestehend aus $mlen = \lceil \log_2 q/8 \rceil$ Octets

Ausgabe: ein Element a des endlichen Körpers $\mathbb{F}_q$ oder 'invalid'

In Abhängigkeit des verwendeten Körpers $\mathbb{F}_q$ wird der übergebene Octet String $M = M_0 M_1 \dots M_{mlen-1}$ mit $M_i = M_i^0 M_i^1 \dots M_i^7$ in eine Integer, falls $q = p$, oder einen Bit String, falls $q = 2^m$, in der folgenden Weise in das Körperelement a konvertiert.

1. Falls $q = p$ eine ungerade Primzahl ist, dann muss a eine Integer im Intervall $[0, p-1]$ sein. In diesem Fall konvertiere zu M a mit Algorithmus 4. Wenn nun a im Intervall $[0, p-1]$ liegt, gib das Ergebnis a aus. Andernfalls gib 'invalid' aus und brich den Vorgang ab.
2. Ist dagegen $q = 2^m$, dann muss a ein binäres Polynom vom maximalen Grad $m - 1$ sein. Setze $a = a_{m-1}x^{m-1} + \cdots a_1 x + a_0$ mit
   $$a_i \;=\; M_{mlen-1-(\lfloor i/8 \rfloor)}^{7-i+8(\lfloor i/8 \rfloor)}$$
   Gib 'invalid' aus und brich den Vorgang ab, falls die linken $8(mlen) - m$ Bits von M_0 nicht sämtlich gleich 0 sind. Andernfalls gib a aus.

7.4.4 EC-Punkte und Octet-Strings

Algorithmus 7 *EllipticCurvePoint* → *OctetString*

Eingabe: ein Punkt P auf einer elliptischen Kurve über $\mathbb{F}_q$, die durch die Körperelemente a, b bestimmt ist

Ausgabe: Octet String, bestehend aus $mlen$ Octets; dabei ist $mlen = 1$, falls $P = \mathcal{O}$, $mlen = \lceil(\log_2 q/8)\rceil+1$, falls $P \neq \mathcal{O}$ und Punkt-Komprimierung verwendet wird oder $mlen = 2\lceil(\log_2 q/8)\rceil+1$, falls $P \neq \mathcal{O}$ und Punkt-Komprimierung nicht verwendet wird.

Punkte auf elliptischen Kurven bestehen aus zwei Koordinaten. Da jedoch $y^2 = f(x)$ zwei y-Werte für ein x-Wert als Lösung besitzt und man beide jederzeit berechnen kann, genügt es lediglich zu wissen, welcher von beiden gemeint ist. Diese Information kann in einem Bit des x-Wertes gespeichert werden. Da man damit den Speicherbedarf für einen Punkt halbiert hat, wird dieses Verfahren $Punkt - Komprimierung$ genannt. Zusätzlich muss noch die Information darüber abgespeichert werden, ob Punkt-Komprimierung verwendet wird, oder nicht. Den kompletten Vorgang für die Konvertierung des Punktes P in den Octet String $M = M_0 M_1 \dots M_{mlen-1}$ oder $M = M_0$ beschreibt das folgende Verfahren:

1. Falls $P = \mathcal{O}$, gib $M = 00_{16}$ aus.
2. Falls $P = (x_P, y_P) \neq \mathcal{O}$ und Punkt-Komprimierung verwendet wird:
   - 2.1. Konvertiere mit Algorithmus 5 das Element x_P in einen Octet String X, bestehend aus $\lceil(\log_2 q/8)\rceil$ Octets.
   - 2.2. Bestimme ein Bit $\tilde{y}_P$ wie folgt (Dieses Bit repräsentiert dann die y-Koordinate):
     - 2.2.1. Wenn $q = p$ eine ungerade Primzahl ist, setze $\tilde{y}_P = y_P \bmod 2$.
     - 2.2.2. Wenn $q = 2^m$, setze $\tilde{y}_P = 0$ falls $x_P = 0$. Ansonsten berechne $z = z_{m-1}x^{m-1} + \cdots z_1 x + z_0$, so dass $z = y_P.{x_P}^{-1}$ und setze $\tilde{y}_P = z_0$.
   - 2.3. Falls $\tilde{y}_P = 0$, setze das Octet $Y = 02_{16}$. Falls stattdessen $\tilde{y}_P = 1$, setze das Octet $Y = 03_{16}$.
   - 2.4. Gib $M = Y \| X$ aus.
3. Falls $P = (x_P, y_P) \neq \mathcal{O}$ und Punkt-Komprimierung nicht verwendet wird:
   - 3.1. Konvertiere mit Algorithmus 5 das Element x_P in einen Octet String X, bestehend aus $\lceil(\log_2 q/8)\rceil$ Octets.
   - 3.2. Konvertiere mit Algorithmus 5 das Element y_P in einen Octet String Y, bestehend aus $\lceil(\log_2 q/8)\rceil$ Octets.
   - 3.3. Gib $M = 04_{16} \| X \| Y$ aus.

Algorithmus 8 *OctetString → EllipticCurvePoint*

Eingabe: eine elliptische Kurve über $\mathbb{F}_q$, die durch die Elemente a, b bestimmt ist und ein Octet String M, der entweder aus dem einzelnen Octet M_0 besteht, aus einem String mit $mlen = \lceil(\log_2 q/8)\rceil + 1$ Octets oder aus einem String mit $mlen = 2\lceil(\log_2 q/8)\rceil + 1$ Octets

Ausgabe: ein Punkt P einer elliptischen Kurve oder 'invalid'

1. Falls Gib $M = 00_{16}$, $P = \mathcal{O}$ aus.
2. Falls M aus $mlen = \lceil(\log_2 q/8)\rceil + 1$ Octets besteht:
   - 2.1. Spalte $M = Y\|X$ in ein einzelnes Octet Y und einen Octet String X, der aus $\lceil(\log_2 q/8)\rceil$ Octets besteht.
   - 2.2. Konvertiere X mit Algorithmus 6 in das Element x_P. Falls die Routine 'invalid' ausgibt, gib ebenfalls 'invalid' aus und brich den Vorgang ab.
   - 2.3. Falls $Y = 02_{16}$, setze $\tilde{y}_P = 0$. Falls $Y = 03_{16}$, setze $\tilde{y}_P = 1$. Ansonsten gib 'invalid' aus und brich den Vorgang ab.
   - 2.4. Bilde aus x_P und $\tilde{y}_P$ den Punkt P wie folgt.
     - 2.4.1. Wenn $q = p$ eine ungerade Primzahl ist, berechne das Element $\alpha \equiv {x_P}^3 + ax_P + b \bmod p$ und berechne als β eine Quadratwurzel mod p von α. Falls keine Quadratwurzeln mod p von α existieren, gib 'invalid' aus und den brich den Vorgang ab. Ansonsten setze $y_P = \beta$ falls $\beta \equiv \tilde{y}_P \bmod 2$ oder $y_P = p - \beta$ falls $\beta \not\equiv \tilde{y}_P \bmod 2$.
     - 2.4.2. Wenn $q = 2^m$ und $x_P = 0$, setze $y_P = {b^{1^m}}^{-1}$ als Element in $\mathbb{F}_{2^m}$.
     - 2.4.3. Wenn $q = 2^m$ und $x_P \neq 0$, berechne das Element β in $\mathbb{F}_{2^m}$ mit $\beta = x_P + a + b{x_P}^{-2}$ und finde ein weiteres Element $z = z_{m-1}x^{m-1} + \cdots z_1 x + z_0$ in $\mathbb{F}_{2^m}$, so dass $z^2 + z = \beta$. Falls kein solches z existiert, gib 'invalid' aus und brich den Vorgang ab. Andernfalls setze $y_P = x_P.z$ wenn $z_0 = \tilde{y}_P$ oder setze stattdessen $y_P = x_P.(z + 1)$ wenn $z_0 \neq \tilde{y}_P$.
   - 2.5. Gib $P = (x_P, y_P)$ aus.
3. Falls M aus $mlen = 2\lceil(\log_2 q/8)\rceil + 1$ Octets besteht:
   - 3.1. Spalte $M = W\|Y\|X$ in ein einzelnes Octet W und zwei aufeinanderfolgende Octet Strings X, Y, die jeweils aus $\lceil(\log_2 q/8)\rceil$ zusammenhängenden Octets bestehen.
   - 3.2. Überprüfe, dass $W = 04_{16}$ ist. Wenn diese Bedingung nicht erfüllt ist gib 'invalid' aus und brich den Vorgang ab.
   - 3.3. Konvertiere X mit Algorithmus 6 in das Element x_P aus $\mathbb{F}_q$. Falls die Routine 'invalid' ausgibt, gib ebenfalls 'invalid' aus und brich den Vorgang ab.
   - 3.4. Konvertiere Y mit Algorithmus 6 in das Element y_P aus $\mathbb{F}_q$. Falls die Routine 'invalid' ausgibt, gib ebenfalls 'invalid' aus und brich den Vorgang ab.

3.5. Überprüfe, dass $P = (x_P, y_P)$ die Gleichung der elliptischen Kurve mit den Parametern a, b erfüllt. Wenn P kein Punkt der Kurve ist, gib 'invalid' aus und brich den Vorgang ab.

3.6. Gib $P = (x_P, y_P)$ aus.

7.5 Kryptographische Komponenten

7.5.1 EC-Domain-Parameter

Die in [SEC1] beschriebenen Signatur- und Verschlüsselungsverfahren, und damit auch das in der vorliegenden Arbeit beschriebene Signaturverfahren, verwenden die additive Gruppe einer elliptischen Kurve über einen endlichen Körper $\mathbb{F}_q$. Diese Gruppe wird spezifiziert und ist eindeutig identifizierbar durch die sogenannten Domain-Parameter, welche in diesem Abschnitt für die zwei zulässigen Körper $\mathbb{F}_p$ und $\mathbb{F}_{2^m}$ (aus Abschnitt 4.3) beschrieben werden.

Domain-Parameter elliptischer Kurven über $\mathbb{F}_p$:

Die Domain Parameter einer elliptischen Kurve über $\mathbb{F}_p$ werden repräsentiert durch ein Sextupel der Form:

$$T = (p, a, b, G, n, h)$$

Die Kompenente p bezeichnet die Charakteristik der verwendeten Gruppe $\mathbb{F}_p$. Die Elemente $a, b \in \mathbb{F}_p$ definieren eine elliptische Kurve in kurzer Weierstraß-Normalform zu

$$E : y^2 \equiv x^3 + ax + b \bmod p.$$

Wie bereits beschrieben, wird diese Gleichung von $\#E(\mathbb{F}_p)$ Punkten erfüllt, wovon einer mit $G = (x_G, y_G)$ gegeben ist. Die Ordnung dieses Punktes ist mit der Integer n definiert und mit einer weiteren Integer h, dem Kofaktor, gilt $h = \#E(\mathbb{F}_p)/n$. Der Punkt G erzeugt also eine Untergruppe der Ordnung n von $E(\mathbb{F}_p)$. Es ist diese Untergruppe, in der die in [SEC1] beschriebenen Verfahren verwendet werden.

Domain Parameter elliptischer Kurven über $\mathbb{F}_{2^m}$:

Die Domain-Parameter einer elliptischen Kurve über $\mathbb{F}_{2^m}$ werden repräsentiert durch ein Septupel der Form:

$$T = (m, f(x), a, b, G, n, h)$$

Die Kompenente m definiert den verwendeten Körper $\mathbb{F}_{2^m}$ zusammen mit dem irreduziblen binären Polynom $f(x)$ vom Grad m, welches das Reduktionspolynom darstellt. Die Elemente $a, b \in \mathbb{F}_{2^m}$ definieren eine elliptische Kurve der folgenden Form:

$$E : y^2 + xy \equiv x^3 + ax^2 + b \text{ in } \mathbb{F}_{2^m}.$$

Diese Gleichung wird von $\#E(\mathbb{F}_{2^m})$ Punkten erfüllt, wovon einer mit $G = (x_G, y_G)$ gegeben ist. Die Ordnung dieses Punktes ist mit der Integer n definiert und mit einer weiteren Integer h, dem Kofaktor, gilt $h = \#E(\mathbb{F}_{2^m})/n$. Der Punkt G erzeugt also eine Untergruppe der Ordnung n von $E(\mathbb{F}_{2^m})$. Es ist diese Untergruppe, in der die in [SEC1] beschriebenen Verfahren verwendet werden.

7.5.2 EC-Domain-Parameter Generierung

Elliptische Kurven über endliche Körper wurden in Abschnitt 4.3 beschrieben. Von besonderem Interesse an dieser Stelle sind die in Abschnitt 4.4.2 erwähnten schwachen Kurven, die durch die Überprüfung in Schritt 2, beziehungsweise Schritt 3 der folgenden Algorithmen ausgeschlossen werden.

Die im folgenden beschriebenen Verfahren sind jedoch keine Algorithmen im strengeren Sinne, da sie die Wahl der zu verwendenden Unteralgorithmen (wenn keine der vorgeschlagenen Kurven verwendet wird) zur Kurvengenerierung nicht vorschreiben. Für eine Übersicht der existierenden Verfahren zur Punktezählung (Bestimmung der Ordnung einer elliptischen Kurve) und zur Erzeugung elliptischer Kurven mittels komplexer Multiplikation sei als Einstieg auf [BSS99], sowie auf die darin enthaltenen Zitate der weiterführenden Hintergrundliteratur verwiesen.

Algorithmus 9 *Generierung von EC-Domain-Parametern über* $\mathbb{F}_p$

Eingabe: der gewünschte ungefähre Sicherheitsparameter in Bits in Form einer Integer $t \in \{56, 64, 80, 96, 112, 128, 192, 256\}$

Ausgabe: EC-Domain-Parameter $T = (p, a, b, G, n, h)$ über einen Körper $\mathbb{F}_p$ derart, das für die Berechnung des diskreten Logarithmus in der mit T definierten elliptischen Kurve mit etwa 2^t Operationen gerechnet werden muss

Aus Gründen der Kompatibilität zu den existierenden Standards sollte eine der in [SEC2] spezifizierten Domain-Parameter verwendet werden, obwohl das beschriebene Verfahren die Verwendung sämtlicher Methoden der Kurvenerzeugung erlaubt.

1. Bestimme eine Primzahl p, welche $\lceil \log_2 p \rceil = 2t$ für $t \neq 256$ oder andernfalls $\lceil \log_2 p \rceil = 521$ für $t = 256$ erfüllt. Diese Primzahl definiert den zu verwendenden Körper $\mathbb{F}_p$.

2. Wähle zwei Elemente $a, b \in \mathbb{F}_p$ um die elliptische Kurve, gegeben durch $y^2 \equiv x^3 + ax + b \bmod p$, zu bestimmen. Wähle weiterhin einen Punkt $G = (x_G, y_G)$ auf dieser Kurve, die prime Ordnung n von G und eine weitere Integer, den Kofaktor $h = \#E(\mathbb{F}_p)/n$, so dass die folgenden Bedingungen erfüllt sind:

   - $4a^3 + 27b^2 \not\equiv 0 \bmod p$
   - $\#E(\mathbb{F}_p) \neq p$
   - $p^B \not\equiv 1 \bmod n$, für sämtliche B des Intervalls $1 \leq B \leq 20$
   - $h \leq 4$

3. Gib $T = (p, a, b, G, n, h)$ aus.

Algorithmus 10 *Generierung von EC-Domain-Parametern über* $\mathbb{F}_{2^m}$

Eingabe: Der gewünschte ungefähre Sicherheits Parameter in Bits in Form einer Integer $t \in \{56, 64, 80, 96, 112, 128, 192, 256\}$

Ausgabe: EC-Domain-Parameter $T = (m, f(x), a, b, G, n, h)$ über den Körper $\mathbb{F}_{2^m}$ derart, das für die Berechnung des diskreten Logarithmus in der mit T definierten elliptischen Kurve mit etwa 2^t Operationen gerechnet werden muss

Aus Gründen der Kompatibilität zu den existierenden Standards sollte, wie schon im letzten Verfahren im Kommentar erwähnt, eine der in [SEC2] spezifizierten Domain-Parameter verwendet werden.

1. Sei t' die kleinste Integer größer t mit $t' \in \{64, 80, 96, 112, 128, 192, 256, 512\}$. Wähle $m \in \{113, 131, 163, 193, 233, 239, 283, 409, 571\}$, so dass die Ungleichung $2t < m < 2t'$ erfüllt ist. Dieses m bestimt den Körper $\mathbb{F}_{2^m}$.
2. Wähle ein irreduzibles Polynom $f(x)$ aus Tabelle 1. Dieses ist das Reduktionspolynom für $\mathbb{F}_{2^m}$.
3. Wähle zwei Elemente $a, b \in \mathbb{F}_{2^m}$, um die elliptische Kurve, gegeben durch $y^2 + xy \equiv x^3 + ax^2 + b$ über $\mathbb{F}_{2^m}$, zu bestimmen. Wähle weiterhin einen Punkt $G = (x_G, y_G)$ auf dieser Kurve, die prime Ordnung n von G und eine weitere Integer, den Kofaktor $h = \#E(\mathbb{F}_{2^m})/n$, so dass die folgenden Bedingungen erfüllt sind:
   - $b \neq 0$
   - $\#E(\mathbb{F}_{2^m}) \neq 2^m$
   - $2^{mB} \not\equiv 1 \bmod n$, für sämtliche B des Intervalls $1 \leq B \leq 20$
   - $h \leq 4$
4. Gib $T = (m, f(x), a, b, G, n, h)$ aus.

$\mathbb{F}_{2^{113}}$	$f(x) = x^{113} + x^9 + 1$
$\mathbb{F}_{2^{131}}$	$f(x) = x^{131} + x^8 + x^3 + x^2 + 1$
$\mathbb{F}_{2^{163}}$	$f(x) = x^{163} + x^7 + x^6 + x^3 + 1$
$\mathbb{F}_{2^{193}}$	$f(x) = x^{193} + x^{15} + 1$
$\mathbb{F}_{2^{233}}$	$f(x) = x^{233} + x^{74} + 1$
$\mathbb{F}_{2^{239}}$	$f(x) = x^{239} + x^{36} + 1$ $f(x) = x^{239} + x^{158} + 1$
$\mathbb{F}_{2^{283}}$	$f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$
$\mathbb{F}_{2^{409}}$	$f(x) = x^{409} + x^{87} + 1$
$\mathbb{F}_{2^{571}}$	$f(x) = x^{571} + x^{10} + x^5 + x^2 + 1$

Tabelle 1: Reduktionspolynome der empfohlenen Körper

7.5.3 EC-Domain-Parameter Validierung

Es sind die folgenden vier Methoden vorgesehen, mit denen sich eine Partei U vergewissern kann, ob die mit den EC-Domain-Parametern T beschriebene elliptische Kurve den Vorgaben entspricht.

1. U verwendet die in diesem Abschnitt beschriebenen Algorithmen 11 und 12, um die EC-Domain-Parameter zu überprüfen.
2. U erzeugt die EC-Domain-Parameter T selbst mit den im letzten Abschnitt beschriebenen Algorithmen 9 und 10.
3. U erhält in einer beglaubigten Art und Weise die Versicherung, dass eine dritte Partei (i.d.R. eine CA) die Überprüfung der EC-Domain-Parameter mit den in diesem Abschnitt beschriebenen Algorithmen 11 und 12 durchgeführt hat.
4. U erhält in einer beglaubigten Art und Weise die Versicherung, dass eine dritte Partei (i.d.R. eine CA) die EC-Domain-Parameter mit den im letzten Abschnitt beschriebenen Algorithmen 9 und 10 erzeugt hat.

Die Überprüfung der EC-Domain-Parameter erfolgt mit den folgenden Algorithmen.

Algorithmus 11 *Validierung von EC-Domain-Parametern über* $\mathbb{F}_p$

Eingabe: EC-Domain-Parameter der Form $T = (p, a, b, G, n, h)$ und eine Integer $t \in \{56, 64, 80, 96, 112, 128, 192, 256\}$, die den geforderten ungefähren Sicherheitsparameter bezeichnet.

Ausgabe: Resultat der Validierung: 'valid' oder 'invalid'

1. Überprüfe, dass p eine ungerade Primzahl ist, welche $\lceil \log_2 p \rceil = 2t$ für $t \neq 256$ oder andernfalls $\lceil \log_2 p \rceil = 521$ für $t = 256$ erfüllt.
2. Überprüfe, dass a, b, x_G und y_G in dem Intervall $[0, p-1]$ enthalten sind.
3. Überprüfe, dass $4a^3 + 27b^2 \not\equiv 0 \bmod p$ erfüllt ist.
4. Überprüfe, dass $y_G^2 \equiv x_G^3 + ax_G + b \bmod p$ erfüllt ist.
5. Überprüfe, dass n eine ungerade Primzahl ist.
6. Überprüfe, dass $h \leq 4$ und $h = \lfloor (\sqrt{p} + 1)^2 / n \rfloor$ erfüllt sind.
7. Überprüfe, dass $nG = \mathcal{O}$ gilt.
8. Überprüfe, dass $p^B \not\equiv 1 \bmod n$, für sämtliche B des Intervalls $1 \leq B \leq 20$ erfüllt ist und $nh \neq p$ gilt.

Falls einer dieser Schritte nicht erfolgreich war, gib 'invalid' aus. Andernfalls gib 'valid' aus.

Algorithmus 12 *Validierung von EC-Domain-Parametern über* $\mathbb{F}_{2^m}$

Eingabe: EC-Domain-Parameter $T = (m, f(x), a, b, G, n, h)$ und eine Integer $t \in \{56, 64, 80, 96, 112, 128, 192, 256\}$, die den geforderten ungefähren Sicherheitsparameter bezeichnet.

Ausgabe: Resultat der Validierung: 'valid' oder 'invalid'

1. Sei t' die kleinste Integer größer t mit $t' \in \{64, 80, 96, 112, 128, 192, 256, 512\}$. Überprüfe, dass $m \in \{113, 131, 163, 193, 233, 239, 283, 409, 571\}$, so dass $2t < m < 2t'$ gilt.
2. Überprüfe, dass $f(x)$ ein irreduzibles Polynom aus Tabelle 1 ist.
3. Überprüfe, dass a, b, x_G und y_G binäre Polynome vom Grad $m-1$ oder weniger sind.
4. Überprüfe, dass $b \neq 0$ erfüllt ist.
5. Überprüfe, dass $y_G^2 + x_G y_G = x_G^3 + a x_G^2 + b$ erfüllt ist.
6. Überprüfe, dass n eine ungerade Primzahl ist.
7. Überprüfe, dass $h \leq 4$ und $h = \lfloor (\sqrt{2^m} + 1)^2 / n \rfloor$ erfüllt sind.
8. Überprüfe, dass $nG = \mathcal{O}$ gilt.
9. Überprüfe, dass $2^{mB} \not\equiv 1 \bmod n$, für sämtliche B des Intervalls $1 \leq B \leq 20$ erfüllt ist und $nh \neq 2^m$ gilt.

Falls einer dieser Schritte nicht erfolgreich war, gib 'invalid' aus. Andernfalls gib 'valid' aus.

7.5.4 Generierung des Schlüsselpaares

Die Schlüsselerzeugung wurde in Kapitel 5 beschrieben. Mit diesem Algorithmus wird der private Schlüssel d sowie der öffentliche Schlüssel Q erzeugt, welche von sämtlichen in SEC1 spezifizierten Verfahren verwendet.

Algorithmus 13 *Generierung des Schlüsselpaares*

Eingabe: gültige EC-Domain-Parameter der Form $T = (p, a, b, G, n, h)$ oder $T = (m, f(x), a, b, G, n, h)$

Ausgabe: ein Schlüsselpaar (d, Q) ür die Parameter T

1. Wähle ein (pseudo-)zufällige Integer d aus dem Intervall $[1, n-1]$.
2. Berechne $Q = dG$.
3. Gib (d, Q) aus.

7.5.5 Validierung des öffentlichen Schlüssels

Mit diesem Algorithmus wird überprüft, ob der übergebene Punkt Q tatsächlich auf der verwendeten durch T spezifizierten elliptischen Kurve liegt, ein skalares Vielfaches von G und nicht das neutrale Element repräsentiert.

Algorithmus 14 *Validierung des öffentlichen Schlüssels*

Eingabe: gültige EC-Domain-Parameter der Form $T = (p, a, b, G, n, h)$ oder $T = (m, f(x), a, b, G, n, h)$, und ein öffentlicher Schlüssel $Q = (x_Q, y_Q)$ für die Parameter T

Ausgabe: das Resultat 'valid' für einen gültigen oder 'invalid' für einen ungültigen Schlüssel

1. Überprüfe, dass $Q \neq \mathcal{O}$ ist.
2. Falls T die Domain-Parameter einer elliptischen Kurve über einen Körper $\mathbb{F}_p$ darstellen, überprüfe dass x_Q und y_Q Integer im Intervall $[1, p-1]$ sind und das außerdem

$$y_Q{}^2 \equiv x_Q{}^3 + ax_Q + b \bmod p.$$

   erfüllt ist.
3. Falls T die Domain-Parameter einer elliptischen Kurve über einen Körper $\mathbb{F}_{2^m}$ darstellen, überprüfe dass x_Q und y_Q binäre Polynome vom Grad maximal $m-1$ sind und dass außerdem

$$y_Q{}^2 + x_Q y_Q \equiv x_Q{}^3 + ax_Q{}^2 + b \bmod f(x).$$

   erfüllt ist.
4. Überprüfe, dass $nQ = \mathcal{O}$ ist.
5. Falls eine der beschriebenen Bedingungen nicht erfüllt ist, 'invalid' aus, ansonsten 'valid' aus.

7.6 EC-ElGamal in SEC

In diesem Abschnitt werden Konfiguration, sowie Signaturerzeugung und Signaturverifikation spezifiziert. Letztere beziehen sich dabei auf das im Abschnitt 5.4 vorgestelle generischen ElGamal Signaturverfahren für elliptische Kurven. Das Vorgehen zur Konfiguration ist äquivalent zu den für die Verwendung von ECDSA erforderlichen Schritten.

Wie bereits eingangs erwähnt, wird lediglich SHA-1 als Hashfunktion unterstützt.

7.6.1 Konfiguration

Die Parteien U und V sollten die folgende Konfigurations Prozedur verwenden, um EC-ElGamal zu verwenden.

1. U sollte festlegen, welche der vorgesehenen Hashfunktionen für die Signaturerzeugung verwendet werden soll. Sei *Hash* die verwendete Funktion, welche Hashwerte, die aus *hashlen* Octets, bestehen produziert.

2. U sollte mit Algorithmus 9 oder Algorithmus 10 eine elliptische Kurve durch EC-Domain-Parameter der Form $T = (m, f(x), a, b, G, n, h)$ oder $T = (p, a, b, G, n, h)$ für die gewünschte Sicherheitsstufe festlegen. U sollte sich mit Algorithmus 11 oder Algorithmus 12 davon überzeugen, dass die generierten EC-Domain-Parameter gültig sind.

3. V sollte in einer beglaubigten Art und Weise die von U festgelegten EC-Domain-Parameter T und Hashfunktion *Hash* erhalten.

V kann optional mit Algorithmus 11 oder Algorithmus 12 die Gültigkeit der EC-Domain-Parameter T überpüfen.

7.6.2 Schlüsselaustausch

Die Parteien U und V sollten die folgende Schlüsselaustausch-Prozedur verwenden, um EC-ElGamal zu verwenden.

1. U sollte mit Algorithmus 13 ein EC-Schlüsselpaar d_U, Q_U für die EC-Domain-Parameter T festlegen.

2. V sollte in einer beglaubigten Art und Weise den öffentlichen Schlüssel Q_U erhalten.

V kann optional mit Algorithmus 14 die Gültigkeit des öffentlichen Schlüssels Q_U überprüfen.

7.6.3 Erzeugen einer Signatur

Algorithmus 15 *Signatuerzeugung*

Eingabe: die Nachricht M, in Form eines Octet Strings

Ausgabe: eine Signatur $S = (R, s)$ der Nachricht M, bestehend aus dem Punkt R einer elliptischen Kurve, definiert durch die während der Konfiguration festgelegten Domain-Parameter T, und einer Integer s, oder 'invalid'

1. Erzeuge ein Schlüsselpaar (k, R) mit Algorithmus 13 für die mit den Domain-Parametern T gegebene elliptische Kurve.
2. Konvertiere den Punkt R mit Algorithmus 7 in einen Octet String $\bar{R}$.
3. Berechne mit der während der Konfigurationsphase festgelegten Hashfunktion den Hashwert $H_R = Hash(\bar{R})$ bestehend aus *hashlen* Octets. Falls die Hashfunktion 'invalid' ausgibt, gib ebenfalls 'invalid' aus und beende den Vorgang.
4. Berechne mit der während der Konfigurationsphase festgelegten Hashfunktion den Hashwert $H_M = Hash(\bar{R}\|M)$ bestehend aus *hashlen* Octets. Falls die Hashfunktion 'invalid' ausgibt, gib ebenfalls 'invalid' aus und beende den Vorgang.
5. Bestimme eine Integer h_m von H_M und eine Integer h_r von H_R wie folgt:
   - 5.1. Konvertiere mit Algorithmus 2 den Octet String H_M in einen Bit String $\bar{H}_M$ und den Octet String H_R in einen Bit String $\bar{H}_R$.
   - 5.2. Falls $\lceil \log_2 n \rceil \geq 8hashlen$, setze $\bar{E}_M = \bar{H}_M$ und $\bar{E}_R = \bar{H}_R$ oder bilde $\bar{E}_M$ aus den linken $\lceil \log_2 n \rceil$ Bits von $\bar{H}_M$ und $\bar{E}_R$ aus den linken $\lceil \log_2 n \rceil$ Bits von $\bar{H}_R$ falls $\lceil \log_2 n \rceil < 8hashlen$.
   - 5.3. Konvertiere mit Algorithmus 1 den Bit String $\bar{E}_M$ in den Octet String E_M und den Bit String $\bar{E}_R$ in den Octet String E_R.
   - 5.4. Konvertiere mit Algorithmus 4 den Octet String E_M in die Integer h_m und den Octet String E_R in die Integer h_r.
6. Berechne:
   $$s \equiv k^{-1}(h_m - h_r d) \bmod n$$
   Falls $s = 0$ oder $h_r = 0$ wiederhole den Vorgang ab Schritt 1.
7. Gib $S = (R, s)$ aus.

7.6.4 Verifizieren einer Signatur

Algorithmus 16 *Signaturverifikation*

Eingabe: eine Nachricht M in Form eines Octet Strings und U's Signatur $S = (R, s)$ für M

Ausgabe: das Resultat 'valid' für eine gültige Signatur oder 'invalid' für eine ungültige Signatur

1. Falls s keine Integer im Intervall $[1, n-1]$ ist, gib 'invalid' aus und beende den Vorgang.
2. Überprüfe mit Algorithmus 14, ob R ein gültiger Punkt der durch die Domain-Parameter T definierten elliptischen Kurve ist.
3. Konvertiere den Punkt R mit Algorithmus 7 in einen Octet String $\bar{R}$.
4. Berechne mit der während der Konfigurationsphase festgelegten Hashfunktion den Hashwert $H_R = Hash(\bar{R})$ bestehend aus *hashlen* Octets. Falls die Hashfunktion 'invalid' ausgibt, gib ebenfalls 'invalid' aus und beende den Vorgang.
5. Berechne mit der während der Konfigurationsphase festgelegten Hashfunktion den Hashwert $H_M = Hash(\bar{R} \| M)$ bestehend aus *hashlen* Octets. Falls die Hashfunktion 'invalid' ausgibt, gib ebenfalls 'invalid' aus und beende den Vorgang.
6. Bestimme eine Integer h_m von H_M und eine Integer h_r von H_R wie folgt:
   - 6.1. Konvertiere mit Algorithmus 2 den Octet String H_M in einen Bit String $\bar{H}_M$ und den Octet String H_R in einen Bit String $\bar{H}_R$.
   - 6.2. Falls $\lceil \log_2 n \rceil \geq 8hashlen$, setze $\bar{E}_M = \bar{H}_M$ und $\bar{E}_R = \bar{H}_R$ oder bilde $\bar{E}_M$ aus den linken $\lceil \log_2 n \rceil$ Bits von $\bar{H}_M$ und $\bar{E}_R$ aus den linken $\lceil \log_2 n \rceil$ Bits von $\bar{H}_R$ falls $\lceil \log_2 n \rceil < 8hashlen$.
   - 6.3. Konvertiere mit Algorithmus 1 den Bit String $\bar{E}_M$ in den Octet String E_M und den Bit String $\bar{E}_R$ in den Octet String E_R.
   - 6.4. Konvertiere mit Algorithmus 4 den Octet String E_M in die Integer h_m und den Octet String E_R in die Integer h_r.
   - 6.5. Falls $h_r = 0$ gib 'invalid' aus.
7. Berechne:
   $$X_1 = (x_{V_1}, y_{V_1}) = h_r Q_U + sR$$
   und
   $$X_2 = (x_{V_2}, y_{V_2}) = h_m G$$
8. Falls $X_1 = X_2$ gib 'valid' aus und falls $X_1 \neq X_2$ gib 'invalid' aus.

7.7 Bemerkungen

An dieser Stelle wird kurz auf eine mögliche Optimierung eingegangen. Schritt 5 in Algorithmus 14 verlangt zu überprüfen, ob der zu überprüfende Punkt durch Multiplikation mit der Untergruppenordnung das neutrale Element ergibt. Dieser Schritt ist für EC-Domainparameter, bei denen der Kofaktor $h = 1$ ist, nicht nötig. Für elliptische Kurven über $\mathbb{F}_{2^m}$ gilt stets $h > 1$, jedoch kann für den Fall $h = 2$ die Überprüfung beschleunigt werden. Dazu sei auf Lemma 1 in [Sma01] verwiesen. Dannach reicht es, wenn für den Punkt $Q = (x, y)$ und den EC-Domainparametern $T = (m, f(x), a, b, G, n, h)$ überprüft wird, ob $\mathrm{Tr}_{\mathbb{F}_{2^m}}(x) = 1$ und $\mathrm{Tr}_{\mathbb{F}_{2^m}}(b/x^2) = 0$ erfüllt ist. Für eine Definition der Spur („Trace“) eines Elementes aus $\mathbb{F}_{2^m}$ sei auf [Ros99] verwiesen.

Die Verifikationsbedingung, Schritt 7 in Algorithmus 16, kann auch alternativ überprüft werden, indem jeweils eine der Punktmultiplikationen durch eine Inversion und zwei Multiplikationen modulo n ersetzt wird.

1. Multiplikation mit h_m^{-1}. Damit ist die Verifikationsbedingung $X_1 = X_2$, für die Punkte

   $$X_1 = (x_{V_1}, y_{V_1}) = (h_m^{-1} h_r \bmod n)Q_U + (h_m^{-1} s \bmod n)R$$

   und

   $$X_2 = (x_{V_2}, y_{V_2}) = G.$$

2. Multiplikation mit h_r^{-1}. Dann gilt die Verifikationsbedingung $X_1 = X_2$, mit den Punkten

   $$X_1 = (x_{V_1}, y_{V_1}) = Q_U + (h_r^{-1} s \bmod n)R$$

   und

   $$X_2 = (x_{V_2}, y_{V_2}) = (h_r^{-1} h_m \bmod n)G.$$

3. Multiplikation mit s^{-1}. Dann lautet die Verifikationsbedingung $X_1 = X_2$, für die Punkte

   $$X_1 = (x_{V_1}, y_{V_1}) = (s^{-1} h_r \bmod n)Q_U + R$$

   und

   $$X_2 = (x_{V_2}, y_{V_2}) = (s^{-1} h_m \bmod n)G.$$

Die Multiplikation mit h_m^{-1} bedeutet einen Performancegewinn, da nun lediglich ein Punkt berechnet werden muss. Die Verbleibenden zwei Punktmultiplikationen können mit einem optimierten Algorithmus parallel durchgeführt werden.[Moe01]

7.8 ASN.1 Syntax

In diesem Abschnitt werden die soeben beschriebenen Ausgaben spezifiziert. Dabei werden zuerst die benötigten Basistypen, die Körperelemente und die daraus zusammengesetzten Punkte auf elliptischen Kurven, beschrieben. Eine Signatur, der öffentliche und der private Schlüssel sowie die EC-Domain-Parameter setzen sich aus diesen Datentypen zusammen und werden daher im Anschluss beschrieben.

Die Objekt-Bezeichner bilden eine Baumstruktur, dessen Wurzel im ANSI X9.62 Standard durch

```
ansi-X9-62 OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) 10045
}
```

repräsentiert ist.

7.8.1 Syntax für endliche Körper

In den SEC, und auch in der vorliegenden Arbeit, werden leiglich zwei Körper, $\mathbb{F}_p$ und $\mathbb{F}_{2^m}$, verwendet. In Abschnitt 7.4 wurde ein Algorithmus spezifiziert, der Elemente eines dieser Körper in Octet Strings konvertiert. Mit diesem Datentyp können daher Elemente beider Körper einheitlich beschrieben werden, weswegen Körperelemente als

```
FieldElement ::= OCTET STRING
```

definiert werden.

Neben den Elementen müssen die Körper selbst beschrieben werden können. Um zwischen den zwei Möglichkeiten zu unterscheiden, wird der Bezeichner

```
id-fieldType OBJECT IDENTIFIER ::= { ansi-X9-62 fieldType(1) }
```

zur Identifikation des verwendeten Körpers definiert. Damit wird nun ein Primzahlkörper großer Charakteristik durch

```
prime-field  OBJECT IDENTIFIER ::= { id-fieldType 1 }
Prime-p ::= INTEGER
```

vollständig beschrieben.

Für einen Körper der Charakteristik 2 müssen, neben einer Möglichkeit zur Identifikation, der Grad der Körpererweiterung m, sowie ein Reduktionspolynom angegeben werden. In Abschnitt 4.3.2 wurde erwähnt, dass für sämtliche m Reduktionspolynome als Trinom der Form $r(x) = x^m + x^k + 1$ oder Pentanom der Form $r(x) = x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, mit $k_3 > k_2 > k_1 > 0$, existieren. Erstere können daher mit einer Zahl, letztere müsssen mit drei Zahlen beschrieben werden. Daher wird die Beschreibung von Reduktionspolynomen definiert als:

```
Trinomial  ::= INTEGER
Pentanomial ::= SEQUENCE {
  k1  INTEGER,
  k2  INTEGER,
  k3  INTEGER
}
```

Der Bezeichner für Körper der Charakteristik 2 ist mit

```
characteristic-two-field  OBJECT IDENTIFIER ::= {
  id-fieldType 2
}
```

gegeben. Dieser Bezeichner wird dann verwendet, um die verschiedenen Basen zu unterscheiden:

```
id-characteristic-two-basis  OBJECT IDENTIFIER ::= {
  characteristic-two-field  basisType(3)
}
```

Die einzelnen Basen werden dann mit

```
gnBasis  OBJECT IDENTIFIER ::= { id-characteristic-two-basis 1 }
tpBasis  OBJECT IDENTIFIER ::= { id-characteristic-two-basis 2 }
ppBasis  OBJECT IDENTIFIER ::= { id-characteristic-two-basis 3 }
```

bezeichnet, wobei der erste Bezeichner die Normalbasis definiert, welche aber in den SEC nicht verwendet wird. Der zweite und dritte identifizieren dann ein Trinom und ein Pentanom. Mit dem durch

```
CHARACTERISTIC-TWO ::=  TYPE-IDENTIFIER
```

abstrakt definierten Typ, werden dann die Basen durch

```
BasisType CHARACTERISTIC-TWO ::= {
  { NULL        IDENTIFIED BY gnBasis } |
  { Trinomial   IDENTIFIED BY tpBasis } |
  { Pentanomial IDENTIFIED BY ppBasis }
}
```

erfasst, wobei die Normalbasis ohne Parameter, und die zwei möglichen Polynombasen mit dem jeweiligen Reduktionspolynom als Parameter beschrieben werden. Damit kann nun ein Körper der Charackteristik 2 vollständig erfasst werden als:

```
Characteristic-two ::= SEQUENCE {
  m          INTEGER
  basis      CHARACTERISTIC-TWO.&id({BasisTypes}),
  parameters CHARACTERISTIC-TWO.&Type({BasisTypes}{@basis})
}
```

Mit der Klasse

```
FIELD-ID ::= TYPE-IDENTIFIER
```

können schliesslich beide Körper einheitlich als

```
FieldTypes FIELD-ID ::= {
  { Prime-p            IDENTIFIED BY prime-field } |
  { Characteristic-two IDENTIFIED BY characteristic-two-field }
}
```

beschrieben werden, wobei an dieser Stelle die Möglichkeit besteht, die Definition um zusätzliche Körper zu erweitern. Damit wird schliesslich eine allgemeine Beschreibung eines endlichen Körpers durch

```
FieldID { FIELD-ID:IOSet } ::= SEQUENCE {
  fieldType  FIELD-ID.&id({IOSet}),
  parameters FIELD-ID.&Type({IOSet}{@fieldType})
}
```

definiert.

7.8.2 Syntax für EC-Domain-Parameter

Wie Körperelemente, so werden auch Punkte auf elliptischen Kurven als

```
ECPoint ::= OCTET STRING
```

definiert, um unabhängig von der optionalen Punktkompression oder des verwendeten Körpers, eine einheitliche Beschreibung zu erhalten. Eine elliptische Kurve sowohl über $\mathbb{F}_p$ als auch über $\mathbb{F}_{2^m}$ wird durch die zwei Paramter a und b, beides Elemente des verwendeten Körpers, definiert, weswegen

```
Curve ::= SEQUENCE {
  a    FieldElement,
  b    FieldElement,
  seed BIT STRING OPTIONAL
}
```

eine beliebige elliptische Kurve beschreibt. Der optionale Parameter `seed` kann zur Überprüfung verwendet werden, dass die Kurve (pseudo-)zufällig erzeugt wurde. Damit werden nun EC-Domain-Parameter aus Abschnitt 7.5.1 explizit als

```
ECParameters ::= SEQUENCE {
  version  INTEGER { ecpVer1(1) } (ecpVer1),
  fieldID  FieldID { {FieldTypes} },
  curve    Curve,
  base     ECPoint,
  order    INTEGER,
  cofactor INTEGER OPTIONAL,
  ...
}
```

definiert.

Als (empfohlene) Alternative zu den expliziten Parametern, kann auch eine der vorgegebenen elliptischen Kurven verwendet werden. Diese werden durch einen Namen identifiziert, welche in [SEC2] und im Anhang A aufgelistet werden. Alle Namen werden, als Bezeichner der Form

```
ellipticCurve OBJECT IDENTIFIER ::= { certicom-arc curve(0) }
```

einzeln durch die Klasse

```
CURVES ::= CLASS {
  &id OBJECT IDENTIFIER UNIQUE
}
WITH SYNTAX { ID &id }
```

repräsentiert, und in der Liste

```
CurveNames CURVES ::= {
  -- Curves over prime-order fields:
  { ID secp112r1 } | { ID secp112r2 } |
  { ID secp128r1 } | { ID secp128r2 } |
  { ID secp160k1 } | { ID secp160r1 } | { ID secp160r2 } |
```

```
  { ID secp192k1 } | { ID secp192r1 } |
  { ID secp224k1 } | { ID secp224r1 } |
  { ID secp256k1 } | { ID secp256r1 } |
  { ID secp384r1 } |
  { ID secp521r1 } |
  -- Curves over characteristic 2 fields:
  { ID sect113r1 } | { ID sect113r2 } |
  { ID sect131r1 } | { ID sect131r2 } |
  { ID sect163k1 } | { ID sect163r1 } | { ID sect163r2 } |
  { ID sect193r1 } | { ID sect193r2 } |
  { ID sect233k1 } | { ID sect233r1 } |
  { ID sect239k1 } |
  { ID sect283k1 } | { ID sect283r1 } |
  { ID sect409k1 } | { ID sect409r1 } |
  { ID sect571k1 } | { ID sect571r1 } ,
  ...
}
```

zusammengefasst.

Eine dritte Möglichkeit besteht darin, feste Parameter zu verwenden, die implizit allen beteiligten Parteien bekannt sind, oder individuelle Parameter, die aber von einer Certification Authority verwaltet werden und deshalb auch nicht abgespeichert oder übertragen werden müssen, da sie in beiden Fällen aus einer anderer Quelle stammen. Alle drei Möglichkeiten werden dann zusammengefasst zu:

```
Parameters ::= CHOICE {
  ecParameters ECParameters,
  namedCurve   CURVES.&id({CurveNames}),
  implicitlyCA NULL
}
```

7.8.3 Syntax für den öffentlichen Schlüssel

Die Syntax für die Repräsentation des öffentlichen Schlüssel basiert auf [BHP00], um beispielsweise in X.509 Zertifikaten verwendet werden zu können, die den öffentlichen Schlüssel in Form des Typs `SubjectPublicKeyInfo` erwarten. Dazu muss zuerst ein Bezeichner

```
id-publicKeyType OBJECT IDENTIFIER ::= { ansi-X9-62 keyType(2) }
```

definiert werden, mit dem dann der öffentliche Schlüssel, der aus einem Punkt der verwendeten elliptischen Kurve besteht, mit

```
id-ecPublicKey OBJECT IDENTIFIER ::= { id-publicKeyType 1 }
```

identifiziert werden kann. Mit der Klasse

```
ALGORITHM ::= CLASS {
  &id   OBJECT IDENTIFIER UNIQUE,
  &Type OPTIONAL
}
WITH SYNTAX { OID &id [PARAMS &Type] }
```

wird dann der öffentliche Schlüssel durch

```
ecPublicKeyType ALGORITHM ::= {
  OID id-ecPublicKey PARAMS Parameters {{SECGCurveNames}}
}
```

definiert, wobei der optionale Parameter `Parameters` den im letzten Abschnitt definierten EC-Domain-Parametern entspricht. Eine Liste aller möglichen Bezeichner für öffentliche Schlüssel unterschiedlicher Algorithmen wird durch

```
ECPKAlgorithms ALGORITHM ::= {
  ecPublicKeyType,
  ...
}
```

bereitgestellt, und kann dementsprechend an anderer Stelle erweitert werden. Damit wird nun der Typ

```
AlgorithmIdentifier{ ALGORITHM:IOSet } ::= SEQUENCE {
  algorithm  ALGORITHM.&id({IOSet}),
  parameters ALGORITHM.&Type({IOSet}{@algorithm })
}
```

definiert, welcher im eingangs erwähnten Typ

```
SubjectPublicKeyInfo ::= SEQUENCE {
  algorithm        AlgorithmIdentifier {{ECPKAlgorithms}},
  subjectPublicKey BIT STRING
}
```

verwendet wird.

7.8.4 Syntax für den privaten Schlüssel

Der in Abschnitt 7.5.4 definierte private Schlüssel, vom Typ Integer, wird durch den Typ Octet String in folgender Struktur erfasst:

```
ECPrivateKey{CURVES:IOSet} ::= SEQUENCE {
  version  INTEGER { ecPrivkeyVer1(1) } (ecPrivKeyVer1),
  privateKey  OCTE STRING,
  parameters  [0] Parameters{{IOSet}} OPTIONAL,
  publicKey   [1] BIT STRING OPTIONAL
}
```

Die Definitionen der beiden optionalen Parameter `parameters` und `publicKey` wurden im letzten Abschnitt besprochen.

7.8.5 Syntax für die EC-ElGamal Signatur

Wie in Abschnitt 7.6.3 beschrieben wurde, besteht eine EC-ElGamal-Signatur aus einem Punkt auf einer elliptischen Kurve und einer Integer, so dass beide in der Struktur

```
EC-ElGamal-Sig-Value ::= SEQUENCE {
  R  ECPoint,
  s  INTEGER
}
```

zusammengefasst werden können. Um diesen in einem X.509 Zertifikat verwenden zu könne, die Signaturen als Bit-Strings voraussetzen, wird eben dieser Bit-String aus der Codierung einer Signatur `EC-ElGamal-Sig-Value` erzeugt. Entsprechend der Syntax für ECDSA, würde die EC-ElGamal-Signatur mit den Bezeichnern

```
id-ecSigType  OBJECT IDENTIFIER ::= { ansi-X9-62 signatures(4) }
ecelgamal-with-SHA1 OBJECT IDENTIFIER ::= { id-ecSigType 2 }
```

und mit der in Abschnitt 7.8.3 eingeführten Klasse `ALGORITHM` der Typ

```
ecelgamal-SHA1 ALGORITHM ::= {
  OID ecelgamal-with-SHA1 PARAMS NULL
}
```

realisiert. Die nicht verwendeten EC-Domain-Parameter müssen beispielsweise aus dem öffentlichen Schlüssel desZ ertifikats extrahiert werden. Zu beachten ist, dass die für EC-ElGamal angegebenen OIDs sind nicht offiziell registriert sind.

8 Zusammenfassung

In der vorliegenden Arbeit wurden verschiedene Varianten des ElGamal-Signaturverfahrens im Kontext des Metal-ElGamal-Signaturverfahrens vorgestellt und auf elliptische Kurven übertragen. Dabei wurde die Verwendung des klassischen Hash-And-Sign-Prinzips modifiziert. Statt lediglich den Hashwert $h(m)$ der Nachricht m zu berechnen, wird der Hashwert $h(m, R)$ auch aus der Nachricht und dem Gruppenelement $R = kG$ gebildet.

Anschliessend wurden zwei Varianten des EC-ElGamal-Signaturverfahrens im Random-Oracle-Modell mit Hilfe des Forking-Lemma untersucht. Dabei wurde gezeigt, dass einem Adaptively-Chosen-Message-Angreifer, dem in polynomieller Zeit erfolgreich die existentielle Fälschung gelingt, ebenfalls die Berechnung des privaten Schlüssels und somit die Bestimmung eines diskreten Logarithmus in Gruppen mit primer Ordnung in polynomieller Zeit möglich ist. Diese Aussage basiert auf der Annahme, dass Hashfunktionen durch Random Oracles modelliert werden können. Da allerdings tatsächliche Hashfunktionen diesem Modell nicht entsprechen, kann die gewonne Aussage nicht als vollständiger Beweis betrachtet werden.

Es wurden anschließend verschiedene Ansätze diskutiert, um die Eigenschaften von Random Oracles besser anzunähern. Diese Ansätze wurden allerdings nicht weiter verfolgt, da zukünftige Hashfunktionen dem Random Oracle eher entsprechen könnten, oder aber andere Beweisverfahren ohne die Annahme eines Random Oracles auskommen könnten.

Schliesslich wurde eines der Verfahren in den Rahmen des existierenden Standards SEC integriert. SEC beinhaltet eine Spezifikation des ECDSA-Verfahrens, die der ursprünglichen im ANSI-Standard X.62 entspricht. Der modulare Aufbau dieser Standards ermöglicht das Weiterverwenden bereits spezifizierter Komponenten (Arithmetik, Schlüssel, Syntax) mit dem hinzugefügten Verfahren. Auf diese Weise sollte eine Implementierung bei einer bestehenden ECDSA-Implementierung mit äußerst geringem Aufwand verbunden sein.

Im Gegensatz zu den in dieser Arbeit betrachteten EC-ElGamal-Verfahren kann die Sicherheit des ECDSA-Signaturverfahren nicht im Random-Oracle gezeigt werden, was somit für die Verwendung des EC-ElGamal-Verfahrens spricht. Signaturgrössen und der Aufwand zu Signaturerzeugung und Verifikation sind für beide Verfahren praktisch gleich.

A SEC2: EC-Domain-Parameter

An dieser Stelle weredn die EC-Domain-Parameter der in SEC2 empfohlenen elliptischen Kurven wiedergegeben. Diese sind von der Form $T = (p, a, b, G, n, h)$ für elliptische Kurven über $\mathbb{F}_p$ oder $T = (m, f(x), a, b, G, n, h)$ für elliptische Kurven über $\mathbb{F}_{2^m}$. Der Erzeuger G der Kurve wird dabei in zwei Varianeten, mit und ohne Punktkompression als G_c bzw. G_u aufgelistet. Zum Abschluss wird die ASN.1 Syntax zur Referenzierung der einzelnen Kurven angegeben.

A.1 Elliptische Kurven über $\mathbb{F}_p$

secp112r1

$$
\begin{aligned}
p &= \texttt{DB7C 2ABF62E3 5E668076 BEAD208B} \\
&= (2^{128} - 3)/76439 \\
a &= \texttt{DB7C 2ABF62E3 5E668076 BEAD2088} \\
b &= \texttt{659E F8BA0439 16EEDE89 11702B22} \\
S &= \texttt{00F50B02 8E4D696E 67687561 51752904 72783FB1} \\
G_c &= \texttt{020948 7239995A 5EE76B55 F9C2F098} \\
G_u &= \texttt{04 09487239 995A5EE7 6B55F9C2 F098A89C E5AF8724 C0A23E0E} \\
&\quad \texttt{0FF77500} \\
n &= \texttt{DB7C 2ABF62E3 5E7628DF AC6561C5} \\
h &= \texttt{01}
\end{aligned}
$$

secp112r2

$$
\begin{aligned}
p &= \texttt{DB7C 2ABF62E3 5E668076 BEAD208B} \\
&= (2^{128} - 3)/76439 \\
a &= \texttt{6127 C24C05F3 8A0AAAF6 5C0EF02C} \\
b &= \texttt{51DE F1815DB5 ED74FCC3 4C85D709} \\
S &= \texttt{002757A1 114D696E 67687561 51755316 C05E0BD4} \\
G_c &= \texttt{034BA3 0AB5E892 B4E1649D D0928643} \\
G_u &= \texttt{04 4BA30AB5 E892B4E1 649DD092 8643ADCD 46F5882E 3747DEF3} \\
&\quad \texttt{6E956E97} \\
n &= \texttt{36DF 0AAFD8B8 D7597CA1 0520D04B} \\
h &= \texttt{04}
\end{aligned}
$$

secp128r1

$$
\begin{aligned}
p &= \texttt{FFFFFFFD FFFFFFFF FFFFFFFF FFFFFFFF} \\
&= 2^{128} - 2^{97} - 1 \\
a &= \texttt{FFFFFFFD FFFFFFFF FFFFFFFF FFFFFFFC} \\
b &= \texttt{E87579C1 1079F43D D824993C 2CEE5ED3} \\
S &= \texttt{000E0D4D 696E6768 75615175 0CC03A44 73D03679} \\
G_c &= \texttt{03 161FF752 8B899B2D 0C28607C A52C5B86} \\
G_u &= \texttt{04 161FF752 8B899B2D 0C28607C A52C5B86 CF5AC839 5BAFEB13} \\
&\quad \texttt{C02DA292 DDED7A83} \\
n &= \texttt{FFFFFFFE 00000000 75A30D1B 9038A115} \\
h &= \texttt{01}
\end{aligned}
$$

secp128r2

$$
\begin{aligned}
p &= \texttt{FFFFFFFD FFFFFFFF FFFFFFFF FFFFFFFF} \\
&= 2^{128} - 2^{97} - 1 \\
a &= \texttt{D6031998 D1B3BBFE BF59CC9B BFF9AEE1} \\
b &= \texttt{5EEEFCA3 80D02919 DC2C6558 BB6D8A5D} \\
S &= \texttt{004D696E 67687561 517512D8 F03431FC E63B88F4} \\
G_c &= \texttt{02 7B6AA5D8 5E572983 E6FB32A7 CDEBC140} \\
G_u &= \texttt{04 7B6AA5D8 5E572983 E6FB32A7 CDEBC140 27B6916A 894D3AEE} \\
&\quad \texttt{7106FE80 5FC34B44} \\
n &= \texttt{3FFFFFFF 7FFFFFFF BE002472 0613B5A3} \\
h &= \texttt{04}
\end{aligned}
$$

secp160k1

$$
\begin{aligned}
p &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFAC73} \\
&= 2^{160} - 2^{32} - 2^{14} - 2^{12} - 2^9 - 2^8 - 2^7 - 2^3 - 2^2 - 1 \\
a &= \texttt{00000000 00000000 00000000 00000000 00000000} \\
b &= \texttt{00000000 00000000 00000000 00000000 00000007} \\
G_c &= \texttt{02 3B4C382C E37AA192 A4019E76 3036F4F5 DD4D7EBB} \\
G_u &= \texttt{04 3B4C382C E37AA192 A4019E76 3036F4F5 DD4D7EBB 938CF935} \\
&\quad \texttt{318FDCED 6BC28286 531733C3 F03C4FEE} \\
n &= \texttt{01 00000000 00000000 0001B8FA 16DFAB9A CA16B6B3} \\
h &= \texttt{01}
\end{aligned}
$$

secp160r1

$$
\begin{aligned}
p &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 7FFFFFFF} \\
&= 2^{160} - 2^{31} - 1 \\
a &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 7FFFFFFC} \\
b &= \texttt{1C97BEFC 54BD7A8B 65ACF89F 81D4D4AD C565FA45} \\
S &= \texttt{1053CDE4 2C14D696 E6768756 1517533B F3F83345} \\
G_c &= \texttt{02 4A96B568 8EF57328 46646989 68C38BB9 13CBFC82} \\
G_u &= \texttt{04 4A96B568 8EF57328 46646989 68C38BB9 13CBFC82 23A62855} \\
&\quad \texttt{3168947D 59DCC912 04235137 7AC5FB32} \\
n &= \texttt{01 00000000 00000000 0001F4C8 F927AED3 CA752257} \\
h &= \texttt{01}
\end{aligned}
$$

secp160r2

$$
\begin{aligned}
p &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFAC73} \\
&= 2^{160} - 2^{32} - 2^{14} - 2^{12} - 2^9 - 2^8 - 2^7 - 2^3 - 2^2 - 1 \\
a &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFAC70} \\
b &= \texttt{B4E134D3 FB59EB8B AB572749 04664D5A F50388BA} \\
S &= \texttt{B99B99B0 99B323E0 2709A4D6 96E67687 56151751} \\
G_c &= \texttt{02 52DCB034 293A117E 1F4FF11B 30F7199D 3144CE6D} \\
G_u &= \texttt{04 52DCB034 293A117E 1F4FF11B 30F7199D 3144CE6D FEAFFEF2} \\
&\quad \texttt{E331F296 E071FA0D F9982CFE A7D43F2E} \\
n &= \texttt{01 00000000 00000000 0000351E E786A818 F3A1A16B} \\
h &= \texttt{01}
\end{aligned}
$$

secp192k1

$$
\begin{aligned}
p &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFEE37} \\
&= 2^{192} - 2^{32} - 2^{12} - 2^8 - 2^7 - 2^6 - 2^3 - 1 \\
a &= \texttt{00000000 00000000 00000000 00000000 00000000 00000000} \\
b &= \texttt{00000000 00000000 00000000 00000000 00000000 00000003} \\
G_c &= \texttt{03 DB4FF10E C057E9AE 26B07D02 80B7F434 1DA5D1B1 EAE06C7D} \\
G_u &= \texttt{04 DB4FF10E C057E9AE 26B07D02 80B7F434 1DA5D1B1 EAE06C7D} \\
&\quad \texttt{9B2F2F6D 9C5628A7 844163D0 15BE8634 4082AA88 D95E2F9D} \\
n &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFE 26F2FC17 0F69466A 74DEFD8D} \\
h &= \texttt{01}
\end{aligned}
$$

secp192r1

$$
\begin{aligned}
p &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFF} \\
&= 2^{192} - 2^{64} - 1 \\
a &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFC} \\
b &= \texttt{64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1} \\
S &= \texttt{3045AE6F C8422F64 ED579528 D38120EA E12196D5} \\
G_c &= \texttt{03 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012} \\
G_u &= \texttt{04 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012} \\
&\quad \texttt{07192B95 FFC8DA78 631011ED 6B24CDD5 73F977A1 1E794811} \\
n &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF 99DEF836 146BC9B1 B4D22831} \\
h &= \texttt{01}
\end{aligned}
$$

secp224k1

$$
\begin{aligned}
p &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFE56D} \\
&= 2^{224} - 2^{32} - 2^{12} - 2^{11} - 2^9 - 2^7 - 2^4 - 2 - 1 \\
a &= \texttt{00000000 00000000 00000000 00000000 00000000 00000000 00000000} \\
b &= \texttt{00000000 00000000 00000000 00000000 00000000 00000000 00000005} \\
G_c &= \texttt{03 A1455B33 4DF099DF 30FC28A1 69A467E9 E47075A9 0F7E650E} \\
&\quad \texttt{B6B7A45C} \\
G_u &= \texttt{04 A1455B33 4DF099DF 30FC28A1 69A467E9 E47075A9 0F7E650E} \\
&\quad \texttt{B6B7A45C 7E089FED 7FBA3442 82CAFBD6 F7E319F7 C0B0BD59 E2CA4BDB} \\
&\quad \texttt{556D61A5} \\
n &= \texttt{01 00000000 00000000 00000000 0001DCE8 D2EC6184 CAF0A971} \\
&\quad \texttt{769FB1F7} \\
h &= \texttt{01}
\end{aligned}
$$

secp224r1

$$
\begin{aligned}
p &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 00000000 00000001} \\
&= 2^{224} - 2^{96} + 1 \\
a &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFE} \\
b &= \texttt{B4050A85 0C04B3AB F5413256 5044B0B7 D7BFD8BA 270B3943 2355FFB4} \\
S &= \texttt{BD713447 99D5C7FC DC45B59F A3B9AB8F 6A948BC5} \\
G_c &= \texttt{02 B70E0CBD 6BB4BF7F 321390B9 4A03C1D3 56C21122 343280D6} \\
&\quad \texttt{115C1D21} \\
G_u &= \texttt{04 B70E0CBD 6BB4BF7F 321390B9 4A03C1D3 56C21122 343280D6} \\
&\quad \texttt{115C1D21 BD376388 B5F723FB 4C22DFE6 CD4375A0 5A074764 44D58199} \\
&\quad \texttt{85007E34} \\
n &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFF16A2 E0B8F03E 13DD2945 5C5C2A3D} \\
h &= \texttt{01}
\end{aligned}
$$

secp256k1

$$
\begin{aligned}
p &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE} \\
&\quad \texttt{FFFFFC2F} \\
&= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1 \\
a &= \texttt{00000000 00000000 00000000 00000000 00000000 00000000 00000000} \\
&\quad \texttt{00000000} \\
b &= \texttt{00000000 00000000 00000000 00000000 00000000 00000000 00000000} \\
&\quad \texttt{00000007} \\
G_c &= \texttt{02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9} \\
&\quad \texttt{59F2815B 16F81798} \\
G_u &= \texttt{04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9} \\
&\quad \texttt{59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448} \\
&\quad \texttt{A6855419 9C47D08F FB10D4B8} \\
n &= \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C} \\
&\quad \texttt{D0364141} \\
h &= \texttt{01}
\end{aligned}
$$

secp256r1

p = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF
= $2^{224}(2^{32}-1)+2^{192}+2^{96}-1$
a = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFC
b = 5AC635D8 AA3A93E7 B3EBBD55 769886BC 651D06B0 CC53B0F6 3BCE3C3E 27D2604B
S = C49D3608 86E70493 6A6678E1 139D26B7 819F7E90
G_c = 03 6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0 F4A13945 D898C296
G_u = 04 6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0 F4A13945 D898C296 4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE CBB64068 37BF51F5
n = FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551
h = 01

secp384r1

p = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF 00000000 00000000 FFFFFFFF
= $2^{384}-2^{128}-2^{96}+2^{32}-1$
a = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF 00000000 00000000 FFFFFFFC
b = B3312FA7 E23EE7E4 988E056B E3F82D19 181D9C6E FE814112 0314088F 5013875A C656398D 8A2ED19D 2A85C8ED D3EC2AEF
S = A335926A A319A27A 1D00896A 6773A482 7ACDAC73
G_c = 03 AA87CA22 BE8B0537 8EB1C71E F320AD74 6E1D3B62 8BA79B98 59F741E0 82542A38 5502F25D BF55296C 3A545E38 72760AB7
G_u = 04 AA87CA22 BE8B0537 8EB1C71E F320AD74 6E1D3B62 8BA79B98 59F741E0 82542A38 5502F25D BF55296C 3A545E38 72760AB7 3617DE4A 96262C6F 5D9E98BF 9292DC29 F8F41DBD 289A147C E9DA3113 B5F0B8C0 0A60B1CE 1D7E819D 7A431D7C 90EA0E5F
n = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF C7634D81 F4372DDF 581A0DB2 48B0A77A ECEC196A CCC52973
h = 01

secp521r1

p = 01FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
= $2^{521}-1$
a = 01FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFC
b = 0051 953EB961 8E1C9A1F 929A21A0 B68540EE A2DA725B 99B315F3 B8B48991 8EF109E1 56193951 EC7E937B 1652C0BD 3BB1BF07 3573DF88 3D2C34F1 EF451FD4 6B503F00
S = D09E8800 291CB853 96CC6717 393284AA A0DA64BA
G_c = 0200C6 858E06B7 0404E9CD 9E3ECB66 2395B442 9C648139 053FB521 F828AF60 6B4D3DBA A14B5E77 EFE75928 FE1DC127 A2FFA8DE 3348B3C1 856A429B F97E7E31 C2E5BD66
G_u = 04 00C6858E 06B70404 E9CD9E3E CB662395 B4429C64 8139053F B521F828 AF606B4D 3DBAA14B 5E77EFE7 5928FE1D C127A2FF A8DE3348 B3C1856A 429BF97E 7E31C2E5 BD660118 39296A78 9A3BC004 5C8A5FB4 2C7D1BD9 98F54449 579B4468 17AFBD17 273E662C 97EE7299 5EF42640 C550B901 3FAD0761 353C7086 A272C240 88BE9476 9FD16650
n = 01FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFA 51868783 BF2F966B 7FCC0148 F709A5D0 3BB5C9B8 899C47AE BB6FB71E 91386409
h = 01

A.2 Elliptische Kurven über $\mathbb{F}_{2^m}$

sect113r1

$f(x) = x^{113} + x^9 + 1$
a = 003088 250CA6E7 C7FE649C E85820F7
b = 00E8BE E4D3E226 0744188B E0E9C723
S = 10E723AB 14D696E6 76875615 1756FEBF 8FCB49A9
G_c = 03009D73 616F35F4 AB1407D7 3562C10F
G_u = 04009D 73616F35 F4AB1407 D73562C1 0F00A528 30277958 EE84D131 5ED31886
n = 010000 00000000 00D9CCEC 8A39E56F
h = 02

sect113r2

$f(x) = x^{113} + x^9 + 1$
a = 006899 18DBEC7E 5A0DD6DF C0AA55C7
b = 0095E9 A9EC9B29 7BD4BF36 E059184F
S = 10C0FB15 760860DE F1EEF4D6 96E67687 5615175D
G_c = 0301A57A 6A7B26CA 5EF52FCD B8164797
G_u = 0401A5 7A6A7B26 CA5EF52F CDB81647 9700B3AD C94ED1FE 674C06E6 95BABA1D
n = 010000 00000000 0108789B 2496AF93
h = 02

sect131r1

$f(x) = x^{131} + x^8 + x^3 + x^2 + 1$
a = 07 A11B09A7 6B562144 418FF3FF 8C2570B8
b = 02 17C05610 884B63B9 C6C72916 78F9D341
S = 4D696E67 68756151 75985BD3 ADBADA21 B43A97E2
G_c = 0300 81BAF91F DF9833C4 0F9C1813 43638399
G_u = 040081 BAF91FDF 9833C40F 9C181343 63839907 8C6E7EA3 8C001F73 C8134B1B 4EF9E150
n = 04 00000000 00000002 3123953A 9464B54D
h = 02

sect131r2

$f(x) = x^{131} + x^8 + x^3 + x^2 + 1$
a = 03 E5A88919 D7CAFCBF 415F07C2 176573B2
b = 04 B8266A46 C55657AC 734CE38F 018F2192
S = 985BD3AD BAD4D696 E6768756 15175A21 B43A97E3
G_c = 0303 56DCD8F2 F95031AD 652D2395 1BB366A8
G_u = 040356 DCD8F2F9 5031AD65 2D23951B B366A806 48F06D86 7940A536 6D9E265D E9EB240F
n = 04 00000000 00000001 6954A233 049BA98F
h = 02

sect163k1

$f(x) = x^{163} + x^7 + x^6 + x^3 + 1$
a = 00 00000000 00000000 00000000 00000000 00000001
b = 00 00000000 00000000 00000000 00000000 00000001
G_c = 0302 FE13C053 7BBC11AC AA07D793 DE4E6D5E 5C94EEE8
G_u = 0402FE 13C0537B BC11ACAA 07D793DE 4E6D5E5C 94EEE802 89070FB0 5D38FF58 321F2E80 0536D538 CCDAA3D9
n = 04 00000000 00000000 00020108 A2E0CC0D 99F8A5EF
h = 02

sect163r1

$f(x)$ = $x^{163} + x^7 + x^6 + x^3 + 1$
a = 07 B6882CAA EFA84F95 54FF8428 BD88E246 D2782AE2
b = 07 13612DCD DCB40AAB 946BDA29 CA91F73A F958AFD9
S = 24B7B137 C8A14D69 6E676875 6151756F D0DA2E5C
G_c = 0303 69979697 AB438977 89566789 567F787A 7876A654
G_u = 040369 979697AB 43897789 56678956 7F787A78 76A65400 435EDB42 EFAFB298 9D51FEFC E3C80988 F41FF883
n = 03 FFFFFFFF FFFFFFFF FFFF48AA B689C29C A710279B
h = 02

sect163r2

$f(x)$ = $x^{163} + x^7 + x^6 + x^3 + 1$
a = 00 00000000 00000000 00000000 00000000 00000001
b = 02 0A601907 B8C953CA 1481EB10 512F7874 4A3205FD
S = 85E25BFE 5C86226C DB12016F 7553F9D0 E693A268
G_c = 0303 F0EBA162 86A2D57E A0991168 D4994637 E8343E36
G_u = 0403F0 EBA16286 A2D57EA0 991168D4 994637E8 343E3600 D51FBC6C 71A0094F A2CDD545 B11C5C0C 797324F1
n = 04 00000000 00000000 000292FE 77E70C12 A4234C33
h = 02

sect193r1

$f(x)$ = $x^{193} + x^{15} + 1$
a = 00 17858FEB 7A989751 69E171F7 7B4087DE 098AC8A9 11DF7B01
b = 00 FDFB49BF E6C3A89F ACADAA7A 1E5BBC7C C1C2E5D8 31478814
S = 103FAEC7 4D696E67 68756151 75777FC5 B191EF30
G_c = 0301 F481BC5F 0FF84A74 AD6CDF6F DEF4BF61 79625372 D8C0C5E1
G_u = 0401F4 81BC5F0F F84A74AD 6CDF6FDE F4BF6179 625372D8 C0C5E100 25E399F2 903712CC F3EA9E3A 1AD17FB0 B3201B6A F7CE1B05
n = 01 00000000 00000000 00000000 C7F34A77 8F443ACC 920EBA49
h = 02

sect193r2

$f(x)$ =
a = 01 63F35A51 37C2CE3E A6ED8667 190B0BC4 3ECD6997 7702709B
b = 00 C9BB9E89 27D4D64C 377E2AB2 856A5B16 E3EFB7F6 1D4316AE
S = 10B7B4D6 96E67687 56151751 37C8A16F D0DA2211
G_c = 0300 D9B67D19 2E0367C8 03F39E1A 7E82CA14 A651350A AE617E8F
G_u = 0400D9 B67D192E 0367C803 F39E1A7E 82CA14A6 51350AAE 617E8F01 CE943356 07C304AC 29E7DEFB D9CA01F5 96F92722 4CDECF6C
n = 01 00000000 00000000 00000001 5AAB561B 005413CC D4EE99D5
h = 02

sect233k1

$f(x)$ = $x^{233} + x^{74} + 1$
a = 0000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
b = 0000 00000000 00000000 00000000 00000000 00000000 00000000 00000001
G_c = 020172 32BA853A 7E731AF1 29F22FF4 149563A4 19C26BF5 0A4C9D6E EFAD6126
G_u = 04 017232BA 853A7E73 1AF129F2 2FF41495 63A419C2 6BF50A4C 9D6EEFAD 612601DB 537DECE8 19B7F70F 555A67C4 27A8CD9B F18AEB9B 56E0C11056FAE6A3
n = 80 00000000 00000000 00000000 00069D5B B915BCD4 6EFB1AD5 F173ABDF
h = 04

sect233r1

$f(x) = x^{233} + x^{74} + 1$

$a =$ 0000 00000000 00000000 00000000 00000000 00000000 00000000 00000001

$b =$ 0066 647EDE6C 332C7F8C 0923BB58 213B333B 20E9CE42 81FE115F 7D8F90AD

$S =$ 74D59FF0 7F6B413D 0EA14B34 4B20A2DB 049B50C3

$G_c =$ 0300FA C9DFCBAC 8313BB21 39F1BB75 5FEF65BC 391F8B36 F8F8EB73 71FD558B

$G_u =$ 04 00FAC9DF CBAC8313 BB2139F1 BB755FEF 65BC391F 8B36F8F8 EB7371FD 558B0100 6A08A419 03350678 E58528BE BF8A0BEF F867A7CA 36716F7E 01F81052

$n =$ 0100 00000000 00000000 00000000 0013E974 E72F8A69 22031D26 03CFE0D7

$h =$ 02

sect239k1

$f(x) = x^{239} + x^{158} + 1$

$a =$ 0000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

$b =$ 0000 00000000 00000000 00000000 00000000 00000000 00000000 00000001

$G_c =$ 0329A0 B6A887A9 83E97309 88A68727 A8B2D126 C44CC2CC 7B2A6555 193035DC

$G_u =$ 04 29A0B6A8 87A983E9 730988A6 8727A8B2 D126C44C C2CC7B2A 65551930 35DC7631 0804F12E 549BDB01 1C103089 E73510AC B275FC31 2A5DC6B7 6553F0CA

$n =$ 2000 00000000 00000000 00000000 005A79FE C67CB6E9 1F1C1DA8 00E478A5

$h =$ 04

sect283k1

$f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$

$a =$ 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

$b =$ 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001

$G_c =$ 02 0503213F 78CA4488 3F1A3B81 62F188E5 53CD265F 23C1567A 16876913 B0C2AC24 58492836

$G_u =$ 04 0503213F 78CA4488 3F1A3B81 62F188E5 53CD265F 23C1567A 16876913 B0C2AC24 58492836 01CCDA38 0F1C9E31 8D90F95D 07E5426F E87E45C0 E8184698 E4596236 4E341161 77DD2259

$n =$ 01FFFFFF FFFFFFFF FFFFFFFF FFFFE9AE 2ED07577 265DFF7F 94451E06 1E163C61

$h =$ 04

sect283r1

$f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$

$a =$ 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001

$b =$ 027B680A C8B8596D A5A4AF8A 19A0303F CA97FD76 45309FA2 A581485A F6263E31 3B79A2F5

$S =$ 77E2B073 70EB0F83 2A6DD5B6 2DFC88CD 06BB84BE

$G_c =$ 03 05F93925 8DB7DD90 E1934F8C 70B0DFEC 2EED25B8 557EAC9C 80E2E198 F8CDBECD 86B12053

$G_u =$ 04 05F93925 8DB7DD90 E1934F8C 70B0DFEC 2EED25B8 557EAC9C 80E2E198 F8CDBECD 86B12053 03676854 FE24141C B98FE6D4 B20D02B4 516FF702 350EDDB0 826779C8 13F0DF45 BE8112F4

$n =$ 03FFFFFF FFFFFFFF FFFFFFFF FFFFEF90 399660FC 938A9016 5B042A7C EFADB307

$h =$ 02

sect409k1

$f(x)$ = $x^{409} + x^{87} + 1$

a = 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000

b = 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000001

G_c = 03 0060F05F 658F49C1 AD3AB189 0F718421 0EFD0987 E307C84C
27ACCFB8 F9F67CC2 C460189E B5AAAA62 EE222EB1 B35540CF E9023746

G_u = 04 0060F05F 658F49C1 AD3AB189 0F718421 0EFD0987 E307C84C
27ACCFB8 F9F67CC2 C460189E B5AAAA62 EE222EB1 B35540CF E9023746
01E36905 0B7C4E42 ACBA1DAC BF04299C 3460782F 918EA427 E6325165
E9EA10E3 DA5F6C42 E9C55215 AA9CA27A 5863EC48 D8E0286B

n = 7FFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFE5F
83B2D4EA 20400EC4 557D5ED3 E3E7CA5B 4B5C83B8 E01E5FCF

h = 04

sect409r1

$f(x)$ = $x^{409} + x^{87} + 1$

a = 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000001

b = 0021A5C2 C8EE9FEB 5C4B9A75 3B7B476B 7FD6422E F1F3DD67 4761FA99
D6AC27C8 A9A197B2 72822F6C D57A55AA 4F50AE31 7B13545F

S = 4099B5A4 57F9D69F 79213D09 4C4BCD4D 4262210B

G_c = 03 015D4860 D088DDB3 496B0C60 64756260 441CDE4A F1771D4D
B01FFE5B 34E59703 DC255A86 8A118051 5603AEAB 60794E54 BB7996A7

G_u = 04 015D4860 D088DDB3 496B0C60 64756260 441CDE4A F1771D4D
B01FFE5B 34E59703 DC255A86 8A118051 5603AEAB 60794E54 BB7996A7
0061B1CF AB6BE5F3 2BBFA783 24ED106A 7636B9C5 A7BD198D 0158AA4F
5488D08F 38514F1F DF4B4F40 D2181B36 81C364BA 0273C706

n = 01000000 00000000 00000000 00000000 00000000 00000000 000001E2
AAD6A612 F33307BE 5FA47C3C 9E052F83 8164CD37 D9A21173

h = 02

sect571k1

$f(x)$ = $x^{571} + x^{10} + x^5 + x^2 + 1$

a = 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000

b = 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000
= 00000000 00000000 00000000 00000001

G_c = 02 026EB7A8 59923FBC 82189631 F8103FE4 AC9CA297 0012D5D4
60248048 01841CA4 43709584 93B205E6 47DA304D B4CEB08C BBD1BA39
494776FB 988B4717 4DCA88C7 E2945283 A01C8972

G_u = 04 026EB7A8 59923FBC 82189631 F8103FE4 AC9CA297 0012D5D4
60248048 01841CA4 43709584 93B205E6 47DA304D B4CEB08C BBD1BA39
494776FB 988B4717 4DCA88C7 E2945283 A01C8972 0349DC80 7F4FBF37
4F4AEADE 3BCA9531 4DD58CEC 9F307A54 FFC61EFC 006D8A2C 9D4979C0
AC44AEA7 4FBEBBB9 F772AEDC B620B01A 7BA7AF1B 320430C8 591984F6
01CD4C14 3EF1C7A3

n = 02000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 131850E1 F19A63E4 B391A8DB 917F4138 B630D84B
E5D63938 1E91DEB4 5CFE778F 637C1001

h = 04

sect571r1

$f(x)$	=	$x^{571} + x^{10} + x^5 + x^2 + 1$
a	=	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001
b	=	02F40E7E 2221F295 DE297117 B7F3D62F 5C6A97FF CB8CEFF1 CD6BA8CE 4A9A18AD 84FFABBD 8EFA5933 2BE7AD67 56A66E29 4AFD185A 78FF12AA 520E4DE7 39BACA0C 7FFEFF7F 2955727A
S	=	2AA058F7 3A0E33AB 486B0F61 0410C53A 7F132310
G_c	=	03 0303001D 34B85629 6C16C0D4 0D3CD775 0A93D1D2 955FA80A A5F40FC8 DB7B2ABD BDE53950 F4C0D293 CDD711A3 5B67FB14 99AE6003 8614F139 4ABFA3B4 C850D927 E1E7769C 8EEC2D19
G_u	=	04 0303001D 34B85629 6C16C0D4 0D3CD775 0A93D1D2 955FA80A A5F40FC8 DB7B2ABD BDE53950 F4C0D293 CDD711A3 5B67FB14 99AE6003 8614F139 4ABFA3B4 C850D927 E1E7769C 8EEC2D19 037BF273 42DA639B 6DCCFFFE B73D69D7 8C6C27A6 009CBBCA 1980F853 3921E8A6 84423E43 BAB08A57 6291AF8F 461BB2A8 B3531D2F 0485C19B 16E2F151 6E23DD3C 1A4827AF 1B8AC15B
n	=	03FFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF E661CE18 FF559873 08059B18 6823851E C7DD9CA1 161DE93D 5174D66E 8382E9BB 2FE84E47
h	=	02

A.3 ASN.1 Syntax

Elliptische Kurven über $\mathbb{F}_p$

```
secp112r1 OBJECT IDENTIFIER ::= { ellipticCurve 6 }
secp112r2 OBJECT IDENTIFIER ::= { ellipticCurve 7 }
secp128r1 OBJECT IDENTIFIER ::= { ellipticCurve 28 }
secp128r2 OBJECT IDENTIFIER ::= { ellipticCurve 29 }
secp160k1 OBJECT IDENTIFIER ::= { ellipticCurve 9 }
secp160r1 OBJECT IDENTIFIER ::= { ellipticCurve 8 }
secp160r2 OBJECT IDENTIFIER ::= { ellipticCurve 30 }
secp192k1 OBJECT IDENTIFIER ::= { ellipticCurve 31 }
secp192r1 OBJECT IDENTIFIER ::= {
  ansi-X9-62 curves(3) prime(1) 1
}
secp224k1 OBJECT IDENTIFIER ::= { ellipticCurve 32 }
secp224r1 OBJECT IDENTIFIER ::= { ellipticCurve 33 }
secp256k1 OBJECT IDENTIFIER ::= { ellipticCurve 10 }
secp256r1 OBJECT IDENTIFIER ::= {
  ansi-X9-62 curves(3) prime(1) 7
}
secp384r1 OBJECT IDENTIFIER ::= { ellipticCurve 34 }
secp521r1 OBJECT IDENTIFIER ::= { ellipticCurve 35 }
```

Elliptische Kurven über $\mathbb{F}_{2^m}$

```
sect113r1 OBJECT IDENTIFIER ::= { ellipticCurve 4 }
sect113r2 OBJECT IDENTIFIER ::= { ellipticCurve 5 }
sect131r1 OBJECT IDENTIFIER ::= { ellipticCurve 22 }
sect131r2 OBJECT IDENTIFIER ::= { ellipticCurve 23 }
sect163k1 OBJECT IDENTIFIER ::= { ellipticCurve 1 }
sect163r1 OBJECT IDENTIFIER ::= { ellipticCurve 2 }
sect163r2 OBJECT IDENTIFIER ::= { ellipticCurve 15 }
sect193r1 OBJECT IDENTIFIER ::= { ellipticCurve 24 }
sect193r2 OBJECT IDENTIFIER ::= { ellipticCurve 25 }
sect233k1 OBJECT IDENTIFIER ::= { ellipticCurve 26 }
sect233r1 OBJECT IDENTIFIER ::= { ellipticCurve 27 }
sect239k1 OBJECT IDENTIFIER ::= { ellipticCurve 3 }
sect283k1 OBJECT IDENTIFIER ::= { ellipticCurve 16 }
sect283r1 OBJECT IDENTIFIER ::= { ellipticCurve 17 }
sect409k1 OBJECT IDENTIFIER ::= { ellipticCurve 36 }
sect409r1 OBJECT IDENTIFIER ::= { ellipticCurve 37 }
sect571k1 OBJECT IDENTIFIER ::= { ellipticCurve 38 }
sect571r1 OBJECT IDENTIFIER ::= { ellipticCurve 39 }
```

Literatur

[ABN02] Abdalla, M.; An, J.; Bellare, M.; Namprempre, C.: *From Identification to Signatures via the Fiat-Shamir Transform: Minimizing Assumptions for Security and Forward-Security* Extended abstract in Advances in Cryptology - EUROCRYPT 2002 Proceedings, LNCS Vol. 2332, Springer, 2002 vollständige Version: http://www.cs.ucsd.edu/users/mihir/papers/id-sig.html 55

[AES01] Draft FIPS *Advanced Encryption Standard*, National Institute for Standards and Technology, 28. Februar 2001 http://csrc.nist.gov/encryption/aes/ 6

[BaP97] Baric, Nico; Pfitzmann, Birgit: *Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees* LNCS, Advances in Cryptology: Proceedings of EUROCRYPT 1997 7

[BB93] Biehl, Ingrid; Buchmann, Johannes *Introduction to theoretical cryptography*, Universität des Saarlandes 1993 24

[BDL98] Boneh, Dan; DeMillo, Richard A.; Lipton, Richard J.: *On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract)*, Advances in Cryptology, Eurocrypt '97, Springer LNCS 1233, pp. 37-51, 1997

[BeR93] Bellare, Mihir; Rogway, Phillip:, *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*, First ACM Conference on Computer and Communication Security Proceedings, November 1993 9

[BeR96] Bellare, Mihir; Rogway, Phillip: *The Exact Security of Digital Signatures-How to Sign with RSA and Rabin*, Advances in Cryptology - Eurocrypt '96 Proceesings, LNCS Vol 1070, U Maurer ed., Springer-Verlag 1996 9

[BHP00] Bassham, L.; Housley, R.; Polk, W.: *Representation of Public Keys and Digital Signatures in Internet X.509 Public Key Infrastructure Certificates* PKIX Working Group Internet Draft, 14. Juli 2000 81

[Bie98] Biehl, Ingried: *Kryptographie II*, Vorlesungsskript, Technische Universität Darmstadt 1998 24

[Ble96] Bleichenbacher, Daniel: *Generating ElGamal signatures without knowing the secret key*, Advances in Cryptology - Eurocrypt '96 Proceesings, LNCS Vol 1070, U Maurer ed., Springer-Verlag 1996 7

[Bro00] Brown, Daniel R.L.: *The exact security of ECDSA*, IEEE P1363: Research Contributions, 16.01.2001, CACR Technical Report 2000-54, University of Waterloo, 2000 8

[BSP95] Bakhtiari, S.; Safavi-Naini, R.; Pieprzyk, J.: *Cryptographic Hash Functions: A Survey* Technical Report 95-09, Department of Computer Science, University of Wollongong, July 1995, http://citeseer.nj.nec.com/bakhtiari95cryptographic.html 5

[BSS99] Blake, Ian; Seroussi, Gadiel; Smart, Nigel:, *Elliptic Curves in Cryptography*, London Mathematical Society, Lecture Note Series 265, Cambridge University Press 1999-2000 31, 34, 35, 39, 69

[Buc00] Buchmann, Johannes: *Einführung in die Kryptographie*, Springer, 2000 7, 31, 32, 39

[Can00] Canetti, Ran: *Towards realizing random oracles: Hash functions that hide all partial information*, Proceedings of Crypto '97, pp. 455-469, Springer Verlag, Updated version, 10.03.2000 55

[CGH98] Canetti, Ran; Goldreich, Oded; Halevi, Shai: *The Random Oracle Methodology, Revisited*, Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, 1998 9

[CMR98] Canetti, Ran; Micciancio, Daniele; Reingold, Omer: *Perfectly One-Way Probabilistic Hash Functions*, ACM Symposium on Theory of Computing 1998, pp. 131-140

[CrS99] Cramer, Ronald; Shoup, Victor: *Signature Schemes Based on the Strong RSA Assumption* ACM Conference on Computer and Communications Security, pp. 46-51, 1999 7

[dBB93] den Boer, Bert; Bosselaers, Antoon: *Collisions for the Compression Function of MD5* Advances in Cryptology: Proceedings of EUROCRYPT 1993, Springer LNCS 5

[DBP96] Dobbertin, H.; Bosselaers, A.; Preneel, B.; *RIPEMD-160, a strengthened version of RIPEMD* Fast Software Encryption, LNCS 1039, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 71-82. 5

[DES99] FIPS PUB 46-3, *Data Encryption Standard (DES)*, National Institute for Standards and Technology, 25. Oktober 1999 7

[DH76] Diffie, Whitfield; Hellman, Martin E.: *New Directions in Cryptography*, IEEE Transactions on Information Theory Vol. 22, No. 6, pp. 644-654, 1976 3

[Dob96] Dobbertin, Hans: *Cryptanalysis of MD5 compress* Announcement on Internet, May 1996.", http://citeseer.nj.nec.com/68442.html 5

[Dob98] Dobbertin, Hans: *Cryptanalysis of MD4* Journal of Cryptology Vol. 11, No. 4, pp. 273-287, Springer Verlag 1998 5

[DRj01] Daemon, Joan; Rijmen, Vincent:
The Rijndael Block Cipher AES Proposal,
http://www.esat.kuleuven.ac.be/~rijmen/rijndael/ 6

[DSS91] National Institute for Standards and Technology: *A PROPOSED DIGITAL SIGNATURE STANDARD (DSS)* FIPS Pub XX, August 1991 57

[DuN00] Durfee, Glenn; Nguyen, Phong: *Cryptanalysis of the RSA Schemes with Short Secret Exponent from Asiacrypt '99,* pp 14-29, ASIACRYPT 2000 Proceedings

[DwN94] Dwork, Cynthia; Naor, Moni: *An Efficient Existentially Unforgeable Signature Scheme and its Applications*, Proceedings of Crypto '94, pp. 234-246, Springer Verlag 7

[ElG85] ElGamal, Taher: *A public key cryptosystem and a signature scheme based on discrete logarithms* Advances in Cryptology: Proceedings of CRYPTO '84, volume 196 of Lecture Notes in Computer Science, pp. 10-18, Springer-Verlag, 1985 4

[FFS86] Feige, Uriel; Fiat, Amos; Shamir, Adi: *Zero Knowledge Proofs of Identity*, 19th ACM Symposium on Computing Theory 1987, Proceedings pp. 210-217 20, 21, 22

[Fis00] sc Fischlin, M.:*A Note on Security Proofs in the Generic Model* Advances in Cryptology - Asiacrypt 2000, Lecture Notes in Computer Science, Vol.1976, pp.458-469, Springer-Verlag, 2000. 8

[FMS01] Fluher, Scott; Mantim, Itsik; Shamir, Adi: *Weaknesses in the Key Scheduling Algorithm of RC4*, Preprint, 2001 7

[GaS99] Galbraith, ; Smart, N.P.: *A Cryptographic Application of Weil Descent*, IMA Conference on Cryptography and Coding, LNCS 1999, (zuvor: Cryptography and Coding II, Edited by Chris Mitchell, Clarendon Press, 1992) 39

[GHR99] Gennaro, Rosario; Halevi, Shai; Rabin, Tal: *Secure Hash-and-Sign Signatures without the Random Oracle* Theory and Application of Cryptographic Techniques, pp. 123-139, 1999 7

[GHS00] Gaudry, P.; Hess, F.; Smart, N.:, *Constructive and destructive facets of Weil descent on elliptic curves*, Preprint, 2000 39

[GHS00] Menezes, A.; Qu, M.: *Analysis of the Weil descent attack of Gaudry, Hess and Smart*, Topics in Cryptology - CT-RSA 2001, Springer LNCS, pp.308-318, 2001 39

[GLV98] Gallant, R.; Lambert R.; Vanstone S.: *Improving the parallelized Pollard λ search on binary anomalous curves*, Mathematics of Computation, 69 (2000), 1699-1705. 15 39

[GMR85] Goldwasser, Shafi; Micalio, Silvio; Rackoff, Charles: *The Knowledge Complexity of interactive Proofsystems*, 17th ACM Symposium on Computing Theory 1985, Proceedings pp. 291-304 18

[Gol99] Goldreich, Oded; *Introduction to Complexity Theory - Lecture Notes*, Weizmann Institute of Science, Israel 1999 24

[Ham00] Hamdy, Safuat: *Anwendung elliptischer Kurven in der Krytologie*, http://www.informatik.tu-darmstadt.de/TI/Mitarbeiter/hamdy/-deutsch/safuat.html 31

[HPN94] Horster, Patrick; Petersen, Holger; Michels, Markus: *Meta-ElGamal signature schemes*, 2nd ACM Conference on Computer and Communication Security 1994, Proceedings pp.96-107 43, 54

[ISO98] *ISO/IEC 10118-3:1998* Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions, International Organization for Standardization, 28.05.1998 5

[Jac00] Jacobson, Michael J.; Koblitz, Neal; Silverman, Joseph H.; Stein, Andreas; Teske, Edlyn: *Analysis of the Xedni Calculus Attack*, Designs, Codes and Cryptography, 2000, Vol.20, No.1, pp.1-64, 39

[JMV99] Johnson, Don; Menezes, Alfred; Vanstone, Scott: *The Elliptic Curve Digital Signature Algorithm (ECDSA)*, Univ. of Waterloo, 1999, http://cacr.math.waterloo.ca 8, 58

[KJJ99] Kocher, Paul C.; Jaffe, Joshua; Jun, Benjamin: *Differential Power Analysis*, Proceedings of Crypto '99, pp. 388-397, Springer Verlag

[Kob87] Koblitz, Neal: *Elliptic Curve Cryptosystems*, Mathematics of Computation Vol. 48, No. 177, pp. 203-209, 1987

[Koc96] Kocher, Paul C.: *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*, Proceedings of Crypto '96, pp. 104-113, Springer Verlag

[KSWH8] Kelsey, J.; Schneier, B.; Wagner, D.; Hall, C.: *Cryptanalytic Attacks on Pseudorandom Number Generators*, Lecture Notes in Computer Science, Vol. 1372, pp. 168-188, Springer Verlag 1998

[Lem99] Lemmermeyer, Franz: *Elliptische Kurven I*, Vorlesungs Skript, Universitäat Saarbrücken 1999, http://www.rzuser.uni-heidelberg.de/~hb3/qzk.html 26, 27, 31

[LoR00] Lopez, Julio; Dahab, Ricardo: *An Overview of Elliptic Curve Cryptography*, Preprint, 2000

[Mil85] Miller, Victor: *Use of Elliptic Curves in Cryptography*, Advances in Cryptology: Proceedings of Crypto '85, pp. 417-426, Springer-Verlag

[MiR98] Micali, Silvio; Reyzin, Leonid: *Signing with Partially Adversarial Hashing* MIT technical report MIT/LCS/TM-575, 1998. http://www.lcs.mit.edu/publications/viewpubs.php?cat=tm 56

[Moe01] Möller, Bodo: *Algorithms for Multi-exponentiation.* Selected Areas in Cryptography 2001, Springer LNCS 2259, pp. 165-180.

[MOV91] Menezes, Alfred; Vanstone, Scott; Okamoto, Tatsuaki: *Reducing elliptic curve logarithms to logarithms in a finite field* Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, pp. 80-89, New Orleans, Louisiana, 6-8 May 1991 77 40

[MOV97] Menezes, Alfred J.; Van Oorshot, Paul C.; Vanstone, Scott A.: *Handbook of Applied Cryptography*, CRC Press LLC, 1997 7, 23, 39

[NgS00] Nguyen, Phong; Stern, Jacques: *Lattice Reduction in Cryptology: An Update*, Proceedings of the 4th Algorithmic Number Theory Symposium, ANTS-IV, pp. 85-112, 2000

[NSh00] Nguyen, P.; Shparlinski, I. E.: *The insecurity of the Digital Signature Algorithm with partially known nonces* Preprint, 2000 http://citeseer.nj.nec.com/nguyen00insecurity.html 7

[NyS00] Nguyen, P.; Shparlinski, I.: *The insecurity of the elliptic curve Digital Signature Algorithm with partially known nonces* Preprint, 2000 60

[OhO98] Ohta, Kazuo; Okamoto, Tatsuaki: *On Concrete Security Treatment of Signatures Derived from Identification*, Advances in Cryptology - CRYPTO'98, Proceedings, p. 354 ff. Springer Verlag

[P1363] IEEE P1363: *Standard Specifications for Public Key Cryptography* 8

[PoS00] Pointcheval, David; Stern, Jaques: *Security Arguments for Digital Signatures and Blind Signatures*, Journal of Cryptology, Vol. 13, No. 3, PP. 361-396, Springer Verlag 2000 47, 48, 50

[Ros99] Rosing, Michael: *Implementing Elliptic Curve Cryptography*, Manning Publications, 1999 31, 34, 77

[RSA78] Rivest, Ron; Shamir, Adi; Adleman, Leonard M.: *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, Communications of the ACM, Vol. 21 No. 2, pp. 120-126, 1978 3

[Sch91] Schnorr, C.P.: *Efficient Signature Generation by Smart Cards* Journal of Cryptology, Vol. 4, No. 3, (1991), pp. 161–174 57

[ScJ98] Schnorr, Claus Peter; Jakobson, Markus: *Security of Discrete Log Cryptosystems in the Ramdom Oracle + Generic Model* Conference on the Mathematics of Public-Key Cryptography, The Fields Institute, Toronto, Canada 7

[SEC1] SEC1: Elliptic Curve Cryptography
Certicom Research, September 20, 2000 68

[SEC2] SEC2: Recommended Elliptic Curve Domain Parameters
Certicom Research, September 20, 2000 69, 70, 80

[SHA1] *FIPS 180-1, Secure Hash Standard (SHS)*, National Institute for Standards and Technology, 1995
http://csrc.nist.gov/cryptval/shs.html 5

[SHA2] *Draft-FIPS 180-2, Secure Hash Standard (SHS)*, National Institute for Standards and Technology, Mai 2001
http://csrc.nist.gov/encryption/tkhash.html 5, 57

[Sha49] Shannon, Claude E.:, *Communication Theorey of Secrecy Systems*, Bell System Technical Journal, pp. 656-715, 1949.

[Sho97] Shoup, Victor: *Lower bounds for discrete logarithms and related problems*, Advances in Cryptology - EUROCRYPT '97 Proceedings, Lecture Notes in Computer Science, Springer-Verlag, 1997, pp. 256-266. 7

[Sho98] Shoup, Victor: *Why Chosen Ciphertext Security Matters*, Research Report, IBM Research Division, RZ 3076, 23//11/1998

[Sil00] Joseph H. Silverman: *The Xedni Calculus and the Elliptic Curve Discrete Logarithm Problem*, Designs, Codes and Cryptography, 2000, Vol.20, No.1, pp.5-40 46

[Sil85] Silverman, Joseph H.: *The Arithmetic of Elliptic Curves*, Graduate Texts in Mathematics, Band 106, Springer 1985-92 25, 27, 31

[Sma01] Smart, N. P: *The Exact Security of ECIES in the Generic Group Model* Cryptography and Coding 2001, Springer LNCS 2260, pp.73-84 77

[Sma99] Smart, N. P.: *The Discrete Logarithm Problem on Elliptic Curves of Trace One*, Journal of Cryptology 1999, Vol 12 No.3, pp. 193-196 40

[SZSH98] Shikata, Junji; Zheng, Yuliang; Suzuki, Joe; Imai, Hideki: *Realizing the Menezes-Okamoto-Vanstone (MOV) Reduction Efficiently for Ordinary Elliptic Curves*, IEICE Transactions on Fundamentals, Vol. E83-A, No.4, pp.756-763, April 2000 39, 40

[Tes98] Teske, Edlyn: *Better Random Walks For Pollard's Rho Method*, Preprint, 1998 http://citeseer.nj.nec.com/teske98better.html 38

[Teu96] Teubner Taschenbuch der Mathematik, B.G. Teubner, 1996 25, 26, 27

[Vau96] Vaudenay, Serge: *Hidden Collisions on DSS* Advances in Cryptology - CRYPTO'96, LNCS 1109, Springer-Verlag, 1996, pp.83-88. 7

[Vau98] Vaudenay, Serge: *Provable Security for block Ciphers by decorrelation*, Symposium on Theoretical Aspects of Computer Science, 1998, pp. 249-275 7

[vOW94] Van Oorschot, Paul C.; Wiener, Michael J.:, *Parallel Collision Search with Application to Hash Functions and Discrete Logarithms*, Proceedings of 2nd ACM Conference on Computer and Communication Security, 1994. 8, 38, 39

[WiZ98] Wiener, Michael J.; Zuccherato, Robert J.:, *Faster Attacks on Elliptic Curve Cryptosystems*, Selected Areas in Cryptography, 1998, pp.190-200 8

[X.68?] International Telecommunication Union: *Abstract syntax notation one (ASN.1): Specification of basic notation* Recommendation X.680, Information object specification (Recommendation X.681), Constraint specification (X.682), Parameterization of ASN.1 (X.683), Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Dec. 1997, http://www.itu.int/ITU-T/studygroups/com17/languages/ 62

[X.69?] International Telecommunication Union: *ASN.1 encoding rules:*Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER) (X.690), Specification of Packed Encoding Rules (PER) (X.691) 62

[X9.62] American National Standard X9.62, *Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)* 8, 40