Über die Sicherheit und Effizienz kryptographischer Verfahren mit Klassengruppen imaginär-quadratischer Zahlkörper

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

Dissertation

zur Erlangung des akademischen Grades
Doctor rerum naturalium (Dr. rer. nat.)

von

Dipl.-Inform. Safuat Hamdy

aus Hamburg

Gutachter:	Prof. Dr. Johannes Buchmann
	Prof. Renate Scheidler, PhD
Tag der Einreichung:	20. Februar 2002
Tag der mündlichen Prüfung:	20. März 2002

Darmstadt, 2002
D 17

Über die Sicherheit und Effizienz kryptographischer Verfahren mit Klassengruppen imaginärquadratischer Zahlkörper

Dipl.-Inform. Safuat Hamdy

Technische Universität Darmstadt
Fachbereich Informatik
Institut für Theoretische Informatik
Alexanderstraße 10
D–64283 Darmstadt

Mathematics Subject Classification (2000): 11–04, 11R29, 11T71, 11Y40, 68P25, 94A60
ACM Computing Classification System (1998): E.3, F.2.1

Für Marion

Danksagung

An dieser Stelle möchte ich mich zunächst sehr herzlich bei meiner Frau Marion bedanken, die durch ihren unermüdlichen Einsatz dafür gesorgt hat, daß ich immer ausreichend Zeit für meine Dissertation hatte. Sie hat mich immer unterstützt und mußte mich besonders in der Endphase hin und wieder an Wochenenden oder Abenden entbehren.

Desweiteren danke ich herzlich Herrn Prof. Johannes Buchmann, der mir meine Dissertation erst ermöglicht hat, und von dem ich in meiner Darmstädter Zeit sehr viel gelernt habe. Ich bedanke mich auch bei Frau Prof. Renate Scheidler von der University of Calgary, die sich trotz eines vollen Terminkalenders die Zeit für eine genaue Begutachtung meiner Dissertation genommen hat. Der DFG danke ich für die finanzielle Unterstützung des Forschungsprojektes Zahlkörperkryptographie, in dem meine Arbeit angesiedelt ist.

Ich danke auch Ingrid Biehl, Andreas Meyer, Bodo Möller, Tsuyoshi Takagi und Ulrich Vollmer für unsere Zusammenarbeit an verschiedenen Projekten und Papieren und für die Diskussionen und Anregungen, und ich danke meinen Korrekturlesern und Gutachtern für die Kommentare, mit deren Hilfe ich meine Dissertation abrunden konnte. Besonderer Dank geht auch an Ralf-Philip Weinmann, der die IQ-Kryptoverfahren in FlexiPKI implementiert hat.

Zusammenfassung

In dieser Arbeit beschreiben wir asymmetrische Kryptoverfahren, deren Sicherheit auf der Schwierigkeit des DL-, des DH- oder des Wurzelproblems in Klassengruppen imaginär-quadratischer Zahlkörper beruht; diese Kryptoverfahren werden im folgenden als IQ-Kryptoverfahren oder einfach nur IQ-Verfahren bezeichnet.

Bislang sind über IQ-Kryptographie nur vereinzelte Arbeiten erschienen, in denen einzelne Aspekte der IQ-Kryptographie beschrieben werden. So wurden z. B. in [14] Klassengruppen für den Diffie-Hellman-Schlüsselaustausch vorgeschlagen und eine erste Analyse zur Schwierigkeit des DL-Problems in Klassengruppen imaginär-quadratischer Zahlkörper gegeben, und in [31], [25] und [40] wurden subexponentielle Methoden zur Berechnung diskreter Logarithmen in Klassengruppen vorgestellt.

Die folgenden Fragestellungen waren aber noch offen geblieben: Es war unklar, welche Signaturverfahren für Klassengruppen verwendet werden sollen, denn die Ordung der Klassengruppe oder eines Teilers davon (außer Potenzen von 2) ist i. allg. nicht effizient berechenbar. Daher können Signaturen vom ElGamal-Typ (z. B. DSA) nicht ohne weiteres auf Klassengruppen übertragen werden, da für Signaturen diesen Typs die Gruppenordnung bekannt sein muß. Weiterhin existierte bisher keine Untersuchung darüber, wie eine Klassengruppe ausgewählt werden sollte, so daß die Berechnung diskreter Logarithmen oder Wurzeln darin selbst mit den besten bekannten Algorithmen hierzu nicht effizient möglich ist. Es existierten bisher auch keine Implementierungen von IQ-Kryptoverfahren, weder experimentelle Implementierungen, noch Implementierungen für den praktischen Gebrauch. Letztendlich existierten daher bisher auch keine Untersuchungen über die Effizienz von IQ-Kryptoverfahren.

Mit dieser Arbeit soll diese Lücke geschlossen werden: Wir stellen eine Reihe von IQ-Kryptoverfahren zur Signatur, zur Verschlüsselung und zum Schlüsselaustausch vor, und wir beschreiben detailliert, wie diese Kryptoverfahren implementiert werden sollten. Wir gehen dabei in einer Weise vor, wie es sich bei diversen Standards für Public-Key-Kryptographie eingebürgert hat, z. B. ANSI X9.62 [2], ANSI X9.63 [3], IEEE P1363 [39] oder SEC [71, 72]; wir haben uns für unsere Spezifikation konkret den Standard SEC als Vorbild genommen.

Wir zeigen, daß die IQ-Kryptoverfahren sicher sind unter der Annahme, daß die Berechnung diskreter Logarithmen und Wurzeln in Klassengruppen nicht effizient möglich ist. Wir zeigen dazu, wie die Diskriminante ausgewählt werden soll, damit diese Annahme mit sehr hoher Wahrscheinlichkeit erfüllt ist, und wir zeigen darüberhinaus, wie eine Diskriminante gewählt werden soll, so daß etwa die Berechnung diskreter Logarithmen in der entsprechenden Klassengruppe so aufwendig ist wie andere bekannte Berechnungsprobleme mit bestimmten Parametern (z. B. Faktorisierung einer ganzen Zahl mit 1024 Bit).

Schließlich untersuchen wir die Effizienz der IQ-Kryptoverfahren und der darunter liegenden Arithmetik für Klassengruppen zu Diskriminanten, die für mittelfristigen Gebrauch kryptographisch geeignet sind, und wir zeigen, daß IQ-Kryptoverfahren so effizient sind, daß sie mit „traditionellen“ Kryptoverfahren wie RSA oder DSA konkurrieren. Wir weisen damit nach, daß IQ-Kryptoverfahren eine sinnvolle und taugliche Alternative in der Familie der asymmetrischen Kryptoverfahren sind.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Aufbau der Arbeit 3

1.2 Notation 6

I Implementierung IQ-basierter Kryptoverfahren **9**

2 Klassengruppen **11**

2.1 Imaginär-quadratische Körper, Ordnungen und Ideale 11

2.2 Klassengruppen 14

3 Kryptographische Komponenten **19**

3.1 Datenformate und Konvertierung 20

3.1.1 Konvertierung von reduzierten Idealen zu Octet-Strings 21

3.1.2 Konvertierung von Octet-Strings zu reduzierten Idealen 23

3.2 Die IQ-Bereichsparameter für Fundamental-Diskriminanten 23

Verfahren zur Erzeugung einer zufälligen Fundamental-Diskriminante 24

Verfahren zur Überprüfung einer Fundamental-Diskriminante 25

Verfahren zur Erzeugung eines zufälligen Gruppenelements 27

Verfahren zur Überprüfung eines Gruppenelements 27

3.2.1 IQ-Bereichsparameter für IQ-DH, IQ-IES und IQ-DSA 28

Erzeugung der Bereichsparameter für IQ-DH, IQ-DSA und IQ-IES . . 28

Überprüfung der Bereichsparameter für IQ-DH, IQ-DSA und IQ-IES . 29

3.2.2 IQ-Bereichsparameter für IQ-RDSA 29

Erzeugung der IQ-RDSA Bereichsparameter 29

Verfahren zur Überprüfung der IQ-RDSA Bereichsparameter 30

3.2.3 IQ-Bereichsparameter für IQ-GQ 30

Erzeugung der IQ-GQ Bereichsparameter 30

Verfahren zur Überprüfung der IQ-GQ Bereichsparameter 30

3.3 IQ-Schlüsselpaare 31

3.3.1 Schlüsselpaare für IQ-DH, IQ-DSA, IQ-RDSA und IQ-IES 31
Verfahren zur Erzeugung von Schlüsselpaaren für IQ-DH, IQ-DSA, IQ-RDSA und IQ-IES 31
Verfahren zur Überprüfung von öffentlichen Schlüsseln für IQ-DH, IQ-DSA, IQ-RDSA und IQ-IES 32
3.3.2 Schlüsselpaare für IQ-GQ 32
Verfahren zur Erzeugung von IQ-GQ Schlüsselpaaren 32
Verfahren zur Überprüfung von öffentlichen IQ-GQ-Schlüsseln 33
3.4 Das IQ-Diffie-Hellman-Primitiv 33
3.5 Weitere Komponenten 33
3.5.1 Kryptographische Hash-Funktionen 34
3.5.2 Schlüsselableitung 34
3.5.3 Message-Authentication-Codes 35
3.5.4 Symmetrische Verschlüsselungen 36

4 IQ-Kryptoverfahren 37
4.1 Signaturverfahren 38
4.1.1 IQ-RDSA 38
Vorbereitung 38
Signatur 39
Verifikation 39
4.1.2 IQ-DSA 40
Vorbereitung 40
Signatur 41
Verifikation 41
4.1.3 IQ-GQ 42
Vorbereitung 42
Signatur 43
Verifikation 43
4.2 Verschlüsselungsverfahren 44
4.2.1 IQ-IES 44
Vorbereitung 44
Verschlüsselung 45
Entschlüsselung 46
4.3 Schlüsselaustauschverfahren 47
4.3.1 IQ-DH 48
Vorbereitung 48
Schlüsselaustausch 48

II Analyse IQ-basierter Kryptoverfahren 51

5 Auswahl der kryptographischen Parameter 53

5.1 Eigenschaften von Klassengruppen 54
5.1.1 Die Größe der Klassengruppe 55
5.1.2 Die Struktur der Klassengruppe 57
Der gerade Anteil der Klassengruppe 57
Der ungerade Anteil der Klassengruppe 59
5.1.3 Die Glattheitswahrscheinlichkeit von Klassenzahlen 59
5.2 Strategien zur Berechnung diskreter Logarithmen in Klassengruppen 64
5.2.1 Reduktion des IQ-DLP auf einfachere Berechnungsprobleme 64
Auswahl einer Fundamental-Diskriminante 65
5.2.2 Angriffe mit Index-Berechnungs-Algorithmen 66
5.2.3 Angriffe mit Quadratwurzel-Algorithmen 72
5.2.4 Angriffe mit dem Pohlig-Hellman-Algorithmus 73
5.3 Die Auswahl eines zufälligen, reduzierten Ideals 77
5.4 Die kryptographische Auswahl der Diskriminante 79

6 Sicherheit der IQ-Kryptoverfahren 83

6.1 Die zugrundeliegenden Berechnungsprobleme 84
6.1.1 Das Diskreter-Logarithmus-Problem 84
6.1.2 Das Ordnungsproblem 84
6.1.3 Das Wurzelproblem 84
6.1.4 Das Diffie-Hellman-Problem 84
6.1.5 Reduktionen 85
6.2 Das Sicherheits-Modell 86
6.2.1 Zufallsorakel 87
6.2.2 Signaturen 88
6.2.3 Verschlüsselungen 88
6.3 IQ-RDSA 89
6.3.1 Motivation 90
6.3.2 RDSA ist sicher 92
6.4 IQ-DSA 96
6.5 IQ-GQ 98
6.5.1 GQ ist sicher 98
6.6 IQ-IES 100
6.7 Die Verwendung kurzer Exponenten bei IQ-Kryptoverfahren 101

7 Effiziente IQ-Arithmetik **103**
7.1 IQ-Arithmetik . . . 103
7.1.1 Multiplikation von Idealen . . . 104
7.1.2 Reduktion von Idealen . . . 105
Der Standard-Algorithmus nach Gauß . . . 106
Der Algorithmus nach Rickert . . . 108
Der Algorithmus nach Schönhage . . . 114
7.1.3 Vergleich der Reduktions-Algorithmen . . . 120
7.1.4 Asymptotische Laufzeit für IQ-Arithmetik . . . 120
7.2 Laufzeiten für die IQ-Signaturverfahren . . . 121
7.2.1 Optimierungen für die IQ-Signaturverfahren . . . 126
IQ-RDSA . . . 126
IQ-DSA . . . 126
IQ-GQ . . . 126

A ASN1-Notation der IQ-Kryptoverfahren **128**

Literaturverzeichnis **133**

Index **139**

Tabellenverzeichnis

5.1 Statistik der Diskriminanten mit $|\Delta|^{1/2u}$-glatter Klassenzahl 63
5.2 Zur Überprüfung der Vermutung 5.1.12 für Klassengruppen mit Diskriminante $\Delta = -p$, $p \equiv 7 \pmod 8$ 66
5.3 Geschätzter Berechnungsaufwand für das GNFS für größere Eingaben 68
5.4 Empirischer Berechnungsaufwand des IQ-MPQS für relativ kleine Diskriminanten 69
5.5 Geschätzter erwarteter Berechnungsaufwand des IQ-MPQS für größere Diskriminanten 70
5.6 Geschätzter erwarteter Berechnungsaufwand für das GNFS und das IQ-MPQS für vergleichbare Eingaben 70
5.7 Geschätzter erwarteter Berechnungsaufwand für den λ-Algorithmus 73
5.8 Größe der Diskriminante zu gegebenem Aufwand W_{total} für verschiedene Glattheitswahrscheinlichkeiten $\Pr_{\Delta,B}$ 76
5.9 Statistik der Ordnungen der Untergruppen, die von $[\mathfrak{a}]$ erzeugt werden ($|\Delta| > 10^{48}$) 79
5.10 Statistik der Ordnungen der Untergruppen, die von $[\mathfrak{a}]$ erzeugt werden ($|\Delta| > 10^{32}$) 79
5.11 t_Δ für verschiedene Werte von t bei einer hypothetischen Laufzeit asymptotisch proportional zu $L_{|\Delta|}\left[\frac{1}{2}, 1\right]$ 81
5.12 t_Δ für verschiedene Werte von t bei einer hypothetischen Laufzeit asymptotisch proportional zu $L_{|\Delta|}\left[\frac{1}{2}, \frac{3}{4}\sqrt{2}\right]$ 82

7.1 Laufzeiten der Reduktions-Algorithmen (auf SPARC V8+) 120
7.2 Mittlere Laufzeiten für IQ-RDSA, IQ-DSA und IQ-GQ in ms 122
7.3 Mittlere Laufzeiten für DSA und RSA ($e = 65537$) in ms 123

Abbildungsverzeichnis

5.1 Empirischer Berechnungsaufwand des IQ-MPQS für relativ kleine Diskriminanten 71
5.2 Graphischer Vergleich der erwarteten Laufzeiten des IQ-MPQS und des GNFS 71

7.1 Die führenden ℓ Bits von $2a$, b und $2c$ 112
7.2 Zum Schönhage-Algorithmus 117
7.3 Vergleich der Parametergrößen von IQ-Kryptoverfahren und von „traditionellen“ Kryptoverfahren 123
7.4 Vergleich der mittleren Laufzeiten der IQ-Signaturverfahren bei Erzeugung einer Signatur 124
7.5 Vergleich der mittleren Laufzeiten der IQ-Signaturverfahren bei Verifikation einer Signatur 124
7.6 Vergleich der mittleren Laufzeiten von IQ-RDSA mit RSA und DSA bei Erzeugung einer Signatur 125
7.7 Vergleich der mittleren Laufzeiten von IQ-RDSA mir RSA und DSA bei Verifikation einer Signatur 125

Algorithmenverzeichnis

2.1 isNormal . . . 12
2.2 normalize . . . 12
2.3 isReduced . . . 13
2.4 primeIdeal . . . 14
2.5 ressol . . . 14
2.6 reduce . . . 16
2.7 isEqual . . . 17
2.8 multiply . . . 17
2.9 xgcd . . . 17
2.10 square . . . 18
2.11 inverse . . . 18
2.12 divide . . . 18
3.1 idealToOctets . . . 22
3.2 octetsToIdeal . . . 23
3.3 discSize . . . 24
3.4 genDiscP . . . 25
3.5 genDiscPQ . . . 25
3.6 checkDiscAny . . . 26
3.7 checkDiscP . . . 26
3.8 checkDiscPQ . . . 26
3.9 checkDisc . . . 27
3.10 randomIdeal . . . 27
3.11 checkIdeal . . . 28
3.12 genDomainIQDH . . . 28
3.13 checkDomainIQDH . . . 29
3.14 genDomainIQRDSA . . . 29
3.15 checkDomainIQRDSA . . . 30
3.16 genDomainIQGQ . . . 30
3.17 checkDomainIQGQ . . . 31

3.18 genKeyPairIQDH . . . 31
3.19 checkPubKeyIQDH . . . 32
3.20 genKeyPairIQGQ . . . 32
3.21 checkPubKeyIQGQ . . . 33
3.22 IQDH . . . 33
3.23 hash . . . 34
3.24 hashToInteger . . . 34
3.25 deriveKey . . . 35
3.26 MAC . . . 35
3.27 ENC . . . 36
3.28 ENC^{-1} . . . 36
4.1 IQ-RDSA Signatur . . . 39
4.2 IQ-RDSA Verifikation . . . 40
4.3 IQ-DSA Signatur . . . 41
4.4 IQ-DSA Verifikation . . . 42
4.5 IQ-GQ Signatur . . . 43
4.6 IQ-GQ Verifikation . . . 44
4.7 IQ-IES Verschlüsselung . . . 46
4.8 IQ-IES Entschlüsselung . . . 47
4.9 IQ-DH Schlüsselaustausch . . . 49
5.1 Berechne den größten Faktor von $|\langle \mathfrak{a} \rangle|$, wenn $h(\Delta)$ glatt ist . . . 74
5.2 randomIdeal2 . . . 78
5.3 discSize . . . 81
7.1 multiply . . . 104
7.2 square . . . 105
7.3 normalizeForm . . . 107
7.4 normalizeForm1 . . . 107
7.5 reduceGauss . . . 108
7.6 reduceRickertFinal . . . 110
7.7 fullStepRickert . . . 110
7.8 simpleStepRickert . . . 111
7.9 reduceRickertCore . . . 113
7.10 reduceRickert . . . 114
7.11 isMinimal . . . 115
7.12 reduceSchoenhageFinal . . . 116
7.13 fullStepSchoenhage . . . 117
7.14 reduceSchoenhageCore . . . 119

Kapitel 1

Einleitung

In dieser Arbeit beschreiben wir asymmetrische Kryptoverfahren, deren Sicherheit auf der Schwierigkeit des DL-, des DH- oder des Wurzelproblems in Klassengruppen imaginär-quadratischer Zahlkörper beruht; diese Kryptoverfahren werden im folgenden als IQ-Kryptoverfahren oder einfach nur IQ-Verfahren bezeichnet.

Bislang sind über IQ-Kryptographie nur vereinzelte Arbeiten erschienen, in denen einzelne Aspekte der IQ-Kryptographie beschrieben werden. So wurden z. B. in [14] Klassengruppen für den Diffie-Hellman-Schlüsselaustausch vorgeschlagen und eine erste Analyse zur Schwierigkeit des DL-Problems in Klassengruppen imaginär-quadratischer Zahlkörper gegeben, und in [31], [25] und [40] wurden subexponentielle Methoden zur Berechnung diskreter Logarithmen in Klassengruppen vorgestellt.

Die folgenden Fragestellungen waren aber noch offen geblieben: Es war unklar, welche Signaturverfahren für Klassengruppen verwendet werden sollen, denn die Ordung der Klassengruppe oder eines Teilers davon (außer Potenzen von 2) ist i. allg. nicht effizient berechenbar. Daher können Signaturen vom ElGamal-Typ (z. B. DSA) nicht ohne weiteres auf Klassengruppen übertragen werden, da für Signaturen diesen Typs die Gruppenordnung bekannt sein muß. Weiterhin existierte bisher keine Untersuchung darüber, wie eine Klassengruppe ausgewählt werden sollte, so daß die Berechnung diskreter Logarithmen oder Wurzeln darin selbst mit den besten bekannten Algorithmen hierzu nicht effizient möglich ist. Es existierten bisher auch keine Implementierungen von IQ-Kryptoverfahren, weder experimentelle Implementierungen, noch Implementierungen für den praktischen Gebrauch. Letztendlich existierten daher bisher auch keine Untersuchungen über die Effizienz von IQ-Kryptoverfahren.

Mit dieser Arbeit soll diese Lücke geschlossen werden: Wir stellen eine Reihe von IQ-Kryptoverfahren zur Signatur, zur Verschlüsselung und zum Schlüsselaustausch vor, und wir beschreiben detailliert, wie diese Kryptoverfahren implementiert werden sollten. Wir gehen dabei in einer Weise vor, wie es sich bei diversen Standards für Public-Key-Kryptographie eingebürgert hat, z. B. ANSI X9.62 [2], ANSI X9.63 [3], IEEE P1363 [39] oder SEC [71, 72]; wir haben uns für unsere Spezifikation konkret den Standard SEC als Vorbild genommen.

Wir zeigen, daß die IQ-Kryptoverfahren sicher sind unter der Annahme, daß die Berechnung diskreter Logarithmen und Wurzeln in Klassengruppen nicht effizient möglich ist. Wir zeigen dazu, wie die Diskriminante ausgewählt werden soll, damit diese Annahme mit sehr hoher Wahrscheinlichkeit erfüllt ist, und wir zeigen darüberhinaus, wie eine Diskriminante gewählt werden soll, so daß etwa die Berechnung diskreter Logarithmen in der entsprechenden

Klassengruppe so aufwendig ist wie andere bekannte Berechnungsprobleme mit bestimmten Parametern (z. B. Faktorisierung einer ganzen Zahl mit 1024 Bit).

Schließlich untersuchen wir die Effizienz der IQ-Kryptoverfahren und der darunter liegenden Arithmetik für Klassengruppen zu Diskriminanten, die für mittelfristigen Gebrauch kryptographisch geeignet sind, und wir zeigen, daß IQ-Kryptoverfahren so effizient sind, daß sie mit „traditionellen“ Kryptoverfahren wie RSA oder DSA konkurrieren. Wir weisen damit nach, daß IQ-Kryptoverfahren eine sinnvolle und taugliche Alternative in der Familie der asymmetrischen Kryptoverfahren sind. Zum Beweis für die Tauglichkeit der IQ-Kryptoverfahren wurden sie nach dieser Spezifikation für FlexiProvider implementiert. FlexiProvider ist eine frei erhältliche Software (`http://www.FlexiProvider.de/`), die eine Vielzahl von kryptographischen (symmetrischen sowie asymmetrischen) Verfahren als Module auf der Basis der sog. Java Cryptographic Architecture/Extension (JCA/JCE, `http://java.sun.com/products/jce/`) zur Verfügung stellt, und wird an der Arbeitsgruppe von J. Buchmann am Fachbereich Informatik der TU Darmstadt entwickelt.

Wir gehen nun kurz etwas näher ein auf die Motivation für die kryptographische Nutzung von Klassengruppen. Klassengruppen sind endliche, Abelsche Gruppen, und das DL-Problem in Klassengruppen (IQ-DLP) ist (mutmaßlich) ein schwieriges Berechnungsproblem und bietet somit eine Basis für ein kryptographisches Verfahren. Es bietet sich auf der Suche nach alternativen Public-Key-Verfahren daher an, Klassengruppen für DL-basierte Kryptoverfahren zu verwenden, z. B. Signatur- und Verschlüsselungsverfahren vom ElGamal-Typ. Ein weiteres, schwieriges Berechnungsproblem, das sich als kryptographisch nutzbar erweist, ist das Problem, Wurzeln in einer beliebigen Klassengruppe zu berechnen (Wurzelproblem, IQ-RP).

Die Idee, Klassengruppen imaginär-quadratischer Zahlkörper kryptographisch zu nutzen, ist zum ersten mal von Buchmann und Williams in [14] vorgeschlagen worden. Dort wurde der einfache Diffie-Hellman-Schlüsselaustausch schematisch beschrieben und eine erste Analyse zur Schwierigkeit des DL-Problems in Klassengruppen imaginär-quadratischer Zahlkörper gegeben. Die Idee für IQ-basierte Kryptographie ist etwa so alt wie die Idee EC[1]-basierter Kryptographie [52], [44], wurde aber nicht weiterverfolgt. Erst in [32] wurde systematisch die Parameterauswahl untersucht, und erst in [13] wurde eine schematische Übersicht über diverse IQ-basierte Kryptoverfahren gegeben. Mit dieser Arbeit wollen wir u. a. die Standardisierung IQ-basierter Kryptoverfahren voranbringen.

Einer der Gründe für das Desinteresse an IQ-basierter Kryptographie war sicherlich, daß traditionelle Kryptoverfahren, wie etwa RSA, auf einfach zu verstehenden Ideen beruhen, während IQ-Kryptoverfahren auf vergleichsweise komplexen Konzepten aufbaut. Im Vergleich mit EC-Kryptoverfahren kommt noch hinzu, daß Hafner und McCurley in [31] einen subexponentiellen Algorithmus zur Berechnung von diskreten Logarithmen in Klassengruppen imaginär-quadratischer Zahlkörper vorgestellt haben. Dieser Algorithmus wurde von Düllmann [25] und Jacobson [40] immer weiter verbessert. Ein entsprechender subexponentieller Algorithmus wurde für elliptische Kurven über endlichen Körpern bislang nicht gefunden. Das hat zur Folge, daß – bei vergleichbarer Sicherheit – Schlüssellängen bei EC-basierten Kryptoverfahren kleiner gewählt werden können, als bei IQ-basierten oder traditionellen Kryptoverfahren (z. B. RSA, Rabin, DSA usw.).

Andererseits ist der beste bekannte Algorithmus zur Berechnung von diskreten Logarithmen in Klassengruppen asymptotisch viel langsamer, als der beste bekannte Algorithmus zur

[1] EC: Elliptic Curve

Berechnung diskreter Logarithmen in multiplikativen Gruppen endlicher Körper oder der beste bekannte Algorithmus zur Faktorisierung ganzer Zahlen. Daher können – bei vergleichbarer Sicherheit – die Schlüssellängen bei IQ-basierten Kryptoverfahren kleiner gewählt werden, als bei den entsprechenden traditionellen Kryptoverfahren.

Die Länge der Schlüssel bestimmt die Länge der Operanden, mit denen bei einem Kryptoverfahren Arithmetik betrieben wird, und hat somit Einfluß auf die Effizienz des entsprechenden Kryptoverfahrens. Andererseits wird die Schlüssellänge durch die Anforderung an die Sicherheit bestimmt. Läßt man die Sicherheitsanforderung steigen, so folgt z. B., daß die Schlüssellänge bei traditionellen Kryptoverfahren schneller wächst, als bei IQ-basierten Kryptoverfahren, deren Schlüssellänge allerdings schneller wächst, als bei EC-basierten Kryptoverfahren.

IQ-basierte Kryptoverfahren bieten dennoch eine sinnvolle Alternative zu traditionellen, und auch zu EC-basierten Kryptoverfahren, denn bei mittelfristig sinnvollen Sicherheitsanforderungen (d. h. Schlüssellängen, vgl. [47]) sind IQ-basierte Kryptoverfahren effizient genug, um mit traditionellen und EC-basierten Kryptoverfahren konkurrieren zu können.

Besonders hervorzuheben ist auch, daß keine effizient nutzbaren Beziehungen zwischen den IQ-basierten Berechnungsproblemen und anderen Berechnungsproblemen bekannt sind, außer daß das Faktorisierungsproblem bewiesenermaßen eine komplexitätstheoretische untere Schranke für die Berechnung diskreter Logarithmen in Klassengruppen ist. Die IQ-basierten Berechnungsprobleme sind daher, bei geeigneter Wahl der Parameter, (mutmaßlich) unabhängig von anderen Berechnungsproblemen. Ein Fortschritt oder ein Durchbruch etwa bei dem Faktorisierungsproblem berührt zunächst nicht die Sicherheit IQ-basierter Kryptoverfahren. IQ-basierte Kryptoverfahren stellen somit nicht nur einfach eine weitere Alternative, sondern eine wertvolle Bereicherung für die Familie aller Public-Key-Verfahren dar.

Schließlich wollen wir noch anmerken, daß IQ-basierte Kryptoverfahren nicht durch Patente irgendwelcher Art „belastet“ sind, anders als bei EC-basierter Kryptographie. Alles, was in dieser Arbeit vorgestellt wird, ist nach unserem besten Wissen und Gewissen frei verfügbar.

1.1 Aufbau der Arbeit

Die Kapitel 2, 3 und 4 bilden den ersten Teil dieser Arbeit, der sich mit den praktischen Aspekten der IQ-basierten Kryptoverfahren befaßt. Hier legen wir dar, wie Arithmetik in Klassengruppen imaginär-quadratischer Zahlkörper effizient betrieben werden kann; wir beschreiben, wie die kryptographischen Parameter ausgewählt werden müssen, um vom Standpunkt der zugrundeliegenden Berechnungsprobleme ein bestimmtes Maß an Sicherheit zu bekommen; wir geben an, in welchem Datenformat Elemente der Klassengruppe abgespeichert und/oder übermittelt werden sollen; wir geben an, wie Schlüsselpaare ausgewählt, und wie öffentliche Schlüssel überprüft werden; schließlich geben wir an, wie die IQ-Kryptoverfahren verwendet werden sollen. Zum Verständnis und Implementierung eines der beschriebenen IQ-basierten Kryptoverfahren enthält dieser Teil, zusammen mit Anhang A, alles Notwendige. In diesem Teil stellen wir dar, *wie* man IQ-Kryptographie betreibt.

Die Kapitel 5, 6 und 7 bilden den zweiten Teil, in dem die theoretischen Hintergründe zur IQ-Kryptographie behandelt werden. In diesem Teil erklären wir, *warum* bestimmte Entscheidungen getroffen wurden. Wir begründen hier die Parameterauswahl; wir stellen dar, welche

Berechnungsprobleme den IQ-basierten Kryptoverfahren zugrundeliegen, und wir zeigen, daß unsere IQ-Kryptoverfahren mit der vorgeschlagenen Auswahl der Parameter sicher sind. Wir zeigen, daß die IQ-Kryptoverfahren effizient sind, und wir geben Möglichkeiten an, Arithmetik in Klassengruppen noch effizienter zu machen.

In Kapitel 2 definieren wir die mathematischen Begriffe und zitieren einige Fakten, die für das Verständnis und die Implementation der IQ-Kryptoverfahren notwendig sind. Wir definieren imaginär-quadratische Ordnungen, Ideale aus solchen Ordnungen, sowie eine Äquivalenz-Relation zwischen Idealen. Wir führen die Klassengruppe ein als endliche, Abelsche Gruppe der Äquivalenzklassen. Wir definieren Reduziertheit von Idealen und zeigen, wie Elemente der Klassengruppe dargestellt werden sollen, und wie wir Arithmetik in Klassengruppen über Arithmetik mit Idealen durchgeführen. Wir geben schließlich effiziente Algorithmen an zur Reduktion, Multiplikation, Quadrierung, Invertierung und Division von Idealen, sowie zum Test der Gleichheit zweier Äquivalenzklassen.

In Kapitel 3 führen wir die Komponenten ein, die anschließend in Kapitel 4 für die IQ-Kryptoverfahren notwendig sind. Es handelt sich hierbei um die sog. Bereichsparameter, Schlüsselpaare, die IQ-Version der Diffie-Hellman-Komponente, sowie weitere, symmetrische, kryptographische Komponenten, wie Hash-Funktionen, Message-Authentication-Codes (MACs) und symmetrische Verschlüsselungsverfahren.

Wir geben zunächst Verfahren an zur Konvertierung zwischen Idealen und Octet-Strings. Dabei beschreiben wir eine Kompressionstechnik, wie sie auch schon bei Punkten elliptischer Kurven angewendet wird; in unserem Kontext ist diese Technik jedoch nicht immer anwendbar. Dann stellen wir Verfahren vor, die in Abhängigkeit eines sog. Sicherheitsparameters eine kryptographisch geeignete Klassengruppe auswählen, und wie man eine Klassengruppe zu gegebenem Sicherheitsparameter auf kryptographische Eignung überprüft. Wir geben auch Verfahren an, wie man ein Ideal zufällig aus einer Klassengruppe auswählen kann, und wie man ein Ideal überprüft.

Wir beschreiben dann jeweils zu jedem IQ-Kryptoverfahren aus Kapitel 4 ein Verfahren zur Erzeugung von Bereichsparametern in Abhängigkeit eines Sicherheitsparameters, sowie Verfahren zur Überprüfung von Bereichsparametern bei gegebenem Sicherheitsparamter. Wir geben dann jeweils ein Verfahren an zur Erzeugung von Schlüsselpaaren, sowie zur Überprüfung von öffentlichen Schlüsseln und beschreiben anschließend die IQ-Variante der Diffie-Hellman-Komponente zur Berechnung eines geheimen Wertes.

Schließlich stellen wir schematisch dar, wie wir die symmetrischen Komponenten verwenden wollen. Diese symmetrischen Komponenten sind nicht Gegenstand dieser Arbeit und werden nur abstrakt beschrieben, für eine Implementierung müssen diese Komponenten mitsamt den dazugehörigen Parametern explizit ausgewählt und eingesetzt werden.

In Kapitel 4 stellen wir IQ-basierte Kryptoverfahren zur Signatur, zur Verschlüsselung und zum Austausch eines Geheimnisses vor. Im einzelnen beschreiben wir die Signaturverfahren IQ-RDSA, IQ-DSA und IQ-GQ, das Verschlüsselungsverfahren IQ-IES und das Schlüsselaustauschverfahren IQ-DH. Wir geben zu jedem Verfahren die Schritte an, die notwendig sind, um das jeweilige Verfahren zu verwenden. Zu jedem Signaturverfahren geben wir die Verfahren zur Erzeugung und zur Überprüfung von Signaturen an; zu dem Verschlüsselungsverfahren geben wir die Verfahren zur Ver- und Entschlüsselung an; und zum Schlüsselaustauschverfahren schließlich geben wir das Verfahren an, das zwei Parteien ausführen müssen, um einen geheimen Wert zu berechnen.

Die Sicherheit der IQ-Kryptoverfahren läßt sich beweisen unter der Annahme, daß in den verwendeten Klassengruppen diskrete Logarithmen nicht effizient berechnet werden können. In Kapitel 5 erklären wir, wie der kryptographische Parameter, die Diskriminante, ausgewählt werden muß damit diese Annahme mit sehr hoher Wahrscheinlichkeit erfüllt ist. Hierbei stützen wir uns in wesentlichen Punkten auf unbewiesene Vermutungen (wie der erweiterten Riemann-Vermutung) und Heuristiken (wie der Cohen-Lenstra-Heuristik). Diese Vermutungen und Heuristiken sind jedoch allgemein anerkannt und empirisch bereits für relativ kleine Diskriminanten nachprüfbar.

Wir betrachten alle bekannten Strategien zur Berechnung diskreter Logarithmen in Klassengruppen (IQ-DLP). Hierzu zählen Reduktionen auf einfachere DL-Probleme, Index-Berechnungs-Algorithmen, Quadratwurzel-Algorithmen, sowie Algorithmen, die bei sehr glatten Klassenzahlen effizient angewendet werden können. Wir zeigen zu jeder Strategie, welche Eigenschaften jeweils erfüllt sein müssen, um eine effiziente Anwendung davon zu verhindern. Es stellt sich heraus, daß die Diskriminante fundamental sein sollte, nicht allzu viele Primteiler enthalten sollte, und daß die Diskriminante ansonsten möglichst zufällig und möglichst groß sein sollte. Da die Größe der Diskriminante Einfluß auf die Effizienz der Ideal-Arithmetik hat, darf die Diskriminante auch nicht zu groß sein.

Wir zeigen, wie die Größe der Diskriminante festgelegt werden muß, um ein Mindestmaß an Sicherheit zu erlangen, wobei wir den erwarteten Aufwand für das IQ-DLP vergleichen mit dem Aufwand für andere bekannte Berechnungsprobleme mit bestimmten Parametern (z. B. Faktorisierung eines m-Bit RSA-Modulus). Hierüber läßt sich die Größe der Diskriminante so bestimmen, daß ein IQ-Kryptoverfahren so sicher ist, wie andere Kryptoverfahren mit bestimmter Parametergröße (z. B. RSA mit 1024-Bit-Modulus). In Anlehnung an [47] geben wir dazu ein Verfahren an, um die Größe der Diskriminante zu bestimmen, die notwendig ist, damit ein IQ-Basiertes Verfahren etwa dieselbe Sicherheit aufweist, wie ein ideales symmetrisches Verschlüsselungsverfahren mit bestimmter Schlüssellänge, welches nur durch vollständige Schlüsselsuche gebrochen werden kann.

Bei jedem IQ-Kryptoverfahren müssen Elemente der Klassengruppe zufällig ausgewählt werden. Für die Sicherheit der IQ-Kryptoverfahren ist es notwendig, daß diese Elemente eine möglichst große Untergruppe der Klassengruppe erzeugen. Wir zeigen, daß dies mit sehr hoher Wahrscheinlichkeit der Fall ist, wenn das Element mit Gleichverteilung aus der Klassengruppe ausgewählt wird. Wir geben ein allgemeines Verfahren an, das bei geeigneter Auswahl der Parameter zufällige reduzierte Ideale mit annähernder Gleichverteilung erzeugt. Wir argumentieren, daß auch eine weniger rigorose Auswahl der Parameter zu einer guten Zufallsauswahl führt.

In Kapitel 6 zeigen wir, daß die IQ-Kryptoverfahren aus Kapitel 4 sicher sind, wenn die zugrundeliegenden Basisprobleme schwierig sind. Wir definieren die Basisprobleme, auf die sich die Sicherheit dieser Verfahren gründet und zeigen Zusammenhänge zwischen diesen Berechnungsproblemen. Anschließend definieren wir, was wir meinen, wenn wir sagen, daß ein Verfahren sicher ist.

Dann zeigen wir für jedes IQ-Kryptoverfahren, daß eines der vorher beschriebenen Basisprobleme effizient gelöst werden kann, falls die Erfolgswahrscheinlichkeit eines aktiven Angriffs in polynomieller Zeit nicht vernachlässigbar ist. Unter der Annahme, daß diese Basisprobleme jedoch schwierig sind, muß die Erfolgswahrscheinlichkeit eines aktiven Angriffs daher vernachlässigbar sein.

Im Fall des RDSA-Signaturverfahrens führen wir einen vollständigen Beweis der Sicherheit, im Fall des GQ-Signaturverfahrens führen wir einen unvollständigen Beweis, da RDSA und GQ in gewisser Hinsicht ähnlich sind und einige Details in der Beweisführung identisch sind; im Fall von IQ-DSA, IQ-IES und IQ-DH verweisen wir zum Beweis der Sicherheitseigenschaften auf die Literatur und zitieren nur die relevanten Ergebnisse.

In Kapitel 7 zeigen wir, daß die IQ-Arithmetik und die vorgestellten IQ-Kryptoverfahren effizient sind, und wir geben alternative Algorithmen zur effizienten Idealreduktion an. Wir beschreiben das Standardverfahren nach Gauß, wir beschreiben das Reduktionsverfahren nach Rickert, das von Grundgedanken dem erweiterten Euklidischen Algorithmus nach Lehmer ähnlich ist, und wir beschreiben das rekursive Reduktionsverfahren nach Schönhage. Da das primäre Ziel dieser Arbeit nicht die Untersuchung, Implementierung und der Vergleich effizienter Reduktionsverfahren ist, gehen wir nicht detailliert auf die Laufzeiten dieser Algorithmen ein.

Wir geben anschließend Laufzeiten für die Erzeugung und Überprüfung einer Signatur für IQ-RDSA, IQ-DSA und IQ-GQ für verschiedene Sicherheitsparameter an; zum Vergleich geben wir Laufzeiten für die Erzeugung und Überprüfung von DSA- und RSA-Signaturen an, wobei die Parameter so gewählt wurden, daß in etwa dieselben Sicherheitsanforderungen erfüllt werden.

Für die Laufzeitmessung wurde dabei nicht auf die Java-Implementierung von FlexiProvider zurückgegriffen, sondern auf eine spezielle C-Implementierung, die auf GNU MP (`http://www.swox.com/gmp/`) aufbaut. Für die Laufzeiten zum Vergleich mit DSA und RSA wurde auf die populäre und effizient implementierte OpenSSL-Bibliothek (`http://www.openssl.org`) zurückgegriffen

In Anhang A schließlich geben wir die vollständige ASN1-Notation aller Verfahren aus Kapitel 4 wieder.

1.2 Notation

Nachstehend listen wir alle Symbole auf, die wir verwenden werden:

$p \mid a$	p teilt a
$p^e \| a$	$p^e \mid a$ aber $p^{e+1} \nmid a$
$a \perp b$	$\mathrm{ggT}(a, b) = 1$
$\mathbb{Z}$	Ring der ganzen Zahlen
$\mathbb{R}$	Körper der reelen Zahlen
$\mathbb{C}$	Körper der komplexen Zahlen
Δ	imaginär-quadratische Diskriminante
$F(\Delta)$	Aufzählung der Primfaktoren von Δ
$\mathcal{O}_\Delta$	imaginär-quadratische Ordnung der Diskriminante Δ
$\mathrm{Cl}(\Delta)$	Klassengruppe von $\mathcal{O}_\Delta$
$h(\Delta)$	Klassenzahl von $\mathcal{O}_\Delta$, $h(\Delta) = \lvert\mathrm{Cl}(\Delta)\rvert$
$1_{\mathrm{Cl}(\Delta)}$	neutrales Element von $\mathrm{Cl}(\Delta)$
$\mathrm{Cl}_2(\Delta)$	Untergruppe der ambigen Klassen von $\mathrm{Cl}(\Delta)$

$h_2(\Delta)$	Anzahl der ambigen Klassen von $\mathrm{Cl}(\Delta)$, $h_2(\Delta) = \lvert\mathrm{Cl}_2(\Delta)\rvert$
$\mathrm{Cl}_{\mathsf{odd}}(\Delta)$	Untergruppe der Klassen ungerader Ordnung von $\mathrm{Cl}(\Delta)$
$h_{\mathsf{odd}}(\Delta)$	Anzahl der Klassen ungerader Ordnung von $\mathrm{Cl}(\Delta)$, $h_{\mathsf{odd}}(\Delta) = \lvert\mathrm{Cl}_{\mathsf{odd}}(\Delta)\rvert$
$\mathfrak{a}$, $\mathfrak{b}$	Ideale aus $\mathcal{O}_\Delta$
$\overline{\mathfrak{a}}$, $\overline{\mathfrak{b}}$	reduzierte Ideale äquivalent zu $\mathfrak{a}$ bzw. $\mathfrak{b}$
$[\mathfrak{a}], [\mathfrak{b}]$	Ideal-Klassen in $\mathrm{Cl}(\Delta)$
$\mathrm{ord}_{\mathrm{Cl}(\Delta)}([\mathfrak{a}])$	Ordnung der Untergruppe von $\mathrm{Cl}(\Delta)$, die von $[\mathfrak{a}]$ erzeugt wird
$\zeta(s)$	Riemannsche zeta-Funktion
$\theta(x)$	Tschebyschevsche theta-Funktion
$\pi(x)$	Anzahl der Primzahlen $\leqslant x$
$\varphi(n)$	Eulersche Totient-Funktion
$\rho(u)$	Dickmannsche rho-Funktion
$\mathrm{size}(n)$	$\lfloor \log_2 \lvert n\rvert \rfloor + 1$, die Bitlänge von n, wobei $\mathrm{size}(0) = 1$ ist
$S_1 \| S_2$	die Konkatenierung der Octet-Strings M_1 und M_2
$\mathbf{s}_1 \| \mathbf{s}_2$	die Konkatenierung der Bit-Strings $\mathbf{s}_1$ und $\mathbf{s}_2$

Teil I

Implementierung IQ-basierter Kryptoverfahren

Kapitel 2

Klassengruppen

Die Kryptoverfahren, die in dieser Arbeit beschrieben und untersucht werden, basieren auf *Klassengruppen imaginär-quadratischer Körper*. In diesem Kapitel definieren wir solche Klassengruppen und beschreiben, wie man in solchen Gruppen effizient Arithmetik betreiben kann. Wir führen die Ideen und mathematischen Konzepte ein, die notwendig sind, um die kryptographischen Komponenten aus Kapitel 3 und die Kryptoverfahren aus Kapitel 4 zu verstehen und zu implementieren. Eine Begründung für die Parameterauswahl wird in Kapitel 5 gegeben, die Eigenschaften von Klassengruppen, die hierfür notwendig sind, stehen ebenfalls dort. Eine fundierte Behandlung zu Klassengruppen quadratischer Körper findet man in den Werken von Borevič und Šafarevič [9, Kapitel II, §7], Buchmann [12], Buell [16], Cohen [19, Kapitel 5], Cox [22], Hua [35, Kapitel 9] und Mollin [54].

2.1 Imaginär-quadratische Körper, Ordnungen und Ideale

Die primitiven, ganzen Ideale imaginär-quadratischer Ordnungen sind die arithmetischen Objekte, mit denen wir später rechnen werden. In diesem Abschnitt definieren wir solche Ideale. Wir definieren auch die grundlegende Eigenschaften der Normalität und Reduziertheit von Idealen und geben effiziente Algorithmen an, mit denen man ein Ideal auf Normalität bzw. Reduziertheit überprüfen kann. Außerdem geben wir einen effizienten Algorithmus zur Erzeugung von Primidealen an.

Definition 2.1.1
Eine negative Zahl $\Delta \in \mathbb{Z}$ heißt imaginär-quadratische Diskriminante, *wenn $\Delta \equiv 0, 1 \pmod 4$ ist.*

Da wir es in dieser Arbeit nur mit imaginär-quadratischen Diskriminanten zu tun haben werden, sagen wir einfach nur „Diskriminante“ anstelle von „imaginär-quadratischer Diskriminante“.

Definition 2.1.2
Der Führer *$f(\Delta)$ einer Diskriminante Δ ist die größte ganze Zahl f, so daß Δ/f^2 eine Diskriminante ist.*

Definition 2.1.3
Eine Diskriminante Δ heißt fundamental, *wenn $f(\Delta) = 1$ ist.*

Proposition 2.1.4
Sei Δ eine Diskriminante. Dann ist Δ ist fundamental genau dann, wenn Δ quadratfrei ist, falls $\Delta \equiv 1 \pmod 4$ ist, bzw. wenn $\Delta/4$ quadratfrei und $\Delta/4 \equiv 2, 3 \pmod 4$ ist, falls $\Delta \equiv 0 \pmod 4$ ist.

Definition 2.1.5
Sei Δ eine imaginär-quadratische Diskriminante, dann ist die imaginär-quadratische Ordnung *$\mathcal{O}_\Delta$ der Modul*

$$\mathcal{O}_\Delta = \mathbb{Z} + \frac{\Delta + \sqrt{\Delta}}{2}\mathbb{Z} \ . \tag{2.1}$$

Der Quotienten-Körper von $\mathcal{O}_\Delta$ ist der imaginär-quadratische Körper $\mathbb{Q}(\sqrt{\Delta})$. $\mathcal{O}_\Delta$ ist ein Ring, der 1 enthält. Jedes vom Null-Ideal verschiedene $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}$ kann folgendermaßen auf eindeutige Weise dargestellt werden:

$$\mathfrak{a} = q\left(a\mathbb{Z} + \frac{b + \sqrt{\Delta}}{2}\mathbb{Z}\right) \ , \tag{2.2}$$

wobei $a, b \in \mathbb{Z}$, $q \in \mathbb{Q}$, $a, q > 0$, a und q eindeutig, b eindeutig modulo $2a$, $4a \mid b^2 - \Delta$ und $\mathrm{ggT}(a, b, c) = 1$ mit $c = (b^2 - \Delta)/4a$. Wegen $4a \mid b^2 - \Delta$ ist $b \equiv \Delta \pmod 2$.

Definition 2.1.6
$\mathfrak{a}$ *heißt* ganzes Ideal, *wenn $q \in \mathbb{Z}$ ist, und ansonsten heißt* $\mathfrak{a}$ gebrochenes Ideal. $\mathfrak{a}$ *heißt* primitiv, *wenn $q = 1$ ist.*

Ein primitives $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}$ stellen wir als Paar (a, b) dar. Wenn wir schreiben $\mathfrak{a} = (a, b)$, dann meinen wir damit, $\mathfrak{a}$ ist das $\mathcal{O}_\Delta$-Ideal, das durch (a, b) dargestellt wird.

Definition 2.1.7
Die Darstellung (a, b) eines $\mathcal{O}_\Delta$-Ideals $\mathfrak{a}$ heißt normal, *wenn $-a < b \leqslant a$ ist.*

Algorithmus 2.1 isNormal

Eingabe: Die Darstellung (a, b) eines $\mathcal{O}_\Delta$-Ideals $\mathfrak{a}$.
Ausgabe: true, falls die Darstellung von $\mathfrak{a}$ normal ist, und ansonsten false.

1: **if** $b \leqslant -a$ **or** $b > a$ **then return** false
2: **return** true

Für alle $m \in \mathbb{Z}$ stellen (a, b) und $(a, b + 2am)$ dasselbe Ideal dar. Eine Darstellung (a, b) von $\mathfrak{a}$ kann normalisiert werden, indem man b durch $a - ((a - b) \bmod 2a)$ ersetzt: Für $r = (a - b) \bmod 2a$ gilt $0 \leqslant r < 2a$, daher ist $-a < a - r \leqslant a$, und außerdem ist $b \equiv a - r \pmod{2a}$.

Algorithmus 2.2 normalize

Eingabe: Die Darstellung (a, b) eines $\mathcal{O}_\Delta$-Ideals.
Ausgabe: Die normalisierte Darstellung (a, b') von $\mathfrak{a}$.

1: **if** isNormal(a, b) **then** $b' \leftarrow b$
2: **else** $b' \leftarrow a - ((a - b) \bmod 2a)$
3: **return** (a, b')

Mit der Darstellung (a, b) für ein $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}$ meinen wir von nun an stets die normale Darstellung von $\mathfrak{a}$, falls nichts anderes gesagt wird.

Definition 2.1.8
Sei (a, b) die normale Darstellung eines $\mathcal{O}_\Delta$-Ideals $\mathfrak{a}$, und sei $c = (b^2 - \Delta)/4a$. $\mathfrak{a}$ heißt reduziert, *wenn $a \leqslant c$, und wenn $a = c$, dann $b \geqslant 0$.*

Reduzierte Ideale sind von beschränkter Größe, denn es gilt:

Proposition 2.1.9
Sei (a, b) die Darstellung eines reduzierten $\mathcal{O}_\Delta$-Ideals. Dann ist $a \leqslant \sqrt{|\Delta|/3}$.

Wenn (a, b) die normale Darstellung eines $\mathcal{O}_\Delta$-Ideals $\mathfrak{a}$ ist, dann ist $|b| \leqslant a$. Daraus folgt, daß zur Speicherung eines reduzierten $\mathcal{O}_\Delta$-Ideals höchstens size(Δ) Bits notwendig sind.

Proposition 2.1.10
Sei (a, b) die Darstellung eines $\mathcal{O}_\Delta$-Ideals $\mathfrak{a}$, und sei $a < \sqrt{|\Delta|}/2$. Dann ist $\mathfrak{a}$ reduziert.

Algorithmus 2.3 isReduced

Eingabe:

1. Eine Diskriminante Δ;
2. die normale Darstellung (a, b) eines $\mathcal{O}_\Delta$-Ideals $\mathfrak{a}$.

Ausgabe: true, falls $\mathfrak{a}$ reduziert ist, und ansonsten false.

1: **if** $a \leqslant 0$ **then return** false
2: **if** $b \leqslant -a$ **or** $b > a$ **then return** false
3: $c \leftarrow \dfrac{b^2 - \Delta}{4a}$
4: **if** $a > c$ **then return** false
5: **if** $a = c$ **and** $b < 0$ **then return** false
6: **return** true

Wir beschreiben nun, wie wir $\mathcal{O}_\Delta$-Ideale erzeugen können. Ist $a \in \mathbb{Z}_{>0}$ eine Zahl, so daß Δ modulo $4a$ einen quadratischen Rest läst, dann existiert ein $b \in \mathbb{Z}$, so daß $\Delta \equiv b^2 \pmod{4a}$, und (a, b) stellt ein $\mathcal{O}_\Delta$-Ideal dar. b kann effizient berechnet werden, wenn die Primfaktorisierung von a vorliegt. Wenn a groß und zufällig gewählt ist, dann kann a i. allg. nicht effizient faktorisiert werden. Wir beschreiben nun einen Algorithmus, der zu einer gegebenen Primzahl p ein b berechnet, so daß (p, b) ein $\mathcal{O}_\Delta$-Primideal ist, falls Δ ein quadratischer Rest modulo $4p$ ist (siehe z. B. auch [12, Abschnitt 3.4]).

Algorithmus 2.4 primeldeal

Eingabe:

1. Eine imaginär-quadratische Diskriminante Δ;
2. eine Primzahl p.

Ausgabe: Ein $\mathcal{O}_\Delta$-Primideal (p, b) oder invalid.

1: **if** $p \neq 2$ **then**
2: **if** $\left(\frac{\Delta}{p}\right) = -1$ **then return** invalid
3: $b \leftarrow \mathsf{ressol}(\Delta, p)$
4: **if** $\Delta \not\equiv b \pmod 2$ **then** $b \leftarrow p - b$
5: **else**
6: // $p = 2$
7: **if** $\Delta \equiv 0 \pmod 8$ **then**
8: $b \leftarrow 0$
9: **else if** $\Delta \equiv 1 \pmod 8$ **then**
10: $b \leftarrow 1$
11: **else if** $\Delta \equiv 4 \pmod 8$ **then**
12: $b \leftarrow 2$
13: **else**
14: // $\Delta \equiv 5 \pmod 8$
15: **return** invalid
16: **end if**
17: **end if**
18: **return** (p, b)

Hierbei bezeichne ressol den Algorithmus von Tonelli und Shanks, das eine Quadratwurzel modulo einer Primzahl p berechnet:

Algorithmus 2.5 ressol

Eingabe:

1. eine Primzahl p;
2. eine ganze Zahl n mit $\left(\frac{n}{p}\right) = 1$.

Ausgabe: Eine ganze Zahl r, $0 \leqslant r < p$ mit $r^2 \equiv n \pmod p$.

Die Details des Algorithmus sind z. B. in [19, Abschnitt 1.5] oder [12, Abschnitt 3.4.7] wiedergegeben.

2.2 Klassengruppen

Die Klassengruppen imaginär-quadratischer Körper sind die Gruppen, die wir für die Public-Key-Verfahren in dieser Arbeit verwenden. In diesem Abschnitt definieren wir die Klassengruppen imaginär-quadratischer Körper, wir beschreiben, wie wir Gruppenelemente darstellen

wollen, und wie wir Arithmetik in Klassengruppen betreiben, indem wir auf Ideal-Arithmetik zurückgreifen. Außerdem geben wir an ein effizienten Algorithmus zur Reduktion von Idealen, sowie effiziente Algorithmen zur Multiplikation, Quadrierung, Invertierung und Division von Idealen (alternative effiziente Reduktions-Algorithmen werden in Kapitel 7 beschrieben).

Definition 2.2.1
Zwei $\mathcal{O}_\Delta$-Ideale $\mathfrak{a}_1, \mathfrak{a}_2$ heißen äquivalent, *wenn eine Zahl $\alpha \in \mathbb{Q}(\sqrt{\Delta})$, $\alpha \neq 0$, existiert, so daß $\mathfrak{a}_2 = \alpha\mathfrak{a}_1$ ist. Sind $\mathfrak{a}_1$ und $\mathfrak{a}_2$ äquivalent, so schreiben wir $\mathfrak{a}_1 \sim \mathfrak{a}_2$, ansonsten $\mathfrak{a}_1 \not\sim \mathfrak{a}_2$.*

Die Relation $\sim$ ist eine Äquivalenzrelation. Die Äquivalenzklasse eines $\mathcal{O}_\Delta$-Ideals $\mathfrak{a}$ schreiben wir als $[\mathfrak{a}]$ und bezeichnen dies kurz auch als Ideal-Klasse von $\mathfrak{a}$.

Beispiel: Das $\mathcal{O}_\Delta$-Ideal (a, b) ist äquivalent zu $(c, -b)$ für $c = (b^2 - \Delta)/4a$ mit $\alpha = 2c/(b + \sqrt{\Delta}) = (b - \sqrt{\Delta})/2a$.

Für das Produkt zweier $\mathcal{O}_\Delta$-Ideale $\mathfrak{a}_1$ und $\mathfrak{a}_2$ schreiben wir $\mathfrak{a}_1 \cdot \mathfrak{a}_2$ oder kurz $\mathfrak{a}_1\mathfrak{a}_2$. Zu jedem primitiven $\mathcal{O}_\Delta$-Ideal $\mathfrak{a} = (a, b)$ existiert ein inverses, gebrochenes Ideal $\mathfrak{a}^{-1}$, das äquivalent zu dem primitiven Ideal $(a, -b)$ ist, (siehe z. B. [19, Proposition 5.2.5]); wir bezeichnen das ganze Ideal $(a, -b)$ mit $\mathfrak{a}^\star$. Für $\mathfrak{a}_1\mathfrak{a}_2^{-1}$ schreiben wir auch $\mathfrak{a}_1/\mathfrak{a}_2$.

Theorem 2.2.2
Die Äquivalenzklassen der Ideale von $\mathcal{O}_\Delta$ bilden unter Ideal-Multiplikation eine endliche, Abelsche Gruppe.

Definition 2.2.3
Die Gruppe der Äquivalenzklassen der Ideale von $\mathcal{O}_\Delta$ heißt Klassengruppe *und wird mit* $\mathrm{Cl}(\Delta)$ *bezeichnet.*

Definition 2.2.4
Die Ordnung der Klassengruppe heißt Klassenzahl *und wird mit $h(\Delta)$ bezeichnet.*

Wir bezeichnen die Elemente von $\mathrm{Cl}(\Delta)$ mit $\mathfrak{a}$, $\mathfrak{b}$ usw. Das Produkt zweier Elemente $\mathfrak{a}_1$ und $\mathfrak{a}_2$ von $\mathrm{Cl}(\Delta)$ bezeichnen wir mit $\mathfrak{a}_1 \cdot \mathfrak{a}_2$ oder kurz mit $\mathfrak{a}_1\mathfrak{a}_2$. Sind $\mathfrak{a}_1$ und $\mathfrak{a}_2$ Vertreter von $\mathfrak{a}_1$ bzw. $\mathfrak{a}_2$, dann ist das Produkt $\mathfrak{a}_1\mathfrak{a}_2$ definiert durch $[\mathfrak{a}_1\mathfrak{a}_2]$. $\mathfrak{a}_2$ ist das zu $\mathfrak{a}_1$ inverse Element, wenn $\mathfrak{a}_1 \sim \mathfrak{a}_2^{-1}$ ist, und wir schreiben $\mathfrak{a}_2 = \mathfrak{a}_1^{-1}$; für $\mathfrak{a}_1\mathfrak{a}_2^{-1}$ schreiben wir auch $\mathfrak{a}_1/\mathfrak{a}_2$.

Um in Klassengruppen Arithmetik betreiben zu können, muß man einen Vertreter aus jeder Äquivalenzklasse auswählen. Wir wählen dafür die reduzierten Ideale aus, denn es gilt:

Theorem 2.2.5
Zu jedem $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}$ existiert genau ein primitives, reduziertes Ideal $\overline{\mathfrak{a}}$, das äquivalent zu $\mathfrak{a}$ ist.

Wir betreiben Arithmetik in Klassengruppen auf folgende Weise: Seien $\mathfrak{a}_1$ und $\mathfrak{a}_2$ Elemente von $\mathrm{Cl}(\Delta)$, und seien $\mathfrak{a}_1$ und $\mathfrak{a}_2$ die reduzierten Vertreter von $\mathfrak{a}_1$ bzw. $\mathfrak{a}_2$. Zum Vergleich und zur Multiplikation von $\mathfrak{a}_1$ und $\mathfrak{a}_2$ verwenden wir $\mathfrak{a}_1$ und $\mathfrak{a}_2$; mit Arithmetik in Klassengruppen meinen wir also Ideal-Arithmetik mit den Vertreter-Idealen. Da das Produkt zweier Ideale i. allg. nicht reduziert ist, reduzieren wir stets das Ergebnis einer Multiplikation. Wenn wir also $\mathfrak{a}_1\mathfrak{a}_2$ schreiben, dann meinen wir damit die Berechnung von $\overline{\mathfrak{a}_1\mathfrak{a}_2}$.

Der nachfolgende Algorithmus berechnet zu einem $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}$ das äquivalente, reduzierte Ideal $\overline{\mathfrak{a}}$ und gibt die normale Darstellung von $\overline{\mathfrak{a}}$ zurück:

Algorithmus 2.6 reduce

Eingabe:

1. Eine Diskriminante Δ;
2. Ein $\mathcal{O}_\Delta$-Ideal $\mathfrak{a} = (a, b)$.

Ausgabe: Das äquivalente, reduzierte $\mathcal{O}_\Delta$-Ideal $\overline{\mathfrak{a}}$.

if $b < 0$ **then**
  $b \leftarrow -b$, $sgn \leftarrow -1$
else
  $sgn \leftarrow 1$
end if
$r \leftarrow b \bmod 2a$
if $r > a$ **then**
  $b \leftarrow 2a - r$, $sgn \leftarrow -sgn$
else
  $b \leftarrow r$
end if
$c \leftarrow (b^2 - \Delta)/4a$
while $a > c$ **do**
  $a \leftarrow c$, $c \leftarrow a$
  $q \leftarrow \lfloor b/2a \rfloor$, $r \leftarrow b \bmod 2a$
  $t_b \leftarrow a - r$
  $c \leftarrow c - q(r + b)/2$
  if $t_b < 0$ **then**
    $b \leftarrow a + t_b$
    $c \leftarrow c + t_b$
  else
    $b \leftarrow r$
    $sgn \leftarrow -sgn$
  end if
end while
if $a = c$ **or** $a = b$ **then** $sgn \leftarrow 1$
if $sgn < 0$ **then** $b \leftarrow -b$
return (a, b)

Dieser Algorithmus ist im wesentlichen aus [40] entnommen. Es ist eine optimierte Variante des Gaußschen Reduktions-Algorithmus (siehe z. B. [12, Abschnitt 5.3]). In Kapitel 7 beschreiben wir alternative Reduktions-Algorithmen.

Korollar 2.2.6
Zwei $\mathcal{O}_\Delta$-Ideale $\mathfrak{a}_1, \mathfrak{a}_2$ sind genau dann äquivalent, wenn $\overline{\mathfrak{a}_1} = \overline{\mathfrak{a}_2}$ ist.

Der nachstehende Algorithmus prüft, ob die Äquivalenzklassen zweier $\mathcal{O}_\Delta$-Ideale gleich sind:

Algorithmus 2.7 isEqual

Eingabe: $\mathcal{O}_\Delta$-Ideale $\mathfrak{a}_1$ und $\mathfrak{a}_2$.
Ausgabe: true, falls $[\mathfrak{a}_1] = [\mathfrak{a}_2]$, und ansonsten false.

1: $(\bar{a}_1, \bar{b}_1) \leftarrow \mathsf{reduce}(\Delta, \mathfrak{a}_1)$
2: $(\bar{a}_2, \bar{b}_2) \leftarrow \mathsf{reduce}(\Delta, \mathfrak{a}_2)$
3: **if** $\bar{a}_1 = \bar{a}_2$ **and** $\bar{b}_1 = \bar{b}_2$ **then return** true
4: **else return** false

Wenn zum Vergleich bereits die reduzierten Vertreter (jeweils in normaler Darstellung) herangezogen werden, dann ist ein Test auf Gleichheit sehr effizient.

Wir präsentieren nun effiziente Algorithmen für die Arithmetik mit $\mathcal{O}_\Delta$-Idealen. Es handelt sich hierbei um Standard-Algorithmen, wie man sie z. B. auch in [12, Kapitel 7] findet. Die hier wiedergegebenen Algorithmen sind optimierte Varianten davon aus [40]. Der folgende Algorithmus kann zur Ideal-Multiplikation verwendet werden:

Algorithmus 2.8 multiply

Eingabe: $\mathcal{O}_\Delta$-Ideale $\mathfrak{a}_1 = (a_1, b_1)$ und $\mathfrak{a}_2 = (a_2, b_2)$.
Ausgabe: Das reduzierte $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}_3 = (a_3, b_3)$ äquivalent zu $\mathfrak{a}_1 \cdot \mathfrak{a}_2$.

1: $d_1, v, w \leftarrow \mathsf{xgcd}(a_1, a_2)$
2: $a_3 \leftarrow a_1 a_2$
3: $b_3 \leftarrow v a_1 (b_2 - b_1)$
4: **if** $d_1 \neq 1$ **then**
5: $\quad d_2, v, w \leftarrow \mathsf{xgcd}(d_1, (b_1 + b_2)/2)$
6: $\quad a_3 \leftarrow a_3 / d_2^2$
7: $\quad b_3 \leftarrow (b_3 v + w(\Delta - b_1^2)/2)/d_2$
8: **end if**
9: $b_3 \leftarrow b_1 + b_3$
10: $(a_3, b_3) \leftarrow \mathsf{reduce}(\Delta, (a_3, b_3))$
11: **return** (a_3, b_3)

xgcd bezeichne die folgende Funktion:

Algorithmus 2.9 xgcd

Eingabe: $a, b \in \mathbb{Z}$.
Ausgabe: d, v, w, wobei $d = \mathrm{ggT}(a, b)$ und $d = av + bw$ ist

Die Details dieses Algorithmus findet man z. B. in [11, Abschnitt 1.9] oder [19, Abschnitt 1.3.2].

Für den Spezialfall der Multiplikation eines Ideals mit sich selbst (d. h. Quadrierung) gibt es ein speziellen Algorithmus, das in der Praxis geringfügig schneller ist, als der Algorithmus zur Ideal-Multiplikation:

Algorithmus 2.10 square

Eingabe: Ein $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}_1 = (a_1, b_1)$
Ausgabe: Das reduzierte $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}_3 = (a_3, b_3)$ äquivalent zu $\mathfrak{a}_1^2$

1: $d, v, w \leftarrow \mathsf{xgcd}(a_1, b_1)$
2: $a_3 \leftarrow (a_1/d)^2$
3: $b_3 \leftarrow w(\Delta - b_1^2)/(2d)$
4: $b_3 \leftarrow b_1 + b_3$
5: $(a_3, b_3) \leftarrow \mathsf{reduce}(\Delta, (a_3, b_3))$
6: **return** (a_3, b_3)

Ist $\mathfrak{a} = (a, b)$ ein $\mathcal{O}_\Delta$-Ideal, dann ist die Berechnung eines ganzen Ideals, das äquivalent zum inversen Ideal $\mathfrak{a}^{-1}$ ist, besonders effizient, denn $\mathfrak{a}^\star = (a, -b)$ ist äquivalent zu $\mathfrak{a}^{-1}$. Ist $\mathfrak{a}$ ein reduziertes $\mathcal{O}_\Delta$-Ideal mit normaler Darstellung (a, b), dann ist auch $\mathfrak{a}^\star$ reduziert, und $(a, -b)$ ist die normale Darstellung von $\mathfrak{a}^\star$, außer wenn $a = b$ ist. In diesem Fall ist bereits $\mathfrak{a} \sim \mathfrak{a}^{-1}$, da (a, a) das äquivalente, reduzierte Ideal zu $(a, -a)$ ist. Dieser Fall tritt aber praktisch nicht auf. Die Invertierung eines Gruppenelements (dargestellt durch das entsprechende reduzierte Ideal) besteht also aus dem Wechsel eines Vorzeichens.

Algorithmus 2.11 inverse

Eingabe: Ein reduziertes $\mathcal{O}_\Delta$-Ideal $\mathfrak{a} = (a, b)$.
Ausgabe: Das reduzierte $\mathcal{O}_\Delta$-Ideal äquivalent zu $\mathfrak{a}^{-1}$.

1: **if** $a \neq b$ **then return** $(a, -b)$
2: **else return** (a, b)

Sind $\mathfrak{a}_1 = (a_1, b_1)$ und $\mathfrak{a}_2 = (a_2, b_2)$ $\mathcal{O}_\Delta$-Ideale, dann erfolgt die Berechnung eines Ideals, das äquivalent zu $\mathfrak{a}_1/\mathfrak{a}_2$ ist, über die Berechnung von $\mathfrak{a}_1\mathfrak{a}_2^\star$. Um bei einer Implementierung der später beschriebenen Kryptoverfahren temporäre Variablen zu sparen, definieren wir dennoch eine eigene Methode zur Division, welche Invertierung und Multiplikation kombiniert:

Algorithmus 2.12 divide

Eingabe: $\mathcal{O}_\Delta$-Ideale $\mathfrak{a}_1 = (a_1, b_1)$ und $\mathfrak{a}_2 = (a_2, b_2)$.
Ausgabe: Das reduzierte $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}_3 = (a_3, b_3)$ äquivalent zu $\mathfrak{a}_1/\mathfrak{a}_2$.

1: $d_1, v, w \leftarrow \mathsf{xgcd}(a_1, a_2)$
2: $a_3 \leftarrow a_1 a_2$
3: $b_3 \leftarrow w a_2 (b_2 + b_1)$
4: **if** $d_1 \neq 1$ **then**
5: $\quad d_2, v, w \leftarrow \mathsf{xgcd}(d_1, (b_1 - b_2)/2)$
6: $\quad a_3 \leftarrow a_3/d_2^2$
7: $\quad b_3 \leftarrow (b_3 v + w(\Delta - b_1^2)/2)/d_2$
8: **end if**
9: $b_3 \leftarrow b_1 + b_3$
10: $(a_3, b_3) \leftarrow \mathsf{reduce}(\Delta, (a_3, b_3))$
11: **return** (a_3, b_3)

Kapitel 3

Kryptographische Komponenten

Um eines der in Kapitel 4 beschriebenen Kryptoverfahren zu verwenden, müssen zuvor einige Parameter (z. B. die Diskriminate, durch welche die Klassengruppe eindeutig bestimmt wird) festgelegt werden, außerdem muß ein Schlüsselpaar gewählt werden. Diese Parameter und Schlüsselpaare müssen so gewählt werden, daß die Berechnung des geheimen Schlüssels oder die Fälschung einer Signatur, die Entschlüsselung oder Verfälschung eines Schlüsseltextes, oder die Berechnung eines geheimen Wertes durch nicht authorisierte Parteien nur mit vernachlässigbarer Wahrscheinlichkeit effizient möglich ist. In Kapitel 5 werden wir dazu zeigen, wie die Diskriminante gewählt werden muß, um die effiziente Berechnung von diskreten Logarithmen in Klassengruppen oder Untergruppen davon, die von zufällig gewählten Elementen erzeugt werden, effektiv zu verhindern; in Kapitel 6 werden wir zeigen, daß unter dieser Voraussetzung die IQ-Kryptoverfahren selber sicher sind. Es muß auch möglich sein, die Parameter und die öffentlichen Schlüssel zu überprüfen, um Angriffe aufgrund kryptographisch ungeeigneter Parameter oder Schlüssel ausschließen zu können.

Für die praktische Verwendbarkeit der IQ-Kryptoverfahren muß auch festgelegt werden, wie Elemente der Klassengruppe dargestellt werden sollen, wobei wir für verschiedene Zwecke verschiedene Darstellungen brauchen. Wir brauchen daher jeweils eine Methode, um eine Darstellung in jede andere zu konvertieren.

Die Verfahren zur Auswahl und Überprüfung der kryptographischen Parameter und Schlüsselpaare werden auch als kryptographische Komponenten oder auch kryptographische Primitive bezeichnet. Hierzu gehören auch die symmetrischen Kryptoverfahren, die mit den IQ-Kryptoverfahren verwendet werden; für die Signaturverfahren brauchen wir z. B. kryptographische Hash-Funktionen. In Anhang A geben wir eine ASN.1-Codierung für Diskriminanten, Ideale, Parameter und Schlüsselpaare für alle IQ-Kryptoverfahren, soweit diese in FlexiPKI implementiert sind.

In diesem Kapitel beschreiben wir ausführlich alle Komponenten und Bausteine, die notwendig sind, um die IQ-Kryptoverfahren aus Kapitel 4 zu verwenden. Die Beschreibung erfolgt in einer Weise, wie es sich bereits durch diverse Standardisierungs-Dokumente eingebürgert hat. Wir nehmen dabei konkret den Standard für elliptische Kurven SEC 1 [71, Kapitel 3 und teilw. Kapitel 2] als Vorbild.

Im Einzelnen beschreiben wir in Abschnitt 3.1 zunächst, welche Datenformate wir für verschiedene Zwecke brauchen und geben Verfahren zur Konvertierung von Idealen in der Darstellung als Paar (a, b) in eine eindeutige Darstellung als Octet-String an, und umgekehrt.

Die Konvertierung von und nach Bit-Strings erfolgt über Octet-Strings und wird hier nicht explizit wiedergegeben.

In Abschnitt 3.2 beschreiben wir die Bereichsparameter für die einzelnen IQ-Kryptoverfahren. Hier geben wir zunächst Verfahren an zur Auswahl und zur Überprüfung von kryptographisch geeigneten Diskiminanten in Abhängigkeit eines gegebenen Sicherheitsparameters, sowie zur zufälligen Auswahl und zur Überprüfung eines Gruppenelements. Darauf aufbauend geben wir anschließend für jedes einzelne IQ-Kryptoverfahren Methoden zur Erzeugung und Überprüfung der Bereichsparameter an.

In Abschnitt 3.3 beschreiben wir, wie die Schlüsselpaare für die IQ-Kryptoverfahren aussehen und geben für jedes IQ-Kryptoverfahren Methoden zur Erzeugung von Schlüsselpaaren und zur Überprüfung von öffentlichen Schlüsseln an.

In Abschnitt 3.4 beschreiben wir die Komponente (Diffie-Hellman) zur Berechnung eines gemeinsamen geheimen Wertes. Diese Komponente brauchen wir in Kapitel 4 nur für die asymmetrische Verschlüsselung sowie für den Schlüsselaustausch

In Abschnitt 3.5 schließlich beschreiben wir schematisch alle sonstigen symmetrischen kryptographischen Komponenten, die wir in Kapitel 4 brauchen werden, wie Hash-Funktionen, Message-Authentication-Codes und symmetrische Verschlüsselungen.

Hilfs-Algorithmen

Wir verwenden im weiteren Verlauf dieser Arbeit die folgenden Hilfs-Algorithmen. Sei n eine ganze Zahl $\geqslant 0$:

- randomInteger(n) gebe ein Zahl zurück, die zufällig mit Gleichverteilung aus dem Intervall $[1, \ldots, n]$ gezogen werde.
- oddRandomInteger(n) gebe ein ungerade Zahl zurück, die zufällig mit Gleichverteilung aus dem Intervall $[1, \ldots, n]$ gezogen werde.
- isPrime(n) prüfe, ob n eine Primzahl ist; dies kann z. B. probabilistisch mit dem Rabin-Miller-Test durchgeführt werden.
- nextPrime(n) gebe die kleinste Primzahl $p > n$ zurück.

3.1 Datenformate und Konvertierung

In Kapitel 2 hatten wir bereits dargelegt, daß wir Elemente der Klassengruppe, die Äquivalenzklassen, durch die eindeutig bestimmten reduzierten Ideale der jeweiligen Äquivalenzklasse darstellen. Wir benötigen später verschiedene Darstellungen eines Ideals für verschiedene Zwecke:

1. Darstellung als Paar (a, b) für arithmetische Operationen.
2. Darstellung als Octet-String zur Speicherung und Übermittlung.
3. Darstellung als Bit-String.

Ein Octet-String $M = M_0 M_1 \ldots$ ist eine Folge von Octets $M_i \in \{00_{16}, \ldots, \mathrm{FF}_{16}\}$, und ein Bit-String $\mathbf{b} = b_0 b_1 \ldots$ ist eine Folge von Bits $b_i \in \{0, 1\}$.

In diesem Abschnitt beschreiben wir die Konvertierung von Idealen zu Octet-Strings und umgekehrt. Wir machen dabei von einer Kompressionstechnik für Ideale gebrauch, wie sie bereits für Punkte elliptischer Kurven angewendet wird. Diese Technik läßt sich in unserem Kontext allerdings nicht immer anwenden.

Zur Konvertierung von Idealen zu Octet-Strings und umgekehrt benötigen wir Verfahren zur Konvertierung zwischen ganzen Zahlen und Octet-Strings. Solche Algorithmen sind z. B. in [71] beschrieben; die entsprechenden Konvertierungs-Algorithmen bezeichnen wir mit integerToOctets und octetsToInteger. Die Konvertierung zwischen Bit-Strings und Octet-Strings ist z. B. in [71] beschrieben; die entsprechenden Konvertierungs-Algorithmen bezeichnen wir mit bitsToOctets und octetsToBits. Die Spezifikation für diese Konvertierungs-Algorithmen lautet folgendermaßen. Es bezeichnet n eine ganze Zahl, N einen Octet-String der Länge ℓ_N und $\mathbf{n}$ einen Bit-String der Länge $\ell_{\mathbf{n}}$.

- integerToOctets(n, ℓ_N) gebe einen Octet-String N der Länge ℓ_N zurück, wobei wir $0 \leqslant n < 256^{\ell_N}$ voraussetzen.
- octetsToInteger(N) gebe eine Zahl $n \geqslant 0$ zurück.
- bitsToOctets$(\mathbf{n})$ gebe einen Octet-String N der Länge $\lceil \ell_{\mathbf{n}}/8 \rceil$ zurück.
- octetsToBits(N) gebe einen Bit-String $\mathbf{n}$ der Länge $8\ell_N$ zurück.

3.1.1 Konvertierung von reduzierten Idealen zu Octet-Strings

Wir beschreiben nun die Konvertierung von reduzierten Idealen zu Octet-Strings. Wie bei Punkten von elliptischen Kurven besteht auch hier die Möglichkeit, Ideale (a, b) zu komprimieren. Dies ist allerdings nur dann effizient möglich, falls die Primfaktor-Zerlegung von a bekannt ist. Da dies nicht immer möglich sein wird, und um das Verfahren so einfach wie möglich zu halten, beschränken wir uns bei der Kompression auf den Fall, daß a eine Primzahl ist.

Algorithmus 3.1 idealToOctets

Eingabe:

1. Die Diskrimante Δ eines imaginär-quadratischen Körpers;
2. ein reduziertes $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}$ dargestellt als Paar (a, b);
3. *compress* $\in \{\mathsf{true}, \mathsf{false}\}$.

Ausgabe: Die Darstellung von $\mathfrak{a}$ als Octet-String $A = A_0 A_1 \dots A_m$ der Länge $m + 1$.

1: $r = \lceil (\log_2 \sqrt{|\Delta|/3})/8 \rceil$
2: $X_a \leftarrow \mathsf{integerToOctets}(a, r)$
3: **if** *compress* **and** $\mathsf{isPrime}(a)$ **then**
4: **if** $b \geqslant 0$ **then** $A_0 \leftarrow 02_{16}$
5: **else** $A_0 \leftarrow 03_{16}$
6: $A \leftarrow A_0 \| X_a$
7: **else**
8: $X_b \leftarrow \mathsf{integerToOctets}(|b|, r)$
9: **if** $b \geqslant 0$ **then** $A_0 \leftarrow 04_{16}$
10: **else** $A_0 \leftarrow 05_{16}$
11: $A \leftarrow A_0 \| X_a \| X_b$
12: **end if**
13: **return** A

Der resultierende Octet-String hat bei Kompression die Länge $r + 1$, und sonst $2r + 1$, wobei $r = \lceil (\log_2 \sqrt{|\Delta|/3})/8 \rceil$.

Beispiel: Sei $\Delta = -156211\,589831\,799264\,009011$, dann ist $r = \lceil (\log_2 \sqrt{|\Delta|/3})/8 \rceil = 5$. $\mathfrak{a} = (152340\,982457,\ -39286\,141991)$ ist z. B. ein reduziertes $\mathcal{O}_\Delta$-Ideal. Die Darstellung von $a = 152340\,982457$ als Octet-String lautet (mit dem niederwertigsten Octet voran)

$$X_a = \mathrm{B9}_{16}\,\mathrm{EA}_{16}\,\mathrm{3A}_{16}\,78_{16}\,23_{16}\ ,$$

und die Darstellung von $|b| = 39286\,141991$ als Octet-String lautet

$$X_b = 27_{16}\,\mathrm{F4}_{16}\,\mathrm{A2}_{16}\,25_{16}\,09_{16}\ .$$

152340 982457 ist eine Primzahl, wir haben also zwei Möglichkeiten für die Konvertierung:

1. Ohne Kompression lautet die Darstellung von $\mathfrak{a}$ als Octet-String der Länge $2r + 1 = 11$

$$A = 05_{16}\,\mathrm{B9}_{16}\,\mathrm{EA}_{16}\,\mathrm{3A}_{16}\,78_{16}\,23_{16}\,27_{16}\,\mathrm{F4}_{16}\,\mathrm{A2}_{16}\,25_{16}\,09_{16}\ .$$

2. Mit Kompression lautet die Darstellung von $\mathfrak{a}$ als Octet-String der Länge $r + 1 = 6$

$$A = 03_{16}\,\mathrm{B9}_{16}\,\mathrm{EA}_{16}\,\mathrm{3A}_{16}\,78_{16}\,23_{16}\ .$$

3.1.2 Konvertierung von Octet-Strings zu reduzierten Idealen

Wir beschreiben nun die Rekonstruktion der Darstellung (a, b) eines $\mathcal{O}_\Delta$-Ideals $\mathfrak{a}$ aus einem Octet-String $A = A_0 \dots A_m$ der Länge $m + 1$.

Algorithmus 3.2 octetsToIdeal

Eingabe:

1. Die Diskriminante Δ eines imaginär-quadratischen Körpers;
2. die Darstellung eines $\mathcal{O}_\Delta$-Ideals $\mathfrak{a}$ als Octet-String $A = A_0 A_1 \dots A_m$ der Länge $m + 1$.

Ausgabe: Das $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}$ dargestellt als Paar (a, b), oder invalid.

1: $r \leftarrow \lceil (\log_2 \sqrt{|\Delta|/3})/8 \rceil$
2: **if** $m = r$ **then**
3: **if** $A_0 \neq 02_{16}$ **or** $A_0 \neq 03_{16}$ **then return** invalid
4: $p \leftarrow \mathsf{octetsToInteger}(A_1 \dots A_m)$
5: **if not** isPrime(p) **or** $\left(\frac{\Delta}{p}\right) = -1$ **then return** invalid
6: $(a, b) \leftarrow \mathsf{primeIdeal}(\Delta, p)$
7: **if** $A_0 = 03_{16}$ **then** $b \leftarrow -b$
8: **else if** $m = 2r$ **then**
9: **if** $A_0 \neq 04_{16}$ **or** $A_0 \neq 05_{16}$ **then return** invalid
10: $a \leftarrow \mathsf{octetsToInteger}(A_1 \dots A_r)$
11: $b \leftarrow \mathsf{octetsToInteger}(A_{r+1} \dots A_m)$
12: **if** $A_0 = 05_{16}$ **then** $b \leftarrow -b$
13: **if** $4a \nmid b^2 - \Delta$ **then return** invalid
14: **else**
15: **return** invalid
16: **end if**
17: **return** (a, b)

3.2 Die IQ-Bereichsparameter für Fundamental-Diskriminanten

Um eines der IQ-Kryptoverfahren aus Kapitel 4 zu verwenden, muß zunächst der sog. Sicherheitsparameter und die Klassengruppe (bzw. eine Untergruppe davon) festgelegt werden. Diese Parameter können entweder für jede Partei einzeln oder auch für mehrere Parteien gemeinsam festgelegt werden, daher bezeichnet man diese Parameter auch als Bereichsparameter. Die Klassengruppe wird eindeutig durch die Diskriminante festgelegt. Es müssen zudem noch weitere Parameter festgelegt werden; für IQ-DH, IQ-DSA, IQ-RDSA und IQ-IES muß z. B. ein Erzeuger einer Untergruppe der Klassengruppe festgelegt werden. In diesem Abschnitt beschreiben wir für jedes IQ-Kryptoverfahren aus 4 das Format der Bereichsparameter, sowie Methoden zur Erzeugung und Überprüfung der Bereichsparameter.

Die Bereichsparameter gehören im weiteren Sinn zum öffentlichen Schlüssel, da die Bereichsparameter öffentlich sind. Wir legen hier aber (wie allgemein üblich) fest, daß zum öffentlichen Schlüssel nur das gehört, was sich (unter Verwendung der Bereichsparameter) aus dem privaten Schlüssel berechnen läßt.

Wir unterscheiden zwischen den Bereichsparametern für IQ-DH, IQ-DSA, IQ-IES, IQ-RDSA und IQ-GQ, wobei die Bereichsparameter für IQ-DH, IQ-DSA und IQ-IES strukturell identisch sind. Die Bereichsparameter werden durch $\mathcal{D}$ bezeichnet und als Tupel dargestellt.

Bevor wir die Bereichsparameter für jedes dieser Kryptoverfahren beschreiben und Methoden zur Erzeugung und Überprüfung von Bereichsparametern angeben, beschreiben wir zunächst Teilverfahren zur Erzeugung und Überprüfung von kryptographisch geeigneten Diskriminanten, sowie zur Überprüfung eines Gruppenelements.

Verfahren zur Erzeugung einer zufälligen Fundamental-Diskriminante

In diesem Abschnitt geben wir Verfahren zur Auswahl kryptographisch geeigneter Diskriminanten an. Wir werden uns für den Rest der Arbeit auf die folgenden Fälle für Fundamental-Diskriminanten beschränken:

1. $\Delta = -p$, p prim und $p \equiv 3 \pmod 4$;
2. $\Delta = -pq$, p, q prim, $pq \equiv 3 \pmod 4$ und $\left(\frac{p}{q}\right) = -1$.

Um eine zerlegbare Fundamental-Diskriminante Δ effizient überprüfen zu können (Abschnitt 3.2), muß anstelle von Δ dessen Primfaktor-Zerlegung übermittelt werden. Eine solche Primfaktor-Zerlegung bezeichnen wir mit $F(\Delta)$ und schreiben $F(\Delta)$ als Aufzählung $(p_1, \ldots, p_k)$, wobei $\Delta = -\prod_{i=1}^{k} p_i$, wobei für $i \neq j$ nicht notwendigerweise $p_i \neq p_j$ gilt.

Das folgende Verfahren berechnet aus einem Sicherheitsparameter t die Größe der Diskriminante in Bits. In Abschnitt 5.4 werden wir näher auf dieses Verfahren eingehen. Das Verfahren ist so kalibriert, daß ein IQ-Kryptoverfahren in $\mathrm{Cl}(\Delta)$ mit $\mathsf{discSize}(t)$ Bits für Δ sicher ist, wenn ein symmetrisches Verschlüsselungsverfahren mit Schlüssellänge t gegen vollständige Schlüsselsuche sicher ist.

Algorithmus 3.3 discSize

Eingabe: Der Sicherheitsparameter t.
Ausgabe: Die Bit-Länge t_Δ für die Diskriminante.

1: $c' \leftarrow (t \ln 2 + \ln 33.526686)^2$
2: $\xi' \leftarrow c'$
3: **repeat**
4: $\quad \xi \leftarrow \xi'$
5: $\quad \xi' \leftarrow \xi - \dfrac{\xi \ln \xi - c'}{\ln \xi + 1}$
6: **until** $|\xi - \xi'| < 1/2$
7: $t_\Delta \leftarrow \lceil \xi / \ln 2 \rceil$
8: **return** t_Δ

Beispiel: Für $t = 16$ gibt discSize 78 zurück, d. h. für die Berechnung diskreter Logarithmen in Klassengruppen mit 78-Bit-Diskriminante erwarten wir etwa vergleichbaren Aufwand wie zum Brechen eines symmetrischen Verschlüsselungsverfahrens mit 16 Bits Schlüssellänge. Dies ist für kryptographische Zwecke viel zu wenig, aber zur Demonstration werden wir im folgenden einige Beispiele für verschiedene Objekte geben, die wir beschreiben, und wir werden hierzu

der Übersichtlichkeit halber $t = 16$ verwenden. Kryptographisch angemessene Werte für t sind in Tabelle 5.11 aufgelistet.

Das folgende Verfahren erzeugt eine prime Fundamental-Diskriminante mit $\mathsf{discSize}(t)$ Bits.

Algorithmus 3.4 genDiscP

Eingabe: Der Sicherheitsparameter t.
Ausgabe: $F(\Delta)$.

1: $t_\Delta \leftarrow \mathsf{discSize}(t)$
2: **repeat**
3: $\quad p \leftarrow \mathsf{oddRandomInteger}(2^{t_\Delta})$
4: **until** $\mathsf{isPrime}(p)$ **and** $\mathrm{size}(p) = t_\Delta$ **and** $p \equiv 3 \pmod 4$
5: **return** (p)

Beispiel: $\mathsf{genDiscP}(16)$ gibt z. B. $F(\Delta) = (156211\,589831\,799264\,009011)$ zurück, d. h. $\Delta = -156211\,589831\,799264\,009011$.

Das folgende Verfahren erzeugt eine Fundamental-Diskriminante mit $\mathsf{discSize}(t)$ Bits, die in zwei Primfaktoren zerlegbar ist.

Algorithmus 3.5 genDiscPQ

Eingabe: Der Sicherheitsparameter t.
Ausgabe: $F(\Delta)$.

1: $t_\Delta \leftarrow \mathsf{discSize}(t)$
2: $t_1 \leftarrow \lceil t_\Delta/2 \rceil$, $t_2 \leftarrow t_\Delta - t_1$
3: **repeat**
4: $\quad p \leftarrow \mathsf{oddRandomInteger}(2^{t_1})$
5: **until** $\mathsf{isPrime}(p)$ **and** $\mathrm{size}(p) = t_1$
6: **repeat**
7: $\quad q \leftarrow \mathsf{oddRandomInteger}(2^{t_2})$
8: **until** $\mathsf{isPrime}(q)$ **and** $\mathrm{size}(pq) = t_\Delta$ **and** $pq \equiv 3 \pmod 4$ **and** $\left(\frac{p}{q}\right) = -1$
9: **return** (p, q)

Beispiel: $\mathsf{genDiscPQ}(16)$ gibt z. B. $F(\Delta) = (451488\,619481,\ 476424\,851339)$ zurück, d. h. $\Delta = -215100\,398417\,485764\,335059$.

Verfahren zur Überprüfung einer Fundamental-Diskriminante

Das folgende Verfahren überprüft eine beliebige Zahl Δ, gegeben als Aufzählung von Primfaktoren $F(\Delta)$, daraufhin, ob Δ eine Fundamental-Diskriminante ist.

Algorithmus 3.6 checkDiscAny

Eingabe:

1. Die Faktorisierung $F(\Delta) = (p_1, \ldots, p_k)$ einer ungeraden Diskriminante Δ;
2. der Sicherheitsparameter t.

Ausgabe: valid oder invalid.

1: **for** $i = 1$ **to** k **do**
2: **if** $p_i < 0$ **or not** isPrime(p_i) **then return** invalid
3: **for** $j = i + 1$ **to** k **do**
4: **if** $p_i = p_j$ **then return** invalid
5: **end for**
6: **end for**
7: $\Delta \leftarrow -\prod_{i=1}^{k} p_i$.
8: **if** $\Delta \not\equiv 1 \pmod 4$ **then return** invalid
9: **if** size$(\Delta) \neq$ discSize(t) **then return** invalid
10: **return** valid

Im folgenden sehen wir wieder nur die beiden einfachsten Fälle vor:

Algorithmus 3.7 checkDiscP

Eingabe:

1. Die Faktorisierung einer Fundamental-Diskriminante $F(\Delta) = (p_1, \ldots, p_k)$;
2. und der Sicherheitsparameter t.

Ausgabe: valid oder invalid.

1: **if** $k \neq 1$ **then return** invalid
2: **if** checkDiscAny$(F(\Delta), t) =$ invalid **then return** invalid
3: **return** valid

Algorithmus 3.8 checkDiscPQ

Eingabe:

1. Die Faktorisierung einer Fundamental-Diskriminante $F(\Delta) = (p_1, \ldots, p_k)$;
2. und der Sicherheitsparameter t.

Ausgabe: valid oder invalid.

1: **if** $k \neq 2$ **then return** invalid
2: **if** checkDiscAny$(F(\Delta), t) =$ invalid **then return** invalid
3: **if** $\left(\frac{p_1}{p_2}\right) \neq -1$ **then return** invalid
4: **if** size$(p_1) \neq \lceil$discSize$(t)/2\rceil$ **then return** invalid
5: **return** valid

Algorithmus 3.9 checkDisc

Eingabe:

1. Die Faktorisierung einer Fundamental-Diskriminante $F(\Delta) = (p_1, \ldots, p_k)$
2. und der Sicherheitsparameter t.

Ausgabe: valid oder invalid.

1: **if** $k = 1$ **then return** checkDiscP($F(\Delta)$, t)
2: **if** $k = 2$ **then return** checkDiscPQ($F(\Delta)$, t)
3: **return** invalid

Verfahren zur Erzeugung eines zufälligen Gruppenelements

Ein zufälliges Gruppenelement aus $\mathrm{Cl}(\Delta)$ wählen wir, indem wir ein zufälliges, reduziertes $\mathcal{O}_\Delta$-Ideal wählen. Das folgende einfache Verfahren erzeugt ein zufälliges, reduziertes $\mathcal{O}_\Delta$-Ideal.

Algorithmus 3.10 randomIdeal

Eingabe:

1. Eine Diskriminante Δ;
2. Parameter B_k, B_p und B_e.

Ausgabe: Ein zufälliges, reduziertes $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}$.

1: $\mathfrak{a} \leftarrow (1, \Delta \bmod 2)$
2: **for** $k \leftarrow 1$ **to** B_k **do**
3: **repeat**
4: $p \leftarrow$ randomInteger(B_p)
5: **until** isPrime(p) **and** $\left(\frac{\Delta}{p}\right) = 1$
6: $\mathfrak{p} \leftarrow$ primeIdeal(Δ, p)
7: $e \leftarrow$ randomInteger(B_e)
8: $\mathfrak{a} \leftarrow$ reduce($\Delta, \mathfrak{a} \cdot \mathfrak{p}^e$)
9: **end for**
10: **return** $\mathfrak{a}$

Hier bezeichne $\left(\frac{\Delta}{p}\right)$ in Zeile 5 das Kronecker-Symbol. Dieses Verfahren ist für die Praxis ausreichend, wenn man z. B. $B_k = 1$, $B_p = 2^{16}$ und $B_e = 2^{2t}$ wählt (in diesem Fall kann die Suche nach einer Primzahl effizient in einfacher Genauigkeit durchgeführt werden). In Abschnitt 5.3 beschreiben wir einen alternativen Algorithmus, der bei geeigneter Wahl der Parameter ein zufälliges, reduziertes $\mathcal{O}_\Delta$-Ideal mit annähernder Gleichverteilung wählt.

Verfahren zur Überprüfung eines Gruppenelements

Gruppenelemente werden durch die normale Darstellung des reduzierten Vertreters dargestellt. Mit Überprüfung eines Gruppenelements aus $\mathrm{Cl}(\Delta)$ meinen wir die Prüfung, ob ein Paar (a, b) die normale Darstellung eines primitiven, reduzierten $\mathcal{O}_\Delta$-Ideals $\mathfrak{a} \neq \mathcal{O}_\Delta$ ist. Das

folgende Verfahren überprüft zu einer Diskriminante Δ und einem Paar (a, b), ob (a, b) ein $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}$ darstellt, und falls das der Fall ist, ob $\mathfrak{a}$ reduziert und $\mathfrak{a} \neq \mathcal{O}_\Delta$ ist.

Algorithmus 3.11 checkIdeal

Eingabe:

1. Die Diskriminante Δ eines imaginär-quadratischen Körpers;
2. ein Paar (a, b)

Ausgabe: valid oder invalid

1: **if** $4a \nmid b^2 - \Delta$ **then return** invalid
2: **if** $\gcd\big(a, b, (b^2 - \Delta)/4a\big) \neq 1$ **then return** invalid
3: **if not** isReduced$(\Delta, \mathfrak{a})$ **then return** invalid
4: **if** $a = 1$ **then return** invalid
5: **return** valid

(a, b) braucht nicht auf Primitivität geprüft zu werden (Zeile 2), wenn Δ eine ungerade Fundamental-Diskriminante ist: Falls (a, b) ein $\mathcal{O}_\Delta$-Ideal darstellt, und $d = \mathrm{ggT}(a, b, c)$ für $c = (b^2 - \Delta)/4a$ ist, dann $d^2 \mid \Delta$.

3.2.1 IQ-Bereichsparameter für IQ-DH, IQ-IES und IQ-DSA

Die IQ-Bereichsparameter für IQ-DH, IQ-IES und IQ-DSA sind gegeben durch ein Tripel

$$\mathcal{D}^{\text{IQ-DH}}, \mathcal{D}^{\text{IQ-DSA}}, \mathcal{D}^{\text{IQ-IES}} = \big(t, F(\Delta), \mathfrak{g}\big) \ . \tag{3.1}$$

t bezeichnet dabei einen Sicherheitsparameter, Δ bezeichnet die fundamentale Diskriminante eines imaginär-quadratischen Körpers, und $\mathfrak{g}$ ist ein reduziertes $\mathcal{O}_\Delta$-Ideal.

Erzeugung der Bereichsparameter für IQ-DH, IQ-DSA und IQ-IES

Das folgende Verfahren wird zur Erzeugung von Bereichsparametern für IQ-DH verwendet.

Algorithmus 3.12 genDomainIQDH

Eingabe: Der Sicherheitsparameter t.
Ausgabe: Die Bereichsparameter $\mathcal{D}^{\text{IQ-DH}} = \big(t, F(\Delta), \mathfrak{g}\big)$.

1: Wähle $F(\Delta)$ und damit Δ (genDiscP oder genDiscPQ)
2: $\mathfrak{g} \leftarrow$ randomIdeal(Δ)
3: **return** $(t, F(\Delta), \mathfrak{g})$

Beispiel: Für $t = 16$ gibt genDomainIQDH z. B.

$$\Big(\underbrace{16}_{t}, \ \underbrace{(156211\,589831\,799264\,009011)}_{F(\Delta)}, \ \underbrace{(92154\,358865, \ -91190\,004293)}_{\mathfrak{g}}\Big)$$

aus.

Dieses Verfahren wird analog für die Erzeugung von Bereichsparametern für IQ-DSA (genDomainIQDSA) oder IQ-IES (genDomainIQIES) verwendet.

Überprüfung der Bereichsparameter für IQ-DH, IQ-DSA und IQ-IES

Das folgende Verfahren wird zur Überprüfung von Bereichsparametern für IQ-DH verwendet.

Algorithmus 3.13 checkDomainIQDH

Eingabe: Die Bereichsparameter $\mathcal{D}^{\text{IQ-DH}} = \big(t, F(\Delta), \mathfrak{g}\big)$;
Ausgabe: valid oder invalid.

1: **if** checkDisc$(F(\Delta), t)$ = invalid **then return** invalid
2: **if** checkIdeal$(\Delta, \mathfrak{g})$ = invalid **then return** invalid
3: **return** valid

Dieses Verfahren wird analog für die Überprüfung von Bereichsparametern für IQ-DSA (checkDomainIQDSA) oder IQ-IES (checkDomainIQIES) verwendet.

3.2.2 IQ-Bereichsparameter für IQ-RDSA

Die IQ-Bereichsparameter für IQ-RDSA sind gegeben durch ein Quadrupel

$$\mathcal{D}^{\text{IQ-RDSA}} = \big(t, F(\Delta), \mathfrak{g}, q\big) \ . \tag{3.2}$$

t bezeichnet dabei einen Sicherheitsparameter, Δ bezeichnet die fundamentale Diskriminante eines imaginär-quadratischen Körpers, $\mathfrak{g}$ ist ein reduziertes $\mathcal{O}_\Delta$-Ideal, und q bezeichnet eine Primzahl mit $1 < q < 2^{2t}$.

Erzeugung der IQ-RDSA Bereichsparameter

Das folgende Verfahren wird zur Erzeugung von Bereichsparametern für IQ-RDSA verwendet.

Algorithmus 3.14 genDomainIQRDSA

Eingabe: Der Sicherheitsparameter t.
Ausgabe: Die Bereichsparameter $\mathcal{D}^{\text{IQ-RDSA}} = (t, F(\Delta), \mathfrak{g}, q)$.

1: Wähle $F(\Delta)$ und damit Δ (genDiscP(t) oder genDiscPQ(t))
2: $\mathfrak{g} \leftarrow$ randomIdeal(Δ)
3: **repeat**
4: $\quad q \leftarrow$ oddRandomInteger(2^{2t})
5: **until** isPrime(q)
6: **return** $(t, F(\Delta), \mathfrak{g}, q)$

Beispiel: Für $t = 16$ gibt genDomainIQRDSA z. B.

$$\Big(\underbrace{16}_{t},\ \underbrace{(156211\,589831\,799264\,009011)}_{F(\Delta)},\ \underbrace{(92154\,358865,\ -91190\,004293)}_{\mathfrak{g}},\ \underbrace{57487}_{q}\Big)$$

aus.

Verfahren zur Überprüfung der IQ-RDSA Bereichsparameter

Das folgende Verfahren wird zur Überprüfung von Bereichsparametern für IQ-DH verwendet.

Algorithmus 3.15 checkDomainIQRDSA

Eingabe: Die Bereichsparameter $\mathcal{D}^{\text{IQ-RDSA}} = (t, F(\Delta), \mathfrak{g}, q)$;
Ausgabe: valid oder invalid.

1: **if** checkDisc$(F(\Delta), t) =$ invalid **then return** invalid
2: **if** checkIdeal$(\Delta, \mathfrak{g}) =$ invalid **then return** invalid
3: **if not** isPrime(q) **or** $q \geqslant 2^{2t}$ **then return** invalid
4: **return** valid

3.2.3 IQ-Bereichsparameter für IQ-GQ

Die IQ-Bereichsparameter für IQ-GQ sind gegeben durch

$$\mathcal{D}^{\text{IQ-GQ}} = \big(t, F(\Delta), n\big) \ . \tag{3.3}$$

t bezeichnet dabei einen Sicherheitsparameter, Δ bezeichnet die fundamentale Diskriminante eines imaginär-quadratischen Körpers, und n eine ganze Zahl mit $1 \leqslant n < 2^{2t}$ ist.

Erzeugung der IQ-GQ Bereichsparameter

Das folgende Verfahren wird zur Erzeugung von Bereichsparametern für IQ-GQ verwendet.

Algorithmus 3.16 genDomainIQGQ

Eingabe: Der Sicherheitsparameter t.
Ausgabe: Die Bereichsparameter $\mathcal{D}^{\text{IQ-GQ}} = (t, F(\Delta), n)$.

1: Wähle $F(\Delta)$ und damit Δ (genDiscP(t) oder genDiscPQ(t))
2: $n \leftarrow$ randomInteger(2^{2t})
3: **return** $(t, F(\Delta), n)$

Beispiel: Für $t = 16$ gibt genDomainIQDH z. B.

$$\Big(\underbrace{16}_{t},\ \underbrace{(156211\,589831\,799264\,009011)}_{F(\Delta)},\ \underbrace{53681}_{n}\Big)$$

aus.

Verfahren zur Überprüfung der IQ-GQ Bereichsparameter

Das folgende Verfahren wird zur Überprüfung von Bereichsparametern für IQ-GQ verwendet.

Algorithmus 3.17 checkDomainIQGQ

Eingabe: Die Bereichsparameter $\mathcal{D}^{\text{IQ-GQ}} = (t, F(\Delta), n)$;
Ausgabe: valid oder invalid.

1: **if** checkDisc$(F(\Delta), t) =$ invalid **then return** invalid
2: **if** $n \geqslant 2^{2t}$ **then return** invalid
3: **return** valid

3.3 IQ-Schlüsselpaare

In diesem Abschnitt beschreiben wir die Schlüsselpaare für die IQ-Kryptoverfahren. Wir beschreiben das Format der Schlüsselpaare und geben Methoden zur Erzeugung von Schlüsselpaaren und zur Überprüfung von öffentlichen Schlüsseln an.

Ein Schlüsselpaar $\mathcal{K}$ ist ein Paar aus öffentlichem und privatem Schlüssel

$$\mathcal{K} = (\mathcal{K}_{\mathsf{pub}}, \mathcal{K}_{\mathsf{prv}}) \ . \tag{3.4}$$

Wir unterscheiden zwischen Schlüsselpaaren für IQ-DH, IQ-DSA, IQ-RDSA, IQ-GQ und IQ-IES; diese werden jeweils mit $\mathcal{K}^{\text{IQ-DH}}$, $\mathcal{K}^{\text{IQ-DSA}}$, $\mathcal{K}^{\text{IQ-RDSA}}$, $\mathcal{K}^{\text{IQ-GQ}}$ bzw. $\mathcal{K}^{\text{IQ-IES}}$ bezeichnet. Die öffentlichen und privaten Schlüssel für IQ-DH, IQ-DSA, IQ-RDSA und IQ-IES haben eine identische Struktur.

3.3.1 Schlüsselpaare für IQ-DH, IQ-DSA, IQ-RDSA und IQ-IES

Gegeben ein Bereichsparameter $\mathcal{D}$ ($\mathcal{D}^{\text{IQ-DH}}$, $\mathcal{D}^{\text{IQ-DSA}}$, $\mathcal{D}^{\text{IQ-IES}}$ oder $\mathcal{D}^{\text{IQ-RDSA}}$), dann sind $\mathcal{K}^{\text{IQ-DH}}$, $\mathcal{K}^{\text{IQ-DSA}}$, $\mathcal{K}^{\text{IQ-RDSA}}$ und $\mathcal{K}^{\text{IQ-IES}}$ jeweils Paare $(\mathfrak{a}, a)$, wobei $\mathfrak{a}$ ein reduziertes $\mathcal{O}_\Delta$-Ideal, a eine ganze Zahl mit $1 \leqslant a < 2^{2t}$ und $\mathfrak{a}$ äquivalent zu $\mathfrak{g}^a$ ist.

Verfahren zur Erzeugung von Schlüsselpaaren für IQ-DH, IQ-DSA, IQ-RDSA und IQ-IES

Das folgende Verfahren erzeugt ein Schlüsselpaar für IQ-DH.

Algorithmus 3.18 genKeyPairIQDH

Eingabe: Die Bereichsparameter $\mathcal{D}^{\text{IQ-DH}} = (t, F(\Delta), \mathfrak{g})$.
Ausgabe: Das IQ-DH-Schlüsselpaar $\mathcal{K}^{\text{IQ-DH}} = (\mathcal{K}^{\text{IQ-DH}}_{\mathsf{pub}}, \mathcal{K}^{\text{IQ-DH}}_{\mathsf{prv}})$.

1: $a \leftarrow$ randomInteger(2^{2t}).
2: $\mathfrak{a} \leftarrow$ reduce$(\Delta, \mathfrak{g}^a)$.
3: **return** $(\mathfrak{a}, a)$

Beispiel: Mit $\mathcal{D}^{\text{IQ-DH}}$ wie im Beispiel zuvor gibt genKeyPairIQDH z. B.

$$\Big(\underbrace{(160769\,806233,\ -138060\,435995)}_{\mathcal{K}^{\text{IQ-DH}}_{\mathsf{pub}}},\ \underbrace{55927}_{\mathcal{K}^{\text{IQ-DH}}_{\mathsf{prv}}}\Big)$$

aus.

Dieses Verfahren wird ganz analog zur Erzeugung von Schlüsselpaaren für IQ-DSA (genKeyPairIQDSA), IQ-RDSA (genKeyPairIQRDSA) und IQ-IES (genKeyPairIQIES) verwendet.

Verfahren zur Überprüfung von öffentlichen Schlüsseln für IQ-DH, IQ-DSA, IQ-RDSA und IQ-IES

Das folgende Verfahren überprüft einen öffentlichen Schlüssel für IQ-DH.

Algorithmus 3.19 checkPubKeyIQDH

Eingabe:

1. Die Bereichsparameter $\mathcal{D}^{\text{IQ-DH}} = (t, F(\Delta), \mathfrak{g})$;
2. einen öffentlichen IQ-DH-Schlüssel $\mathcal{K}_{\mathsf{pub}}^{\text{IQ-DH}} = \mathfrak{a}$.

Ausgabe: valid oder invalid.

1: **if** checkIdeal$(\Delta, \mathfrak{a})$ = invalid **then return** invalid
2: **return** valid

Dieses Verfahren wird ganz analog zur Überprüfung von öffentlichen Schlüsseln für IQ-DSA (checkPubKeyIQDSA), IQ-RDSA (checkPubKeyIQRDSA) und IQ-IES (checkPubKeyIQIES) verwendet.

3.3.2 Schlüsselpaare für IQ-GQ

Gegeben ein Bereichsparameter $\mathcal{D}^{\text{IQ-GQ}} = \big(t, F(\Delta), n\big)$, dann ist $\mathcal{K} = \mathcal{K}^{\text{IQ-GQ}}$ das Paar $(\mathfrak{n}, \mathfrak{a})$, wobei $\mathfrak{a}$ ein reduziertes $\mathcal{O}_\Delta$-Ideal und $\mathfrak{n}$ das reduzierte Ideal äquivalent zu $\mathfrak{a}^n$ ist.

Verfahren zur Erzeugung von IQ-GQ Schlüsselpaaren

Das folgende Verfahren erzeugt ein Schlüsselpaar für IQ-GQ.

Algorithmus 3.20 genKeyPairIQGQ

Eingabe: Die Bereichsparameter $\mathcal{D} = (t, F(\Delta), n)$.
Ausgabe: Das IQ-GQ-Schlüsselpaar $\mathcal{K}^{\text{IQ-GQ}} = (\mathcal{K}_{\mathsf{pub}}^{\text{IQ-GQ}}, \mathcal{K}_{\mathsf{prv}}^{\text{IQ-GQ}})$.

1: $\mathfrak{a} \leftarrow$ randomIdeal(Δ)
2: $\mathfrak{n} \leftarrow$ reduce$(\Delta, \mathfrak{a}^n)$.
3: **return** $(\mathfrak{n}, \mathfrak{a})$

Beispiel: Mit $\mathcal{D}^{\text{IQ-GQ}}$ wie im Beispiel zuvor gibt genKeyPairIQGQ z. B.

$$\Big(\underbrace{(187183\ 795289, 135134\ 309781)}_{\mathcal{K}_{\mathsf{pub}}^{\text{IQ-GQ}}},\ \underbrace{(145395\ 683831,\ 89456\ 467535)}_{\mathcal{K}_{\mathsf{prv}}^{\text{IQ-GQ}}}\Big)$$

aus.

Verfahren zur Überprüfung von öffentlichen IQ-GQ-Schlüsseln

Das folgende Verfahren überprüft einen öffentlichen Schlüssel für IQ-GQ.

Algorithmus 3.21 checkPubKeyIQGQ

Eingabe:

1. Die Bereichsparameter $\mathcal{D} = (t, F(\Delta), n)$;
2. einen öffentlichen IQ-GQ-Schlüssel $\mathcal{K}_{\mathsf{pub}}^{\text{IQ-GQ}} = \mathfrak{n}$.

Ausgabe: valid oder invalid.

1: **if** checkIdeal$(\Delta, \mathfrak{n}) =$ invalid **then return** invalid
2: **return** valid

3.4 Das IQ-Diffie-Hellman-Primitiv

in diesem Abschnitt beschreiben wir das Diffie-Hellman-Primitiv für Klassengruppen. Diese Komponente wird in Kapitel 4 lediglich für asymmetrische Verschlüsselungen (IQ-IES) und Schlüsselaustausch (IQ-DH) benötigt.

A sollte folgendes tun, um einen gemeinsamen geheimen Wert mit **B** mit dem IQ-DH-Primitiv zu berechnen:

Algorithmus 3.22 IQDH

Eingabe:

1. Gültige Bereichsparameter $\mathcal{D} = \big(t, F(\Delta), \mathfrak{g}\big)$;
2. ein privater IQ-DH-Schlüssel $a_{\mathbf{A}}$, der zu **A** gehört;
3. ein gültiger öffentlicher IQ-DH-Schlüssel $\mathfrak{a}_{\mathbf{B}}$, der mutmaßlich zu **B** gehört.

Ausgabe: Ein geheimes $\mathcal{O}_\Delta$-Ideal $\mathfrak{k}$ oder invalid

1: $\mathfrak{k} \leftarrow$ reduce$(\Delta, \mathfrak{a}_{\mathbf{B}}^{a_{\mathbf{A}}})$.
2: **if** $a_{\mathfrak{k}} = 1$ **then return** invalid
3: **return** $\mathfrak{k}$

3.5 Weitere Komponenten

In diesem Abschnitt beschreiben wir schematisch einige weitere, symmetrische kryptographische Komponenten, die für die IQ-Kryptoverfahren benötigt werden. Wir beschreiben im einzelnen schematisch Hash-Funktionen (für Signaturverfahren), sowie Message-Authentication-Codes und symmetrische Verschlüsselungen (für asymmetrische Verschlüsselungsverfahren).

3.5.1 Kryptographische Hash-Funktionen

Für unsere Anwendungen definieren wir eine Hash-Funktion *hash* als Abbildung von Octet-Strings mit einer maximalen Länge von *hashmaxlen* zu Octet-Strings der Länge *hashlen*. *hash* wird als *kryptographisch* bezeichnet, wenn sie einweg und kollisionsresistent ist (siehe [51, Abschnitt 9.2.2])

Eine solche kryptographische Hash-Funktion ist z. B. SHA-1[1], mit $hashmaxlen = 2^{60}$ und $hashlen = 20$.

Wir werden in dieser Arbeit das folgende generische Verfahren verwenden, welches für eine Anwendung mit *hash*, *hashmaxlen* und *hashlen* instanziiert werden muß.

Algorithmus 3.23 hash

Eingabe: Ein Octet-String $M = M_0 M_1 \dots M_{m-1}$ der Länge m.
Ausgabe: Ein Octet-String $H = H_0 H_1 \dots H_{hashlen-1}$ oder invalid.

1: **if** $m > hashmaxlen$ **then return** invalid
2: $H \leftarrow hash(M)$
3: **return** H

Das folgende Verfahren bildet einen Hash-Wert auf eine ganze Zahl $< 2^\ell$ ab. Dieses Verfahren benötigen wir für die Signaturverfahren in Kapitel 4.

Algorithmus 3.24 hashToInteger

Eingabe:

1. Ein Octet-String $H = H_0 H_1 \dots H_{hashlen-1}$ der Länge der Länge *hashlen*;
2. eine ganze positive Zahl ℓ.

Ausgabe: Eine Zahl n, $0 \leqslant n < 2^\ell$.

1: $\mathbf{h} \leftarrow \mathsf{octetsToBits}(H)$
2: // $\mathbf{h} = h_0 h_1 \dots h_{8 \cdot hashlen-1}$
3: **if** $\ell \geqslant 8 \cdot hashlen$ **then** $\mathbf{n} \leftarrow \mathbf{h}$
4: **else** $\mathbf{n} \leftarrow h_0 h_1 \dots h_{\ell-1}$
5: $N \leftarrow \mathsf{bitsToOctets}(\mathbf{n})$
6: $n \leftarrow \mathsf{octetsToInteger}(N)$
7: **return** n

3.5.2 Schlüsselableitung

Eine Funktion *derive* zur Schlüsselableitung leitet einen Schlüssel aus einem beliebigen Octet-String ab. Üblicherweise geschieht dies unter Verwendung einer kryptographischen Hash-Funktion. Ein Beispiel für eine Funktion zur Schlüsselableitung ist im ANSI X9.63 Standard ausgeführt [3].

In dieser Arbeit wird das folgende generische Verfahren angewendet, welches für eine Anwendung mit *derive* und *keydatalen* instanziiert werden muß.

[1] SHA: Secure Hash Algorithm

Algorithmus 3.25 deriveKey

Eingabe:

1. Ein Octet-String $S = S_0 S_1 \dots S_{s-1}$ der Länge s;
2. eine positive Zahl *keydatalen*, welche die gewünschte Größe der Ausgabe bezeichnet.
3. Ggf. noch weitere Daten (siehe z. B. [3]).

Ausgabe: Ein Octet-String $K = K_0 K_1 \dots K_{keydatalen-1}$ oder invalid.

1: $K \leftarrow deriveKey(S, keydatalen, \dots)$
2: **return** K

3.5.3 Message-Authentication-Codes

Ein Message-Authentication-Code (MAC) *MAC* ist eine kryptographische Hash-Funktion, die in Verbindung mit einem geheimen symmetrischen Schlüssel verwendet wird (siehe [51, Abschnitt 9.5]). Die Ausgabe eines MACs wird als *Tag* und die Anwendung eines MACs auf einen Text als *Tagging* bezeichnet. Die Verwendung eines geheimen symmetrischen Schlüssels soll bewirken, daß nur legitimierte Benutzer gültige Paare von Texten und Tags erzeugen können. MACs werden im Zusammenhang mit Public-Key-Verschlüsselungsverfahren eingesetzt, um semantische Sicherheit zu erlangen. Genau wie gewöhnliche Hash-Funktionen operieren MACs auf Bit-Strings und geben den Tag als Bit-String zurück. Ein Beispiel für einen MAC ist HMAC–SHA-1–160 mit einem 160-Bit Schlüssel und 160-Bit Tag.

In dieser Arbeit wird das folgende generische Verfahren angewendet, welches für eine Anwendung mit *MAC* und *mackeylen* und *maclen* instanziiert werden muß.

Algorithmus 3.26 MAC

Eingabe:

1. Ein Octet-String $M = M_0 M_1 \dots M_{m-1}$ der Länge m;
2. einen Schlüssel, gegeben als Octet-String $K = K_0 K_1 \dots K_{mackeylen} - 1$ der Länge *mackeylen*.

Ausgabe: Ein Octet-String $T = T_0 T_1 \dots T_{maclen-1}$ oder invalid.

1: $\mathbf{m} \leftarrow$ octetsToBits(M)
2: $\mathbf{k} \leftarrow$ octetsToBits(K)
3: $\mathbf{t} \leftarrow MAC(\mathbf{m}, \mathbf{k})$
4: **if** $\mathbf{t}$ = invalid **then return** invalid
5: $T \leftarrow$ bitsToOctets$(\mathbf{t})$
6: **return** T

3.5.4 Symmetrische Verschlüsselungen

Symmetrische Verschlüsselungsverfahren werden im Zusammenhang mit Public-Key-Verschlüsselungen eingesetzt, um semantische Sicherheit zu erlangen. Es kann z. B. AES[2] [26] verwendet werden; aufgrund des speziellen Einsatz-Zwecks käme für IQ-IES (genau wie bei EC-IES) aber auch eine einfache XOR-Verschlüsselung in Frage.

In dieser Arbeit werden für die symmetrische Verschlüsselung bzw. Entschlüsselung die folgenden generischen Verfahren angewendet, welche für eine Anwendung mit ENC bzw. ENC^{-1} und *enckeylen* instanziiert werden müssen.

Algorithmus 3.27 ENC

Eingabe:

1. Ein Octet-String $M = M_0 M_1 \ldots$;
2. einen Schlüssel, gegeben als Octet-String $K = K_0 K_1 \ldots K_{enckeylen} - 1$ der Länge *enckeylen*.

Ausgabe: Ein Octet-String $C = C_0 C_1 \ldots$ oder invalid.

1: $\mathbf{m} \leftarrow \mathsf{octetsToBits}(M)$
2: $\mathbf{k} \leftarrow \mathsf{octetsToBits}(K)$
3: $\mathbf{c} \leftarrow ENC(\mathbf{m}, \mathbf{k})$
4: **if** $\mathbf{c} =$ invalid **then return** invalid
5: $C \leftarrow \mathsf{bitsToOctets}(\mathbf{c})$
6: **return** C

Algorithmus 3.28 ENC^{-1}

Eingabe:

1. Ein Octet-String $C = C_0 C_1 \ldots$;
2. einen Schlüssel, gegeben als Octet-String $K = K_0 K_1 \ldots K_{enckeylen} - 1$ der Länge *enckeylen*.

Ausgabe: Ein Octet-String $M = M_0 M_1 \ldots$ oder invalid.

1: $\mathbf{c} \leftarrow \mathsf{octetsToBits}(C)$
2: $\mathbf{k} \leftarrow \mathsf{octetsToBits}(K)$
3: $\mathbf{m} \leftarrow ENC^{-1}(\mathbf{c}, \mathbf{k})$
4: **if** $\mathbf{m} =$ invalid **then return** invalid
5: $M \leftarrow \mathsf{bitsToOctets}(\mathbf{m})$
6: **return** M

[2] AES: Advanced Encryption Standard

Kapitel 4

IQ-Kryptoverfahren

In diesem Kapitel beschreiben wir verschiedene Kryptoverfahren für Klassengruppen imaginär-quadratischer Zahlkörper (IQ-Kryptoverfahren). Ein Kryptoverfahren oder auch kryptographisches Protokoll besteht aus mehreren Einzelverfahren, die von den teilnehmenden Parteien angewendet werden, um eine gemeinsame Berechnung durchzuführen. Die einzelnen Verfahren wiederum bauen auf verschiedenen einfacheren Komponenten oder auch Primitiven auf, wie z. B. Methoden zur Erzeugung und Überprüfung von Bereichsparametern und Schlüsselpaaren, usw. Diese Komponenten sind in Kapitel 3 detailliert beschrieben worden. Die Beschreibung in diesem Kapitel erfolgt ebenfalls in Anlehnung an den Standard für elliptische Kurven SEC 1 [71, Kapitel 4–6]. In diesem Kapitel geben wir lediglich eine Beschreibung der IQ-Kryptoverfahren wieder; nähere Details zur Sicherheit und zur Motivation der Signaturverfahren stehen in Kapitel 6.

Im einzelnen beschreiben wir in Abschnitt 4.1 drei Signaturverfahren, die mit Klassengruppen verwendet werden können. Es handelt sich hierbei um IQ-DSA[1], IQ-GQ[2] und IQ-RDSA[3]. DSA an sich ist nicht ohne weiteres auf Klassengruppen übertragbar, und IQ-RDSA und IQ-DSA sind Varianten des traditionellen DSA-Signaturverfahrens, die stark davon abweichen, während IQ-GQ hingegen eine einfache Übertragung des GQ-Signaturverfahrens [30] auf Klassengruppen ist. Das IQ-RDSA-Signaturverfahren ist Gegenstand unserer Arbeit in [8]. Zu jedem dieser Signaturverfahren beschreiben wir, welche Schritte die beteiligten Parteien durchführen müssen, um das jeweilige Signaturverfahren verwenden zu können, und wir geben jeweils die Verfahren zur Erzeugung und Überprüfung einer Signatur an.

In Abschnitt 4.2 beschreiben wir die Variante für Klassengruppen des IES[4]-Verschlüsselungsverfahren (auch als DHIES bekannt) nach [1]. Dieses Verschlüsselungsverfahren kann ohne weiteres auf Klassengruppen übertragen werden. Wir beschreiben, welche Schritte die beteiligten Parteien durchführen müssen, um IQ-IES verwenden zu können, und wir geben Verfahren zur Verschlüsselung und Entschlüsselung von Texten an.

In Abschnitt 4.3 schließlich beschreiben wir IQ-DH[5], das klassische Diffie-Hellman-Verfahren zum Schlüsselaustausch nach [24] mit Klassengruppen. Auch dieses Kryptover-

[1] DSA: Digital Signature Algorithm
[2] GQ: Guillou-Quisquater
[3] RDSA: Root-DSA, DSA-Variante, deren Sicherheit auf der Schwierigkeit des Wurzelproblems beruht
[4] IES: Integrated Encryption Scheme
[5] DH: Diffie-Hellman

fahren kann ohne weiteres auf Klassengruppen übertragen werden. Wir beschreiben, welche Schritte die beteiligten Parteien durchführen müssen, um IQ-DH verwenden zu können, und wir geben ein Verfahren zur Berechnung eines gemeinsamen Geheimnisses an.

4.1 Signaturverfahren

Dieser Abschnitt beschreibt die Signaturverfahren IQ-RDSA, IQ-DSA und IQ-GQ. Wir gehen dabei von dem Szenario aus, daß eine Partei **A** einen Text M signieren möchte, während eine Partei **B** die Signatur $\mathcal{S}_{\mathbf{A},M}$ von **A** zu dem signierten Text M überprüfen möchte. Zu jedem dieser Signaturverfahren beschreiben wir die notwendigen Schritte für **A** und **B**, um das jeweilige Signaturverfahren zu verwenden, sowie die Verfahren zur Erzeugung und Überprüfung (Verifikation) von Signaturen.

4.1.1 IQ-RDSA

Wir beschreiben nun IQ-RDSA[6]. IQ-RDSA ist eine spezielle DSA-Variante für die Verwendung mit Klassengruppen. Bei traditionellem DSA, wie auch bei allen anderen Signaturverfahren von ElGamal-Typ, werden Exponenten von Gruppenelementen modulo der Gruppenordnung reduziert. Da die Klassenzahlen für große Fundamental-Diskriminanten im allgemeinen nicht effizient berechenbar sind, können Signaturverfahren vom ElGamal-Typ nicht ohne weiteres mit Klassengruppen verwendet werden. In [8] haben wir mit dem RDSA-Signaturverfahren einen Vorschlag für ein modifiziertes DSA-Signaturverfahren gemacht, das in endlichen Gruppen mit unbekannter Gruppenordnung angewendet werden kann.

Eine schematische Darstellung von RDSA für beliebige endliche Gruppen mit unbekannter Gruppenordnung, sowie Sicherheitsbeweise für RDSA stehen in Abschnitt 6.3. In diesem Abschnitt beschreiben wir die Spezifikation für IQ-RDSA.

Vorbereitung

A sollte die folgenden Schritte unternehmen, um die Benutzung von IQ-RDSA vorzubereiten:

1. Lege eine kryptographische Hash-Funktion $hash_{\mathbf{A}}$ sowie die Parameter *hashlen* und *hashmaxlen* für die Verwendung in der Prozedur hash fest.
2. Lege Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-RDSA}} = (t, F(\Delta), \mathfrak{g}, q)$ in Abhängigkeit des gewünschten Sicherheitsparameters t fest (genDomainIQRDSA), und überprüfe die Bereichsparameter (checkDomainIQRDSA).
3. Lege ein Schlüsselpaar $\mathcal{K}_{\mathbf{A}}^{\text{IQ-RDSA}} = (\mathfrak{a}, a)$ fest, welches bezüglich der Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-RDSA}}$ ausgewählt wurde (genKeyPairIQRDSA).

B sollte die folgenden Schritte unternehmen, um die Benutzung von IQ-RDSA vorzubereiten:

1. Beschaffe **A**s authentische Hash-Funktion $hash_{\mathbf{A}}$ und deren Parameter *hashlen* und *hashmaxlen*.

[6] RDSA: Root-DSA, DSA-Variante, deren Sicherheit auf der Schwierigkeit des Wurzelproblem beruht

2. Beschaffe **A**s authentische Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-RDSA}}$, und überprüfe die Bereichsparameter (checkDomainIQRDSA).

3. Beschaffe **A**s authentischen öffentlichen Schlüssel $\mathcal{K}_{\mathbf{A},\mathsf{pub}}^{\text{IQ-RDSA}} = \mathfrak{a}$, und überprüfe den öffentlichen Schlüssel (checkPubKeyIQRDSA).

Signatur

Mit dem folgenden Verfahren sollte **A** eine IQ-RDSA-Signatur für einen Text M, erzeugen, wobei die Bereichsparameter und die Schlüssel aus der Vorbereitung verwendet werden sollen. Die Signatur bezeichnen wir mit $\mathcal{S}_{\mathbf{A},M}^{\text{IQ-RDSA}}$.

Algorithmus 4.1 IQ-RDSA Signatur

Eingabe:

1. Die Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-RDSA}} = (t, F(\Delta), \mathfrak{g}, q)$;
2. den privaten IQ-RDSA-Schlüssel $\mathcal{K}_{\mathbf{A},\mathsf{prv}}^{\text{IQ-RDSA}} = a$;
3. ein Octet-String $M = M_0 M_1 \ldots$.

Ausgabe: Eine Signatur $\mathcal{S}_{\mathbf{A},M}^{\text{IQ-RDSA}} = (s, \mathfrak{r}, \mathfrak{l})$ zu M oder invalid

1: $(\mathfrak{r}, k) \leftarrow \mathsf{genKeyPairIQRDSA}(\mathcal{D}_{\mathbf{A}}^{\text{IQ-RDSA}})$
2: $R \leftarrow \mathsf{idealToOctets}(\mathfrak{r})$
3: $H \leftarrow \mathsf{hash}(M \| R)$
4: **if** $H = \mathsf{invalid}$ **then return** invalid
5: $h \leftarrow \mathsf{hashToInteger}(H, 2t)$
6: $x \leftarrow -ah + k$
7: $s \leftarrow x \bmod q$, $\ell \leftarrow \lfloor x/q \rfloor$
8: $\mathfrak{l} \leftarrow \mathsf{reduce}(\Delta, \mathfrak{g}^{\ell})$
9: **return** $(s, \mathfrak{r}, \mathfrak{l})$

Verifikation

Mit dem folgenden Verfahren sollte **B** die IQ-RDSA-Signatur $\mathcal{S}_{\mathbf{A},M}^{\text{IQ-RDSA}}$ von **A** für den Text M überprüfen. Proposition 6.3.2 in Abschnitt 6.3 zeigt, daß die Verifikation für jede Signatur erfolgreich ist, die mit Algorithmus 4.1 erzeugt wurde.

Algorithmus 4.2 IQ-RDSA Verifikation

Eingabe:

1. Die Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-RDSA}} = (t, F(\Delta), \mathfrak{g}, q)$;
2. den öffentlichen IQ-RDSA-Schlüssel $\mathcal{K}_{\mathbf{A},\mathsf{pub}}^{\text{IQ-RDSA}} = \mathfrak{a}$;
3. ein Octet-String $M = M_0 M_1 \ldots$;
4. die Signatur $\mathcal{S}_{\mathbf{A},M}^{\text{IQ-RDSA}} = (s, \mathfrak{r}, \mathfrak{l})$.

Ausgabe: valid oder invalid

1: **if not** $0 \leqslant s < q$ **then return** invalid
2: **if** checkIdeal$(\Delta, \mathfrak{r}) =$ invalid **then return** invalid
3: **if** checkIdeal$(\Delta, \mathfrak{l}) =$ invalid **then return** invalid
4: $R \leftarrow$ idealToOctets$(\mathfrak{r})$
5: $H \leftarrow$ hash$(M \| R)$
6: **if** $H =$ invalid **then return** invalid
7: $h \leftarrow$ hashToInteger$(H, 2t)$
8: $\mathfrak{v} \leftarrow$ reduce$(\mathfrak{g}^s \mathfrak{a}^h \mathfrak{l}^q)$
9: **if** $\mathfrak{v} \neq \mathfrak{r}$ **then return** invalid
10: **return** valid

4.1.2 IQ-DSA

Wir beschreiben nun IQ-DSA[7]. IQ-DSA ist wie IQ-RDSA eine DSA-Variante für Klassengruppen, bei der man ohne die Kenntnis der Klassenzahl auskommt. Die Bezeichnung IQ-DSA ist insofern irreführend, als daß es sich hierbei nicht einfach um eine Übertragung von traditionellem DSA auf Klassengruppen handelt. Tatsächlich ist das IQ-DSA-Verfahren mehr dem Schnorr-Signaturverfahren [65] ähnlich, wir bezeichnen es dennoch als IQ-DSA. IQ-DSA folgt einer Idee aus [59], bei der tatsächlich das Schnorr-Signaturverfahren so modifiziert wurde, so daß man ohne die Reduktion der Exponenten modulo einer Gruppenordnung auskommt.

Eine schematische Darstellung von DSA für beliebige endliche Gruppen mit unbekannter Gruppenordnung haben wir in Abschnitt 6.4 gegeben; Beweise zur Sicherheit für das Verfahren aus [59] stehen dort. In diesem Abschnitt beschreiben wir die Spezifikation für IQ-DSA.

Vorbereitung

A sollte die folgenden Schritte unternehmen, um die Benutzung von IQ-DSA vorzubereiten:

1. Lege eine kryptographische Hash-Funktion $hash_{\mathbf{A}}$ sowie die Parameter *hashlen* und *hashmaxlen* für die Verwendung in der Prozedur hash fest.
2. Lege Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-DSA}} = (t, F(\Delta), \mathfrak{g})$ in Abhängigkeit des gewünschten Sicherheitsparameters t fest (genDomainIQDSA), und überprüfe die Bereichsparameter (checkDomainIQDSA).

[7] DSA: Digital Signature Algorithm

3. Lege ein Schlüsselpaar $\mathcal{K}_{\mathbf{A}}^{\text{IQ-DSA}} = (\mathfrak{a}, a)$ fest, welches bezüglich der Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-DSA}}$ ausgewählt wurde (genKeyPairIQDSA).

B sollte die folgenden Schritte unternehmen, um die Benutzung von IQ-DSA vorzubereiten:

1. Beschaffe **A**s authentische Hash-Funktion $hash_{\mathbf{A}}$ und deren Parameter *hashlen* und *hashmaxlen*.

2. Beschaffe **A**s authentische Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-DSA}}$, und überprüfe die bereichsparameter (checkDomainIQDSA).

3. Beschaffe **A**s authentischen öffentlichen Schlüssel $\mathcal{K}_{\mathbf{A},\mathsf{pub}}^{\text{IQ-DSA}} = \mathfrak{a}$, und überprüfe den öffentlichen Schlüssel (checkPubKeyIQDSA).

Signatur

Mit dem folgenden Verfahren sollte **A** eine IQ-DSA-Signatur für einen Text M erzeugen, wobei die Bereichsparameter und die Schlüssel aus der Vorbereitung verwendet werden sollen. Die Signatur bezeichnen wir mit $\mathcal{S}_{\mathbf{A},M}^{\text{IQ-DSA}}$.

Algorithmus 4.3 IQ-DSA Signatur

Eingabe:

1. Die Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-DSA}} = (t, F(\Delta), \mathfrak{g})$;
2. den privaten IQ-DSA-Schlüssel $\mathcal{K}_{\mathbf{A},\mathsf{prv}}^{\text{IQ-DSA}} = a$;
3. ein Octet-String $M = M_0 M_1 \ldots$.

Ausgabe: Eine Signatur $\mathcal{S}_{\mathbf{A},M}^{\text{IQ-DSA}} = (s, \mathfrak{r})$ zu M oder invalid

1: $k \leftarrow \mathsf{randomInteger}(2^{5t})$
2: $\mathfrak{r} \leftarrow \mathsf{reduce}(\Delta, \mathfrak{g}^k)$
3: $R \leftarrow \mathsf{idealToOctets}(\mathfrak{r})$
4: $H \leftarrow \mathsf{hash}(M \| R)$
5: **if** $H = \mathsf{invalid}$ **then return** invalid
6: $h \leftarrow \mathsf{hashToInteger}(H, 2t)$
7: $s \leftarrow -ah + k$
8: **return** $(s, \mathfrak{r})$

Verifikation

Mit dem folgenden Verfahren sollte **B** die IQ-DSA-Signatur $\mathcal{S}_{\mathbf{A},M}^{\text{IQ-DSA}}$ von **A** für den Text M überprüfen. Proposition 6.4.1 in Abschnitt 6.4 zeigt, daß die Verifikation für jede Signatur erfolgreich ist, die mit Algorithmus 4.3 erzeugt wurde.

Algorithmus 4.4 IQ-DSA Verifikation

Eingabe:

1. Die Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-DSA}} = (t, F(\Delta), \mathfrak{g})$;
2. den öffentlichen IQ-DSA-Schlüssel $\mathcal{K}_{\mathbf{A},\mathsf{pub}}^{\text{IQ-DSA}} = \mathfrak{a}$;
3. ein Octet-String M;
4. die Signatur $\mathcal{S}_{\mathbf{A},M}^{\text{IQ-DSA}} = (s, \mathfrak{r})$.

Ausgabe: valid oder invalid

1: **if** checkIdeal$(\Delta, \mathfrak{r})$ = invalid **then return** invalid
2: $R \leftarrow$ idealToOctets$(\mathfrak{r})$
3: $H \leftarrow$ hash$(M \| R)$
4: **if** H = invalid **then return** invalid
5: $h \leftarrow$ hashToInteger$(H, 2t)$
6: $\mathfrak{v} \leftarrow$ reduce$(\mathfrak{g}^s \mathfrak{a}^h)$
7: **if** $\mathfrak{v} \neq \mathfrak{r}$ **then return** invalid
8: **return** valid

4.1.3 IQ-GQ

Wir beschreiben nun IQ-GQ[8]. Das GQ-Signaturverfahren wurde ursprünglich für Gruppen $\mathbb{Z}/n\mathbb{Z}$ vorgeschlagen [30], wobei n wie ein RSA-Modulus ausgewählt sei. Dieses Verfahren hat die Eigenschaft, daß weder der Verifizierer noch der Signierer die Gruppenordnung für die Überprüfung bzw. Erzeugung einer Signatur benötigen. Das GQ-Signaturverfahren ist daher ein idealer Kandidat für die Verwendung mit Klassengruppen.

Eine schematische Darstellung von GQ für beliebige endliche Gruppen mit unbekannter Gruppenordnung, sowie eine Beweisskizze für die Sicherheit von GQ haben wir in Abschnitt 6.5 gegeben. In diesem Abschnitt beschreiben wir die Spezifikation für IQ-GQ.

Vorbereitung

A sollte die folgenden Schritte unternehmen, um die Benutzung von IQ-GQ vorzubereiten:

1. Lege eine kryptographische Hash-Funktion *hash* sowie die Parameter *hashlen* und *hashmaxlen* für die Verwendung in der Prozedur hash fest.
2. Lege Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-GQ}} = (t, F(\Delta), n)$ in Abhängigkeit des gewünschten Sicherheitsparameters t fest (genDomainIQGQ), und überprüfe die Bereichsparameter (checkDomainIQGQ).
3. Lege ein Schlüsselpaar $\mathcal{K}_{\mathbf{A}}^{\text{IQ-GQ}} = (\mathfrak{n}, \mathfrak{a})$ fest, welches bezüglich der Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-GQ}}$ ausgewählt wurde (genKeyPairIQGQ).

[8] GQ: Guillou-Quisquater

B sollte die folgenden Schritte unternehmen, um die Benutzung von IQ-GQ vorzubereiten:

1. Beschaffe **A**s authentische Hash-Funktion $hash_{\mathbf{A}}$ und deren Parameter *hashlen* und *hashmaxlen*.

2. Beschaffe **A**s authentische Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-GQ}}$, und überprüfe die Bereichsparameter (checkDomainIQGQ).

3. Beschaffe **A**s authentischen öffentlichen Schlüssel $\mathcal{K}_{\mathbf{A},\mathsf{pub}}^{\text{IQ-GQ}} = \mathfrak{n}$, und überprüfe den öffentlichen Schlüssel (checkPubKeyIQGQ).

Signatur

Mit dem folgenden Verfahren sollte **A** eine IQ-GQ-Signatur für einen Text M erzeugen, wobei die Bereichsparameter und die Schlüssel aus der Vorbereitung verwendet werden sollen. Die Signatur bezeichnen wir mit $\mathcal{S}_{\mathbf{A},M}^{\text{IQ-GQ}}$.

Algorithmus 4.5 IQ-GQ Signatur

Eingabe:

1. Die Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-GQ}} = (t, F(\Delta), n)$;
2. der private IQ-GQ-Schlüssel $\mathcal{K}_{\mathbf{A},\mathsf{prv}}^{\text{IQ-GQ}} = \mathfrak{a}$;
3. ein Octet-String $M = M_0 M_1 \ldots$.

Ausgabe: Eine Signatur $\mathcal{S}_{\mathbf{A},M}^{\text{IQ-GQ}} = (s, \mathfrak{s})$ zu M oder invalid

1: $\mathfrak{k} \leftarrow \mathsf{randomIdeal}(\Delta)$
2: $\mathfrak{r} \leftarrow \mathsf{reduce}(\Delta, \mathfrak{k}^n)$
3: $R \leftarrow \mathsf{idealToOctets}(\mathfrak{r})$
4: $H \leftarrow \mathsf{hash}(M \| R)$
5: **if** $H = \mathsf{invalid}$ **then return** invalid
6: $s \leftarrow \mathsf{hashToInteger}(H, 2t)$
7: $\mathfrak{s} \leftarrow \mathsf{reduce}(\mathfrak{k}\mathfrak{a}^s)$
8: **return** $(s, \mathfrak{s})$

Verifikation

Mit dem folgenden Verfahren sollte **B** die IQ-GQ-Signatur $\mathcal{S}_{\mathbf{A},M}^{\text{IQ-GQ}}$ von **A** für den Text M überprüfen. Proposition 6.5.1 in Abschnitt 6.5 zeigt, daß die Verifikation für jede Signatur erfolgreich ist, die mit Algorithmus 4.5 erzeugt wurde.

Algorithmus 4.6 IQ-GQ Verifikation

Eingabe:

1. Die Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-GQ}} = (t, F(\Delta), n)$;
2. den öffentlichen IQ-GQ-Schlüssel $\mathcal{K}_{\mathbf{A},\mathsf{pub}}^{\text{IQ-GQ}} = \mathfrak{n}$;
3. ein Octet-String M;
4. die Signatur $\mathcal{S}_{\mathbf{A},M}^{\text{IQ-GQ}} = (s, \mathfrak{s})$

Ausgabe: valid oder invalid

1: **if** checkIdeal$(\Delta, \mathfrak{s}) =$ invalid **then return** invalid
2: $\mathfrak{v} \leftarrow$ reduce$(\Delta, \mathfrak{s}^n \mathfrak{n}^s)$
3: $V \leftarrow$ idealToOctets$(\mathfrak{v})$
4: $H \leftarrow$ hash$(M \| V)$
5: **if** $H =$ invalid **then return** invalid
6: $v \leftarrow$ hashToInteger$(H, 2t)$
7: **if** $v \neq s$ **then return** invalid
8: **return** valid

4.2 Verschlüsselungsverfahren

Dieser Abschnitt beschreibt, wie man IQ-basierte Verschlüsselungsverfahren verwenden sollte. Im folgenden werden wir von dem Szenario ausgehen, daß eine Partei **B** einen Text M verschlüsseln und an eine Partei **A** senden möchte, und **A** diesen Text aus der Verschlüsselung C wieder rekonstruieren möchte.

Wir beschränken uns in diesem Abschnitt auf das Verschlüsselungsverfahren IQ-IES; eine Alternative, die hier nicht beschrieben wird, bestünde in der Übertragung des Cramer-Shoup-Verschlüsselungsverfahren [23] auf Klassengruppen.

4.2.1 IQ-IES

Wir beschreiben nun IQ-IES[9]. IQ-IES ist die IQ-Variante des DHIES-Verfahrens aus [1]. Da für DHIES die Kenntnis der Gruppenordnung nicht notwendig ist (weder für den Absender, noch den Empfänger eines verschlüsselten Textes), kann das DHIES-Verfahren ohne weiteres auf Klassengruppen übertragen werden.

Eine schematische Darstellung von DHIES für beliebige endliche Gruppen steht in Abschnitt 6.6; eine ausführliche Diskussion der Sicherheit von DHIES steht in [1]. In diesem Abschnitt beschreiben wir die Spezifikation für IQ-IES.

Vorbereitung

A sollte die folgenden Schritte unternehmen, um die Benutzung von IQ-IES vorzubereiten:

[9] IES: Integrated Encryption Scheme

1. Lege eine Schlüsselableitungs-Funktion $derive_{\mathbf{A}}$ für die Verwendung in der Prozedur deriveKey fest.

2. Lege einen MAC $MAC_{\mathbf{A}}$ sowie die Parameter *mackeylen* und *maclen* für die Verwendung in der Prozedur MAC fest.

3. Lege das symmetrische Verschlüsselungsverfahren $ENC_{\mathbf{A}}$ sowie den Parameter *enckeylen* für die Verwendung in der Prozedur ENC bzw. ENC^{-1} fest.

4. Lege Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-IES}} = (t, F(\Delta), \mathfrak{g})$ in Abhängigkeit des gewünschten Sicherheitsparameters t fest (genDomainIQIES) und überprüfe die Bereichsparameter (checkDomainIQIES).

5. Lege ein Schlüsselpaar $\mathcal{K}_{\mathbf{A}}^{\text{IQ-IES}} = (\mathfrak{a}, a)$ fest, welches bezüglich der Bereichsparameter $\mathcal{D}^{\text{IQ-IES}}$ ausgewählt wurde (genKeyPairIQIES).

B sollte die folgenden Schritte unternehmen, um die Benutzung von IQ-IES vorzubereiten:

1. Beschaffe **A**s authentische Schlüsselableitungs-Funktion $derive_{\mathbf{A}}$, den MAC $MAC_{\mathbf{A}}$ und das symmetrische Verschlüsselungsverfahren $ENC_{\mathbf{A}}$, sowie die Parameter *maclen*, *mackeylen* und *enckeylen*.

2. Beschaffe **A**s authentischen Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-IES}}$, und überprüfe die Bereichsparameter (checkDomainIQIES).

3. Beschaffe **A**s authentischen öffentlichen Schlüssel $\mathcal{K}_{\mathbf{A},\mathsf{pub}}^{\text{IQ-IES}} = \mathfrak{a}$, und überprüfe den öffentlichen Schlüssel (checkPubKeyIQIES).

Verschlüsselung

Mit dem folgenden Verfahren sollte **B** eine IQ-IES-Verschlüsselung zu einen Text M für **A** erzeugen, wobei die Bereichsparameter und die Schlüssel aus der Vorbereitung verwendet werden sollen. Den Schlüsseltext bezeichnen wir mit $\mathcal{C}_{\mathbf{A},M}^{\text{IQ-IES}}$

Algorithmus 4.7 IQ-IES Verschlüsselung

Eingabe:

1. Die Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-IES}} = (t, F(\Delta), \mathfrak{g})$;
2. den öffentlichen IQ-IES-Schlüssel $\mathcal{K}_{\mathbf{A},\mathsf{pub}}^{\text{IQ-IES}} = \mathfrak{a}$;
3. ein Octet-String $M = M_0 M_1 \ldots$.

Ausgabe: Einen Schlüsseltext $\mathcal{C}_{\mathbf{A},M}^{\text{IQ-IES}} = C$ oder invalid

1: $(\mathfrak{r}, k) \leftarrow \mathsf{genKeyPairIQIES}(\mathcal{D}_{\mathbf{A}}^{\text{IQ-IES}})$
2: $R \leftarrow \mathsf{idealToOctets}(\mathfrak{r})$
3: $\mathfrak{s} \leftarrow \mathsf{IQDH}(\mathcal{D}_{\mathbf{A}}^{\text{IQ-IES}}, k, \mathfrak{a})$
4: **if** $\mathfrak{s} = \mathsf{invalid}$ **then return** invalid
5: $S \leftarrow \mathsf{idealToOctets}(\mathfrak{s})$
6: $K \leftarrow \mathsf{deriveKey}(S)$
7: **if** $K = \mathsf{invalid}$ **then return** invalid
8: $//K = K_0 K_1 \ldots K_{enckeylen+mackeylen-1}$
9: $K_{\mathsf{ENC}} \leftarrow K_0 K_1 \ldots K_{enckeylen-1}$
10: $K_{\mathsf{MAC}} \leftarrow K_{enckeylen} K_{enckeylen+1} \ldots K_{enckeylen+mackeylen-1}$
11: $EM \leftarrow \mathsf{ENC}(M, K_{\mathsf{ENC}})$
12: **if** $EM = \mathsf{invalid}$ **then return** invalid
13: $T \leftarrow \mathsf{MAC}(EM, K_{\mathsf{MAC}})$
14: **if** $T = \mathsf{invalid}$ **then return** invalid
15: $C \leftarrow R \| EM \| T$
16: **return** C

Entschlüsselung

Mit dem folgenden Verfahren sollte $\mathbf{A}$ aus einer IQ-IES-Schlüsseltext $\mathcal{C}_{\mathbf{A},M}^{\text{IQ-IES}} = C$ den Klartext M rekonstruieren. Proposition 6.6.1 aus Abschnitt 6.6 zeigt, daß der Klartext immer aus dem Schlüsseltext rekonstruiert werden kann, wenn der Schlüsseltext mit Algorithmus 4.7 erzeugt wurde.

Algorithmus 4.8 IQ-IES Entschlüsselung

Eingabe:

1. Die Bereichsparameter $\mathcal{D}_{\mathbf{A}}^{\text{IQ-IES}} = (t, F(\Delta), \mathfrak{g})$;
2. den privaten IQ-IES-Schlüssel $\mathcal{K}_{\mathbf{A},\mathsf{prv}}^{\text{IQ-IES}} = a$;
3. einen Schlüsseltext $\mathcal{C}_{\mathbf{A},M}^{\text{IQ-IES}}$, dargestellt als Octet-String $C = C_0 C_1 \ldots C_{\ell-1}$.

Ausgabe: Der Klartext M oder invalid

1: **if** $C_0 \neq 04_{16}$ **and** $C_0 \neq 05_{16}$ **then return** invalid
2: $r \leftarrow \lceil (\log_2 \sqrt{|\Delta|/3})/8 \rceil$
3: $R \leftarrow C_1 C_2 \ldots C_{2r}$
4: $EM \leftarrow C_{2r+1} C_{2r+2} \ldots C_{\ell-maclen-1}$
5: $T \leftarrow C_{\ell-maclen} C_{\ell-maclen+1} \ldots C_{\ell-1}$
6: $\mathfrak{r} \leftarrow \mathsf{octetsToIdeal}(R)$
7: **if** $\mathsf{checkPubKeyIQIES}(\mathfrak{r}) = \text{invalid}$ **then return** invalid
8: $\mathfrak{s} \leftarrow \mathsf{IQDH}(\mathcal{D}_{\mathbf{A}}^{\text{IQ-IES}}, a, \mathfrak{r})$
9: **if** $\mathfrak{s} = \text{invalid}$ **then return** invalid
10: $S \leftarrow \mathsf{idealToOctets}(\mathfrak{s})$
11: $K \leftarrow \mathsf{deriveKey}(S)$
12: **if** $K = \text{invalid}$ **then return** invalid
13: $//K = K_0 K_1 \ldots K_{enckeylen+mackeylen-1}$
14: $K_{\mathsf{ENC}} \leftarrow K_0 K_1 \ldots K_{enckeylen-1}$
15: $K_{\mathsf{MAC}} \leftarrow K_{enckeylen} K_{enckeylen+1} \ldots K_{enckeylen+mackeylen-1}$
16: $T' \leftarrow \mathsf{MAC}(EM, K_{\mathsf{MAC}})$
17: **if** $T' = \text{invalid}$ **or** $T' \neq T$ **then return** invalid
18: $M \leftarrow \mathsf{ENC}^{-1}(EM, K_{\mathsf{ENC}})$
19: **if** $M = \text{invalid}$ **then return** invalid
20: **return** M

4.3 Schlüsselaustauschverfahren

Dieser Abschnitt beschreibt, wie man IQ-basierte Schlüsselaustauschverfahren verwenden sollte. Im folgenden werden wir von dem Szenario ausgehen, daß die Parteien **A** und **B** ein gemeinsames Geheimnis berechnen wollen. Ein gemeinsamer geheimer Schlüssel könnte daraus dann anschließend mit Hilfe einer Schlüsselableitungs-Funktion berechnet werden.

In diesem Abschnitt beschränken wir uns auf die Übertragung des einfachen Diffie-Hellman-Schlüsselaustauschs auf Klassengruppen. Andere Alternativen bestünden in den MTI-Protokollen [51, Abschnitt 12.6]. Das MQV-Protokoll, wie es in [3] für elliptische Kurven beschrieben wurde, kann nicht ohne weiteres auf Klassengruppen übertragen werden, da für das MQV-Protokoll die Kenntnis der Gruppenordnung erforderlich ist.

4.3.1 IQ-DH

Wir beschreiben nun IQ-DH[10]. Hierbei handelt es sich um eine Variante des klassischen Schlüsselaustauschverfahrens nach Diffie und Hellman [24] für Klassengruppen. Da hierfür die Kenntnis der Gruppenordnung nicht erforderlich ist, läßt sich dieses Verfahren ohne weiteres übertragen.

IQ-DH ist mit IQ-IES verwandt; der Unterschied zu IQ-IES besteht darin, daß bei IQ-DH vor dem Austausch die Bereichsparameter *gemeinsam* festgelegt werden müssen, und daß die Auswahl der Schlüsselpaare erst unmittelbar vor dem Austausch erfolgt. Außerdem wird beim Schlüsselaustausch jedes Schlüsselpaar nur für einen einzigen Durchgang des Protokolls verwendet wird, um bei jedem Austausch ein unterschiedliches gemeinsames Geheimnis zu berechnen.

Vorbereitung

A und **B** sollten die folgenden Schritte gemeinsam unternehmen, um die Benutzung von IQ-DH vorzubereiten:

1. Vereinbare eine Schlüsselableitungs-Funktion *derive* für die Verwendung in der Prozedur deriveKey.

2. Vereinbare einen Sicherheitsparameter t und Bereichsparameter $\mathcal{D}^{\text{IQ-DH}}$ in Abhängigkeit von t (genDomainIQDH), und überprüfe die Bereichsparameter (checkDomainIQDH).

A und **B** sollten dann die folgenden Schritte für sich unternehmen, um die Benutzung von IQ-DH vorzubereiten:

3. Lege ein Schlüsselpaar $\mathcal{K}_{\mathbf{A}}^{\text{IQ-IES}} = (\mathfrak{a}_{\mathbf{A}}, a_{\mathbf{A}})$ bzw. $\mathcal{K}_{\mathbf{B}}^{\text{IQ-IES}} = (\mathfrak{a}_{\mathbf{B}}, a_{\mathbf{B}})$ fest, welches bezüglich der gemeinsamen Bereichsparameter $\mathcal{D}^{\text{IQ-DH}}$ ausgewählt wurde (genKeyPairIQDH).

4. Beschaffe **A**s bzw. **B**s authentischen öffentlichen IQ-DH-Schlüssel, und überprüfe den öffentlichen Schlüssel (checkPubKeyIQDH).

Schlüsselaustausch

Mit dem folgenden Verfahren sollte **A** einen gemeinsamen geheimen Schlüssel mit **B** berechnen, wobei die Bereichsparameter $\mathcal{D}^{\text{IQ-DH}}$ und die Schlüssel $\mathfrak{a}_{\mathbf{B}}$ und $a_{\mathbf{A}}$ aus der Vorbereitung verwendet werden sollen:

[10] DH: Diffie-Hellman

Algorithmus 4.9 IQ-DH Schlüsselaustausch

Eingabe:

1. Die Bereichsparameter $\mathcal{D}^{\text{IQ-DH}} = (t, F(\Delta), \mathfrak{g})$;
2. den privaten IQ-DH-Schlüssel $a_{\mathbf{A}}$;
3. den öffentlichen IQ-DH-Schlüssel $\mathfrak{b}_{\mathbf{A}}$;

Ausgabe: Einen gemeinsamen geheimen Schlüssel K oder invalid

1: $\mathfrak{s} \leftarrow \mathsf{IQDH}(\mathcal{D}^{\text{IQ-DH}}, a_{\mathbf{A}}, \mathfrak{a}_{\mathbf{B}})$
2: **if** $\mathfrak{s} =$ invalid **then return** invalid
3: $S \leftarrow \mathsf{idealToOctets}(\mathfrak{s})$
4: $K \leftarrow \mathsf{deriveKey}(S)$
5: **if** $K =$ invalid **then return** invalid
6: **return** K

B sollte dieses Verfahren in entsprechender Weise anwenden, um den gemeinsamen geheimen Schlüssel K für sich zu berechnen. Mit einer Schlüsselableitungs-Funktion können **A** und **B** aus dem gemeinsamen Geheimnis K z. B. ein Schlüssel für symmetrische Verschlüsselung erzeugt werden.

Teil II

Analyse IQ-basierter Kryptoverfahren

Kapitel 5

Auswahl der kryptographischen Parameter

In Kapitel 4 haben wir verschiedene IQ-basierte Kryptoverfahren beschrieben. In Kapitel 6 werden wir zeigen, daß diese Kryptoverfahren sicher sind unter der Voraussetzung, daß das Diskrete-Logarithmus-Problem (für IQ-DSA), das Wurzelproblem (für IQ-RDSA und IQ-GQ) oder das Diffie-Hellman-Problem (für IQ-IES und IQ-DH) in der verwendeten Klassengruppe i. allg. nicht effizient gelöst werden können. Es ist kein besseres Verfahren bekannt, um eine Wurzel oder ein Diffie-Hellman-Geheimniss in Klassengruppen zu berechnen, als zuvor einen diskreten Logarithmus zu berechnen; dies werden wir in Kapitel 6 noch näher ausführen. Wir beschränken uns in diesem Kapitel daher darauf, Klassengruppen zu identifizieren, in denen die Berechnung diskreter Logarithmen selbst mit den besten bekannten Algorithmen nicht effizient möglich ist. Wir betrachten dazu in diesem Kapitel alle bekannten Strategien, um diskrete Logarithmen in Klassengruppen zu berechnen, und wir zeigen, welche Eigenschaften erfüllt sein müssen, um die effiziente Berechnung diskreter Logarithmen in Klassengruppen zu verhindern.

Um die effiziente Berechnung von diskreten Logarithmen durch eine Reduktion auf einfachere DL-Probleme zu verhindern, muß die Diskriminante fundamental gewählt werden. In Abschnitt 5.2.1 zeigen wir, wie man effizient eine zufällige Fundamental-Diskriminante wählen kann.

Um die effiziente Berechnung von diskreten Logarithmen mit Hilfe von Index-Berechnungs-Algorithmen zu verhindern, muß die Diskriminante groß sein. In Abschnitt 5.2.2 schätzen wir ab, wie groß die Diskriminante sein muß, um mit einem Index-Berechnungs-Algorithmus einen bestimmten Berechnungsaufwand erwarten zu können.

Bei einem Kryptoverfahren wie IQ-DSA oder IQ-DH wählen wir ein Basiselement $\mathfrak{g}$ und einen geheimen Wert a; $\mathfrak{g}$ und $\mathfrak{a} = \mathfrak{g}^a$ werden veröffentlicht. Um die effiziente Berechnung von a aus $\mathfrak{a}$ und $\mathfrak{g}$ mit einem der sog. Quadratwurzel-Algorithmen zu verhindern, muß $|\langle\mathfrak{g}\rangle|$ möglichst groß sein. In Abschnitt 5.1.1 sehen wir, daß die Klassengruppe mit sehr hoher Wahrscheinlichkeit eine sehr große zyklische Untergruppe enthält. Wenn $\mathfrak{g}$ mit Gleichverteilung zufällig aus $\mathrm{Cl}(\Delta)$ gezogen wird, dann erzeugt $\mathfrak{g}$ auch mit hoher Wahrscheinlichkeit eine sehr große Untergruppe von $\mathrm{Cl}(\Delta)$. In Abschnitt 5.3 beschreiben wir einen Algorithmus, der bei geeigneter Wahl der Parameter ein zufälliges reduziertes $\mathcal{O}_\Delta$-Ideal mit annähernder Gleichverteilung wählt. In Abschnitt 5.1.2 sehen wir, daß die Klassengruppe sehr groß ist,

wenn auch die Diskriminante sehr groß ist. In Abschnit 5.2.3 schätzen wir ab, wie groß die Diskriminante sein muß, um mit dem λ-Algorithmus einen bestimmten Berechnungsaufwand erwarten zu können.

Um die effiziente Berechnung von a aus $\mathfrak{a}$ und $\mathfrak{g}$ mit dem Pohlig-Hellman-Algorithmus zu verhindern, muß $|\langle\mathfrak{g}\rangle|$ einen möglichst großen Primteiler enthalten. Ist die Klassenzahl sehr glatt, dann kann $|\langle\mathfrak{g}\rangle|$ effizient berechnet werden, und der Pohlig-Hellman-Algorithmus kann ebenfalls effizient eingesetzt werden. Enthält die Klassenzahl dagegen einen großen Primteiler, dann ist die Wahrscheinlichkeit sehr groß, daß auch die Ordnung eines zufällig gewählten Elements diesen großen Primteiler enthält. In Abschnitt 5.1.3 zeigen wir, daß die Wahrscheinlichkeit sehr klein ist, daß die Klassenzahl sehr glatt ist, wenn die Diskriminante sehr groß ist. In Abschnitt 5.2.4 schätzen wir ab, wie groß die Diskriminante sein muß, um für die Berechnung des größten Primteilers der Klassenzahl einen bestimmten Berechnungsaufwand erwarten zu können.

Daraus schließen wir in Abschnitt 5.4, daß diskrete Logarithmen in Klassengruppen mit sehr hoher Wahrscheinlichkeit nicht effizient berechnet werden können, wenn die Diskriminante fundamental und möglichst groß ist. Wie groß die Diskriminante sein muß, wird einerseits bestimmt durch den schnellsten Algorithmus zur DL-Berechnung, und andererseits durch den geforderten Aufwand für eine DL-Berechnung. Wir geben an, wie groß die Diskriminante sein muß, damit wir für die Berechnung von diskreten Logarithmen mit dem besten bekannten Algorithmen einen bestimmten Aufwand erwarten können. Den Aufwand für die Berechnung von diskreten Logarithmen in Klassengruppen vergleichen wir mit dem Aufwand, um andere bekannte Berechnungsprobleme zu lösen (z. B. Faktorisierung ganzer Zahlen). Hierüber läßt sich die Größe der Diskriminante so abschätzen, daß ein IQ-basiertes Kryptoverfahren so sicher ist, wie andere Public-Key-Verfahren mit bestimmter Parametergröße (z. B. RSA mit 1024-Bit-Modulus). Die Ergebnisse dieses Kapitels sind im wesentlichen Bestandteil unserer Arbeit in [32].

Dieses Kapitel ist folgendermaßen aufgebaut: In Abschnitt 5.1 haben wir alle notwendigen Ergebnisse über Klassengruppen gesammelt, die wir später in diesem Kapitel verwenden. In Abschnitt 5.2 betrachten wir alle Strategien zur DL-Berechnung in Klassengruppen und zeigen, wie die Diskriminante jeweils ausgewählt werden muß, um eine effiziente Anwendung dieser Strategien zu verhindern. In Abschnitt 5.3 beschreiben wir einen Algorithmus, der mit annähernder Gleichverteilung ein zufälliges primitives, reduziertes Ideal wählt. In Abschnitt 5.4 kombinieren wir die Ergebnisse aus Abschnitt 5.1 mit denen aus Abschnitt 5.2 und geben an, wie die Diskriminante ausgewählt werden muß, damit in der entsprechenden Klassengruppe diskrete Logarithmen mit sehr hoher Wahrscheinlichkeit nicht effizient berechnet werden können.

5.1 Eigenschaften von Klassengruppen

In Abschnitt 5.2 werden wir für alle bekannten Strategien zur Berechnung von diskreten Logarithmen in Klassengruppen zeigen, welche Eigenschaften erfüllt sein müssen, um die jeweilige Strategie zu verhindern.

1. Δ soll fundamental sein, um Angriffe durch Reduktionen auf einfachere DL-Probleme zu verhindern.

2. Δ soll groß sein, um Angriffe durch Index-Berechnungs-Algorithmen zu verhindern.

3. Die Klassengruppe soll möglichst groß sein oder eine möglichst große zyklische Untergruppe enthalten, um einen Angriff durch einen der sog. Quadratwurzel-Algorithmen zu verhindern.

4. Die Klassengruppe soll darüberhinaus einen möglichst großen Primteiler enthalten, um das Pohlig-Hellman-Algorithmus zu verhindern. Wenn die Klassenzahl sehr glatt ist, kann die Ordnung der von einem beliebigen Element erzeugten Untergruppe effizient berechnet werden, und in diesem Fall kann auch das Pohlig-Hellman-Algorithmus erfolgreich angewendet werden.

In diesem Abschnitt zeigen wir, daß die Klassengruppe einer sehr großen Fundamental-Diskriminante tatsächlich selber sehr groß ist, und daß die Glattheitswahrscheinlichkeit für Klassenzahlen zufälliger Fundamental-Diskriminanten vernachlässigbar klein ist, wenn die Diskriminante sehr groß ist. Damit kann die Anforderung an eine kryptographisch geeignete Diskriminante folgendermaßen zusammengefaßt werden: Eine kryptographisch geeignete Diskriminante muß fundamental und groß sein.

Die Ergebnisse in diesem Abschnitt, etwa die Abschätzung für die Größe der Klassenzahl oder Aussagen über die Struktur der Klassengruppe, stützen sich in wesentlichen Punkten auf unbewiesene Annahmen und Heuristiken, nämlich die erweiterte Riemann-Vermutung und die Cohen-Lenstra-Heuristik. Unsere Argumentation für die Parameterauswahl ist denn auch von dieser Annahme und Heuristik abhängig.

5.1.1 Die Größe der Klassengruppe

Wir geben in diesem Abschnitt die bekannten Resultate zur Abschätzung der Klassenzahl an. Die Klassenzahl $h(\Delta)$ ist die Ordnung der Klassengruppe $\mathrm{Cl}(\Delta)$. Von der Größenordnung der Klassenzahl ist die erwartete Laufzeit mit einem der sog. Quadratwurzel-Algorithmen abhängig (siehe Abschnitt 5.2.3). Asymptotisch wächst $h(\Delta)$ wie $\sqrt{|\Delta|}$, und unter Annahme der erweiterten Riemann-Vermutung kann gezeigt werden, daß die oberen und unteren Schranken für $h(\Delta)$ nicht wesentlich größer bzw. kleiner als $\sqrt{|\Delta|}$ sind.

Es ist kein *effizienter* Algorithmus bekannt, um $h(\Delta)$ exakt zu bestimmen (siehe [19, Abschnitt 5.3] für verschiedene Ansätze). Wir definieren

$$L_\Delta(s) = \sum_{n \geqslant 1} \left(\frac{\Delta}{n}\right) n^{-s} \ , \quad s \in \mathbb{C} \ , \tag{5.1}$$

und es gilt:

Proposition 5.1.1 (Analytische Klassenzahl-Formel)
Sei Δ eine fundamentale, imaginär-quadratische Diskriminante, $\Delta < -4$. Dann ist

$$L_\Delta(1) = \frac{\pi h(\Delta)}{\sqrt{|\Delta|}} \ . \tag{5.2}$$

Somit ist

$$h(\Delta) = \frac{L_\Delta(1)\sqrt{|\Delta|}}{\pi} \tag{5.3}$$

für $\Delta < -4$. Ein Näherung von $h(\Delta)$ kann mittels (5.3) gemacht werden, indem eine Approximation von $L_\Delta(1)$ berechnet wird (siehe z. B. [40, Abschnitt 2.5]).

Für kryptographische Anwendungen ist es von Bedeutung, daß die Klassengruppe bei zufälliger Wahl der Diskriminante möglichst groß ist. Dies ist der Fall, wenn der Betrag der Diskriminante selber sehr groß ist. Die Klassenzahl $h(\Delta)$ hat asymptotisch die Größenordnung $\sqrt{|\Delta|}$. Dies folgt aus dem folgenden Theorem, welches ein Spezialfall des Brauer-Siegel-Theorems [45, Kapitel 16] für imaginär-quadratische Körper ist.

Theorem 5.1.2 (Brauer, Siegel – Spezialfall)
Sei Δ eine imaginär-quadratische Fundamental-Diskriminante. Dann ist

$$\lim_{\Delta \to -\infty} \frac{\ln h(\Delta)}{\ln \sqrt{|\Delta|}} = 1 \ . \tag{5.4}$$

Das bedeutet, daß $\sqrt{|\Delta|}^{1-\epsilon} \leqslant h(\Delta) \leqslant \sqrt{|\Delta|}^{1+\epsilon}$ für $\Delta \to -\infty$ und jedes reelles $\epsilon > 0$. Das Hauptproblem damit besteht darin, daß keine expliziten Konstanten bekannt sind, für die diese Aussage gilt.

Die asymptotische Aussage wird durch den arithmetischen Mittelwert der Klassenzahlen fundamentaler Diskriminanten unterstützt [19, Abschnitt 5.10.1]:

Proposition 5.1.3
Sei $\overline{h_D}$ der arithmetische Mittelwert aller Klassenzahlen der fundamentalen, imaginär-quadratischen Diskriminanten Δ für $|\Delta| < D$. Dann ist

$$\lim_{D \to \infty} \frac{\overline{h_D}}{\sqrt{D}} = c_1 \tag{5.5}$$

mit

$$c_1 = \frac{\pi}{9} \prod_p \left(1 + \frac{1}{p^2(p+1)}\right) \approx 0.307715 \ .$$

Dies folgt aus

$$\lim_{D \to \infty} \frac{\sum_{|\Delta| \leqslant D} h(\Delta)}{D^{3/2}} = \frac{C}{3\pi}$$

und

$$\lim_{D \to \infty} \frac{\left|\{\Delta : |\Delta| \leqslant D\}\right|}{D} = \frac{3}{\pi^2}$$

für Fundamental-Diskriminanten Δ.

Um den kryptographischen Parameter, nämlich die Diskriminante, richtig zu wählen, wird eine untere Schranke für die Klassenzahl benötigt. Unter der Annahme der erweiterten Riemann-Vermutung hat Littlewood dazu folgendes gezeigt [50]:

Theorem 5.1.4 (Littlewood)
Sei Δ eine fundamentale, imaginär-quadratische Diskriminante. Dann gilt unter Annahme der erweiterten Riemann-Vermutung mit $\Delta \to -\infty$

$$\frac{1+o(1)}{2c_3 \ln\ln|\Delta|} \;<\; L_\Delta(1) \;<\; (1+o(1))2c_2 \ln\ln|\Delta| \;, \tag{5.6}$$

wobei

$$c_2 = e^\gamma \approx 1.781072 \;, \quad c_3 = \frac{6}{\pi^2} e^\gamma \approx 1.082762$$

und $\gamma \approx 0.577216$ die Eulersche Konstante ist.

Daraus folgt mit (5.3), daß $h(\Delta)$ nur geringfügig von $\sqrt{|\Delta|}$ abweicht, falls die erweiterte Riemann-Vermutung wahr ist. Z. B. ist $\ln\ln|\Delta| < 8$ für alle Diskriminanten Δ, $|\Delta| \leqslant 2^{4300}$. Für kryptographische Zwecke sind Diskriminanten in dieser Größenordnung ausreichend. Mit dieser Beschränkung erhalten wir die folgende untere Schranke für $h(\Delta)$: Mit

$$c_4 = 2 \cdot 8\pi c_3 = 96 e^\gamma/\pi \approx 54.425564$$

ist

$$h(\Delta) \;>\; \frac{\sqrt{|\Delta|}}{c_4} \;>\; \frac{\sqrt{|\Delta|}}{2^6} \tag{5.7}$$

für $|\Delta| \leqslant 2^{4600}$.

5.1.2 Die Struktur der Klassengruppe

In diesem Abschnitt betrachten wir die Struktur der Klassengruppe. Unter Annahme der Heuristik von Cohen und Lenstra kann gezeigt werde, daß die Klassengruppe mit sehr hoher Wahrscheinlichkeit entweder zyklisch ist, oder aber eine große zyklische Untergruppe enthält. Das ist eine wichtige Voraussetzung dafür, daß ein zufälliges Element der Klassengruppe mit sehr hoher Wahrscheinlichkeit eine sehr große Untergruppe der Klassengruppe erzeugt (siehe Abschnitt 5.3).

Wir betrachten den geraden und den ungeraden Anteil der Klassengruppe getrennt. Als geraden Anteil einer Gruppe bezeichnen wir die Untergruppe mit Elementen, deren Ordnungen Potenzen von 2 sind, und als ungeraden Anteil einer Gruppe bezeichnen wir die Untergruppe mit Elementen ungerader Ordnung.

Der gerade Anteil der Klassengruppe

Der gerade Anteil der Klassengruppe wird durch die Primfaktor-Zerlegung der Diskriminante bestimmt. Dadurch ist der gerade Anteil nicht zufällig. Die Klassengruppe soll für kryptographische Zwecke aber möglichst zufällig sein. Der gerade Anteil sollte daher so klein wie möglich sein. Wir betrachten zunächst den Sonderfall des 2-Anteils und definieren ambige Ideale und den 2-Anteil von Klassengruppen:

Definition 5.1.5
Ein $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}$ heißt ambig, *wenn $\mathfrak{a}$ äquivalent zu $\mathfrak{a}^{-1}$ ist, d. h. wenn $\mathfrak{a}^2$ äquivalent zu $(1, \Delta \bmod 2)$ ist.*

Die Ideal-Klasse eines ambigen $\mathcal{O}_\Delta$-Ideals hat Ordnung 2 in der Klassengruppe. Die Untergruppe der ambigen Ideal-Klassen bezeichnen wir mit $\mathrm{Cl}_2(\Delta)$, und $|\mathrm{Cl}_2(\Delta)| = h_2(\Delta)$.

Proposition 5.1.6
Sei $\mathfrak{a} = (a, b)$ ein reduziertes $\mathcal{O}_\Delta$-Ideal, und sei $c = (b^2 - \Delta)/4a$. $\mathfrak{a}$ ist genau dann ambig, wenn $b = 0$, $b = a$ oder $c = a$ ist.

Dies kann analog zu Theorem 2.8.6 in [12] gezeigt werden (siehe auch Theorem III aus [66]).

Aus Proposition 5.1.6 folgt, daß sich aus der Darstellung eines reduzierten ambigen Ideals Faktoren der Diskriminante ablesen lassen. Diese Eigenschaft ambiger Ideale ist von zentraler Bedeutung für die Reduktion des Faktorisierungsproblem auf das Problem, die Ordnung der von einer zufälligen Ideal-Klasse erzeugten Untergruppe zu berechnen (siehe Abschnitt 5.2.1); diese Reduktion wird z. B. in dem Faktorisierungs-Algorithmus von Schnorr und Lenstra [66] ausgenutzt.

Wenn z. B. $b = 0$ ist, dann ist $\Delta = -4ac$; falls $a \neq 1$ und $a \neq \Delta/4$, dann ist a ein nicht-trivialer Faktor von Δ. Wenn $b = a$ ist, dann ist $\Delta = a^2 - 4ac = a(a - 4c)$; falls $a \neq 1$ und $a \neq \Delta$, dann ist a ein nicht-trivialer Faktor von Δ. (Diese beiden Fälle sind Spezialfälle von $b = ak$, $k \in \mathbb{Z}$.) Wenn $c = a$ ist, dann ist $\Delta = b^2 - 4a^2 = (b - 2a)(b + 2a)$; falls $2a \pm b \neq 1$ ist, dann ist $2a \pm b$ ein nicht-trivialer Faktor von Δ.

Die Anzahl der ambigen Ideale wird durch folgendes Lemma bestimmt:

Lemma 5.1.7
Sei Δ eine fundamentale Diskriminante, und sei t die Anzahl der unterschiedlichen Primfaktoren von Δ. Dann ist

$$h_2(\Delta) = \begin{cases} 2^t & \text{falls } \Delta \equiv 0 \pmod{32}, \\ 2^{t-2} & \text{falls } \Delta \equiv 4 \pmod{16}, \\ 2^{t-1} & \text{in allen anderen Fällen.} \end{cases}$$

Siehe z. B. [48, Theorem 2.5].

Aus Lemma 5.1.7 folgt beispielsweise, daß die Klassenzahl einer primen Diskriminante ungerade ist:

Proposition 5.1.8
Sei p eine Primzahl, $p \equiv 3 \pmod 4$ und $\Delta = -p$. Dann $2 \nmid h(\Delta)$.

Lemma 5.1.7 sagt nur etwas über die Anzahl der Elemente mit Ordnung genau 2 (oder 1). Für den gesamten geraden Anteil der Klassengruppe sind weitere Untersuchungen notwendig. Wir geben hier zwei weitere Beispiele für Klassenzahlen mit kleinem geraden Anteil:

Proposition 5.1.9
Seien p und q Primzahlen, $pq \equiv 3 \pmod 4$, $\left(\frac{p}{q}\right) = -1$ und $\Delta = -pq$. Dann $2 \parallel h(\Delta)$.

Siehe [60].

Proposition 5.1.10
Seien p und q Primzahlen, $p \equiv 1 \pmod 8$, $p + q \equiv 8 \pmod{16}$, $\left(\frac{p}{q}\right) = -1$ und $\Delta = -8pq$. Dann $2^3 \parallel h(\Delta)$.

Siehe [41].

Der ungerade Anteil der Klassengruppe

Zur Betrachtung des ungeraden Anteils ziehen wir die Heuristiken von Cohen und Lenstra heran, siehe [19, Abschnitt 5.10.1], [20] und [21]. Wir bezeichnen den ungeraden Anteil von $\mathrm{Cl}(\Delta)$ mit $\mathrm{Cl}_{\mathsf{odd}}(\Delta)$ und entsprechend $h_{\mathsf{odd}}(\Delta) = |\mathrm{Cl}_{\mathsf{odd}}(\Delta)|$. In Anlehnung an [19, Abschnitt 5.10.1] definieren wir

$$(p)_r = \prod_{1 \leqslant k \leqslant r} \left(1 - \frac{1}{p^k}\right)$$

für alle $r \in \mathbb{Z}_{>0}$ oder $r = \infty$.

Vermutung 5.1.11 (Cohen, Lenstra)
Sei Δ eine zufällig gewählte, imaginär-quadratische Fundamental-Diskriminante mit $|\Delta| \leqslant D$ für $D \in \mathbb{Z}_{>0}$.

1. *Für die Wahrscheinlichkeit, daß $\mathrm{Cl}_{odd}(\Delta)$ zyklisch ist, gilt*

$$\lim_{D\to\infty} \Pr\left[\ \mathrm{Cl}_{odd}(\Delta) \text{ ist zyklisch } \right] = \frac{\zeta(2)\zeta(3)}{3(2)_\infty \zeta(6) \prod_{k\geqslant 2} \zeta(k)} \approx 0.977575 \ .$$

2. *Sei p eine beliebige, feste, ungerade Primzahl. Für die Wahrscheinlichkeit, daß die p-Sylow-Untergruppe $\mathrm{Cl}_p(\Delta)$ von $\mathrm{Cl}(\Delta)$ isomorph zu einer gegebenen endlichen, Abelschen p-Gruppe $\mathcal{G}_p$ ist, gilt*

$$\lim_{D\to\infty} \Pr\left[\ \mathrm{Cl}_p(\Delta) \cong \mathcal{G}_p \ \right] = \frac{(p)_\infty}{|\operatorname{Aut} \mathcal{G}_p|} \ .$$

3. *Sei p eine beliebige, feste, ungerade Primzahl. Für die Wahrscheinlichkeit, daß die p-Sylow-Untergruppe $\mathrm{Cl}_p(\Delta)$ von $\mathrm{Cl}(\Delta)$ Rang r hat, gilt*

$$\lim_{D\to\infty} \Pr\left[\ \operatorname{rank}(\mathrm{Cl}_p(\Delta)) = r \ \right] = \frac{(p)_\infty}{p^{r^2}(p)_r^2} \ .$$

Die Bedeutung dieser Aussagen ist die folgende: $\mathrm{Cl}_{\mathsf{odd}}(\Delta)$ ist mit relativ großer Wahrscheinlichkeit zyklisch, aber wenn $\mathrm{Cl}_{\mathsf{odd}}(\Delta)$ nicht zyklisch ist, dann hat $\mathrm{Cl}_{\mathsf{odd}}(\Delta)$ mit sehr hoher Wahrscheinlichkeit eine sehr große zyklische Untergruppe.

5.1.3 Die Glattheitswahrscheinlichkeit von Klassenzahlen

In diesem Abschnitt betrachten wir die Glattheitswahrscheinlichkeit von Klassenzahlen zufällig gewählter Fundamental-Diskriminanten bei einer bestimmten Glattheitsschranke. Unter geeigneten Annahmen (s. u.) zeigen wir, daß die Glattheitswahrscheinlichkeit vernachlässigbar ist, wenn die Diskriminante sehr groß ist. Dies ist eine bedeutende Voraussetzung dafür, daß der Pohlig-Hellman-Algorithmus zur DL-Berechnung in Klassengruppen mit sehr hoher Wahrscheinlichkeit nicht effizient ist (siehe Abschnitt 5.2.4).

Wir betrachten zunächst die Wahrscheinlichkeit, daß eine ganze Zahl $m \leqslant \sqrt{D}$ für $D \to \infty$ die Klassenzahl einer zufälligen Fundamental-Diskriminante Δ teilt mit $|\Delta| < D$. Hierzu ziehen wir wiederum die Heuristik von Cohen und Lenstra heran, siehe [19, Abschnitt 5.10.1], [20] und [21].

Vermutung 5.1.12 (Cohen, Lenstra)
Sei p eine beliebige, feste, ungerade Primzahl und Δ eine zufällige, imaginär-quadratische Fundamental-Diskriminante mit $|\Delta| \leqslant D$ für $D \in \mathbb{Z}_{>0}$. Dann gilt für die Wahrscheinlichkeit, daß p ein Teiler von $h(\Delta)$ ist

$$\lim_{D\to\infty} \Pr\left[\ p \mid h(\Delta)\ \right] = 1 - \prod_{k\geqslant 1}\left(1 - \frac{1}{p^k}\right) = \frac{1}{p} + \frac{1}{p^2} - \frac{1}{p^5} - \frac{1}{p^7} + \frac{1}{p^{12}} + \frac{1}{p^{15}} \ldots \ . \tag{5.8}$$

Dabei ist

$$\Pr\left[\ p \mid h(\Delta)\ \right] < \frac{1}{p} + \frac{1}{p^2} = \frac{1}{p}\left(1 + \frac{1}{p}\right) \ . \tag{5.9}$$

Wir machen außerdem noch die folgenden Annahmen, die durch empirische Daten gestützt werden [15]:

Annahme 5.1.13
Sei Δ eine zufällige, imaginär-quadratische Fundamental-Diskriminante mit $|\Delta| \leqslant D$ für $D \in \mathbb{Z}_{>0}$.

1. *Sei p eine beliebige, feste, ungerade Primzahl. Dann gilt, in Erweiterung zu* (5.8) *bzw.* (5.9)*, für jede positive, ganze Zahl e*

$$\lim_{D\to\infty} \Pr\left[\ p^e \mid h(\Delta)\ \right] \leqslant \frac{1}{p^e} + \frac{1}{p^{e+1}} = \frac{1}{p^e}\left(1 + \frac{1}{p}\right) \ . \tag{5.10}$$

2. *Seien $p_1, p_2, \ldots, p_k$ beliebige, feste, paarweise verschiedene, ungerade Primzahlen. Dann sind die Wahrscheinlichkeiten in* (5.8) *und* (5.10) *für alle Primzahlen p_i unabhängig voneinander:*

$$\lim_{D\to\infty} \Pr\left[\ \prod_{1\leqslant i\leqslant k} p_i \mid h(\Delta)\ \right] = \prod_{1\leqslant i\leqslant k} \Pr\left[\ p_i \mid h(\Delta)\ \right] \tag{5.11}$$

und

$$\lim_{D\to\infty} \Pr\left[\ \prod_{1\leqslant i\leqslant k} p_i^{e_i} \mid h(\Delta)\ \right] = \prod_{1\leqslant i\leqslant k} \Pr\left[\ p_i^{e_i} \mid h(\Delta)\ \right] \ , \quad e_i \in \mathbb{Z}_{>0} \ . \tag{5.12}$$

Wir betrachten nun die Wahrscheinlichkeit, daß die Klassenzahl einer zufälligen Fundamental-Diskriminante B-glatt ist, d. h. daß eine Klassenzahl sich nur aus Primzahlen $\leqslant B$ zusammensetzt. Wir vergleichen dies mit der Glattheitswahrscheinlichkeit von Zufallszahlen aus $\mathbb{Z}$. Dazu wählen wir Fundamental-Diskriminanten Δ mit $|\Delta| \leqslant D$ für $D \in \mathbb{Z}_{>0}$; in Anlehnung an das Brauer-Siegel-Theorem wählen wir zum Vergleich mit den ganzen Zahlen $x \leqslant \sqrt{D}$.

Wir zeigen unter Annahme der Cohen-Lenstra-Heuristik und der Erweiterung 5.1.13: Sei $B \in \mathbb{Z}_{>0}$ fest, Δ eine zufällig gewählte Fundamental-Diskriminante mit $|\Delta| \leqslant D$ und x eine Zufallszahl mit $x \leqslant \sqrt{D}$, dann gilt für $D \to \infty$ und eine Konstante c_5, die später noch bestimmt wird

$$\Pr\left[\ h(\Delta) \text{ ist } B\text{-glatt}\ \right] \leqslant c_5 \ln\ln D \cdot \Pr\left[\ x \text{ ist } B\text{-glatt}\ \right] \ .$$

Sei p eine ungerade Primzahl und i eine positive ganze Zahl. Für die Wahrscheinlichkeit, daß eine ganze Zahl $x \in \{1, \ldots, \lfloor\sqrt{D}\rfloor\}$ durch p^i teilbar ist, gilt

$$\lim_{D\to\infty} \Pr\left[\ p^i \mid x\ \right] = 1/p^i \ . \tag{5.13}$$

In Kombination mit (5.10) erhält man

$$\lim_{D\to\infty} \frac{\Pr\left[\ p^i \mid h(\Delta)\ \right]}{\Pr\left[\ p^i \mid x\ \right]} \leqslant 1 + \frac{1}{p} \ . \tag{5.14}$$

Dies bedeutet, daß die Klassenzahlen zufälliger Fundamental-Diskriminanten mit höherer Wahrscheinlichkeit von kleineren Primzahlen geteilt werden, als zufällige ganze Zahlen. Wir erwarten daher, daß die Klassenzahl einer zufälligen Fundamental-Diskriminante mit höherer Wahrscheinlichkeit glatt ist, als eine zufällige ganze Zahl.

Seien $p_1, p_2, \ldots, p_k$ feste, paarweise verschiedene, ungerade Primzahlen und x zufällig aus $\{1, \ldots, \lfloor\sqrt{D}\rfloor\}$ gewählt. Es gilt

$$\lim_{D\to\infty} \Pr\left[\ \prod_{1\leqslant i\leqslant k} p_i^{e_i} \mid x\ \right] = \prod_{1\leqslant i\leqslant k} \Pr\left[\ p_i^{e_i} \mid x\ \right] \ . \tag{5.15}$$

Sei $m \in \{1, \ldots, \lfloor\sqrt{D}\rfloor\}$ ungerade und fest, x aus $\{1, \ldots, \lfloor\sqrt{D}\rfloor\}$ und eine Fundamental-Diskriminante Δ mit $|\Delta| \leqslant D$ zufällig gewählt. Wir schreiben $m = \prod_{p|m} p^{e_p(m)}$. Die Erweiterung zu (5.14) lautet dann

$$\lim_{D\to\infty} \frac{\Pr\left[\ m \mid h(\Delta)\ \right]}{\Pr\left[\ m \mid x\ \right]} = \frac{\prod_{p|m} \Pr\left[\ p^{e_p(m)} \mid h(\Delta)\ \right]}{\prod_{p|m} \Pr\left[\ p^{e_p(m)} \mid x\ \right]} \leqslant \prod_{p|m} \left(1 + \frac{1}{p}\right) \ . \tag{5.16}$$

Wir bestimmen den maximalen Wert für die obere Schranke. Der maximale Wert wird offensichtlich dann erreicht, wenn m aus möglichst vielen kleinen, unterschiedlichen Primzahlen besteht, also $m = \prod_{p\leqslant t} p$. Wir bestimmen nun t so, daß $m = \sqrt{D}$ wird. Die Tschebyschewsche θ-Funktion ist definiert durch

$$\theta(t) = \sum_{p\leqslant t} \ln p \ , \tag{5.17}$$

und es gilt $0.998684\,t < \theta(t) < 1.001102\,t$ für $t \geqslant 1319007$ [64]. Unter der Annahme der erweiterten Riemann-Vermutung kann sogar gezeigt werden, daß $|\theta(t) - t| = 1/(8\pi)\sqrt{t}\ln^2 t$ für $x \geqslant 599$ [67]. Somit ist $\prod_{p\leqslant t} p \approx e^t$ für $t \to \infty$. Daraus folgt, daß $t = \ln m = \ln\sqrt{D}$ sein muß. Wir erhalten als obere Schranke für das Produkt in (5.16)

$$\prod_{p\leqslant t} \left(1 + \frac{1}{p}\right) \leqslant \prod_{p\leqslant \ln\sqrt{D}} \left(1 + \frac{1}{p}\right) \approx \ln\ln\sqrt{D} \tag{5.18}$$

Die rechte Näherung kommt dadurch zustande, daß

$$1 + \frac{1}{p} = \frac{1 - 1/p^2}{1 - 1/p} \ ,$$

und

$$\prod_{p\leqslant t} \left(1 - \frac{1}{p}\right) = \frac{1}{e^\gamma \ln t} + O\left(\frac{1}{\ln^2 t}\right)$$

(Mertens' Theorem [62, Kapitel 12]), sowie

$$\prod_p \left(1 - \frac{1}{p^2}\right) = \frac{1}{\zeta(2)} = \frac{6}{\pi^2} \ .$$

Daher gilt für $t \to \infty$

$$\prod_{p \leqslant t} \left(1 + \frac{1}{p}\right) = c_5 \ln t \ , \tag{5.19}$$

wobei

$$c_5 = \frac{6e^\gamma}{\pi^2} \approx 1.082762 \ .$$

Damit wird (5.16) zu

$$\frac{\Pr\left[\ m \mid h(\Delta)\ \right]}{\Pr\left[\ m \mid x\ \right]} \leqslant c_5 \ln\ln D \tag{5.20}$$

für $D \to \infty$. Diese Abschätzung gilt für alle ungeraden $m \leqslant \sqrt{D}$.

Wir schätzen mit Hilfe von (5.20) die Glattheitswahrscheinlichkeit für Klassenzahlen ab. Sei $B \leqslant \sqrt{D}$ eine Glattheitsschranke, $h(\Delta)$ die Klassenzahl einer zufällig gewählten Fundamental-Diskriminante mit $|\Delta| \leqslant D$ und $x \leqslant \sqrt{D}$ eine positive Zufallszahl. Vergleicht man die Klassenzahlen $h(\Delta)$ mit den ganzen Zahlen $x \leqslant \sqrt{D}$, dann folgt aus (5.8) für $D \to \infty$, daß eine Klassenzahl mit vergleichsweise höherer Wahrscheinlichkeit durch kleine Primzahlen $p \leqslant B$ teilbar ist, als eine ganze Zahl; wir vernachlässigen im folgenden die erhöhte Wahrscheinlichkeit für Primzahlen $p > B$. Verallgemeinert folgt aus (5.12), daß eine Klassenzahl mit vergleichsweise höherer Wahrscheinlichkeit durch eine B-glatte Zahl m teilbar ist, als eine ganze Zahl; wir vernachlässigen im folgenden die erhöhte Wahrscheinlichkeit für nicht-B-glatte Zahlen m. Wir schließen daraus, daß eine Klassenzahl selber mit vergleichsweise höherer Wahrscheinlichkeit B-glatt ist, als eine ganze Zahl. Für eine obere Abschätzung der Glattheitswahrscheinlichkeit legen wir die obere Schranke aus (5.20) für alle B-glatten Klassenzahlen zugrunde und vernachlässigen die erhöhte Wahrscheinlichkeit für alle anderen Klassenzahlen, die glatte Teiler enthalten, aber selber nicht glatt sind. Dann ist für $D \to \infty$

$$\Pr\left[\ h(\Delta) \text{ ist } B\text{-glatt }\right] \leqslant c_5 \ln\ln D \cdot \Pr\left[\ x \text{ ist } B\text{-glatt }\right] \ . \tag{5.21}$$

Wenn wir D beschränken auf $D \leqslant 2^{4300}$, dann ist $c_5 \ln\ln\sqrt{D} < 2^3$. Somit ist die Wahrscheinlichkeit, daß die Klassenzahl einer zufälligen Fundamental-Diskriminante Δ, $|\Delta| \leqslant D$, B-glatt ist, höchstens 8 mal so groß wie die Wahrscheinlichkeit, daß eine Zufallszahl $x \leqslant \sqrt{D}$ B-glatt ist.

Die Wahrscheinlichkeit, daß eine Zufallszahl $x < \sqrt{D}$ B-glatt ist, wird recht gut durch die Dickmannsche ρ-Funktion angenähert, wobei ρ definiert ist als die stetige Lösung von

$$\begin{aligned} \rho(u) &= 1 \quad \text{für } 0 \leqslant u \leqslant 1 \ , \\ \rho(u-1) + u\rho'(u) &= 0 \quad \text{für } u > 1 \ . \end{aligned}$$

Es ist $\rho(u) = u^{-u+o(1)}$; eine Möglichkeit, $\rho(u)$ zu berechnen, ist die numerische Auswertung eines der Integrale

$$\rho(u) = \frac{1}{u} \int_{u-1}^{u} \rho(t)\, dt \tag{5.22}$$

Tabelle 5.1: Statistik der Diskriminanten mit $|\Delta|^{1/2u}$-glatter Klassenzahl

u	Anzahl	Anteil	$\rho(u)$
1.5	63089	0.58756	0.59453
2.0	32047	0.29846	0.30685
2.5	13380	0.12461	0.13033
3.0	4879	0.04544	0.04861
3.5	1580	0.01472	0.01623
4.0	472	0.0044	0.00491
4.5	120	0.00112	0.00137
5.0	30	0.00028	0.00035
5.5	5	0.00005	0.00009
6.0	1	0.00001	0.00002
6.5	1	0.00001	0.00000

oder

$$\rho(u) = 1 - \int_1^u \frac{\rho(t-1)}{t} dt \ . \tag{5.23}$$

Mit $x \leqslant \sqrt{D}$ und $u = \ln\sqrt{D}/\ln B$ ist [38, Theorem 1]

$$\Pr\big[\ x \text{ ist } B\text{-glatt } \big] = \rho(u)\left(1 + O_\epsilon\left(\frac{\ln(u+1)}{\ln B}\right)\right) \ ,$$

wobei $B \geqslant 2$, $1 \leqslant u \leqslant \exp((\ln B)^{3/5-\epsilon})$ und $\epsilon > 0$; die O-Konstante in dem Ausdruck ist nur von ϵ abhängig. Für die Klassenzahlen von zufälligen Fundamental-Diskriminanten Δ, $|\Delta| \leqslant D$, gilt dann entsprechend

$$\Pr\big[\ h(\Delta) \text{ ist } B\text{-glatt } \big] \leqslant c_5 \ln\ln D\rho(u)\left(1 + O_\epsilon\left(\frac{\ln(u+1)}{\ln B}\right)\right) \ . \tag{5.24}$$

Beispiel: Wir haben u. a. die Klassenzahlen aller primen Diskriminanten Δ mit $-10^{48} > \Delta \geqslant -10^{48} - 47\,523\,087$ und $\Delta \equiv 1 \pmod 8$ berechnet und vollständig faktorisiert. In diesem Intervall gibt es 107 374 solcher Diskriminanten. In Tabelle 5.1 haben wir die Statistik für die Diskriminanten mit $|\Delta|^{1/2u}$-glatter Klassenzahl für einige u zusammengefaßt und diese Zahlen mit den Werten für $\rho(u)$ verglichen. Die Tabelle zeigt, daß die Wahrscheinlichkeiten dafür, daß die Klassenzahlen zufälliger Fundamental-Diskriminanten glatt sind und daß zufällige ganze Zahlen glatt sind, viel dichter beieinander liegen, als vorausgesagt; die Voraussage scheint also zu pessimistisch zu sein. Das liegt u. a. auch daran, daß wir in der vorangegangenen Abschätzung für *alle* glatten Klassenzahlen den maximalen Wert für die obere Schranke in (5.20) verwendet haben.

5.2 Strategien zur Berechnung diskreter Logarithmen in Klassengruppen

In diesem Abschnitt betrachten wir diverse Strategien zur Berechnung von diskreten Logarithmen in Klassengruppen (IQ-DLP), und wir zeigen, welche Anforderungen erfüllt werden müssen, um die effiziente Berechnung diskreter Logarithmen zu verhindern.

5.2.1 Reduktion des IQ-DLP auf einfachere Berechnungsprobleme

Wir zeigen in diesem Abschnitt, daß die Wahl einer nicht-fundamentalen Diskriminante Nachteile mit sich bringt. Die Wahl einer nicht-fundamentalen Diskriminante wird motiviert durch die Schwierigkeit, die Klassenzahl einer fundamentalen Diskriminante zu bestimmen, und durch die folgende Tatsache:

Proposition 5.2.1
Sei Δ eine imaginär-quadratische Fundamental-Diskriminante, $\Delta \neq -3, -4$, und sei $f \in \mathbb{Z}_{>1}$. Dann ist

$$h(\Delta f^2) = h(\Delta) f \prod_{p|f} \left(1 - \left(\frac{\Delta}{p}\right)\frac{1}{p}\right) \quad , \tag{5.25}$$

wobei $\left(\frac{\Delta}{p}\right)$ das Kronecker-Symbol bezeichne.

Z. B. ist $h(-8) = 1$, und nach (5.25) ist $h(-8p^2) = p - \left(\frac{-8}{p}\right)$. Dieser Ansatz zur Wahl einer Klassengruppe erscheint attraktiv, weil man hier die Klassenzahl effizient berechnen kann. In diesem Fall wären Kryptoverfahren, welche explizit von der Gruppenordnung Gebrauch machen (z. B. DSA), sofort und direkt auf Klassengruppen übertragbar.

Das IQ-DLP in $\mathrm{Cl}(-8p^2)$ ist jedoch effizient reduzierbar auf das GF-DLP[1] in $\mathrm{GF}(p)^*$ [37]. Allgemein gilt:

Proposition 5.2.2
Sei Δ eine imaginär-quadratische Fundamental-Diskriminante, $\Delta \neq -3, -4$, und sei $f \in \mathbb{Z}_{>1}$. Dann ist das IQ-DLP in $\mathrm{Cl}(\Delta f^2)$ effizient reduzierbar auf das IQ-DLP in $\mathrm{Cl}(\Delta)$ und das GF-DLP in $\mathrm{GF}(p)^*$ für jede Primzahl $p \mid f$.

Diese Tatsache verbietet es, nicht-fundamentale Diskriminanten zu verwenden. Ist die Primfaktor-Zerlegung von Δ bekannt, dann ließe sich das IQ-DLP effizient auf eine Kombination einfacherer Probleme reduzieren; hält man die Primfaktor-Zerlegung geheim, dann hängt die Sicherheit des verwendeten Kryptoverfahrens an der Schwierigkeit, die Diskriminante zu faktorisieren (solche Kryptoverfahren werden z. B. in [74] ausführlich vorgestellt). Da in dieser Arbeit Kryptoverfahren beschrieben werden, die vom Faktorisierungsproblem (mutmaßlich) unabhängig sind, legen wir fest, daß nur fundamentale Diskriminanten für unsere Kryptoverfahren verwendet werden sollen.

Daß trotz der Einschränkung auf Fundamental-Diskriminanten immer noch genügend Diskriminanten zur Auswahl stehen, zeigt Theorem 333 aus [33] in Verbindung mit Definition 2.1.3 und Proposition 2.1.4:

[1] DLP für multiplikative Gruppen von Galois-Körpern

Theorem 5.2.3
Sei $Q(x)$ die Anzahl der quadratfreien Zahlen $\leqslant x$. Dann ist

$$Q(x) = \frac{6}{\pi^2}x + O(\sqrt{x}) \ . \tag{5.26}$$

Auswahl einer Fundamental-Diskriminante

Um zu prüfen, ob eine Diskriminante Δ fundamental ist, muß Δ (falls $\Delta \equiv 1 \pmod 4$) bzw. $\Delta/4$ (falls $\Delta \equiv 0 \pmod 4$) auf Quadratfreiheit überprüft werden. Es ist hierzu kein besseres Verfahren bekannt, als Δ zu faktorisieren, was für großes $|\Delta|$ praktisch unmöglich ist.

Die Alternative besteht darin, Δ gleich fundamental zu konstruieren, indem man nicht die Diskriminante selbst, sondern ihre Primfaktoren auswählt. Die einfachsten Fälle sind:

- $\Delta = -p$ mit $p \equiv 3 \pmod 4$;
- $\Delta = -4p$ mit $p \equiv 1 \pmod 4$;
- $\Delta = -8p$ mit $p \equiv 1, 3 \pmod 4$;
- $\Delta = -pq$ mit $pq \equiv 3 \pmod 4$;
- $\Delta = -4pq$ mit $pq \equiv 1 \pmod 4$;
- $\Delta = -8pq$ mit $pq \equiv 1, 3 \pmod 4$;
- $\vdots$

usw. Die Anzahl der unterschiedlichen Primfaktoren bestimmt den geraden Anteil der Klassengruppe (siehe Abschnitt 5.1.2). Der gerade Anteil der Klassengruppe soll möglichst klein sein, daher beschränken wir uns für unsere Kryptoverfahren auf die folgenden beiden Varianten:

1. $\Delta = -p$ mit $p \equiv 3 \pmod 4$;
2. $\Delta = -pq$ mit $pq \equiv 3 \pmod 4$ und $\left(\frac{p}{q}\right) = -1$.

Im ersten Fall ist $h(\Delta)$ ungerade (nach Proposition 5.1.8), und im zweiten Fall wird durch die Zusatzbedingung $\left(\frac{p}{q}\right) = -1$ sichergestellt, daß $h(\Delta)$ lediglich durch 2, aber nicht durch 4 Teilbar ist (nach Proposition 5.1.9).

Wir schränken damit die Auswahl der Fundamental-Diskriminante weiter ein. Aus dem Theorem von Dirichlet über die Verteilung von Primzahlen in Restklassen folgt, daß asymptotisch die Hälfte aller Primzahlen kongruent zu 3 modulo 4 sind. Somit stehen immer noch sehr viele Fundamental-Diskriminanten zur Auswahl zur Verfügung.

Zum Nachweis einiger kryptographisch notwendiger Eigenschaften haben wir in den Abschnitten 5.1.2 und 5.1.3 die Heuristiken von Cohen und Lenstra verwendet (Vermutungen 5.1.11 und 5.1.12, sowie Annahme 5.1.13). Diese Heuristiken wurden jedoch für alle Fundamental-Diskriminanten formuliert. In Erweiterung dazu machen wir daher folgende Annahme:

Tabelle 5.2: Zur Überprüfung der Vermutung 5.1.12 für Klassengruppen mit Diskriminante $\Delta = -p$, $p \equiv 7 \pmod 8$

p	$\Pr[\, p \mid h(\Delta)\,]$ nach (5.8)	$\Pr[\, p \mid h(\Delta)\,]$ empirisch
3	0.439873	0.4395
5	0.239667	0.2374
7	0.163205	0.1643
11	0.099167	0.0994
13	0.082838	0.0834
17	0.062283	0.0616
19	0.055401	0.0552

Annahme 5.2.4
Die Heuristiken von Cohen und Lenstra für imaginär-quadratische Körper gelten auch dann, wenn die Menge der imaginär-quadratischen Körper eingeschränkt wird auf diejenigen Körper, deren Diskriminante $\Delta = -p$ oder $\Delta = -pq$ ist.

Diese Annahme wird durch empirische Daten gestützt. Um z. B. die Vermutung 5.1.12 für spezielle Fundamental-Diskriminanten zu überprüfen, haben wir u. a. die Klassenzahlen aller Prim-Diskriminanten Δ im Intervall von -10^{48} bis $-10^{48} - 47\,523\,087$ mit $\Delta \equiv 1 \pmod 8$ untersucht. In diesem Intervall gibt es 107 374 solcher Diskriminanten. In Tabelle 5.2 haben wir die nach (5.8) vorausgesagte Wahrscheinlichkeit, daß p die Klassenzahl einer beliebigen Fundamental-Diskriminante teilt, mit der Statistik für die berechneten Klassenzahlen verglichen.

5.2.2 Angriffe mit Index-Berechnungs-Algorithmen

In diesem Abschnitt betrachten wir Index-Berechnungs-Algorithmen zur Berechnung diskreter Logarithmen in Klassengruppen. Diese Algorithmen haben asymptotisch eine subexponentielle Laufzeit. Wir werden die besten Algorithmen hierzu mit den besten Index-Berechnungs-Algorithmen zur Ganzzahl-Faktorisierung vergleichen. Auch diese Algorithmen haben asymptotisch eine subexponentielle Laufzeit. Der beste praktische Algorithmus zur Lösung des IQ-DLP ist eine Variante des MPQS[2] [40]. Der beste praktische Algorithmus zur Ganzzahl-Faktorisierung ist das GNFS[3] [46].

Wir definieren zunächst

$$L_x[\epsilon, c] = \exp\bigl(c(\ln x)^{\epsilon}(\ln\ln x)^{1-\epsilon}\bigr) \tag{5.27}$$

für jedes reelle $x \geqslant e \approx 2.718282$, jedes reelle $c > 0$ und $0 \leqslant \epsilon \leqslant 1$. Um eine Zahl n zu faktorisieren, benötigt das GNFS eine erwartete Laufzeit, die asymptotisch proportional zu $L_n\left[\frac{1}{3}, \sqrt[3]{\frac{64}{9}} + o(1)\right]$ ist (vgl. [47] oder [18], diese Laufzeit wurde bisher nicht bewiesen, sondern

[2]MPQS: Multiple Polynomial Quadratic Sieve
[3]GNFS: Generalized Number Field Sieve

nur heuristisch begründet). In [80] wurde unter Annahme der verallgemeinerten Riemann-Vermutung bewiesen, daß ein Index-Berechnungs-Algorithmus zur Lösung des IQ-DLP existiert, dessen erwartete Laufzeit asymptotisch proportional zu $L_{|\Delta|}\left[\frac{1}{2}, \frac{3}{4}\sqrt{2} + o(1)\right]$ ist. Bei Eingaben gleicher Größenordnung ist DL-Berechnung in Klassengruppen asymptotisch also viel aufwändiger als Ganzzahl-Faktorisierung. Dieser Algorithmus ist jedoch kein praktisches Verfahren. Jacobson hat in [40] eine Variante des MPQS-Faktorisierungs-Algorithmus zur Berechnung von diskreten Logarithmen in Klassengruppen quadratischer Körper vorgestellt (die Variante für imaginär-quadratische Körper bezeichnen wir im folgenden mit IQ-MPQS).

Der MPQS-Faktorisierungs-Algorithmus hat eine heuristisch erwartete Laufzeit, die asymptotisch proportional zu $L_n\left[\frac{1}{2}, 1 + o(1)\right]$ ist. Die erwartete Laufzeit des IQ-MPQS wurde bisher noch nicht analysiert. Der Lineare-Algebra-Schritt bei Index-Berechnungs-Algorithmen zur DL-Berechnung ist grundsätzlich aufwendiger, als bei Algorithmen zur Faktorisierung. Dennoch nehmen wir für das IQ-MPQS in dieser Arbeit ebenfalls eine erwartete Laufzeit an, die asymptotisch proportional zu $L_{|\Delta|}\left[\frac{1}{2}, 1 + o(1)\right]$ ist, weil diese Laufzeit zu unseren empirischen Messungen paßt. Indem wir diese Laufzeit der bewiesenen Laufzeit vorziehen, unterstellen wir, daß das IQ-MPQS schneller ist, als der bewiesene Algorithmus. Dies führt zu etwas größeren Schlüssellängen. Sollte sich herausstellen, daß die Annahme falsch und das IQ-MPQS tatsächlich langsamer ist, dann benutzt man im Zweifelsfall Schlüssel, die etwas größer sind, als erforderlich wäre (vgl. Tabellen 5.11 und 5.12 in Abschnitt 5.4).

Da die Laufzeit für Diskriminanten in kryptographisch relevanter Größenordnung nicht gemessen werden kann, muß ausgehend von Laufzeiten für kleiner Diskriminanten eine Extrapolation gemacht werden. Seien x_1 und x_2 Eingaben für einen Algorithmus mit einer erwarteten Laufzeit $L_x[\epsilon, c]$, und sei t_1 die Laufzeit dieses Algorithmus mit Eingabe x_1. Dann kann die Laufzeit t_2 des Algorithmus für die Eingabe x_2 abgeschätzt werden durch

$$\frac{L_{x_1}[\epsilon, c]}{L_{x_2}[\epsilon, c]} = \frac{t_1}{t_2} \ , \tag{5.28}$$

wie z. B. in [55] oder [47]. Wir haben in (5.28) den Term $o(1)$ ignoriert. Dies funktioniert allerdings nur, solange die Größen von x_1 und x_2 nicht zu stark differieren; andernfalls kann man nicht ignorieren, daß $o(1) \to 0$. Wenn x_2 also sehr viel größer als x_1 ist, dann wird t_2 zu sehr überschätzt. In [36] wurde der Versuch unternommen, den Term $o(1)$ bei dieser Abschätzung besser zu berücksichtigen. Da sich unsere Ergebnisse nur wenig von denen in [36] unterscheiden, bleiben wir bei (5.28).

In Tabelle 5.3 haben wir einige Laufzeiten für das GNFS für größere Eingaben extrapoliert. Als Daten-Punkt für diese Abschätzung diente uns die Faktorisierung von RSA-155 (155 Dezimal-Stellen, 512 Bits). Hierfür wurden in [18] 8000 MIPS-Jahre veranschlagt. Eine einfache Anwendung von (5.28) für größere Eingaben führt zu Tabelle 5.3.

Um die erwartete Laufzeit für das IQ-MPQS für zu extrapolieren, haben wir umfangreiche Experimente mit dem IQ-MPQS gemacht. Wir haben hierzu die Implementierung von Jacobson verwendet, die Bestandteil der C++-Bibliothek LiDIA [49] ist. Die Berechnungen wurden auf einer SUN mit SPARC-V8-Prozessor (170 MHz) gemacht. Die Ergebnisse sind in Tabelle 5.4 aufgelistet und in Abbildung 5.1 graphisch dargestellt. Für die Mittelwert-Bildung wurden jeweils ein diskreter Logarithmus in 20 unterschiedlichen Klassengruppen berechnet.

Wie man an der rechten Spalte von Tabelle 5.4 sehen kann, wird die Annahme für die erwartete Laufzeit von $L_{|\Delta|}\left[\frac{1}{2}, 1 + o(1)\right]$ für alle Diskriminanten-Größen unterstützt; die Quo-

Tabelle 5.3: Geschätzter Berechnungsaufwand für das GNFS für größere Eingaben

Größe von n	Erwartete Anz. der MIPS-Jahre um n zu faktorisieren
2^{512}	$8.00 \cdot 10^3$
2^{768}	$4.91 \cdot 10^7$
2^{1024}	$5.99 \cdot 10^{10}$
2^{1280}	$2.68 \cdot 10^{13}$
2^{1536}	$5.97 \cdot 10^{15}$
2^{1792}	$7.91 \cdot 10^{17}$
2^{2048}	$6.98 \cdot 10^{19}$
2^{2560}	$2.16 \cdot 10^{23}$
2^{3072}	$2.64 \cdot 10^{26}$
2^{3584}	$1.63 \cdot 10^{29}$
2^{4096}	$5.87 \cdot 10^{31}$

tienten $L_{|\Delta|}/\overline{t_\Delta}$ bewegen sich in einem relativ engen Intervall. Ausgehend von Tabelle 5.4 rechnen wir im folgenden mit $L_{|\Delta|}/\overline{t_\Delta} = c_6 = 1.8 \cdot 10^7\,\mathrm{sec}^{-1}$.

Um die Ergebnisse aus Tabelle 5.4 mit Tabelle 5.3 in Beziehung zu setzen, müssen wir die Laufzeiten aus 5.4 in MIPS-Jahre umrechnen. Die Einheit MIPS (Millionen CPU-Instruktionen pro Sekunde) ist keine geeignete Einheit, um die wahre Leistungsfähigkeit eines Rechners auszudrücken [73]. Der Grund liegt u. a. darin, daß die Eigenheiten moderner Rechner-Architekturen nicht berücksichtigt werden (z. B. Pipelining, Sprung-Voraussagen, Speicher-Caches usw.). Aus diesem Grund werden von SUN keine MIPS-Raten für SUN-Rechner veröffentlicht. Dennoch ist die Einheit MIPS bzw. MIPS-Jahr recht populär; für eine SUN mit SPARC-V8-Prozessor haben wir bei unseren Berechnungen eine Rate von etwa 100 MIPS zugrundegelegt.

Durch Extrapolation erhalten wir dann in Tabelle 5.5, wobei die Laufzeiten gleich in MIPS-Jahre umgerechnet wurden. Um einen Vergleich mit den Laufzeiten für das GNFS zu bekommen, muß $L_{|\Delta|}\left[\frac{1}{2}, 1\right]$ in Beziehung zu $L_n\left[\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right]$ gesetzt werden, wobei die Randbedingungen, d. h. die Daten-Punkte berücksichtigt werden müssen. Hierzu eignet sich das Newtonsche Näherungs-Verfahren hervorragend, da die Funktionen L_n und $L_{|\Delta|}$ und deren Ableitungen stetig und monoton wachsend sind. Das Ergebnis ist in Tabelle 5.6 zusammengefaßt und in Abbildung 5.2 veranschaulicht.

Tabelle 5.4: Empirischer Berechnungsaufwand des IQ-MPQS für relativ kleine Diskriminanten

Größe von $\lvert\Delta\rvert$	Laufzeit (sec)			Standard-Abweichung	$L_{\lvert\Delta\rvert}\left[\frac{1}{2},1\right] / \overline{t_\Delta}$ $(\sec^{-1})$
	Minimum	Mittel ($\overline{t_\Delta}$)	Maximum		
2^{140}	$4.75 \cdot 10^1$	$8.59 \cdot 10^1$	$2.01 \cdot 10^2$	$3.49 \cdot 10^1$	$1.65 \cdot 10^7$
2^{142}	$4.98 \cdot 10^1$	$1.29 \cdot 10^2$	$3.54 \cdot 10^2$	$8.66 \cdot 10^1$	$1.31 \cdot 10^7$
2^{144}	$7.03 \cdot 10^1$	$1.36 \cdot 10^2$	$2.48 \cdot 10^2$	$5.32 \cdot 10^1$	$1.50 \cdot 10^7$
2^{146}	$7.57 \cdot 10^1$	$1.32 \cdot 10^2$	$2.01 \cdot 10^2$	$3.87 \cdot 10^1$	$1.85 \cdot 10^7$
2^{148}	$9.87 \cdot 10^1$	$1.98 \cdot 10^2$	$4.22 \cdot 10^2$	$6.98 \cdot 10^1$	$1.47 \cdot 10^7$
2^{150}	$8.39 \cdot 10^1$	$2.20 \cdot 10^2$	$7.24 \cdot 10^2$	$1.38 \cdot 10^2$	$1.59 \cdot 10^7$
2^{152}	$1.38 \cdot 10^2$	$2.63 \cdot 10^2$	$7.07 \cdot 10^2$	$1.44 \cdot 10^2$	$1.59 \cdot 10^7$
2^{154}	$1.84 \cdot 10^2$	$3.26 \cdot 10^2$	$9.57 \cdot 10^2$	$1.82 \cdot 10^2$	$1.53 \cdot 10^7$
2^{156}	$1.62 \cdot 10^2$	$3.52 \cdot 10^2$	$7.38 \cdot 10^2$	$1.64 \cdot 10^2$	$1.69 \cdot 10^7$
2^{158}	$1.62 \cdot 10^2$	$4.90 \cdot 10^2$	$1.47 \cdot 10^3$	$3.28 \cdot 10^2$	$1.44 \cdot 10^7$
2^{160}	$2.12 \cdot 10^2$	$4.41 \cdot 10^2$	$8.20 \cdot 10^2$	$1.98 \cdot 10^2$	$1.90 \cdot 10^7$
2^{162}	$3.01 \cdot 10^2$	$7.67 \cdot 10^2$	$2.07 \cdot 10^3$	$4.21 \cdot 10^2$	$1.30 \cdot 10^7$
2^{164}	$3.53 \cdot 10^2$	$6.84 \cdot 10^2$	$1.21 \cdot 10^3$	$2.20 \cdot 10^2$	$1.73 \cdot 10^7$
2^{166}	$3.60 \cdot 10^2$	$8.79 \cdot 10^2$	$1.56 \cdot 10^3$	$3.22 \cdot 10^2$	$1.60 \cdot 10^7$
2^{168}	$4.89 \cdot 10^2$	$1.07 \cdot 10^3$	$2.29 \cdot 10^3$	$4.12 \cdot 10^2$	$1.56 \cdot 10^7$
2^{170}	$4.79 \cdot 10^2$	$1.49 \cdot 10^3$	$3.23 \cdot 10^3$	$8.25 \cdot 10^2$	$1.33 \cdot 10^7$
2^{172}	$7.28 \cdot 10^2$	$1.74 \cdot 10^3$	$3.82 \cdot 10^3$	$8.99 \cdot 10^2$	$1.34 \cdot 10^7$
2^{174}	$6.57 \cdot 10^2$	$1.54 \cdot 10^3$	$4.08 \cdot 10^3$	$9.83 \cdot 10^2$	$1.79 \cdot 10^7$
2^{176}	$6.88 \cdot 10^2$	$1.61 \cdot 10^3$	$3.42 \cdot 10^3$	$8.45 \cdot 10^2$	$2.03 \cdot 10^7$
2^{178}	$1.20 \cdot 10^3$	$2.77 \cdot 10^3$	$6.61 \cdot 10^3$	$1.37 \cdot 10^3$	$1.39 \cdot 10^7$
2^{180}	$1.18 \cdot 10^3$	$2.73 \cdot 10^3$	$7.16 \cdot 10^3$	$1.39 \cdot 10^3$	$1.67 \cdot 10^7$
2^{184}	$1.58 \cdot 10^3$	$3.37 \cdot 10^3$	$8.88 \cdot 10^3$	$1.82 \cdot 10^3$	$1.87 \cdot 10^7$
2^{188}	$2.01 \cdot 10^3$	$4.07 \cdot 10^3$	$9.99 \cdot 10^3$	$1.95 \cdot 10^3$	$2.14 \cdot 10^7$
2^{192}	$3.42 \cdot 10^3$	$5.96 \cdot 10^3$	$1.44 \cdot 10^4$	$2.86 \cdot 10^3$	$2.02 \cdot 10^7$
2^{196}	$3.99 \cdot 10^3$	$9.23 \cdot 10^3$	$1.80 \cdot 10^4$	$3.80 \cdot 10^3$	$1.79 \cdot 10^7$
2^{200}	$5.71 \cdot 10^3$	$1.36 \cdot 10^4$	$2.41 \cdot 10^4$	$5.12 \cdot 10^3$	$1.66 \cdot 10^7$
2^{210}	$1.37 \cdot 10^4$	$2.61 \cdot 10^4$	$4.22 \cdot 10^4$	$8.17 \cdot 10^3$	$1.89 \cdot 10^7$
2^{220}	$2.85 \cdot 10^4$	$5.71 \cdot 10^4$	$1.33 \cdot 10^5$	$2.37 \cdot 10^4$	$1.85 \cdot 10^7$

Tabelle 5.5: Geschätzter erwarteter Berechnungsaufwand des IQ-MPQS für größere Diskriminanten

Größe von $\lvert\Delta\rvert$	Erwartete Anz. der MIPS-Jahre um das IQ-DLP in Cl(Δ) zu lösen
2^{256}	2.58
2^{348}	$9.75 \cdot 10^{3}$
2^{512}	$1.18 \cdot 10^{7}$
2^{640}	$6.74 \cdot 10^{9}$
2^{768}	$2.24 \cdot 10^{12}$
2^{896}	$4.94 \cdot 10^{14}$
2^{1024}	$7.79 \cdot 10^{16}$
2^{1280}	$8.90 \cdot 10^{20}$
2^{1536}	$4.56 \cdot 10^{24}$
2^{1792}	$1.26 \cdot 10^{28}$
2^{2048}	$2.13 \cdot 10^{31}$
2^{2560}	$1.92 \cdot 10^{37}$
2^{3072}	$5.30 \cdot 10^{42}$
2^{3584}	$5.88 \cdot 10^{47}$
2^{4096}	$3.15 \cdot 10^{52}$

Tabelle 5.6: Geschätzter erwarteter Berechnungsaufwand für das GNFS und das IQ-MPQS für vergleichbare Eingaben

Größe von		Erwartete Anz. der MIPS-Jahre
n	$\lvert\Delta\rvert$	
2^{768}	2^{540}	$4.99 \cdot 10^{7}$
2^{1024}	2^{687}	$6.01 \cdot 10^{10}$
2^{1536}	2^{958}	$5.95 \cdot 10^{15}$
2^{2048}	2^{1208}	$7.05 \cdot 10^{19}$
2^{3072}	2^{1665}	$2.65 \cdot 10^{26}$
2^{4096}	2^{2084}	$5.87 \cdot 10^{31}$

Abbildung 5.1: Empirischer Berechnungsaufwand des IQ-MPQS für relativ kleine Diskriminanten

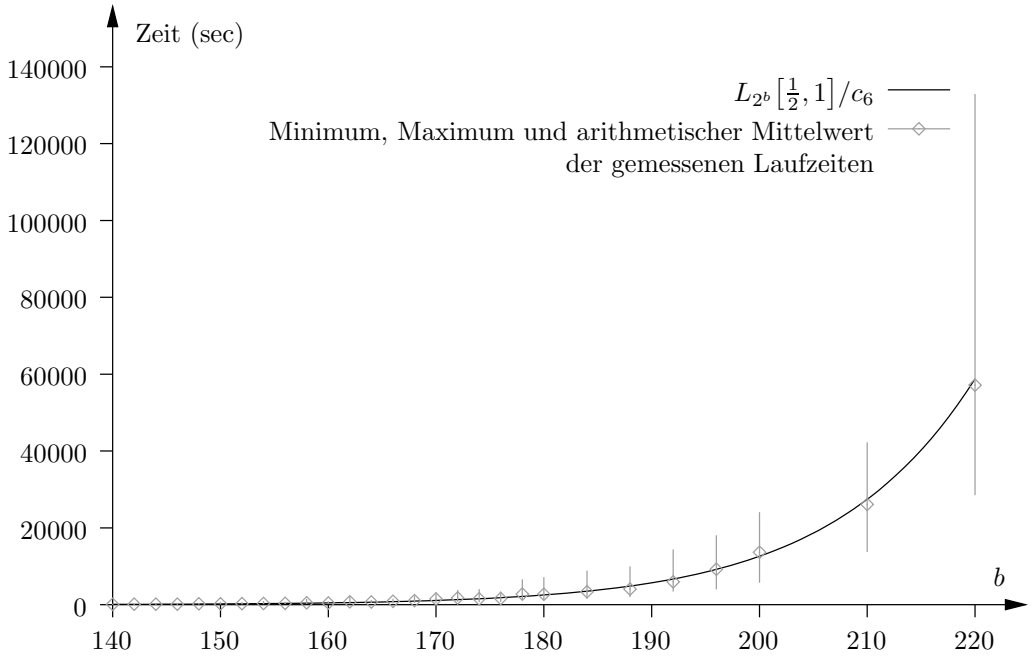
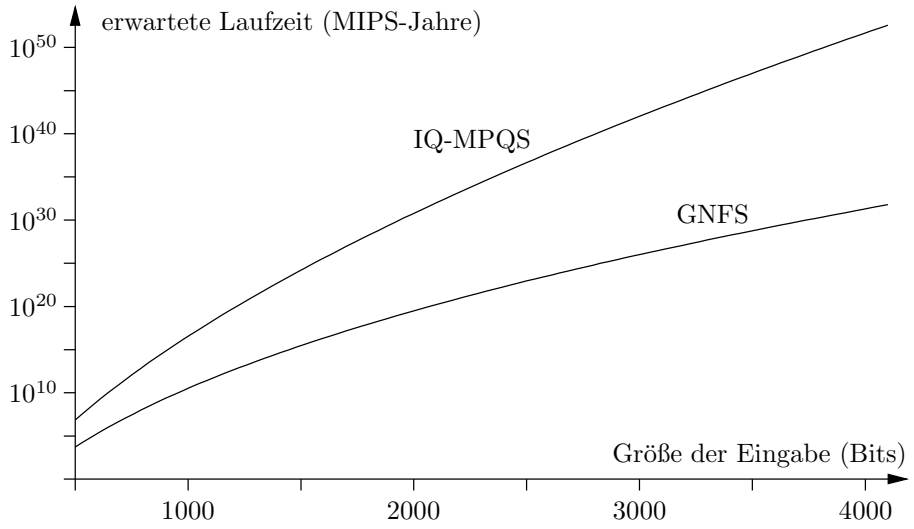


Abbildung 5.2: Graphischer Vergleich der erwarteten Laufzeiten des IQ-MPQS und des GNFS



5.2.3 Angriffe mit Quadratwurzel-Algorithmen

Wir betrachten nun die sog. Quadratwurzel-Algorithmen zur Berechnung diskreter Logarithmen. Es handelt sich dabei um Pollards ρ- und λ-Algorithmus, sowie um Shanks Baby-Step-Giant-Step-Algorithmus (BSGS) [42]. Diese Algorithmen arbeiten in beliebigen endlichen, Abelschen Gruppen $\mathcal{G}$ und benötigen $O(\sqrt{|\mathcal{G}|})$ Gruppenoperationen (für eine Übersicht, siehe z. B. [77]). Um die effiziente Berechnung von diskreten Logarithmen in $\mathcal{G}$ mit diesen Algorithmen zu verhindern, muß also $|\mathcal{G}|$ sehr groß sein. In Abschnitt 5.1.1 haben wir gesehen, daß die Klassenzahl einer großen Diskriminante selber sehr groß ist. Deshalb sind die Quadratwurzel-Algorithmen in Klassengruppen nicht effizient, wenn die Diskriminante sehr groß ist.

Der BSGS-Algorithmus ist ein deterministischer Algorithmus. Dieser Algorithmus funktioniert auch dann, wenn die Gruppenordnung unbekannt ist, aber eine Abschätzung vorliegt (in [19, Abschnitt 5.4.3] ist ein BSGS-Algorithmus zur Berechnung der Klassengruppe angegeben). Terr hat in [75] eine Variante vorgestellt, die auch dann gut funktioniert, wenn die Gruppenordnung komplett unbekannt ist. Bei den BSGS-Algorithmen müssen im Gegensatz zu dem ρ- oder λ-Algorithmus auch $O(\sqrt{|\mathcal{G}|})$ Gruppenelemente abgespeichert werden. Der ρ- und der λ-Algorithmus sind probabilistische Algorithmen, die sehr speichereffizient sind. Um den ρ-Algorithmus in einer Gruppe anzuwenden, muß die Gruppenordnung bekannt sein; der ρ-Algorithmus ist im Fall von Klassengruppen daher nicht verwendbar. Für den λ-Algorithmus (der auch als „Känguru-Algorithmus“ bekannt ist) hingegen genügt eine Abschätzung der Gruppenordnung. Wir werden daher im folgenden den λ-Algorithmus zur Grundlage unserer Betrachtung machen.

Für die einfache (nicht-parallelisierte) Version des λ-Algorithmus werden etwa $\sqrt{\pi|\mathcal{G}|/2}$ Gruppenoperationen erwartet (wir ignorieren hierbei die Terme niedrigerer Ordnung). Für Klassengruppen werden demnach mindestens

$$\sqrt{\pi\sqrt{|\Delta|}/(4c_3 \ln\ln|\Delta|)} \tag{5.29}$$

Gruppenoperationen erwartet (siehe Abschnitt 5.1.1); bei einer Beschränkung der Diskriminante auf $|\Delta| \leqslant 2^{4300}$ werden nach (5.7) mindestens

$$\sqrt{\pi\sqrt{|\Delta|}/2^7} > \frac{\sqrt[4]{|\Delta|}}{2^3} \tag{5.30}$$

Gruppenoperationen erwartet. Eine r-fache Parallelisierung bringt eine r-fache Beschleunigung [79], [76].

In Abschnitt 5.2.2 haben wir das IQ-MPQS mit den GNFS zur Ganzzahl-Faktorisierung verglichen, indem wir für feste Parameter für das GNFS die Parameter für das IQ-MPQS so bestimmt haben, daß die erwarteten Laufzeiten des IQ-MPQS und des GNFS jeweils etwa gleich waren. Wir werden nun wiederum das GNFS als Bezug wählen, d. h. wir werden zu festen Parametern für das GNFS die Parameter für die einfache (nicht-parallelisierte) Version des λ-Algorithmus so auswählen, daß die erwarteten Laufzeiten für beide Algorithmen etwa gleich sind.

Wir legen wiederum eine Rate von 100 MIPS zugrunde. Eine einzelne Gruppenoperation (d. h. eine Ideal-Multiplikation mit anschließender Reduktion) benötigt eine Zeit proportional zu $\log^2|\Delta|$ (siehe Abschnitt 7.1.4). Für die folgende Rechnung machen wir die Vereinfachung, daß die Zeit für eine Gruppenoperation fest 1 ms betrage (auf einer 100-MIPS-Maschine);

Tabelle 5.7: Geschätzter erwarteter Berechnungsaufwand für den λ-Algorithmus

Größe von $\lvert\Delta\rvert$	erwartete Anz. der Gruppenoperationen $\sqrt{\pi h(\Delta)/2}$	Erwartete Anz. der MIPS-Jahre
2^{228}	2^{54}	$5.72 \cdot 10^{7}$
2^{268}	2^{64}	$5.86 \cdot 10^{10}$
2^{334}	2^{81}	$5.43 \cdot 10^{15}$
2^{389}	2^{94}	$7.48 \cdot 10^{19}$
2^{476}	2^{116}	$2.64 \cdot 10^{26}$
2^{547}	2^{134}	$5.82 \cdot 10^{31}$

wir überschätzen damit z. B. einen Angreifer für $\Delta \to -\infty$. Dann entspräche die Arbeit von einem MIPS-Jahr etwa $10^3 \cdot 60^2 \cdot 24 \cdot 365/100 \approx 2^{28.23}$ Gruppenoperationen.

Mit Hilfe von Tabelle 5.3 und (5.29) oder (5.30) (falls $|\Delta| \leqslant 2^{4300}$) kann man dann ausrechnen, wie groß $|\Delta|$ mindestens sein muß, damit die erwarteten Laufzeiten des λ-Algorithmus und des GNFS für vorgegebene Eingabegrößen etwa gleich sind. Einige Werte sind sind in Tabelle 5.7 zusammengefaßt.

5.2.4 Angriffe mit dem Pohlig-Hellman-Algorithmus

Der Pohlig-Hellman-Algorithmus [56] ist ein Verfahren, um die Berechnung diskreter Logarithmen in einer endlichen Gruppe $\mathcal{G}$ zu vereinfachen, wenn die Primzahl-Zerlegung von $|\mathcal{G}|$ bekannt ist. Dieser Algorithmus läßt sich insbesondere dann vorteilhaft einsetzen, wenn in der Primfaktor-Zerlegung von $|\mathcal{G}|$ nur kleine Primzahlen vorkommen, d. h. wenn $|\mathcal{G}|$ sehr glatt ist. Der beste, bekannte Algorithmus, um die Klassenzahl einer Klassengruppe auszurechnen, ist eine Variante des IQ-MPQS mit derselben erwarteten Laufzeit. Wenn $|\Delta|$ groß ist, dann kann $h(\Delta)$ praktisch nicht berechnet werden. Darüberhinaus ist auch kein effizienter Algorithmus bekannt, der Vielfache oder Teiler von $h(\Delta)$ berechnet, im Speziellen ist auch kein Algorithmus bekannt, der den glatten Anteil von $h(\Delta)$ effizient berechnet. Somit ist der Pohlig-Hellman-Algorithmus für beliebige Klassengruppen i. allg. nicht anwendbar.

In dem speziellen Fall, daß $h(\Delta)$ sehr glatt ist, gibt es allerdings eine Möglichkeit, den Pohlig-Hellman-Algorithmus effizient anzuwenden. In diesem Fall kann auch die Primfaktorisierung der Ordnung einer Untergruppe von $\mathrm{Cl}(\Delta)$ effizient bestimmt werden. In Abschnitt 5.1.3 haben wir gesehen, daß die Wahrscheinlichkeit sehr gering ist, daß die Klassenzahl einer zufällig gewählten Fundamental-Diskriminante sehr glatt ist, wenn die Diskriminante sehr groß ist. Der Pohlig-Hellman-Algorithmus ist deshalb bei zufälliger Wahl einer großen Fundamental-Diskriminante mit sehr hoher Wahrscheinlichkeit nicht effizient anwendbar.

Algorithmus 5.1 bestimmt den größten Primfaktor p von $|\langle\mathfrak{a}\rangle|$ für ein beliebiges $\mathfrak{a} \in \mathrm{Cl}(\Delta)$, falls $p \leqslant B$ für eine Glattheitsschranke B.

Indem man in Algorithmus 5.1 $\mathfrak{a}$ durch $\mathfrak{a}^{p_i^j}$ ersetzt, kann sukzessive die gesamte Primfaktorisierung von $|\langle\mathfrak{a}\rangle|$ berechnet werden; dieses Verfahren bezeichnen wir als smoothOrder. Da wir $h(\Delta)$ als glatt vorausgesetzt hatten, kann dann der Pohlig-Hellmann-Algorithmus effizient angewendet werden, um diskrete Logarithmen in $\langle\mathfrak{a}\rangle$ zu berechnen.

Algorithmus 5.1 Berechne den größten Faktor von $|\langle \mathfrak{a} \rangle|$, wenn $h(\Delta)$ glatt ist

Eingabe:

1. Eine ungerade Diskriminante Δ;
2. ein Element $\mathfrak{a} \in \mathrm{Cl}(\Delta)$;
3. eine Glattheitsschranke B;
4. eine Liste aller Primzahlen p_i zwischen 2 und B in aufsteigender Reihenfolge, mit $p_1 = 2$;
5. Exponenten $e(p_i, B)$ (s. u.).

Ausgabe: (p, e), wobei p der größte Primfaktor von $|\langle \mathfrak{a} \rangle|$ ist und $p^e \,\|\, |\langle \mathfrak{a} \rangle|$ oder $(1, 1)$.

```
1:  b ← a
2:  i ← 1
3:  while p_i ⩽ B do
4:     for j = 1 to e(p_i, B) do
5:        b ← b^{p_i}
6:        if b = 1_{Cl(Δ)} then return (p_i, j)
7:     end for
8:     i ← i + 1
9:  end while
10: return (1, 1)
```

Wählt man $\mathfrak{a}$ jeweils zufällig aus $\mathrm{Cl}(\Delta)$, so hat man nach wenigen Versuchen mit hoher Wahrscheinlichkeit den Gruppenexponenten von $\mathrm{Cl}(\Delta)$ berechnet. Ein ähnlicher Verfahren kommt auch in dem Faktorisierungs-Algorithmus von Schnorr und Lenstra [66] zum Einsatz (siehe z. B. [19, Abschnitt 10.2]). Es besteht auch eine Ähnlichkeit zu Pollards $(p-1)$-Faktorisierungs-Algorithmus (siehe z. B. [19, Abschnitt 8.8]), und genau wie bei dem $(p-1)$-Algorithmus kann man in Algorithmus 5.1 eine zweite Stufe zwischen Zeile 9 und 10 einfügen, bei der man nach einer einfachen Vorberechnung nur mit Hilfe von einfachen Multiplikationen prüft, ob eine der Primzahlen zwischen B und einem $B' > B$ ein Teiler von $|\langle \mathfrak{a} \rangle|$ ist.

Wir diskutieren die Laufzeit von Algorithmus 5.1. Wenn der Algorithmus Erfolg hat (Zeile 6), dann teilt p_i^j offensichtlich die Ordnung von $|\langle \mathfrak{a} \rangle|$; darüberhinaus ist offensichtlich, daß p_i die größte Primzahl ist, die $|\langle \mathfrak{a} \rangle|$ teilt. Wenn Algorithmus 5.1 lediglich das Ergebnis $(1, 1)$ zurück liefert, dann war die Glattheitsschranke B zu klein, oder mindestens einer der Exponenten $e(p_i, B)$ war zu klein.

Sei q der größte Primteiler von $|\langle \mathfrak{a} \rangle|$. Dann werden in Algorithmus 5.1 mit einem Standard-Verfahren zur schnellen Exponentiation [51, Abschnitt 14.6] mindestens

$$\sum_{p \leqslant q} e(p, B) \lfloor \log_2 p \rfloor \tag{5.31}$$

Gruppenoperationen durchgeführt, bis q gefunden wird. Der einfachste Fall ist $e(p_i, B) = 1$ für alle p_i. Es ist $\sum_{p \leqslant q} \lfloor \log_2 p \rfloor \leqslant \theta(q) / \ln 2$ (siehe (5.17)). Wie bereits erwähnt, ist $\theta(q) \approx q$ für $q \to \infty$. Mit dieser Wahl für $e(p_i, B)$ müssen also in der Größenordnung q viele Gruppenoperationen gemacht werden, um q zu finden. Eine andere sinnvolle Wahl für die Exponenten

ist $e(p_i, B) = \lfloor \log_{p_i} q \rfloor$. Diese Wahl kommt der Tatsache entgegen, daß kleine Primzahlen mit höherer Wahrscheinlichkeit in höheren Potenzen in einer Klassenzahl vorkommt. Das Ergebnis ist jedoch im wesentlichen dasselbe, denn $\sum_{p \leqslant q} \lfloor \log_2 p \rfloor \lfloor \log_p q \rfloor \leqslant \pi(q) \log_2 q = q/\ln 2$ für $q \to \infty$, wobei $\pi(q)$ die Anzahl aller Primzahlen $\leqslant q$ ist. Diese Wahl von $e(p_i, B)$ werden wir für die nachfolgenden Überlegungen zugrundelegen.

Zu einer zufällig ausgewählten Fundamental-Diskriminante Δ und einer beliebigen Glattheitsschranke $B \leqslant h(\Delta)$ kann man nicht effizient entscheiden, ob die Klassenzahl $h(\Delta)$ B-glatt ist. Ist z. B. ein öffentlicher IQ-DH-Schlüssel $\mathfrak{a}$ und eine Basis $\mathfrak{g}$ gegeben (dargestellt durch die reduzierten Vertreter), dann kann ein Angreifer a priori nicht entscheiden, ob der Algorithmus smoothOrder für die Berechnung des geheimen Schlüssels $a = \log_{\mathfrak{g}} \mathfrak{a}$ bei gegebener Glattheitsschranke Erfolg haben wird, oder nicht. Eine ähnliche Problematik besteht bei der Auswahl eines RSA-Modulus im Zusammenhang mit dem $(p-1)$-Faktorisierungs-Algorithmus. Ist ein Angriff mit einer Schranke B erfolglos, dann muß B größer gewählt werden, was zu einer entsprechend längeren Laufzeit für den Algorithmus führt.

Um die Größe von D zu bestimmen, so daß Algorithmus 5.1 mit hoher Wahrscheinlichkeit nicht den größten Primfaktor der Klassenzahl $h(\Delta)$ einer zufällig gewählten Fundamental-Diskriminante Δ mit $|\Delta| \leqslant D$ findet, verwenden wir das folgende Modell: In einem Experiment versuche ein Angreifer zu gebener Glattheitsschranke B den größten Primteiler von zufälligen Fundamental-Diskriminanten zu bestimmen. Sei W_{max} der maximale Berechnungsaufwand, den der Angreifer aufzuwenden bereit ist für einen Versuch mit Algorithmus 5.1 für eine zufällige Diskriminante Δ und ein zufälliges $\mathcal{O}_\Delta$-Ideal $\mathfrak{g}$ ($\mathfrak{g} = [\mathfrak{g}]$). Ist p der größte Primteiler von $h(\Delta)$, dann ist $\Pr[\, p \nmid |\langle \mathfrak{g} \rangle| \,] \leqslant 1/p$, wenn $\mathfrak{g}$ mit Gleichverteilung aus $\mathrm{Cl}(\Delta)$ ausgwählt wird (siehe Abschnitt 5.3). Ist p sehr groß, dann ist diese Wahrscheinlichkeit vernachlässigbar. Unter der Annahme, daß der größte Primfaktor p von $h(\Delta)$ auch $|\langle \mathfrak{g} \rangle|$ teilt, ist $|\langle \mathfrak{g} \rangle|$ genau dann B-glatt, wenn $h(\Delta)$ B-glatt ist, und dann ist

$$W_{\mathsf{total}} = \frac{W_{\mathsf{max}}}{\Pr\left[\ h(\Delta) \text{ ist } B\text{-glatt } \right]} \tag{5.32}$$

der erwartete Berechnungsaufwand, bis Algorithmus 5.1 bei einer Diskriminante Erfolg hat. Es müssen also zwei Parameter festgelegt werden.

Wie in den vorigen Abschnitten legen wir zugrunde, daß eine Gruppenoperation auf einer 100-MIPS-Maschine 1 ms benötige, dann entsprechen $2^{28.23}$ Gruppenoperationen einem MIPS-Jahr. Ein realistischer Wert für W_{max} ist z. B. $W_{\mathsf{max}} = 2^{42}$ Gruppenoperationen, was knapp 14000 MIPS-Jahren entspricht; dies ist in der Größenordnung dessen, was zur Faktorisierung einer 512-Bit Zahl aufgewendet werden muß (vgl. Tabelle 5.3).

Wenn man W_{total} und $\Pr_{\Delta,B} = \Pr[\, h(\Delta) \text{ ist } B\text{-glatt} \,]$ festlegt, dann ergibt sich mit (5.24) aus Abschnitt 5.1.3 für $|\Delta| \leqslant 2^{4300}$, daß $\Pr_{\Delta,B} = W_{\mathsf{total}}/W_{\mathsf{max}} \leqslant 8\rho(u)$ ist, d. h. $\rho(u) \geqslant W_{\mathsf{total}}/8W_{\mathsf{max}}$. u kann z. B. durch Tabellen für $\rho(u)$ bestimmt werden. Über die Beziehung $|\Delta| \geqslant B^{2u}$ kann schließlich die Größenordnung von Δ bestimmt werden.

Beispiel: Gefordert sei eine Glattheitswahrscheinlichkeit von $\Pr_{\Delta,B} = 2^{-32}$. Wir bestimmen eine Glattheitsschranke B und die Größenordnung von Δ, damit erst für den Gesamtaufwand $W_{\mathsf{total}} = 2^{64}$ ein Erfolg für Algorithmus 5.1 zu erwarten ist. Damit ist $W_{\mathsf{max}} = W_{\mathsf{total}} \cdot \Pr_{\Delta,B} = 2^{32}$ nach (5.32) festgelegt. Bei der zuvor zugrundegelegten Wahl für $e(p_i, B)$ in Algorithmus 5.1 ergibt sich die Glattheitsschranke $B = 2^{32}$ (wir vernachlässigen hier den Faktor $1/\ln(2) \approx 1.442695$).

Tabelle 5.8: Größe der Diskriminante zu gegebenem Aufwand W_{total} für verschiedene Glattheitswahrscheinlichkeiten $\Pr_{\Delta,B}$

W_{total}	$\Pr_{\Delta,B}$	B, W_{max}	$\Pr_{\Delta,B} = 8\rho(u)$			$\Pr_{\Delta,B} = \rho(u)$		
			$\rho(u)$	u	Größe von $\lvert\Delta\rvert$	$\rho(u)$	u	Größe von $\lvert\Delta\rvert$
2^{64}	2^{-24}	2^{40}	2^{-27}	8.43	2^{675}	2^{-24}	7.82	2^{626}
	2^{-32}	2^{32}	2^{-35}	9.99	2^{640}	2^{-32}	9.42	2^{603}
	2^{-40}	2^{24}	2^{-43}	11.47	2^{551}	2^{-40}	10.92	2^{525}
2^{81}	2^{-24}	2^{57}	2^{-27}	8.43	2^{961}	2^{-24}	7.82	2^{892}
	2^{-32}	2^{49}	2^{-35}	9.99	2^{979}	2^{-32}	9.42	2^{924}
	2^{-40}	2^{41}	2^{-43}	11.47	2^{941}	2^{-40}	10.92	2^{896}
2^{94}	2^{-24}	2^{70}	2^{-27}	8.43	2^{1181}	2^{-24}	7.82	2^{1095}
	2^{-32}	2^{62}	2^{-35}	9.99	2^{1239}	2^{-32}	9.42	2^{1168}
	2^{-40}	2^{54}	2^{-43}	11.47	2^{1239}	2^{-40}	10.92	2^{1180}
2^{116}	2^{-24}	2^{92}	2^{-27}	8.43	2^{1552}	2^{-24}	7.82	2^{1439}
	2^{-32}	2^{84}	2^{-35}	9.99	2^{1679}	2^{-32}	9.42	2^{1583}
	2^{-40}	2^{76}	2^{-43}	11.47	2^{1744}	2^{-40}	10.92	2^{1660}
2^{134}	2^{-24}	2^{110}	2^{-27}	8.43	2^{1855}	2^{-24}	7.82	2^{1721}
	2^{-32}	2^{102}	2^{-35}	9.99	2^{2038}	2^{-32}	9.42	2^{1922}
	2^{-40}	2^{94}	2^{-43}	11.47	2^{2157}	2^{-40}	10.92	2^{2053}

Die Größenordnung von Δ ergibt sich durch $|\Delta| \geqslant B^{2u}$; für eine so gewählte Fundamental-Diskriminante ergibt sich mit der Glattheitsschranke $B = 2^{32}$ die Glattheitswahrscheinlichkeit $\Pr_{\Delta,B} \leqslant 2^{-32}$. Wir bestimmen nun u. Wir legen zugrunde, daß $\Pr_{\Delta,B} = 8\rho(u)$ ist, d. h. $2^{-3}\Pr_{\Delta,B} = 2^{-35} = \rho(u)$. Aus Tabellen für $\rho(u)$ läßt sich u mit 9.99 bestimmen, und damit ergibt sich $|\Delta| \geqslant B^{2u} = 2^{640}$. Legt man für $\Pr_{\Delta,B}$ nicht $8\rho(u)$ sondern $\rho(u)$ zugrunde, wie durch Tabelle 5.1 in Abschnitt 5.1.3 nahegelegt wird, dann erhält man für u den Wert 9.42, und es ergibt sich $|\Delta| \geqslant B^{2u} = 2^{603}$.

In Tabelle 5.8 haben wir die Berechnungen zu Δ für verschiedene W_{total} und geforderte Glattheitswahrscheinlichkeiten $\Pr_{\Delta,B}$ tabelliert. Wir haben für die Glattheitswahrscheinlichkeit in den mittleren drei Spalten den maximalen Wert $8\rho(u)$ für $\Pr_{\Delta,B}$ zugrundegelegt; in den rechten drei Spalten haben wir für $\Pr_{\Delta,B}$ den hypothetischen Wert $\rho(u)$ nach Tabelle 5.1 zugrundegelegt. Die Laufzeiten für W_{total} in Tabelle 5.8 kann man aus Tabelle 5.7 entnehmen, wenn man W_{total} dort in Spalte 2 abliest.

Bei der Interpretation der Tabelle muß berücksichtigt werden, daß für die Glattheitswahrscheinlichkeit eine pessimistische obere Schranke verwendet wurde, für eine Gruppenoperation eine feste Zeit von 1 ms veranschlagt wurde, und daß bei der Berechnung der Größe der Diskriminante die Laufzeit von Algorithmus 5.1 und nicht von smoothOrder zugrunde gelegt wurde. Es muß auch berücksichtigt werden, daß der Aufwand für den Pohlig-Hellman-Algorithmus in die Berechnung nicht mit einbezogen wurde. Tabelle 5.8 vermittelt daher nur einen Eindruck von den ungefähren Größenordnungen, die eine Diskriminante haben muß, um einen Angriff gegen Klassengruppen mit glatten Klassenzahlen zu verhindern.

5.3 Die Auswahl eines zufälligen, reduzierten Ideals

Für die IQ-basierten Kryptoverfahren benötigen wir zufällige Basiselemente. Diese Basiselemente sollen eine möglichst große Untergruppe der Klassengruppe erzeugen, und die Ordnung dieser Basiselemente soll einen möglichst großen Primfaktor enthalten. In diesem Abschnitt gehen wir auf die Auswahl eines Zufallsideals ein. Die Klassenzahl einer zufälligen, großen Diskriminante kann praktisch nicht berechnet werden, ebenso kann die Ordnung eines zufälligen Elements der Klassengruppe praktisch nicht berechnet werden. Daher kann von einem zufälligen Element aus der Klassengruppe auch praktisch nicht festgestellt werden, ob es sich um einen Erzeuger der Klassengruppe (oder einer großen Untergruppe davon) handelt. Wir argumentieren, daß bei einer zufälligen Auswahl eines Ideals mit sehr hoher Wahrscheinlichkeit ein Erzeuger einer sehr großen Untergruppe gefunden wird, selbst wenn die Auswahl nicht unter Gleichverteilung stattfindet.

Aus der Gruppentheorie wissen wir, daß eine endliche, zyklische Gruppe $\mathcal{G}$ genau $\varphi(|\mathcal{G}|)$ Erzeuger hat, wobei φ die Eulersche Totient-Funktion ist. Für φ folgt nach [63, Theorem 15, (3.41)] die Abschätzung

$$\varphi(n) > \frac{n}{2\ln\ln n} \quad , \quad n \geqslant 5\,571156\,552776 \quad . \tag{5.33}$$

Nach den Heuristiken von Cohen und Lenstra (Vermutung 5.1.11) ist die Klassengruppe mit sehr hoher Wahrscheinlichkeit zyklisch oder hat eine sehr große zyklische Untergruppe. Nach den Theoremen von Brauer und Siegel (Theorem 5.1.2) und Littlewood (Theorem 5.1.4) ist die Klassengruppe einer sehr großen Diskriminante selber sehr groß. Es folgt, daß die Anzahl der Elemente, die nur eine kleine Untergruppe der Klassengruppe erzeugen, sehr klein im Vergleich zur Klassenzahl ist. Somit ist die Wahrscheinlichkeit, daß ein zufällig gewähltes Element der Klassengruppe eine sehr kleine Untergruppe erzeugt, selber sehr klein. Darüberhinaus enthält die Klassenzahl einer zufällig gewählten Fundamental-Diskriminante mit hoher Wahrscheinlichkeit einen großen Primfaktor. Somit enthält auch die Ordnung eines zufällig gewähltes Elements mit hoher Wahrscheinlichkeit diesen großen Primfaktor.

Die einzigen Ideale, die sich direkt erzeugen lassen, sind die Primideale. Beliebige reduzierte Ideale werden erzeugt, indem man das reduzierte Potenzprodukt von Primidealen berechnet. Ein zufälliges reduziertes $\mathcal{O}_\Delta$-Ideal läßt sich z. B. mit dem einfachen Algorithmus 3.10 (randomIdeal) auswählen. Bei diesem Verfahren ist jedoch über die Verteilung der zufällig gewählten, reduzierten $\mathcal{O}_\Delta$-Ideale keine Aussage bekannt.

Eine annähernde Gleichverteilung unter den reduzierten $\mathcal{O}_\Delta$-Idealen läßt sich nach Theorem 5.2 aus [48] erzielen, wenn ein Erzeugendensystem $\mathcal{E}_\Delta$ von $\mathrm{Cl}(\Delta)$ bekannt ist, indem man das reduzierte Potenzprodukt aller Ideale aus $\mathcal{E}_\Delta$ mit zufälligen Exponenten $e \in \{1, \ldots, |\Delta|\}$ berechnet. Mit Theorem 4 in [4] folgt unter Annahme der erweiterten Riemann-Vermutung, daß alle $\mathcal{O}_\Delta$-Primideale (p, b) mit $p < 6\ln^2|\Delta|$ die Klassengruppe $\mathrm{Cl}(\Delta)$ erzeugen. Wenn also die erweiterte Riemann-Vermutung gilt, dann kann man effizient und annähernd gleichverteilt zufällige Elemente aus $\mathrm{Cl}(\Delta)$ wählen. Hierzu kann man den nachfolgenden Algorithmus mit $B_p = 6\ln^2|\Delta|$ und $B_e = |\Delta|$ verwenden:

Algorithmus 5.2 randomIdeal2

Eingabe:

1. Eine Diskriminante Δ;
2. Parameter B_p, B_e.

Ausgabe: Ein zufälliges, reduziertes $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}$.

```
1: a ← (1, Δ mod 2)
2: p = 2
3: repeat
4:    p ← primeIdeal(Δ, p)
5:    if p ≠ invalid then
6:       e ← randomInteger(B_e)
7:       a ← reduce(Δ, a · p^e)
8:    end if
9:    p ← nextPrime(p)
10: until p ⩾ B_p
11: return a
```

Mit diesen Parametern ist der Algorithmus in der Praxis nicht sehr effizient. In der Praxis wählt man die Parameter (für Algorithmus 3.10 oder 5.2) viel kleiner und akzeptiert, daß die zufällige Auswahl eines Ideals möglicherweise nicht mehr unter (annähernder) Gleichverteilung erfolgt. (5.33), sowie empirische Daten rechtfertigen diese Vorgehensweise.

Wir haben dazu experimentelle Untersuchungen gemacht, die zeigen, daß eine ganz einfache Wahl von B_p, B_e und B_k in der Praxis ausreichend ist. Im ersten Fall haben wir $\mathfrak{a} = (2, 1)$ als „zufälliges“ Ideal gewählt; so ein Ideal ist genau dann ein $\mathcal{O}_\Delta$-Ideal, wenn $\Delta \equiv 1 \pmod 8$ ist. Wir habe für alle solche Diskriminanten Δ mit Δ prim und $-10^{48} > \Delta \geqslant -10^{48} - 47\,523\,087$ die Klassenzahl mit der Ordnung der Untergruppe verglichen, die von $[(2, 1)]$ erzeugt wird.

Im zweiten Fall haben wir $\mathfrak{a} = (3, 1)$ als „zufälliges“ Ideal gewählt; so ein Ideal ist genau dann ein $\mathcal{O}_\Delta$-Ideal, wenn $\Delta \equiv 1 \pmod{12}$ ist. Wir habe für alle solche Diskriminanten Δ mit Δ prim und $-10^{48} > \Delta \geqslant -10^{48} - 17\,089\,231$ die Klassenzahl mit der Ordnung der Untergruppe verglichen, die von $[(3, 1)]$ erzeugt wird.

Eine Zusammenfassung der Statistiken für beide Fälle ist in Tabelle 5.9 wiedergegeben. Die Statistik zeigt für beide Fälle übereinstimmendes Verhalten. In ausnahmslos allen Fällen erzeugt $\mathfrak{a}$ eine sehr große Untergruppe der Klassengruppe.

Zum Vergleich haben wir dieselbe Untersuchung für kleinere Diskriminanten durchgeführt. Dazu haben wir die Klassenzahlen von primen Diskriminanten $\Delta < -10^{32}$ mit $\Delta \equiv 1 \pmod 8$ bzw. $\Delta \equiv 1 \pmod{12}$ und die entsprechenden Ordnungen der von $[(2, 1)]$ bzw. $[(3, 1)]$ erzeugten Untergruppe berechnet. Die Ergebnisse dieser Untersuchung sind in Tabelle 5.10 zusammengefaßt. Ein Vergleich der Tabellen 5.9 und 5.10 legt die Vermutung nahe, daß ein ähnliches Ergebnis auch für sehr viel größere Diskriminanten gilt. Dieselbe Untersuchung für andere $\mathcal{O}_\Delta$-Ideale $\mathfrak{a}$ als $(2, 1)$ oder $(3, 1)$ führt darüberhinaus zu der Vermutung, daß die Wahl eines beliebigen Primideals (p, b) mit zufällig gewählter Primzahl p mit sehr hoher Wahrscheinlichkeit zu einer sehr großen Untergruppe der Klassengruppe führt.

Tabelle 5.9: Statistik der Ordnungen der Untergruppen, die von $[\mathfrak{a}]$ erzeugt werden ($|\Delta| > 10^{48}$)

$q = \frac{h(\Delta)}{\mathrm{ord}_{\mathrm{Cl}(\Delta)}([\mathfrak{a}])}$	$\mathfrak{a} = (2,1)$		$\mathfrak{a} = (3,1)$	
	Anzahl	Anteil	Anzahl	Anteil
$q = 1$	81093	0.7552	29256	0.7530
$1 < q < 10$	21167	0.1971	7678	0.1976
$10^1 \leqslant q < 10^2$	4621	0.0430	1701	0.0438
$10^2 \leqslant q < 10^3$	445	0.0041	196	0.0050
$10^3 \leqslant q < 10^4$	41	0.0004	19	0.0005
$10^4 \leqslant q < 10^5$	7	0.0000	1	0.0000
Gesamt	107374		38851	

Tabelle 5.10: Statistik der Ordnungen der Untergruppen, die von $[\mathfrak{a}]$ erzeugt werden ($|\Delta| > 10^{32}$)

$q = \frac{h(\Delta)}{\mathrm{ord}_{\mathrm{Cl}(\Delta)}([\mathfrak{a}])}$	$\mathfrak{a} = (2,1)$		$\mathfrak{a} = (3,1)$		$\mathfrak{a} = (1009,\cdot)$		$\mathfrak{a} = (1000003,\cdot)$	
	Anzahl	Anteil	Anzahl	Anteil	Anzahl	Anteil	Anzahl	Anteil
$q = 1$	86599	0.7537	38857	0.7551	43562	0.7535	53013	0.7555
$1 < q < 10$	22718	0.1977	10073	0.1957	11481	0.1986	13784	0.1964
$10^1 \leqslant q < 10^2$	5007	0.0436	2268	0.0441	2475	0.0428	3043	0.0434
$10^2 \leqslant q < 10^3$	522	0.0045	245	0.0048	263	0.0045	288	0.0041
$10^3 \leqslant q < 10^4$	50	0.0004	17	0.0003	27	0.0005	43	0.0006
$10^4 \leqslant q < 10^5$	6	0.0000	2	0.0000	2	0.0000	2	0.0000
Gesamt	114902		51462		57810		70173	

Zur Erzeugung von zufälligen, reduzierten Idealen $\mathfrak{a}$ in Kapitel 3 ist daher der einfache Algorithmus 3.10 mit $B_k = 1$ in der Praxis ausreichend, wobei es verschiedene Möglichkeiten für die Wahl der Parameter B_p und B_e gibt.

Beispiel: Wähle B_p klein, z. B. $B_p = 2^{16}$, wähle B_e groß, z. B. $B_e = 2^{2t}$, wobei t der Sicherheitsparameter ist. Die Suche nach einer kleinen Primzahl kann hier effizient mit einfachgenauer Arithmetik durchgeführt werden.

5.4 Die kryptographische Auswahl der Diskriminante

Wir fassen die Ergebnisse des vorigen Abschnittes zusammen.

1. Um eine Reduktion des IQ-DLP auf einfachere Probleme zu verhindern, muß eine Fundamental-Diskriminante gewählt werden. Das beste praktische Verfahren hierzu besteht in der Auswahl des Diskriminante über die Auswahl der unterschiedlichen Primfaktoren der Diskriminante. Um den geraden Anteil der Klassengruppe möglichst klein

zu halten, muß die Anzahl der Primfaktoren möglichst klein sein. Daher schränken wir die Auswahl der Diskriminante weiter ein auf Diskriminanten der Form $\Delta = -p$ oder $\Delta = -pq$ mit $\left(\frac{p}{q}\right) = -1$.

2. Um einen Angriff mit dem Index-Berechnungs-Algorithmen zu verhindern, muß die Diskriminante möglichst groß sein.

3. Um einen Angriff mit den Quadratwurzel-Algorithmen zu verhindern, muß die Klassengruppe möglichst groß sein. Nach dem Brauer-Siegel-Theorem wächst die Klassenzahl asymptotisch wie die Quadratwurzel des Absolutbetrages der Diskriminante, und unter der Annahme der erweiterten Riemann-Vermutung lassen sich obere und untere Schranken für die Klassenzahl zeigen, die nur unwesentlich vom Mittelwert abweichen. Deswegen muß die Diskriminante möglichst groß sein, um einen Angriff mit den Quadratwurzel-Algorithmen zu verhindern.

4. Um einen Angriff mit dem Pohlig-Hellman-Algorithmus zu verhindern, muß die Klassenzahl einen großen Primfaktor enthalten. Damit die Wahrscheinlichkeit möglichst klein ist, daß bei zufällig gewählter Fundamental-Diskriminante die Klassenzahl keinen großen Primfaktor enthält, muß die Diskriminante möglichst groß sein.

Eine kryptographisch geeignete Diskriminante soll also fundamental und möglichst groß sein. Für die Größenordnung der Diskriminante ist für $\Delta \to -\infty$ der schnellste Algorithmus maßgeblich, mit dem man diskrete Logarithmen in Klassengruppen ausrechnen kann. Asymptotisch sind die Index-Berechnungs-Algorithmen die schnellsten Algorithmen, denn diese Algorithmen haben asymptotisch eine subexponentielle Laufzeit, während die Quadratwurzel-Algorithmen eine exponentielle Laufzeit haben. Wenn man für Algorithmus 5.1 eine feste Erfolgswahrscheinlichkeit zugrundegelegt, dann hat auch dieser Algorithmus eine exponentielle Laufzeit. Wird eine sehr geringe Erfolgswahrscheinlichkeit zugrundegelegt, dann ist der Pohlig-Hellman-Algorithmus in Verbindung mit Algorithmus 5.1 für kleinere Diskriminanten möglicherweise effizienter (wir haben diesen Algorithmus nicht in der Praxis untersucht), als ein Index-Berechnungs-Algorithmus. Aus den Tabellen 5.6, 5.7 und 5.8 sehen wir, daß die Index-Berechnungs-Algorithmen auch für Eingaben mit kryptographisch relevanter Größenordnung die schnellsten Algorithmen sind. Für unsere Zwecke legen wir daher zugrunde, daß die Größe der Diskriminante von der asymptotischen Laufzeit der Index-Berechnungs-Algorithmen bestimmt wird.

Wir diskutieren nun den Algorithmus discSize (siehe auch Algorithmus 5.3), der in Kapitel 3 bei der Auswahl der Bereichsparameter verwendet wird. Der Algorithmus bekommt als Eingabe einen Sicherheitsparameter t, die Ausgabe ist die Bit-Länge t_Δ der Diskriminante. Zwischen t und t_Δ bestehe *unter Berücksichtigung der gesammelten Datenpunkte* (siehe Tabelle 5.4) die Beziehung

$$2^t c_\Delta = L_{2^{t_\Delta}}\left[\tfrac{1}{2}, 1\right] \tag{5.34}$$

für eine Konstante c_Δ. Wir bestimmen c_Δ. Als Maßstab nehmen wir Tabelle 1 aus [47]. Dort wird z. B. mit $t = 80$ ein Berechnungsaufwand von $7.14 \cdot 10^{12}$ MIPS-Jahren assoziiert. Dies entspricht auf einer Maschine mit einer Leistung von 100 MIPS einer Laufzeit von $7.14 \cdot 10^{12} \cdot 60^2 \cdot 24 \cdot 365/100$ Sekunden. Für die Umrechnung von Laufzeit in Sekunden nach Anzahl der Bitoperationen wurde in Abschnitt 5.2.2 für eine 100-MIPS-Maschine die Konstante $c_6 = 1.8 \cdot 10^7\,\mathrm{sec}^{-1}$ ermittelt. $7.14 \cdot 10^{12}$ MIPS Jahre entsprechen somit $1.8 \cdot 7.14 \cdot 10^{17} \cdot 60^2 \cdot 24 \cdot 365 = 2^{80} c_\Delta \approx 1.047677 \cdot 2^{85}$ Bitoperationen. Es folgt, daß $c_\Delta \approx 33.526686$ ist.

Tabelle 5.11: t_Δ für verschiedene Werte von t bei einer hypothetischen Laufzeit asymptotisch proportional zu $L_{|\Delta|}\left[\frac{1}{2}, 1\right]$

t	t_Δ	Erwartete Anz. MIPS-Jahre
70	641	$7.06 \cdot 10^9$
72	671	$2.87 \cdot 10^{10}$
74	701	$1.14 \cdot 10^{11}$
76	732	$4.59 \cdot 10^{11}$
78	763	$1.80 \cdot 10^{12}$
80	795	$7.22 \cdot 10^{12}$
82	828	$2.94 \cdot 10^{13}$
84	861	$1.17 \cdot 10^{14}$
86	895	$4.74 \cdot 10^{14}$
88	929	$1.87 \cdot 10^{15}$
90	964	$7.54 \cdot 10^{15}$

Mit der Substitution $\xi \leftarrow \ln|\Delta|$ und $c' \leftarrow (t \ln 2 + \ln c_\Delta)^2$ ist (5.34) äquivalent zu

$$c' = \xi \ln \xi \ , \tag{5.35}$$

wobei $t_\Delta = \lceil \log_2 |\Delta| \rceil = \lceil \xi / \ln 2 \rceil$ ist. Der folgende Algorithmus berechnet ξ und daraus t_Δ mit Hilfe des Newtonschen Näherungs-Verfahren nach (5.35):

Algorithmus 5.3 discSize

Eingabe: Der Sicherheitsparameter t.
Ausgabe: Die Bit-Länge t_Δ für die Diskriminante.

1: $c' \leftarrow (t \ln 2 + \ln c_\Delta)^2$
2: $\xi' \leftarrow c'$
3: **repeat**
4: $\quad \xi \leftarrow \xi'$
5: $\quad \xi' \leftarrow \xi - \dfrac{\xi \ln \xi - c'}{\ln \xi + 1}$
6: **until** $|\xi - \xi'| < \epsilon$
7: $t_\Delta \leftarrow \lceil \xi / \ln 2 \rceil$
8: **return** t_Δ

In Tabelle 5.11 haben wir einige Werte für t_Δ aufgelistet; die Berechnungen erfolgten mit $\epsilon = 1/2$. Ein Vergleich mit Tabelle 5.6 oder Tabelle 1 aus [47] zeigt, daß die Wahl von $t = 72$ der Wahl eines 1024-Bit RSA-Modulus entspricht.

Wir haben hier für das IQ-MPQS eine Laufzeit angenommen, die asymptotisch proportional zu $L_{|\Delta|}\left[\frac{1}{2}, 1\right]$ ist. Wir haben dies durch die gemessenen Laufzeiten für das IQ-MPQS begründet (Tabelle 5.4). Wir wollen abschließend zeigen, wie sich die geforderte Größe der Diskriminante verhält, wenn man eine Laufzeit für das IQ-MPQS zugrunde legt, die asymptotisch proportional zu $L_{|\Delta|}\left[\frac{1}{2}, \frac{3}{4}\sqrt{2}\right]$ ist ($\frac{3}{4}\sqrt{2} = \sqrt{9/8} \approx 1.06066$). Diese Laufzeit wurde

Tabelle 5.12: t_Δ für verschiedene Werte von t bei einer hypothetischen Laufzeit asymptotisch proportional zu $L_{|\Delta|}\left[\frac{1}{2}, \frac{3}{4}\sqrt{2}\right]$

t	t'_Δ	Erwartete Anz. MIPS-Jahre
70	613	$7.29 \cdot 10^9$
72	640	$2.86 \cdot 10^{10}$
74	668	$1.15 \cdot 10^{11}$
76	696	$4.50 \cdot 10^{11}$
78	725	$1.81 \cdot 10^{12}$
80	755	$7.41 \cdot 10^{12}$
82	785	$2.96 \cdot 10^{13}$
84	815	$1.16 \cdot 10^{14}$
86	846	$4.63 \cdot 10^{14}$
88	878	$1.89 \cdot 10^{15}$
90	910	$7.56 \cdot 10^{15}$

für einen hypothetischen Index-Berechnungs-Algorithmus bewiesen [80]. Wir legen hierfür als Datenpunkt die durchschnittliche gemessene Laufzeit für Δ in der Größenordnung 2^{220} aus Tabelle 5.4 zugrunde. Daraus ergibt sich

$$c'_6 = \frac{L_{|\Delta|}\left[\frac{1}{2}, \frac{3}{4}\sqrt{2}\right]}{5.71 \cdot 10^4 \,\mathrm{sec}} = 9.94 \cdot 10^7 \,\mathrm{sec}^{-1} \quad .$$

Wir berechnen nun c'_Δ, so daß

$$2^t c'_\Delta = L_{2^{t'_\Delta}}\left[\tfrac{1}{2}, \tfrac{3}{4}\sqrt{2}\right] \tag{5.36}$$

gelte, wobei wir wieder mit $t = 80$ einen Berechnungsaufwand von $7.14 \cdot 10^{12}$ MIPS-Jahren assoziieren. Dann ergibt eine Rechnung wie für c_Δ, daß $c'_\Delta \approx 185.096967$ ist.

Mit der Substitution $\xi \leftarrow \ln|\Delta|$ und $c' \leftarrow \frac{8}{9}(t \ln 2 + \ln c'_\Delta)^2$ ist (5.36) äquivalent zu

$$c' = \xi \ln \xi \quad , \tag{5.37}$$

wobei $t'_\Delta = \lceil \log_2 |\Delta| \rceil = \lceil \xi / \ln 2 \rceil$ ist. In Tabelle 5.12 haben wir einige Werte für t_Δ aufgelistet, die mit Algorithmus 5.3 nach den entprechenden Modifikationen berechnet wurden.

Vergleicht man die Tabellen 5.11 und 5.12 miteinander, dann sieht man, daß für $t \in [70, 90]$ etwa $t'_\Delta \approx 0.95 \cdot t_\Delta$ ist. Unterstellt man eine quadratische Laufzeit für Arithmetik in Klassengruppen (siehe Abschnitt 7.1.4), dann ergibt sich hieraus eine Effizienz-Steigerung um etwa 10 Prozent, wenn man t'_Δ anstelle von t_Δ für die Größe der Diskriminante zugrunde legt. Ein weiterer Vergleich zeigt, daß $t_\Delta(t-3) \approx t'_\Delta(t)$ für $t \in [70, 90]$ ist. Das bedeutet folgendes: Angenommen, es gelte tatsächlich die Asymptotik $L_{|\Delta|}\left[\frac{1}{2}, 1\right]$. Berechnet man die Größe der Diskriminante in Abhängigkeit von t gemäß der Asymptotik $L_{|\Delta|}\left[\frac{1}{2}, \frac{3}{4}\sqrt{2}\right]$, dann ist der tatsächliche Aufwand zur Berechnung von diskreten Logarithmen in Klassengruppen in Wirklichkeit etwa nur 2^{-3}-mal so groß, wie der nach der Asymptotik $L_{|\Delta|}\left[\frac{1}{2}, \frac{3}{4}\sqrt{2}\right]$ erwartete Aufwand. Ob dieses Sicherheits-Risiko zugunsten des Effizienzgewinns vernachlässigbar ist, oder nicht, liegt letztendlich im Ermessen des Benutzers.

Kapitel 6

Sicherheit der IQ-Kryptoverfahren

In diesem Kapitel beschreiben wir die Sicherheit der kryptographischen Verfahren aus Kapitel 4. Die Sicherheit eines Public-Key-Verfahrens beruht immer auf der Schwierigkeit, ein Berechnungsproblem zu lösen. Die Sicherheit des RSA-Verschlüsselungs- und des RSA-Signaturverfahrens z. B. beruht auf der Schwierigkeit, große Zahlen zu faktorisieren. Wir zeigen, daß unsere IQ-Kryptoverfahren sicher sind unter der Annahme, daß die Berechnung diskreter Logarithmen oder Wurzeln in Klassengruppen nicht effizient möglich ist. Zur Berechnung von Wurzeln in Klassengruppen ist kein besseres Verfahren bekannt, als zuvor einen diskreten Logarithmus zu berechnen. In Kapitel 5 haben wir gezeigt, unter welchen Bedingungen die Berechnung von diskreten Logarithmen in Klassengruppen mit den besten bekannten Algorithmen nicht effizient möglich ist.

In Abschnitt 6.1 definieren wir die Berechnungsprobleme, auf deren Schwierigkeit die Sicherheit der IQ-Kryptoverfahren beruht. Was hierbei „Sicherheit“ bedeutet, wird in Abschnitt 6.2 näher ausgeführt werden.

In Abschnitt 6.3 gehen wir detailliert auf die Sicherheit des RDSA-Protokolls ein und zeigen, daß die Berechnung von Wurzeln in endlichen Gruppen mit unbekannter Ordnung und die existenzielle Fälschung einer RDSA-Signatur komplexitätstheoretisch äquivalent sind. Dies ist u. a. Inhalt unserer Arbeit in [8].

In Abschnitt 6.4 gehen wir nur kurz ein auf die Sicherheit der hier verwendeten Variante des DSA-Protokolls für endliche Gruppen mit unbekannter Ordnung. Eine umfassende Sicherheitsanalyse zu einer fast identischen Variante des Schnorr-Signaturverfahren steht in [59]; wir zitieren hier nur das entscheidende Ergebnis dieser Analyse.

In Abschnitt 6.5 gehen wir wieder etwas detaillierter auf die Sicherheit des GQ-Signaturverfahrens [30] ein, da die Beweisführung zur Sicherheit teilweise der für RDSA ähnlich ist.

In Abschnitt 6.6 gehen wir wiederum nur kurz auf die Sicherheit von DHIES ein, da die Sicherheit von DHIES umfassend in [1] beleuchtet wird. Wir zitieren auch hier nur das entscheidende Ergebnis der Analyse.

In Abschnitt 6.7 gehen wir schließlich darauf ein, warum wir bei den IQ-Kryptoverfahren relativ kurze Exponenten verwenden können.

6.1 Die zugrundeliegenden Berechnungsprobleme

In diesem Abschnitt definieren wir die Berechnungsprobleme, die im Zusammenhang mit Klassengruppen kryptographisch interessant sind, weil diese Berechnungsprobleme bei Klassengruppen mutmaßlich sehr schwierig sind. Wir zeigen, wie diese Berechnungsprobleme miteinander in Beziehung stehen, und wir werden den Bezug zum Faktorisierungsproblem zeigen.

6.1.1 Das Diskreter-Logarithmus-Problem

Definition 6.1.1 (Discrete Logarithm Problem, DLP)
Sei $\mathcal{G}$ eine endliche, Abelsche Gruppe. Seien $\alpha, \gamma \in \mathcal{G}$. Entscheide ob $\alpha \in \langle\gamma\rangle$ ist. Falls $\alpha \in \langle\gamma\rangle$ ist, finde die kleinste positive Zahl x, so daß $\gamma^x = \alpha$ ist. x heißt diskreter Logarithmus *von α zur Basis γ und wird als $\log_\gamma \alpha$ geschrieben.*

Falls $\mathcal{G}$ die Klassengruppe eines imaginär-quadratischen Körpers ist, so sprechen wir vom IQ-DLP.

6.1.2 Das Ordnungsproblem

Definition 6.1.2 (Order Problem, OP)
Sei $\mathcal{G}$ eine endliche Abelsche Gruppe. Sei $\gamma \in \mathcal{G}$. Berechne die kleinste positive Zahl x, so daß $\gamma^x = 1_\mathcal{G}$ ist. x heißt Ordnung *von γ in $\mathcal{G}$ und wird als $\mathrm{ord}_\mathcal{G}\,\gamma$ geschrieben.*

Falls $\mathcal{G}$ die Klassengruppe eines imaginär-quadratischen Körpers ist, so sprechen wir vom IQ-OP.

6.1.3 Das Wurzelproblem

Definition 6.1.3 (Root Problem, RP)
Sei $\mathcal{G}$ eine endliche Abelsche Gruppe. Sei $\alpha \in \mathcal{G}$ und p eine Primzahl, die kein Teiler von $|\mathcal{G}|$ ist. Berechne das Element $\varrho \in \mathcal{G}$, so daß $\varrho^p = \alpha$ ist. ϱ heißt p-te Wurzel *von α.*

Falls $\mathcal{G}$ die Klassengruppe eines imaginär-quadratischen Körpers ist, so sprechen wir vom IQ-RP.

6.1.4 Das Diffie-Hellman-Problem

Definition 6.1.4 (Diffie-Hellman Problem, DHP)
Sei $\mathcal{G}$ eine endliche Abelsche Gruppe. Sei $\gamma, \gamma^a, \gamma^b \in \mathcal{G}$ für $a, b \in \mathbb{Z}$. Berechne γ^{ab}.

Falls $\mathcal{G}$ die Klassengruppe eines imaginär-quadratischen Körpers ist, so sprechen wir vom IQ-DHP.

6.1.5 Reduktionen

Wir zeigen nun die Beziehungen zwischen dem IQ-DLP, dem IQ-OP und dem IQ-RP auf. Wir verwenden hierzu sog. Polynomzeit-Reduktionen. Wir setzen das Konzept der Turing-Maschine als bekannt voraus (andenfalls siehe z. B. [5]). Eine probabilistische Turing-Maschine modellieren wir als deterministische Turing-Maschine mit einem zusätzlichen Eingabe-Band, das mit einer endlos langen Zufallsfolge von 0 und 1 beschriftet ist. Dieses Band darf nur gelesen werden, und zwar nur in einer Richtung; wir bezeichnen dieses zusätzliche Band als Zufallsband. Die Turing-Maschine zieht zufällig ein Bit, indem sie das nächste Symbol von dem Zufallsband liest.

Wir werden das Konzept der Polynomzeit-Reduktion nun kurz und informal erklären. Polynomzeit-Reduktionen wurden in der Komplexitätstheorie ursprünglich definiert für Entscheidungsprobleme der folgenden Art: Sei Σ ein Alphabet, $L \subseteq \Sigma^*$ eine Sprache und $x \in \Sigma^*$ ein Wort, ist $x \in L$? Eine in der Komplexitätstheorie häufig verwendete Reduktion ist die sog. *Many-One-Reduktion*. Eine Sprache B heißt auf A many-one-reduzierbar genau dann, wenn es eine deterministisch in Polynomzeit berechenbare Funktion $R : \Sigma^* \to \Sigma^*$ gibt, so daß $x \in B$ genau dann, wenn $R(x) \in A$ für alle $x \in \Sigma^*$, und man schreibt $B \leqslant_m A$.

Da wir es hier nicht mit Entscheidungsproblemen sondern mit Berechnungsproblemen zu tun haben, benötigen wir noch eine geeignete Rück-Reduktion, die eine Lösung für das berechnete Problem in eine Lösung für das ursprüngliche Problem transformiert. Wir geben hier nur informal eine Idee dazu wieder. Seien A und B Berechnungsprobleme, und seien $\mathcal{X}_A$ und $\mathcal{X}_B$ die Menge aller Exemplare von A bzw. B, und seien $\mathcal{L}_A$ und $\mathcal{L}_B$ die Menge aller Lösungen für A bzw. B. B heißt *Polynomzeit-reduzierbar* auf A, wenn es deterministisch in Polynomzeit berechenbare Funktionen $R : \mathcal{X}_B \to \mathcal{X}_A$ und $R' : \mathcal{L}_A \times \mathcal{X}_A \times \mathcal{X}_B \to \mathcal{L}_B$ gibt, so daß L_A eine Lösung für $R(X_B)$ genau dann ist, wenn $R'(L_A, X_A, X_B)$ eine Lösung für X_B ist. Wir schreiben dafür $B \leqslant_m A$. Falls $B \leqslant_m A$ und $A \leqslant_m B$ ist, dann heißen A und B *Polynomzeit-äquivalent*, und wir schreiben $A =_m B$.

Die Information, die hinter $B \leqslant_m A$ steckt, kann auch folgendermaßen formuliert werden: Wenn es einen Polynomzeit-Algorithmus für A gibt, dann gibt es auch einen Polynomzeit-Algorithmus für B. Wenn es dagegen keinen Polynomzeit-Algorithmus für B gibt, dann gibt es auch keinen Polynomzeit-Algorithmus für A. Wenn A also ein leichtes Problem ist, dann auch B; falls umgekehrt B aber ein schwieriges Problem ist, dann auch A. Dies ist ein zentraler Bestandteil der Sicherheitsbeweise in diesem Kapitel.

Wir zeigen nun einige Reduktionen. Offensichtlich ist die Lösung zu dem DLP für $(\mathcal{G}, 1_\mathcal{G}, \gamma)$ die Lösung zu dem OP für $(\mathcal{G}, \gamma)$, daher gilt:

Proposition 6.1.5
$OP(\mathcal{G}, \gamma)$ ist reduzierbar auf $DLP(\mathcal{G}, 1_\mathcal{G}, \gamma)$.

Die Lösung zu dem DHP für $(\mathcal{G}, \gamma, \gamma^a, \gamma^b)$ ergibt sich aus der Berechnung von a oder b, daher gilt:

Proposition 6.1.6
$DHP(\mathcal{G}, \gamma, \gamma^a, \gamma^b)$ ist reduzierbar auf $DLP(\mathcal{G}, \gamma^a, \gamma)$.

Proposition 6.1.7
Sei p eine Primzahl, die kein Teiler von $|\mathcal{G}|$ ist. Dann ist $RP(\mathcal{G}, \alpha, p)$ reduzierbar auf $OP(\mathcal{G}, \alpha)$.

Beweis: Sei $\mathcal{G}$ eine endliche, Abelsche Gruppe, und sei $\alpha \in \mathcal{G}$. Berechne $\mathrm{ord}_{\mathcal{G}}\,\alpha$ via Unterprogramm für das OP. Da $p \nmid \mathrm{ord}_{\mathcal{G}}\,\alpha$, gibt es genau eine p-te Wurzel von α. Berechne dazu x mit dem erweiterten Euklidischen Verfahren (xgcd, siehe Algorithmus 2.9), so daß $px \equiv 1 \pmod{\mathrm{ord}_{\mathcal{G}}\,\alpha}$ ist, und berechne $\varrho = \alpha^x$, dann ist $\alpha = \varrho^p$. □

Wir wissen nun, daß IQ-RP $\leqslant_m$ IQ-OP $\leqslant_m$ IQ-DLP und IQ-DHP $\leqslant_m$ IQ-DLP ist, aber es ist unbekannt, ob IQ-RP $=_m$ IQ-OP oder IQ-RP $<_m$ IQ-OP, oder ob IQ-OP $=_m$ IQ-DLP oder IQ-OP $<_m$ IQ-DLP ist, oder in welcher Beziehung das IQ-DHP zum IQ-OP oder IQ-RP steht. Dennoch erscheinen alle vier Probleme in der Praxis gleich schwierig, denn in der Praxis wird genau von den Reduktionen Gebrauch gemacht. Es ist kein besseres Verfahren bekannt, um das IQ-OP zu lösen, als vorher ein IQ-DLP zu lösen; ebenso ist kein besseres Verfahren bekannt, um das IQ-RP zu lösen, als vorher ein IQ-OP zu lösen; schließlich ist kein besseres Verfahren bekannt, um das IQ-DHP zu lösen, als vorher ein IQ-DLP zu lösen. Verfahren zur Lösung des IQ-DLP werden in 5.2 vorgestellt. Das beste bekannte Verfahren zur Lösung des IQ-DLP aber ist für große Diskriminanten nicht effizient, und somit sind die besten bekannten Verfahren zur Lösung des IQ-OP, des IQ-RP und des IQ-DHP für große Diskriminanten ebenfalls ineffizient.

Daneben stellen wir fest: Es ist kein effizientes Verfahren zur Berechnung der Klassenzahl, eines Teilers der Klassenzahl (insbesondere glatte Anteile), oder eines Vielfachen der Klassenzahl bekannt.

Wir setzen nun das IQ-OP in Beziehung zum Faktorisierungsproblem:

Proposition 6.1.8
Das Faktorisierungsproblem ist reduzierbar auf das IQ-OP.

Beweis: Sei n eine positive, ganze, ungerade, zerlegbare Zahl. Falls $n \equiv 1 \pmod 4$, dann setze $\Delta = -4n$, andernfalls setze $\Delta = -n$. Da Δ nach Voraussetzung keine Primzahl ist, ist $h(\Delta)$ gerade. Wähle ein zufälliges, reduziertes $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}$, und bestimme via Unterprogramm für das IQ-OP $a = |\langle[\mathfrak{a}]\rangle|$. Falls a ungerade ist, wähle solange ein anderes zufälliges, reduziertes $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}$, bis $a = |\langle[\mathfrak{a}]\rangle|$ gerade ist. Wenn $\mathfrak{a}$ gleichverteilt aus der Menge der reduzierten $\mathcal{O}_\Delta$-Ideale gewählt wird (z. B. Algorithmus 5.2), dann ist $\Pr[a \text{ ist gerade}] \geqslant \frac{1}{2}$. Wenn a gerade ist, dann ist $[\mathfrak{a}]^{(a/2)}$ ambig. □

Das bedeutet, daß das IQ-OP mindestens genauso schwierig ist, wie das Faktorisierungsproblem. Das Faktorisierungsproblem ist also eine komplexitäts-theoretische untere Schranke für das IQ-DLP und das IQ-OP. Nähere Details über Faktorisierungs-Algorithmen, die auf dieser Reduktion beruhen, findet man in [66] und [48].

6.2 Das Sicherheits-Modell

In diesem Abschnitt definieren wir, was wir unter „sicher“ verstehen, und welche Modelle wir dafür verwenden. Eine fundierte Übersicht über Sicherheit von kryptographischen Protokollen findet man in [29] oder [28]. Wir unterscheiden zwischen Signaturen und Verschlüsselungen und gehen von dem folgenden Szenario aus: **A** ist ein legitimer Signierer bzw. Empfänger von verschlüsselten Texten, und **E** ist ein Angreifer. **E** hat Zugriff auf **A**s öffentlichen Schlüssel und Bereichsparameter, nicht aber auf **A**s privaten Schlüssel.

Wir modellieren **A** und **E** in diesem Kapitel als probabilistische Turing-Maschinen. Das bedeutet, daß **E** mit einem Angriff auf ein Kryptoverfahren eine gewisse Erfolgswahrscheinlichkeit hat. Wir bezeichnen ein Kryptoverfahren als sicher, wenn die Erfolgswahrscheinlichkeit für einen Angriff *vernachlässigbar* ist.

Definition 6.2.1
Eine Funktion $f : \mathbb{R} \to \mathbb{R}$ *heißt* vernachlässigbar, *wenn für jedes* $c > 0$ *gilt: Es existiert ein* x_c, *so daß* $x^{-c} > |f(x)|$ *für alle* $x \geqslant x_c$.

Die Sicherheitsbeweise in diesem Kapitel sind Reduktionsbeweise, d. h. wir folgen z. B. im Fall der Signaturen der Strategie zu zeigen, daß das IQ-DLP oder das IQ-RP auf die existenzielle Fälschung einer Signatur reduzierbar ist. Da das IQ-DLP oder das IQ-RP mutmaßlich schwierige Probleme sind, folgt daraus, daß auch die existenzielle Fälschung einer Signatur ein mutmaßlich schwieriges Problem ist.

6.2.1 Zufallsorakel

Für die Beweise zur Sicherheit von Signaturen verwenden wir eine Konstruktion, die als Zufallsorakel bezeichnet wird [6] (siehe aber auch [17]). Wir erklären das Konzept informal. Mit einem Zufallsorakel wird eine kryptographische Hash-Funktion modelliert. Das Konzept von Orakeln stammt aus der Komplexitätstheorie. Seien n und m fest gewählte, ganze, positive Zahlen, und sei $f : \{0,1\}^m \to \{0,1\}^n$ eine Funktion, dann ist in unserem Kontext ein *Orakel für* f ein Mechanismus, der zu einer Eingabe $x \in \{0,1\}^m$ den Wert von $f(x)$ berechnet; alles andere über f bleibt verborgen (daher die Bezeichnung „Orakel“). Wird f unter allen möglichen Abbildungen von $\{0,1\}^m$ nach $\{0,1\}^n$ zufällig ausgewählt, dann bezeichnet man das Orakel für f als Zufallsorakel. Die Idee besteht darin, die Hash-Funktion $h : \{0,1\}^m \to \{0,1\}^n$ für den Beweis durch ein Zufallsorakel zu ersetzen, und in der konkreten Anwendung mit einer zufällig gewählten Hash-Funktion zu instanziieren.

Das sog. Forking-Lemma (Lemma 6.2.2) ist ein wichtiger Bestandteil in den Beweisen zur Sicherheit von Signaturen mit Zufallsorakeln. Für unsere Zwecke ist die einfache Version aus [57] ausreichend. Dort ist die Signatur zu einem Text M ein Quadrupel $(M, \sigma_1, h, \sigma_2)$, wobei σ_1 ein zufällig gewählter Wert aus einer großen Menge ist, h hängt nur von M und σ_1 ab, und σ_2 hängt nur von M, σ_1 und h ab. In bestimmten Signaturverfahren kann σ_1 oder h weggelassen werden. Z. B. ist bei RDSA, wie in Abschnitt 6.3 beschrieben, $\sigma_1 = \varrho$ und $\sigma_2 = (s, \lambda)$, während h weggelassen wurde.

Wir zitieren nun das Forking-Lemma aus [57]:

Lemma 6.2.2 (Forking-Lemma)
Sei $\mathcal{M}$ *eine probabilistische Polynomzeit-Turing-Maschine, mit einem Zufallsorakel, einem Zufallsband, und nur den öffentlichen Daten (öffentlicher Schlüssel, Bereichsparameter) als Eingabe. Wenn* $\mathcal{M}$ *mit nicht vernachlässigbarerWahrscheinlichkeit eine gültige Signatur* $(M, \sigma_1, h, \sigma_2)$ *erzeugen kann, dann kann* $\mathcal{M}$ *mit demselben Zufallsband und einem anderen Zufallsorakel eine weitere gültige Signatur* $(M, \sigma_1, h', \sigma_2')$ *erzeugen, so daß* $h \neq h'$ *ist.*

6.2.2 Signaturen

Wir definieren in diesem Abschnitt, wann wir eine Signatur als sicher verstehen. Eine Signatur von **A** für einen Text M bezeichnen wir mit $\mathcal{S}_{\mathbf{A},M}$. Ein Angreifer **E** kann die folgenden drei Angriffs-Ziele gegen ein Signaturverfahren verfolgen:

1. Existenzielle Fälschung: **E** erzeugt eine gültige Signatur $\mathcal{S}_{\mathbf{A},M}$ für einen Text M, auf den **E** keinen Einfluß hat.
2. Universelle Fälschung: **E** erzeugt eine gültige Signatur $\mathcal{S}_{\mathbf{A},M}$ für einen beliebigen Text M seiner Wahl.
3. Schlüsselgewinn: **E** berechnet **A**s privaten Schlüssel.

Das am schwierigsten zu erreichende Ziel ist ein Schlüsselgewinn, das am leichtesten zu erreichende Ziel ist eine existenzielle Fälschung. Ein Signaturverfahren, welches einer existenziellen Fälschung widersteht, widersteht erst recht einer universellen Fälschung und einem Schlüsselgewinn.

Bei einem Angriff unterscheiden wir außerdem nach den Möglichkeiten eines Angreifers. Im einzelnen unterscheiden wir die folgenden Fälle:

1. Angriff nur mit dem Schlüsselmaterial (Kein-Text-Angriff, No-Message-Attack).
2. Angriff mit bekannten Text/Signatur-Paaren (Bekannter-Text-Angriff, Known-Message-Attack).
3. Angriff mit ausgewählten Texten (Gewählter-Text-Angriff, Chosen-Message-Attack). Hier wird noch unterschieden, wann der Angreifer die Auswahl der der Texte treffen darf, nämlich
   (a) a priori, also vor dem eigentlichen Angriff (Indifferent-Gewählter-Text-Angriff, Indifferent-Chosen-Message-Attack);
   (b) während des Angriffs, mit Anpassung der Auswahl an bisher gewonnene Ergebnisse (Angepaßt-Gewählter-Text-Angriff, Adaptively-Chosen-Message-Attack).

Der stärkste Angriff ist ein Angriff mit angepaßt gewähltem Text, ein Signaturverfahren, das einem solchen Angriff widersteht, widersteht auch einem Angriff mit indifferent gewähltem, bekanntem oder keinem Text.

Definition 6.2.3
*Ein Signaturverfahren heißt sicher, wenn die Wahrscheinlichkeit vernachlässigbar klein ist, daß **E** durch einen Angriff mit angepaßt gewählten Texten die existenzielle Fälschung einer Signatur $\mathcal{S}_{\mathbf{A},M}$ in polynomiell beschränkter Zeit erzeugen kann.*

6.2.3 Verschlüsselungen

Wir definieren in diesem Abschnitt, wann wir eine Verschlüsselung als sicher verstehen. Einen Schlüsseltext, aus dem nur **A** den Klartext M rekonstruieren kann, bezeichnen wir mit $\mathcal{C}_{\mathbf{A},M}$. Ein Angreifer **E** kann die folgenden drei Angriffs-Ziele gegen ein Verschlüsselungsverfahren verfolgen:

1. Unterscheidung: **E** entscheidet zu zwei Klartexten M_1 und M_2 und der Verschlüsselungen $\mathcal{C} = \mathcal{C}_{\mathbf{A},M_?}$ zu einem dieser Klartexte, ob $\mathcal{C} = \mathcal{C}_{\mathbf{A},M_1}$ oder $\mathcal{C} = \mathcal{C}_{\mathbf{A},M_2}$ ist.

2. Entschlüsselung: **E** entschlüsselt $\mathcal{C}_{\mathbf{A},M}$.

3. Schlüsselgewinn: **E** berechnet **A**s privaten Schlüssel.

Das am schwierigsten zu erreichende Ziel ist ein Schlüsselgewinn, das am leichtesten zu erreichende Ziel ist Unterscheidung. Ein Verschlüsselungsverfahren, welches einer Unterscheidung widersteht, widersteht erst recht einer Entschlüsselung und einem Schlüsselgewinn.

Bei einem Angriff unterscheiden wir außerdem nach den Möglichkeiten eines Angreifers. Im einzelnen unterscheiden wir die folgenden Fälle:

1. Angriff nur mit dem Schlüsselmaterial (Kein-Text-Angriff).

2. Angriff mit bekannten Schlüsseltexten (Bekannter-Schlüsseltext-Angriff, Known-Ciphertext-Attack).

3. Angriff mit ausgewählten Schlüsseltexten (Gewählter-Schlüsseltext-Angriff, Chosen-Ciphertext-Attack). Hier wird noch unterschieden, wann der Angreifer die Auswahl der der Texte treffen darf, nämlich

   (a) a priori, also vor dem eigentlichen Angriff (Indifferent-Gewählter-Schlüsseltext-Angriff, Indifferently-Chosen-Ciphertext-Attack);

   (b) während des Angriffs, mit Anpassung der Auswahl an bisher gewonnene Ergebnisse (Angepaßt-Gewählter-Schlüsseltext-Angriff, Adaptively-Chosen-Ciphertext-Attack).

Der stärkste Angriff ist ein Angriff mit angepaßt gewähltem Schlüsseltext, ein Verschlüsselungsverfahren, das einem solchen Angriff widersteht, widersteht auch einem Angriff mit indifferent gewähltem oder bekanntem Schlüsseltext oder keinem Text.

Definition 6.2.4
Ein Verschlüsselungsverfahren heißt sicher, wenn die Wahrscheinlichkeit vernachlässigbar klein ist, daß ***E*** *durch einen Angriff mit angepaßt gewählten Schlüsseltexten zwei Klartexte M_1 und M_2 in polynomiell beschränkter Zeit findet, die* ***E*** *in polynomiell beschränkter Zeit bezüglich $\mathcal{C}$ wie oben beschrieben unterscheiden kann.*

6.3 IQ-RDSA

Wir geben in diesem Abschnitt einen Sicherheits-Beweis für RDSA. Die Grundlage hierfür ist unsere Arbeit in [8]. Wir zeigen, daß RDSA im Zufallsorakel-Modell sicher gegen einen Angriff mit angepaßt ausgewählten Texten ist. Wir motivieren zunächst die Einführung von RDSA für endliche Gruppen mit unbekannter Ordnung. Danach zeigen wir, daß bei in solchen Gruppen die Berechnung von Wurzeln und die existenzielle Fälschung von RDSA-Signaturen mit Angepaßt-Gewählter-Text-Angriffen komplexitätstheoretisch äquivalent sind.

Für die nachfolgende Diskussion verwenden wir eine schematische Darstellung von RDSA, bei der wir die technischen Details auslassen, die dafür nicht von Bedeutung sind. Wir abstrahieren darüberhinaus von Klassengruppen und argumentieren allgemein für beliebige Abelsche, endliche Gruppen $\mathcal{G}$ mit neutralem Element $1_{\mathcal{G}}$, und unterstellen, daß es für $\mathcal{G}$ die folgenden Algorithmen mit polynomiell beschränkter Laufzeit gibt:

- Wähle zufällig ein Element aus $\mathcal{G}$ unter Gleichverteilung aus.
- Gegeben $\alpha, \beta \in \mathcal{G}$, entscheide ob $\alpha = \beta$.
- Gegeben $\alpha, \beta \in \mathcal{G}$, berechne $\alpha \cdot \beta$.
- Gegeben $\alpha \in \mathcal{G}$, berechne α^{-1}.

Außerdem setzen wir voraus, daß weder $|\mathcal{G}|$ noch ein großer Teiler davon effizient berechnet werden kann; wir unterstellen jedoch, daß Abschätzungen $\underline{B}_{\mathcal{G}}$ und $\overline{B}_{\mathcal{G}}$ für $|\mathcal{G}|$ bekannt sind, so daß

$$\underline{B}_{\mathcal{G}} \leqslant |\mathcal{G}| \leqslant \overline{B}_{\mathcal{G}} \tag{6.1}$$

und $1/(\overline{B}_{\mathcal{G}} - \underline{B}_{\mathcal{G}})$ sowie $B_{\mathcal{G}}/|\mathcal{G}|$ vernachlässigbar ist mit $B_{\mathcal{G}} = \overline{B}_{\mathcal{G}}/\underline{B}_{\mathcal{G}}$.

Wir werden unter diesen Voraussetzungen zeigen:

Theorem 6.3.1
Sei $\mathcal{G}$ eine endliche, Abelsche Gruppe. Unter der Annahme, daß es keinen Algorithmus gibt, der das Wurzelproblem in $\mathcal{G}$ mit nicht vernachlässigbarer Wahrscheinlichkeit in polynomiell beschränkter Zeit berechnet, ist RDSA in $\mathcal{G}$ im Zufallsorakel-Modell sicher.

6.3.1 Motivation

Nach Voraussetzung kann das Ordnungsproblem in $\mathcal{G}$ nicht effizient gelöst werden. Damit können Signaturverfahren vom ElGamal-Typ, z. B. DSA, nicht ohne weiteres auf $\mathcal{G}$ übertragen werden, da für Signaturverfahren vom ElGamal-Typ die Kenntnis von $|\mathcal{G}|$ vorausgesetzt wird.

Wir rekapitulieren, wie „traditionelles“ DSA in $\mathcal{G} = \mathbb{Z}/p\mathbb{Z}^*$ funktioniert:

Vorbereitung: **A** macht die folgenden Schritte:
Wähle zufällig eine 160-Bit-Primzahl q, und wähle eine 1024-Bit-Primzahl p, so daß $q \mid p-1$;
wähle $\gamma_0 \in \mathbb{Z}/p\mathbb{Z}$ und berechne

$$\gamma = \gamma_0^{(p-1)/q} \bmod p \ , \tag{6.2}$$

und falls $\gamma = 1$, dann wähle ein anderes γ_0 ($\mathcal{G}$ ist die von γ erzeugte Untergruppe von $\mathbb{Z}/p\mathbb{Z}^*$ der Ordnung q);
wähle zufällig eine ganze Zahl a mit $0 < a < q$ und berechne $\alpha = \gamma^a$.

As öffentlicher Schlüssel ist (p, q, γ, α), der private Schlüssel ist a.

Signatur: **A** macht die folgenden Schritte:
Wähle zufällig eine ganze Zahl k mit $0 < k < q$ und berechne $\varrho = \gamma^k \bmod p$;
berechne $r = \varrho \bmod q$ und

$$s = k^{-1}(h(M) + ar) \bmod q \ . \tag{6.3}$$

Die Signatur $\mathcal{S}^{\mathrm{DSA}}_{\mathbf{A},M}$ ist das Paar (s, r).

Verifikation: **B** macht die folgenden Schritte:
Prüfe, ob $0 < s < q$;
berechne $w = s^{-1} \bmod q$, $u_1 = wh(M) \bmod q$ und $u_2 = wr \bmod q$;
berechne $v = (\gamma^{u_1}\alpha^{u_2} \bmod p) \bmod q$.

B akzeptiert $\mathcal{S}^{\mathrm{DSA}}_{\mathbf{A},M}$, wenn $v = r$.

Ohne die Kenntnis von $|\mathcal{G}|$ kann weder (6.2) noch (6.3) berechnet werden. Mit (6.2) wird sichergestellt, daß γ ein Erzeuger von $\mathcal{G}$ ist. Falls $\mathcal{G}$ zyklisch ist, dann ist die Wahrscheinlichkeit sehr hoch, daß ein zufälliges $\gamma \in \mathcal{G}$ ein Erzeuger von $\mathcal{G}$ oder einer sehr großen Untergruppe davon ist. In diesem Fall ist die zufällige Auswahl von γ ausreichend. Es bleibt zu klären, wie mit (6.3) verfahren werden soll. Wir betrachten zwei Möglichkeiten:

1. Ersetze q durch eine beliebige Primzahl q', die nicht notwendigerweise ein Teiler von $\mathcal{G}$ ist. Die Motivation für diese Vorgehensweise besteht darin, daß $s = x \bmod q$ gleichbedeutend ist mit $s = x - \ell q$ für eine ganze Zahl ℓ. Da i. allg. $q' \neq |\langle\gamma\rangle|$ sein wird, ist dann auch $s \not\equiv x \pmod{|\langle\gamma\rangle|}$, und daher braucht man zur Verifikation den zusätzlichen Korrektur-Faktor γ^{ℓ} als Teil der Signatur. Dieser Ansatz, dem wir als nächstes nachgehen werden, ist Gegenstand unserer Arbeit in [8].

   Der Preis, den man bei diesem Ansatz bezahlt, besteht darin, daß man bei der Erzeugung einer Signatur zwei Exponentiationen statt einer durchführen muß, und daß die Signatur eine ganze Zahl und zwei Gruppenelemente enthält statt zweier ganzer Zahlen. Darüberhinaus muß bei der Überprüfung einer Signatur ein Potenzprodunkt von drei Elementen statt zwei berechnet werden.

2. Verzichte auf die modulare Reduktion. Dieser Ansatz wurde in [59] für das Schnorr-Signaturverfahren vorgeschlagen und geht auf Ideen aus [27] zurück. Diesem Ansatz werden wir in Abschnitt 6.4 nachgehen.

   Der Preis, den man hier bezahlen muß, besteht darin, daß sowohl bei der Erzeugung, als auch bei der Überprüfung einer Signatur die Exponentiation mit größeren Exponenten durchgeführt werden muß.

Für die weitere Diskussion modifizieren wir zunächst das Signaturverfahren, um die modulare Invertierung in (6.3) zu vermeiden. Dazu wählen wir die DSA-Variante EG II.3 aus [34] aus. Dann wird aus (6.3)

$$s = -ah(M, \varrho) + k \bmod q \ , \tag{6.4}$$

mit entsprechender Modifikation der Signatur ($\mathcal{S}^{\mathrm{EG\ II.3}}_{\mathbf{A},M} = (s, \varrho)$) und der Verifikations-Prozedur. Das resultierende Verfahren hat eigentlich mehr Ähnlichkeit mit dem Schnorr-Signaturverfahren, als mit DSA, aber wir werden dies um der Einheitlichkeit Willen auch als DSA bezeichnen.

Wir verfolgen in diesem Abschnitt die erste Möglichkeit. Nachstehend ist RDSA schematisch für beliebige endliche, Abelsche Gruppen $\mathcal{G}$ beschrieben:

Vorbereitung: **A** macht die folgenden Schritte:
Wähle eine endliche, Abelsche Gruppe $\mathcal{G}$ und eine zufällige 160-Bit-Primzahl q;
wähle ein zufälliges Element $\gamma \in \mathcal{G}$;
wähle eine Zufallszahl a aus $\{2, \ldots, q-1\}$ und berechne $\alpha = \gamma^a$.

As öffentlicher Schlüssel ist $(\mathcal{G}, \gamma, \alpha, q)$, der private Schlüssel ist a.

Signatur: **A** macht die folgenden Schritte:
Wähle eine Zufallszahl k aus $\{0, \ldots, q-1\}$;
berechne $\varrho = \gamma^k$ und $x = -ah(M, \varrho) + k$;
berechne ganze Zahlen s und ℓ, so daß $x = \ell q + s$ und $0 \leqslant s < q$;
berechne $\lambda = \gamma^\ell$.

Die Signatur des Textes M ist $\mathcal{S}_{\mathbf{A},M} = (s, \varrho, \lambda)$.

Verifikation: **B** akzeptiert die Signatur $\mathcal{S}_{\mathbf{A},M}$ genau dann, wenn $0 \leqslant s < q$ und $\gamma^s \alpha^{h(M,\varrho)} \lambda^q = \varrho$ ist.

6.3.2 RDSA ist sicher

Wir zeigen in diesem Abschnitt, daß das RDSA-Signaturverfahren im Zufallsorakel-Modell sicher gegen existenzielle Fälschungen mit Angepaßt-Gewählter-Text-Angriffen ist, wenn das Wurzelproblem schwierig ist.

Wir zeigen zunächst, daß das RDSA-Signaturverfahren *vollständig* ist, d. h. gültige Signaturen werden immer (mit Wahrscheinlichkeit 1) erkannt.

Proposition 6.3.2
*Wenn **A** und **B** dem RDSA-Protokoll folgen, dann ist die Verifikation immer erfolgreich.*

Beweis: Nach Auflösen von α und λ erhält man

$$\gamma^s \alpha^{h(M,\varrho)} \lambda^q = \gamma^{-ah(M,\varrho)+k-\ell q} = \gamma^k = \varrho \ . \tag{6.5}$$

□

Eine RDSA-Signatur $\mathcal{S}_{\mathbf{A},M} = (s, \varrho, \lambda)$ heiße gültig, wenn (6.5) für $\mathcal{S}_{\mathbf{A},M}$ erfüllt ist. Zur Unterscheidung von legitimen Signaturen durch **A** schreiben wir gefälschte Signaturen als $\tilde{\mathcal{S}}_{\mathbf{A},M}$.

Wir zeigen nun, daß die Sicherheit von RDSA auf dem Wurzelproblem beruht (daher der Name, „R“ steht für „Root“):

Proposition 6.3.3
*Ein Angreifer **E**, der in Polynomzeit das Wurzelproblem lösen kann, kann gültige Signaturen für beliebige Texte in Polynomzeit ohne Kenntnis des privaten Schlüssels erzeugen.*

Beweis: Sei $\gamma, \alpha \in \mathcal{G}$ und ein Text M gegeben. **E** wähle $\tilde{\varrho} \in \mathcal{G}$ und $\tilde{s} \in \{0, \ldots, q-1\}$ beliebig und berechne $h(M, \tilde{\varrho})$ und $\tau = \tilde{\varrho} \alpha^{-h(M,\tilde{\varrho})} \gamma^{-\tilde{s}}$. Dann berechne **E** $\tilde{\lambda}$, die q-te Wurzel von τ. Da q eine zufällig gewählte, große Primzahl ist, ist die Wahrscheinlichkeit vernachlässigbar klein,

daß q ein Teiler von $|\mathcal{G}|$ ist; falls τ keine q-te Wurzel hat, treffe $\mathbf{E}$ eine andere Wahl für $\tilde{\varrho}$ und $\tilde{s}$. $\tilde{\mathcal{S}}_{\mathbf{A},M} = (\tilde{s}, \tilde{\varrho}, \tilde{\lambda})$ ist eine gültige Signatur von $\mathbf{A}$ für M. □

Ohne das Zufallsorakel h, wenn also $h(M, \varrho)$ durch M ersetzt wird (wobei wir M hier als ganze Zahl verstehen), sind die folgenden existenziellen Fälschungen möglich:

1. Fälschung mit einem Parameter:
   Sei y eine Zufallszahl aus $\{0, \ldots, q-1\}$. Sei $\tilde{\varrho} = \alpha\gamma^y$, $\tilde{M} \equiv 1 \pmod{q}$, $\tilde{s} \equiv -y \pmod{q}$, $\ell_M = (1 - \tilde{M})/q$ und $\ell_s = (y - \tilde{s})/q$. Sei $\tilde{\lambda} = \alpha^{\ell_M}\gamma^{\ell_s}$. Dann ist das Tripel $(\tilde{s}, \tilde{\varrho}, \tilde{\lambda})$ eine gültige RDSA-Signatur von $\mathbf{A}$ für $\tilde{M}$.

2. Fälschung mit zwei Parametern:
   Seien x und y Zufallszahlen aus $\{0, \ldots, q-1\}$. Sei $\tilde{\varrho} = \alpha^x\gamma^y$, $\tilde{M} \equiv 1 \pmod{q}$, $\tilde{s} \equiv -y \pmod{q}$, $\ell_M = (x - \tilde{M})/q$ und $\ell_s = (y - \tilde{s})/q$. Sei $\tilde{\lambda} = \alpha^{\ell_M}\gamma^{\ell_s}$. Dann ist das Tripel $(\tilde{s}, \tilde{\varrho}, \tilde{\lambda})$ eine gültige RDSA-Signatur von $\mathbf{A}$ für $\tilde{M}$.

Wir zeigen nun, daß das Wurzelproblem in Polynomzeit reduziert werden kann auf das Problem der existenziellen Fälschung einer RDSA Signatur unter Verwendung eines Kein-Text-Angriffs.

Lemma 6.3.4
Wenn die Wahrscheinlichkeit für die existenzielle Fälschung einer RDSA-Signatur in Polynomzeit mit einem Kein-Text-Angriff nicht vernachlässigbar ist, dann kann das Wurzelproblem probabilistisch in Polynomzeit gelöst werden; die Wahrscheinlichkeit wird über alle Zufallsorakel, alle Zufallsbänder und alle gültigen öffentlichen Schlüssel genommen.

Beweis: Angenommen, das Wurzelproblem soll für $(\mathcal{G}, \alpha, q)$ gelöst werden. Wähle $\tau = \alpha^b$ für ein zufälliges b mit $1 < b < q\underline{B}^2_{\mathcal{G}}$, und setze $\gamma = \tau^q$. Nach Voraussetzung kann mit nicht vernachlässigbarer Wahrscheinlichkeit in Polynomzeit mit dem Zufallsorakel h eine gültige Signatur (s, ϱ, λ) für ein M bezüglich des öffentlichen Schlüssels $(\mathcal{G}, \gamma, \alpha, q)$ mittels eines Kein-Text-Angriffs erzeugt werden. Es folgt, daß ein a existieren muß, so daß $\alpha = \gamma^a$.

Seien h_1 und h_2 zwei unterschiedliche Zufallsorakel. Nach dem Forking-Lemma [57, 10, 58] können unter Verwendung desselben Zufallsbandes mit nicht vernachlässigbarer Wahrscheinlichkeit gültige Signaturen $(s_1, \varrho, \lambda_1)$ und $(s_2, \varrho, \lambda_2)$ für dasselbe M erzeugt werden. Seien ℓ_1 und ℓ_2 die diskreten Logarithmen von λ_1 bzw. λ_2 bezüglich γ. Da $\gamma^{s_1}\alpha^{h_1}\lambda_1^q = \gamma^{s_2}\alpha^{h_2}\lambda_2^q = \varrho$ ist, ist

$$s_1 - s_2 \equiv a(h_2 - h_1) + q(\ell_2 - \ell_1) \pmod{|\mathcal{G}|} \ . \tag{6.6}$$

Sei $w \equiv (h_2 - h_1)^{-1} \pmod{q}$. Dann gibt es eine ganze Zahl ℓ_w, so daß $w(h_2 - h_1) = 1 + \ell_w q$ ist. Wir multiplizieren (6.6) mit w. Sei $w(s_1 - s_2) = v + \ell_v q$ für ganze Zahlen v und ℓ_v, so daß $0 \leqslant v < q$. Dann ist

$$a \equiv v + \ell_v q - a\ell_w q + qw(\ell_1 - \ell_2) \pmod{|\mathcal{G}|} \ . \tag{6.7}$$

Mit $\theta = \gamma^{\ell_v}\alpha^{-\ell_w}\lambda_1^w\lambda_2^{-w}$ ist $\alpha = \gamma^v\theta^q = (\tau^v\theta)^q$. Also ist $\alpha^{bv}\theta$ eine q-te Wurzel von α. □

Wir gehen nun einen Schritt weiter und zeigen, daß das Wurzelproblem in Polynomzeit reduziert werden kann auf das Problem der existenziellen Fälschung einer RDSA Signatur

unter Verwendung eines Angepaßt-Gewählter-Text-Angriffs. Bei einem Angepaßt-Gewählter-Text-Angriff darf der Angreifer **E** den legitimen Signierer **A** als Orakel benutzen. Wir bezeichnen zwei statistische Verteilungen als *polynomiell ununterscheidbar*, wenn keine probabilistische Turing-Maschine existiert, welche die Verteilungen in polynomiell beschränkter Zeit mit nicht vernachlässigbarer Wahrscheinlichkeit unterscheiden kann. Um die Reduktion zu zeigen, müssen wir **A** durch einen Simulator ersetzen (modelliert durch eine probabilistische Turing-Maschine), dessen Ausgabeverteilung polynomiell ununterscheidbar ist von **A**s Ausgabeverteilung. Wenn es so einen Simulator gibt, dessen Laufzeit polynomiell beschränkt ist, dann kann ein Angepaßt-Gewählter-Text-Angriff reduziert werden auf einen Keinen-Text-Angriff, und nach Lemma 6.3.4 folgt dann Theorem 6.3.1.

Wir beschreiben nun die Konstruktion des Simulators im Detail. Da wir die Hash-Funktion h beim Signierer als Zufallsorakel annehmen, ist die Ausgabe von h für alle Texte M gleichverteilt. Um eine Ausgabeverteilung zu erzielen, die sich von der Ausgabeverteilung des Signierers in Polynomzeit nicht unterscheiden läßt, wählt der Simulator für ein gegebenes M das entsprechende $\tilde{h}$ zufällig aus und wendet die Fälschung mit zwei Parametern an, um ein Tripels $(\tilde{s}, \tilde{\varrho}, \tilde{\lambda})$ zu erzeugen. Im einzelnen wählt der Simulator zufällig ein y aus $\{1, \ldots, B_k\}$, wobei B_k etwas später noch spezifiziert wird, und setzt $\tilde{\varrho} = \alpha^{\tilde{h}}\gamma^y$, $\tilde{s} = -\tilde{h} \bmod q$ und $\tilde{\lambda} = \gamma^{(y-\tilde{s})/q}$. Es folgt, daß $\gamma^{\tilde{s}}\alpha^{\tilde{h}}\tilde{\lambda}^q = \gamma^{\tilde{s}+y-\tilde{s}}\alpha^{\tilde{h}} = \tilde{\varrho}$.

Wir bestimmen nun B_k, um zu garantieren, daß die Ausgabeverteilungen des Signierers und des Simulators in polynomieller Zeit nur mit vernachlässigbarer Wahrscheinlichkeit unterschieden werden können. Der Signierer wählt k zufällig aus $\{1, \ldots, B_k\}$ mit Gleichverteilung aus, wobei $B_k = q - 1$, somit wählt der Signierer ϱ zufällig aus $\mathcal{R} = \{\gamma, \ldots, \gamma^{B_k}\}$ mit Gleichverteilung aus. Beim Simulator ist $\log_\gamma \tilde{\varrho} = \tilde{k} = \tilde{h}a + y \bmod |\mathcal{G}|$. $\tilde{k}$ ist möglicherweise größer, als $q - 1$, somit ist möglicherweise $\tilde{\varrho} \notin \mathcal{R}$. Unter der Annahme, daß in polynomieller Zeit nur mit vernachlässigbarer Wahrscheinlichkeit entschieden werden kann, ob $\tilde{\varrho} \in \mathcal{R}$ oder $\tilde{\varrho} \notin \mathcal{R}$ (d. h. ob eine Signatur vom Signierer erzeugt wurde oder gefälscht ist), dann genügt $B_k = q-1$ für Theorem 6.3.1.

Andernfalls muß das Signaturverfahren noch modifiziert werden, um für den Signierer und den Simulator dieselbe Wahrscheinlichkeits-Verteilung für ϱ bzw. $\tilde{\varrho}$ zu erreichen; idealerweise wird ϱ bzw. $\tilde{\varrho}$ gleichverteilt aus $\{\gamma^i : 0 \leqslant i < |\mathcal{G}|\}$ gewählt. Dies kann erreicht werden, indem B_k entsprechend gewählt wird. Wir erinnern daran, daß $\underline{B_\mathcal{G}}$ und $\overline{B_\mathcal{G}}$ Abschätzungen für $|\mathcal{G}|$ seien, d. h. $\underline{B_\mathcal{G}} \leqslant |\mathcal{G}| \leqslant \overline{B}_\mathcal{G} = B_\mathcal{G}\underline{B_\mathcal{G}}$, mit $B_\mathcal{G}/|\mathcal{G}|$ vernachlässigbar. Wir zeigen, daß die Verteilungen von ϱ und $\tilde{\varrho}$ polynomiell ununterscheidbar sind, wenn $B_k = q\underline{B}_\mathcal{G}^2$ ist. Dazu betrachten wir zunächst den (hypothetischen) Fall, daß $B_k = q|\mathcal{G}|$ gewählt wäre.

Lemma 6.3.5
Sei $B_k = q|\mathcal{G}|$. Dann gilt:

1. *Die Wahrscheinlichkeit, daß der Signierer eine bestimmte Signatur (s, ϱ, λ) erzeugt, beträgt $1/B_k$.*

2. *Die Wahrscheinlichkeit, daß der Simulator eine bestimmte Signatur $(\tilde{s}, \tilde{\varrho}, \tilde{\lambda})$ erzeugt, beträgt $1/B_k$.*

Beweis: Nach Voraussetzung sind h und $\tilde{h}$ Zufallsorakel, welche alle Eingaben gleichverteilt auf $\{1, \ldots q-1\}$ abbilden. Die Wahrscheinlichkeit für jede mögliche Ausgabe von h bzw. $\tilde{h}$

beträgt also $1/(q-1)$. Weiterhin war die Primzahl q kein Teiler von $|\mathcal{G}|$. Dann gibt es für jedes Tripel (s, ϱ, h) ein eindeutig bestimmtes λ mit $\lambda^q = \varrho\gamma^{-s}\alpha^{-h}$.

1. Sei k_0 der diskrete Logarithmus von ϱ bezüglich γ. Für alle verschiedenen q Werte für $k = k_0 + i|\mathcal{G}|$ mit $0 \leqslant k_0 < |\mathcal{G}|$ und $0 \leqslant i < q$ ist $\varrho = \gamma^k$. Zu einem gegebenen h ist i durch $s \equiv -ah + k \pmod q$ jedoch eindeutig bestimmt. Das bedeutet, daß es für jedes h genau ein k aus $\{1, \ldots, B_k\}$ gibt, um eine bestimmte Signatur (s, ϱ, λ) zu erzeugen. Daher beträgt die Wahrscheinlichkeit für jede Signatur (s, ϱ, λ)

$$\sum_{1 \leqslant h < q} \frac{1}{q-1} \cdot \frac{1}{B_k} = \frac{1}{B_k} \ .$$

2. Für alle verschiedenen q Werte für $y = y_0 + j|\mathcal{G}|$ mit $0 \leqslant y_0 < |\mathcal{G}|$ und $0 \leqslant j < q$ ist $\tilde{\varrho} = \alpha^{\tilde{h}}\gamma^y$. Nach Voraussetzung ist $y \equiv -\tilde{s} \pmod q$, daher existiert zu jedem $\tilde{h}$ genau ein y, zu welchem der Simulator die Signatur $(\tilde{s}, \tilde{\varrho}, \tilde{\lambda})$ erzeugen kann. Daher beträgt die Wahrscheinlichkeit für jede Signatur $(\tilde{s}, \tilde{\varrho}, \tilde{\lambda})$

$$\sum_{1 \leqslant h < q} \frac{1}{q-1} \cdot \frac{1}{B_k} = \frac{1}{B_k} \ .$$

□

Mit $B_k = q|\mathcal{G}|$ erzeugen Signierer und Simulator also identische Ausgabeverteilungen. Nach Voraussetzung war $|\mathcal{G}|$ aber unbekannt, also kann B_k nicht so gewählt werden. Wir zeigen nun, daß sich die Ausgabeverteilungen für den Signierer und den Simulator mit $B_k = q\underline{B}_{\mathcal{G}}^2$ jeweils nur unwesentlich von den Ausgabeverteilungen mit $B_k = q|\mathcal{G}|$ unterscheiden. Daraus folgt, daß sich die Ausgabeverteilungen von Signierer und Simulator mit $B_k = q\underline{B}_{\mathcal{G}}^2$ nur unwesentlich voneinander unterscheiden.

Lemma 6.3.6
Sei $B_k = q\underline{B}_{\mathcal{G}}^2$. Wenn k beim RDSA-Signaturverfahren gleichverteilt aus $\{1, \ldots, B_k\}$ gewählt wird, dann erzeugt ein Simulator unter Verwendung der Fälschung mit zwei Parametern Signaturen $(\tilde{s}, \tilde{\varrho}, \tilde{\lambda})$, deren Verteilung polynomiell ununterscheidbar von der Verteilung der Signaturen (s, ϱ, λ) ist, die vom Signierer erzeugt werden.

Beweis: Wir setzen voraus, daß für alle Texte M und jedes Element $\varrho \in \mathcal{G}$ die Ausgaben jedes Zufallsorakels h gleichverteilt sind. Wir setzen weiterhin voraus, daß k und y von Signierer bzw. vom Simulator gleichverteilt aus $\{1, \ldots, B_k\}$ gewählt werden.

Nach Lemma 6.3.5 sind die Ausgaben-Verteilungen des Signierers und des Simulators identisch, wenn $B_k = q|\mathcal{G}|$, d. h. die statistische Differenz $\Delta_{q|\mathcal{G}|} = 0$. Da nach Voraussetzung $|\mathcal{G}|$ aber unbekannt war, kann B_k nicht so gewählt werden. Wir wählen stattdessen $B_k = q\underline{B}_{\mathcal{G}}^2$ und zeigen, daß die statistische Differenz vernachlässigbar ist.

Es existieren ganze Zahlen v und w, mit $0 < v$ und $0 \leqslant w < q|\mathcal{G}|$, so daß

$$q\underline{B}_{\mathcal{G}}^2 = vq|\mathcal{G}| + w \tag{6.8}$$

ist. Insbesondere ist $(v+1)q|\mathcal{G}| > q\underline{B}_{\mathcal{G}}^2 \geqslant vq|\mathcal{G}|$, woraus $\dfrac{v+1}{q\underline{B}_{\mathcal{G}}^2} > \dfrac{1}{q|\mathcal{G}|} \geqslant \dfrac{v}{q\underline{B}_{\mathcal{G}}^2}$ folgt.

Jede Signatur (s, ϱ, λ) ist eindeutig bestimmt durch $h \in \{1, \ldots, q-1\}$ und $k \in \{0, \ldots, q|\mathcal{G}|\}$. Sei nun $k \in \{1, \ldots, q\underline{B}_{\mathcal{G}}^2\}$, dann gibt es für jedes solche k ganze Zahlen k_0 und ℓ_k mit $0 \leqslant k_0 < q|\mathcal{G}|$, so daß $k = k_0 + \ell_k q|\mathcal{G}|$.

Seien h und k_0 fest. Falls $0 \leqslant k_0 \leqslant w$ ist, dann gibt es genau $v+1$ unterschiedliche k mit $k \equiv k_0 \pmod{q|\mathcal{G}|}$, und falls $w < k_0 < q|\mathcal{G}|$ ist, dann gibt es genau v unterschiedliche k mit $k \equiv k_0 \pmod{q|\mathcal{G}|}$. Die Wahrscheinlichkeit, daß der Signierer eine bestimmte Signatur zu einem bestimmten k_0 erzeugt, beträgt daher $\dfrac{v+1}{q\underline{B}_{\mathcal{G}}^2}$ bzw. $\dfrac{v}{q\underline{B}_{\mathcal{G}}^2}$. Dann ist die statistische Differenz der Ausgabeverteilungen für $B_k = q|\mathcal{G}|$ und $B_k = q\underline{B}_{\mathcal{G}}^2$ bestimmt durch

$$\begin{aligned}\Delta_{\mathsf{sig}} &= \sum_{(s,\varrho,\lambda)} \left| \Pr\left[\ (s,\varrho,\lambda) : B_k = q|\mathcal{G}|\ \right] - \Pr\left[\ (s,\varrho,\lambda) : B_k = q\underline{B}_{\mathcal{G}}^2\ \right] \right| \\ &= (w+1)\left(\frac{v+1}{q\underline{B}_{\mathcal{G}}^2} - \frac{1}{q|\mathcal{G}|}\right) + (q|\mathcal{G}| - w - 1)\left(\frac{1}{q|\mathcal{G}|} - \frac{v}{q\underline{B}_{\mathcal{G}}^2}\right) \ .\end{aligned}$$

Wegen $q\underline{B}_{\mathcal{G}}^2 \geqslant vq|\mathcal{G}|$ ist $\dfrac{v+1}{q\underline{B}_{\mathcal{G}}^2} - \dfrac{1}{q|\mathcal{G}|} \leqslant \dfrac{1}{q\underline{B}_{\mathcal{G}}^2}$, wegen $q\underline{B}_{\mathcal{G}}^2 < (v+1)q|\mathcal{G}|$ ist $\dfrac{1}{q|\mathcal{G}|} - \dfrac{v}{q\underline{B}_{\mathcal{G}}^2} < \dfrac{1}{q\underline{B}_{\mathcal{G}}^2}$. Es ist daher

$$\Delta_{\mathsf{sig}} < (w+1)\frac{1}{q\underline{B}_{\mathcal{G}}^2} + (q|\mathcal{G}| - w - 1)\frac{1}{q\underline{B}_{\mathcal{G}}^2} = \frac{q|\mathcal{G}|}{q\underline{B}_{\mathcal{G}}^2} \leqslant \frac{B_{\mathcal{G}}^2}{|\mathcal{G}|} \ . \tag{6.9}$$

Diese Differenz ist vernachlässigbar. Die gleiche Argumentation für den Simulator zeigt, daß die entsprechende Differenz Δ_{sim} ebenfalls vernachlässigbar ist. Nach der Dreiecks-Ungleichung ist daher die statistische Differenz Δ zwischen den Ausgabeverteilungen des Signierers und des Simulators mit $B_k = q\underline{B}_{\mathcal{G}}^2$ beschränkt durch

$$\Delta \leqslant \Delta_{q|\mathcal{G}|} + \Delta_{\mathsf{sig}} + \Delta_{\mathsf{sim}} \ , \tag{6.10}$$

was vernachlässigbar ist, d. h. die Ausgabeverteilungen von Signierer und Simulator sind in Polynomzeit nur mit vernachlässigbarer Wahrscheinlichkeit unterscheidbar. □

Beweis zu Theorem 6.3.1: Sei **E** ein Angreifer, der mit nicht vernachlässigbarer Wahrscheinlichkeit eine RDSA-Signatur mittels eines Angepaßt-Gewählter-Text-Angriff in polynomiell beschränkter Zeit fälschen kann. Nach Lemma 6.3.6 kann der Signierer durch einen Poloynomzeit-Simulator ersetzt werden, dessen Ausgabeverteilung polynomiell ununterscheidbar ist von der Ausgabeverteilung des Signierers. Das bedeutet, daß **E** durch eine probabilistische Turing-Maschine mit polynomieller Laufzeit modelliert werden kann. Diese Turing-Maschine kann in Lemma 6.3.4 verwendet werden, um mit nicht vernachlässigbarer Wahrscheinlichkeit in polynomiell beschränkter Laufzeit das Wurzelproblem zu lösen. □

6.4 IQ-DSA

Wir folgen in diesem Abschnitt der Idee, auf die modulare Reduktion zu verzichten. Dies wurde z. B. in [59] für das Schnorr-Signaturverfahren vorgeschlagen und geht auf Ideen aus [27] zurück.

Wir zeigen zunächst die schematische Darstellung der DSA-Variante EG II.3 ohne modulare Reduktion des Exponenten nach [34].

Vorbereitung: **A** macht die folgenden Schritte:
Wähle eine endliche, Abelsche Gruppe $\mathcal{G}$;
wähle ein zufälliges Element $\gamma \in \mathcal{G}$;
wähle eine Zufallszahl a aus $\{1, \ldots, B_a\}$ und berechne $\alpha = \gamma^a$.

As öffentlicher Schlüssel ist $(\mathcal{G}, \gamma, \alpha)$, der private Schlüssel ist a.

Signatur: **A** macht die folgenden Schritte:
Wähle eine Zufallszahl k aus $\{1, \ldots, B_k\}$;
berechne $\varrho = \gamma^k$ und $s = -ah(M, \varrho) + k$.

Die Signatur des Textes M ist $\mathcal{S}_{\mathbf{A},M} = (s, \varrho)$.

Verifikation: **B** akzeptiert die Signatur $\mathcal{S}_{\mathbf{A},M}$ genau dann, wenn $\gamma^s \alpha^{h(M,\varrho)} = \varrho$ ist.

Das DSA-Signaturverfahren ist vollständig, eine gültige Signaturen wird immer akzeptiert.

Proposition 6.4.1
*Wenn **A** und **B** dem vorstehenden DSA-Protokoll folgen, dann ist die Verifikation immer erfolgreich.*

Beweis: Nach Auflösen von α erhält man

$$\gamma^s \alpha^{h(M,\varrho)} = \gamma^{-ah(M,\varrho)+k} = \gamma^k = \varrho \ . \tag{6.11}$$

□

Eine ausführliche Diskussion der Sicherheit für das fast identische Schnorr-Signaturverfahren steht in [59]. Wir zitieren hier nur das Ergebnis [59, Theorem 10]:

Theorem 6.4.2
Seien $a \in \{1, \ldots, B_a\}$, $k \in \{1, \ldots, B_k\}$ zufällig gewählt, sei $h \in \{1, \ldots, B_h\}$ und sei B_n die maximale Anzahl von Texten, die mit einem festen Schlüssel signiert werden. Unter der Voraussetzung, daß $B_h B_a B_n / B_k$ und $1/B_h$ vernachlässigbar sind, gilt: Wenn die existenzielle Fälschung einer DSA-Signatur mit Hilfe eines Angepaßt-gewählter-Text-Angriffs in Polynomzeit eine nicht vernachlässigbare Erfolgswahrscheinlichkeit hat, dann können diskrete Logarithmen in $\mathcal{G}$ mit nicht vernachlässigbarer Wahrscheinlichkeit in Polynomzeit berechnet werden.

Die geforderte Vernachlässigbarkeit von $B_h B_a B_n / B_k$ erfordert, daß B_k erheblich viel größer sein muß, als z. B. bei RDSA. Wenn 2^{-t} als vernachlässigbar gilt, dann wird man $B_h = B_a = 2^{2t}$ wählen, und dann muß $B_k \geqslant 2^{5t}$ sein. Falls z. B. 2^{-80} als vernachlässigbar gilt und $B_h = B_a = 2^{160}$ ist, dann muß $B_k \geqslant 2^{400}$ sein.

Dies wirkt sich unmittelbar auf die Effizienz der Signatur-Erzeugung aus, da der zufällig gewählte Exponent k etwa 400 Bits lang sein muß, anstelle von etwa 160 Bits bei RDSA oder „traditionellem“ DSA. Durch geeignete Multi-Exponentiations-Verfahren mit Vorberechnung kann dies zu einem gewissen Grad wieder kompensiert werden, siehe Abschnitt 7.2.1.

6.5 IQ-GQ

Wir zeigen nun, daß das Guillou-Quisquater-Signaturverfahren (GQ) im Zufallsorakel-Modell sicher gegen einen Angriff mit angepaßt ausgewählten Texten ist, d. h. wir zeigen, daß bei in solchen Gruppen die Berechnung von Wurzeln und die existenzielle Fälschung von GQ-Signaturen mit Angepaßt-Gewählter-Text-Angriffen komplexitätstheoretisch äquivalent sind.

Das GQ-Signaturverfahren geht aus dem gleichnamigen Identifikationsverfahren hervor [30]. Es wurde ursprünglich für für Gruppen $\mathbb{Z}/n\mathbb{Z}^*$ formuliert, wobei $n = p \cdot q$ analog zum RSA-Modulus gewählt wird. Das besondere am GQ-Signaturverfahren ist, daß weder der Signierer, noch der Verifizierer die Gruppenordnung benötigen. Diese Eigenschaft macht das GQ-Signaturverfahren zu einem geeigneten Kandidaten für die Verwendung mit Klassengruppen.

Wir formulieren zunächst eine für beliebige Gruppen $\mathcal{G}$ verallgemeinerte Version des GQ-Signaturverfahrens:

Vorbereitung: **A** macht die folgenden Schritte:
Wähle eine endliche, Abelsche Gruppe $\mathcal{G}$;
wähle ein zufälliges Element $\alpha \in \mathcal{G}$;
wähle eine Zufallszahl n aus $\{1, \ldots, 2^{160}\}$ und berechne $\nu = \alpha^{-n}$.

As öffentlicher Schlüssel ist $(\mathcal{G}, \nu, n)$, der private Schlüssel ist α.

Signatur: **A** macht die folgenden Schritte:
Wähle ein zufälliges Element $\kappa \in \mathcal{G}$;
berechne $\varrho = \kappa^n$;
berechne $s = h(M, \varrho)$ und $\sigma = \kappa\alpha^s$.

Die Signatur des Textes M ist $\mathcal{S}_{\mathbf{A},M} = (s, \sigma)$.

Verifikation: **B** berechnet $\phi = \sigma^n\nu^s$ und akzeptiert die Signatur $\mathcal{S}_{\mathbf{A},M}$ genau dann, wenn $h(M, \phi) = s$ ist.

6.5.1 GQ ist sicher

Wir zeigen zunächst, daß das GQ-Signaturverfahren vollständig ist, d. h. gültige Signaturen werden immer (mit Wahrscheinlichkeit 1) erkannt.

Proposition 6.5.1
*Wenn **A** und **B** dem GQ-Protokoll folgen, dann ist die Verifikation immer erfolgreich.*

Beweis: Nach Auflösen von σ und ν erhält man

$$\phi = \sigma^n\nu^s = (\kappa\alpha^s)^n\alpha^{-sn} = \kappa^n = \varrho \ , \tag{6.12}$$

und daher ist $h(M, \phi) = h(M, \varrho) = s$. □

Eine GQ-Signatur $\mathcal{S}_{\mathbf{A},M} = (s, \sigma)$ heiße gültig, wenn (6.12) für $\mathcal{S}_{\mathbf{A},M}$ erfüllt ist. Zur Unterscheidung von legitimen Signaturen durch **A** schreiben wir gefälschte Signaturen als $\tilde{\mathcal{S}}_{\mathbf{A},M}$.

Aus der Schlüsselerzeugung von GQ ist unmittelbar einsichtig, daß die Sicherheit von GQ auf dem Wurzelproblem beruht:

Proposition 6.5.2
Ein Angreifer ***E****, der in Polynomzeit das Wurzelproblem lösen kann, kann gültige Signaturen für beliebige Texte in Polynomzeit ohne Kenntnis des privaten Schlüssels erzeugen.*

Ohne das Zufallsorakel h, wenn also $h(M, \varrho)$ keine Einweg-Funktion ist, sind z. B. die folgenden existenziellen Fälschungen möglich:

1. Wähle $\tilde{s}$ und κ zufällig, setze $\tilde{\sigma} = \kappa^{\tilde{s}}$ und berechne $\tilde{\varrho} = (\kappa^n \nu)^{\tilde{s}}$. Nach Voraussetzung war h keine Einweg-Funktion, d. h. es kann ein M berechnet werden, so daß $h(M, \tilde{\varrho}) = \tilde{s}$ ist. Dann ist $(\tilde{s}, \tilde{\sigma})$ eine gültige GQ-Signatur für dieses M.

2. Wähle $\tilde{s}$ und κ zufällig, wähle $y \in \{1, \ldots, B_y\}$ für ein noch zu bestimmendes B_y, setze $\tilde{\sigma} = \kappa^{\tilde{s}y}$ und berechne $\tilde{\varrho} = (\kappa^{ny} \nu)^{\tilde{s}}$. Nach Voraussetzung war h keine Einweg-Funktion, d. h. es kann ein M berechnet werden, so daß $h(M, \tilde{\varrho}) = \tilde{s}$ ist. Dann ist $(\tilde{s}, \tilde{\sigma})$ eine gültige GQ-Signatur für dieses M.

Wir zeigen nun einen Kein-Text-Angriff auf GQ:

Lemma 6.5.3
Wenn die Wahrscheinlichkeit für die existenzielle Fälschung einer GQ-Signatur in Polynomzeit mit einem Kein-Text-Angriff nicht vernachlässigbar ist, dann kann das Wurzelproblem probabilistisch in Polynomzeit gelöst werden; die Wahrscheinlichkeit wird über alle Zufallsorakel, alle Zufallsbänder und alle gültigen öffentlichen Schlüssel genommen.

Beweis: Angenommen, das Wurzelproblem soll für $(\mathcal{G}, \nu, n)$ gelöst werden. Nach Voraussetzung kann mit nicht vernachlässigbarer Wahrscheinlichkeit in Polynomzeit mit dem Zufallsorakel h eine gültige Signatur (κ, s, σ) für ein M bezüglich des öffentlichen Schlüssels $(\mathcal{G}, \nu, n)$ mittels eines Kein-Text-Angriffs erzeugt werden. Es folgt, daß ein n existieren muß, so daß $\nu = \alpha^n$.

Seien h_1 und h_2 zwei unterschiedliche Zufallsorakel. Nach dem Forking-Lemma [57, 10, 58] können unter Verwendung desselben Zufallsbandes mit nicht vernachlässigbarer Wahrscheinlichkeit gültige Signaturen (s_1, σ_1) und (s_2, σ_2) für dasselbe M erzeugt werden. Da das Zufallsband in beiden Fällen dasselbe ist, gilt $\varrho = \sigma_1^n \nu^{s_1} = \sigma_2^n \nu^{s_2}$. Dann ist $(\sigma_1/\sigma_2)^n = \nu^{s_2 - s_1}$. Die Wahrscheinlichkeit, daß $\mathrm{ggT}(n, s_2 - s_1) > 1$ ist, ist vernachlässigbar. Sei also $n \perp s_2 - s_1$, dann existieren ganze Zahlen u und v, so daß $(s_2 - s_1)u + nv = 1$ ist. Dann ist $(\sigma_1/\sigma_2)^{nu} = \nu^{1-nv}$, also ist

$$\nu = \left((\sigma_1/\sigma_2)^u \nu^v\right)^n \ , \tag{6.13}$$

somit ist $(\sigma_1/\sigma_2)^u \nu^v$ eine n-te Wurzel von ν. □

Die Konstruktion des Simulators ist nun ähnlich wie bei RDSA. Mittels der Fälschung mit einem Parameter wird zu einem zufällig gewählten $\tilde{s}$ passendes $\tilde{\varrho}$, $\tilde{h}$ und $\tilde{\sigma}$ bestimmt. Je nachdem, wie der Signierer κ tatsächlich auswählt, kann B_y so gewählt werden, daß die Ausgaben des Simulators polynomiell ununterscheidbar von den Ausgaben des Signierers sind. Damit wird der Angepaßt-gewählter-Text-Angriff zurückgeführt auf den Kein-Text-Angriff, und mit Lemma 6.5.3 folgt dann das folgende Theorem:

Theorem 6.5.4
Sei $\mathcal{G}$ eine endliche, Abelsche Gruppe. Unter der Annahme, daß es keinen Algorithmus gibt, der das Wurzelproblem in $\mathcal{G}$ mit nicht vernachlässigbarer Wahrscheinlichkeit in polynomiell beschränkter Zeit berechnet, ist GQ in $\mathcal{G}$ im Zufallsorakel-Modell sicher.

6.6 IQ-IES

IES (auch bekannt als DHIES) geht zurück auf [1] (bzw. dessen Vorläufer). Wir geben eine schematische Darstellung von IES:

Vorbereitung: **A** macht die folgenden Schritte:
Wähle eine endliche, Abelsche Gruppe $\mathcal{G}$;
wähle ein zufälliges Element $\gamma \in \mathcal{G}$;
wähle eine Zufallszahl a aus $\{1, \ldots, B_a\}$ und berechne $\alpha = \gamma^a$.

As öffentlicher Schlüssel ist $(\mathcal{G}, \gamma, \alpha)$, der private Schlüssel ist a.

Verschlüsselung: **B** macht die folgenden Schritte:
Wähle eine Zufallszahl k aus $\{1, \ldots, B_a\}$;
berechne $\varrho = \gamma^k$ und $\sigma = \alpha^k$;
berechne $K = \mathit{deriveKey}(\sigma)$;
berechne K_{ENC} und K_{MAC} aus K;
berechne $EM = ENC(M, K_{ENC})$;
berechne $T = MAC(EM, K_{MAC})$.

Die Verschlüsselung des Textes M ist $\mathcal{C}_{\mathbf{A},M} = (\varrho, EM, T)$.

Entschlüsselung: **A** macht die folgenden Schritte:
berechne $\sigma = \varrho^a$;
berechne $K = \mathit{deriveKey}(\sigma)$;
berechne K_{ENC} und K_{MAC} aus K;
berechne $T' = MAC(EM, K_{MAC})$.

Falls $T' \neq T$ verwerfe $\mathcal{C}_{\mathbf{A},M}$, andernfalls berechne den Klartext $M = ENC^{-1}(EM, K_{ENC})$.

Das IES-Verschlüsselungsverfahren ist vollständig, d. h. jeder Klartext, der mit der vorstehenden Verschlüsselungs-Methode verschlüsselt wurde, kann wieder eindeutig rekonstruiert werden.

Proposition 6.6.1
*Wenn **A** und **B** dem vorstehenden IES-Protokoll folgen, dann kann aus der Verschlüsselung eines jeden Klartexts derselbe Klartext immer eindeutig rekonstruiert werden.*

Beweis: Nach Auflösen von α und ϱ erhält man

$$\alpha^k = \varrho^a = \sigma \ . \tag{6.14}$$

Damit sind die Schlüssel K_{MAC} und K_{ENC} bei Verschlüsselung und Entschlüsselung jeweils identisch, und daher ist $ENC^{-1}(EM, K_{ENC}) = ENC^{-1}(ENC(M, K_{ENC}), K_{ENC}) = M$ und $T' = MAC(EM, K_{MAC} = T$. □

Eine ausführliche Untersuchung der Sicherheit von IES steht in [1]. IES ist sicher gegen Angepaßt-Gewählter-Schlüsseltext-Angriffe, wenn *deriveKey*, *MAC* und *ENC* geeignet gewählt werden:

Theorem 6.6.2
Seien deriveKey, ENC und MAC gegeben mit folgenden Eigenschaften:

1. *Gegeben γ^a, γ^k und γ^{ak}. Die Wahrscheinlichkeit für den Erfolg einer Unterscheidung von $deriveKey(\gamma^{ak})$ von einem zufällig gewählten Bit-String ist vernachlässigbar.*

2. *Die Wahrscheinlichkeit für den Erfolg eines Angriffs auf ENC mittels ausgewählten Klartexten ist vernachlässigbar.*

3. *Die Wahrscheinlichkeit für den Erfolg eines Angriffs auf MAC mittels ausgewählten Texten ist vernachlässigbar.*

Dann die Wahrscheinlichkeit für den Erfolg einer Unterscheidung mittels eines Angepaßt-Gewählter-Schlüsseltext-Angriffs vernachlässigbar.

Bezüglich Klassengruppen bedeutet das, daß IQ-IES sicher ist gegen eine Unterscheidung mittels eines Angepaßt-Gewählter-Schlüsseltext-Angriffs, wenn das IQ-DHP nur mit vernachlässigbarer Wahrscheinlichkeit effizient lösbar ist, andernfalls wäre die Bedingung 1 nicht erfüllt.

In [1] wird für das Verfahren wie vorstehend die Annahme gemacht, daß $|\mathcal{G}|$ prim sei, da das Verfahren andernfalls nicht sicher gegen Verformbarkeit (malleability) ist, d. h. es wäre möglich, daß zu gegebenem a verschiedene Werte von k_1 und k_2 existieren, so daß $\gamma^{k_1} \neq \gamma^{k_2}$ aber $\gamma^{ak_1} = \gamma^{ak_2}$ ist.

Beispiel: Sei $\Delta = -167215\ 239731\ 234120\ 638999$, dann ist $h(\Delta) = 61 \cdot 5189\ 363363$. $\mathfrak{a} = (82882\ 425130,\ 55412\ 500549)$ ist ein $\mathcal{O}_\Delta$-Ideal, und $\mathrm{Cl}(\Delta)$ wird von $[\mathfrak{a}]$ erzeugt. Seien $a = 238746\ 897343$, $k_1 = 192036\ 866839$ und $k_2 = 254281\ 846979$. Dann ist $\mathfrak{r}_1 = \overline{\mathfrak{a}^{k_1}} = (36907\ 476134,\ 18229\ 914661)$ und $\mathfrak{r}_2 = \overline{\mathfrak{a}^{k_2}} = (16883\ 905193,\ 12834\ 269249)$, aber $\overline{\mathfrak{r}_1^a} = \overline{\mathfrak{r}_2^a} = (124852\ 512466,\ -87074\ 144557)$.

Dieses Beispiel beruht darauf, daß $61 \mid a$, und daß $\mathfrak{r}_1$ und $\mathfrak{r}_2$ in der Abbildung auf die Untergruppe der Ordnung 5189 363363 identisch sind, aber nicht in der Abbildung auf die Untergruppe der Ordnung 61. Das Beispiel konnte konstruiert werden, weil die Faktorisierung von $|\langle[\mathfrak{a}]\rangle| = h(\Delta)$ bekannt war.

Um dies zu beheben, wird in [1] vorgeschlagen, nicht nur $\sigma = \gamma^{ak}$ sondern auch $\varrho = \gamma^k$ zur Schlüsselableitung heranzuziehen, also $K = deriveKey(\sigma, \varrho)$. Dardurch ist das Verfahren auch dann gegen Verformbarkeit sicher, wenn $|\mathcal{G}|$ keine Primzahl ist.

Im Fall von Klassengruppen ist die Ordnung eines Elementes für große Diskriminanten jedoch nicht effizient berechenbar. Dies ist aber notwendig für eine Verformung, daher ist die Wahrscheinlichkeit einer erfolgreichen Verformung beim gegebenen Verfahren im Fall von Klassengruppen vernachlässigbar.

6.7 Die Verwendung kurzer Exponenten bei IQ-Kryptoverfahren

Die Klassenzahl einer großen, zufällig gewählten Diskriminante kann i. allg. nicht effizient berechnet werden, dies gilt insbesondere für Fundamental-Diskriminanten. Wir haben in Abschnitt 5.2.4 ein Verfahren gesehen, mit dem die Klassenzahl effizient berechnen werden kann, wenn die Klassenzahl sehr glatt ist. Wir haben aber auch gezeigt, daß die Wahrscheinlichkeit sehr gering ist, daß die Klassenzahl einer großen, zufällig gewählten Fundamental-Diskriminante sehr glatt ist.

Wir betrachten nun den Fall, daß die Klassenzahl selber nicht glatt ist, aber einen großen, sehr glatten Faktor enthält. Wenn dieser glatte Faktor bekannt wäre, dann könnte der Pohlig-Hellman-Algorithmus zur Berechnung diskreter Logarithmen in der Klassengruppe herangezogen werden, deren Beträge kleiner sind, als der glatte Faktor [78].

Es ist jedoch kein Algorithmus bekannt, mit dem effizient Vielfache oder Teiler (insbesondere glatte Teiler, außer Potenzen von 2) der Klassenzahl berechnet werden können, ohne die Klassenzahl selber zu kennen.

Dies hat zur Folge, daß wir bei allen IQ-Kryptoverfahren nicht gezwungen sind, Exponenten in der Größenordnung der Klassenzahl zu verwenden. Stattdessen können relativ kleine Exponenten verwendet werden. Wenn z. B. t den Sicherheitsparameter bezeichnet, dann sind Exponenten in der Größenordnung von 2^{2t} ausreichend (sofern keine anderen Gründe dagegen sprechen, wie z. B. beim Verfahren zur Erzeugung einer IQ-DSA-Signatur).

Kapitel 7

Effiziente IQ-Arithmetik

In diesem Kapitel zeigen wir, daß IQ-Arithmetik und damit die IQ-Kryptoverfahren effizient sind. In Kapitel 2 hatten wir bereits dargelegt, daß wir unter Arithmetik in Klassengruppen Ideal-Arithmetik mit den reduzierten Vertreter-Idealen verstehen: Sind $\mathfrak{a}$ und $\mathfrak{b}$ Elemente einer Klassengruppe $\mathrm{Cl}(\Delta)$, vertreten durch die *reduzierten* $\mathcal{O}_\Delta$-Ideale $\mathfrak{a}$ und $\mathfrak{b}$, d. h. $\mathfrak{a} = [\mathfrak{a}]$ und $\mathfrak{b} = [\mathfrak{b}]$, dann meinen wir mit der Berechnung von $\mathfrak{a} \cdot \mathfrak{b}$ oder $\mathfrak{a}/\mathfrak{b}$, die Berechnung des reduzierten Ideals, das äquivalent zu $\mathfrak{a} \cdot \mathfrak{b}$ bzw. $\mathfrak{a}/\mathfrak{b}$ ist; darüberhinaus ist $\mathfrak{a} = \mathfrak{b}$ genau dann, wenn $\mathfrak{a} = \mathfrak{b}$ ist.

In Abschnitt 7.1 beschreiben wir verschiedene bekannte, effiziente Algorithmen für Ideal-Arithmetik. Wir beschreiben den Standard-Algorithmus zur Ideal-Multiplikation und -Quadrierung, und wir beschreiben zur Reduktion von Idealen den Standard-Algorithmus nach Gauß, den Algorithmus nach Rickert, sowie den rekursiven Algorithmus nach Schönhage. In einer ersten Untersuchung haben wir die Effizienz dieser Reduktions-Algorithmen verglichen. Wir haben diese Algorithmen implementiert und Laufzeiten gemessen. Da die Untersuchung, Implementierung und der Vergleich von effizienten Reduktions-Algorithmen nicht primär Gegenstand dieser Arbeit ist, stehen die angegebenen Laufzeiten unter dem entsprechenden Vorbehalt.

In Abschnitt 7.2 geben wir Laufzeiten an für die Erzeugung und Überprüfung von Signaturen für unsere Implementierung von IQ-RDSA, IQ-DSA und IQ-GQ für verschiedenen Sicherheitsparameter. Diese Laufzeiten vergleichen wir mit Laufzeiten für die Erzeugung und Überprüfung von RSA- und DSA-Signaturen, wobei wir hiefür die populäre OpenSSL-Implementierung verwendet haben. Die kryptographischen Parameter sind jeweils so gewählt, daß die Kryptoverfahren vergleichbare Sicherheitsanforderungen erfüllen. Es zeigt sich daß, die IQ-Kryptoverfahren selbst für moderate Parametergrößen mit RSA und DSA konkurrieren. Wir beschreiben abschließend, welche Optimierungen wir in unseren Implementierungen vorgenommen haben.

7.1 IQ-Arithmetik

In diesem Abschnitt beschreiben wir verschiedene effiziente Algorithmen für Arithmetik in Klassengruppen imaginär-quadratischer Zahlkörper. Die Gruppenelemente werden durch die primitiven, reduzierten Ideale dargestellt; diese Darstellung ist im Fall der Klassengruppen

imaginär-quadratischer Zahlkörper eindeutig. Wir stellen die Standard-Algorithmen zur Multiplikation und zur Quadrierung von Idealen vor, wie sie z. B. in [12] zu finden sind. Danach stellen wir verschiedene Algorithmen zur Reduktion von Idealen vor. Wir beschreiben hier den Standard-Algorithmus nach Gauß, einen Algorithmus nach Rickert, und schließlich den rekursiven Algorithmus nach Schönhage. Wir haben alle diese Algorithmen implementiert; am Ende dieses Abschnitts werden wir einen Vergleich dieser Algorithmen vornehmen, wobei wir uns auf erste Laufzeitmessungen mit unseren Implementierungen stützen. Einen detaillierter Vergleich effizienter Implementierungen wird Gegenstand zukünftiger Forschungen sein.

7.1.1 Multiplikation von Idealen

Wir haben bereits in Kapitel 2 Algorithmen zur effizienten Ideal-Arithmetik vorgestellt. Wir beschreiben hier nochmals die Algorithmen zur Multiplikation und zur Quadrierung (jeweils ohne abschließende Reduktion); diese Algorithmen wurden aus [40] entnommen.

Algorithmus 7.1 multiply

Eingabe: $\mathcal{O}_\Delta$-Ideale $\mathfrak{a}_1 = (a_1, b_1)$ und $\mathfrak{a}_2 = (a_2, b_2)$.
Ausgabe: Das $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}_3 = (a_3, b_3) = \mathfrak{a}_1 \cdot \mathfrak{a}_2$.

1: $d_1, v, w \leftarrow \mathsf{xgcd}(a_1, a_2)$
2: $a_3 \leftarrow a_1 a_2$
3: $b_3 \leftarrow v a_1 (b_2 - b_1)$
4: **if** $d_1 \neq 1$ **then**
5: $\quad d_2, v, w \leftarrow \mathsf{xgcd}(d_1, (b_1 + b_2)/2)$
6: $\quad a_3 \leftarrow a_3 / d_2^2$
7: $\quad b_3 \leftarrow (b_3 v + w(\Delta - b_1^2)/2)/d_2$
8: **end if**
9: $b_3 \leftarrow b_1 + b_3$
10: **return** (a_3, b_3)

In der Praxis ist d_1 sehr klein; insbesondere, wenn a_1 oder a_2 eine große Primzahl und $a_1 \neq a_2$ ist, dann ist praktisch immer $d_1 = 1$. Wenn andernfalls $d_1 > 1$ ist, dann kann die Berechnung von d_2, v und w in Zeile 5 i. allg. mit wenigen Schritten durchgeführt werden, und darüberhinaus werden auch d_2 und w klein sein. Vernachlässigt man einfache Operationen wie Addition und Subtraktion, sowie Multiplikation und Division mit kleinen Zahlen, so werden für diesen Algorithmus i. allg. 1 xgcd, 3 mehrfach-genaue Multiplikationen vor der Abfrage, sowie ggf. 1 xgcd mit kleinem d_1 und 2 weitere mehrfach-genaue Multiplikationen durchgeführt. Wenn $a_1, a_2, b_1, b_2 < 2^n$ ist, dann ist am Ende der Berechnung $a_3 < 2^{2n}$ und $b_3 < 2^{3n}$, falls $d_1 = 1$, und $b_3 < 2^{4n}$, falls $d > 1$.

Seien $\mathfrak{a}_1$ und $\mathfrak{a}_2$ reduzierte $\mathcal{O}_\Delta$-Ideale. Falls a priori bekannt ist, daß ein Ideal mit sehr kleinem a multipliziert werden soll, dann sollte die Eingabe so arrangiert werden, daß $a_1 < a_2$ ist. In diesem Fall wird 1 xgcd mit kleinem a_1 und 1 mehrfach-genaue Multiplikation, sowie ggf. 1 xgcd mit kleinem d_1 und 1 weitere mehrfach-genaue Multiplikation durchgeführt (wegen $|b_1| \leqslant a_1$).

Im Fall von $a = 2$ ist es nun wiederum günstiger, wenn $a_2 = 2$ ist, z. B. $\mathfrak{a}_2 = (2, 1)$ (wenn $\Delta \equiv 1 \pmod 8$, dann ist $(2, 1)$ ein $\mathcal{O}_\Delta$-Ideal). Falls $2 \nmid a_1$, dann ist $d_1 = 1$, $v = -1$ und

$w = (a_1+1)/2$, und es wird lediglich eine mehrfach-genaue Multiplikation durchgeführt. Falls $d_1 = 2$ ist, dann ist $v = 0$ und $w = 1$; falls nun $d_2 = 1$, dann ist $v = (b_1 + 3)/4$ und $w = -1$; falls $d_2 = 2$, dann ist $v = 1$ und $w = 0$; in den letzten beiden Fällen wird überhaupt keine mehrfach-genaue Multiplikation durchgeführt.

Ein anderer Sonderfall entsteht, wenn $a_2 = a_1$ und $b_2 = b_1$ ist, d. h. bei Quadrierung. Im diesem Fall setzen wir in Zeile 1 $v = 0$ und $w = 1$, und es ergibt sich der nachstehende Algorithmus:

Algorithmus 7.2 square

Eingabe: Ein $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}_1 = (a_1, b_1)$
Ausgabe: Das $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}_3 = (a_3, b_3) = \mathfrak{a}_1^2$

1: $d, v, w \leftarrow \mathsf{xgcd}(a_1, b_1)$
2: $a_3 \leftarrow (a_1/d)^2$
3: $b_3 \leftarrow w(\Delta - b_1^2)/(2d)$
4: $b_3 \leftarrow b_1 + b_3$
5: **return** (a_3, b_3)

Auch hier ist d (welches dem d_2 aus multiply entspricht) in der Praxis sehr klein; unter Vernachlässigung einfacher Operationen werden für diesen Algorithmus 1 xgcd ausgerechnet und 3 mehrfach-genaue Multiplikationen durchgeführt. Wenn $a_1, b_1 < 2^n$ ist, dann ist am Ende der Berechnung $a_3 < 2^{2n}$ und $b_3 < 2^{3n}$.

7.1.2 Reduktion von Idealen

In diesem Abschnitt beschreiben wir drei Reduktions-Algorithmen für Ideale imaginär-quadratischer Ordnungen. Diese Algorithmen wurden ursprüglich für binäre quadratische Formen formuliert. Der Zusammenhang zwischen quadratischen Formen und Idealen quadratischer Ordnungen ist z. B. in [12, Kapitel 7] detailliert dargestellt. Wir rekapitulieren nur kurz die notwendigen Fakten:

Eine *binäre quadratische Form* ist definiert durch

$$f(x, y) = ax^2 + bxy + cy^2 \quad , \quad a, c \neq 0 \ .$$

Wir schreiben auch kurz $f = (a, b, c)$. Die Diskriminante der Form ist definiert durch $\Delta(f) = b^2 - 4ac$. Die quadratische Form f heißt *positiv definit*, wenn $\Delta < 0$ und $a > 0$, *negativ definit*, wenn $\Delta < 0$ und $a < 0$, und ansonsten *indefinit*, wenn $\Delta > 0$. Seien $a, b, c \in \mathbb{Z}$, und sei f so, daß Δ kein Quadrat in $\mathbb{Z}$ ist. Sei f außerdem *primitiv*, d. h. $\mathrm{ggT}(a, b, c) = 1$ ist. Es ist nun klar, daß die primitiven, positiv definiten quadratischen Formen (a, b, c) ($\Delta < 0$, $a > 0$ und daher auch $c > 0$) den primitive, ganzen $\mathcal{O}_\Delta$-Idealen (a, b) mit $\Delta = b^2 - 4ac$ entsprechen. Wenn wir von quadratischen Formen oder einfach nur von Formen sprechen, dann meinen wir primitive, positiv definite, quadratische Formen.

Zwei Formen f und g heißen äquivalent, wenn s, t, u, v existieren mit $sv - tu = 1$, so daß $g(x, y) = f(sx + ty, ux + vy)$ ist. Wir schreiben auch $g(x, y) = f((x, y)M)$ oder kurz $g = fM$ mit

$$M = \begin{pmatrix} s & t \\ u & v \end{pmatrix} \in \mathrm{SL}(2, \mathbb{Z}) \ .$$

Für $f = (a, b, c)$ und $g = (A, B, C)$ ist dann

$$\begin{aligned} A &= as^2 + bsu + cu^2 = f(s, u) \\ B &= 2ast + b(sv + tu) + 2cuv \\ C &= at^2 + btv + cv^2 = f(t, v) \end{aligned} \tag{7.1}$$

und äquivalent dazu ist

$$\begin{aligned} a &= Av^2 - Bvu + Cu^2 = g(v, -u) \\ b &= -2Avt + B(sv + tu) - 2cus \\ c &= At^2 - Bts + Cs^2 = g(-t, s) \end{aligned} \tag{7.2}$$

Wenn f und g äquivalent sind, schreiben wir $f \sim g$, ansonsten $f \not\sim g$.

Beispiel:

$$(a, b, c) \sim (c, -b, a) \quad \text{mit} \quad M = T = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \ ; \tag{7.3}$$

$$(a, b, c) \sim (a, b + 2a, c + b + a) \quad \text{mit} \quad M = S = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \ ; \tag{7.4}$$

$$(a, b, c) \sim (a + b + c, b + 2c, c) \quad \text{mit} \quad M = S^t = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \ ; \tag{7.5}$$

$$(a, b, c) \sim (a, b + 2am, c + bm + am^2) \quad \text{mit} \quad M = S^m = \begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix} \quad \text{für} \quad m \in \mathbb{Z} \ ; \tag{7.6}$$

$$(a, b, c) \sim (a + bm + cm^2, b + 2cm, c) \quad \text{mit} \quad M = (S^t)^m = \begin{pmatrix} 1 & 0 \\ m & 1 \end{pmatrix} \quad \text{für} \quad m \in \mathbb{Z} \ . \tag{7.7}$$

Eine (positiv definite) quadratische Form $f = (a, b, c)$ heißt normal, wenn $-a < b \leqslant a$, und f heißt reduziert, wenn f normal ist und $a \leqslant c$, und wenn $a = c$, dann $b \geqslant 0$. Die reduzierte, zu f äquivalente Form bezeichnen wir mit $\overline{f}$.

Der Standard-Algorithmus nach Gauß

Der Standard-Algorithmus zur Reduktion quadratischer Formen (bzw. Ideale quadratischer Ordnungen) nach Gauß ist dem erweiterten Euklidischen Algorithmus sehr ähnlich. Auf einen Schritt mit einer Division mit Rest (Normalisierung) erfolgt eine Vertauschung. Jede Transformation von einer Form f zur äquivalenten, reduzierten Form $\overline{f}$ hat eine eindeutige Faktorisierung in eine Folge von Potenzen von S und T (siehe (7.6) bzw. (7.3)).

Um eine Form (a, b, c) oder das entsprechende $\mathcal{O}_\Delta$-Ideal $\mathfrak{a}$, dargestellt durch (a, b), zu normalisieren, müssen wir zunächst q finden, so daß $-a < b + 2aq \leqslant a$ ist. In Kapitel 2 haben wir bereits gezeigt, daß dies erreicht ist nach den Schritten

$q \leftarrow \lfloor (a - b)/2a \rfloor$, $r \leftarrow (a - b) \bmod 2a$
// $a - b = 2aq + r$, $\quad 0 \leqslant r < 2a$

$b \leftarrow a - r$

Zum selben Ergebnis führt

$q \leftarrow \lfloor b/2a \rfloor$, $r \leftarrow b \bmod 2a$
// $b = 2aq + r \ , \quad 0 \leqslant r < 2a$

if $r > a$ **then** $r \leftarrow r - 2a$, $q \leftarrow q + 1$
// $b = 2aq + r \ , \quad -a < r \leqslant a$

$b \leftarrow r$

Die dazugehörige Transformation lautet S^{-q}. Zur Normalisierung der ursprünglichen Darstellung (a, b) wären diese Schritte ausreichend. Um die ursprüngliche quadratische Form (a, b, c) zu normalisieren, muß c angepaßt werden, da die Diskriminante $\Delta = b^2 - 4ac$ invariant bleibt. c kann ausgerechnet werden durch $c \leftarrow (b^2 - \Delta)/4a$ oder, äquivalent, durch $c \leftarrow c - bq + aq^2 = c - q(b - aq)$ (siehe (7.6) mit $m = -q$). Vernachlässigt man simple Operationen wie Additionen und Subtraktionen, sowie Multiplikation und Division mit Zweierpotenzen und kleines Zahlen, so braucht man im ersten Fall eine Quadrierung und eine Division, und im zweiten Fall braucht man effektiv 2 Multiplikationen. Durch geschickte Ausnutzung des Restes r kann $(b - aq)$ auch ohne Multiplikation berechnet werden; unmittelbar vor der Zuweisung an b in der zweiten Folge von Anweisungen ist $b = 2aq + r$, d. h. $b + r = 2(b - aq)$. Somit kommt man bei der Normalisierung mit einer Division mit Rest und einer Multiplikation aus.

Algorithmus 7.3 normalizeForm

Eingabe: Eine quadratische Form (a, b, c).
Ausgabe: Die äquivalente, normale quadratische Form.

1: $q \leftarrow \lfloor (a - b)/2a \rfloor$, $r \leftarrow (a - b) \bmod 2a$
2: // $a - b = 2aq + r \ , \quad 0 \leqslant r < 2a$
3: **if** $q \neq 0$ **then**
4: $\quad t_b \leftarrow a - r$
5: $\quad t_c \leftarrow (b + t_b)/2$
6: $\quad$ // $t_c = b - aq$
7: $\quad b \leftarrow t_b$
8: $\quad c \leftarrow c - qt_c$
9: **end if**

Sei $\delta = \text{size}(b) - \text{size}(2a)$. Wenn δ sehr klein ist, (was in der Praxis sehr oft vorkommt), dann kommt man auch nur mit wenigen Additionen und Subtraktionen aus. Falls z. B. $b > 0$ ist, dann kann man anstelle des vorstehenden Algorithmus auch den folgen Algorithmus verwenden.

Algorithmus 7.4 normalizeForm1

Eingabe: Eine quadratische Form (a, b, c) mit $b \geqslant 0$.
Ausgabe: Die äquivalente, normale quadratische Form.

1: **while** $b > a$ **do**
2: $\quad b \leftarrow b - 2a$
3: $\quad c \leftarrow c - b - a$
4: **end while**

Der vollständige Reduktions-Algorithmus ist ein Folge aus Normalisierungen und Vertauschungen:

Algorithmus 7.5 reduceGauss

Eingabe:

1. Eine Diskriminante Δ;
2. ein $\mathcal{O}_\Delta$-Ideal (a, b).

Ausgabe: Das äquivalente, reduzierte Ideal.

1: $(a, b) \leftarrow$ normalize(a, b)
2: $c \leftarrow (b^2 - \Delta)/4a$
3: **while** $a > c$ **do**
4: $\quad (a, b, c) \leftarrow$ normalizeForm$(c, -b, a)$
5: **end while**
6: **if** $b < 0$ **and** $a = c$ **then** $b \leftarrow -b$
7: **return** (a, b)

Der Reduktions-Algorithmus (Algorithmus 2.6 aus Kapitel 2 oder Algorithmus 5.4.2 aus [19] ist eine Variante dieses Algorithmus, wobei dort das Vorzeichen von b in der Variablen *sgn* abgelegt wird, so daß immer $b \geqslant 0$ gegeben ist. Allen diesen Algorithmen ist gemeinsam, daß sie mit einer Division mit Rest und mit einer Multiplikation für eine Normalisierung auskommen. Die Laufzeit-Analyse zum Gauß-Algorithmus ist ähnlich der Laufzeit-Analyse zum erweiterten Euklidischen Algorithmus. Es gilt:

Theorem 7.1.1
Die Anzahl der Normalisierungs-Schritte in Algorithmus 7.5 ist höchstens $\lfloor \log_2(a/\sqrt{|\Delta|}) + 2 \rfloor$.

Theorem 7.1.2
Sei (a, b) ein $\mathcal{O}_\Delta$-Ideal mit $a, b, (b^2 - \Delta)/4a < 2^n$, dann benötigt Algorithmus 7.5 $O(n^2)$ Bitoperationen.

Zu Beweisen, siehe [7] oder [12, Abschnitt 5.6]. Die Aussagen gelten auch dann, wenn man die beiden Normalisierungs-Algorithmen normalizeForm und normalizeForm1 mittels eines Tests kombiniert.

Der Algorithmus nach Rickert

Die Idee von Rickert zur Reduktion quadratischer Formen entspricht der Idee Lehmers für den erweiterten Euklidischen Algorithmus [61]. Die Idee besteht darin, anstelle für alle Operationen mit voller Präzision zu rechnen, nur mit den führenden Bits der Operanden zu rechnen.

Wir beschreiben zunächst grob die Idee, bevor wir die näheren Details präsentieren. Sei $f = (a, b, c)$ eine positiv definite quadratische Form, und seien a, b und c jeweils n-Bit-Zahlen, wobei wir führende Nullen für zwei Operanden zulassen. Sei ℓ die Breite eines CPU-Registers, z. B. $\ell = 32$ oder $\ell = 64$, und seien a_0, b_0, c_0 die führenden ℓ Bits von a, b und c. Wir reduzieren die Form $f_0 = (a_0, b_0, c_0)$, wobei alle Operationen mit einfacher Genauigkeit durchgeführt werden und die Transformation M_0, mit $\overline{f_0} = f_0 M_0$, protokolliert werden.

Anschließend wenden wir M_0 nach (7.1) auf f an und erhalten f'. Wir werden noch sehen, daß die Einträge von M_0 durch $2^{\ell/2}$ beschränkt sind, daher sind die Produkte aus je zwei Einträgen von M noch mit einfacher Genauigkeit darstellbar, und daher müssen genau 9 Multiplikationen von mehrfach-genauen Zahlen und einfach-genauen Zahlen durchgeführt werden (gemischt-genaue Arithmetik). Falls f' nicht reduziert ist, wiederholen wir den gesamten Vorgang, ansonsten sind wir fertig.

Bei dem Algorithmus gibt es allerdings einige Feinheiten zu beachten. Im allgemeinen ist $\Delta(f) \neq \Delta(f_0)$, und f_0 kann sogar indefinit sein, und die Reduktion von indefiniten Formen unterscheidet sich etwas von der Reduktion definiter Formen. Die Anwendung von M_0 auf f führt daher möglicherweise zu weit. Beim erweiterten Euklidischen Algorithmus nach Lehmer gibt es die entsprechende Situation, wo sich der Quotient der einfach-genauen Operanden vom Quotient der mehrfach-genauen Operanden unterscheidet.

Um damit umzugehen, führen wir zunächst einen anderen Reduktions-Begriff ein:

Definition 7.1.3
Eine quadratische Form (a, b, c) *heißt* semi-reduziert*, wenn* $|b| < \min(2a, 2c)$ *ist.*

Ist eine (positiv definite) quadratische Form semi-reduziert, dann braucht man nur wenige Additionen und Subtraktionen, um die äquivalente reduzierte Form zu erhalten.

Algorithmus 7.6 reduceRickertFinal

Eingabe: Eine semi-reduzierte, positiv definite quadratische Form (a, b, c) mit $0 \leqslant |b| < \min(2a, 2c)$.
Ausgabe: Die äquivalente, reduzierte quadratische Form.

1: **if** $b < 0$ **then**
2: $b = -b$
3: $sgn = -1$
4: **else**
5: $sgn = 1$
6: **end if**
7: **if** $a > c$ **then**
8: $a \leftarrow c$, $c \leftarrow a$
9: $sgn \leftarrow -sgn$
10: **end if**
11: **if** $b > a$ **then**
12: $b \leftarrow b - 2a$
13: $c \leftarrow c - b - a$
14: **if** $a > c$ **then**
15: $a \leftarrow c$, $c \leftarrow a$
16: $sgn \leftarrow -sgn$
17: **end if**
18: **end if**
19: **if** $a = c$ **or** $b = a$ **then**
20: $b \leftarrow |b|$
21: **else if** $sgn < 0$ **then**
22: $b \leftarrow -b$
23: **end if**
24: **return** (a, b, c)

Als vollen Semi-Reduktionsschritt bezeichnen wir:

Algorithmus 7.7 fullStepRickert

Eingabe: Eine quadratische Form (a, b, c).
Ausgabe: Die äquivalente Form mit $b < \min(2a, 2c)$.

1: **if** $a \leqslant c$ **then**
2: $q \leftarrow \lfloor b/2a \rfloor$, $r \leftarrow b \bmod 2a$
3: $c \leftarrow c - q(r + b)/2$
4: $b \leftarrow r$
5: **else**
6: $q \leftarrow \lfloor b/2c \rfloor$, $r \leftarrow b \bmod 2c$
7: $a \leftarrow a - q(r + b)/2$
8: $b \leftarrow r$
9: **end if**
10: **return** (a, b, c)

Als einfachen Semi-Reduktionsschritt bezeichnen wir:

Algorithmus 7.8 simpleStepRickert

Eingabe: Eine quadratische Form (a, b, c).
Ausgabe: Die äquivalente Form nach (7.4) oder (7.5).

```
1: if a ⩽ c then
2:     c ← c − b + a
3:     b ← b − 2a
4: else
5:     a ← a − b + c
6:     b ← b − 2c
7: end if
8: return (a, b, c)
```

Wir erklären nun die Details eines Rickert-Reduktionsschrittes. Zunächst vertauschen wir nach einer Normalisierung einer quadratischen Form a und b nicht explizit, sondern wir vertauschen implizit, indem wir abwechselnd (7.6) und (7.7) anwenden. Außerdem wird in dem Algorithmus stets $b \geqslant 0$ sein; das tatsächliche Vorzeichen von b speichern wir in einer separaten Variablen *sgn* ab. Dadurch wird in (7.6) oder (7.7) immer $m \leqslant 0$ sein.

Wenn $g = (A, B, C)$ eine semi-reduzierte quadratische Form ist, die äquivalent zu $f = (a, b, c)$ ist, wobei $g = fM$ mit $M = \begin{pmatrix} s & t \\ u & v \end{pmatrix}$, dann ist

$$s > 0 \ , \quad v > 0 \ , \quad t \leqslant 0 \ , \quad u \leqslant 0 \ , \tag{7.8}$$

die Vorzeichen von s, t, u und v sind also a priori bekannt und brauchen nicht gespeichert zu werden. Wir machen nun eine Abschätzung für die Einträge von M. Mit $a, b, c < 2^n$ und $b \geqslant 0$ wird aus (7.2)

$$\begin{aligned} 2^n &> Av^2 + Bv|u| + Cu^2 \\ 2^n &> 2Av|t| + B(sv + tu) + 2C|u|s \\ 2^n &> At^2 + B|t|s + Cs^2 \end{aligned} \tag{7.9}$$

Unter der Bedingung, daß $B \geqslant 0$ ist, folgt daraus $s, |t|, |u|, v < 2^{n/2}$, und darüber hinaus ist

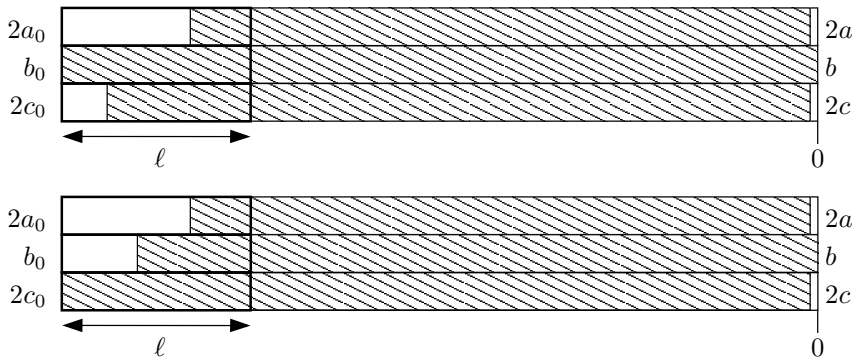
$$s^2, t^2, u^2, v^2, v|u|, v|t|, s|u|, s|t|, sv + tu < 2^n \ . \tag{7.10}$$

Seien $2a$, b und $2c$ jeweils n-Bit-Operanden, wobei maximal zwei Operanden führende Nullen aufweisen dürfen. $2a$ und $2c$ müssen nicht explizit ausgerechnet werden. Seien $2a_0$, b_0 und $2c_0$ jeweils die führenden Bits von $2a$, b bzw. $2c$. In Abbildung 7.1 haben wir zwei mögliche Situationen mit $2a < 2c < b$ und $2a < b < 2c$ dargestellt.

Wir modellieren die Situation so, als wären a, b und c durch $2^{n-\ell}$ dividiert worden. Dann sind $2a_0$, b_0 und $2c_0$ die Ganzzahligen Anteile von $2a$, b bzw. $2c$. Insbesondere haben wir

$$a = a_0 + a_1 \ , \quad b = b_0 + b_1 \ , \quad c = c_0 + c_1 \ ,$$

Abbildung 7.1: Die führenden ℓ Bits von $2a$, b und $2c$



wobei

$$0 \leqslant a_1 < \tfrac{1}{2} \ , \quad 0 \leqslant b_1 < 1 \ , \quad 0 \leqslant c_1 < \tfrac{1}{2} \ . \tag{7.11}$$

Falls wir gemischte Arithmetik mit a_0, b_0 und c_0 betreiben wollen, müssen wir aufpassen, weil a_0 und c_0 nicht mit b_0 kompatibel sind: a_0 und c_0 sind als Operanden mit $\ell - 1$ Vorkomma- und einem Nachkomma-Bit zu interpretieren, während b_0 als ein Operand mit ℓ Vorkomma-Bits zu interpretieren ist. Die Register, in denen a_0, b_0 und c_0 gespeichert sind, sind daher tatsächlich als $2a_0$, b_0 bzw. $2c_0$ zu interpretieren.

Nun wird die quadratische Form $f_0 = (a_0, b_0, c_0)$ semi-reduziert, wobei die entsprechende Transformation M_0 protokolliert wird, und man erhält $g_0 = (A_0, B_0, C_0)$. Anschließend wird M_0 nach (7.1) auf $f = (a, b, c)$ angewendet, um $g = (A, B, C)$ zu erhalten. Da $a_0, b_0, c_0 < 2^\ell$ ist, müssen wegen (7.10) lediglich 9 Multiplikationen von jeweils einem einfach-genauen und einem mehrfach-genauen Operanden durchgeführt werden.

Bei der Semi-Reduktion dürfen die Abschätzungen (7.9) und (7.10) nicht verletzt werden. Diese Abschätzungen beruhen u. a. darauf, daß $B \geqslant 0$ ist. Diese Bedingung muß auch dann eingehalten werden, wenn B nicht explizit ausgerechnet wird. Bei der Semi-Reduktion von (a_0, b_0, c_0) zu (A_0, B_0, C_0) genügt es daher nicht, wenn $B_0 \geqslant 0$ ist.

Nehmen wir beispielsweise an, daß in einem Zwischenschritt der Berechnung $A_0 \leqslant C_0$ und $q_0 = \lfloor B_0/2A_0 \rfloor$ ist. Nach Voraussetzung ist $B_0 - 2A_0q_0 \geqslant 0$, aber es wäre möglich, daß an dieser Stelle $B - 2Aq_0 < 0$ ist. Dies kann passieren, wenn q_0 zu groß ist, also wenn $q_0 > q$ für $q = \lfloor B/2A \rfloor$ ist. Aus (7.1) und den Ungleichungen (7.11) erhält man die Abschätzungen

$$\begin{aligned} 2A &< 2A_0 + v^2 + u^2 \\ B &\geqslant B_0 - (vt + su) \\ 2C &< 2C_0 + t^2 + s^2 \end{aligned}$$

Falls $A_0 \leqslant C_0$ ist, dann ist

$$q_0 = \left\lfloor \frac{B_0 - (vt + su)}{2A_0 + v^2 + u^2} \right\rfloor \leqslant \left\lfloor \frac{B}{2A} \right\rfloor \ ,$$

und entsprechend für $A_0 > C_0$. q_0 ist dann möglicherweise kleiner als notwendig, aber $B \geqslant 0$ ist gewährleistet. Ein Rickert-Reduktionschritt sieht also folgendermaßen aus:

Algorithmus 7.9 reduceRickertCore

Eingabe: Operanden $2a_0$, b_0 und $2c_0$ der Bitlänge ℓ.
Ausgabe: Die Transformation M zu einem Rickert-Reduktionsschritt.

```
 1: M ← I
 2: repeat
 3:   if 2a_0 ⩽ 2c_0 then
 4:     q_0 ← ⌊(b_0 − (vt + su))/(2a_0 + v^2 + u^2)⌋
 5:     if q_0 > 0 then
 6:       t_b ← b_0 − 2a_0 q
 7:       2c_0 ← 2c_0 − q(b_0 + t_b)
 8:       b_0 ← t_b
 9:       M ← M · S^{−q}
10:     end if
11:   else
12:     q_0 ← ⌊(b_0 − (vt + su))/(2c_0 + t^2 + s^2)⌋
13:     if q_0 > 0 then
14:       t_b ← b_0 − 2c_0 q
15:       2a_0 ← 2a_0 − q(b_0 + t_b)
16:       b_0 ← t_b
17:       M ← M · (S^t)^{−q}
18:     end if
19:   end if
20: until q_0 = 0
21: return M
```

In diesem Teil-Algorithmus werden sämtliche einfach-genauen Operationen mit ℓ-Bit-Registern durchgeführt.

Der Reduktions-Algorithmus besteht nun darin, reduceRickertCore auf führenden ℓ Bits von a, b und c anzuwenden, um die berechnete Transformation M auf (a, b, c) anzuwenden. Dieser Algorithmus wird solange wiederholt, bis (a, b, c) selber semi-reduziert ist, die abschließende Reduktion wird mittels reduceRickertFinal durchgeführt. Es gibt aber zwei Situationen, in denen kein Rickert-Reduktionsschritt gemacht werden kann:

1. $|\operatorname{size}(2a) - \operatorname{size}(2c)| \geqslant \ell$ oder $\operatorname{size}(b) - \operatorname{size}(\min(2a, 2c)) \geqslant \ell$. Dieser Fall tritt in der Praxis sehr selten auf. Wenn in Abbildung 7.1 z. B. $2a$ sehr viel kleiner wäre, als b (obere Abbildung) oder als $2c$ (untere Abbildung), dann wäre $a_0 = 0$. In diesem Fall muß ein voller Reduktionsschritt gemacht werden. Dies bedeutet in diesen Beispielen aber, daß entweder q sehr groß ist und ein substantieller Fortschritt bei der Reduktion gemacht wird (wenn b sehr viel größer als $2a$ ist), oder daß die quadratische Form im nächsten Schritt semi-reduziert ist (wenn $2c$ sehr viel größer als $2a$ ist, aber nicht b).

2. Ein Rickert-Reduktionsschritt gibt $M = I$ zurück, obwohl (a, b, c) selber noch nicht semi-reduziert ist. Das passiert, wenn in reduceRickertCore im ersten Durchlauf der Schleife $q_0 = 0$ ist. In diesem Fall ist $\operatorname{size}(b) - \operatorname{size}(\min(2a, 2c)) \leqslant 1$, und es sollte $q = 1$ sein. Hier behilft man sich mit einem einfachen Reduktionsschritt.

Der komplette Reduktions-Algorithmus nach Rickert sieht dann folgendermaßen aus:

Algorithmus 7.10 reduceRickert

1. Eine Diskriminante Δ;
2. ein $\mathcal{O}_\Delta$-Ideal (a, b).

Ausgabe: Das äquivalente, reduzierte Ideal.

if $b < 0$ **then**
  $b \leftarrow -b$
  $sgn = -1$
else
  $sgn = 1$
end if
$c \leftarrow (b^2 - \Delta)/4a$
while $b > \min(2a, 2c)$ **do**
  if $\text{size}(b) - \text{size}(\min(2a, 2c)) \geqslant \ell$ **or** $|\text{size}(2a) - \text{size}(2c)| \geqslant \ell$ **then**
    $(a, b, c) \leftarrow \mathsf{fullStepRickert}(a, b, c)$
  else if $\text{size}(b) - \text{size}(\min(2a, 2c)) \leqslant 1$ **then**
    $(a, b, c) \leftarrow \mathsf{simpleStepRickert}(a, b, c)$
  else
    // Seien a_0, b_0 und c_0 die führenden ℓ Bits von a, b bzw. c
    $M \leftarrow \mathsf{reduceRickertCore}(2a_0, b_0, 2c_0)$
    $A \leftarrow as^2 - bs|u| + cu^2$
    $B \leftarrow -2as|t| + b(sv + tu) - 2c|u|v$
    $C \leftarrow at^2 - b|t|v + cv^2$
    $(a, b, c) \leftarrow (A, B, C)$
  end if
end while
if $sgn < 0$ **then** $b \leftarrow -b$
$(a, b, c) \leftarrow \mathsf{reduceRickertFinal}(a, b, c)$
return (a, b)

Der Algorithmus nach Schönhage

Der Reduktions-Algorithmus von Schönhage wurde zuerst in [68] beschrieben. Dieser Algorithmus ist asymptotisch das schnellste Verfahren: Sei (a, b, c) eine quadratische Form mit $a, b, c < 2^n$, und sei $\mu(n)$ die Laufzeit für eine Multiplikation von zwei n-Bit-Operanden. Dann hat der Schönhage-Algorithmus eine Laufzeit von $O(\mu(n) \log n)$ Bitoperationen. Wird z. B. die asymptotisch schnelle Schönhage-Strassen-Multiplikation verwendet, dann ist $\mu(n) = O(n \log n \log\log n)$, und die Laufzeit des Schönhage-Algorithmus liegt asymptotisch deutlich unter der quadratischen Laufzeit für den Standard-Algorithmus oder den Rickert-Algorithmus.

Wie beim Reduktions-Algorithmus nach Rickert ist während des gesamten Algorithmus $b \geqslant 0$, außer ggf. ganz am Anfang und ganz am Ende. Die Grundidee des Schönhage-Algorithmus besteht darin, (a, b, c) in zwei Stufen zu reduzieren, wobei in jeder Stufe rekursiv verfahren wird. Der entscheidende Punkt bei dem Algorithmus ist, daß in jeder Stufe nur

konstant viele Multiplikationen durchgeführt werden müssen. Der Algorithmus wird auch als *monotone Reduktion* bezeichnet. Der Algorithmus ist eine Variante ähnlicher Algorithmen zur ggT-Berechnung in $\mathbb{Z}$ und $\mathbb{Z}[i]$, sowie zur Berechnung von Idealsummen in quadratischen Ordnungen, siehe [81].

In [68] wird zunächst ein Maß eingeführt, um die Größe der Operanden zu steuern.

Definition 7.1.4
Sei B_{min} eine positive, reelle Zahl. Eine Form (a, b, c) heißt minimal über B_{min}, *wenn $a, \frac{1}{2}b, c \geqslant B_{min}$ und*

$$a - b + c \geqslant B_{min}$$

oder

$$\tfrac{1}{2}b - a < B_{min} \quad \text{und} \quad \tfrac{1}{2}b - c < B_{min} \ .$$

Ist eine quadratische Form minimal über B_{min}, kann kein Reduktionsschritt mehr gemacht werden, ohne die Bedingungen $a, \frac{1}{2}b, c \geqslant B_{\mathsf{min}}$ anschließend zu verletzen.

Algorithmus 7.11 isMinimal

Eingabe: Eine quadratische Form $f = (a, b, c)$ und eine ganze Zahl k.
Ausgabe: true, falls f minimal über 2^k ist, false sonst.

1: $B_{\mathsf{min}} \leftarrow 2^k$
2: **if** $a < B_{\mathsf{min}}$ **or** $b/2 < B_{\mathsf{min}}$ **or** $c < B_{\mathsf{min}}$ **then return** false
3: **if** $a - b + c < B_{\mathsf{min}}$ **or** $b/2 - a < B_{\mathsf{min}}$ **and** $b/2 - c < B_{\mathsf{min}}$ **then return** true
4: **return** false

Bei der monotonen Reduktion vermeiden wir wiederum explizite Vertauschungen, wir vertauschen implizit durch alternierende Anwendung von (7.6) und (7.7).

Proposition 7.1.5 (Schönhage)
Sei $B_{min} > 0$, sei $f = (a, b, c)$ eine quadratische Form mit $a, \frac{1}{2}b, c \geqslant B_{min}$, und sei $g = (A, B, C)$ eine quadratische Form, die minimal über B_{min} und äquivalent zu f ist. Sei $M = \begin{pmatrix} s & t \\ u & v \end{pmatrix}$ die Transformation, so daß $f = gM$ ist. Dann gilt:

1. *g ist eindeutig bestimmt, und mit $s, v > 0$ und $t, u \geqslant 0$ ist auch M eindeutig bestimmt.*
2. *Die Einträge von M sind beschränkt durch*

$$(s+t)^2 \leqslant \frac{a}{B_{min}} \ , \quad 2(s+t)(u+v) \leqslant \frac{b}{B_{min}} \ , \quad (u+v)^2 \leqslant \frac{c}{B_{min}} \ . \tag{7.12}$$

Zum Beweis, siehe [68, Lemma 1 und 2]. Die Transformation M kann in eine eindeutige Folge von Potenzen aus S und S^t faktorisiert werden:

$$M = S^{q_1}(S^t)^{q_2} \dots S^{q_{2k-1}}(S^t)^{q_{2k}}$$

mit $q_1 \geqslant 0$, $q_2, \dots q_{2k-1} \geqslant 1$ und $q_{2k} \geqslant 0$.

Zur Abschätzung von k gilt: Es ist $M \geqslant (SS^t)^{k-1}$ und $M \geqslant (S^tS)^{k-1}$, somit ist $s \geqslant 2^{k-1}$ bzw. $v \geqslant 2^{k-1}$. Daraus folgt, daß $a \geqslant B_{\mathsf{min}} 2^{2k-2}$ bzw. $c \geqslant B_{\mathsf{min}} 2^{2k-2}$ ist, und daher ist

$$k \leqslant \left\lfloor \frac{\log_2(\min(a,c)/B_{\mathsf{min}})}{2} \right\rfloor + 1 \ . \tag{7.13}$$

Zur monotonen Reduktion einer Form wird der Schönhage-Algorithmus mit $B_{\mathsf{min}} = \frac{1}{2}$ gestartet. Ist eine positiv definite quadratische Form minimal über $\frac{1}{2}$, so braucht man nur wenige einfache Schritte, um die äquivalente reduzierte Form zu finden. Wegen $\Delta = b^2 - 4ac < 0$ ist $b^2 < 4ac$, und daraus folgt

$$a - b + c > a - 2\sqrt{ac} + c = \left(\sqrt{a} - \sqrt{c}\right)^2 \geqslant 0 \ ,$$

also ist $a - b + c \geqslant 1$. Da (a, b, c) aber nach Voraussetzung minimal über $\frac{1}{2}$ ist, muß $b - 2a < 1$ und $b - 2c < 1$ sein, also ist $b \leqslant 2a$ und $b \leqslant 2c$. Falls z. B. $a \leqslant c$ und $b > a$ ist, dann ist entweder $(a, b - 2a, c - b + a)$ oder $(a - b + c, 2a - b, a)$ die äquivalente reduzierte Form. Entsprechend verfährt man für den Fall $a > c$. Im Detail sind die abschließenden Schritte identisch mit denen beim Rickert-Algorithmus.

Algorithmus 7.12 reduceSchoenhageFinal

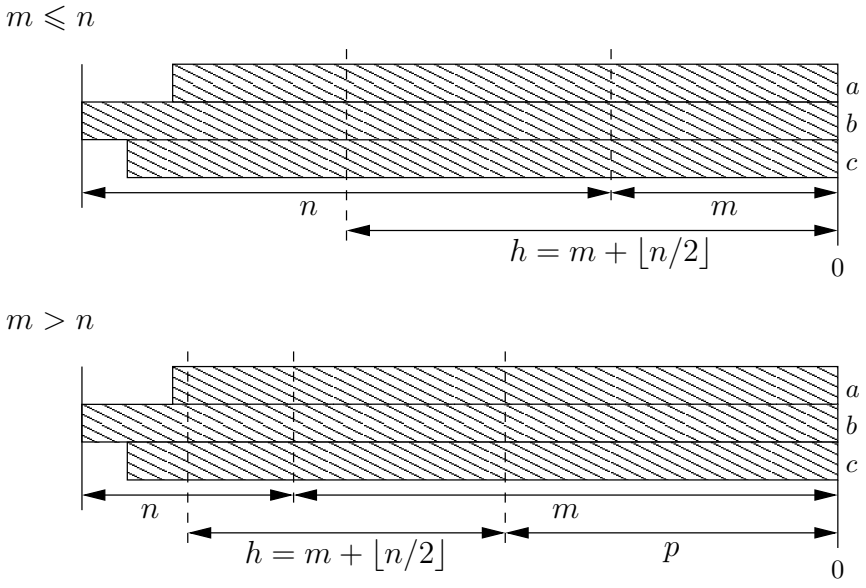
Eingabe: Eine positiv definite quadratische Form (a, b, c) mit $(a, |b|, c)$ minimal über $\frac{1}{2}$.
Ausgabe: Die äquivalente, reduzierte quadratische Form.
Siehe Algorithmus 7.6

Wir erklären nun, wie eine Normalisierung über B_{min} aussieht (dies bezeichnen wir auch als vollen Schritt). Sei $f = (a, b, c)$ mit $a, \frac{1}{2}b, c \geqslant B_{\mathsf{min}}$, und sei z. B. $a \leqslant c$, gesucht ist $g = (a, B, C) = fS^q$, wobei $B = b - 2aq$, $C = c - q(b - aq)$ und q maximal ist unter der Bedingung, daß $\frac{1}{2}B \geqslant B_{\mathsf{min}}$ und $C \geqslant B_{\mathsf{min}}$ ist. Wegen $b^2 - 4ac = B^2 - 4aC = \Delta'$ bedeutet $C \geqslant s$ dasselbe wie $B^2 \geqslant \Delta' + 4aB_{\mathsf{min}}$, somit muß $B^2 \geqslant \max(4B_{\mathsf{min}}^2, \Delta' + 4aB_{\mathsf{min}})$ sein, und es ist

$$q = \left\lfloor \frac{b - \max(2B_{\mathsf{min}}, \lceil\sqrt{\Delta' + 4aB_{\mathsf{min}}}\rceil)}{2a} \right\rfloor \ . \tag{7.14}$$

Der ganzzahlige Anteil der Wurzel läßt sich in einer Laufzeit proportional zu $\mu(n)$ berechnen [69], folglich läßt sich q in einer Laufzeit proportional zu $\mu(n)$ berechnen. Ein entsprechendes Ergebnis erhält man für $a > c$. Der komplette Algorithmus ist nachstehend abgebildet.

Abbildung 7.2: Zum Schönhage-Algorithmus



Algorithmus 7.13 fullStepSchoenhage

Eingabe: Eine quadratische Form (a, b, c) mit $a, \frac{1}{2}b, c \geqslant B_{\mathsf{min}}$.
Ausgabe: Die äquivalente quadratische Form (A, B, C) mit B minimal, so daß $A, \frac{1}{2}B, C \geqslant B_{\mathsf{min}}$ ist.

1: $B_{\mathsf{min}} \leftarrow 2^k$
2: **if** $a \leqslant c$ **then**
3: **if** $b^2 - 4a(c - B_{\mathsf{min}}) < 4B_{\mathsf{min}}^2$ **then** $r \leftarrow 2B_{\mathsf{min}}$
4: **else** $r \leftarrow \lceil \sqrt{b^2 - 4a(c - B_{\mathsf{min}})} \rceil$
5: $q \leftarrow \lfloor (b - r)/2a \rfloor$
6: $c \leftarrow c - q(b - aq)$, $b \leftarrow b - 2aq$
7: $M \leftarrow \begin{pmatrix} 1 & q \\ 0 & 1 \end{pmatrix} \cdot M$
8: **else**
9: **if** $b^2 - 4(a - B_{\mathsf{min}})c < 4B_{\mathsf{min}}^2$ **then** $r \leftarrow 2B_{\mathsf{min}}$
10: **else** $r \leftarrow \lceil \sqrt{b^2 - 4(a - B_{\mathsf{min}})c} \rceil$
11: $q \leftarrow \lfloor (b - r)/2c \rfloor$
12: $a \leftarrow a - q(b - cq)$, $b \leftarrow b - 2cq$
13: $M \leftarrow \begin{pmatrix} 1 & 0 \\ q & 1 \end{pmatrix} \cdot M$
14: **end if**
15: **return** (a, b, c), M

Wir erklären nun die Details des Schönhage-Algorithmus. Sei $B_{\mathsf{min}} = 2^m$, sei $f = (a, b, c)$ eine quadratische Form, und sei n die kleinste positive, ganze Zahl, so daß $a, b, c < 2^{m+n}$.

Wenn n sehr klein ist, dann werden einige wenige volle Schritte ausreichen, um die äquivalente, über B_{min} minimale Form zu finden. Andernfalls werden maximal zwei rekursive Schritte ausgeführt, bei denen die Operanden jeweils um etwa $n/2$ Bits verkürzt werden, siehe Abbildung 7.2 zum Beispiel für zwei mögliche Situationen. Für die Berechnung reichen im wesentlichen die führenden $2n$ Bits aus. Sind die Operanden zu Beginn einer Stufe länger, so werden die unteren Bits abgeschnitten und die monotone Reduktion für die oberen $2n$ Bits fortgesetzt, wobei die Transformation protokolliert wird; am Ende der Stufe werden die unteren Bits mit Hilfe der Transformation wieder rekonstruiert.

In jeder Stufe des Algorithmus wird zunächst die äquivalente, über 2^h minimale Form ($h = m + \lfloor n/2 \rfloor$), und daraus anschließend die äquivalente, über 2^m minimale Form berechnet, siehe Abbildung 7.2. Der Algorithmus zur vollständigen Reduktion ist in Algorithmus 7.14 wiedergegeben.

Für die Korrektheit vor Algorithmus 7.14 ist entscheidend, daß nach Zeile 30 $A, \frac{1}{2}B, C \geqslant 2^m$ gilt, wenn $p > 0$ ist. Es gelten die Abschätzungen $-2^m < \delta A, \delta B, \delta C < 2^{m+1}$, und vor Zeile 30 ist (A, B, C) minimal über $2^{m'}$. Also ist (vor Zeile 30) $A, \frac{1}{2}B, C \geqslant 2^{m'}$ und $2^p A, 2^p \frac{1}{2}B, 2^p C \geqslant 2^{m+1}$ (wenn $p > 0$ ist, ist $p + m' = m + 1$). Damit gilt dann $A, \frac{1}{2}B, C \geqslant 2^m$, wie gefordert.

Die Laufzeit dieses Algorithmus ist $O(\mu(n) \log n)$. Entscheidend hierfür ist, daß die Anzahl der Iterationen beider Schleifen durch eine Konstante beschränkt sind; abgesehen von den rekursiven Aufrufen wird in jeder Stufe daher nur eine beschränkte Anzahl von Ganzzahl-Operationen durchgeführt. Für Details zum Beweis der Korrektheit und der Laufzeit dieses Algorithmus verweisen wir auf [68].

Algorithmus 7.14 reduceSchoenhageCore

Eingabe: Eine quadratische Form $f = (a, b, c)$ und eine ganze Zahl m, wobei $a, \frac{1}{2}b, c \geqslant 2^m$ ist.

Ausgabe:

1. Die äquivalente, über 2^m minimale quadratische Form $g = (A, B, C)$;
2. die Transformation M, so daß $f = gM$.

if $\min(a, b, c) < 2^{m+2}$ **then**
  $(A, B, C) \leftarrow (a, b, c)$, $M \leftarrow I$
else
  $n \leftarrow \max(\text{size}(a), \text{size}(b), \text{size}(c)) - m$
  // $a, b, c < 2^{m+n}$
  if $m < n$ **then**
    $m' \leftarrow m$, $p \leftarrow 0$
  else
    $m' \leftarrow n$, $p \leftarrow m + 1 - n$
    Zerlege a, b und c, so daß $a = a_1 + 2^p a_2$, $b = b_1 + 2^p b_2$, $c = c_1 + 2^p c_2$
    $(a, b, c) \leftarrow (a_2, b_2, c_2)$
  end if
  $h \leftarrow m' + \lfloor n/2 \rfloor$
  if $\min(a, b, c) < 2^h$ **then**
    $(A_1, B_1, C_1) \leftarrow (a, b, c)$, $M \leftarrow I$
  else
    // $(a, b, c < 2^{2n-1}$
    $(A_1, B_1, C_1), M \leftarrow$ reduceSchoenhageCore$((a, b, c), h)$
  end if
  while not isMinimal$((A_1, B_1, C_1), 2^{m'})$ **and** $\max(A_1, B_1, C_1) \geqslant 2^h$ **do**
    $(A_1, B_1, C_1), M \leftarrow$ fullStepSchoenhage$((A_1, B_1, C_1), m')$
  end while
  if isMinimal$((A_1, B_1, C_1), 2^{m'})$ **then**
    $(A, B, C) \leftarrow (A_1, B_1, C_1)$
  else
    // $A_1, B_1, C_1 < 2^h$
    $(A, B, C), M_1 \leftarrow$ reduceSchoenhageCore$((A_1, B_1, C_1), m')$
    $M \leftarrow M_1 \cdot M$
  end if
  if $p > 0$ **then**
    $(\delta A, \delta B, \delta C) \leftarrow (a_1, b_1, c_1)M^{-1}$
    $(A, B, C) \leftarrow (2^p A + \delta A, 2^p B + \delta B, 2^p C + \delta C)$
  end if
end if
while not isMinimal$((A, B, C), 2^m)$ **do**
  $(A, B, C), M \leftarrow$ fullStepSchoenhage$((A, B, C), m)$
end while
return $(A, B, C), M$

Tabelle 7.1: Laufzeiten der Reduktions-Algorithmen (auf SPARC V8+)

Größe von $\|\Delta\|$	Gauß Alg. 7.5	Gauß Alg. 2.6	Gauß Alg. 2.6+7.4	Rickert	Schönhage
512	0.2110	0.1963	0.1883	0.1875	1.6811
724	0.3500	0.3358	0.3163	0.3234	2.6770
1024	0.5629	0.5550	0.5052	0.5302	3.9141
1448	0.9903	0.9428	0.8659	0.9622	5.8997
2048	1.6781	1.6223	1.4824	1.6625	8.9690
2896	3.1071	2.8896	2.7128	3.0404	13.5244
4096	6.0125	5.4797	5.0489	5.8221	21.2168
5793	11.1981	9.9948	9.5639	10.8874	32.9499
8192	21.8215	19.1246	18.4112	21.7855	51.6453

7.1.3 Vergleich der Reduktions-Algorithmen

Wir haben alle vorstehend beschriebenen Reduktions-Algorithmen implementiert und auf der SPARC V8+-Architektur miteinander verglichen, siehe Tabelle 7.1. Die Implementierung erfolgte mit GMP–4.0 (`http://www.swox.com/gmp/`). Die vollständige Untersuchung ist aber noch nicht abgeschlossen und war auch nicht das Ziel dieser Arbeit, die Laufzeiten in Tabelle 7.1 sind daher noch nicht als endgültig anzusehen.

Unsere Implementierung des Rickert-Algorithmus brachte auf SPARC-Hardware keinen Vorteil (bei einem Probelauf auf Pentium-Hardware war der Rickert-Algorithmus etwa doppelt so schnell wie der Gauß-Algorithmus). Erste Versuche, auf der SPARC-Hardware die einfach-genaue Arithmetik nicht mit Ganzzahl- sondern mit Fließkomma-Registern zu implementieren (der SPARC-Prozessor kennt keine Multiplikationsoperation für Ganzzahl-Register, sondern nur für Fließkomma-Register), waren bisher noch erfolglos. Den Grund für dieses Verhalten haben wir nicht näher untersucht, dies bleibt Gegenstand zukünftiger Untersuchungen.

Für die Implementierung der IQ-Kryptoverfahren auf SPARC-Hardware haben wir daher die Kombination von Algorithmus 2.6 mit Algorithmus 7.4 verwendet.

7.1.4 Asymptotische Laufzeit für IQ-Arithmetik

Wie in Kapitel 2 und zu Beginn dieses Kapitels bereits gesagt, stellen wir Gruppenelemente der Klassengruppe durch die reduzierten Ideale der entsprechenden imaginär-quadratischen Ordnung dar. Eine Gruppenoperation in der Klassengruppe ist daher effektiv eine Ideal-Multiplikation mit anschließender Reduktion des Ergebnisses. Wir geben in diesem Abschnitt asymptotische Laufzeiten für diese Ausführung der Gruppenoperation an.

Seien $\mathfrak{a}_1 = (a_1, b_1)$ und $\mathfrak{a}_2 = (a_2, b_2)$ $\mathcal{O}_\Delta$-Ideale mit $a_1, b_1, a_2, b_2 < 2^n$ für eine positive ganze Zahl n. Die asymptotischen Laufzeiten für die Algorithmen 7.1 und 7.2 (multiply und square) hängen davon ab, welche Algorithmen für die Ganzzahl-Multiplikation und für xgcd verwendet werden. Die Standard-Algorithmen zur Ganzzahl-Multiplikation und zur Berechnung von größten gemeinsamen Teilern mit Darstellung haben mit Operanden der Größe

$O(n)$ eine asymptotische Laufzeit in der Größenordnung von $O(n^2)$ (siehe z. B. [11], [19] oder [43]). Mit diesen Algorithmen haben die Algorithmen 7.1 und 7.2 ebenfalls eine asymptotische Laufzeit in der Größenordnung von $O(n^2)$. Die Laufzeit für den Reduktions-Algorithmus nach Gauß hat die Größenordnung $O(n^2)$. Eine Analyse des Rickert-Algorithmus zeigt, daß dieser Algorithmus ebenfalls eine Laufzeit in der Größenordnung $O(n^2)$ hat. Verwendet man für die Reduktion den Gauß- oder den Rickert-Algorithmus, dann ist die Laufzeit für eine Gruppenoperation also in der Größenordnung von $O(n^2)$. Da wir voraussetzen, daß wir Gruppenelemente durch die reduzierten Ideale darstellen, ist $n \leqslant \log_2 \sqrt{|\Delta|/3}$ (Proposition 2.1.9), und daher ist die Laufzeit einer Gruppenoperation dann in der Größenordnung von $O(\log^2 |\Delta|)$.

Ein asymptotisch besseres Ergebnis erhält man, wenn in den Algorithmen 7.1 und 7.2 für die Ganzzahl-Multiplikation der Schönhage-Strassen-Algorithmus [70] und für `xgcd` der Schönhage-Algorithmus zur Berechnung größter gemeinsamer Teiler mit Darstellung verwendet wird (siehe z. B. [81]). Dann haben die Algorithmen 7.1 und 7.2 eine asymptotische Laufzeit in der Größenordnung von $O(n \log^2 n \log\log n)$. Dies ist auch die asymptotische Laufzeit des Schönhage-Algorithmus zur Ideal-Reduktion (mit Schönhage-Strassen-Multiplikation). Unter der Voraussetzung, daß nur reduzierte Ideale als Eingaben verwendet werden, ist die asymptotische Laufzeit einer Gruppenoperation dann in der Größenordnung von $O(\log |\Delta| \log^2 \log |\Delta| \log\log\log |\Delta|)$.

Wie man aus Tabelle 7.1 sehen kann, ist der Schönhage-Algorithmus zur Ideal-Reduktion für Eingaben in Praxisrelevanter Größenordnung vergleichsweise ineffizient; der Vorteil des Schönhage-Algorithmus kommt erst für sehr viel größere Eingaben zur Geltung. Dasselbe gilt auch für den Schönhage-Algorithmus zur Berechnung größter gemeinsamer Teiler mit Darstellung; dieser Algorithmus ist eine Variante des Reduktions-Algorithmus. Schließlich ist auch der Schönhage-Strassen-Algorithmus zur Ganzzahl-Multiplikation erst für sehr viel größere Operanden schneller, als der einfache Standard-Algorithmus oder der Karatsuba-Algorithmus (siehe z. B. [43]). Für die praktische Anwendung wird man daher die Algorithmen mit quadratischer Laufzeit verwenden, d. h. wir unterstellen in der Praxis für eine Gruppenoperation eine Laufzeit in der Größenordnung von $O(\log^2 |\Delta|)$.

7.2 Laufzeiten für die IQ-Signaturverfahren

In diesem Abschnitt vergleichen wir die Effizienz der drei IQ-Signaturverfahren IQ-RDSA, IQ-DSA und IQ-GQ mit RSA und DSA. Alle IQ-Kryptoverfahren in diesem Abschnitt wurden mit der Bibliothek GMP–4.0 (`http://www.swox.com/gmp/`) für mehrfach-genaue Arithmetik implementiert. Die GMP-Bibliothek, wie auch unsere Programme wurden mit dem GCC–3.0.3 C-Compiler für SPARC V8+-Architektur übersetzt. Auf die verwendeten Optimierungen gehen wir in Abschnitt 7.2.1 ein. Für RSA und DSA haben wir auf die Implementierung in OpenSSL–0.9.6c (`http://www.openssl.org/`) zurückgegriffen. Alle Laufzeiten wurden auf einer SunBlade 100 (500 MHz) gemessen. Die Laufzeiten wurde über jeweils 10000 Durchläufe gemittelt.

Eine Übersicht über die mittleren Laufzeiten der IQ-Signaturverfahren sind in Tabelle 7.2 zusammengestellt und in den Abbildungen 7.4 und 7.5 graphisch dargestellt, die Vergleichswerte für RSA und DSA sind in Tabelle 7.3 dargestellt. In den Abbildungen 7.6 und 7.7 haben wir die mittleren Laufzeiten von IQ-RDSA mit denen von RSA und DSA verglichen.

Tabelle 7.2: Mittlere Laufzeiten für IQ-RDSA, IQ-DSA und IQ-GQ in ms

t	Größe von $\|\Delta\|$	IQ-RDSA		IQ-DSA		IQ-GQ	
		Sig.	Ver.	Sig.	Ver.	Sig.	Ver.
72	671	61.20	108.57	62.15	206.07	139.06	93.74
76	732	72.40	128.04	73.13	243.78	163.21	109.75
80	795	80.16	145.70	85.58	282.48	184.19	125.25
84	861	97.17	170.94	99.15	333.96	219.64	146.79
88	929	110.35	194.56	114.74	376.11	253.29	168.16
92	999	125.31	225.34	132.08	430.62	290.48	192.99
96	1072	138.89	251.88	147.00	489.18	321.15	215.40
100	1146	161.49	284.75	171.42	558.80	372.91	247.74

Die Parametergrößen für RSA und DSA wurden gemäß Tabelle 1 aus [47] zu dem Sicherheitsparameter t entnommen (vgl. Abbildung 7.3). t wurde in den Tabellen 7.2 und 7.3 gemäß dieser Tabelle 1 „realistisch“ ausgewählt in dem Sinn, daß die dazugehörigen Parameter in der Praxis aktuell oder in näherer Zukunft relevant sind (siehe dazu auch die Bemerkungen in [47, Abschnitt 4.3.2]).

Ein Vergleich der Tabellen zeigt, daß die Signaturerzeugung bereits für relevante Sicherheitsparameter bei jedem der IQ-Kryptoverfahren früher oder später schneller ist, als beim RSA-Signaturverfahren. Bei der Verifikation bleibt das RSA-Verfahren (hier mit öffentlichem Exponenten $e = 2^{16}+1 = 65537$) selbst für größere Sicherheitsparameter am schnellsten, dennoch nimmt das Verhältnis der Laufzeiten für eine RSA-Verifikation und die Verifikation mit einem der IQ-Kryptoverfahren erkennbar ab. Beim Vergleich von (traditionellem) DSA mit den IQ-Kryptoverfahren wird deutlich, daß DSA für alle Sicherheitsparameter zwar schneller sowohl bei der Signaturerzeugung, als auch bei der Verifikation ist, aber IQ-RDSA reicht zumindest bei der Signatur-Erzeugung beinahe an DSA heran, und es ist zu erwarten, daß bereits für nur geringfügig größere Sicherheitsparameter IQ-RDSA bei der Signaturerzeugung schneller ist, als DSA.

Dabei enthält die IQ-Arithmetik möglicherweise noch großes Potential für Verbesserungen, insbesondere da auf diesem Gebiet bisher vergleichsweise wenig Forschung betrieben wurde. Ein Möglichkeit besteht in einer verbesserten Implementierung des Reduktions-Algorithmus nach Rickert, und auch unsere Implementierung des Schönhage-Algorithmus ist noch nicht optimal. Alternativ dazu muß auch untersucht werden, wie sich die Verwendung von Shanks' Algorithmen NUCOMP und NUDUPL auf die Effizienz auswirkt, bei denen Ideal-Multiplikation bzw. -Quadrierung derart mit der Reduktion kombiniert wird, daß die Operanden für die Zwischenergebnisse deutlich kleiner sind, als bei den Algorithmen multiply und square.

Tabelle 7.3: Mittlere Laufzeiten für DSA und RSA ($e = 65537$) in ms

t	Größe von p bzw. n	DSA		RSA	
		Sig.	Ver.	Sig.	Ver.
72	1028	19.84	25.07	34.79	1.91
76	1250	26.14	32.49	55.40	2.67
80	1500	35.97	43.91	89.19	3.57
84	1771	47.82	58.02	137.38	4.88
88	2054	68.69	85.93	220.84	6.73
92	2362	84.03	103.70	311.54	8.45
96	2768	115.08	140.37	493.60	11.44
100	3138	146.59	178.64	705.45	14.55

Abbildung 7.3: Vergleich der Parametergrößen von IQ-Kryptoverfahren und von „traditionellen“ Kryptoverfahren

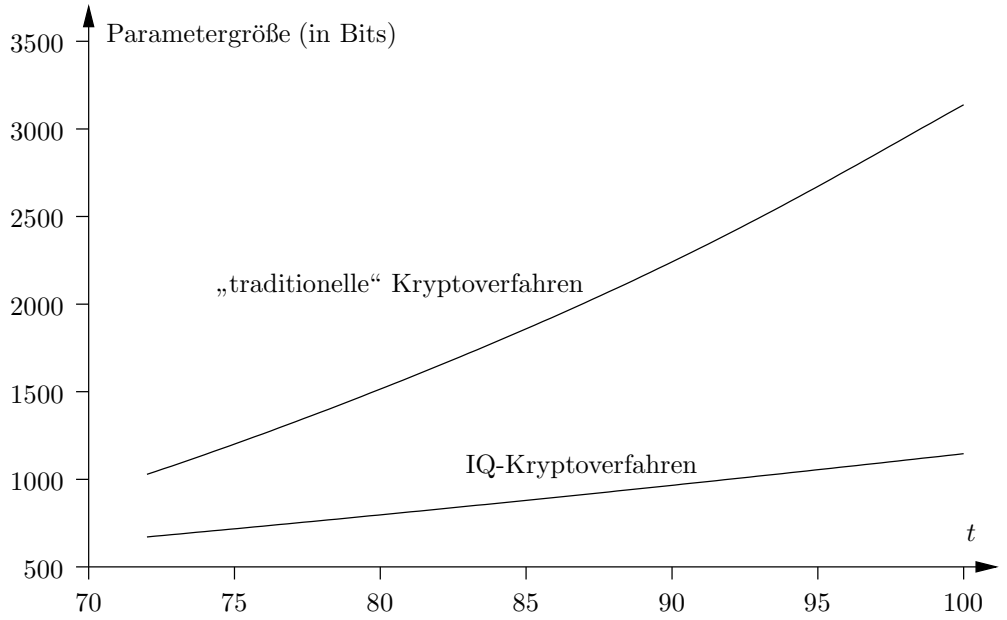


Abbildung 7.4: Vergleich der mittleren Laufzeiten der IQ-Signaturverfahren bei Erzeugung einer Signatur

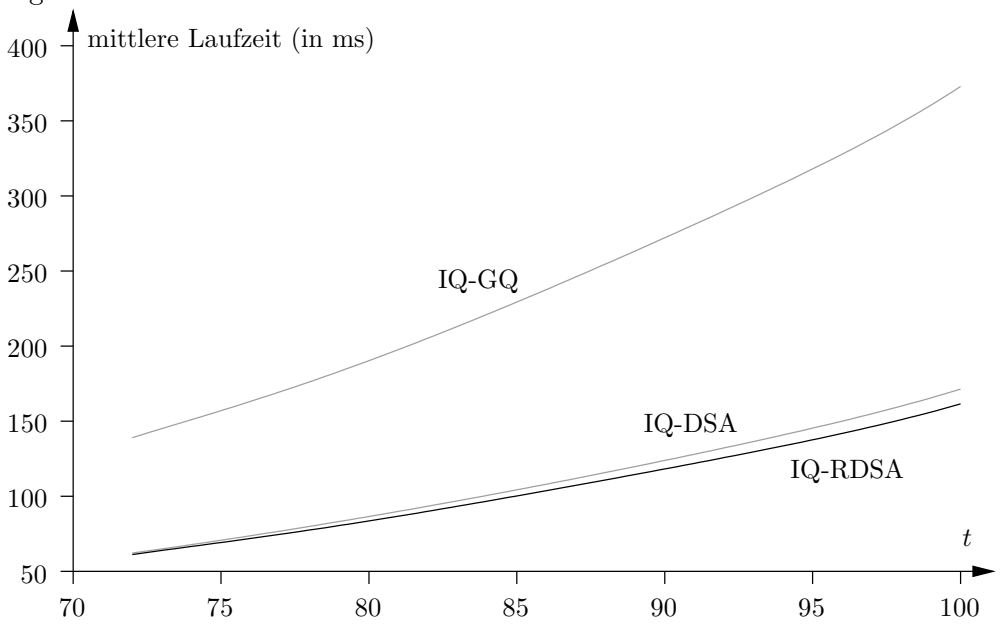


Abbildung 7.5: Vergleich der mittleren Laufzeiten der IQ-Signaturverfahren bei Verifikation einer Signatur

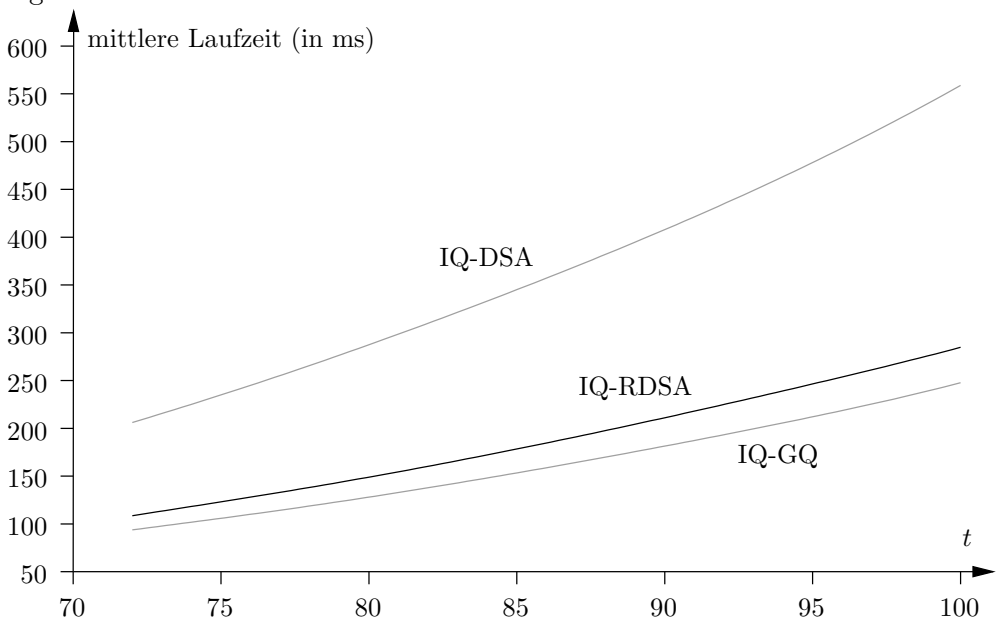


Abbildung 7.6: Vergleich der mittleren Laufzeiten von IQ-RDSA mit RSA und DSA bei Erzeugung einer Signatur

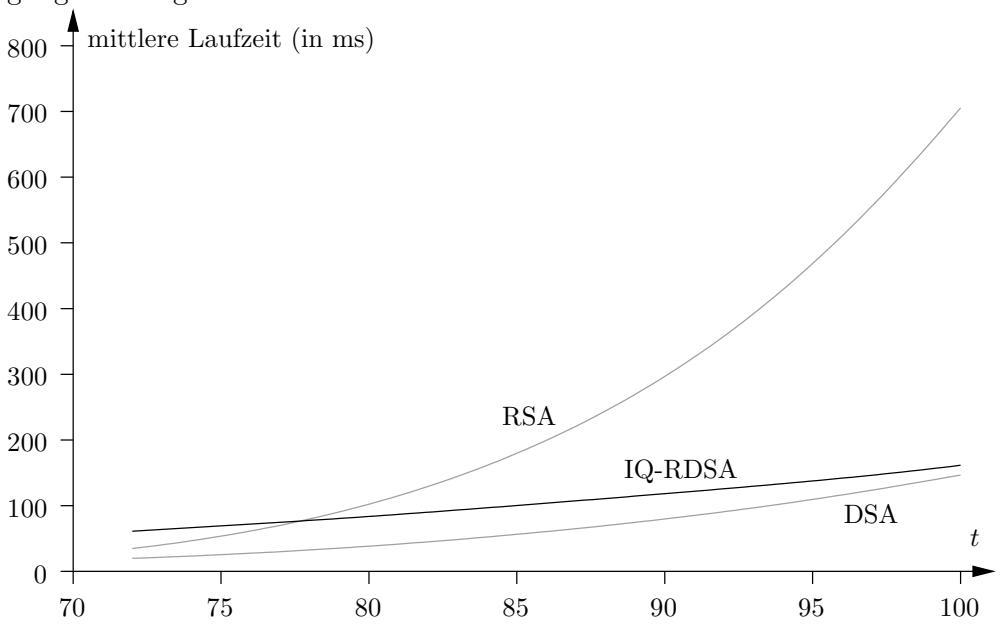
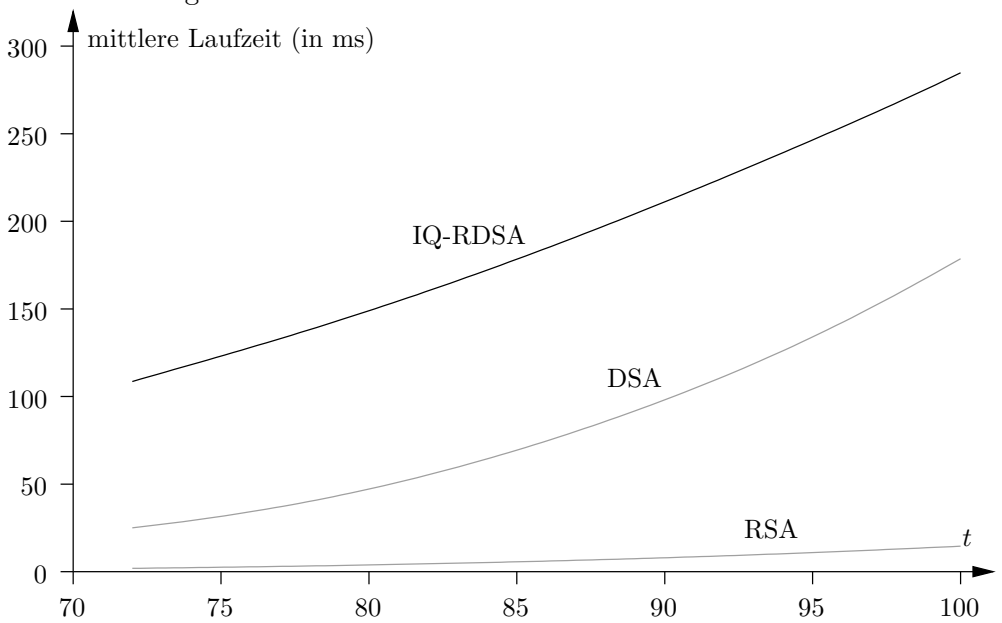


Abbildung 7.7: Vergleich der mittleren Laufzeiten von IQ-RDSA mir RSA und DSA bei Verifikation einer Signatur



7.2.1 Optimierungen für die IQ-Signaturverfahren

Wir beschreiben nun, welche Optimierungen wir für unsere Implementierung der IQ-Kryptoverfahren vorgenommen haben:

IQ-RDSA

Die beiden Exponentiationen bei der Erzeugung der Signatur werden mit der festen Basis $\mathfrak{g}$ durchgeführt. Wir haben dafür die sog. *wNAF-based interleaving Exponentiation* [53, Abschnitt 3.2] verwendet. Dies ist ein generischer Algorithmus zur effizienten Berechnung eines Potenzproduktes mit m Basen $\mathfrak{g}_i$

$$\prod_{0 \leqslant i < m} \mathfrak{g}_i^{e_i} \ .$$

Die Idee besteht nun darin (siehe z. B. auch [53, Abschnitt 5]), die Potenz $\mathfrak{g}^e$ parallel zu berechnen, indem man $\mathfrak{g}_i = \overline{\mathfrak{g}^{2^{ib}}}$ für $b = \lceil \mathrm{size}(e)/m \rceil$ ausrechnet und e in m Stücke der Länge b zerlegt, so daß $e = \sum_{0 \leqslant i < m} 2^{ib} e_i$ ist. Dann ist $\mathfrak{g}^e \sim \prod_{0 \leqslant i < m} \mathfrak{g}_i^{e_i}$. Wichtig ist, daß $\mathrm{size}(e) \leqslant b \cdot m$ ist.

In unserer Implementierung haben wir $m = 4$ gewählt, und für die wNAF-Darstellung der Exponenten haben wir $w = 3$ gewählt, so daß insgesamt $4 \cdot 2^{3-1} = 16$ vorberechnete Werte gespeichert werden müssen. Damit der Algorithmus auch für die Berechnung von $\mathfrak{l} = \overline{\mathfrak{g}^\ell}$ angewendet werden kann, muß $\ell < 2^{4b}$ sein, daher wurde q so ausgewählt, daß $2^{2t-1} < q < 2^{2t}$, in diesem Fall ist $\ell < 2^{2t+1}$, und daher haben wir $b = \lceil (2t+1)/4 \rceil$ gewählt.

Alternativ könnte man auch $b = \lceil 2t/4 \rceil$ wählen und für den Fall $\ell \geqslant 2^{4b}$ eine einfache Sonderbehandlung vorsehen, z. B. $\ell' \leftarrow \lfloor \ell/2 \rfloor$, $\ell'' \leftarrow \ell \bmod 2$ und $\mathfrak{l} \leftarrow \mathsf{reduce}(\Delta, (\mathfrak{g}^{\ell'})^2 \cdot \mathfrak{g}^{\ell''})$, wobei $\mathfrak{g}^{\ell'}$ parallel ausgerechnet wird.

Im Fall der Verifikation der Signatur muß das Produkt aus drei Potenzen ausgerechnet werden. Hierfür haben wir ebenfalls die wNAF-based interleaving Exponentiation verwendet, wobei wir hier $w = 4$ gewählt haben.

IQ-DSA

Die Exponentiation bei der Erzeugung der Signatur wird mit der festen Basis $\mathfrak{g}$ durchgeführt. Auch hier können wir $\mathfrak{g}^k$ als Potenzprodukt parallel ausrechnen, allerdings ist zu beachten, daß k wegen $k < 2^{5t}$ die $2\frac{1}{2}$-fache Länge im Vergleich zu IQ-RDSA hat. Wir haben daher $m = 5$ und $w = 4$ gewählt, so daß $5 \cdot 2^{4-1} = 40$ vorberechnete Werte gespeichert werden müssen.

Im Fall der Verifikation der Signatur muß das Produkt aus zwei Potenzen ausgerechnet werden. Hierfür haben wir ebenfalls die wNAF-based interleaving Exponentiation verwendet, wobei wir hier ebenfalls $w = 4$ gewählt haben.

IQ-GQ

Bei der Erzeugung einer Signatur müssen zwei Potenzen ausgerechnet und ein Zufallsideal gewählt werden. Die Berechnung von $\overline{\mathfrak{a}^s}$ kann wiederum parallel mit entsprechend vorberechneten Werten durchgeführt werden. Hierfür haben wir $m = 4$ und $w = 3$ gewählt, so daß 16

Ideale vorberechnet werden müssen. Zur effizienten Auswahl des Zufallsideals $\mathfrak{k}$ kann man entweder den Algorithmus 3.10 (randomIdeal) mit relativ kleinem B_p verwenden (z. B. $B_p = 2^{16}$, $B_e = 2^{2t}$ und $B_k = 1$, wobei t der Sicherheitsparameter ist, so daß man die relativ aufwändige Suche nach geeigneten Primzahlen mit einfach-genauer Arithmetik durchführen kann), oder man berechnet parallel $\mathfrak{k}$ als Potenz mit zufälligem Exponenten von einem Element, zu dem man die entsprechenden Vorberechnungen gemacht hat.

Für die Berechnung der Potenz $\mathfrak{r} = \overline{\mathfrak{k}^n}$ haben wir eine einfache wNAF-based interleaving Exponentiation verwendet ($m = 1$), wobei wir $w = 4$ gewählt haben.

Bei der Verifikation der Signatur muß das Produkt aus zwei Potenzen ausgerechnet werden. Hierfür haben wir ebenfalls die wNAF-based interleaving Exponentiation verwendet, wobei wir auch hier wieder $w = 4$ gewählt haben.

Anhang A

ASN1-Notation der IQ-Kryptoverfahren

In diesem Anhang geben wir die ASN.1-Beschreibung unserer IQ-Signaturverfahren wieder, wie sie in FlexiProvider implementiert sind. Die ASN.1-Codierung wurde freundlicherweise von Herrn R.-P. Weinmann aufbereitet.

```
-- TUD IQCrypto ASN.1 module
-- Version 0.5
-- Author: Ralf-Philipp Weinmann

TUD-IQCrypto {
    iso(1) identified-organization(3) dod(6) internet(1) private(4)
    enterprises(1) tu-darmstadt-informatik(8301) cdc(3) modules(3)
    iqcrypto(1) version(1)
}


DEFINITIONS EXPLICIT TAGS ::=
BEGIN

-- EXPORTS ALL
-- IMPORTS NONE


tud-cdc OBJECT IDENTIFIER ::= {
    iso(1) identified-organization(3) dod(6) internet(1) private(4)
    enterprises(1) tu-darmstadt-informatik(8301) cdc(3)
}


--
-- Object identifier for the IQ-cryptography algorithms hierarchy
--

id-IQCrypto-Type OBJECT IDENTIFIER { tud-cdc algorithms(1) IQCrypto(1) }
```

```
--
-- Quadratic ideals in the class group are encoded using the
-- IdealToOctets/OctetsToIdeal conversion algorithms.
--

Quadratic-Ideal::= OCTET STRING

IQ-SecurityLevel::= INTEGER(4..1024)


--
-- OIDs for the actual algorithms
--

IQDSA-with-SHA1 OBJECT IDENTIFIER { id-IQCrypto-Type 2 }
IQDSA-with-RIPEMD160 { id-IQCrypto-Type 3 }
IQGQ-with-SHA1 { id-IQCrypto-Type 5 }
IQGQ-with-RIPEMD160 { id-IQCrypto-Type 6 }
IQRDSA-with-SHA1 { id-IQCrypto-Type 8 }
IQRDSA-with-RIPEMD160 { id-IQCrypto-Type 9 }

ALGORITHM-IDENTIFIER::= TYPE-IDENTIFIER

AlgorithmIdentifier {ALGORITHM-IDENTIFIER:InfoObjectSet } ::= SEQUENCE {
    algorithm   ALGORITHM-IDENTIFIER.&id({InfoObjectSet}),
    parameters  ALGORITHM-IDENTIFIER.&Type({InfoObjectSet}{@algorithm}) OPTIONAL
}


--
-- Type identifier definitions for the IQ Cryptography OIDs
--

IQCrypto-Algorithms ALGORITHM-IDENTIFIER ::= {
    { IQDSA-DomainParameters  IDENTIFIED by IQDSA-with-SHA1 }       |
    { IQDSA-DomainParameters  IDENTIFIED by IQDSA-with-RIPEMD160 }  |
    { IQGQ-DomainParameters   IDENTIFIED by IQGQ-with-SHA1 }        |
    { IQGQ-DomainParameters   IDENTIFIED by IQGQ-with-RIPEMD160 }   |
    { IQRDSA-DomainParameters IDENTIFIED by IQRDSA-with-SHA1 }      |
    { IQRDSA-DomainParameters IDENTIFIED by IQRDSA-with-RIPEMD160 }
}


--
-- Factors of the discriminant of the class group we operate in.
-- The product of these factors has to be 3 modulo 4.
--

FactorizedDiscriminant::= SEQUENCE OF INTEGER
```

```
--
-- Domain parameter definitions
--

--
-- IQDSA domain parameter definition
--
IQDSA-DomainParameters::= SEQUENCE {
    version          INTEGER { iqdsaParamVers1(1) } (iqdsaParamVers1),
    t                IQ-SecurityLevel,
    discriminant     FactorizedDiscriminant,
    G                Quadratic-Ideal
}


--
-- IQGQ domain parameter definition
--
IQGQ-DomainParameters::= SEQUENCE {
    version          INTEGER { iqgqParamVers1(1) } (iqgqParamVers1),
    t                IQ-SecurityLevel,
    discriminant     FactorizedDiscriminant,
    n                INTEGER
}


--
-- IQRDSA domain parameter definition
--
IQDSA-DomainParameters::= SEQUENCE {
    version          INTEGER { iqrdsaParamVers1(1) } (iqrdsaParamVers1),
    t                IQ-SecurityLevel,
    discriminant     FactorizedDiscriminant,
    G                Quadratic-Ideal,
    q                INTEGER
}


--
-- Public and private key definitions
--

--
-- IQ-DSA key pair definition
--
IQDSAPublicKey::= SEQUENCE {
    algorithm        AlgorithmIdentifier {{IQDSA-with-SHA1}, {IQDSA-with-RIPEMD160}},
    version          INTEGER,
    A                Quadratic-Ideal
}

IQDSAPrivateKey::= SEQUENCE {
    algorithm        AlgorithmIdentifier {{IQDSA-with-SHA1}, {IQDSA-with-RIPEMD160}},
```

```
    version          INTEGER,
    params           IQDSA-DomainParameters,
    a                INTEGER
}


--
-- IQ-GQ key pair definition
--
IQGQPublicKey::= SEQUENCE {
    algorithm        AlgorithmIdentifier {{IQGQ-with-SHA1}, {IQGQ-with-RIPEMD160}},
    version          INTEGER,
    N                Quadratic-Ideal
}

IQGQPrivateKey::= SEQUENCE {
    algorithm        AlgorithmIdentifier {{IQGQ-with-SHA1}, {IQGQ-with-RIPEMD160}},
    version          INTEGER,
    A                Quadratic-Ideal
}


--
-- IQ-RDSA key pair definition
--
IQRDSAPublicKey::= SEQUENCE {
    algorithm        AlgorithmIdentifier {{IQRDSA-with-SHA1}, {IQRDSA-with-RIPEMD160}},
    version          INTEGER,
    A                Quadratic-Ideal
}

IQRDSAPrivateKey::= SEQUENCE {
    algorithm        AlgorithmIdentifier {{IQRDSA-with-SHA1}, {IQRDSA-with-RIPEMD160}},
    version          INTEGER,
    a                INTEGER
}


--
-- Signature definitions
--

--
-- Value of an IQDSA signature computed with either SHA-1 or RIPEMD-160
--
IQDSA-Sig-Value::= SEQUENCE {
    s                INTEGER,
    R                Quadratic-Ideal
}

--
-- Value of an IQGQ signature computed with either SHA-1 or RIPEMD-160
--
```

```
IQGQ-Sig-Value::= SEQUENCE {
    s                INTEGER,
    S                Quadratic-Ideal
}

--
-- Value of an IQRDSA signature computed with either SHA-1 or RIPEMD-160
--
IQRDSA-Sig-Value::= SEQUENCE {
    s                INTEGER,
    R                Quadratic-Ideal,
    L                Quadratic-Ideal
}



END -- TUD IQCrypto ASN.1 module
```

Literaturverzeichnis

[1] Abdalla, M., Bellare, M. und Rogaway, P. DHIES: An encryption scheme based on the Diffie-Hellman Problem. Aus *Topics in Cryptology — CT-RSA 01* (2001), D. Naccache, Hrsg., Bd. 2020 aus *Lecture Notes in Computer Science*, Springer-Verlag, S. 143–158. Available from `http://www-cse.ucsd.edu/users/mihir/papers/dhies.html`.

[2] ANSI X9.62. *Public Key Cryptography for the Financial Services Industry: the Elliptic Curve Digital Signature Algorithm (ECDSA).* American Bankers Association, 1999.

[3] ANSI X9.63. *Public Key Cryptography for the Financial Services Industry: Key Agreement and Transport Using Elliptic Curve Cryptography.* American Bankers Association, 1999. Working Draft.

[4] Bach, E. Explicit bounds for primality testing and related problems. *Mathematics of Computation 55*, 191 (1990), 355–380.

[5] Balcazar, J. L., Diaz, J. und Gabarró, J. *Structural Complexity I*, 2. Aufl. Texts in Theoretical Computer Science. Springer–Verlag, 1995.

[6] Bellare, M. und Rogaway, P. Random Oracles are Practical: a Paradigm for Designing Efficient Protocols. Aus *1st ACM Conference on Computer and Communications Security* (1993), S. 62–73.

[7] Biehl, I. und Buchmann, J. An analysis of the reduction algorithms for binary quadratic forms. Aus *Voronoi's Impact on Modern Science, Kyiv, Ukraine 1998* (1999), P. Engel und H. M. Syta, Hrsg., National Academy of Sciences of Ukraine, S. 71–98.

[8] Biehl, I., Buchmann, J., Hamdy, S. und Meyer, A. A Signature Scheme Based on the Intractability of Extracting Roots. *Designs, Codes and Cryptography* (to appear).

[9] Borevič, S. I. und Šafarevič, I. R. *Number theory.* Academic Press, 1966.

[10] Brickell, E., Pointcheval, D., Vaudenay, S. und Yung, M. Design Validations for Discrete Logarithm Based Signature Schemes. Aus *Practice and Theory in Public Key Cryptography, PKC 2000* (2000), H. Imai und Y. Zheng, Hrsg., Bd. 1751 aus *Lecture Notes in Computer Science*, Springer-Verlag, S. 276–292.

[11] Buchmann, J. *Einführung in die Kryptographie.* Springer–Verlag, 1999.

[12] Buchmann, J. Algorithms for binary quadratic forms, 2001. Manuscript. `http://www.cdc.informatik.tu-darmstadt.de/~buchmann/AlgorithmsForQuadraticForms.ps`.

[13] Buchmann, J. und Hamdy, S. A Survey on IQ Cryptography. Aus *Proceedings of Public Key Cryptography and Computational Number Theory, 2000* (2001), K. Alster, J. Urbanowicz, und H. C. Williams, Hrsg., deGruyter, S. 1–15.

[14] Buchmann, J. und Williams, H. C. A key-exchange system based on imaginary quadratic fields. *Journal of Cryptology 1*, 3 (1988), 107–118.

[15] Buell, D. A. The Expectation of Success Using a Monte Carlo Factoring Method – Some Statistics on Quadratic Class Numbers. *Mathematics of Computation 43*, 167 (1984), 313–327.

[16] Buell, D. A. *Binary Quadratic Forms: Classical Theory and Modern Computations*. Springer–Verlag, 1989.

[17] Canetti, R., Goldreich, O. und Halevi, S. The random oracle methodology, rivisited. Aus *13th Annual ACM Symposium on Theory of Computing* (1998), ACM Press, S. 209–218.

[18] Cavallar, S., Dodson, B., Lenstra, A. K., Lioen, W., Montgomery, P. L., Murphy, B., te Riele, H., Aardal, K., Gilchrist, J., Guillerm, G., Leyland, P., Marchand, J., Morain, F., Muffett, A., Putnam, C., Putnam, C. und Zimmermann, P. Factorization of a 512-Bit RSA Modulus. Aus *Advances in Cryptology – EUROCRYPT 2000* (2000), B. Preneel, Hrsg., Bd. 1807 aus *Lecture Notes in Computer Science*, Springer-Verlag, S. 1–18.

[19] Cohen, H. *A Course in Computational Algebraic Number Theory*, Bd. 138 aus *Graduate Texts in Mathematics*. Springer–Verlag, 1995.

[20] Cohen, H. und Lenstra, Jr., H. W. Heuristics on class groups. Aus *Number Theory, New York 1982*, D. V. Chudnovski, G. V. Chudnovski, H. Cohn, und M. B. Nathanson, Hrsg., Bd. 1052 aus *Lecture Notes in Mathematics*. Springer-Verlag, 1984, S. 26–36.

[21] Cohen, H. und Lenstra, Jr., H. W. Heuristics on class groups of number fields. Aus *Number Theory, Noordwijkerhout 1983*, H. Jager, Hrsg., Bd. 1068 aus *Lecture Notes in Mathematics*. Springer-Verlag, 1984, S. 33–62.

[22] Cox, D. A. *Primes of the form $x^2 + ny^2$*. John Wiley & Sons, 1989.

[23] Cramer, R. und Shoup, V. A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack. Aus *Advances in Cryptology – CRYPTO '98* (1998), H. Krawczyk, Hrsg., Bd. 1462 aus *Lecture Notes in Computer Science*, Springer-Verlag, S. 13–25.

[24] Diffie, W. und Hellman, M. E. New Directions in Cryptography. *IEEE Transactions on Information Theory 22*, 6 (1976), 644–654.

[25] Düllmann, S. *Ein Algorithmus zur Bestimmung der Klassengruppe positiv definiter binärer quadratischer Formen*. Dissertation, Universität des Saarlandes, Saarbrücken, Germany, 1991. German.

[26] FIPS 197. *Specifications for the Advanced Encryption Standard (AES)*. NIST, November 2001. Federal Information Processing Standards Publication 197. Available from `http://csrc.nist.gov/encryption/aes/`.

[27] Girault, M. An identity-based identification scheme based discrete logarithm modulo a composite number. Aus *Advances in Cryptology – EUROCRYPT '90* (1991), I. Damgård, Hrsg., Bd. 473 aus *Lecture Notes in Computer Science*, Springer-Verlag, S. 481–486.

[28] Goldreich, O. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, Bd. 17 aus *Algorithms and Combinatorics*. Springer–Verlag, 1999.

[29] Goldwasser, S. und Bellare, M. Lecture Notes on Cryptography. `http://www-cse.ucsd.edu/user/mihir/papers/gb.html`, 1999.

[30] Guillou, L. C. und Quisquater, J.-J. A Practical Zero-Knowledge Protocol Fitted To Security Microprocessors Minimizing Both Transmission and Memory. Aus *Advances in Cryptology – EUROCRYPT '88* (1988), C. G. Günther, Hrsg., Bd. 330 aus *Lecture Notes in Computer Science*, Springer-Verlag, S. 123–128.

[31] Hafner, J. L. und McCurley, K. S. A rigorous subexponential algorithm for computation of class groups. *Journal of the American Mathematical Society 2*, 4 (1989), 837–850.

[32] Hamdy, S. und Möller, B. Security of Cryptosystems Based on Class Groups of Imaginary Quadratic Orders. Aus *Advances in Cryptology – ASIACRYPT 2000* (2000), T. Okamoto, Hrsg., Bd. 1976 aus *Lecture Notes in Computer Science*, Springer-Verlag, S. 234–247.

[33] Hardy, G. H. und Wright, E. M. *An Introduction to the Theory of Numbers*, 5. Aufl. Oxford University Press, 1979.

[34] Horster, P., Michels, M. und Petersen, H. Meta-ElGamal signature schemes. Aus *2nd ACM Conference on Computers and Communications Security* (1994), ACM Press, S. 96–107.

[35] Hua, L. K. *Introduction to Number Theory*. Springer–Verlag, 1982.

[36] Hühnlein, D. Quadratic orders for NESSIE – Overview and parameter sizes of three public key families. Tech. Ber. TI-3/00, Technische Universität Darmstadt, Fachbereich Informatik, 2000. `http://www.informatik.tu-darmstadt.de/TI/Veroeffentlichung/TR/`.

[37] Hühnlein, D. und Takagi, T. Reducing Logarithms in Totally Non-maximal Imaginary Quadratic Orders to Logarithms in Finite Fields. Aus *Advances in Cryptology – ASIACRYPT '99* (1999), K. Y. Lam, E. Okamato, und C. Xing, Hrsg., Bd. 1716 aus *Lecture Notes in Computer Science*, Springer-Verlag, S. 219–231.

[38] Hunter, S. und Sorenson, J. Approximating the Number of Integers Free of Large Prime Factors. *Mathematics of Computation 66*, 220 (1997), 1729–1741.

[39] IEEE P1363. IEEE P1363: Standard Specification for Public-Key Cryptography. `http://grouper.ieee.org/groups/1363/`.

[40] Jacobson, Jr., M. J. *Subexponential Class Group Computation in Quadratic Orders*. Dissertation, Technische Universität Darmstadt, Fachbereich Informatik, Darmstadt, Germany, 1999.

[41] Kaplan, P. Sur le 2-groupe des classes d'idéaux des corps quadratiques. *Journal für die reine und angewandte Mathematik 283/284* (1976), 313–363. French.

[42] Knuth, D. E. *The Art of Computer Programming — Sorting and Searching*, Bd. 3. Addison–Wesley, 1973.

[43] Knuth, D. E. *The Art of Computer Programming — Seminumerical Algorithms*, 2. Aufl., Bd. 2. Addison–Wesley, 1981.

[44] Koblitz, N. Elliptic Curve Cryptosystems. *Mathematics of Computation 48*, 177 (1987), 203–209.

[45] Lang, S. *Algebraic Number Theory*, 2. Aufl., Bd. 110 aus *Graduate Texts in Mathematics*. Springer–Verlag, 1994.

[46] Lenstra, A. K. und Lenstra, H. W., Hrsg. *The development of the number field sieve*. Nr. 1554 aus Lecture Notes in Mathematics. Springer–Verlag, 1993.

[47] Lenstra, A. K. und Verheul, E. R. Selecting Cryptographic Keysizes. *Journal of Cryptology 14*, 4 (2001), 255–293.

[48] Lenstra, Jr., H. W. und Pomerance, C. A Rigorous Time Bound for Factoring Integers. *Journal of the American Mathematical Society 5*, 3 (1992), 483–516.

[49] LiDIA – A C++ Library For Computational Number Theory. `http://www.informatik.tu-darmstadt.de/TI/LiDIA/`. The LiDIA Group.

[50] Littlewood, J. E. On the Class Number of the Corpus $P(\sqrt{-k})$. Aus *Proceedings of the London Mathematical Society*, Bd. 27 aus *2nd series*. Cambridge University Press, 1928, S. 358–372.

[51] Menezes, A. J., van Oorschot, P. C. und Vanstone, S. A. *Handbook of Applied Cryptography*. CRC Press, 1997.

[52] Miller, V. S. Use of Elliptic Curves in Cryptography. Aus *Advances in Cryptology – CRYPTO '85* (1986), H. C. Williams, Hrsg., Bd. 218 aus *Lecture Notes in Computer Science*, Springer-Verlag.

[53] Möller, B. Algorithms for Multi-exponentiation. Aus *Selected Areas in Cryptography, SAC 2001* (2001), S. Vaudenay und A. M. Youssef, Hrsg., Bd. 2259 aus *Lecture Notes in Computer Science*, S. 165–180.

[54] Mollin, R. A. *Quadratics*. CRC Press, 1995.

[55] Odlyzko, A. M. The Future of Integer Factorization. *CryptoBytes 1*, 2 (1995). `http://www.rsa.com/rsalabs/pubs/cryptobytes/`.

[56] Pohlig, S. C. und Hellman, M. E. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Transactions on Information Theory 24* (1978), 106–110.

[57] Pointcheval, D. und Stern, J. Security Proofs for Signature Schemes. Aus *Advances in Cryptology – EUROCRYPT '96* (1996), U. Maurer, Hrsg., Bd. 1070 aus *Lecture Notes in Computer Science*, Springer-Verlag, S. 387–398.

[58] Pointcheval, D. und Stern, J. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology 13*, 3 (2000), 361–396.

[59] Poupard, G. und Stern, J. Security Analysis of a Practical "on the fly" Authentication and Siganture Generation. Aus *Advances in Cryptology – EUROCRYPT '98* (1998), K. Nyberg, Hrsg., Bd. 1403 aus *Lecture Notes in Computer Science*, Springer-Verlag, S. 422–436.

[60] Rédei, L. Über die Klassenzahl des imaginären quadratischen Zahlkörpers. *Journal für die reine und angewandte Mathematik 159* (1928), 210–219.

[61] Rickert, N. W. Efficient reduction of Quadratic Forms. Aus *Computers and Mathematics, Cambridge, Massachusetts, 1989* (1989), E. Kaltofen und S. M. Watt, Hrsg., Springer-Verlag, S. 135–139.

[62] Rose, H. E. *A Course in Number Theory*, 2. Aufl. Oxford University Press, 1994.

[63] Rosser, J. B. und Schoenfeld, L. Approximate Formulas for some Functions of Prime Numbers. *Illinois Journal of Mathematics 6*, 1 (1962), 64–94.

[64] Rosser, J. B. und Schoenfeld, L. Sharper bounds for the Chebyshev Functions $\theta(x)$ and $\psi(x)$. *Mathematics of Computation 29*, 129 (1975), 243–269.

[65] Schnorr, C. P. Efficient Signature Generation by Smart Cards. *Journal of Cryptology 4*, 3 (1991), 161–174.

[66] Schnorr, C. P. und Lenstra, Jr., H. W. A Monte Carlo Factoring Algorithm With Linear Storage. *Mathematics of Computation 43*, 167 (1984), 289–311.

[67] Schoenfeld, L. Sharper bounds for the Chebyshev Functions $\theta(x)$ and $\psi(x)$, II. *Mathematics of Computation 30*, 134 (1976), 337–360.

[68] Schönhage, A. Fast reduction and composition of binary quadratic forms. Aus *Proc. ISSAC 91* (1991), S. M. Watt, Hrsg., ACM Press, S. 128–133.

[69] Schönhage, A., Grotefeld, A. und Vetter, E. *Fast algorithms. A multitape Turing machine implementation.* B.I. Wissenschaftsverlag, 1994.

[70] Schönhage, A. und Strassen, V. Schnelle Multiplikation großer Zahlen. *Computing 7* (1971), 281–292.

[71] SEC 1. *Elliptic Curve Cryptography.* Standards for Efficient Cryptography Group, September 2000. Working Draft. Available from `http://www.secg.org/`.

[72] SEC 2. *Recommended Elliptic Curve Domain Parameters*. Standards for Efficient Cryptography Group, September 2000. Working Draft. Available from `http://www.secg.org/`.

[73] Silverman, R. D. Exposing the Mythical MIPS Year. *IEEE Computer 32*, 8 (1999), 22–26.

[74] Takagi, T. *New public-key cryptosystems with fast decryption*. Dissertation, Technische Universität Darmstadt, Fachbereich Informatik, Darmstadt, Germany, 2001. `http://elib.tu-darmstadt.de/diss/000104/`.

[75] Terr, D. C. A modification of Shanks' Baby-Step-Giant-Step algorithm. *Mathematics of Computation 69*, 230 (1999), 767–773.

[76] Teske, E. E. Computing discrete logarithms with the parallelized kangaroo method. Tech. Ber., Centre for Applied Cryptographic Research, University of Waterloo, 2001. `http://www.cacr.math.uwaterloo.ca/techreports/2001/corr2001-01.ps`.

[77] Teske, E. E. Square-root algorithms for the discrete logarithm problem (a survey). Tech. Ber., Centre for Applied Cryptographic Research, University of Waterloo, 2001. `http://www.cacr.math.uwaterloo.ca/techreports/2001/corr2001-07.ps`.

[78] van Oorschot, P. C. und Wiener, M. J. On Diffie-Hellman Key Agreement with Short Exponents. Aus *Advances in Cryptology – EUROCRYPT '96* (1996), U. Maurer, Hrsg., Bd. 1070 aus *Lecture Notes in Computer Science*, Springer-Verlag, S. 332–343.

[79] van Oorschot, P. C. und Wiener, M. J. Parallel Collusion Search with Cryptanalytic Applications. *Journal of Cryptology 12*, 1 (1999), 1–28.

[80] Vollmer, U. Asymptotically Fast Discrete Logarithms in Quadratic Number Fields. Aus *Algorithmic Number Theory, ANTS-IV* (2000), W. Bosma, Hrsg., Bd. 1838 aus *Lecture Notes in Computer Science*, Springer-Verlag, S. 581–594.

[81] Weilert, A. *Effiziente Algorithmen zur Berechnung von Idealsummen in quadratischen Ordnungen*. Dissertation, Rheinische Friedrich-Wilhelms-Universität Bonn, 2000. German.

Index

Äquivalenzklasse **15**, 16, 20
  Vertreter 15
    reduzierter 15, 17

Adaptively-Chosen-Ciphertext-Attack . . . *siehe* Angepaßt-Gewählter-Schlüsseltext-Angriff
Adaptively-Chosen-Message-Attack . . . . . *siehe* Angepaßt-Gewählter-Text-Angriff
AES 36
Angepaßt-Gewählter-Schlüsseltext-Angriff . 89, 100
Angepaßt-Gewählter-Text-Angriff . . 88, 94, 96, 97
Arithmetik
  einfach-genaue 79, 108, 113, 120, 124
  gemischt-genaue 109, 112
  mehrfach-genaue 104–105, 121

Baby-Step-Giant-Step-Algorithmus 72
Bekannter-Schlüsseltext-Angriff 89
Bekannter-Text-Angriff 88
Berechnungsproblem 84, 85
Bereichsparameter 23–32
  IQ-DH . . *siehe* IQ-DH, Bereichsparameter
  IQ-DSA *siehe* IQ-DSA, Bereichsparameter
  IQ-GQ . . *siehe* IQ-GQ, Bereichsparameter
  IQ-IES . . *siehe* IQ-IES, Bereichsparameter
  IQ-RDSA . . . . . *siehe* IQ-RDSA, Bereichsparameter
Bit-String **21**, 35
bitsToOctets **21**, 34–36
Brauer-Siegel-Theorem **56**, 60, 77, 80

checkDisc **27**, 30, 31
checkDiscAny **26**
checkDiscP **26**, 27
checkDiscPQ **26**, 27
checkDomainIQDH **29**, 48
checkDomainIQDSA **29**, 40
checkDomainIQGQ **31**, 42
checkDomainIQIES **29**, 45
checkDomainIQRDSA **30**, 38
checkIdeal **28**, 30, 33, 40, 42, 44
checkPubKeyIQDH **32**, 48
checkPubKeyIQDSA **32**, 41
checkPubKeyIQGQ **33**, 43
checkPubKeyIQIES **32**, 45, 47
checkPubKeyIQRDSA **32**, 39
Chosen-Ciphertext-Attack . . . . . *siehe* Gewählter-Schlüsseltext-Angriff
Chosen-Message-Attack . . . . . *siehe* Gewählter-Text-Angriff
Cl(Δ) . . . . . *siehe* Klassengruppe
Cohen-Lenstra-Heuristik 59, 60, 77
  Erweiterung 60, 66

Δ . . . . . *siehe* Diskriminante
deriveKey **35**, 45–49
DHP . . . . . *siehe* Diffie-Hellman-Problem
Diffie-Hellman-Problem 53, **84**, 101
discSize **24**, 25, 26, 80–82
Diskretes-Logarithmus-Problem 53, **84**, 97
Diskriminante **11**, 105
  Führer **11**
  fundamentale **11**, 23–25, 28–30, 55–66, 79, 101
  nicht-fundamentale 64
  Primfaktoren 57, 65
divide 18
DLP . . . . *siehe* Diskretes-Logarithmus-Problem
DSA 90, 123
  Variante 91, 96

ENC **36**, 45–47
Entscheidungsproblem 85
Entschlüsselung
  IQ-IES . . . . *siehe* IQ-IES, Entschlüsselung
Exponentiation
  wNAF-based interleaving 122–124

Fälschung . . . . . *siehe* Signatur, Fälschung
Führer . . . . . *siehe* Diskriminante, Führer
Faktorisierungsproblem 58, 64, 86
FlexiProvider 2, 125
Forking-Lemma **87**, 93, 99
Form **105**
  äquivalente **105**

indefinite **105**
minimale über **115**, 116
normale **106**
Normalisierung 107
positiv definite **105**
primitive **105**
Reduktion 105–120
reduzierte **106**, 116
semi-reduzierte **109**
fullStepRickert **110**, 114
fullStepSchoenhage **117**, 119
Fundamental-Diskriminante . . . . . *siehe* Diskriminante, fundamentale

genDiscP **25**, 29, 30
genDiscPQ **25**, 29, 30
genDomainIQDH **28**, 48
genDomainIQDSA **28**, 40
genDomainIQGQ **30**, 42
genDomainIQIES **28**, 45
genDomainIQRDSA **29**, 38
genKeyPairIQDH **31**
genKeyPairIQDSA **32**, 41
genKeyPairIQGQ **32**, 42
genKeyPairIQIES **32**, 45, 46
genKeyPairIQRDSA **32**, 38, 39
Gewählter-Schlüsseltext-Angriff 89
Gewählter-Text-Angriff 88
GF-DLP 64
GNFS 66, 72
erwartete Laufzeit 66
GQ-Signaturverfahren 98
Guillou-Quisquater-Signaturverfahren . . . *siehe* GQ-Signaturverfahren

hash **34**, 38, 40, 42
Hash-Funktion 34, 94
kryptographische **34**, 35, 38, 40, 42
hashToInteger **34**, 39–44
$h(\Delta)$ . . . . . *siehe* Klassenzahl
HMAC–SHA-1–160 35

Ideal **12**
äquivalentes **15**, 16
ambiges **57**, 86
Arithmetik 15, 17, 122
Darstellung **12**, 20, 22, 23
Kompression 21
Konvertierung 20–23
normale **12**, 15, 18, 27
Normalisierung 12, 107
Division **18**
Erzeugung 13
ganzes **12**
gebrochenes **12**, 15
inverses 15, 18
Konvertierung 21
Multiplikation 15, **17**
primitives **12**, 15, 27, 103
Quadrierung **17**
Reduktion 15, 105–120
reduziertes . . **13**, 15, 18, 20, 27–29, 31, 32, 103
zufällig gewähltes 27, 77–79
Ideal-Klasse **15**, 58
idealToOctets **22**, 39–44, 46, 47, 49
Index-Berechnungs-Algorithmus . 66–68, 80, 82
integerToOctets **21**, 22
IQ-DH 33, **48**, 48–49
Bereichsparameter **28**, 48
Schlüssel
öffentlicher 33, 48
privater 33
Schlüsselaustausch 48–49
Schlüsselpaar **31**, 48
Vorbereitung 48
IQ-DHP **84**, 86, 101
IQ-DLP 64, 79, **84**, 86
IQ-DSA **40**, 40–42, 122
Bereichsparameter **28**, 40, 41
Effizienz 97
Optimierung 123
Schlüssel
öffentlicher 41
Schlüsselpaar **31**, 41
Sicherheit 96–97
Signatur 41
Verifikation 41–42
Vorbereitung 40–41
IQ-GQ **42**, 42–44, 122
Bereichsparameter **30**, 42, 43
Optimierung 124
Schlüssel
öffentlicher 43
Schlüsselpaar **32**, 42
Sicherheit 98–100
Signatur 43
Verifikation 43–44
Vorbereitung 42–43
IQ-IES 33, 36, **44**, 44–47
Bereichsparameter **28**, 45
Entschlüsselung 46–47
Schlüssel
öffentlicher 45
Schlüsselpaar **31**, 45
Sicherheit 100–101

Verschlüsselung 45–46
Vorbereitung 44–45
IQ-MPQS **67**
erwartete Laufzeit 67, 80–82
IQ-OP **84**, 86
IQ-RDSA **38**, 38–40, 122
Bereichsparameter **29**, 38, 39
Optimierung 122–123
Schlüssel
öffentlicher 39
Schlüsselpaar **31**, 38
Sicherheit 89–96
Signatur 39
Verifikation 39–40
Vorbereitung 38–39
IQ-RP **84**, 86
IQDH **33**, 46, 47, 49
inverse **18**
isEqual **17**
isMinimal **115**, 119
isNormal **12**
isPrime **20**, 22, 25, 29, 30
isReduced **13**, 28

Känguru-Algorithmus 72
Kein-Text-Angriff 88, 89, 93, 94, 99
Klassengruppe **15**, 23, 54
2-Anteil 57
Arithmetik **15**
Berechnung diskreter Logarithmen 24, 64–76
Elemente **15**
Darstellung 20, 103
Erzeugendensystem 77
gerader Anteil 57–58, 79
Größe 55–57
Ordnung . . . . . *siehe* Klassenzahl
Struktur 57–59
ungerader Anteil 59
Klassenzahl . **15**, 38, 55, 58, 64, 72, 77, 86, 101
arithmetischer Mittelwert 56
Glattheitsschranke 59–63, 73–76
Glattheitswahrscheinlichkeit 59–63, 75–76, 101
Größenordnung 56
Schranken 56, 80
Klassenzahl-Formel, analytische 55
Known-Message-Attack . . . . . *siehe* Bekannter-Text-Angriff
Kronecker-Symbol 27

λ-Algorithmus 72

MAC . . . . . *siehe* Message-Authentication-Code
MAC **35**, 45–47
malleable . . . . . *siehe* verformbar
Many-One-Reduktion 85
Message-Authentication-Code 35
MPQS 66
multiply **17**, **104**

nextPrime **20**, 78
No-Message-Attack . . . *siehe* Kein-Text-Angriff
normalize **12**
normalizeForm **107**, 108
normalizeForm1 . . . . . **107**

Octet-String **21**, 23, 34
octetsToBits **21**, 34–36
octetsToIdeal **23**, 47
octetsToInteger **21**, 23, 34
oddRandomInteger **20**, 25, 29
OP . . . . . *siehe* Ordnungsproblem
$\mathcal{O}_\Delta$-Ideal . . . . . *siehe* Ideal
$\mathcal{O}_\Delta$ . . . . . *siehe* Ordnung
Ordnung **12**
Ordnungsproblem **84**, 90

$(p-1)$-Faktorisierungs-Algorithmus 74, 75
Pohlig-Hellman-Algorithmus . . . 73–76, 80, 102
polynomiell ununterscheidbar **94**, 95, 96
Polynomzeit-Reduktion 85–86
primeIdeal **14**, 23, 27, 78
Primideal 13
Erzeugung **13**, 77

quadratfrei 12, 65
Quadratwurzel-Algorithmus 55, 72–73, 80

randomIdeal **27**, 28, 29, 32
randomIdeal2 . . . . . **78**
randomInteger **20**, 27, 30, 31, 41, 78
reduce **16**, 27, 31–33, 39–44, 78
reduceGauss **108**
reduceRickert **114**
reduceRickertCore **113**, 114
reduceRickertFinal **110**, 114, **116**
reduceSchoenhageCore **119**
reduceSchoenhageFinal **116**
Reduktion . . . . . *siehe* Polynomzeit-Reduktion
einer Form . . . . . *siehe* Form, Reduktion
eines Ideal . . . . . *siehe* Ideal, Reduktion
Reduktions-Algorithmus
Laufzeit
asymptotische 120
gemessene 120
nach Gauß 16, 106–108, 121
nach Rickert 108–114, 121

nach Schönhage 114–121
ressol **14**
ρ-Algorithmus 72
Riemann-Vermutung
erweiterte 56, 61, 77, 80
verallgemeinerte 67
RP . . . . . *siehe* Wurzelproblem
RSA 123
RSA-155 67
RSA-Modulus 75, 98

Schlüssel
öffentlicher 23, 31–33
geheimer . . . . . *siehe* Schlüssel, privater
privater 23, 31–33
symmetrischer 35
Schlüsselableitung **34**, 45, 48, 101
Schlüsselaustausch
IQ-DH . . *siehe* IQ-DH, Schlüsselaustausch
Vorbereitung
IQ-DH . . . . . *siehe* IQ-DH, Vorbereitung
Schlüsselaustauschverfahren 47–49
Schlüsselgewinn 88, 89
Schlüsselpaar 31–33
IQ-DH . . . . . *siehe* IQ-DH, Schlüsselpaar
IQ-DSA . . . . . *siehe* IQ-DSA, Schlüsselpaar
IQ-GQ . . . . . *siehe* IQ-GQ, Schlüsselpaar
IQ-IES . . . . . *siehe* IQ-IES, Schlüsselpaar
IQ-RDSA . *siehe* IQ-RDSA, Schlüsselpaar
Schlüsseltext
Entschlüsselung 89
Unterscheidung 89
Schnorr-Signaturverfahren 91, 96
SHA-1 34
Sicherheit 86–89
IQ-DSA . . . . . *siehe* IQ-DSA, Sicherheit
IQ-GQ . . . . . *siehe* IQ-GQ, Sicherheit
IQ-IES . . . . . *siehe* IQ-IES, Sicherheit
IQ-RDSA . . . . . *siehe* IQ-RDSA, Sicherheit
semantische 35
Sicherheitsparameter . . 23–31, 79, 80, 102, 121, 124
Signatur
Fälschung
existenzielle 88, 93, 97, 99
universelle 88
IQ-DSA . . . . . *siehe* IQ-DSA, Signatur
IQ-GQ . . . . . *siehe* IQ-GQ, Signatur
IQ-RDSA . . . . . *siehe* IQ-RDSA, Signatur
Verifikation
IQ-DSA . . . . *siehe* IQ-DSA, Verifikation
IQ-GQ . . . . . *siehe* IQ-GQ, Verifikation
IQ-RDSA . *siehe* IQ-RDSA, Verifikation
Vorbereitung
IQ-DSA . . . *siehe* IQ-DSA, Vorbereitung
IQ-GQ . . . . . *siehe* IQ-GQ, Vorbereitung
IQ-RDSA *siehe* IQ-RDSA, Vorbereitung
Signaturverfahren 38–44
vom ElGamal-Typ 38, 90
simpleStepRickert **111**, 114
Simulator 94–96, 99
smoothOrder 73
square **18**, **105**

Tag 35
Tagging 35
Turing-Maschine 85
probabilistische 85, 87, 94, 96

ununterscheidbar . . . . . *siehe* polynomiell ununterscheidbar

verformbar 101
vernachlässigbar 55, 59, 75, **87**, 88–90, 93, 95–97, 99, 100
Verschlüsselung
IQ-IES . . . . . *siehe* IQ-IES, Verschlüsselung
symmetrische **36**, 45
Vorbereitung
IQ-IES . . *siehe* IQ-IES, Verschlüsselung
Verschlüsselungsverfahren 44–47

Wurzelproblem 53, **84**, 90, 93, 96, 99

xgcd **17**, 18, 104, 105
XOR 36

Zufallsband 85
Zufallsorakel 87, 90, 93, 94, 99