UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

– Studienarbeit –

Von identitätsbasierten Kryptosystemen zu CCA-Sicherheit

Jonas Schrieb
jonas@upb.de

betreut durch:

Prof. Dr. Johannes Blömer
AG Codes und Kryptographie
Universität Paderborn

Paderborn, den 15. August 2006

Zusammenfassung

Identitätsbasierte Kryptographie ist eine neuere Form der asymmetrischen Kryptographie, bei der die Identität eines Benutzers (z.B. seine Emailadresse) zugleich sein öffentlicher Schlüssel ist. Zertifikate, welche die Zugehörigkeit eines öffentlichen Schlüssels zu einer Identität bestätigen, werden dadurch überflüssig.
Kern dieser Studienarbeit ist eine Konstruktion von Boneh, Canetti, Halevi und Katz mit der sich beliebige „schwach-sichere“ (IND-sID-CPA) identitätsbasierte Kryptosysteme in „sehr sichere“ (IND-CCA2) Public-Key-Kryptosysteme umwandeln lassen. Das Interessante daran ist, dass nur wenige Verfahren bekannt sind, um sehr sichere Public-Key-Kryptosysteme zu erzeugen und dieses einen völlig neuartigen Ansatz darstellt.
Im Rahmen dieser Studienarbeit wird das Verfahren ausführlich vorgestellt und auch konkrete Umsetzungen davon untersucht (d.h. es wird das allgemeine Verfahren auf ein konkretes identitätsbasiertes Kryptosystem angewendet).

Inhaltsverzeichnis

1 Einführung **7**

2 Grundlagen **8**
2.1 Notation 8
2.2 Zwei asymmetrische Verschlüsselungsparadigmen 13
2.3 Zwei weitere kryptographische Primitive 19
2.4 Bilineare Gruppen 22

3 Das allgemeine Verfahren für PKE aus beliebigem IBE **25**
3.1 Die Idee 25
3.2 Die Umsetzung 27
3.3 Beweis der IND-CCA2 Sicherheit 29

4 Konkrete Umsetzungen mit Boneh-Boyen-IBE **47**
4.1 Boneh-Boyen-IBE 47
4.2 Eine konkrete Umsetzung: Boneh-Katz-PKE 53
4.3 Eine alternative Umsetzung: my-$\mathcal{PKE}$ 56

5 Fazit **63**

Literatur **64**

1 Einführung

In Public-Key-Kryptosystemen (PKE[1]) wird festgelegt, wie Alice mit Hilfe des öffentlichen Schlüssels von Bob eine Nachricht an ihn verschlüsseln und Bob diese mit seinem privaten Schlüssel rekonstruieren kann. Es wird nicht geklärt, woher Alice diesen öffentlichen Schlüssel bezieht und wie Alice sicherstellt, dass dieser Schlüssel auch wirklich Bob gehört. Für gewöhnlich ist der öffentliche Schlüssel auf einem Schlüsselserver hinterlegt und die Authentizität wird durch das Zertifikat einer vertrauenswürdigen Stelle bestätigt.

1984 schlägt Shamir in [Sha85] das vereinfachende Konzept der *identitätsbasierten Verschlüsselung* (IBE[2]) vor. Alice soll nicht aufwändig den öffentlichen Schlüssel von Bob ermitteln müssen, statt dessen soll eine eindeutige Identität von Bob (z.B. Mail-Adresse, Loginname oder Personalausweisnummer) gleichzeitig auch sein öffentlicher Schlüssel sein. Die privaten Schlüssel aller Benutzer werden von einer zentralen vertrauenswürdigen Stelle mit Hilfe eines Masterkeys berechnet und nach erfolgreicher Authentifizierung an die Benutzer verteilt.

Erst 2001 stellen Boneh und Franklin in [BF01] sowie Cocks in [Coc01] die ersten voll funktionsfähigen IBE vor. Dies hat die Forschung zur identitätsbasierten Kryptographie sehr belebt. Neben Verbesserungen der Kryptosysteme an sich gibt es zahlreiche Arbeiten, die IBE nutzen, um andere kryptographische Primitive wie z.B. Signaturen [BF01, Wat05, §6 bzw. §7] oder PKE (siehe unten) zu konstruieren.

Canetti, Halevi und Katz entwerfen in [CHK04] eine Konstruktion, mit der sich aus beliebigen „schwach-sicheren“ (IND-sID-CPA-sicher – exakte Definitionen folgen) IBE „sehr sichere“ (IND-CCA2-sicher) PKE erzeugen lassen. Das Interessante daran ist, dass nur wenige Verfahren bekannt sind, um sehr sichere PKE zu erzeugen und dieses einen völlig neuartigen Ansatz darstellt. Boneh und Katz verbessern in [BK05] diese Konstruktion und schlagen eine effiziente Umsetzung der konkreten kryptographischen Primitive vor (unter anderem das IBE von Boneh und Boyen aus [BB04]). Ziel dieser Studienarbeit ist es, diese Konstruktionen vorzustellen und die konkrete Umsetzung genauer zu untersuchen.

Die Studienarbeit ist wie folgt strukturiert: Zuerst werden in Abschnitt 2 alle für diese Arbeit notwendigen Definitionen gegeben und erläutert. In Abschnitt 3 wird erst die grundlegende Idee aus [CHK04] erläutert, wie man aus IBE sichere PKE erzeugt. Anschließend wird das Verfahren aus [BK05], welches diese Idee anwendet, vorgestellt und seine Sicherheit detailliert bewiesen. In Abschnitt 4 wird zunächst das IBE aus [BB04] eingeführt, um dann die in [BK05] vorgeschlagene Umsetzung und eine im Rahmen dieser Studienarbeit gefundene Alternative zu vorzustellen.

[1] PKE steht sowohl für „public-key encryption“ als auch für „public-key encryption scheme“.
[2] IBE steht sowohl für „identity-based encryption“ als auch für „identity-based encryption scheme“

2 Grundlagen

Zunächst wird in Abschnitt 2.1 die Notation vorgestellt, die allen folgenden Teilen zugrundeliegt. Es wird insbesondere auf Zufallsexperimente eingegangen. Abschnitt 2.2 stellt die zwei asymmetrischen Verschlüsselungsparadigmen „Public-Key-Kryptographie“ und „identitätsbasierte Kryptographie“ vor, welche für diese Arbeit eine zentrale Rolle spielen. Weitere verwendete kryptographische Primitive werden im darauf folgenden Abschnitt 2.3 erläutert. Im letzten Abschnitt 2.4 werden bilineare Gruppen eingeführt, welche für die konkreten Umsetzungen in Kapitel 4 notwendig sind.

2.1 Notation

Allgemeines: Mit $|x|$ ist die Bit-Länge der Darstellung von x gemeint. Mit $x.y$ ist die Konkatenation der Darstellungen von x und y gemeint. Sei $\mathcal{A}$ ein Algorithmus, dann ist $T(\mathcal{A})$ die Laufzeitfunktion in Abhängigkeit der Bit-Länge der Eingabe.

Zufallsexperimente: Moderne Verschlüsselungsverfahren bestehen in der Regel aus probabilistischen Algorithmen (das sind solche Algorithmen, deren Ablauf neben den eigentlichen Eingabewerten auch von Zufallsbits abhängt). Ihre Sicherheit wird durch Zufallsexperimente definiert, in denen ein Angreifer $\mathcal{A}$ mit einer möglichst geringen Wahrscheinlichkeit einen Vorteil erlangen soll. Für die Analyse dieser Zufallsexperimente soll zunächst die notwendige Notation festgelegt werden.

Ist M eine Menge, so bedeutet $m :=_R M$, dass $m \in M$ gemäß der Gleichverteilung zufällig gewählt wird.

Um Wahrscheinlichkeiten in den Zufallsexperimenten ausdrücken zu können, führt man Zufallsvariablen über die verwendeten (Teil-)Algorithmen ein:

Definition 1

Sei $\mathcal{A} : X \times \{0,1\}^* \to Y, (x,r) \mapsto \mathcal{A}(x,r)$ ein probabilistischer Algorithmus, der $x \in X$ als Eingabe bekommt und die zufälligen Bits $r \in \{0,1\}^*$ verwendet. Im Folgenden wird die Angabe r oft weggelassen und nur explizit aufgeführt, wenn die Verwendung der zufälligen Bits betont werden soll.

Zu jeder Eingabe generiert $\mathcal{A}$ eine bestimmte Anzahl an Zufallsbits. Da es nur endlich viele Eingaben der Länge n gibt, existiert ein Maximum $max(n)$ über die benötigten Zufallsbits bei allen Eingaben der Länge n. Dann kann man ohne Einschränkung voraussetzen, dass $\mathcal{A}$ für all diese Eingaben genau $max(n)$ Zufallsbits generiert und $\mathcal{A}$ gegebenfalls die nicht benötigten Zufallsbits verwirft. In dieser Arbeit wird es weiterhin so sein, dass bei $\mathcal{A}$ für Eingaben gleicher Länge die Ausgabe auch gleichlang ist.

Dann ist die *zu $\mathcal{A}$ bei Eingabe x gehörende Zufallsvariable*:

$$\mathcal{A}(x) : \{0,1\}^{max(|x|)} \to Y, r \mapsto \mathcal{A}(x,r)$$

Es gibt also zu jeder möglichen Eingabe x eine Zufallsvariable. Für diese Zufallsvariable ist dann das x fest und nur die Wahl der zufälligen Bits r variabel. Alle Wahrscheinlichkeiten über diese Zufallsvariable gehen nur über die zufällig und gleichverteilte Wahl von $r \in \{0,1\}^{max(|x|)}$.

Zufallsexperimente bestehen meist aus dem Zusammenspiel mehrerer Algorithmen, so dass man zusammengesetzte Algorithmen benötigt.

Definition 2

Seien $\mathcal{A}_1 : X \times \{0,1\}^* \to Y$ und $\mathcal{A}_2 : Y \times \{0,1\}^* \to Z$ probabilistische Algorithmen. Diese werden durch die Schreibweise

$$\mathcal{A}(x) := \{y := \mathcal{A}_1(x); z := \mathcal{A}_2(y)\}$$

zu einem neuen Algorithmus $\mathcal{A}$ zusammengesetzt werden. $\mathcal{A}_2$ wird nach $\mathcal{A}_1$ ausgeführt und übernimmt als Eingabe die Ausgabe von $\mathcal{A}_1$. Formal ist $\mathcal{A}$ dann definiert als

$$\mathcal{A} : X \times \{0,1\}^* \to \underbrace{Y \times Z}_{=:Z'}, \quad (x, r_1.r_2) \mapsto (\mathcal{A}_1(x, r_1), \mathcal{A}_2(\mathcal{A}_1(x, r_1), r_2))$$

$\mathcal{A}$ bekommt also als Eingabe den Eingabewert für $\mathcal{A}_1$ und hat als Ausgabe alle Ausgabewerte der Teilalgorithmen.

Nach Definition 1 muss die Anzahl der Zufallsbits einer zum Algorithmus gehörenden Zufallsvariable fest sein. Dies ist hier gegeben. Dazu beachte man, dass bei fester Eingabe x nach der Annahme aus Definition 1 alle Ausgabewerte von $\mathcal{A}_1$ (und damit Eingabewerte von $\mathcal{A}_2$) die gleiche Länge haben und somit $\mathcal{A}_2$ in jedem Fall die selbe Anzahl an Zufallsbits benötigt. Damit ist der Definitionsbereich der zu $\mathcal{A}$ bei Eingabe x gehörenden Zufallsvariable gleich $\{0,1\}^{max_1(|x|)+max_2(|\mathcal{A}_1(x)|)}$. Dabei sind max_1 und max_2 die Funktionen, die einer Eingabelänge die Anzahl der von $\mathcal{A}_1$ bzw. $\mathcal{A}_2$ verbrauchten Zufallsbits zuweisen.

Lemma 2.1

Seien $\mathcal{A}_1, \mathcal{A}_1^* : X \times \{0,1\}^* \to Y$ und $\mathcal{A}_2 : Y \times \{0,1\}^* \to Z$ probabilistische Algorithmen, welche wie folgt zu den neuen Algorithmen $\mathcal{A}$ und $\mathcal{A}^*$ zusammengesetzt werden:

$$\mathcal{A}(x) := \{y := \mathcal{A}_1(x); z := \mathcal{A}_2(y)\}$$
$$\mathcal{A}^*(x) := \{y := \mathcal{A}_1^*(x); z := \mathcal{A}_2(y)\}$$

Falls für $x \in X$ die Wahrscheinlichkeitsverteilung von $\mathcal{A}_1(x)$ und $\mathcal{A}_1^*(x)$ gleich ist, also $\forall y \in Y : Pr[\mathcal{A}_1(x) = y] = Pr[\mathcal{A}_1^*(x) = y]$ gilt, so ist auch die Wahrscheinlichkeitsverteilung von $\mathcal{A}(x)$ und $\mathcal{A}^*(x)$ gleich.

Beweis

x ist schon durch das Lemma fest gewählt. Seien $y \in Y$ und $z \in Z$ beliebig:

$$\begin{aligned} & Pr[\mathcal{A}(x) = (y, z)] \\ = \; & Pr[\mathcal{A}_1(x) = y \wedge \mathcal{A}_2(y) = z] \\ = \; & Pr[\mathcal{A}_1(x) = y] \cdot Pr[\mathcal{A}_2(y) = z)] \\ = \; & Pr[\mathcal{A}_1^*(x) = y] \cdot Pr[\mathcal{A}_2(y) = z)] \\ = \; & Pr[\mathcal{A}_1^*(x) = y \wedge \mathcal{A}_2(y) = z] \\ = \; & Pr[\mathcal{A}^*(x) = (y, z)] \end{aligned}$$

Im zweiten und vierten Schritt wird angewendet, dass $\mathcal{A}_1(x)$ (bzw. $\mathcal{A}_1^*(x)$) und $\mathcal{A}_2(y)$ voneinander unabhängig sind, da sie unterschiedliche Zufallsbitstrings verwenden. Im dritten Schritt wird die Voraussetzung $Pr[\mathcal{A}_1(x) = y] = Pr[\mathcal{A}_1^*(x) = y]$ benutzt.

Durch iteriertes Anwenden von Definition 2 und Lemma 2.1 können komplexe Zufallsexperimente zusammengesetzt werden. Dabei soll im Folgenden auf unnötig starken Formalismus verzichtet werden. So kann es z.B. vorkommen, dass $\mathcal{A}_2$ nur einen Teil der Ausgabe von $\mathcal{A}_1$ nutzt.
Es kann auch vorkommen, dass $\mathcal{A}_1$ durch $\mathcal{A}_1^*$ ersetzt werden soll, sich aber die Bildbereiche beider Algorithmen unterscheiden. Dies kann z.B. der Fall sein, wenn $\mathcal{A}_1^*$ selbst so zusammengesetzt ist, dass es eine Variable benutzt, die in $\mathcal{A}_1$ nicht vorkommt und auch für alle nachfolgenden Algorithmen unwichtig ist. Nach obiger Konstruktion würde diese „interne Variable“ dann aber mit zur Ausgabe von $\mathcal{A}_1^*$ gehören. Folgendes Beispiel soll zeigen, wie diese Fälle zu verstehen sind:

Im ersten Experiment werden eine Münze und ein 12-seitiger Würfel geworfen. Das Ergebnis des Würfelwurfs ist die Eingabe eines Algorithmus $\mathcal{A}$.

$$\mathcal{E}_1 := \{m :=_R \{Z, K\}; w :=_R \{1, \ldots, 12\}; y := \mathcal{A}(w)\}$$

Die verwendeten Algorithmen sind (jetzt mit expliziter Angabe der zufälligen Bits):

$$\begin{aligned} \mathcal{M}: \quad & \{0,1\}^* \to \{Z, K\}, && r_1 && \mapsto \text{"}m :=_R \{Z, K\}\text{"} \\ \mathcal{W}: \quad & \{0,1\}^* \to \{1, \ldots, 12\}, && r_2 && \mapsto \text{"}w :=_R \{1, \ldots, 12\}\text{"} \\ \mathcal{A}: \quad & \{1, \ldots, 12\} \times \{0,1\}^* \to Y, && (w, r_3) && \mapsto \mathcal{A}(w, r_3) \end{aligned}$$

Hier besteht das erstgenannte Problem: weder beim Würfelwurf noch bei $\mathcal{A}$ wird m verwendet. Wohlwollend kann man $\mathcal{E}_1$ allerdings aus folgenden Algorithmen $\mathcal{M}', \mathcal{W}', \mathcal{A}'$

zusammengesetzt interpretieren:

$$\begin{aligned}
\mathcal{M}' :&\ \{0,1\}^* \to \{Z,K\}, & r_1 &\mapsto \mathcal{M}(r_1) =: m\\
\mathcal{W}' :&\ \{Z,K\} \times \{0,1\}^* \to \{1,\dots,12\}, & (m,r_2) &\mapsto \mathcal{W}(r_2) =: w\\
\mathcal{A}' :&\ \{Z,K\} \times \{1,\dots,12\} \times \{0,1\}^* \to Y, & (m,w,r_3) &\mapsto \mathcal{A}(w,r_3)
\end{aligned}$$

Anders ausgedrückt: um die Konstruktion aus Definition 2 zu erfüllen, werden die „eigentlichen“ Algorithmen $\mathcal{M}, \mathcal{W}, \mathcal{A}$ von „Wrapper-Algorithmen“ $\mathcal{M}', \mathcal{W}', \mathcal{A}'$ umschlossen, welche die komplette vorherige Ausgabe als Eingabe nehmen, aber die nicht benötigten Werte einfach verfallen lassen.

Das zweite Experiment läuft genau so ab, nur dass der Würfelwurf wie folgt simuliert wird: es wird eine Münze geworfen und das Ergebnis als 0 oder 1 interpretiert, dann wird ein normaler 6-seitiger Würfel geworfen. Die Kombination der beiden Zahlen liefert das Ergebnis.

$$\mathcal{E}_2 := \left\{ m :=_R \{Z,K\}; m' :=_R \{0,1\}; w' :=_R \{1,\dots,6\}; w := 6 \cdot m' + w'; y := \mathcal{A}(w) \right\}$$

Offensichtlich sind beide Varianten in dem Sinne „äquivalent“, dass für w ein zufällig und gleichverteilter Wert aus $\{1,\dots,12\}$ herauskommt. Nun möchte man Lemma 2.1 anwenden um zu erhalten, dass dann auch $\mathcal{E}_1$ und $\mathcal{E}_2$ in obigem Sinne äquivalent sind. Der mittlere Teil in $\mathcal{E}_2$ wäre eigentlich zu interpretieren als:

$$\begin{aligned}
\mathcal{W}^* : \{Z,K\} \times \{0,1\}^* &\to \{0,1\} \times \{1,\dots,6\} \times \{1,\dots,12\},\\
(m,r_2) &\mapsto \text{"}m' :=_R \{0,1\}; w' :=_R \{1,\dots,6\}; w := 6 \cdot m' + w'\text{"}
\end{aligned}$$

Jetzt liegt aber das zweitgenannte Problem vor: das Lemma ließe sich eigentlich nicht anwenden, da die Bildbereiche von $\mathcal{W}$ und $\mathcal{W}^*$ unterschiedlich sind, da $\mathcal{W}^*$ die zusätzliche Ausgabe von m', w' macht. Diese ist aber für den folgenden Verlauf des Zufallsexperiments nicht notwendig. Daher kann man auch hier mit einem gewissem Wohlwollen $\mathcal{E}_2$ aus den Algorithmen $\mathcal{M}', \pi_w \circ \mathcal{W}^*, \mathcal{A}'$ zusammengesetzt interpretieren, wobei π_w die Projektion nach der Komponente w ist:

$$\pi_w : \{0,1\} \times \{1,\dots,6\} \times \{1,\dots,12\} \to \{1,\dots,12\}, (m',w',w) \mapsto w$$

Dann hat $\pi_m \circ \mathcal{W}^*$ die passende Form

$$\pi_w \circ \mathcal{W}^* : \{Z,K\} \times \{0,1\}^* \to \{1,\dots,12\}, (m,r_2) \mapsto \text{"}m' \dots, w', \dots, w \dots\text{"} \mapsto w$$

und die Wahrscheinlichkeitsverteilung der Ausgabe gleicht der von $\mathcal{W}$. Damit ist das Lemma auf $\mathcal{W}, \pi_m \circ \mathcal{W}^*$ und $\mathcal{E}_1, \mathcal{E}_2$ anwendbar.

Fazit: Wie oben erwähnt, soll unnötig starker Formalismus vermieden werden. Das Beispiel zeigt, wie sich gewisse Ungenauigkeiten sinnvoll interpretieren lassen. Als wichtiges Ergebnis sollte man im Hinterkopf behalten, dass es bei den Zufallsexperimenten nicht darauf ankommt, wie ein Wert gebildet wird, sondern welche Wahrscheinlichkeitsverteilung dabei herauskommt.

Vernachlässigbare Funktionen: Kryptosysteme werden normalerweise in Abhängigkeit eines Sicherheitsparameters definiert. Dies ist eine natürliche Zahl, die z.B. die Länge der Schlüssel oder die Größe des Chiffretextraumes bestimmt.
Im Folgenden werden in Abhängigkeit dieses Sicherheitsparameters k Funktionen $Adv_{\mathcal{A}}(k)$ definiert, die den Vorteil eines zeitpolynomiellen Angreifers $\mathcal{A}$ beim Brechen eines Kryptosystems widerspiegeln. Intuitiv ist ein Verfahren sicher, wenn die Chance des Angreifers, das Verfahren zu brechen, mit steigendendem Sicherheitsparameter verschwindend gering wird. Man fordert, dass die Vorteilsfunktion $Adv_{\mathcal{A}}(k)$ vernachlässigbar ist:

Definition 3

Eine Funktion $f : \mathbb{N} \to \mathbb{R}$ ist *vernachlässigbar*, falls:

$\forall n \in \mathbb{N}: \quad f \in o\left(\frac{1}{k^n}\right)$ (Hier ist o das Landau-Symbol „klein-O“)

Es wird also gefordert, dass $Adv_{\mathcal{A}}(k)$ für *jedes* n schneller als $\frac{1}{k^n}$ gegen 0 geht. Dies hat folgenden Grund: Forderte man nur, dass $Adv_{\mathcal{A}}(k)$ schneller als z.B. $\frac{1}{k^2}$ gegen 0 geht (dies ist sicherlich schon sehr schnell), würde man das Kryptosystem im Mittel einmal brechen können, wenn man den Angreifer $\mathcal{A}$ k^2-mal ausführt. Analog kann man dies für jedes $\frac{1}{k^n}$ tun. Jedesmal wäre der neue Angreifer, der $\mathcal{A}$ k^n-mal ausführt, immer noch ein zeitpolynomieller Algorithmus. Um diesen Vorteilsgewinn durch wiederholte Ausführung auszuschließen, muss die Forderung für jedes n gelten.

Lemma 2.2

Seien f, g vernachlässigbar und $p : \mathbb{N} \to \mathbb{R}$ ein beliebiges Polynom. Es gilt:

1. $f + g$ ist vernachlässigbar
2. $f \cdot p$ ist vernachlässigbar
3. Falls $\forall k \in \mathbb{N} : p(k) \neq 0$: $\frac{f}{p}$ ist vernachlässigbar

Beweis

Beachte: $f \in o\left(\frac{1}{k^n}\right) \overset{\text{Def.}}{\iff} 0 = \lim\limits_{k\to\infty} \left|\frac{f(k)}{\frac{1}{k^n}}\right| = \lim\limits_{k\to\infty} |f(k) \cdot k^n|$

1. Sei $n \in \mathbb{N}$ beliebig. Es ist zu zeigen: $\lim\limits_{k\to\infty} |(f(k) + g(k)) \cdot k^n| = 0$
   Es ist: $0 \leq \lim\limits_{k\to\infty} |(f(k) + g(k)) \cdot k^n| \leq \lim\limits_{k\to\infty} |f(k) \cdot k^n| + \lim\limits_{k\to\infty} |g(k) \cdot k^n| = 0+0 = 0$
2. Sei $n \in \mathbb{N}$ beliebig. Zu zeigen: $\lim\limits_{k\to\infty} |(f(k) \cdot p(k)) \cdot k^n| = 0$. Sei m der Grad von p.
   Dann ist: $\lim\limits_{k\to\infty} |(f(k) \cdot p(k)) \cdot k^n| = \lim\limits_{k\to\infty} |f(k) \cdot k^{n+m}| \cdot \lim\limits_{k\to\infty} \left|\frac{p(k)}{k^m}\right| = 0 \cdot \underset{\in\mathbb{R}}{c} = 0$
3. Sei $n \in \mathbb{N}$ beliebig. Es ist zu zeigen: $\lim\limits_{k\to\infty} \left|\left(\frac{f(k)}{p(k)}\right) \cdot k^n\right| = 0$
   Es ist: $\lim\limits_{k\to\infty} \left|\left(\frac{f(k)}{p(k)}\right) \cdot k^n\right| = \lim\limits_{k\to\infty} |f(k) \cdot k^n| \cdot \lim\limits_{k\to\infty} \left|\frac{1}{p(k)}\right| = 0 \cdot \underset{\in\mathbb{R}}{c} = 0$

2.2 Zwei asymmetrische Verschlüsselungsparadigmen

Public-Key-Kryptosystem: Ein Public-Key-Kryptosystem ist das klassische und sicherlich am meisten bekannte asymmetrische Verfahren. Jeder Benutzer generiert sich ein Schlüsselpaar, welches aus einem öffentlichen und einem privaten Teil besteht. Mit Hilfe des öffentlichen Schlüssels kann eine Nachricht verschlüsselt werden, welche mit Kenntnis des privaten Schlüssels wiederhergestellt werden kann.

Definition 4

Ein *Public-Key-Kryptosystem* (PKE) wird durch drei probabilistische zeitpolynomielle Algorithmen definiert: Keygen, Encrypt, Decrypt (siehe Abbildung 1).

Keygen(1^k) bekommt als Eingabe den Sicherheitsparameter 1^k (in unärer Notation[3]). Es wird der Klartextraum $\mathcal{M}$, der Chiffretextraum $\mathcal{C}$ und der Schlüsselraum $\mathcal{K}$ festgelegt und das Schlüsselpaar $(E, D) \in \mathcal{K}$ ausgegeben, wobei E der öffentliche Schlüssel und D der private Schlüssel ist.

Encrypt(m, E) bekommt als Eingabe eine Nachricht $m \in \mathcal{M}$ und einen öffentlichen Schlüssel E des Empfängers. Es wird der Chiffretext $c \in \mathcal{C}$ ausgegeben.

Decrypt(c, D) bekommt als Eingabe den Chiffretext $c \in \mathcal{C}$ und den passenden privaten Schlüssel D des Empfängers. Es wird entweder die zuvor verschlüsselte Nachricht $m \in \mathcal{M}$ oder ein Fehlersymbol $\perp \notin \mathcal{M}$ ausgegeben.

Außerdem muss das PKE *korrekt* sein, d.h. für beliebige Schlüssel $(E, D) \in \mathcal{K}$, die von Keygen(1^k) erzeugt wurden, und jede Nachricht $m \in \mathcal{M}$ muss gelten:

$$\mathsf{Decrypt}(\mathsf{Encrypt}(m, E), D) = m$$

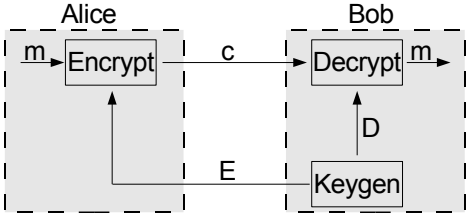

Abbildung 1: **Public-Key-Kryptosystem:** Alice möchte eine Nachricht an Bob verschlüsseln. Dieser erzeugt zunächst per Keygen den privaten und öffentlichen Schlüssel D bzw. E. Letzterer muss an Alice übertragen werden. Nun kann Alice mit Encrypt und dem öffentlichen Schlüssels E eine Nachricht m verschlüsseln und den Chiffretext c an Bob senden. Dieser kann die Nachricht m per Decrypt und dem nur ihm bekannten privaten Schlüssels D wiederherstellen.

[3] Der Algorithmus erzeugt $\mathcal{O}(k)$-Bit lange Schlüssel. Da die Laufzeit polynomiell durch die Bit-Länge der Eingabe beschränkt ist, könnte er bei binärer Notation nur $\mathcal{O}(\log(k))$-Bit lange Schlüssel erzeugen.

Wie weiter oben erwähnt, wird die Sicherheit von Kryptosystemen häufig mit Zufallsexperimenten definiert, in denen ein Herausforderer und ein Angreifer interagieren. Der Herausforderer benutzt ein Kryptosystem, welches der Angreifer in diesem Experiment zu brechen versucht. Die Sicherheit des Verfahrens hängt mit der Erfolgswahrscheinlichkeit des Angreifers zusammen.

Für PKE soll hier das Experiment IND-CCA2 definiert werden (dabei steht IND für „indistinguishable“ und CCA2 für „adaptive chosen-ciphertext attack“). Es modelliert sowohl das zu erreichende Sicherheitsziel als auch die Fähigkeiten des Angreifers. Das Sicherheitsziel ist, dass kein Angreifer in der Lage sein soll, die Chiffretexte zweier Nachrichten seiner Wahl voneinander zu unterscheiden (IND). Die Fähigkeiten des Angreifers sind in diesem Fall sehr hoch. Er darf (fast) beliebige Chiffretexte seiner Wahl entschlüsseln lassen (CCA2).

Das Sicherheitsziel der Ununterscheidbarkeit hat den Vorteil, dass es sehr nützlich für Beweise ist – allerdings ist intuitiv nicht unbedingt klar, dass dies ein wünschenswertes Sicherheitsziel ist. Es ist jedoch im Falle eines adaptiven Chosen-Ciphertext-Angriffs (also wie bei IND-CCA2) äquivalent zu zwei Sicherheitszielen, die intuitiv besagen, dass der Angreifer zu einem gegebenen Chiffretext nicht in der Lage sein soll, irgendwelche Informationen über den Klartext herauszufinden und außerdem keinen zweiten Chiffretext finden soll, so dass die beiden zugehörigen Nachrichten in irgendeiner sinnvollen Beziehung zueinander stehen. Für genaue Definitionen und Beweise sei auf [BDPR98] verwiesen.

Das Experiment IND-CCA2 läuft wie folgt ab:

Setup: Der Herausforderer erzeugt ein Schlüsselpaar $(E, D) := \mathsf{Keygen}(1^k)$. Er gibt seinen öffentlichen Schlüssel E an den Angreifer weiter, während sein privater Schlüssel D geheim bleibt.

Phase 1: Der Angreifer darf beliebige Chiffretexte seiner Wahl entschlüsseln. Dazu stellt er mit Hilfe des Entschlüsselungsorakels Anfragen der Form $\mathsf{DecryptOracle}(c_i)$ an den Herausforderer, welcher $m_i := \mathsf{Decrypt}(c_i, D)$ zurückgeben muss.

Challenge: Zu einem Zeitpunkt seiner Wahl übergibt der Angreifer dem Herausforderer zwei Nachrichten $m_0, m_1 \in \mathcal{M}$ mit $m_0 \neq m_1$ und $|m_0| = |m_1|$ und erhält den Chiffretext von einer der beiden Nachrichten. Dazu wählt der Herausforderer mittels $b :=_R \{0, 1\}$ eine der beiden Nachrichten und gibt $c^* := \mathsf{Encrypt}(m_b, E)$ zurück.

Phase 2: Wie Phase 1, nur dass die Anfrage mit dem Challenge-Chiffretext c^* nicht erlaubt ist.

Guess: Der Angreifer äußert seine Vermutung $b' \in \{0, 1\}$, welche der beiden Nachrichten verschlüsselt wurde.

Der Angreifer gewinnt, wenn er mit seiner Vermutung richtig liegt, also $b = b'$ gilt. Ein Angreifer, der b' zufällig wählt (also „rät“), hat die Erfolgswahrscheinlichkeit $\frac{1}{2}$. Der

Vorteil eines Angreifers ist die Abweichung seiner eigenen Erfolgswahrscheinlichkeit von der beim Raten.

In der folgenden Definition soll dies formalisiert werden. Dabei ist $\mathcal{A}$ in zwei durch die Challengephase getrennte Teilalgorithmen zerlegt. $\mathcal{A}_{\mathrm{I}}$ arbeitet in Phase 1 und gibt neben den zwei Nachrichten m_0, m_1 auch Zustandsinformationen s aus, die $\mathcal{A}_{\mathrm{II}}$ (Phase 2) als zusätzliche Eingabe erhält. $\mathcal{A}_{\mathrm{I}}$ darf mittels dieser Zustandsinformationen alle Werte übermitteln, die $\mathcal{A}_{\mathrm{II}}$ zum Weiterarbeiten benötigt. Insbesondere könnte dies der öffentliche Schlüssel E, die beiden Nachrichten m_0, m_1 oder alle bisher beim $\mathsf{DecryptOracle}$ angefragten Chiffretexte mit der erhaltenen Antwort sein.

Im Folgenden deutet das $\mathsf{DecryptOracle}$ im Superskript von $\mathcal{A}_{\mathrm{I}}$ und $\mathcal{A}_{\mathrm{II}}$ an, dass den Algorithmen das Entschlüsselungsorakel $\mathsf{DecryptOracle}$ zur Verfügung steht.

Definition 5

Sei $\mathcal{A}$ ein zeitpolynomieller Angreifer auf ein PKE namens $\mathcal{PKE}$ in einem IND-CCA2-Experiment. Der *Vorteil von* $\mathcal{A}$ ist gegeben durch:

$$Adv^{IND-CCA2}_{\mathcal{A},\mathcal{PKE}}(1^k) := \left| Pr\left[\begin{array}{l} (E,D) := \mathsf{Keygen}(1^k); \\ (m_0, m_1, s) := \mathcal{A}_{\mathrm{I}}^{\mathsf{DecryptOracle}}(E); \\ b :=_R \{0,1\}; c^* := \mathsf{Encrypt}(m_b, E); \\ b' := \mathcal{A}_{\mathrm{II}}^{\mathsf{DecryptOracle}}(c^*, s) \end{array} : b = b' \right] - \frac{1}{2} \right|$$

Jetzt (und auch in folgenden Definitionen) sei $t := T(\mathcal{A})$ die Laufzeit von $\mathcal{A}$ polynomiell, q ein Polynom in k, welches die Anzahl der Anfragen an $\mathsf{DecryptOracle}$ beschränkt[4] und $\epsilon := Adv^{IND-CCA2}_{\mathcal{A},\mathcal{PKE}}(1^k)$ die Vorteilsfunktion. $\mathcal{A}$ wird dann auch (t, q, ϵ)-*IND-CCA2-Angreifer von* $\mathcal{PKE}$ genannt.

Ein PKE ist *IND-CCA2-sicher*, wenn für jeden Algorithmus $\mathcal{A}$ gilt:

$$\mathcal{A} \text{ ist } (t,q,\epsilon)\text{-IND-CCA2-Angreifer von } \mathcal{PKE} \Longrightarrow \epsilon \text{ ist vernachlässigbar}$$

In der Analyse wird es teilweise notwendig sein, eine noch feinere Zerlegung von $\mathcal{A}$ zu betrachten. $\mathcal{A}$ wird dann zwischen jeder Entschlüsselungsanfrage unterteilt. Der Teilalgorithmus $\mathcal{A}_i$ ist dann jeweils der *i-te Schritt* von $\mathcal{A}$ der und $\mathcal{A}_{\leq i}$ ist der Teil *bis zum i-ten Schritt*:

$$\begin{aligned} \mathcal{A}_{\leq 1}^{\mathsf{DecryptOracle}}(E) &:= \{(c_1, s_1) := \mathcal{A}_1(E)\} \\ \mathcal{A}_{\leq i}^{\mathsf{DecryptOracle}}(E) &:= \left\{ \begin{array}{l} (c_{i-1}, s_{i-1}) := \mathcal{A}_{\leq i-1}^{\mathsf{DecryptOracle}}(E); \\ m_{i-1} := \mathsf{DecryptOracle}(c_{i-1}); \\ (c_i, s_i) := \mathcal{A}_i(m_{i-1}, s_{i-1}) \end{array} \right\} \end{aligned}$$

[4] Die Anzahl der Anfragen ist natürlich schon durch die Laufzeit beschränkt, eine explizite obere Grenze der Anfragen ist aber für die Analyse nützlich.

c_i ist die *i-te Anfrage* von $\mathcal{A}$. s_i ist der *Zustand im i-ten Schritt*. Die Verteilung aller Eingabewerte E, c^*, s, s_i, m_i bilden zusammen die *Sicht* von $\mathcal{A}$. Mit dem *Verhalten* von $\mathcal{A}$ ist die Verteilung der Ausgabewerte $m_0, m_1, b', s, s_i, c_i$ gemeint.

Es gibt ein paar Abweichungen, nämlich beim Schritt vor der Challengephase, nach der Challengephase und ganz am Ende – hier ist die Ein- bzw. Ausgabe von $\mathcal{A}_i$ eine andere. $\mathcal{A}$ stellt für ein festes k maximal $q(1^k)$ Anfragen. Ohne Einschränkung kann jedoch auch angenommen werden, dass $\mathcal{A}$ zu jeder Eingabe genau $q := q(1^k)$ Anfragen stellt und die Challengephase nach genau $q' := q'(1^k)$ Anfragen beginnt. Dann sind diese drei Sonderfälle:

$$\mathcal{A}^{\mathsf{DecryptOracle}}_{\leq q'}(E) := \left\{ \begin{array}{l} (c_{q'-1}, s_{q'-1}) := \mathcal{A}^{\mathsf{DecryptOracle}}_{\leq q'-1}(E); \\ m_{q'-1} := \mathsf{DecryptOracle}(c_{q'-1}); \\ (m_0, m_1, s) := \mathcal{A}_{q'}(m_{q'-1}, s_{q'-1}) \end{array} \right\}$$

$$\mathcal{A}^{\mathsf{DecryptOracle}}_{\leq q'+1}(E) := \left\{ \begin{array}{l} (m_0, m_1, s) := \mathcal{A}^{\mathsf{DecryptOracle}}_{\leq q'}(E); \\ b :=_R \{0,1\}; c^* := \mathsf{Encrypt}(m_b, E); \\ (c_{q'+1}, s_{q'+1}) := \mathcal{A}_{q'+1}(c^*, s) \end{array} \right\}$$

$$\mathcal{A}^{\mathsf{DecryptOracle}}_{\leq q}(E) := \left\{ \begin{array}{l} (c_{q-1}, s_{q-1}) := \mathcal{A}^{\mathsf{DecryptOracle}}_{\leq q-1}(E); \\ m_{q-1} := \mathsf{DecryptOracle}(c_{q-1}); \\ b' := \mathcal{A}_q(m_{q-1}, s_{q-1}) \end{array} \right\}$$

Identitätsbasiertes Kryptosystem: Ein identitätsbasiertes Kryptosystem ist wie ein PKE ein asymmetrisches Verfahren. Auch hier hat jeder Benutzer einen öffentlichen und einen privaten Schlüssel – allerdings mit der Besonderheit, dass der öffentliche Schlüssel nicht frei gewählt wird, sondern mit der systemweit eindeutigen Identität des Benutzers (z.B. Mail-Adresse, Loginname oder Personalausweisnummer) übereinstimmt. Weiterhin werden die privaten Schlüssel nicht von jedem Benutzer selbst, sondern von einer zentralen Stelle generiert. Diese muss als erstes das Kryptosystem einrichten, indem sie öffentliche Systemparameter und einen geheimen Masterkey erzeugt. Mit Hilfe des Masterkeys lassen sich dann die privaten Schlüssel der Benutzer berechnen. Ver- und Entschlüsselung verlaufen analog zu PKE.

Definition 6

Ein *identitätsbasiertes Kryptosystem* (IBE) mit Identitäten der Länge l (wobei l ein Polynom in k ist) wird durch vier probabilistische zeitpolynomielle Algorithmen definiert: Setup, Keygen, Encrypt, Decrypt (siehe Abbildung 2).

$\mathsf{Setup}(1^k)$ bekommt als Eingabe den Sicherheitsparameter 1^k. Es wird der Klartextraum $\mathcal{M}$, der Chiffretextraum $\mathcal{C}$, der Schlüsselraum $\mathcal{K}$ und der Parameterraum $\mathcal{P}$ festgelegt und das Parameterpaar $(P, M) \in \mathcal{P}$ ausgegeben, wobei P die öffentlichen Systemparameter und M der geheime Masterkey ist.

$\mathsf{Keygen}(ID, M, P)$ bekommt als Eingabe die Identität $ID \in \{0,1\}^{l(k)}$ eines Benutzers,

sowie den Masterkey M und die Systemparameter P. Es wird das Schlüsselpaar $(ID, D_{ID}) \in \mathcal{K}$ ausgegeben.

$\mathsf{Encrypt}(m, ID, P)$ bekommt als Eingabe eine Nachricht $m \in \mathcal{M}$, die Identität ID des Empfängers sowie die Systemparameter P. Es wird der für ID verschlüsselte Chiffretext $c \in \mathcal{C}$ ausgegeben.

$\mathsf{Decrypt}(c, D_{ID}, P)$ bekommt als Eingabe den Chiffretext $c \in \mathcal{C}$, den privaten Schlüssel D_{ID} des Empfängers sowie die Systemparameter P. Es wird entweder die zuvor verschlüsselte Nachricht $m \in \mathcal{M}$ oder ein Fehlersymbol $\perp \notin \mathcal{M}$ ausgegeben.

Außerdem muss das IBE *korrekt* sein, d.h. für alle Parameter $(P, M) \in \mathcal{P}$, die von $\mathsf{Setup}(1^k)$ erzeugt wurden, für alle Schlüssel $(ID, D_{ID}) \in \mathcal{K}$, die von $\mathsf{Keygen}(ID, M, P)$ erzeugt wurden, und jede Nachricht $m \in \mathcal{M}$ muss gelten:

$$\mathsf{Decrypt}(\mathsf{Encrypt}(m, ID, P), D_{ID}, P) = m$$

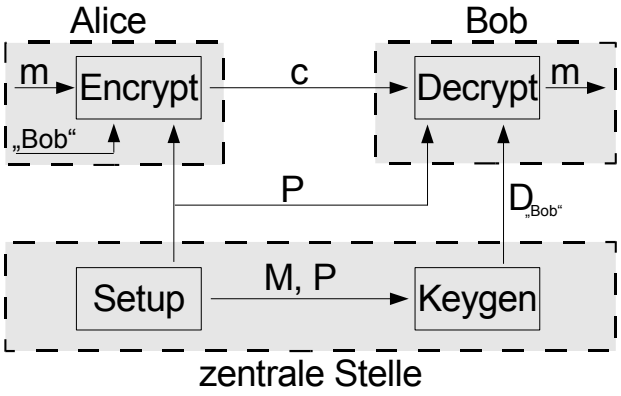

Abbildung 2: **Identitätsbasiertes Kryptosystem:** Alice möchte eine Nachricht an Bob verschlüsseln. Dazu muss zunächst die zentrale Stelle per Setup das Kryptosystem einrichten. Es werden der geheime Masterkey M und die öffentlichen Systemparameter P erzeugt. Letztere müssen an alle Benutzer (hier: Alice und Bob) übertragen werden. Nun kann Alice mit Encrypt, den Parametern P und der Identität „Bob“ eine Nachricht m verschlüsseln und den Chiffretext c an Bob senden. Dieser kann sich (falls noch nicht geschehen) bei der zentralen Stelle seinen privaten Schlüssel erzeugen lassen. Nur diese kann Keygen ausführen, da nur sie den geheimen Masterkey M kennt. Anschließend kann Bob die Nachricht m per Decrypt, den Parametern P und seinem privaten Schlüssel D_{Bob} wiederherstellen.

Für IBE soll hier die IND-sID-CPA-Sicherheit definiert werden (dabei steht IND wieder für „indistinguishable“, sID steht für „selective identity“ und CPA für „chosen-plaintext attack“). Das Sicherheitsziel ist wieder die Ununterscheidbarkeit von Chiffre-

texten (IND). Die Fähigkeiten des Angreifers sind diesmal allerdings geringer. Er kann lediglich zu Nachrichten seiner Wahl Chiffretexte erstellen (CPA). Während dies bei symmetrischer Kryptographie ein großer Vorteil ist, ist es für IBE nichts besonderes, da dazu nur die öffentlich bekannten Systemparameter und eine Identität notwendig ist. Aus Gründen der einheitlichen Benennung wird dennoch auch bei IBE diese „Fähigkeit“ des Angreifers aufgeführt.

Bei IBE basieren die privaten Schlüssel verschiedener Identitäten alle auf dem gemeinsamen Masterkey M, sie stehen also in einem mathematischen Zusammenhang. Ein Angreifer kann bereits mehrere private Schlüssel kennen, wenn er eine weitere Identität attackiert. Ein identitätsbasiertes Kryptosystem soll solch einem Angriff stand halten. Daher werden (im Gegensatz zu PKE) die Fähigkeiten des Angreifers noch erweitert. Er muss zu Beginn des Experiments eine anzugreifende Identität auswählen (sID), für die in der Challengephase eine Nachricht verschlüsselt wird. Zu allen anderen Identitäten darf sich der Angreifer von einem Orakel die privaten Schlüssel generieren lassen und sich somit beliebige, an diese Identitäten verschlüsselte Nachrichten entschlüsseln.

Es gibt noch weitere Sicherheitsdefinitionen, die andere Sicherheitsziele verfolgen oder dem Angreifer mehr Fähigkeiten zugestehen. Einen Überblick liefert [ACG+06]. Hier soll nur das oben beschriebene Experiment IND-sID-CPA definiert werden:

Setup: Der Angreifer benennt die anzugreifende Identität ID^*.
Der Herausforderer erzeugt ein Parameterpaar $(P, M) := \mathsf{Setup}(1^k)$. Er gibt die Parameter P an den Angreifer, während der Masterkey M geheim bleibt.

Phase 1: Der Angreifer darf zu jeder Identität $ID \neq ID^*$ die privaten Schlüssel generieren lassen. Dazu stellt er mit Hilfe des Orakels $\mathsf{KeygenOracle}(ID)$ eine Anfrage an den Herausforderer, welcher $(ID, D_{ID}) := \mathsf{Keygen}(ID, M, P)$ zurückgibt.

Challenge: Zu einem Zeitpunkt seiner Wahl übergibt der Angreifer dem Herausforderer zwei Nachrichten $m_0, m_1 \in \mathcal{M}$ mit $m_0 \neq m_1$ und $|m_0| = |m_1|$ und erhält den Chiffretext von einer der beiden Nachrichten zu der vorher bestimmten Identität ID^*. Dazu wählt der Herausforderer $b :=_R \{0, 1\}$ und gibt $c^* := \mathsf{Encrypt}(m_b, ID, P)$ zurück.

Phase 2: Wie Phase 1.

Guess: Der Angreifer äußert seine Vermutung $b' \in \{0, 1\}$, welche der beiden Nachrichten verschlüsselt wurde.

Bei diesem Experiment sind die Sicherheitsdefinitionen ganz analog zu denen von PKE. $\mathcal{A}$ wird allerdings nicht nur in $\mathcal{A}_{\mathrm{I}}$ und $\mathcal{A}_{\mathrm{II}}$ unterteilt, sondern es gibt auch noch den Teil $\mathcal{A}_0$, welcher vorab die anzugreifende Identität festlegt:

Definition 7

Sei $\mathcal{A}$ ein zeitpolynomieller Angreifer auf ein IBE namens $\mathcal{IBE}$ in einem IND-sID-CPA-Experiment. Der *Vorteil von* $\mathcal{A}$ ist gegeben durch:

$$Adv^{IND-sID-CPA}_{\mathcal{A},\mathcal{IBE}}(1^k) := \left| Pr\left[\begin{array}{l}(ID^*, s_0) := \mathcal{A}_0(1^k); (P, M) := \mathsf{Setup}(1^k);\\ (m_0, m_1, s) := \mathcal{A}_{\mathrm{I}}^{\mathsf{KeygenOracle}}(P, s_0);\\ b :=_R \{0,1\}; c^* := \mathsf{Encrypt}(m_b, ID^*, P);\\ b' := \mathcal{A}_{\mathrm{II}}^{\mathsf{KeygenOracle}}(c^*, s)\end{array} : b = b'\right] - \tfrac{1}{2}\right|$$

Wie in Definition 5 nennt man $\mathcal{A}$ einen (t, q, ϵ)-*IND-sID-CPA-Angreifer von* $\mathcal{IBE}$. Ein IBE ist *IND-sID-CPA-sicher*, wenn für jeden Algorithmus $\mathcal{A}$ gilt:

$$\mathcal{A} \text{ ist } (t, q, \epsilon)\text{-IND-sID-CPA-Angreifer von } \mathcal{IBE} \Longrightarrow \epsilon \text{ ist vernachlässigbar}$$

2.3 Zwei weitere kryptographische Primitive

Message-Authentication-Code: Ein Message-Authentication-Code dient der Integrität und Authentizität von Nachrichten. Der Sender einer Nachricht kann diese mit dem geheimen Schlüssel unterschreiben. Die Unterschrift kann vom Empfänger mit Hilfe des selben geheimen Schlüssels überprüft werden. Ein Message-Authentication-Code ist also die symmetrische Variante einer Signatur.

Definition 8

Ein *Message-Authentication-Code* (MAC) wird durch einen Klartextraum $\mathcal{M}$, einen Unterschriftenraum $\mathcal{S}$ und drei probabilistische zeitpolynomielle Algorithmen definiert: Init, Sign, Verify.

Init(1^k) bekommt als Eingabe den Sicherheitsparameter 1^k. Der Schlüssel $\kappa :=_R \{0,1\}^k$ wird erzeugt und ausgegeben.

Sign(m, κ) bekommt als Eingabe eine Nachricht $m \in \mathcal{M}$ und den Schlüssel κ. Es wird die Unterschrift $\sigma \in \mathcal{S}$ ausgegeben.

Verify(m, σ, κ) bekommt als Eingabe eine Nachricht $m \in \mathcal{M}$, die Unterschrift $\sigma \in \mathcal{S}$ und den Schlüssel $\kappa \in \{0,1\}^k$. Es wird ein Bit $b \in \{0,1\}$ ausgegeben. Dabei bedeutet 1, dass die Nachricht und Unterschrift zusammenpassen, und 0, dass diese nicht zusammenpassen.

Außerdem muss der MAC *korrekt* sein, d.h. für beliebige Schlüssel $\kappa \in \{0,1\}^k$ und jede Nachricht $m \in \mathcal{M}$ muss gelten:

$$\mathsf{Verify}(m, \mathsf{Sign}(m, \kappa), \kappa) = 1$$

An die Sicherheit eines MAC gibt es folgende Anforderung: Ein Angreifer soll nicht in der Lage sein, ohne Kenntnis des geheimen Schlüssels κ gültige Paare von Nachricht und Unterschrift zu erzeugen oder gegebene in andere gültige Paare abzuwandeln. Dabei hat er Zugriff auf ein SignOracle, welches ihm Klartexte seiner Wahl signiert.

Das Zufallsexperiment dazu ist sehr überschaubar, so dass die Definitionen direkt in kompakter Notation angegeben werden.

Definition 9

Sei $\mathcal{A}$ ein zeitpolynomieller Angreifer auf ein MAC namens $\mathcal{MAC}$. Der *Vorteil von* $\mathcal{A}$ ist gegeben durch:

$$Adv^{\text{Fälschung}}_{\mathcal{A},\mathcal{MAC}}(1^k) := Pr\left[\begin{array}{l|c}\kappa := \mathsf{Init}(1^k); & \mathsf{Verify}(m',\sigma',\kappa) = 1 \\ (m',\sigma') := \mathcal{A}^{\mathsf{SignOracle}}(1^k) & \wedge \forall i : (m',\sigma') \neq (m_i,\sigma_i)\end{array}\right]$$

Dabei sind (m_i, σ_i) die Paare der Anfragen und Antworten vom SignOracle.

Analog zu Definition 5 ist $\mathcal{A}$ ein (t, q, ϵ)-*Angreifer von* $\mathcal{MAC}$. Ein MAC ist *stark-fälschungssicher*, wenn für jeden Algorithmus $\mathcal{A}$ gilt:

$$\mathcal{A} \text{ ist } (t,q,\epsilon)\text{-Angreifer} \Longrightarrow \epsilon \text{ ist vernachlässigbar}$$

Man beachte, dass sich die von $\mathcal{A}$ gefundene Lösung von den Nachricht/Unterschrift-Paaren (m_i, σ_i), die das SignOracle liefert, nur in einer Komponente (Nachricht oder Unterschrift) unterscheiden muss. $\mathcal{A}$ muss damit also nicht notwendigerweise völlig neue unterschriebene Nachrichten erzeugen. Es reicht auch, wenn $\mathcal{A}$ zu einer unterschriebenen Nachricht eine neue Unterschrift oder zu einer Unterschrift eine neue Nachricht findet.

Encapsulation-Schema: In einem Commitment-Schema werden zu einer frei wählbaren Nachricht m ein Commitment-String *com* und ein Decommitment-String *dec* festgelegt. *com* liefert einerseits keine Informationen über m (Vertraulichkeit), legt aber gleichzeitig den Erzeuger von *com* und *dec* auf die Wahl der Nachricht m fest (Verbindlichkeit). Erst mit der zusätzlichen Information *dec* ist es möglich, m wiederherzustellen und zu überprüfen, auf welche Nachricht sich der Erzeuger von *com* und *dec* verbindlich festgelegt hat. In einem Encapsulation-Schema wird m nicht frei, sondern zufällig gewählt (und hier dann ρ wie „random“ genannt).

Definition 10

Ein *Encapsulation-Schema* (ENC) wird durch einen Parameterraum $\mathcal{P}$, einen Commitmentraum $\mathcal{C}$ und drei probabilistische zeitpolynomielle Algorithmen definiert: Start, Commit, Decommit.

$\mathsf{Start}(1^k)$ bekommt als Eingabe den Sicherheitsparameter 1^k. Es werden die öffentlichen Parameter $\pi \in \mathcal{P}$ ausgegeben.

$\mathsf{Commit}(1^k, \pi)$ bekommt als Eingabe die öffentlichen Parameter $\pi \in \mathcal{P}$. Es wird ein Tripel (ρ, com, dec) mit $\rho \in \{0,1\}^k$ und $(com, dec) \in \mathcal{C}$ ausgegeben.

$\mathsf{Decommit}(\pi, com, dec)$ bekommt als Eingabe die öffentlichen Parameter π und ein zu-

sammengehöriges Commitmenttupel $(com, dec) \in \mathcal{C}$. Es wird das $\rho \in \{0,1\}^k$ wiederhergestellt oder im Fehlerfall $\perp \notin \{0,1\}^k$ ausgegeben.

Außerdem muss das ENC *korrekt* sein, d.h. für beliebige Parameter $\pi \in \mathcal{P}$, die von $\mathsf{Start}(1^k)$ erzeugt wurden, und beliebige Tripel $(\rho, com, dec) \in \{0,1\}^k \times \mathcal{C}$, die von $\mathsf{Commit}(1^k, \pi)$ erzeugt wurden, muss gelten:

$$\mathsf{Decommit}(\pi, com, dec) = \rho$$

Das Schema soll „vertraulich“ und „verbindlich“ sein. Ersteres meint, dass π und com keine Informationen über ρ preisgeben sollen. Letzteres meint, dass man durch die Wahl von com auf ein einziges gültiges ρ festgelegt sein soll.

Definition 11

Sei $\mathcal{A}$ ein zeitpolynomieller Angreifer auf ein ENC namens $\mathcal{ENC}$. Der *Vorteil von* $\mathcal{A}$ *beim Angriff der Vertraulichkeit von* $\mathcal{ENC}$ ist gegeben durch:

$$Adv^{\text{Vertraulichkeit}}_{\mathcal{A},\mathcal{ENC}}(1^k) := \left| Pr\left[\begin{array}{l} \pi := \mathsf{Start}(1^k); \\ (\rho_0^*, com^*, dec^*) := \mathsf{Commit}(1^k, \pi); \\ \rho_1^* :=_R \{0,1\}^k; b :=_R \{0,1\}; \\ b' := \mathcal{A}(\pi, com^*, \rho_b^*) \end{array} : \quad b = b' \right] - \frac{1}{2} \right|$$

Analog zu Definition 5 ist $\mathcal{A}$ ein (t, ϵ)-*Angreifer der Vertraulichkeit von* $\mathcal{ENC}$.

Definition 12

Sei $\mathcal{A}$ ein zeitpolynomieller Angreifer auf ein ENC namens $\mathcal{ENC}$. Der *Vorteil von* $\mathcal{A}$ *beim Angriff der Verbindlichkeit von* $\mathcal{ENC}$ ist gegeben durch:

$$Adv^{\text{Verbindlichkeit}}_{\mathcal{A},\mathcal{ENC}}(1^k) := Pr\left[\begin{array}{l} \pi := \mathsf{Start}(1^k); \\ (\rho_0^*, com^*, dec^*) := \mathsf{Commit}(1^k, \pi); \\ dec' := \mathcal{A}(\pi, \rho^*, com^*, dec^*); \\ \rho' := \mathsf{Decommit}(\pi, com^*, dec') \end{array} : \quad \rho' \notin \{\rho^*, \perp\} \right]$$

Analog zu Definition 5 ist $\mathcal{A}$ ein (t, ϵ)-*Angreifer der Verbindlichkeit von* $\mathcal{ENC}$.

Definition 13

Ein ENC ist *sicher*, wenn für jeden Algorithmus $\mathcal{A}$ gilt:

$\mathcal{A}$ ist (t, ϵ)-Angreifer der Vertraulichkeit
oder (t, ϵ)-Angreifer der Verbindlichkeit von $\mathcal{ENC} \implies \epsilon$ ist vernachlässigbar

2.4 Bilineare Gruppen

Bilineare Gruppen bestehen aus zwei Gruppen, zwischen denen eine bilineare Abbildung existiert. Sie werden im Bereich der identitätsbasierten Kryptographie für zahlreiche Kryptosysteme verwendet. In Kapitel 4 wird ein solches IBE vorgestellt.

Definition 14

$\mathbb{G}$ sei eine zyklische Gruppe von Primordnung p, in der eines der erzeugenden Elemente $g \in \mathbb{G}^*$ ($:= \mathbb{G} \setminus \{1\}$) explizit gewählt ist. Dazu existiere eine weitere zyklische Gruppe $\mathbb{G}_T$ von Primordnung p und eine Abbildung $e : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$. Weiterhin gelte:

- e ist eine bilineare Abbildung, d.h. es gilt
$$e(x \cdot y, z) = e(x, z) \cdot e(y, z) \qquad e(x, y \cdot z) = e(x, y) \cdot e(x, z)$$
- e ist nicht entartet, d.h. $Bild(e) \neq \{1\}$
- es existieren zeitpolynomielle Algorithmen zur Berechnung der Bildwerte von e und der Gruppenoperationen in $\mathbb{G}$ und $\mathbb{G}_T$.

Dann ist $\mathbb{G}$ eine *bilineare Gruppe* mit der *zulässigen bilinearen Abbildung* e.

Der *bilineare Gruppengenerator* $\mathsf{Gen}(1^k)$ sei ein probabilistischer zeitpolynomieller Algorithmus, der zu einem Sicherheitsparameter k eine bilineare Gruppe $\mathbb{G}$ mit Primordnung p und eine bilineare Abbildung $e : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$ in Form des Viertupels $(p, \mathbb{G}, \mathbb{G}_T, e)$ ausgibt. Dabei kann der Sicherheitsparameter z.B. die Bit-Länge der Gruppenordnung p bestimmen.

Boneh und Franklin geben in [BF01, §5] die konkrete Konstruktion eines bilinearen Gruppengenerators an. Die bilineare Gruppe $\mathbb{G}$ basiert auf elliptischen Kurven über endlichen Körpern. Die bilineare Abbildung ist eine Modifikation der Weil- oder Tate-Paarung.

Oben wurde die allgemeine Definition für Bilinearität von Abbildungen verwendet. Da es sich bei $\mathbb{G}$ aber um eine zyklische Gruppe handelt, lässt sich die Bilinearität auch wie folgt charakterisieren:

Lemma 2.3

Sei $\mathbb{G}$ eine zyklische Gruppe der Ordnung n (nicht notwendigerweise prim) und g ein erzeugendes Element von $\mathbb{G}$. Für eine Abbildung $e : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$ gilt:

$$e \text{ ist bilinear} \iff \forall \alpha, \beta \in \mathbb{N} : e(g^\alpha, g^\beta) = e(g, g)^{\alpha\beta}$$

(Die Richtung von links nach rechts gilt unabhängig davon, ob $\mathbb{G}$ zyklisch ist.)

Beweis

„⇒“: Der Beweis erfolgt per Induktion über das Maximum von α und β.
Für den Induktionsanfang fasse man e nicht als Abbildung mit zwei Parametern aus $\mathbb{G}$ auf, sondern mit einem Parameter aus der Produktgruppe $\mathbb{G} \times \mathbb{G}$. Dann ist e ein Gruppenhomomorphismus von $\mathbb{G} \times \mathbb{G}$ nach $\mathbb{G}_T$ und somit $e((g^0, g^0)) = e((1,1)) \overset{(\star)}{=} 1 = e((g,g))^0$. Dabei gilt $(\star)$, da ein Gruppenhomomorphismus neutrale Elemente immer auf neutrale Elemente abbildet.
Nun fasse man e wieder als bilineare Abbildung auf. Es gilt

$$\begin{aligned}
\boxed{e(g^\alpha, g^\beta)} &= e(g^{\alpha-1}, g^\beta) \cdot e(g, g^\beta) \\
&= e(g^{\alpha-1}, g^{\beta-1}) \cdot e(g^{\alpha-1}, g) \cdot e(g, g^{\beta-1}) \cdot e(g,g) \\
&\overset{\text{Indukt.}}{=} e(g,g)^{(\alpha-1)(\beta-1)} \cdot e(g,g)^{\alpha-1} \cdot e(g,g)^{\beta-1} \cdot e(g,g) \\
&= e(g,g)^{(\alpha-1)(\beta-1)+(\alpha-1)+(\beta-1)+1} = \boxed{e(g,g)^{\alpha\beta}}
\end{aligned}$$

„⇐“: Seien $x, y, z \in \mathbb{G}$ beliebig. Da g ein erzeugendes Element von $\mathbb{G}$ ist, gibt es Exponenten $\alpha, \beta, \gamma \in \{1, \ldots, p\}$ mit $x = g^\alpha, y = g^\beta, z = g^\gamma$. Dann gilt:

$$\begin{aligned}
\boxed{e(x \cdot y, z)} &= e(g^\alpha \cdot g^\beta, g^\gamma) = e(g,g)^{(\alpha+\beta)\cdot\gamma} \\
&= e(g,g)^{\alpha\gamma} \cdot e(g,g)^{\beta\gamma} = e(g^\alpha, g^\gamma) \cdot e(g^\beta, g^\gamma) = \boxed{e(x,z) \cdot e(y,z)}
\end{aligned}$$

Die zweite Bedingung der Bilinearität folgt analog.

Sicherheit: In allgemeinen Gruppen gibt es einige verbreitete Sicherheitsannahmen:

- Problem des diskreten Logarithmus: „Berechne zu einem erzeugenden Element $g \in \mathbb{G}^*$ und einem zufälligen Element $x \in \mathbb{G}$ einen Exponenten $\alpha \in \mathbb{Z}_{|\mathbb{G}|}$, so dass $x = g^\alpha$ gilt.“
- Diffie-Hellman-Problem: „Berechne zu einem erzeugenden Element $g \in \mathbb{G}^*$ und zufälligen Elementen $x, y \in \mathbb{G}$ ein Element $z \in \mathbb{G}$, so dass (g, x, y, z) ein Diffie-Hellman-Tupel ist, d.h. von der Form $(g, g^\alpha, g^\beta, g^{\alpha\beta})$ mit Exponenten $\alpha, \beta \in \mathbb{Z}_{|\mathbb{G}|}$.“
- Diffie-Hellman-Entscheidungsproblem: „Entscheide zu einem erzeugenden Element $g \in \mathbb{G}^*$, zufälligen Elementen $x, y \in \mathbb{G}$ und einem weiteren Element $z \in \mathbb{G}$, ob (g, x, y, z) ein Diffie-Hellman-Tupel ist.“

Die letzteren beiden lassen sich auf bilineare Gruppen übertragen. Sie lauten dann:

- bilineares Diffie-Hellman-Problem: „Berechne zu dem erzeugenden Element $g \in \mathbb{G}^*$ und zufälligen Elementen $x, y, z \in \mathbb{G}$ ein Element $T \in \mathbb{G}_T$, so dass (g, x, y, z, T) ein

bilineares Diffie-Hellman-Tupel ist, d.h. von der Form $(g, g^\alpha, g^\beta, g^\gamma, e(g,g)^{\alpha\beta\gamma})$ mit Exponenten $\alpha, \beta, \gamma \in \mathbb{Z}_{|\mathbb{G}|}$.“

- bilineares Diffie-Hellman-Entscheidungsproblem: „Entscheide zu einem erzeugenden Element $g \in \mathbb{G}^*$, zufälligen Elementen $x, y, z \in \mathbb{G}$ und einem weiteren Element $T \in \mathbb{G}_T$, ob (g, x, y, z, T) ein bilineares Diffie-Hellman-Tupel ist.“

Da das letzte dieser Probleme im Folgenden benötigt wird, soll es formalisiert werden:

Definition 15

Sei Gen ein bilinearer Gruppengenerator. Ein zeitpolynomieller Angreifer $\mathcal{A}$ hat folgenden *Vorteil beim Lösen des bilinearen Diffie-Hellman-Entscheidungsproblem (BDDHP[5]) in den von* Gen *erzeugten Gruppen*:

$$Adv^{BDDHP}_{\mathcal{A},\mathsf{Gen}}(1^k) := \left| Pr \left[\begin{array}{l} (p, \mathbb{G}, \mathbb{G}_T, e) := \mathsf{Gen}(1^k); g :=_R \mathbb{G}^*; \\ \alpha, \beta, \gamma :=_R \{1, \ldots, p\}; (x, y, z) := (g^\alpha, g^\beta, g^\gamma); \\ T_0 := e(g,g)^{\alpha\beta\gamma}; T_1 :=_R \mathbb{G}_T; b := \{0,1\}; \\ b' := \mathcal{A}(p, \mathbb{G}, \mathbb{G}_T, e, g, x, y, z, T_b) \end{array} : b = b' \right] - \frac{1}{2} \right|$$

Wie in Definition 5 nennt man $\mathcal{A}$ einen (t, ϵ)-*BDDHP-Angreifer von* Gen. Das BDDHP ist *hart* für die von Gen erzeugten Gruppen, wenn für jeden Algorithmus $\mathcal{A}$ gilt:

$$\mathcal{A} \text{ ist } (t, \epsilon)\text{-BDDHP-Angreifer von } \mathsf{Gen} \Longrightarrow \epsilon \text{ ist vernachlässigbar}$$

[5] BDDHP steht für „bilinear decision Diffie-Hellman problem“.

3 Das allgemeine Verfahren für PKE aus beliebigem IBE

In diesem Kapitel soll vorgestellt werden, wie aus einem beliebigen IND-sID-CPA-sicheren IBE und einigen weiteren kryptographischen Primitiven ein PKE definiert werden kann, welches IND-CCA2-sicher ist. Zunächst wird in Abschnitt 3.1 die grundlegende Idee erläutert. Anschließend wird in Abschnitt 3.2 mit dieser Idee ein korrektes PKE definiert. Zuletzt wird in Abschnitt 3.3 die IND-CCA2-Sicherheit bewiesen.

3.1 Die Idee

Zunächst soll noch einmal die Schlüsselerzeugung und -verteilung bei IBE etwas näher beleuchtet werden:

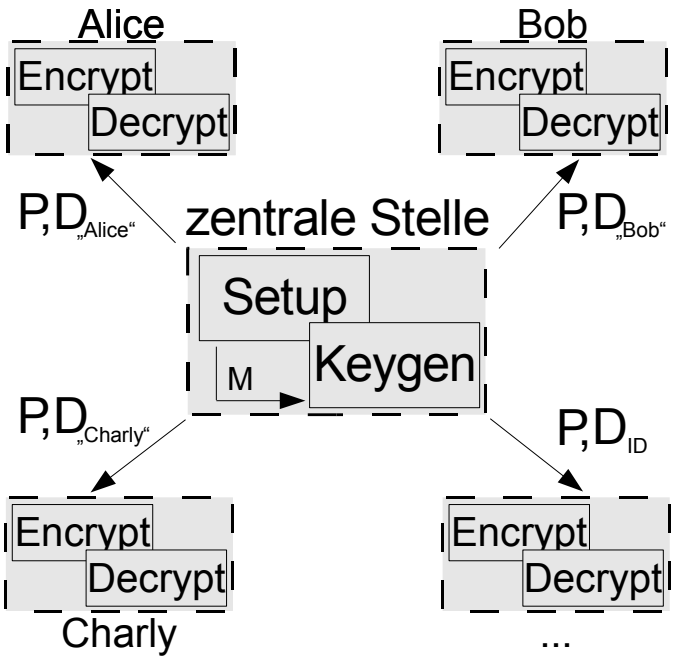

Abbildung 3: Schlüsselerzeugung und -verteilung bei IBE

Es gibt eine einzige zentrale Stelle, welche einmalig das Kryptosystem einrichtet (Setup) und anschließend ausschließlich dafür zuständig ist, die Schlüssel der Benutzer zu erzeugen (Keygen). Jeder Benutzer des Kryptosystems erhält die öffentlichen Parameter und seinen privaten Schlüssel D_{ID}. Mit diesen Informationen und der Identität des Empfängers können die Benutzer Nachrichten ver- und entschlüsseln (Encrypt, Decrypt).
Es gibt also nur ein einziges globales Paar von öffentlichen Parametern P und geheimen Masterkey M und jeder Benutzer hat eine Identität mit zugehörigem privaten Schlüssel.

Weiterhin herrscht eine strikte Trennung zwischen Schlüsselerzeugung (zentrale Stelle) und Nachrichtenverschlüsselung (Benutzer).

Um daraus ein PKE zu erzeugen, wird diese identitätsbasierte Struktur verworfen und nur die Algorithmen übernommen. Es gibt keine zentrale Stelle mehr, welche die Schlüssel erzeugt. Vielmehr verwendet jeder Benutzer die Algorithmen Setup und Keygen selbständig und jeder besitzt ein eigenes Paar von öffentlichen Systemparametern (im Bild: P, P', P'') und Masterkey (im Bild: M, M', M''). Die bei der Verschlüsselung verwendeten Identitäten sind nicht mehr an reale Identitäten (z.B. Mail-Adresse, Loginname oder Personalausweisnummer) gebunden, sondern nur noch „bedeutungslose“ zufällig gewählte Zeichenketten.

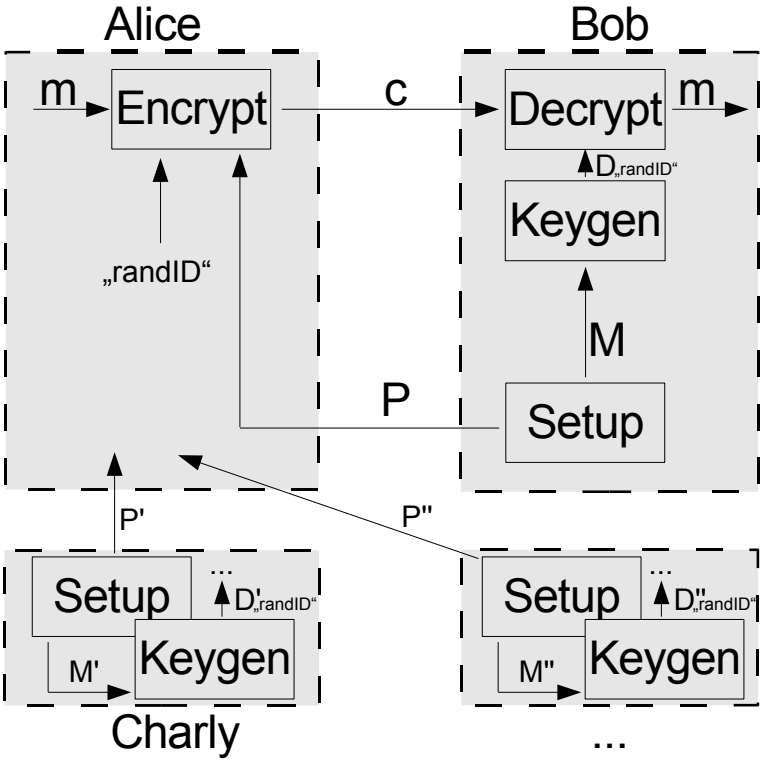

Abbildung 4: Grundsätzliche Struktur bei dem neuen PKE

Das PKE funktioniert wie folgt: Jeder Benutzer muss bei der Schlüsselerzeugung den Setup-Algorithmus des IBE ausführen. Die öffentlichen Systemparameter P werden Teil des öffentlichen Schlüssels und der Masterkey M wird Teil des privaten Schlüssels. Der zentrale Kern der Verschlüsselung ist das Ausführen des Encrypt-Algorithmus von IBE. Es wird dazu eine (fast) zufällige Identität „randID“ ausgewählt, an die verschlüsselt wird. Mit Hilfe anderer kryptographischer Primitive werden die Chiffretexte so erweitert, dass sie sich in „gültig“ und „ungültig“ unterteilen lassen. Bei der Entschlüsselung muss

nun zunächst die Gültigkeit geprüft werden und dann mit Hilfe des Decrypt-Algorithmus von IBE die eigentliche Nachricht wiederhergestellt werden. Dies ist möglich, da der Besitzer des privaten Schlüssels unter anderem den Masterkey kennt und sich dann mit Hilfe des Keygen-Algorithmus von IBE zu jeder Identität den privaten Schlüssel $D_{\text{„randID“}}$ erzeugen kann.

Wie erwartet, ist die Sicherheit des IBE von zentraler Bedeutung für die Sicherheit des PKE. Der Sicherheitsbeweis erfolgt per Reduktion. Ein Angriff auf das PKE soll in einen Angriff „gleicher Güte“ (d.h. unter Beibehalt des Vorteils) gegen IBE gewandelt werden. Sei dazu $\mathcal{A}$ ein IND-CCA2-Angreifer des PKE. Es wird für das IBE ein IND-sID-CPA-Angreifer $\mathcal{B}$ konstruiert, der für $\mathcal{A}$ den Herausforderer simuliert und sich durch das Verhalten von $\mathcal{A}$ einen Vorteil für seinen eigenen Angriff verschafft:

$\mathcal{B}$ legt zunächst für den eigenen Angriff die anzugreifende Identität ID^* fest und bekommt dann die öffentlichen Systemparameter des IBE. Diese werden nach Vorgaben der Schlüsselerzeugung des PKE zu einem gültigen öffentlichen Schlüssel erweitert. Der private Schlüssel hingegen ist $\mathcal{B}$ zu Teilen unbekannt, da der Masterkey des IBE unbekannt ist. Die Nachrichten m_0 und m_1 sowie die Vermutung b' werden von $\mathcal{B}$ im Wesentlichen an den eigenen Herausforderer durchgereicht. Es wird sich herausstellen, dass genau wenn $\mathcal{A}$ mit der Vermutung b' richtig liegt, auch $\mathcal{B}$ gewinnt.
Problematischer ist, dass $\mathcal{B}$ für $\mathcal{A}$ das DecryptOracle bereitstellen muss, selbst aber auf keines zurückgreifen kann. Allerdings darf $\mathcal{B}$ ein KeygenOracle benutzen und kann damit fast jede Anfrage von $\mathcal{A}$ in gültiger Weise beantworten. Wichtig ist dabei, an welche Identität ID verschlüsselt wurde:

- Falls $ID \neq ID^*$ ist, kann $\mathcal{B}$ mit dem KeygenOracle den zu ID gehörigen privaten Schlüssel D_{ID} bestimmen und dann die Entschlüsselung wie definiert vornehmen.
- Falls $ID = ID^*$ ist, funktioniert diese Vorgehensweise allerdings nicht, da $\mathcal{B}$ keine Anfragen an das KeygenOracle mit der angegriffenen ID^* stellen darf. $\mathcal{B}$ kann die Nachricht also nicht entschlüsseln. Daher bleibt $\mathcal{B}$ in diesem Fall nur die Möglichkeit $\perp$ zurückzugeben – also zu behaupten, dass der Chiffretext „ungültig“ ist.

Falls der Chiffretext wirklich ungültig war, stellt dies kein Problem dar. Falls er aber gültig war, hat $\mathcal{B}$ sich nicht korrekt verhalten. Ziel ist es also, das PKE so zu konstruieren, dass eine Anfrage von $\mathcal{A}$ zur Entschlüsselung eines **gültigen Chiffretextes mit ID = ID* nur mit geringer Wahrscheinlichkeit** auftreten kann. Dann liefert $\mathcal{B}$ nämlich „fast immer“ eine perfekte Simulation für $\mathcal{A}$.

3.2 Die Umsetzung

Die oben beschriebene Idee geht auf Canetti et al. zurück und wird in [CHK04] beschrieben. Sie implementieren diese Idee mit IBE und Signaturen. Hier soll allerdings eine effizientere Konstruktion von Boneh und Katz aus [BK05] vorgestellt werden. Seien dazu ein IND-sID-CPA-sicheres $\mathcal{IBE} = (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Encrypt}, \mathsf{Decrypt})$, ein sicheres $\mathcal{ENC} =$

$(\mathsf{Start}, \mathsf{Commit}, \mathsf{Decommit})$ und ein stark-fälschungssicherer $\mathcal{MAC} = (\mathsf{Init}, \mathsf{Sign}, \mathsf{Verify})$ gegeben. Definiert wird $\mathcal{PKE} = (\mathsf{Keygen}^*, \mathsf{Encrypt}^*, \mathsf{Decrypt}^*)$.

$\mathsf{Keygen}^*(1^k)$**:** Zur Schlüsselerzeugung werden die initialen Algorithmen $(P, M) := \mathsf{Setup}(1^k)$ und $\pi := \mathsf{Start}(1^k)$ ausgeführt. Der Schlüssel ist $(E, D) := ((P, \pi), M)$.

$\mathsf{Encrypt}^*(m, E)$**:** Zunächst wird $(\rho, com, dec) := \mathsf{Commit}(1^k, \pi)$ bestimmt. ρ wird als Unterschriftenschlüssel für $\mathcal{MAC}$ dienen. Der Empfänger wird dec benötigen, um ρ wiederherzustellen. Daher wird dec an die eigentliche Nachricht angehängt und mit dem $\mathcal{IBE}$ an die „Identität“ com verschlüsselt: $\overline{c} := \mathsf{Encrypt}(m.dec, com, P)$. Zuletzt wird dieser Chiffretext mit ρ unterschrieben: $\sigma := \mathsf{Sign}(\overline{c}, \rho)$. Alles zusammen wird als Chiffretext $c := (com, \overline{c}, \sigma)$ zurückgegeben.

$\mathsf{Decrypt}^*(c, D)$**:** Der Chiffretext wird als $c = (com, \overline{c}, \sigma)$ eingelesen. Zunächst muss der private Schlüssel zur „Identität“ com bestimmt werden: $D_{com} := \mathsf{Keygen}(com, M, P)$. Dann wird die Nachricht wiederhergestellt: $m'.dec' := \mathsf{Decrypt}(\overline{c}, D_{com}, P)$. Nun kann der Unterschriftenschlüssel bestimmt werden: $\rho' := \mathsf{Decommit}(\pi, com, dec')$. Zuletzt wird die Gültigkeit des Chiffretextes überprüft: $\mathsf{Verify}(\overline{c}, \sigma, \rho')$. Falls einer der Zwischenschritte $\perp$ ausgegeben hat oder die Ausgabe von Verify 0 war, so wird der Chiffretext abgelehnt und $\perp$ zurückgegeben, ansonsten m'.

Satz 3.1

Dieses $\mathcal{PKE}$ ist korrekt.

Beweis

Die Korrektheit nach Definition 4 folgt direkt aus der von $\mathcal{IBE}$, $\mathcal{ENC}$ und $\mathcal{MAC}$ (siehe Definitionen 6, 10 und 8).

Sei $((P, \pi), M)$ ein von Keygen^* erzeugtes Schlüsselpaar und m eine beliebige Nachricht. Es ist zu zeigen:

$$\mathsf{Decrypt}^*(\mathsf{Encrypt}^*(m, E), D) = m$$

(P, M) wurde von Setup erzeugt. Nach der Korrektheit von $\mathcal{IBE}$ gilt dann für die „Nachricht“ $m.dec$ und das von Keygen erzeugte $\mathcal{IBE}$-Schlüsselpaar (com, D_{com}):

$$m'.dec' := \mathsf{Decrypt}(\mathsf{Encrypt}(m.dec, com, P), D_{com}, P) = m.dec \neq \perp$$

π wurde von Start und (ρ, com, dec) von $\mathsf{Commit}(1^k, \pi)$ erzeugt. Nach der Korrektheit von $\mathcal{ENC}$ gilt dann:

$$\rho' := \mathsf{Decommit}(\pi, com, \underbrace{dec'}_{= dec}) = \rho \neq \perp$$

Zuletzt gilt mit der Korrektheit von $\mathcal{MAC}$ für den Schlüssel ρ und die „Nachricht“ $\overline{c}$:

$$\mathsf{Verify}(\overline{c}, \mathsf{Sign}(\overline{c}, \rho), \underbrace{\rho'}_{= \rho}) = 1$$

Der Chiffretext wird also als gültig akzeptiert und $m' = m$ zurückgegeben.

3.3 Beweis der IND-CCA2 Sicherheit

Wie angekündigt erfolgt der Beweis per Reduktion. Es werden Angreifer von $\mathcal{IBE}$, $\mathcal{ENC}$ und $\mathcal{MAC}$ definiert, die einen Angreifer von $\mathcal{PKE}$ zu ihrem eigenen Vorteil nutzen:

Satz 3.2

Sei $\mathcal{A}$ ein (t, q, ϵ)-IND-CCA2-Angreifer von $\mathcal{PKE}$, dann lassen sich mit $\mathcal{A}$ folgende Algorithmen konstruieren:

$\mathcal{B}_1$ ein (t_1, q, ϵ_1)-IND-sID-CPA-Angreifer von $\mathcal{IBE}$

$\mathcal{B}_2$ ein (t_2, ϵ_2)-Angreifer der Verbindlichkeit von $\mathcal{ENC}$

$\mathcal{B}_3$ ein (t_3, q, ϵ_3)-IND-sID-CPA-Angreifer von $\mathcal{IBE}$

$\mathcal{B}_4$ ein (t_4, ϵ_4)-Angreifer der Vertraulichkeit von $\mathcal{ENC}$

$\mathcal{B}_5$ ein $(t_5, 1, \epsilon_5)$-Angreifer von $\mathcal{MAC}$

wobei alle t_i polynomiell beschränkt sind und $\epsilon \leq \epsilon_1 + 2 \cdot \epsilon_2 + 4 \cdot \epsilon_3 + 4 \cdot \epsilon_4 + 2q \cdot \epsilon_5$ ist.

Folgerung 3.3

Aus Satz 3.2 und der Sicherheit von $\mathcal{IBE}$, $\mathcal{ENC}$ und $\mathcal{MAC}$ folgt unmittelbar die Sicherheit von $\mathcal{PKE}$. ϵ ist durch vernachlässigbare Funktionen ϵ_i beschränkt, und somit nach Lemma 2.2 selbst auch vernachlässigbar.

Der Beweis erfolgt durch eine Reihe von jeweils leicht abgeänderten Zufallsexperimenten, mit Hilfe derer der Angriff von $\mathcal{A}$ auf $\mathcal{PKE}$ schrittweise in Angriffe auf $\mathcal{IBE}$, $\mathcal{MAC}$ und $\mathcal{ENC}$ umgewandelt werden kann. Dabei werden zuerst alle Experimente, wichtige Zufallsereignisse und alle Lemmata ohne Beweis formuliert, um einen Überblick über die Beweisführung zu geben. Anschließend werden die einzelnen Lemmata bewiesen.

Die Experimente sind von 0 bis 3 durchnummeriert. Sie haben jeweils etwa die Form:

$$E_i^{\mathcal{A}}(1^k) := \left\{ \begin{array}{l} (E, D) := Keygen_i(1^k); \\ (m_0, m_1, s) := \mathcal{A}_{\mathrm{I}}^{DecryptOracle_i}(E); \\ c^* := Challenge_i(m_0, m_1); \\ b' := \mathcal{A}_{\mathrm{II}}^{DecryptOracle_i}(c^*, s) \end{array} \right\}$$

wobei jeweils die Teilalgorithmen $Keygen_i, DecryptOracle_i, Challenge_i$ für jedes Experiment individuell definiert werden.

Ausgangspunkt ist das IND-CCA2-Experiment, bei dem man die konkreten Algorithmen $\mathsf{Keygen}^*, \mathsf{Encrypt}^*$ und $\mathsf{Decrypt}^*$ einsetzt:

$$Keygen(1^k) := \left\{ \begin{array}{l} (P, M) := \mathsf{Setup}(1^k); \\ \pi := \mathsf{Start}(1^k); \\ (E, D) := ((P, \pi), M) \end{array} \right\}$$

$$\begin{array}{l} DecryptOracle(com_i, \overline{c}_i, \sigma_i) \\ = \ \mathsf{Decrypt}^*((com_i, \overline{c}_i, \sigma_i), M, P) \end{array} := \left\{ \begin{array}{l} D_{com_i} := \mathsf{Keygen}(com_i, M, P); \\ m_i.dec_i := \mathsf{Decrypt}(\overline{c}_i, D_{com_i}, P); \\ \rho_i := \mathsf{Decommit}(\pi, com_i, dec_i) \\ v_i := \mathsf{Verify}(\overline{c}, \sigma, \rho') \\ \text{Antwort: } m_i, \text{ falls } m_i.dec_i \neq \bot,\ \rho_i \neq \bot,\ v_i = 1 \\ \text{sonst } \bot. \end{array} \right\}$$

$$Challenge(m_0, m_1) := \left\{ \begin{array}{l} b :=_R \{0, 1\}; \\ (\rho^*, com^*, dec^*) := \mathsf{Commit}(1^k, \pi); \\ \overline{c}^* := \mathsf{Encrypt}(m_b.dec^*, com^*, P); \\ \sigma := \mathsf{Sign}(\overline{c}^*, \rho^*); \\ c^* := (com^*, \overline{c}^*, \sigma^*) \end{array} \right\}$$

Experiment 0: Dieses Experiment entspricht nahezu exakt dem IND-CCA2-Experiment. Nur $(\rho^*, com^*, dec^*) := \mathsf{Commit}(1^k, \pi)$ wird statt im Challenge- schon im Keygen-Schritt ausgeführt. Dies hat keinerlei Auswirkungen auf das Experiment an sich, ist allerdings notwendig, damit das im Folgenden eingeführte Zufallsereignis VALID wohldefiniert ist.

$$Keygen_0(1^k) := \left\{ \begin{array}{l} (P, M) := \mathsf{Setup}(1^k); \\ \pi := \mathsf{Start}(1^k); \\ \boxed{(\rho^*, com^*, dec^*) := \mathsf{Commit}(1^k, \pi);} \\ (E, D) := ((P, \pi), M) \end{array} \right\}$$

$$DecryptOracle_0(com_i, \overline{c}_i, \sigma_i) := DecryptOracle(com_i, \overline{c}_i, \sigma_i)$$

$$Challenge_0(m_0, m_1) := \left\{ \ldots;\ \cancel{(\rho^*, com^*, dec^*) := \mathsf{Commit}(1^k, \pi);}\ \ldots \right\}$$

WIN: $b = b'$

„Die Vermutung von $\mathcal{A}$ ist richtig“. Man vergleiche dies mit Definition 5. Es ist $Adv^{IND-CCA2}_{\mathcal{A},\mathcal{PKE}}(1^k) = \left| Pr[E_0^{\mathcal{A}}(1^k) : \mathsf{WIN}] - \frac{1}{2} \right|$.

VALID: $\exists i \in \{1, \ldots, q(1^k)\} : \quad com_i = com^* \ \wedge \ DecryptOracle_0(com_i, \overline{c}_i, \sigma_i) \neq \bot$

„In irgendeiner Anfrage von $\mathcal{A}$ an $DecryptOracle_0$ wird ein gültiger Chiffretext mit $com_i = com^*$ übermittelt.“

Man beachte: hätte man nicht obige kleine Änderung gegenüber dem eigentlichen IND-CCA2-Experiment vorgenommen, so wäre dieses Ereignis nicht wohldefiniert, da com^* bei den Anfragen an $DecryptOracle_0$, die vor der Challenge-Phase gemacht werden, noch undefiniert wäre.

Experiment 1: spiegelt genau den in Abschnitt 3.1 skizzierten Angriff auf $\mathcal{IBE}$ wieder. Es ist identisch mit Experiment 0, außer dass Anfragen an das $DecryptOracle_1$ mit $com_i = com^*$ immer mit $\perp$ beantwortet werden.

$$Keygen_1(1^k) := Keygen_0(1^k) \qquad Challenge_1(m_0, m_1) := Challenge_0(m_0, m_1)$$

$$DecryptOracle_1(com_i, \overline{c}_i, \sigma_i) := \left\{ \begin{array}{l} \text{Falls } com_i \neq com^*\text{: siehe } DecryptOracle_0 \\ \text{Falls } \boxed{com_i = com^*}\text{: } \perp \end{array} \right\}$$

Man beachte, dass Experiment 0 und 1 identisch ablaufen, solange VALID nicht eintritt. Falls VALID aber eintritt, so antwortet $DecryptOracle_0$ mit einem $m_i \in \mathcal{M}$ und $DecryptOracle_1$ mit $\perp \notin \mathcal{M}$. Diese Beobachtung nutzt

Lemma 3.4

Sei $\mathcal{A}$ ein (t, q, ϵ)-IND-CCA2-Angreifer von $\mathcal{PKE}$, dann gilt

$$\left| Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{WIN}\right] - \frac{1}{2} \right| = \left| Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{WIN}\right] - \frac{1}{2} \right| + 2 \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}\right]$$

Nun kann ein IND-sID-CPA-Angreifer auf $\mathcal{IBE}$ $\mathcal{A}$ zu seinem Vorteil nutzen.

Lemma 3.5

Sei $\mathcal{A}$ ein (t, q, ϵ)-IND-CCA2-Angreifer von $\mathcal{PKE}$, dann gibt es einen Algorithmus $\mathcal{B}_1$, der ein (t_1, q, ϵ_1)-IND-sID-CPA-Angreifer von $\mathcal{IBE}$ ist. Dabei ist t_1 polynomiell beschränkt und

$$\epsilon_1(1^k) := Adv^{IND-sID-CPA}_{\mathcal{B}_1, \mathcal{IBE}}(1^k) = \left| Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{WIN}\right] - \frac{1}{2} \right|$$

Damit ist der erste Teil der Erfolgswahrscheinlichkeit von $\mathcal{A}$ beschränkt. Nun muss die Wahrscheinlichkeit, dass VALID eintritt, abgeschätzt werden. Man beachte, dass $DecryptOracle_0(com_i, \overline{c}_i, \sigma_i) \neq \perp$ genau dann gilt, wenn die Entschlüsselung, das Decommitment und die Überprüfung der Unterschrift ohne Fehler abläuft. Damit lässt sich VALID disjunkt aufteilen in:

$\exists i \in \{1, \ldots, q(1^k)\} : com_i = com^*,$
$m_i.dec_i := \mathsf{Decrypt}(\overline{c}_i, D_{\underset{= com^*}{com_i}}, P) \neq \perp,$
$\rho_i := \mathsf{Decommit}(\pi, \underset{= com^*}{com_i}, dec_i) \neq \perp,$
$\boxed{\rho_i \neq \rho^*}, \mathsf{Verify}(\overline{c}_i, \sigma_i, \rho_i) = 1$

$\exists i \in \{1, \ldots, q(1^k)\} : com_i = com^*,$
$m_i.dec_i := \mathsf{Decrypt}(\overline{c}_i, D_{\underset{= com^*}{com_i}}, P) \neq \perp,$
$\rho_i := \mathsf{Decommit}(\pi, \underset{= com^*}{com_i}, dec_i) \neq \perp,$
$\boxed{\rho_i = \rho^*}, \mathsf{Verify}(\overline{c}_i, \sigma_i, \underset{= \rho^*}{\rho_i}) = 1$

Die folgenden beiden Ereignisse umfassen die obige disjunkte Zerlegung:

NOBIND:	FORGE:
$\exists i \in \{1, \ldots, q(1^k)\} : com_i = com^*$, $m_i.dec_i := \mathsf{Decrypt}(\overline{c}_i, D_{com^*}, P) \neq \perp$, $\rho_i := \mathsf{Decommit}(\pi, com^*, dec_i) \neq \perp$, $\rho_i \neq \rho^*$	$\exists i \in \{1, \ldots, q(1^k)\} : com_i = com^*$, $\mathsf{Verify}(\overline{c}_i, \sigma_i, \rho^*) = 1$

Beobachtung 3.6

Die Wahrscheinlichkeit, dass VALID eintritt, ist beschränkt durch

$$Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}\right] \leq Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{NOBIND}\right] + Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right]$$

Das Eintreten von NOBIND bedeutet aber gerade, dass die Verbindlichkeit von $\mathcal{ENC}$ nicht gegeben ist:

Lemma 3.7

Sei $\mathcal{A}$ ein (t, q, ϵ)-IND-CCA2-Angreifer von $\mathcal{PKE}$, dann gibt es einen Algorithmus $\mathcal{B}_2$, der ein (t_2, ϵ_2)-Angreifer der Verbindlichkeit von $\mathcal{ENC}$ ist. Dabei ist t_2 polynomiell beschränkt und

$$\epsilon_2(1^k) := Adv_{\mathcal{B}_2,\mathcal{ENC}}^{\text{Verbindlichkeit}}(1^k) = Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{NOBIND}\right]$$

Das Eintreten von FORGE bedeutet, dass $\mathcal{MAC}$ gebrochen wird. In einem Angriff darauf ist allerdings der geheime Schlüssel κ zufällig gewählt, während er in Experiment 1 gleich ρ^* und damit abhängig von π, com^* und dec^* ist. Für den Angriff muss das Experiment so abgewandelt werden, dass im Challenge-Schritt mit einem zufällig gewählten Schlüssel unterschrieben wird. Daher kommt zunächst ein Zwischenschritt.

Experiment 2: läuft genau so ab wie Experiment 1, außer dass beim Challenge-Schritt dec^* durch Nullen ersetzt wird. Dies ist notwendig, da in einem folgenden Angriff gegen die Vertraulichkeit von $\mathcal{ENC}$ dec^* nicht bekannt ist und somit ein Algorithmus $\mathcal{B}_4$, der das Experiment für $\mathcal{A}$ simulieren muss, im Challenge-Schritt Probleme bekäme. Experiment 2 hingegen kann aufgrund dieser Änderung problemlos simuliert werden:

$$Keygen_2(1^k) := Keygen_1(1^k)$$

$$DecryptOracle_2(com_i, \overline{c}_i, \sigma_i) := DecryptOracle_1(com_i, \overline{c}_i, \sigma_i)$$

$$Challenge_2(m_0, m_1) := \left\{ \begin{array}{l} b :=_R \{0,1\}; \\ \overline{c}^* := \mathsf{Encrypt}(\boxed{m_b.0^{|dec^*|}}, com^*, P); \\ \sigma := \mathsf{Sign}(\overline{c}^*, \rho^*); \\ c^* := (com^*, \overline{c}^*, \sigma^*) \end{array} \right\}$$

Die Wahrscheinlichkeit, dass FORGE eintritt, unterscheidet sich in Experiment 1 und 2 nur minimal, wie folgendes Lemma zeigt:

Lemma 3.8

Sei $\mathcal{A}$ ein (t, q, ϵ)-IND-CCA2-Angreifer von $\mathcal{PKE}$, dann gibt es einen Algorithmus $\mathcal{B}_3$, der ein (t_3, q, ϵ_3)-IND-sID-CPA-Angreifer von $\mathcal{IBE}$ ist. Dabei ist t_3 polynomiell beschränkt und

$$\epsilon_3(1^k) := Adv^{IND-sID-CPA}_{\mathcal{B}_3,\mathcal{IBE}}(1^k) = \frac{1}{2}\left|Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right] - Pr\left[E_2^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right]\right|$$

Nun kann die für den Angriff auf $\mathcal{MAC}$ notwendige Modifikation am Experiment eingeführt werden.

Experiment 3: läuft genau so ab wie Experiment 2, außer dass im Challenge-Schritt beim Unterschreiben der Nachricht der Schlüssel ρ^* durch ein im Keygen-Schritt zufällig und gleichverteilt gewähltes $\rho \in \{0,1\}^k$ ersetzt wird.

$$Keygen_3(1^k) := \left\{\begin{array}{l}(P, M) := \mathsf{Setup}(1^k);\\ \pi := \mathsf{Start}(1^k);\\ (\rho^*, com^*, dec^*) := \mathsf{Commit}(1^k, \pi);\\ \boxed{\rho :=_R \{0,1\}^k}\\ (E, D) := ((P, \pi), M)\end{array}\right\}$$

$$DecryptOracle_3(com_i, \overline{c}_i, \sigma_i) := DecryptOracle_2(com_i, \overline{c}_i, \sigma_i)$$

$$Challenge_3(m_0, m_1) := \left\{\begin{array}{l}b :=_R \{0,1\};\\ \overline{c}^* := \mathsf{Encrypt}(m_b.0^{|dec^*|}, com^*, P);\\ \sigma := \mathsf{Sign}(\overline{c}^*, \boxed{\rho});\\ c^* := (com^*, \overline{c}^*, \sigma^*)\end{array}\right\}$$

Da sich der $\mathcal{MAC}$-Schlüssel geändert hat, muss nun dass Ereignis FORGE entsprechend angepasst werden:

FORGE': $\exists i \in \{1, \ldots, q(1^k)\} : com_i = com^*, \mathsf{Verify}(\overline{c}, \sigma, \boxed{\rho}) = 1$

Nun kann man wieder zeigen, dass die Wahrscheinlichkeit mit der FORGE bzw. FORGE' eintritt, sich in Experiment 2 und 3 nur minimal unterscheidet:

Lemma 3.9

Sei $\mathcal{A}$ ein (t, q, ϵ)-IND-CCA2-Angreifer von $\mathcal{PKE}$, dann gibt es einen Algorithmus $\mathcal{B}_4$, der ein (t_4, ϵ_4)-Angreifer der Vertraulichkeit von $\mathcal{ENC}$ ist. Dabei ist t_4 polynomiell beschränkt und

$$\epsilon_4(1^k) := Adv^{\text{Vertraulichkeit}}_{\mathcal{B}_4,\mathcal{ENC}}(1^k) = \left|Pr\left[E_2^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right] - Pr\left[E_3^{\mathcal{A}}(1^k) : \mathsf{FORGE'}\right]\right|$$

Letztendlich lässt sich dann der Angriff auf $\mathcal{MAC}$ durchführen und somit die letzte Restwahrscheinlichkeit abschätzen:

Lemma 3.10

Sei $\mathcal{A}$ ein (t, q, ϵ)-IND-CCA2-Angreifer von $\mathcal{PKE}$, dann gibt es einen Algorithmus $\mathcal{B}_5$, der ein $(t_5, 1, \epsilon_5)$-Angreifer der Fälschungssicherheit von $\mathcal{MAC}$ ist. Dabei ist t_5 polynomiell beschränkt und

$$\epsilon_5(1^k) := Adv^{\text{Fälschung}}_{\mathcal{B}_5,\mathcal{MAC}}(1^k) \geq \frac{Pr\left[E_3^{\mathcal{A}}(1^k) : \mathsf{FORGE}'\right]}{q(1^k)}$$

All diese Lemmata zusammengesetzt ergeben den eigentlichen Sicherheitsbeweis.

Beweis von Satz 3.2

$\mathcal{A}$ ist ein (t, q, ϵ)-IND-CCA2-Angreifer von $\mathcal{PKE}$. Nach Lemmata 3.5, 3.7, 3.8, 3.9, 3.10 gibt es die im Satz beschriebenen Algorithmen $\mathcal{B}_1, \ldots, \mathcal{B}_5$ und ϵ ist wie folgt beschränkt:

$$\begin{aligned}
& \epsilon(1^k) = Adv^{IND-CCA2}_{\mathcal{A},\mathcal{PKE}}(1^k) \\
\overset{\text{nach Konstr.}}{=} \; & \left|Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{WIN}\right] - \tfrac{1}{2}\right| \\
\overset{\text{Lemma 3.4}}{=} \; & \left|Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{WIN}\right] - \tfrac{1}{2}\right| + 2 \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}\right] \\
\overset{\text{Lemma 3.5}}{=} \; & \epsilon_1(1^k) + 2 \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}\right] \\
\overset{\text{Beob. 3.6}}{\leq} \; & \epsilon_1(1^k) + 2 \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{NOBIND}\right] + 2 \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right] \\
\overset{\text{Lemma 3.7}}{=} \; & \epsilon_1(1^k) + 2 \cdot \epsilon_2(1^k) + 2 \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right] \\
= \; & \epsilon_1(1^k) + 2 \cdot \epsilon_2(1^k) + 2 \cdot \Big|Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right] \\
& - Pr\left[E_2^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right] + Pr\left[E_2^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right] \\
& - Pr\left[E_3^{\mathcal{A}}(1^k) : \mathsf{FORGE}'\right] + Pr\left[E_3^{\mathcal{A}}(1^k) : \mathsf{FORGE}'\right]\Big| \\
\leq \; & \epsilon_1(1^k) + 2 \cdot \epsilon_2(1^k) + 2 \cdot \left|Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right] - Pr\left[E_2^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right]\right| \\
& + 2 \cdot \left|Pr\left[E_2^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right] - Pr\left[E_3^{\mathcal{A}}(1^k) : \mathsf{FORGE}'\right]\right| \\
& + 2 \cdot \left|Pr\left[E_3^{\mathcal{A}}(1^k) : \mathsf{FORGE}'\right]\right| \\
\overset{\text{Lemmata 3.8,3.9,3.10}}{\leq} \; & \epsilon_1(1^k) + 2 \cdot \epsilon_2(1^k) + 4 \cdot \epsilon_3(1^k) + 4 \cdot \epsilon_4(1^k) + 2q(1^k) \cdot \epsilon_5(1^k)
\end{aligned}$$

Nachdem der Beweisverlauf erläutert ist, können die Lemmata bewiesen werden:

Beweis von Lemma 3.4

Der Beweis erfolgt über einige Zwischenschritte. Zunächst wird das Ereignis VALID („In irgendeiner Anfrage von $\mathcal{A}$ an $DecryptOracle_0$ wird ein gültiger Chiffretext mit $com_i = com^*$ übermittelt.“) weiter zerlegt:

$\mathsf{VALID}_{\leq j}$**:** $\exists i \in \{1, \ldots, j\} : com_i = com^* \ \wedge \ DecryptOracle_0(com_i, \bar{c}_i, \sigma_i) \neq \bot$
„Bis zur j-ten Anfrage von $\mathcal{A}$ an $DecryptOracle_0$ wird ein gültiger Chiffretext mit $com_i = com^*$ übermittelt.“

$\mathsf{VALID}_j = \mathsf{VALID}_{\leq j} \cap \overline{\mathsf{VALID}_{\leq j-1}}$
„Zum ersten Mal bei der j-ten Anfrage von $\mathcal{A}$ an $DecryptOracle_0$ wird ein gültiger Chiffretext mit $com_i = com^*$ übermittelt.“

1. Teilbehauptung: Falls $\overline{\mathsf{VALID}_{\leq j}}$ gilt, ist das Verhalten von $\mathcal{A}$ (also die Wahrscheinlichkeitsverteilungen aller Ausgabewerte) bis zum $j+1$-ten Schritt in Experiment 0 und 1 identisch.

Während des Beweises sei j fest gewählt. Per Induktion wird die Gültigkeit der Aussage für die Ausgabewerte des i-ten Schrittes ($1 \leq i \leq j+1$) gezeigt.

Induktionsanfang ($i = 1$): Für alle E gilt $\mathcal{A}_{\leq 1}^{DecryptOracle_0}(E) = \mathcal{A}_1(E) = \mathcal{A}_{\leq 1}^{DecryptOracle_1}(E)$. Wenn nun auch die Wahrscheinlichkeitsverteilung der Eingabe E in beiden Experimenten gleich ist, dann gilt dies auch für die der Ausgabe von $\mathcal{A}_1(E)$. Da beide Experimente bis zur ersten Anfrage identisch verlaufen, ist dies der Fall.

Induktionsschritt ($i - 1 \rightsquigarrow i \leq j + 1$): Es ist die Wahrscheinlichkeitsverteilung der Ausgabe von $\mathcal{A}_{\leq i}^{DecryptOracle_0}(E)$ bzw. $\mathcal{A}_{\leq i}^{DecryptOracle_1}(E)$ zu untersuchen. Zur Erinnerung:

$$\mathcal{A}_{\leq i}^{DecryptOracle_0}(E) := \left\{ \begin{array}{l} (c_{i-1}, s_{i-1}) := \mathcal{A}_{\leq i-1}^{DecryptOracle_0}(E); \\ m_{i-1} := \mathsf{DecryptOracle_0}(c_{i-1}); \\ (c_i, s_i) := \mathcal{A}_i(m_{i-1}, s_{i-1}) \end{array} \right\}$$

$$\mathcal{A}_{\leq i}^{DecryptOracle_1}(E) := \left\{ \begin{array}{l} (c_{i-1}, s_{i-1}) := \mathcal{A}_{\leq i-1}^{DecryptOracle_1}(E); \\ m_{i-1} := \mathsf{DecryptOracle_1}(c_{i-1}); \\ (c_i, s_i) := \mathcal{A}_i(m_{i-1}, s_{i-1}) \end{array} \right\}$$

Nach Induktionsvoraussetzung haben die Ausgabewerte (c_{i-1}, s_{i-1}) des $i-1$-ten Schrittes in beiden Experimenten die gleiche Wahrscheinlichkeitsverteilung. c_{i-1} hat die Form $(com_{i-1}, \bar{c}_{i-1}, \sigma_{i-1})$. Man beachte, dass $DecryptOracle_0$ und $DecryptOracle_1$ sich nur bei der Entschlüsselung eines gültigen Chiffretextes mit $com_{i-1} = com^*$ unterscheiden. Nach Voraussetzung gilt aber $\overline{\mathsf{VALID}_{\leq j}}$ (mit $i \leq j + 1$, also $i - 1 \leq j$) und damit ist entweder $com_{i-1} \neq com^*$ oder c_{i-1} ein ungültiger Chiffretext. Für Eingaben dieser Form sind die Ausgabwerte von $DecryptOracle_0$ und $DecryptOracle_1$ aber identisch. Insgesamt ist also die Wahrscheinlichkeitsverteilung der Eingabe (m_{i-1}, s_{i-1}) von $\mathcal{A}$ in

beiden Experimenten identisch und damit auch die der Ausgabe (m_i, s_i).

Hinweis: für die in Abschnitt 2 identifizierten Spezialfälle um den Challenge- bzw. Guess-Schritt herum kann man den Beweis analog führen, da der Challenge-Schritt in beiden Experimenten identisch gehandhabt wird.

2. Teilbehauptung: $Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j}\right] = Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j}\right]$

Der Beweis erfolgt per Induktion über j. Der Induktionsanfang verläuft mit den selben Argumenten wie bei Teilbehauptung 1. Der Induktionsschritt $(i-1 \rightsquigarrow i)$ ist:

$$
\begin{aligned}
& Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j}\right] \\
= \quad & Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j} \mid \mathsf{VALID}_{\leq j-1}\right] \cdot Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j-1}\right] \\
& + Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j} \mid \overline{\mathsf{VALID}_{\leq j-1}}\right] \cdot Pr\left[E_0^{\mathcal{A}}(1^k) : \overline{\mathsf{VALID}_{\leq j-1}}\right] \\
\overset{\text{Indukt.}}{=} \quad & Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j} \mid \mathsf{VALID}_{\leq j-1}\right] \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j-1}\right] \\
& + Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j} \mid \overline{\mathsf{VALID}_{\leq j-1}}\right] \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \overline{\mathsf{VALID}_{\leq j-1}}\right] \\
\overset{(\star)}{=} \quad & Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j} \mid \mathsf{VALID}_{\leq j-1}\right] \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j-1}\right] \\
& + Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j} \mid \overline{\mathsf{VALID}_{\leq j-1}}\right] \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \overline{\mathsf{VALID}_{\leq j-1}}\right] \\
\overset{\text{Teilbeh. 1}}{=} \quad & Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j} \mid \mathsf{VALID}_{\leq j-1}\right] \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j-1}\right] \\
& + Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j} \mid \overline{\mathsf{VALID}_{\leq j-1}}\right] \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \overline{\mathsf{VALID}_{\leq j-1}}\right] \\
= \quad & Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j}\right]
\end{aligned}
$$

In $(\star)$ geht ein: $Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j} | \mathsf{VALID}_{\leq j-1}\right] = 1 = Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j} | \mathsf{VALID}_{\leq j-1}\right]$.

3. Teilbehauptung: $Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{VALID}\right] = Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}\right]$

Betrachte zunächst VALID_j:

$$
\begin{aligned}
& Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{VALID}_j\right] \\
\overset{\text{Def.}}{=} \quad & Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j} \cap \overline{\mathsf{VALID}_{\leq j-1}}\right] \\
= \quad & Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j} \mid \overline{\mathsf{VALID}_{\leq j-1}}\right] \cdot Pr\left[E_0^{\mathcal{A}}(1^k) : \overline{\mathsf{VALID}_{\leq j-1}}\right] \\
\overset{\text{Teilbeh. 1 und 2}}{=} \quad & Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}_{\leq j} \mid \overline{\mathsf{VALID}_{\leq j-1}}\right] \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \overline{\mathsf{VALID}_{\leq j-1}}\right] \\
= \quad & Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}_j\right]
\end{aligned}
$$

Dann gilt: $\mathsf{VALID} = \bigcup_{j=1}^{q(1^k)} \mathsf{VALID}_j$ und alle VALID_j sind paarweise disjunkt:

$$
\begin{aligned}
Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{VALID}\right] &= \textstyle\sum_{j=1}^{q(1^k)} Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{VALID}_j\right] \\
&= \textstyle\sum_{j=1}^{q(1^k)} Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}_j\right] = Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}\right]
\end{aligned}
$$

eigentliche Behauptung:

$$
\begin{aligned}
& \left|Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{WIN}\right] - \tfrac{1}{2}\right| \\
= \; & \left|Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{WIN} \cap \overline{\mathsf{VALID}}\right] + Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{WIN} \cap \mathsf{VALID}\right] - \tfrac{1}{2}\right| \\
\leq \; & \left|Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{WIN} \cap \overline{\mathsf{VALID}}\right] - \tfrac{1}{2}\right| + Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{WIN} \cap \mathsf{VALID}\right] \\
\leq \; & \left|Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{WIN} \cap \overline{\mathsf{VALID}}\right] - \tfrac{1}{2}\right| + Pr\left[E_0^{\mathcal{A}}(1^k) : \mathsf{VALID}\right]
\end{aligned}
$$

Mit Teilbehauptung 1, 2 und 3 folgt:

$$
\begin{aligned}
= \; & \left|Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{WIN} \cap \overline{\mathsf{VALID}}\right] - \tfrac{1}{2}\right| + Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}\right] \\
= \; & \left|Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{WIN} \cap \overline{\mathsf{VALID}}\right] - \tfrac{1}{2}\right| - Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}\right] + 2 \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}\right] \\
\leq \; & \left|Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{WIN} \cap \overline{\mathsf{VALID}}\right] - \tfrac{1}{2}\right| - \left|Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{WIN} \cap \mathsf{VALID}\right]\right| + 2 \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}\right] \\
\leq \; & \left|Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{WIN} \cap \overline{\mathsf{VALID}}\right] + Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{WIN} \cap \mathsf{VALID}\right] - \tfrac{1}{2}\right| + 2 \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}\right] \\
= \; & \left|Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{WIN}\right] - \tfrac{1}{2}\right| + 2 \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{VALID}\right]
\end{aligned}
$$

Beweis von Lemma 3.5

$\mathcal{B}_1$ simuliert für $\mathcal{A}$ den Herausforderer in Experiment 1 und versucht selbst dadurch $\mathcal{IBE}$ in einem IND-sID-CPA-Experiment zu brechen, d.h. $\mathcal{B}_1$ muss eine anzugreifende Identität bestimmen, dann zwei Nachrichten auswählen und herausfinden, welche von beiden an diese Identität verschlüsselt wurde.
Für die Simulation von Experiment 1 reicht $\mathcal{B}_1$ im Wesentlichen alles, was mit $\mathcal{IBE}$ zu tun hat, an den eigenen Herausforderer weiter und ergänzt den Rest. Im Challenge-Schritt wird an die von $\mathcal{B}_1$ angriffene Identität verschlüsselt. $\mathcal{B}_1$ ist wie folgt über die drei Teilalgorithmen $\mathcal{B}_{1,0}, \mathcal{B}_{1,\mathrm{I}}$ und $\mathcal{B}_{1,\mathrm{II}}$ (siehe Definition 7) definiert:

$$
\mathcal{B}_{1,0}(1^k) \quad := \left\{ \begin{array}{l} \pi := \mathsf{Start}(1^k); (\rho^*, com^*, dec^*) := \mathsf{Commit}(1^k, \pi); \\ ID^* := com^*; s_0 := (\pi, \rho^*, com^*, dec^*) \\ \text{Ausgabe: } (ID^*, s_0) \end{array} \right\}
$$

Nachdem $\mathcal{B}_{1,0}$ die anzugreifende Identität festgelegt hat, werden die Systemparameter von $\mathcal{IBE}$ generiert und $\mathcal{B}_{1,\mathrm{I}}$ mit den öffentlichen Parametern gestartet:

$$
\begin{array}{l} \mathcal{B}_{1,\mathrm{I}}^{\mathsf{KeygenOracle}}(P, s_0) \\ \text{mit } s_0 = (\pi, \rho^*, com^*, dec^*) \end{array} \quad := \left\{ \begin{array}{l} (m_0', m_1', s') := \mathcal{A}_{\mathrm{I}}^{SimDecryptOracle_1}((P, \pi)); \\ m_0 := m_0'.dec^*; m_1 := m_1'.dec^*; \\ s := (\pi, \rho^*, com^*, dec^*, s') \\ \text{Ausgabe: } (m_0, m_1, s) \end{array} \right\}
$$

Irgendwann gibt $\mathcal{B}_{1,\mathrm{I}}$ zwei Nachrichten aus. Eine davon wird verschlüsselt und an $\mathcal{B}_{1,\mathrm{II}}$ übergeben:

$$\left.\begin{array}{l}\mathcal{B}_{1,\mathrm{II}}^{\mathsf{KeygenOracle}}(\overline{c}^*, s) \\ \text{mit } s = (\pi, \rho^*, com^*, dec^*, s')\end{array}\right. := \left\{\begin{array}{l}\sigma^* := \mathsf{Sign}(\overline{c}^*, \rho^*); c^* := (com^*, \overline{c}^*, \sigma^*); \\ b' := \mathcal{A}_{\mathrm{II}}^{SimDecryptOracle_1}(c^*, s') \\ \text{Ausgabe: } b'\end{array}\right\}$$

Das Orakel in $\mathcal{B}_{1,\mathrm{I}}$ und $\mathcal{B}_{1,\mathrm{II}}$ funktioniert wie folgt:

$$\left.\begin{array}{l}SimDecryptOracle_1 \\ (com_i, \overline{c}_i, \sigma_i)\end{array}\right. := \left\{\begin{array}{l}\text{Falls } com_i \neq com^*: \\ \quad D_{com_i} := \mathsf{KeygenOracle}(com_i); \\ \quad m_i.dec_i := \mathsf{Decrypt}(\overline{c}_i, D_{com_i}, P); \\ \quad \rho_i := \mathsf{Decommit}(\pi, com_i, dec_i) \\ \quad v_i := \mathsf{Verify}(\overline{c}, \sigma, \rho') \\ \quad \text{Ausgabe: } m_i, \text{ falls } m_i.dec_i \neq \bot,\ \rho_i \neq \bot,\ v_i = 1 \\ \quad \text{sonst } \bot. \\ \text{Falls } com_i = com^*\text{: Ausgabe: } \bot\end{array}\right\}$$

Die Laufzeit von $\mathcal{B}_1$ ist im Wesentlichen beschränkt durch $t_1 := T(\mathsf{Start}) + T(\mathsf{Commit}) + T(\mathsf{Sign}) + t + q \cdot (T(\mathsf{KeygenOracle}) + T(\mathsf{Decrypt}) + T(\mathsf{Decommit}) + T(\mathsf{Verify}))$ und damit polynomiell.
$\mathcal{B}_1$ stellt nur in $SimDecryptOracle_1$ Anfragen an das $\mathsf{KeygenOracle}$. Dies sind maximal so viele Anfragen wie $\mathcal{A}$ tätigt, also beschränkt durch q. Nach Konstruktion wird dabei niemals eine Anfrage mit der angegriffenen Identität com^* gestellt. Weiterhin beachte man, dass sich $SimDecryptOracle_1$ und $DecryptOracle_1$ nur durch die Art der Erzeugung von D_{com_i} unterscheiden. Beide liefern aber die selbe Wahrscheinlichkeitsverteilung für D_{com_i} und sind damit in der folgenden Analyse austauschbar.

Es bleibt der Vorteil von $\mathcal{B}_1$ zu bestimmen:

$$\begin{aligned} & Adv_{\mathcal{B}_1,\mathcal{IBE}}^{IND-sID-CPA}(1^k) && =: \epsilon_1(1^k) \\ = & \left| Pr\left[\begin{array}{ll}(ID^*, s_0) := \mathcal{B}_{1,0}(1^k); (P, M) := \mathsf{Setup}(1^k); & \\ (m_0, m_1, s) := \mathcal{B}_{1,\mathrm{I}}^{\mathsf{KeygenOracle}}(P, s_0); & \\ b :=_R \{0,1\}; \overline{c}^* := \mathsf{Encrypt}(m_b, ID^*, P); & : b = b' \\ b' := \mathcal{B}_{1,\mathrm{II}}^{\mathsf{KeygenOracle}}(\overline{c}^*, s) & \end{array}\right] - \frac{1}{2} \right| \end{aligned}$$

Definition von $\mathcal{B}_1$ einsetzen:

$$= \left| Pr\left[\begin{array}{ll}\pi := \mathsf{Start}(1^k); (\rho^*, com^*, dec^*) := \mathsf{Commit}(1^k, \pi); (P, M) := \mathsf{Setup}(1^k); & \\ (m_0', m_1', s') := \mathcal{A}_{\mathrm{I}}^{SimDecryptOracle_1}((P, \pi)); & \\ b :=_R \{0,1\}; \overline{c}^* := \mathsf{Encrypt}(m_b'.dec^*, com^*, P); & : b = b' \\ \sigma^* := \mathsf{Sign}(\overline{c}^*, \rho^*); c^* := (com^*, \overline{c}^*, \sigma^*); b' := \mathcal{A}_{\mathrm{II}}^{SimDecryptOracle_1}(c^*, s') & \end{array}\right] - \frac{1}{2} \right|$$

Umsortieren und $SimDecryptOracle_1$ durch $DecryptOracle_1$ austauschen.

$$= \left| Pr \left[\begin{array}{l} (P, M) := \mathsf{Setup}(1^k); \pi := \mathsf{Start}(1^k); \\ (\rho^*, com^*, dec^*) := \mathsf{Commit}(1^k, \pi); (E, D) := ((P, \pi), M) \\ (m_0', m_1', s') := \mathcal{A}_{\mathrm{I}}^{DecryptOracle_1}(E); \\ b :=_R \{0,1\}; \overline{c}^* := \mathsf{Encrypt}(m_b'.dec^*, com^*, P); \\ \sigma^* := \mathsf{Sign}(\overline{c}^*, \rho^*); c^* := (com^*, \overline{c}^*, \sigma^*); \\ b' := \mathcal{A}_{\mathrm{II}}^{DecryptOracle_1}(c^*, s') \end{array} : b = b' \right] - \frac{1}{2} \right|$$

$$= \left| Pr \left[E_1^{\mathcal{A}}(1^k) : \mathsf{WIN} \right] - \frac{1}{2} \right|$$

Insgesamt ist B_1 also ein (t_1, q, ϵ_1)-IND-sID-CPA-Angreifer von $\mathcal{IBE}$, wobei t_1 polynomiell beschränkt und $\epsilon_1(1^k) = \left| Pr \left[E_1^{\mathcal{A}}(1^k) : \mathsf{WIN} \right] - \frac{1}{2} \right|$ ist.

Beweis von Lemma 3.7

$\mathcal{B}_2$ simuliert für $\mathcal{A}$ den Herausforderer in Experiment 1 und versucht selbst dadurch die Verbindlichkeit von $\mathcal{ENC}$ zu brechen, d.h. zu gegebenen $\pi, \rho^*, com^*, dec^*$ ein dec' mit $\mathsf{Decommit}(\pi, com^*, dec') \notin \{\rho^*, \perp\}$ zu finden.
Für die Simulation von Experiment 1 lässt $\mathcal{B}_1$ sich $\pi, \rho^*, com^*, dec^*$ vorgeben und bestimmt alle restlichen Werte selbst. Taucht ein passendes dec_i unter den Anfragen von $\mathcal{A}$ an das $DecryptOracle_1$ auf, so gibt $\mathcal{B}_2$ dieses zurück. $\mathcal{B}_2$ ist wie folgt definiert (siehe Definition 12):

$$\mathcal{B}_2(\pi, \rho^*, com^*, dec^*) := \left\{ \begin{array}{l} (P, M) := \mathsf{Setup}(1^k); (E, D) := ((P, \pi), M) \\ (m_0, m_1, s) := \mathcal{A}_{\mathrm{I}}^{SimDecryptOracle_2}(E); \\ b :=_R \{0,1\}; \overline{c}^* := \mathsf{Encrypt}(m_b.dec^*, com^*, P); \\ \sigma^* := \mathsf{Sign}(\overline{c}^*, \rho^*); c^* := (com^*, \overline{c}^*, \sigma^*); \\ b' := \mathcal{A}_{\mathrm{II}}^{SimDecryptOracle_2}(c^*, s) \\ \text{Prüfe, ob } \mathsf{NOBIND} \text{ eingetreten ist. Falls ja:} \\ \exists i \in \{1, \ldots, q(1^k)\} \text{ mit } com_i = com^*, \\ m_i.dec_i := \mathsf{Decrypt}(\overline{c}_i, D_{com_i}, P) \neq \perp, \\ \rho_i := \mathsf{Decommit}(\pi, com_i, dec_i) \neq \perp \text{ und } \rho_i \neq \rho^* \\ \text{Ausgabe: } dec_i, \text{ falls } \mathsf{NOBIND} \text{ eingetreten, sonst } dec^*. \end{array} \right\}$$

Da $\mathcal{B}_2$ den kompletten privaten Schlüssel kennt, kann er $DecryptOracle_1$ wie vorgesehen simulieren.

$$SimDecryptOracle_2(com_i, \overline{c}_i, \sigma_i) := DecryptOracle_1(com_i, \overline{c}_i, \sigma_i)$$

Ebenso kann er die Überprüfung, ob NOBIND eingetreten ist, vornehmen. Dies erfordert die Entschlüsselung der von $\mathcal{A}$ angefragten Nachrichten mit $com_i = com^*$.

Die Laufzeit von $\mathcal{B}_2$ ist im Wesentlichen beschränkt durch $t_2 := T(\mathsf{Setup})+T(\mathsf{Encrypt})+T(\mathsf{Sign})+t+q\cdot(T(\mathsf{Keygen})+T(\mathsf{Decrypt})+T(\mathsf{Decommit})+T(\mathsf{Verify}))$ und damit polynomiell. Es bleibt der Vorteil von $\mathcal{B}_2$ zu bestimmen:

$$
\begin{aligned}
&Adv^{\text{Verbindlichkeit}}_{\mathcal{B}_2,\mathcal{ENC}}(1^k) && =: \epsilon_2(1^k)\\
=&Pr\left[\begin{array}{l}\pi := \mathsf{Start}(1^k);\\ (\rho_0^*, com^*, dec^*) := \mathsf{Commit}(1^k,\pi);\\ dec' := \mathcal{B}_2(\pi,\rho^*,com^*,dec^*);\\ \rho' := \mathsf{Decommit}(\pi,com^*,dec')\end{array} \;:\; \rho' \notin \{\rho^*,\perp\}\right]
\end{aligned}
$$

Ganz analog zum Beweis von Lemma 3.5 überführt man dies durch Einsetzen der Definition von $\mathcal{B}_2$ und ein wenig umsortieren in:

$$
=Pr\left[\begin{array}{l}E_1^{\mathcal{A}}(1^k);\\ \text{Prüfe, ob } \mathsf{NOBIND} \text{ eingetreten ist: } \ldots\\ \text{Ausgabe: } dec' := dec_i \text{ oder } dec^*\\ \rho' := \mathsf{Decommit}(\pi,com^*,dec')\end{array} \;:\; \rho' \notin \{\rho^*,\perp\}\right]
$$

Falls NOBIND eingetreten ist, gilt nach Definition des Ereignisses, dass die Entschlüsselung eines Chiffretext angefragt wird, die ein dec_i enthält mit $\mathsf{Decommit}(\pi,com^*,dec_i) \notin \{\rho^*,\perp\}$. Falls NOBIND nicht eingetreten ist, wird dec^* ausgegeben, für welches gilt $\mathsf{Decommit}(\pi,com^*,dec^*) = \rho^* \in \{\rho^*,\perp\}$. Zusammengefasst: falls NOBIND eintritt, liegt $\mathcal{B}_2$ immer richtig mit seiner Antwort und sonst immer falsch.

$$
\begin{aligned}
= \quad & \underbrace{Pr\left[\begin{array}{l}E_1^{\mathcal{A}}(1^k); dec' := dec_i\\ \rho' := \mathsf{Decommit}(\pi,com^*,dec')\end{array} \;:\; \rho' \notin \{\rho^*,\perp\}\right]}_{=1} \cdot Pr[E_1^{\mathcal{A}}(1^k) : \mathsf{NOBIND}]\\
&+ \underbrace{Pr\left[\begin{array}{l}E_1^{\mathcal{A}}(1^k); dec' := dec^*\\ \rho' := \mathsf{Decommit}(\pi,com^*,dec')\end{array} \;:\; \rho' \notin \{\rho^*,\perp\}\right]}_{=0} \cdot Pr[E_1^{\mathcal{A}}(1^k) : \overline{\mathsf{NOBIND}}]\\
=&Pr[E_1^{\mathcal{A}}(1^k) : \mathsf{NOBIND}]
\end{aligned}
$$

Insgesamt ist B_2 also ein (t_2,ϵ_2)-Angreifer der Verbindlichkeit von $\mathcal{ENC}$, wobei t_2 polynomiell beschränkt und $\epsilon_2(1^k) = Pr[E_1^{\mathcal{A}}(1^k) : \mathsf{NOBIND}]$ ist.

Beweis von Lemma 3.8

$\mathcal{B}_3$ simuliert für $\mathcal{A}$ den Herausforderer in Experiment 1 oder 2 und versucht selbst dadurch $\mathcal{IBE}$ in einem IND-sID-CPA-Experiment zu brechen, d.h. $\mathcal{B}_3$ muss eine anzugreifende Identität bestimmen, dann zwei Nachrichten auswählen und herausfinden, welche von beiden an diese Identität verschlüsselt wurde.

In weiten Teilen ist $\mathcal{B}_3$ definiert wie $\mathcal{B}_1$ – die wesentliche neue Idee steckt aber in der Challenge-Phase. $\mathcal{B}_3$ wählt die Nachrichten m_0 und m_1 so, dass, falls die erste verschlüsselt wird, die Sicht von $\mathcal{A}$ genau wie in Experiment 1 und sonst genau wie in Experiment 2 ist. Anhand des Verhaltens von $\mathcal{A}$ kann $\mathcal{B}_3$ dann herausfinden, welche der beiden Nachrichten verschlüsselt wurde. $\mathcal{B}_3$ ist wie folgt über die drei Teilalgorithmen $\mathcal{B}_{3,0}, \mathcal{B}_{3,\mathrm{I}}$ und $\mathcal{B}_{3,\mathrm{II}}$ (siehe Definition 7) definiert:

$$\mathcal{B}_{3,0}(1^k) \quad := \left\{ \begin{array}{l} \pi := \mathsf{Start}(1^k); (\rho^*, com^*, dec^*) := \mathsf{Commit}(1^k, \pi); \\ ID^* := com^*; s_0 := (\pi, \rho^*, com^*, dec^*) \\ \text{Ausgabe: } (ID^*, s_0) \end{array} \right\}$$

Die anzugreifende Identität wird genau wie in $\mathcal{B}_{1,0}$ festgelegt. Dann werden vom Herausforderer im IND-sID-CPA-Experiment die Systemparameter von $\mathcal{IBE}$ generiert und $\mathcal{B}_{3,\mathrm{I}}$ mit den öffentlichen Parametern gestartet:

$$\begin{array}{l} \mathcal{B}_{3,\mathrm{I}}^{\mathsf{KeygenOracle}}(P, s_0) \\ \text{mit } s_0 = (\pi, \rho^*, com^*, dec^*) \end{array} \quad := \left\{ \begin{array}{l} (m_0', m_1', s') := \mathcal{A}_{\mathrm{I}}^{SimDecryptOracle_3}((P, \pi)); \\ dec_0 := dec^*; dec_1 := 0^{|dec^*|}; b :=_R \{0,1\}; \\ m_0 := m_b'.dec_0; m_1 := m_b'.dec_1; \\ s := (\pi, \rho^*, com^*, dec^*, s') \\ \text{Ausgabe: } (m_0, m_1, s) \end{array} \right\}$$

Irgendwann gibt $\mathcal{B}_{3,\mathrm{I}}$ zwei Nachrichten aus. Eine davon wird vom Herausforderer per Zufallsbit $\overline{b}$ (der Name b ist schon vergeben) ausgewählt, verschlüsselt und an $\mathcal{B}_{3,\mathrm{II}}$ übergeben:

$$\begin{array}{l} \mathcal{B}_{3,\mathrm{II}}^{\mathsf{KeygenOracle}}(\overline{c}^*, s) \\ \text{mit } s = (\pi, \rho^*, com^*, dec^*, s') \end{array} \quad := \left\{ \begin{array}{l} \sigma^* := \mathsf{Sign}(\overline{c}^*, \rho^*); c^* := (com^*, \overline{c}^*, \sigma^*); \\ b' := \mathcal{A}_{\mathrm{II}}^{SimDecryptOracle_3}(c^*, s') \\ \text{Prüfe, ob } \mathsf{FORGE} \text{ eingetreten ist.} \\ \overline{b}' := 0, \text{ falls } \mathsf{FORGE} \text{ eingetreten, sonst } \overline{b}' := 1. \\ \text{Ausgabe: } \overline{b}' \end{array} \right\}$$

Da $\mathcal{B}_3$ den Wert ρ^* kennt, kann er leicht überprüfen, ob FORGE eingetreten ist. Das Orakel in $\mathcal{B}_3$ funktioniert genau so wie das in $\mathcal{B}_1$:

$$SimDecryptOracle_3(com_i, \overline{c}_i, \sigma_i) := SimDecryptOracle_1(com_i, \overline{c}_i, \sigma_i)$$

Die Laufzeit von $\mathcal{B}_3$ ist im Wesentlichen beschränkt durch $t_3 := T(\mathsf{Start}) + T(\mathsf{Commit}) + T(\mathsf{Sign}) + t + q \cdot (T(\mathsf{KeygenOracle}) + T(\mathsf{Decrypt}) + T(\mathsf{Decommit}) + T(\mathsf{Verify})) + q \cdot T(\mathsf{Verify})$ und damit polynomiell.
Wie bei $\mathcal{B}_1$ stellt $\mathcal{B}_3$ maximal q-viele gültige Anfragen an das $\mathsf{KeygenOracle}$. Außerdem sind $SimDecryptOracle_3$ und $DecryptOracle_1$ $(= DecryptOracle_2)$ wieder in der folgenden Analyse austauschbar.

Es bleibt der Vorteil von $\mathcal{B}_3$ zu bestimmen:

$$
\begin{aligned}
& Adv^{IND-sID-CPA}_{\mathcal{B}_3,\mathcal{IBE}}(1^k) \qquad =: \epsilon_3(1^k) \\
= & \left| Pr \left[\begin{array}{l} (ID^*, s_0) := \mathcal{B}_{3,0}(1^k); (P, M) := \mathsf{Setup}(1^k); \\ (m_0, m_1, s) := \mathcal{B}_{3,\mathrm{I}}^{\mathsf{KeygenOracle}}(P, s_0); \\ \overline{b} :=_R \{0,1\}; \overline{c}^* := \mathsf{Encrypt}(m_{\overline{b}}, ID^*, P); \\ \overline{b}' := \mathcal{B}_{3,\mathrm{II}}^{\mathsf{KeygenOracle}}(\overline{c}^*, s) \end{array} : \overline{b} = \overline{b}' \right] - \frac{1}{2} \right|
\end{aligned}
$$

Definition von $\mathcal{B}_1$ einsetzen, $SimDecryptOracle_3$ durch $SimDecryptOracle_1$ ersetzen und umsortieren:

$$
\begin{aligned}
= & \left| Pr \left[\begin{array}{l} \boxed{\overline{b} :=_R \{0,1\};} \\ \\ (P, M) := \mathsf{Setup}(1^k); \pi := \mathsf{Start}(1^k); \\ (\rho^*, com^*, dec^*) := \mathsf{Commit}(1^k, \pi); (E, D) := ((P, \pi), M); \\ \\ (m_0', m_1', s') := \mathcal{A}_{\mathrm{I}}^{SimDecryptOracle_1}(E); \\ \boxed{dec_0 := dec^*; dec_1 := 0^{|dec^*|};} \\ b :=_R \{0,1\}; \overline{c}^* := \mathsf{Encrypt}(m_b'.dec_{\overline{b}}, com^*, P); \\ \sigma^* := \mathsf{Sign}(\overline{c}^*, \rho^*); c^* := (com^*, \overline{c}^*, \sigma^*); \\ \\ b' := \mathcal{A}_{\mathrm{II}}^{SimDecryptOracle_1}(c^*, s') \\ \\ \text{Prüfe, ob } \mathsf{FORGE} \text{ eingetreten ist.} \\ \overline{b}' := 0, \text{ falls } \mathsf{FORGE} \text{ eingetreten, sonst } \overline{b}' := 1. \end{array} : \overline{b} = \overline{b}' \right] - \frac{1}{2} \right|
\end{aligned}
$$

Nun fällt auf: falls $\overline{b} = 0$ ist, verläuft die Simulation exakt wie in Experiment 1, in dem $m_b'.dec^*$ verschlüsselt wird. Ansonsten verläuft sie exakt wie Experiment 2, in dem $m_b'.0^{dec^*}$ verschlüsselt wird:

$$
\begin{aligned}
= & \left| Pr \left[E_1^{\mathcal{A}}(1^k); \text{ Prüfe } \mathsf{FORGE}\ldots \quad : \overline{b}' = 0 \right] \cdot \underbrace{Pr \left[\overline{b} :=_R \{0,1\} : \overline{b} = 0 \right]}_{= \frac{1}{2}} \right. \\
& \left. + Pr \left[E_2^{\mathcal{A}}(1^k); \text{ Prüfe } \mathsf{FORGE}\ldots \quad : \overline{b}' = 1 \right] \cdot \underbrace{Pr \left[\overline{b} :=_R \{0,1\} : \overline{b} = 1 \right]}_{= \frac{1}{2}} - \frac{1}{2} \right|
\end{aligned}
$$

Nun kann man diese Wahrscheinlichkeiten weiter danach aufschlüsseln, ob FORGE eintritt oder nicht. Man beachte dabei, dass $\overline{b'} := 0$ genau dann gesetzt wird, wenn FORGE eintritt. Somit ist je nachdem, ob FORGE eingetreten ist oder nicht, die Wahrscheinlichkeit für $\overline{b'} = 0$ exakt 0 oder 1:

$$= \frac{1}{2} \Bigg| \underbrace{Pr\left[E_1^{\mathcal{A}}(1^k);\ \text{Prüfe FORGE}\ldots \quad : \overline{b}' = 0 \mid \mathsf{FORGE}\right]}_{=1} \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right]$$
$$+ \underbrace{Pr\left[E_1^{\mathcal{A}}(1^k);\ \text{Prüfe FORGE}\ldots \quad : \overline{b}' = 0 \mid \overline{\mathsf{FORGE}}\right]}_{=0} \cdot Pr\left[E_1^{\mathcal{A}}(1^k) : \overline{\mathsf{FORGE}}\right]$$
$$+ \underbrace{Pr\left[E_2^{\mathcal{A}}(1^k);\ \text{Prüfe FORGE}\ldots \quad : \overline{b}' = 1 \mid \mathsf{FORGE}\right]}_{=0} \cdot Pr\left[E_2^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right]$$
$$+ \underbrace{Pr\left[E_2^{\mathcal{A}}(1^k);\ \text{Prüfe FORGE}\ldots \quad : \overline{b}' = 1 \mid \overline{\mathsf{FORGE}}\right]}_{=1} \cdot Pr\left[E_2^{\mathcal{A}}(1^k) : \overline{\mathsf{FORGE}}\right] - 1 \Bigg|$$

Mit $Pr\left[E_2^{\mathcal{A}}(1^k) : \overline{\mathsf{FORGE}}\right] = 1 - Pr\left[E_2^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right]$ erhält man:

$$= \frac{1}{2} \left| Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right] - Pr\left[E_2^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right] \right|$$

Insgesamt ist B_3 also ein (t_3, q, ϵ_3)-IND-sID-CPA-Angreifer von $\mathcal{IBE}$, wobei t_3 polynomiell beschränkt und $\epsilon_3(1^k) = \frac{1}{2}\left|Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right] - Pr\left[E_2^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right]\right|$ ist.

Beweis von Lemma 3.9

Ganz ähnlich wie im vorherigen Beweis simuliert $\mathcal{B}_4$ für $\mathcal{A}$ den Herausforderer in Experiment 2 oder 3 und versucht selbst dadurch die Vertraulichkeit von $\mathcal{ENC}$ anzugreifen. $\mathcal{B}_4$ bekommt π, com^*, ρ^* und muss entscheiden, ob ρ^* gemeinsam mit com^* oder zufällig erzeugt wurde.
$\mathcal{B}_4$ ergänzt diese Werte mit selbst erzeugten Systemparametern von $\mathcal{IBE}$. Es wird so sein, dass, falls ρ^* zu com^* „passt“, die Sicht von $\mathcal{A}$ wie in Experiment 2 und sonst wie in Experiment 3 ist. Genau wie zuvor wird $\mathcal{B}_4$ anhand des Verhaltens von $\mathcal{A}$ erkennen, welche Situation vorliegt. $\mathcal{B}_4$ ist wie folgt definiert (siehe Definition 11):

$$\mathcal{B}_4(\pi, com^*, \rho_{\overline{b}}^*) \quad := \left\{ \begin{array}{l} (P, M) := \mathsf{Setup}(1^k); (E, D) := ((P, \pi), M) \\ (m_0, m_1, s) := \mathcal{A}_{\mathrm{I}}^{SimDecryptOracle_4}(E); \\ b :=_R \{0,1\}; \overline{c}^* := \mathsf{Encrypt}(m_b.0^{|dec^*|}, com^*, P); \\ \sigma^* := \mathsf{Sign}(\overline{c}^*, \rho_{\overline{b}}^*); c^* := (com^*, \overline{c}^*, \sigma^*); \\ b' := \mathcal{A}_{\mathrm{II}}^{SimDecryptOracle_3}(c^*, s) \\ \text{Überprüfe alle } (com_i, \overline{c}_i, \sigma_i) \text{ mit } com_i = com^*, \\ \text{ob } \mathsf{Verify}(\overline{c}_i, \sigma_i, \rho_{\overline{b}}^*) = 1 \text{ für mindestens ein } i \text{ gilt.} \\ \text{Falls ja: } \overline{b}' := 0, \text{ sonst: } \overline{b}' := 1 \\ \text{Ausgabe: } \overline{b}' \end{array} \right\}$$

Da $\mathcal{B}_4$ den Masterkey kennt, kann er die Überprüfung leicht durchführen. Außerdem kann das Orakel wie vorgesehen betrieben werden:

$$SimDecryptOracle_4(com_i, \overline{c}_i, \sigma_i) := DecryptOracle_2(com_i, \overline{c}_i, \sigma_i)$$

Die Laufzeit von $\mathcal{B}_4$ ist im Wesentlichen beschränkt durch $t_4 := T(\mathsf{Setup}) + T(\mathsf{Encrypt}) + T(\mathsf{Sign}) + t + q \cdot (T(\mathsf{Keygen}) + T(\mathsf{Decrypt}) + T(\mathsf{Decommit}) + T(\mathsf{Verify})) + q \cdot T(\mathsf{Verify})$ und damit polynomiell.

Es bleibt der Vorteil von $\mathcal{B}_4$ zu bestimmen. Wie im Beweis von Lemma 3.8 entscheidet die Wahl von $\overline{b}$, ob die Sicht von $\mathcal{A}$ wie in Experiment 2 oder 3 ist. Man beachte, dass am Ende von $\mathcal{B}_4$ für den Fall $\overline{b} = 0$ gerade geprüft wird, ob FORGE eingetreten ist, und für den Fall $\overline{b} = 1$, ob FORGE' eingetreten ist. Zusammen ergibt sich ganz analog:

$$\epsilon_4(1^k) := Adv^{\text{Vertraulichkeit}}_{\mathcal{B}_4,\mathcal{ENC}}(1^k) = \ldots = \frac{1}{2}\left|Pr\left[E_2^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right] - Pr\left[E_3^{\mathcal{A}}(1^k) : \mathsf{FORGE}'\right]\right|$$

B_4 ist also ein (t_4, ϵ_4)-Angreifer der Vertraulichkeit von $\mathcal{ENC}$, wobei t_4 polynomiell beschränkt und $\epsilon_4(1^k) = \frac{1}{2}\left|Pr\left[E_2^{\mathcal{A}}(1^k) : \mathsf{FORGE}\right] - Pr\left[E_3^{\mathcal{A}}(1^k) : \mathsf{FORGE}'\right]\right|$ ist.

Beweis von Lemma 3.10

$\mathcal{B}_5$ simuliert für $\mathcal{A}$ den Herausforderer in Experiment 3 und versucht selbst dadurch $\mathcal{MAC}$ zu brechen. $\mathcal{B}_5$ muss also nur mit Hilfe eines SignOracle ein gültiges Paar von Nachricht und Unterschrift finden.
Für die Simulation von Experiment 3 erzeugt $\mathcal{B}_5$ alles bis auf die Unterschriften selbstständig. Die Anfragen, die FORGE eintreten lassen, sind gerade gültige Lösungen für den Angriff von $\mathcal{B}_5$. Allerdings hat $\mathcal{B}_5$ den Schlüssel von $\mathcal{MAC}$ nicht und muss daher raten, welche Anfragen dies sind. $\mathcal{B}_5$ ist wie folgt definiert (siehe Definition 9):

$$\mathcal{B}_5(1^k) \quad := \left\{ \begin{array}{l} \text{Wählt den Schritt } i :=_R \{1, \ldots, q(1^k)\}, \text{ bei dem } \mathcal{A} \text{ abgebrochen wird.} \\ (P, M) := \mathsf{Setup}(1^k); \quad \pi := \mathsf{Start}(1^k); \\ (\rho^*, com^*, dec^*) := \mathsf{Commit}(1^k, \pi); \\ \left.\begin{array}{l} (m_0, m_1, s) := \mathcal{A}_{\mathrm{I}}^{SimDecryptOracle_5}((P, \pi)); \\ b :=_R \{0,1\}; \overline{c}^* := \mathsf{Encrypt}(m_b.0^{|dec^*|}, com^*, P); \\ \sigma := \mathsf{SignOracle}(\overline{c}^*); c^* := (com^*, \overline{c}^*, \sigma^*) \\ b' := \mathcal{A}_{\mathrm{II}}^{SimDecryptOracle_5}((P, \pi)); \end{array}\right\} \begin{array}{l} \text{Abbruch nach } i\text{-ter} \\ \text{Anfrage von } \mathcal{A} \text{ an das} \\ SimDecryptOracle_5. \\ \text{Sei } c_i = (com_i, \overline{c}_i, \sigma_i). \end{array} \\ \text{Ausgabe: } (\overline{c}_i, \sigma_i) \end{array} \right\}$$

Da $\mathcal{B}_5$ den Masterkey kennt, kann er das Orakel wie vorgesehen simulieren:

$$SimDecryptOracle_5(com_i, \overline{c}_i, \sigma_i) := DecryptOracle_3(com_i, \overline{c}_i, \sigma_i)$$

Die Laufzeit von $\mathcal{B}_5$ ist beschränkt durch $t_5 := T(\mathsf{Setup}) + T(\mathsf{Start}) + T(\mathsf{Commit}) + T(\mathsf{Encrypt}) + T(\mathsf{SignOracle}) + t + q \cdot (T(\mathsf{Keygen}) + T(\mathsf{Decrypt}) + T(\mathsf{Decommit}) + T(\mathsf{Verify}))$ und damit polynomiell. Je nachdem, in welchem Schritt $\mathcal{A}$ abgebrochen wird (vor oder nach dem Challenge-Schritt), stellt $\mathcal{B}_5$ keine oder eine Anfrage an das $\mathsf{SignOracle}$.

Für die folgende Analyse muss das Ereignis FORGE' weiter zerlegt werden:

$\mathsf{FORGE}'_{\leq j}$: $\exists i \in \{1, \ldots, j\} : com_i = com^* \ \wedge \ \mathsf{Verify}(\overline{c}_i, \sigma_i, \rho) = 1$
„FORGE' tritt innerhalb der ersten j Anfragen ein."

$\mathsf{FORGE}'_j = \ \mathsf{FORGE}'_{\leq j} \cap \overline{\mathsf{FORGE}'_{\leq j-1}}$
„FORGE' tritt zum ersten mal in der j-ten Anfrage ein."

Es gilt: $\mathsf{FORGE}' = \bigcup_{j=1}^{q(1^k)} \mathsf{FORGE}'_j$ und alle FORGE'_j sind paarweise disjunkt:

Nun kann der Vorteil von $\mathcal{B}_5$ bestimmt werden:

$$
\begin{aligned}
& Adv^{\text{Fälschung}}_{\mathcal{B}_5, \mathcal{MAC}}(1^k) && =: \epsilon_5(1^k) \\
=& Pr \left[\begin{array}{l} \rho := \mathsf{Init}(1^k); \\ (\overline{c}_i, \sigma_i) := \mathcal{B}_5^{\mathsf{SignOracle}}(1^k) \end{array} \ : \ \begin{array}{l} \mathsf{Verify}(\overline{c}_i, \sigma_i, \rho) = 1 \\ \wedge\ (\overline{c}_i, \sigma_i) \neq (\overline{c}^*, \sigma^*) \\ \text{[falls die Anfrage } (\overline{c}^*, \sigma^*) \text{ an} \\ \text{das } \mathsf{SignOracle} \text{ gestellt wird]} \end{array} \right]
\end{aligned}
$$

Einsetzen der Definition von $\mathcal{B}_5$, Austauschen von $\mathsf{SignOracle}$ durch Sign (beide liefern gleiche Wahrscheinlichkeitsverteilung) und Umsortieren liefert:

$$
=Pr \left[\begin{array}{l} i :=_R \{1, \ldots, q(1^k)\}; \quad E_3^{\mathcal{A}}(1^k); \\ \text{Abbruch nach } i\text{-ter Anfrage von } \mathcal{A}. \end{array} \ : \ \begin{array}{l} \mathsf{Verify}(\overline{c}_i, \sigma_i, \rho) = 1 \\ \wedge\ (\overline{c}_i, \sigma_i) \neq (\overline{c}^*, \sigma^*) \\ \text{[falls die Anfrage } (\overline{c}^*, \sigma^*) \text{ an} \\ \text{das } \mathsf{SignOracle} \text{ gestellt wird]} \end{array} \right]
$$

Nun gibt es viele Fälle, in denen die von $\mathcal{B}_5$ gefundene Lösung richtig sein kann. Auf jeden Fall ist sie aber richtig, wenn FORGE'_i eintritt:
Nach Definition gilt schon $\mathsf{Verify}(\overline{c}_i, \sigma_i, \rho) = 1$. Es bleibt aber auch noch zu überprüfen, ob für den Fall, dass von $\mathcal{B}_5$ die Anfrage $(\overline{c}^*, \sigma^*)$ an das $\mathsf{SignOracle}$ gestellt wird, die zweite Bedingung $(\overline{c}_i, \sigma_i) \neq (\overline{c}^*, \sigma^*)$ auch erfüllt wird. $\mathcal{B}_5$ stellt maximal eine Anfrage – in der Challenge-Phase. Falls die i-te Anfrage von $\mathcal{A}$ an das $\mathsf{DecryptOracle}$ vor der Challenge-Phase stattfindet, wird also $\mathsf{SignOracle}$ nie benutzt – dieser Fall ist also kein Problem.
Angenommen die i-te Anfrage von $\mathcal{A}$ fände nach der Challenge-Phase stattfindet und es gälte außerdem $(\overline{c}_i, \sigma_i) = (\overline{c}^*, \sigma^*)$. Wegen FORGE'_i müsste auch $com_i = com^*$ und somit insgesamt $(com_i, \overline{c}_i, \sigma_i) = (com^*, \overline{c}^*, \sigma^*)$ gelten. Das bedeutete aber, dass $\mathcal{A}$ die Entschlüsselung des Challenge-Chiffretextes angefordert hätte, was nicht zulässig ist.

Insgesamt gilt dann weiter:

$$\geq \sum_{j=1}^{q(1^k)} Pr\left[i :=_R \{1,\ldots,q(1^k)\}; E_3^{\mathcal{A}}(1^k) : i = j \ \wedge \ \mathsf{FORGE}'_j\right]$$

Man beachte, dass das Eintreten von FORGE'_j völlig unabhängig von der Wahl von i ist:

$$= \sum_{j=1}^{q(1^k)} Pr\left[E_3^{\mathcal{A}}(1^k) : \mathsf{FORGE}'_j\right] \cdot \underbrace{Pr\left[i :=_R \{1,\ldots,q(1^k)\} : i = j\right]}_{\frac{1}{q(1^k)}}$$

$$= \frac{Pr\left[E_3^{\mathcal{A}}(1^k) : \mathsf{FORGE}'\right]}{q(1^k)}$$

Insgesamt ist B_5 also ein $(t_5, 1, \epsilon_5)$-Angreifer von $\mathcal{MAC}$, wobei t_5 polynomiell beschränkt und $\epsilon_5(1^k) = \frac{Pr\left[E_3^{\mathcal{A}}(1^k):\mathsf{FORGE}'\right]}{q(1^k)}$ ist.

4 Konkrete Umsetzungen mit Boneh-Boyen-IBE

In diesem Kapitel soll untersucht werden, wie man obige Konstruktion auf ein konkretes IBE anwenden kann. Dieses IBE wird das von Boneh und Boyen sein – genauer: eine Vereinfachung des ersten Verfahren aus [BB04]. Zunächst wird Boneh-Boyen-IBE in Abschnitt 4.1 vorgestellt. In Abschnitt 4.2 wird ein Vorschlag von Boneh und Katz erläutert, wie man Boneh-Boyen-IBE zusammen mit einem effizienten ENC und einem nicht näher spezifizierten MAC zusammensetzen kann. Im Abschnitt 4.3 wird ein alternatives ENC konstruiert, welches der Struktur von Boneh-Boyen-IBE mehr ähnelt. Die Hoffnung dabei ist, dass man unter Ausnutzen der strukturellen Gemeinsamkeiten das resultierende PKE verbessern kann. Leider konnte keine Verbesserung erzielt werden.

4.1 Boneh-Boyen-IBE

Das PKE von Boneh und Boyen basiert auf bilinearen Gruppen (siehe Kapitel 2.4). Der IND-sID-CPA-Sicherheit liegt das bilineare Diffie-Hellman-Entscheidungsproblem (BDDHP, siehe Definition 15 auf Seite 24) zugrunde. Die zentrale Idee ist, dass man in der Simulation unter bestimmten Umständen trotz Unkenntnis des Masterkeys die privaten Schlüssel zu fast allen Identitäten berechnen kann. Genaueres dazu folgt im Sicherheitsbeweis. Sei Gen ein bilinearer Gruppengenerator (siehe Definition 14 auf Seite 22), BB-$\mathcal{IBE}$ ist dann wie folgt definiert:

$\mathsf{Setup}(1^k)$: Es werden mittels $(p, \mathbb{G}, \mathbb{G}_T, e) := \mathsf{Gen}(1^k)$ zwei Gruppen mit bilinearer Abbildung bestimmt, ein erzeugendes Element $g :=_R \mathbb{G}^*$ sowie ein Exponent $M :=_R \mathbb{Z}_p$ gewählt. Dann wird $g_1 := g^M$ gesetzt, $g_2, g_3 :=_R \mathbb{G}$ gewählt und $Z := e(g_1, g_2) \in \mathbb{G}_T$ gesetzt.
Die öffentlichen Systemparameter sind $P = ((p, \mathbb{G}, \mathbb{G}_T, e), g, g_1, g_2, g_3, Z)$ und der geheime Masterkey ist der Exponent M. Komme die Ausgabe $(p, \mathbb{G}, \mathbb{G}_T, e)$ von Gen aus einer nicht näher spezifizierten Menge G aller Beschreibungen von bilinearen Gruppen. Dann ist $(G \times \mathbb{G}^* \times \mathbb{G} \times \mathbb{G} \times \mathbb{G} \times \mathbb{G}_T) \times \mathbb{Z}_p$ der Parameterraum $\mathcal{P}$. Der Klartextraum $\mathcal{M}$ ist $\mathbb{G}_T$, der Chiffretextraum $\mathcal{C}$ ist $\mathbb{G}_T \times \mathbb{G} \times \mathbb{G}$ und $\mathbb{Z}_p \times (\mathbb{G} \times \mathbb{G})$ ist der Schlüsselraum $\mathcal{K}$. Die Identitäten stammen in leichter Abweichung zu Definition 6 aus $\mathbb{Z}_p$.

$\mathsf{Keygen}(ID, M, P)$: Es ist $ID, M \in \mathbb{Z}_p$. Zunächst wird ein Exponent $k :=_R \mathbb{Z}_p$ gewählt. Der private Schlüssel ist dann $D_{ID} := (g_2^M \cdot (g_3 \cdot g_1^{ID})^k, g^k) = (D_0, D_1)$.

$\mathsf{Encrypt}(m, ID, P)$: Es ist $m \in \mathbb{G}_T$ und $ID \in \mathbb{Z}_p$. Zunächst wird ein Exponent $r :=_R \mathbb{Z}_p$ gewählt. Der Chiffretext ist dann $c := (m \cdot Z^r, g^r, (g_3 \cdot g_1^{ID})^r) = (c_0, c_1, c_2)$.

$\mathsf{Decrypt}(c, D_{ID}, P)$: Die ursprüngliche Nachricht ist $m := c_0 \cdot \frac{e(D_1, c_2)}{e(c_1, D_0)}$.

Lemma 4.1

BB-$\mathcal{IBE}$ ist korrekt.

Beweis

Seien $(P, M) \in \mathcal{P}$ beliebige Parameter, die von $\mathsf{Setup}(1^k)$ erzeugt wurden. Sei $ID \in \mathbb{Z}_p$ eine beliebige Identität und $D_{ID} = (D_0, D_1) = (g_2^M \cdot (g_3 \cdot g_1^{ID})^k, g^k)$ der zugehörige private Schlüssel. Sei m eine Nachricht und $c = (c_0, c_1, c_2) = (m \cdot Z^r, g^r, (g_3 \cdot g_1^{ID})^r)$ der zugehörige Chiffretext. Es ist zu zeigen, dass $\mathsf{Decrypt}(c, D_{ID}, P) = m$ ist:

$$
\begin{aligned}
& \mathsf{Decrypt}(c, D_{ID}, P) \\
= \; & c_0 \cdot \frac{e(D_1, c_2)}{e(c_1, D_0)} \\
= \; & m \cdot Z^r \cdot \frac{e(g^k, (g_3 \cdot g_1^{ID})^r)}{e(g^r, g_2^M \cdot (g_3 \cdot g_1^{ID})^k} \\
\overset{(\star)}{=} \; & m \cdot Z^r \cdot \frac{e(g^k, (g_3 \cdot g_1^{ID})^r)}{e(g^r, g_2^M) \cdot e(g^r, (g_3 \cdot g_1^{ID})^k)} \\
= \; & m \cdot \frac{e(\overset{=g^M}{g_1}, g_2)^r}{e(g^r, g_2^M)} \cdot \frac{e(g^k, (g_3 \cdot g_1^{ID})^r)}{e(g^r, (g_3 \cdot g_1^{ID})^k)} \\
\overset{(\star)}{=} \; & m \cdot \frac{e(g, g_2)^{rM}}{e(g, g_2)^{rM}} \cdot \frac{e(g, g_3 \cdot g_1^{ID})^{rk}}{e(g, g_3 \cdot g_1^{ID})^{rk}} \; = \; m
\end{aligned}
$$

Dabei geht in die mit $(\star)$ gekennzeichneten Stellen die Bilinearität von e ein, und zwar einmal die Bedingung aus der Definition und einmal die aus Lemma 2.3.

Lemma 4.2

Sei $\mathcal{A}$ ein (t, q, ϵ)-IND-sID-CPA-Angreifer von BB-$\mathcal{IBE}$, dann lässt sich mit $\mathcal{A}$ ein Algorithmus $\mathcal{B}$ konstruieren, der ein $(t', \frac{\epsilon}{2})$-BDDHP-Angreifer von Gen ist. Dabei ist t' polynomiell beschränkt.

Folgerung 4.3

Mit Lemma 4.2 und Lemma 2.2 folgt die IND-sID-CPA-Sicherheit von BB-$\mathcal{IBE}$, falls das BDDHP in den von Gen erzeugten Gruppen hart ist.

Beweis von Lemma 4.2

$\mathcal{B}$ simuliert für $\mathcal{A}$ den Herausforderer in einem IND-sID-CPA-Experiment und versucht selbst dadurch das BDDHP in einer von Gen erzeugten Gruppe zu lösen. Als Eingabe bekommt $\mathcal{B}$ eine Gruppenbeschreibung $(p, \mathbb{G}, \mathbb{G}_T, e)$, die vier Gruppenelemente $g, x = g^\alpha, y = g^\beta, z = g^\gamma \in \mathbb{G}$ und ein $T_{\overline{b}} \in \mathbb{G}_T$. $\mathcal{B}$ muss entscheiden, ob entweder $T_{\overline{b}} = e(g, g)^{\alpha\beta\gamma}$ ist oder zufällig aus $\mathbb{G}_T$ gewählt wurde, d.h. ob $\overline{b} = 0$ oder $\overline{b} = 1$ ist.

Nachdem $\mathcal{B}$ die angegriffene Identität ID^* von $\mathcal{A}$ bekommen hat, kann er die Systemparameter von BB-$\mathcal{IBE}$ aus den Eingabewerten und mit selbst erzeugten Gruppenelementen bilden. Diese lassen sich so konstruieren, dass zu jeder Identität ungleich ID^* ein gültiger privater Schlüssel berechnet werden kann, obwohl $\mathcal{B}$ den Masterkey nicht kennt. Anhand des Erfolgs von $\mathcal{A}$ kann $\mathcal{B}$ herausfinden, auf welche Weise $T_{\overline{b}}$ gebildet wurde.
$\mathcal{B}$ ist wie folgt definiert (siehe Definition 15 auf Seite 24):

$$\mathcal{B}(p,\mathbb{G},\mathbb{G}_T,e,g,x,y,z,T_{\overline{b}}) \quad := \left\{ \begin{array}{l} g \text{ wird übernommen; } g_1 := x; g_2 := y; Z := e(g_1,g_2); \\ (ID^*, s_0) := \mathcal{A}_0(1^k); t :=_R \mathbb{Z}_p; g_3 := g^t \cdot g_1^{-ID^*}); \\ P := (p,\mathbb{G},\mathbb{G}_T,e,g,g_1,g_2,g_3,Z); \\ (m_0,m_1,s) := \mathcal{A}_{\mathrm{I}}^{SimKeygenOracle}(P,s_0); \\ b :=_R \{0,1\}; c^* := (m_b \cdot T_{\overline{b}}, z, z^t) \\ b' := \mathcal{A}_{\mathrm{II}}^{SimKeygenOracle}(c^*,s) \\ \text{Falls } b = b' \text{: } \overline{b'} := 0, \text{ sonst } \overline{b'} := 1 \\ \text{Ausgabe: } \overline{b'} \end{array} \right\}$$

Dabei arbeitet das $SimKeygenOracle$ wie folgt (man beachte, dass die Anfrage nach dem privaten Schlüssel zur angegriffenen Identität, also $ID_i = ID^*$, nicht erlaubt ist und somit im Exponenten niemals durch $ID_i - ID^* = 0$ geteilt werden kann):

$$SimKeygenOracle(ID_i) \quad := \left\{ \begin{array}{l} k' :=_R \mathbb{Z}_p \\ D_{ID_i} := \left(g_2^{\frac{-t}{ID_i - ID^*}} \cdot (g_3 \cdot g_1^{ID_i})^{k'}, g_2^{\frac{-1}{ID_i - ID^*}} \cdot g^{k'} \right) \\ \text{Ausgabe } (ID_i, D_{ID_i}) \end{array} \right\}$$

Offenbar ist die Laufzeit $t' := T(\mathcal{B})$ im Wesentlichen durch die Laufzeit von $\mathcal{A}$ und einigen Gruppenoperationen und -exponentiationen bestimmt, also polynomiell.

Im Gegensatz zu allen vorherigen Beweisen ist hier nicht klar ersichtlich, dass $\mathcal{B}$ sich wie erwartet verhält (also dass alle Eingabewerte von $\mathcal{A}$ die korrekte Wahrscheinlichkeitsverteilung haben). Daher soll dies hier ausführlich untersucht werden. Zunächst muss aber noch ein Argument erläutert werden, welches im Folgenden mehrfach benötigt wird:

Lemma 4.4

Gegeben sei eine beliebige endliche Gruppe G mit Verknüpfung $\star$. In einem Zufallsexperiment werde $c \in G$ bezüglich einer beliebigen Wahrscheinlichkeitsverteilung ausgewählt. Weiterhin werde ein Element $r \in G$ zufällig und gleichverteilt gewählt. Dann ist der Wert $r' := c \star r$ zufällig und gleichverteilt aus G und unabhängig von c.

Beweis

Seien $x, y \in G$ beliebig. Es ist:

$$\begin{aligned}
&Pr\left[r' = x \mid c = y\right] \\
=&Pr\left[c \star r = x \mid c = y\right] && \text{Setze } r' = c \star r \text{ ein.} \\
=&Pr\left[r = y^{-1} \star x \mid c = y\right] && \text{Beachte } c = y \text{ und bringe } y \text{ auf andere Seite.} \\
=&Pr\left[r = y^{-1} \star x\right] && r \text{ wird unabhängig von } c \text{ gewählt.} \\
=&\tfrac{1}{|G|} && r \text{ wird gleichverteilt in } G \text{ gewählt.}
\end{aligned}$$

Weiterhin ist:

$$\begin{aligned}
&Pr\left[r' = x\right] \\
=&\sum_{y \in G} \underbrace{Pr\left[r' = x \mid c = y\right]}_{= \frac{1}{|G|}} \cdot Pr\left[c = y\right] \\
=&\tfrac{1}{|G|} \cdot \underbrace{\sum_{y \in G} Pr\left[c = y\right]}_{= 1} \\
=&\tfrac{1}{|G|}
\end{aligned}$$

Beides zusammengeführt ergibt:

$$Pr\left[r' = x \mid c = y\right] = \frac{1}{|G|} = Pr\left[r' = x\right]$$

Also ist r' von c unabhängig und außerdem zufällig und gleichverteilt in G gewählt.

Beweis von Lemma 4.2 (Fortsetzung)

Nun kann die Untersuchung der Wahrscheinlichkeitsverteilungen aller Werte beginnen. Für die öffentlichen Systemparameter $P = (p, \mathbb{G}, \mathbb{G}_T, e, g, g_1, g_2, g_3, Z)$ und den Masterkey M gilt:

$p, \mathbb{G}, \mathbb{G}_T, e$	Die Gruppen werden nicht von $\mathcal{B}$ bestimmt, sondern $\mathcal{B}$ nur als Aufrufparameter übergeben. Nach Definition 15 werden sie allerdings mittels Gen gebildet, also wie vorgeschrieben.
g	Auch das erzeugende Element wird nicht von $\mathcal{B}$ selbst, aber trotzdem exakt wie vorgeschrieben ausgewählt (siehe Definition 15 auf Seite 24).
g_1	Der Wert soll von der Form g^M sein, wobei M zufällig und gleichverteilt aus $\mathbb{Z}_p$ kommt. Es ist $g_1 := x = g^\alpha$ und α wie gefordert gewählt.

g_2	Der Wert soll zufällig und gleichverteilt aus $\mathbb{G}$ gewählt werden. Da β zufällig und gleichverteilt aus $\mathbb{Z}_p$ und g ein erzeugendes Element der Gruppe (mit Ordnung p) ist, ist $g_2 := y = g^\beta$ ebenfalls zufällig und gleichverteilt in $\mathbb{G}$.
g_3	Auch dieser Wert soll zufällig und gleichverteilt aus $\mathbb{G}$ gewählt werden. Da g^t zufällig und gleichverteilt in $\mathbb{G}$ ist, folgt mit Lemma 4.4, dass auch $g_3 := g^t \cdot g_1^{-ID^*}$ gleichverteilt und unabhängig in $\mathbb{G}$ gewählt ist.
Z	Dieser Wert wird exakt wie vorgegeben gebildet.
M	Der Wert soll zufällig und gleichverteilt aus $\mathbb{Z}_p$ gewählt werden. In der Simulation wird M nicht explizit gewählt, die Wahl von $g_1 := x = g^\alpha$ impliziert allerdings $\alpha = M$. Da α zufällig und gleichverteilt aus $\mathbb{Z}_p$ gewählt ist, hat damit auch M die richtige Wahrscheinlichkeitsverteilung.

Für die Antworten $(ID_i, D_{ID_i}) = (ID_i, D_0, D_1)$ des *KeygenOracle* gilt:

ID_i	Ist trivialerweise wie erwartet gesetzt.
D_1	Der Wert soll von der Form g^k sein, wobei k zufällig und gleichverteilt aus $\mathbb{Z}_p$ kommt. Es ist $D_1 := g_2^{\frac{-1}{ID_i - ID^*}} \cdot g^{k'} = (g^\beta)^{\frac{-1}{ID_i - ID^*}} \cdot g^{k'} = g^{\frac{-\beta}{ID_i - ID^*} + k'}$. Dies impliziert also $k = \frac{-\beta}{ID_i - ID^*} + k'$. Da k' zufällig und gleichverteilt gewählt ist, gilt dies nach Lemma 4.4 auch für k.
D_0	D_0 soll von der Form $g_2^M \cdot (g_3 \cdot g_1^{ID_i})^k$ sein, wobei nach vorherigen Ausführungen $k = \frac{-\beta}{ID_i - ID^*} + k'$ und somit auch $k' = \frac{\beta}{ID_i - ID^*} + k$ gilt. Berücksichtigt man zusätzlich $g_1 = g^\alpha$, $g_2 = g^\beta$ und $g_3 = g^t \cdot g_1^{-ID^*} = g^t \cdot (g^\alpha)^{-ID^*}$, dann ist $\begin{aligned} & D_0 := g_2^{\frac{-t}{ID_i - ID^*}} \cdot (g_3 \cdot g_1^{ID_i})^{k'} \\ \overset{(\star)}{=} \; & g_2^{\frac{-t}{ID_i - ID^*}} \cdot (g_3 \cdot g_1^{ID_i})^{\frac{\beta}{ID_i - ID^*}} \cdot (g_3 \cdot g_1^{ID_i})^k \\ \overset{(\star\star)}{=} \; & g_2^{\frac{-t}{ID_i - ID^*}} \cdot (g^t \cdot (g^\alpha)^{-ID^*} \cdot (g^\alpha)^{ID_i})^{\frac{\beta}{ID_i - ID^*}} \cdot (g_3 \cdot g_1^{ID_i})^k \\ = \; & g_2^{\frac{-t}{ID_i - ID^*}} \cdot \underbrace{(g^t)^{\frac{\beta}{ID_i - ID^*}}}_{= g_2^{\frac{t}{ID_i - ID^*}}} \cdot \underbrace{((g^\alpha)^{-ID^*} \cdot (g^\alpha)^{ID_i})^{\frac{\beta}{ID_i - ID^*}}}_{= (g^\alpha)^{ID_i - ID^*}} \cdot (g_3 \cdot g_1^{ID_i})^k \\ = \; & 1 \cdot g^{\alpha \cdot \beta} \cdot (g_3 \cdot g_1^{ID_i})^k \\ \overset{(\star\star)}{=} \; & g_2^\alpha \cdot (g_3 \cdot g_1^{ID_i})^k = g_2^M \cdot (g_3 \cdot g_1^{ID_i})^k \end{aligned}$ Dabei wurde in $(\star)$ k' durch $\frac{\beta}{ID_i - ID^*} + k$ ersetzt und in $(\star\star)$ die Werte g_1, g_2, g_3 teilweise durch Potenzen von g ersetzt (bzw. anders herum). Insgesamt gilt, dass D_0 genau die geforderte Form $g_2^M \cdot (g_3 \cdot g_1^{ID_i})^k$ hat.

Im Challenge-Schritt gilt für den Chiffretext $c^* = (c_0, c_1, c_2)$:

c_1	Der Wert soll von der Form g^r sein, wobei r zufällig und gleichverteilt aus $\mathbb{Z}_p$ kommt. Es ist $c_1 := z = g^\gamma$ und γ wie gefordert gewählt.
c_2	Es ist $c_2 := z^t = (g^\gamma)^t = (g^t)^\gamma = (g^t \cdot g_1^{-ID^*} \cdot g_1^{ID^*})^\gamma = (g_3 \cdot g_1^{ID^*})^\gamma$ wie gefordert gesetzt.
c_0	Entweder ist $T_{\overline{b}} = e(g,g)^{\alpha\beta\gamma} = e(g^\alpha, g^\beta)^\gamma = e(g_1, g_2)^\gamma = Z^\gamma$. Dann ist $c_0 = m_b \cdot T_{\overline{b}} = m_b \cdot Z^\gamma$ von der gewünschten Form. Andernfalls ist $T_{\overline{b}}$ zufällig und gleichverteilt aus $\mathbb{G}_T$. Nach Lemma 4.4 ist c_0 dann insbesondere unabhängig von b, was im Folgenden sehr wichtig sein wird.

Insgesamt gilt für den Fall $T_{\overline{b}} = e(g,g)^{\alpha\beta\gamma}$ also, dass die Simulation, die $\mathcal{B}$ liefert, und das IND-CPA-Experiment identisch ablaufen. Für den Fall, dass $T_{\overline{b}}$ zufällig gewählt ist, ist dies natürlich nicht der Fall. Hier ist es allerdings so, dass sämtliche Werte, die zu der Sicht von $\mathcal{A}$ gehören (das sind die öffentlichen Systemparameter, alle Antworten des *KeygenOracle* sowie der Challenge-Chiffretext), unabhängig von b sind. Somit muss auch b', die Ausgabe von $\mathcal{A}$, unabhängig von b sein.

Nun kann der Vorteil von $\mathcal{B}$ bestimmt werden.

$$
\begin{aligned}
& Adv^{BDDHP}_{\mathcal{B},\mathsf{Gen}}(1^k) \\
= & \left| Pr \left[\begin{array}{l} (p, \mathbb{G}, \mathbb{G}_T, e) := \mathsf{Gen}(1^k); g :=_R \mathbb{G}^*; \\ \alpha, \beta, \gamma :=_R \{1, \ldots, p\}; (x, y, z) := (g^\alpha, g^\beta, g^\gamma); \\ T_0 := e(g,g)^{\alpha\beta\gamma}; T_1 :=_R \mathbb{G}_T; \overline{b} := \{0,1\}; \\ \overline{b'} := \mathcal{B}(p, \mathbb{G}, \mathbb{G}_T, e, g, x, y, z, T_{\overline{b}}) \end{array} : \overline{b'} = \overline{b} \right] - \frac{1}{2} \right|
\end{aligned}
$$

Zunächst macht man eine Fallunterscheidung zu $\overline{b}$. Dieses kann entweder den Wert 0 oder den Wert 1 annehmen. Das Ereignis $\overline{b'} = \overline{b}$ ist im jeweiligen Fall dann gleichbedeutend damit, dass $\overline{b'} = 0$ bzw. $\overline{b'} = 1$ ist.

$$
= \left| Pr\left[\ldots : \overline{b'} = 0 \mid \overline{b} = 0\right] \cdot \underbrace{Pr\left[\ldots : \overline{b} = 0\right]}_{= \frac{1}{2}} + Pr\left[\ldots : \overline{b'} = 1 \mid \overline{b} = 1\right] \cdot \underbrace{Pr\left[\ldots : \overline{b} = 1\right]}_{= \frac{1}{2}} - \tfrac{1}{2} \right|
$$

Zur Erinnerung: $\overline{b}$ und $\overline{b'}$ gehören zum Angriff von $\mathcal{B}$ auf das BDDHP und geben an, wie T gewählt ist. Dahingegen gehören b und b' zum Angriff von $\mathcal{A}$ auf die IND-sID-CPA-Sicherheit von BB-$\mathcal{IBE}$ und geben an, welche Nachricht für den Challenge-Chiffretext gewählt wird. Nach Konstruktion setzt $\mathcal{B}$ genau dann $\overline{b'} := 0$, wenn die Vermutung von $\mathcal{A}$ richtig ist, also $b = b'$ gilt. Im Falle $b \neq b'$ wird $\overline{b'} := 1$ gesetzt.

$$
= \left| \tfrac{1}{2} Pr\left[\ldots : b' = b \mid \overline{b} = 0\right] + \tfrac{1}{2} Pr\left[\ldots : b' \neq b \mid \overline{b} = 1\right] - \tfrac{1}{2} \right|
$$

Nun kann man die $\frac{1}{2}$ außerhalb des Betrags bringen und für das Ereignis $b' \neq b$ eine Fallunterscheidung nach b vornehmen.

$$=\tfrac{1}{2}|Pr\left[\ldots : b' = b \mid \bar{b} = 0\right] + Pr\left[\ldots : b' = 1 \mid b = 0\right] \cdot \underbrace{Pr\left[\ldots : b = 0\right]}_{=\frac{1}{2}} + Pr\left[\ldots : b' = 0 \mid b = 1\right] \cdot \underbrace{Pr\left[\ldots : b = 1\right]}_{=\frac{1}{2}} - 1|$$

Nach den Vorüberlegungen ist die Sicht von $\mathcal{A}$ im Falle $\bar{b} = 0$, also $T_{\bar{b}} = e(g,g)^{\alpha\beta\gamma}$, genau wie im IND-sID-CPA-Experiment. Im anderen Fall ist die Sicht von $\mathcal{A}$ und damit zwangsweise auch die Ausgabe von $\mathcal{A}$, also b', unabhängig von b.

$$=\tfrac{1}{2}|Pr\left[\text{IND-sID-CPA}^{\mathcal{A}}_{BB-\mathcal{IBE}}(1^k) : b = b'\right] + \tfrac{1}{2}\underbrace{\left(Pr\left[\ldots : b' = 1\right] + Pr\left[\ldots : b' = 0\right]\right)}_{=1} - 1|$$

$$=\tfrac{1}{2}|Pr\left[\text{IND-sID-CPA}^{\mathcal{A}}_{BB-\mathcal{IBE}}(1^k) : b = b'\right] - \tfrac{1}{2}|$$

$$=\tfrac{1}{2}Adv^{IND-sID-CPA}_{\mathcal{A},BB-\mathcal{IBE}}(1^k) = \tfrac{1}{2}\,\epsilon(1^k)$$

Insgesamt ist also $\mathcal{B}$ ein $(t', \frac{\epsilon}{2})$-BDDHP-Angreifer von Gen, wobei t' polynomiell ist.

4.2 Eine konkrete Umsetzung: Boneh-Katz-PKE

Um mit Hilfe der allgemeinen Konstruktion ein konkretes PKE zu erhalten, muss man Algorithmen für IBE, ENC und MAC angeben. Boneh und Katz schlagen folgende Umsetzung vor:

IBE: Es wird BB-$\mathcal{IBE}$ mit einer leichten Modifikation der Algorithmen Encrypt und Decrypt verwendet. Die Notwendigkeit der Modifikation liegt darin, dass nach dem allgemeinen Verfahren aus Kapitel 3 mit dem $\mathcal{IBE}$ die Konkatenation der eigentlichen Nachricht m mit dec verschlüsselt werden muss. Diese Konkatenation wäre zu lang, um durch ein einzelnes Element aus $\mathbb{G}_T$ maskiert zu werden. Die Modifikation ist wie folgt:

Bisher wurde bei der Verschlüsselung eine Nachricht $m \in \mathbb{G}_T$ durch Multiplikation von Z^r maskiert. Nun sind m und dec Bitstrings, die per bitweisem XOR mit $G(Z^r)$ maskiert werden. Dabei ist $G : \mathbb{G}_T \to \{0,1\}^*$ ein Pseudozufallszahlengenerator mit ausreichend großer Ausgabelänge. Bei der Verschlüsselung wird also $c = (c_0, c_1, c_2) := (m.dec \oplus G(Z^r), \ldots, \ldots)$ gesetzt, wobei $m \in \{0,1\}^k$ ist.

Bei der Entschlüsselung wird $m.dec := c_0 \oplus G(\frac{e(c_1,D_0)}{e(D_1,c_2)})$ statt $m := c_0 \cdot \frac{e(D_1,c_2)}{e(c_1,D_0)}$ gesetzt. Man beachte, dass sich hier die Reihenfolge von Zähler und Nenner vertauscht hat. Nach Korrektheit von BB-$\mathcal{IBE}$ ist nämlich $\frac{e(D_1,c_2)}{e(c_1,D_0)} = Z^{-r}$ und demnach $\frac{e(c_1,D_0)}{e(D_1,c_2)} = Z^r$. Für die Modifikation folgt damit:

$$m.dec := c_0 \oplus G(\tfrac{e(c_1,D_0)}{e(D_1,c_2)}) = c_0 \oplus G(Z^r) = m.dec \oplus G(Z^r) \oplus G(Z^r) = m.dec.$$

MAC: Die Wahl des $\mathcal{MAC}$ ist laut Boneh und Katz nicht ganz so entscheidend, da die Laufzeit- und Platzkosten im wesentlichen durch das $\mathcal{IBE}$ bestimmt werden. Der Vollständigkeit halber schlagen sie einen CBC-MAC mit AES als zugrundeliegenden Blockchiffre vor, gehen aber sonst nicht weiter darauf ein.

ENC: Das von Boneh und Katz vorgeschlagene Schema (im Folgenden Hash-$\mathcal{ENC}$ genannt) basiert auf einer Familie von universellen Einweg-Hashfunktionen (UOWHF) $\{H_s : \{0,1\}^{k_1} \to \{0,1\}^k\}$ (siehe [NY89]), wobei $k_1 \geq 3k$ ist. Es ist durch folgende Algorithmen definiert:

$\mathsf{Start}(1^k)$ Es wird eine Hashfunktion h aus einer Familie von paarweise unabhängigen Hashfunktionen $\{h : \{0,1\}^{k_1} \to \{0,1\}^k\}$ ausgewählt. Weiterhin wird ein Schlüssel s bestimmt, welcher die Hashfunktion H_s aus der Familie der universellen Einweg-Hashfunktionen definiert. Die öffentlichen Parameter sind $\pi := (h, s)$.

$\mathsf{Commit}(1^k, \pi)$ Zunächst wird ein zufälliges $x :=_R \{0,1\}^{k_1}$ gewählt. Die Ausgabe ist $(\rho, com, dec) := (h(x), H_s(x), x)$.

$\mathsf{Decommit}(\pi, com, dec)$ Es wird überprüft, ob $com = H_s(dec)$ ist. Wenn dies der Fall ist, wird $\rho := h(dec)$ andernfalls $\perp$ ausgegeben.

Lemma 4.5

Hash-$\mathcal{ENC}$ ist korrekt.

Beweis

Für alle nach Start gewählten h und H_s und alle nach Commit gebildeten $(\rho, com, dec) = (h(x), H_s(x), x)$ gilt $com = H_s(x) = H_s(dec)$ (die Überprüfung in Decommit ist also erfolgreich – es wird nicht $\perp$ ausgegeben) sowie $h(dec) = h(x) = \rho$ (es wird also der korrekte Wert für ρ ausgegeben).

Lemma 4.6

Hash-$\mathcal{ENC}$ ist sicher, falls universelle Einweg-Hashfunktionen existieren.
(Der Sicherheitsbeweis kann in [BCHK06, Theorem 4] nachgelesen werden.)

Alles zusammen: Nun kann man mit diesen drei Verfahren gemäß der Konstruktion aus Kapitel 3.2 das BK-$\mathcal{PKE}$ konstruieren. Dabei sei Gen ein bilinearer Gruppengenerator, $\{h : \{0,1\}^{k_1} \to \{0,1\}^k\}$ eine Familie von paarweise unabhängigen Hashfunktionen (mit $k_1 \geq 3k$), $\{H_s : \{0,1\}^{k_1} \to \{0,1\}^k\}$ eine Familie von universellen Einweg-Hashfunktionen, $G : \mathbb{G}_T \to \{0,1\}^{k+k_1}$ ein Pseudozufallszahlengenerator und eine Abbildung $map : \{0,1\}^k \to \mathbb{Z}_p$, welche die Commitmentstrings com auf Identitäten abbildet. Die drei Algorithmen Keygen, Encrypt und Decrypt funktionieren wie folgt:

$\mathsf{Keygen}(1^k)$ Wie in der Definition von BB-$\mathcal{IBE}$ werden $(p, \mathbb{G}, \mathbb{G}_T, e) := \mathsf{Gen}(1^k)$ und ein erzeugendes Element $g :=_R \mathbb{G}^*$ sowie ein Exponent $M :=_R \mathbb{Z}_p$ gewählt. Dann wird $g_1 := g^M$ gesetzt, $g_2, g_3 :=_R \mathbb{G}$ gewählt und $Z := e(g_1, g_2) \in \mathbb{G}_T$ gesetzt.
Wie in der Definition von Hash-$\mathcal{ENC}$ wird eine Hashfunktion h aus der Familie $\{h\}$ und ein Schlüssel s bestimmt, welcher eine Hashfunktion aus der Familie $\{H_s\}$ definiert.
Der öffentliche Schlüssel ist $E = (p, \mathbb{G}, \mathbb{G}_T, e, g, g_1, g_2, g_3, Z, h, s)$ und der private Schlüssel ist der Exponent M.

$\mathsf{Encrypt}(m, E)$ Wie in der Definition von Hash-$\mathcal{ENC}$ wird ein zufälliges $x :=_R \{0,1\}^{k_1}$ gewählt und $(\rho, com, dec) := (h(x), H_s(x), x)$ gesetzt.
Anschließend wird die Nachricht m mit dec konkateniert und mit der Identität $ID := map(com)$ verschlüsselt: dazu wird zunächst ein zufälliges $r :=_R \mathbb{Z}_p$ gewählt und dann $\overline{c} := (m.dec \oplus G(Z^r), g^r, (g_3 \cdot g_1^{ID})^r)$ gesetzt. Dieses wird nun mit Hilfe des MAC und dem Schlüssel ρ unterschrieben: $\sigma := \mathsf{Sign}(\overline{c}, \rho)$.
Der Chiffretext ist $(com, \overline{c}, \sigma) = (H_s(x), (m.x \oplus G(Z^r), g^r, (g_3 \cdot g_1^{map(H_s(x))})^r), \sigma)$. Sei $\mathcal{S}$ der Unterschriftenraum von MAC, dann ist der Chiffretext ein Element aus $\underbrace{\{0,1\}^k}_{com} \times (\underbrace{\{0,1\}^{k+k_1} \times \mathbb{G} \times \mathbb{G}}_{\overline{c}}) \times \underbrace{\mathcal{S}}_{\sigma}$.

$\mathsf{Decrypt}(c, D)$ Der Chiffretext wird als $c = (com, \overline{c}, \sigma)$ eingelesen. Zunächst muss der zur „Identität“ com gehörende private Schlüssel bestimmt werden: Es wird ein Exponent $k :=_R \mathbb{Z}_p$ gewählt und $D_{com} := (g_2^M \cdot (g_3 \cdot g_1^{map(com)})^k, g^k)$ gesetzt.
Anschließend wird die „Nachricht“ $m.dec$ wiederhergestellt: Sei dazu der Chiffretext $\overline{c} = (c_0, c_1, c_2)$ und der private Schlüssel $D_{com} = (D_0, D_1)$. Zur Entschlüsselung berechnet man $m.dec := c_0 \oplus G(\frac{e(c_1, D_0)}{e(D_1, c_2)})$.
Nun kann der Unterschriftenschlüssel für MAC bestimmt werden: $\rho := h(dec)$.
Zuletzt wird die Gültigkeit des Chiffretextes und des Unterschriftenschlüssels überprüft. Dazu wird getestet, ob $com = H_s(dec)$ und ob $\mathsf{Verify}(\overline{c}, \sigma, \rho) = 1$ ist. Wenn dies der Fall ist, wird m ausgegeben; ansonsten $\perp$.

Anmerkung: Die hier verwendete ist die einfachste Umsetzung der Entschlüsselung. In [BCHK06, BMW05, §7.4 bzw. §4.1] werden noch effizientere Varianten verwendet, in denen ausgenutzt wird, dass bei der Entschlüsselung nicht nur der zur Identität com gehörende private Schlüssel D_{com} bekannt ist, sondern auch die diskreten Logarithmen von g_1, g_2 und g_3. Für diese Ausführungen soll es jedoch reichen, die ineffizientere aber einfachere Variante zu verwenden.

Satz 4.7

BK-$\mathcal{PKE}$ ist korrekt und IND-CCA2-sicher, wenn das BDDHP in den von Gen erzeugten Gruppen hart ist und universelle Einweg-Hashfunktionen sowie ein stark-fälschungssicherer MAC existieren.

Beweis

Da exakt nach der Konstruktion aus Kapitel 3.2 vorgegangen wurde und die verwendeten Verfahren nach den Lemmata 4.1 und 4.5 korrekt sowie unter obigen Sicherheitsannahmen nach Folgerung 4.3 und Lemma 4.6 sicher sind, muss mit Satz 3.1 und 3.3 auch die Korrektheit und Sicherheit für BK-$\mathcal{PKE}$ gelten.

4.3 Eine alternative Umsetzung: my-$\mathcal{PKE}$

Im eben vorgestellten Verfahren haben die verwendeten BB-$\mathcal{IBE}$ und Hash-$\mathcal{ENC}$ keine Gemeinsamkeiten. Im Rahmen dieser Studienarbeit kam die Idee auf, ein alternatives ENC zu definieren, welches auf der selben bilinearen Gruppe wie BB-$\mathcal{IBE}$ basiert und sich möglichst auch an der Struktur von BB-$\mathcal{IBE}$ orientiert. Die Hoffnung war, auf diese Weise das IBE und das ENC so kombinieren zu können, dass weitere Verbesserungen möglich sind.

Zunächst muss eine zugrundeliegende Sicherheitsannahme für das neue ENC ausgewählt werden. In Kapitel 2.4 wurden einige vorgestellt.
Für den Sicherheitsbeweis muss es möglich sein, einen Angriff auf die Verbindlichkeit oder Vertraulichkeit des ENC in einen Angriff auf die Sicherheitsannahme abzuwandeln. Da der Angriff auf die Vertraulichkeit ein Entscheidungsproblem darstellt, während das Problem des diskreten Logarithmus, das Diffie-Hellman-Problem und das bilineare Diffie-Hellman-Problem aber Berechnungsprobleme sind, scheinen diese drei keine guten Kandidaten zu sein.
Das Diffie-Hellman-Entscheidungsproblem wäre eine sehr schlechte Wahl, da dieses nach [BF01, §3] in bilinearen Gruppen grundsätzlich einfach ist. Also bleibt das bilineare Diffie-Hellman-Entscheidungsproblem (BDDHP). Dies scheint für das Vorhaben gut geeignet, da auch BB-$\mathcal{IBE}$ auf dem BDDHP beruht und sich somit vermutlich besser strukturelle Gemeinsamkeiten schaffen lassen.

IBE und MAC: Diese bleiben exakt wie im vorherigen Kapitel beschrieben. Das IBE ist eine leichte Modifikation von BB-$\mathcal{IBE}$ und der MAC wird nicht näher spezifiziert.

ENC: Sei Gen der selbe bilineare Gruppengenerator wie für BB-$\mathcal{IBE}$. Das Encapsulation-Schema BDDHP-$\mathcal{ENC}$ ist durch folgende drei Algorithmen definiert:

$\mathsf{Start}(1^k)$ Es werden zwei Gruppen mit bilinearer Abbildung $(p, \mathbb{G}, \mathbb{G}_T, e) := \mathsf{Gen}(1^k)$ bestimmt und ein erzeugendes Element $g :=_R \mathbb{G}^*$ sowie ein Exponent $M :=_R \mathbb{Z}_p$ gewählt. Dann wird $g_1 := g^M$ gesetzt, $g_2 :=_R \mathbb{G}$ gewählt und $Z := e(g_1, g_2) \in \mathbb{G}_T$ gesetzt. Die öffentlichen Parameter sind $\pi = (p, \mathbb{G}, \mathbb{G}_T, e, g, g_1, g_2, Z)$.
Da M nicht weiter benötigt wird, könnte man g_1 auch direkt zufällig wählen. Um

allerdings die Gemeinsamkeit mit BB-$\mathcal{IBE}$ besser aufzeigen zu können, wird g_1 mit Hilfe von M gebildet.

$\mathsf{Commit}(1^k, \pi)$ Es wird ein Exponent $r :=_R \mathbb{Z}_p$ gewählt und $(\rho, com, dec) := (Z^r, g^r, r)$ zurückgegeben.

$\mathsf{Decommit}(\pi, com, dec)$ Zunächst wird überprüft, ob $g^{dec} = com$ ist. Wenn dies der Fall ist, wird $\rho := Z^{dec}$ ausgegeben; andernfalls $\perp$.

Schon bei der Erzeugung der öffentlichen Parameter erkennt man die Analogie zu BB-$\mathcal{IBE}$. Die öffentlichen Parameter bzw. der öffentliche Schlüssel werden auf identische Weise gebildet, außer dass es bei BDDHP-$\mathcal{ENC}$ kein Element g_3 gibt. Auch zwischen Commit und Encrypt gibt es Parallelen. In beiden Algorithmen wird ein Exponent r gewählt und damit die Elemente Z^r und g^r gebildet. Dabei ist in beiden Fällen g^r öffentlich bekannt (als *com* bzw. als ein Teil des Chiffretextes) und Z^r geheim (als *dec* bzw. als Maskierung der Nachricht).

Lemma 4.8

BDDHP-$\mathcal{ENC}$ ist korrekt.

Beweis

Für alle nach Start gewählten Gruppen und Gruppenelemente und alle nach Commit gebildeten $(\rho, com, dec) = (Z^r, g^r, r)$ gilt $g^{dec} = g^r = com$ (es wird also nicht $\perp$ ausgegeben) sowie $Z^{dec} = Z^r = \rho$ (es wird also der korrekte Wert für ρ ausgegeben).

Lemma 4.9

Für jeden (t, ϵ)-Angreifer der Verbindlichkeit von BDDHP-$\mathcal{ENC}$ ist $\epsilon = 0$.

Für jeden (t, ϵ)-Angreifer der Vertraulichkeit von BDDHP-$\mathcal{ENC}$ lässt sich ein (t, ϵ)-BDDHP-Angreifer der von Gen erzeugten Gruppen konstruieren.

Folgerung 4.10

Mit Satz 4.9 und Lemma 2.2 folgt die Sicherheit von BDDHP-$\mathcal{ENC}$, falls das BDDHP in den von Gen erzeugten Gruppen hart ist.

Beweis von Lemma 4.9

Der Angreifer der Verbindlichkeit bekommt öffentliche Parameter und ein regulär gebildetes Tripel (ρ^*, com^*, dec^*) als Eingabe. Dazu soll er ein weiteres dec' finden, so dass $\mathsf{Decommit}(\pi, com^*, dec') \notin \{\rho^*, \perp\}$ ist.

Falls $dec' = dec^*$, so ist $\mathsf{Decommit}(\pi, com^*, dec') = \rho^*$. Falls $dec' \neq dec^*$, so ist $g^{dec'} \neq g^{dec^*} = com$ (beachte dazu: $dec', dec^* \in \mathbb{Z}_p$, wobei p die Ordnung vom erzeugenden Element g ist). Damit wäre aber $\mathsf{Decommit}(\pi, com^*, dec') = \perp$. Es ist für den Angreifer also unmöglich, ein wie oben gefordertes dec' zu finden.

Ein BDDHP-Angreifer $\mathcal{B}$ lässt sich relativ direkt aus $\mathcal{A}$, einem Angreifer der Vertraulichkeit, konstruieren. Als Eingabe bekommt $\mathcal{B}$ eine Gruppenbeschreibung $(p, \mathbb{G}, \mathbb{G}_T, e)$, die vier Gruppenelemente $g, x = g^\alpha, y = g^\beta, z = g^\gamma \in \mathbb{G}$ und ein $T_b \in \mathbb{G}_T$. $\mathcal{B}$ muss entscheiden, ob entweder $T_b = e(g,g)^{\alpha\beta\gamma}$ ist oder zufällig aus $\mathbb{G}_T$ gewählt wurde.
$\mathcal{B}$ ist wie folgt definiert (siehe Definition 15 auf Seite 24):

$$\mathcal{B}(p, \mathbb{G}, \mathbb{G}_T, e, g, x, y, z, T_b) \quad := \left\{ \begin{array}{l} g_1 := x; g_2 := y; Z := e(g_1, g_2); \\ \pi := (p, \mathbb{G}, \mathbb{G}_T, e, g, g_1, g_2, Z); \\ (\rho^*, com^*, dec^*) := (T_b, z, ?); \quad \text{(eigentlich ist } dec^* = \gamma) \\ b' := \mathcal{A}(\pi, \rho^*, com^*); \\ \text{Ausgabe: } b' \end{array} \right\}$$

Im Wesentlichen führt $\mathcal{B}$ nur $\mathcal{A}$ aus und hat damit Laufzeit t. Der Vorteil von $\mathcal{B}$ ist:

$$\begin{aligned} & Adv^{BDDHP}_{\mathcal{B},\mathsf{Gen}}(1^k) \\ = & \left| Pr \left[\begin{array}{l} (p, \mathbb{G}, \mathbb{G}_T, e) := \mathsf{Gen}(1^k); g :=_R \mathbb{G}^*; \\ \alpha, \beta, \gamma :=_R \{1, \ldots, p\}; (x, y, z) := (g^\alpha, g^\beta, g^\gamma); \\ T_0 := e(g,g)^{\alpha\beta\gamma}; T_1 :=_R \mathbb{G}_T; b := \{0,1\}; \\ b' := \mathcal{B}(p, \mathbb{G}, \mathbb{G}_T, e, g, x, y, z, T_b) \end{array} : b = b' \right] - \frac{1}{2} \right| \end{aligned}$$

Definition von $\mathcal{B}$ einsetzen:

$$= \left| Pr \left[\begin{array}{l} (p, \mathbb{G}, \mathbb{G}_T, e) := \mathsf{Gen}(1^k); g :=_R \mathbb{G}^*; \\ \alpha, \beta, \gamma :=_R \{1, \ldots, p\}; (x, y, z) := (g^\alpha, g^\beta, g^\gamma); \\ T_0 := e(g,g)^{\alpha\beta\gamma}; T_1 :=_R \mathbb{G}_T; b := \{0,1\}; \\ g_1 := x; g_2 := y; Z := e(g_1, g_2); \\ \pi := (p, \mathbb{G}, \mathbb{G}_T, e, g, g_1, g_2, Z); \\ (\rho^*, com^*, dec^*) := (T_b, z, ?); \\ b' := \mathcal{A}(\pi, \rho^*, com^*) \end{array} : b = b' \right] - \frac{1}{2} \right|$$

Man beachte, dass π, ρ^* und com^* bei $b = 0$ die gleiche Wahrscheinlichkeitsverteilung haben, als wenn sie durch Start bzw. Commit gebildet worden wären; bei $b = 1$ ist ρ^* zufällig und gleichverteilt aus $\mathbb{G}_T$. Durch Umsortieren und Umbenennen lässt sich obiges in das Zufallsexperiments des Angriffs auf die Vertraulichkeit umformen.

$$\begin{aligned} & = \left| Pr \left[\begin{array}{l} \pi := \mathsf{Start}(1^k); \\ (\rho_0^*, com^*, dec^*) := \mathsf{Commit}(1^k, \pi); \\ \rho_1^* :=_R \mathbb{G}_T; b :=_R \{0,1\}; \\ b' := \mathcal{A}(\pi, com^*, \rho_b^*) \end{array} : b = b' \right] - \frac{1}{2} \right| \\ & = Adv^{\text{Vertraulichkeit}}_{\mathcal{A},\mathcal{ENC}}(1^k) = \epsilon(1^k) \end{aligned}$$

Insgesamt ist $\mathcal{B}$ also ein (t, ϵ)-BDDHP-Angreifer von Gen.

Alles zusammen: Nun kann man wie im Abschnitt zuvor mit BB-$\mathcal{IBE}$ und diesem alternativen BDDHP-$\mathcal{ENC}$ ein PKE zusammensetzen. Allerdings bekommt man folgendes unerwünschtes Resultat:
BB-$\mathcal{IBE}$ und BDDHP-$\mathcal{ENC}$ sind zwar bei der Erzeugung des öffentlichen Schlüssels bzw. der öffentlichen Parameter nahezu identisch, hält man sich jedoch exakt an die Konstruktionsweise aus Kapitel 3.2, so werden die Algorithmen Setup und Start jeweils unabhängig voneinander ausgeführt. Es werden also für BB-$\mathcal{IBE}$ eine andere bilineare Gruppe und andere Gruppenelemente erzeugt als für BDDHP-$\mathcal{ENC}$.

Das Verfahren my-1_{st}-$\mathcal{PKE}$ mit den drei Algorithmen Keygen, Encrypt und Decrypt funktioniert wie folgt (dabei sind Gen, G und map definiert wie bei BK-$\mathcal{PKE}$):

Keygen(1^k) Wie in der Definition von BB-$\mathcal{IBE}$ werden $(p, \mathbb{G}, \mathbb{G}_T, e) := \mathsf{Gen}(1^k)$ und ein erzeugendes Element $g :=_R \mathbb{G}^*$ sowie ein Exponent $M :=_R \mathbb{Z}_p$ gewählt. Dann wird $g_1 := g^M$ gesetzt, $g_2, g_3 :=_R \mathbb{G}$ gewählt und $Z := e(g_1, g_2) \in \mathbb{G}_T$ gesetzt.
Wie in der Definition von BDDHP-$\mathcal{ENC}$ werden $(\hat{p}, \hat{\mathbb{G}}, \hat{\mathbb{G}}_T, \hat{e}) := \mathsf{Gen}(1^k)$ und ein erzeugendes Element $\hat{g} :=_R \hat{\mathbb{G}}^*$ sowie ein Exponent $\hat{M} :=_R \mathbb{Z}_{\hat{p}}$ gewählt. Dann wird $\hat{g}_1 := \hat{g}^{\hat{M}}$ gesetzt, $\hat{g}_2 :=_R \hat{\mathbb{G}}$ gewählt und $\hat{Z} := \hat{e}(\hat{g}_1, \hat{g}_2) \in \hat{\mathbb{G}}_T$ gesetzt.
Der öffentliche Schlüssel ist $E = (p, \mathbb{G}, \mathbb{G}_T, e, g, g_1, g_2, g_3, Z, \hat{p}, \hat{\mathbb{G}}, \hat{\mathbb{G}}_T, \hat{e}, \hat{g}, \hat{g}_1, \hat{g}_2, \hat{Z})$ und der private Schlüssel ist der Exponent M.

Encrypt(m, E) Wie in der Definition von BDDHP-$\mathcal{ENC}$ wird ein zufälliges $\hat{r} :=_R \mathbb{Z}_{\hat{p}}$ gewählt und $(\rho, com, dec) := (\hat{Z}^{\hat{r}}, \hat{g}^{\hat{r}}, \hat{r})$ gesetzt.
Anschließend wird die Nachricht m mit dec konkateniert und mit der Identität $ID := map(com)$ verschlüsselt: dazu wird zunächst ein zufälliges $r :=_R \mathbb{Z}_p$ gewählt und dann $\overline{c} := (m.dec \oplus G(Z^r), g^r, (g_3 \cdot g_1^{ID})^r)$ gesetzt. Dieses wird nun mit Hilfe des MAC und dem Schlüssel ρ unterschrieben: $\sigma := \mathsf{Sign}(\overline{c}, \rho)$.
Der Chiffretext ist $(com, \overline{c}, \sigma) = (\hat{g}^{\hat{r}}, (m.\hat{r} \oplus G(Z^r), g^r, (g_3 \cdot g_1^{map(\hat{g}^{\hat{r}})})^r), \sigma)$. Sei $\mathcal{S}$ der Unterschriftenraum von MAC, dann ist der Chiffretext ein Element aus $\underbrace{\hat{\mathbb{G}}}_{com} \times (\underbrace{\{0,1\}^{k+|\hat{p}|} \times \mathbb{G} \times \mathbb{G}}_{\overline{c}}) \times \underbrace{\mathcal{S}}_{\sigma}$.

Decrypt(c, D) Der Chiffretext wird als $c = (com, \overline{c}, \sigma)$ eingelesen. Zunächst muss der zur „Identität“ com gehörende private Schlüssel bestimmt werden: Es wird ein Exponent $k :=_R \mathbb{Z}_p$ gewählt und $D_{com} := (g_2^M \cdot (g_3 \cdot g_1^{map(com)})^k, g^k)$ gesetzt.
Anschließend wird die „Nachricht“ $m.dec$ wiederhergestellt: Sei dazu der Chiffretext $\overline{c} = (c_0, c_1, c_2)$ und der private Schlüssel $D_{com} = (D_0, D_1)$. Entschlüsselt erhält man dann $m.dec := c_0 \oplus G(\frac{e(c_1, D_0)}{e(D_1, c_2)})$.
Nun kann der Unterschriftenschlüssel für MAC bestimmt werden: $\rho := \hat{Z}^{dec}$.
Zuletzt wird die Gültigkeit des Chiffretextes und des Unterschriftenschlüssels überprüft. Dazu wird getestet, ob $com = \hat{g}^{dec}$ und ob $\mathsf{Verify}(\overline{c}, \sigma, \rho) = 1$ ist. Wenn dies der Fall ist, wird m ausgegeben; ansonsten $\perp$.

Satz 4.11

My-1_{st}-$\mathcal{PKE}$ ist korrekt und IND-CCA2-sicher, wenn das BDDHP in den von Gen erzeugten Gruppen hart ist und ein stark-fälschungssicherer MAC existiert.

Beweis

Dies gilt genau wie bei Satz 4.7, da exakt nach der Konstruktion aus Kapitel 3.2 vorgegangen wurde.

Eine verbesserte Variante: Eine naheliegende Idee ist, dass man für BB-$\mathcal{IBE}$ und BDDHP-$\mathcal{ENC}$ die selbe bilineare Gruppe und die selben Gruppenelemente verwendet, also $(\hat{p}, \hat{\mathbb{G}}, \hat{\mathbb{G}}_T, \hat{e}, \hat{g}, \hat{g}_1, \hat{g}_2, \hat{Z}) := (p, \mathbb{G}, \mathbb{G}_T, e, g, g_1, g_2, Z)$ setzt. Wichtig: bei der Verschlüsselung werden weiterhin zwei Exponenten r und $\hat{r}$ gewählt.

Dies bietet einige Vorteile: die Länge des öffentlichen Schlüssels wird nahezu halbiert. Bei der Verschlüsselung werden Exponentiationen mit festen Basen durchgeführt, wobei für gewöhnlich Vorabberechnungen gemacht werden, um die eigentliche Exponentiation beschleunigen (siehe [MOV97, §14.6.3]). Da hier nur noch zwei (g und Z) statt vier ($g, \hat{g}, Z$ und $\hat{Z}$) feste Basen verwendet werden, müssen auch weniger Vorabberechnungen durchgeführt und gespeichert werden.

Es gibt allerdings auch einen kleinen Nachteil: Das resultierende my-2_{nd}-$\mathcal{PKE}$ beruht nicht mehr ausschließlich auf der Konstruktion aus Kapitel 3.2. Damit darf man auch den Korrektheits- und den Sicherheitsbeweis nicht direkt übernehmen. Statt dessen muss man einen eigenständigen Beweis führen, welcher sich allerdings in weiten Teilen analog zum allgemeinen Beweis führen lässt.

Satz 4.12

My-2_{nd}-$\mathcal{PKE}$ ist korrekt.

Beweis

Der Fall $(\hat{p}, \hat{\mathbb{G}}, \hat{\mathbb{G}}_T, \hat{e}, \hat{g}, \hat{g}_1, \hat{g}_2, \hat{Z}) = (p, \mathbb{G}, \mathbb{G}_T, e, g, g_1, g_2, Z)$ kann (wenn auch unwahrscheinlich) in my-1_{st}-$\mathcal{PKE}$ vorkommen. Da die Korrektheit von my-1_{st}-$\mathcal{PKE}$ die Gültigkeit der Korrektheitsgleichung $\mathsf{Decrypt}(\mathsf{Encrypt}(m, E), D) = m$ aber für alle möglichen Parameterkombinationen fordert, muss sie insbesondere auch für diesen Spezialfall gelten. Damit folgt die Korrektheit von my-2_{nd}-$\mathcal{PKE}$ direkt aus der von my-1_{st}-$\mathcal{PKE}$.

Satz 4.13

Sei $\mathcal{A}$ ein (t, q, ϵ)-IND-CCA2-Angreifer von my-2_{nd}-$\mathcal{PKE}$, dann lassen sich mit $\mathcal{A}$ folgende Algorithmen konstruieren:

$\mathcal{B}_1$ ein (t_1, ϵ_1)-BDDHP-Angreifer der von Gen erzeugten Gruppen

$\mathcal{B}_2$ ein (t_2, ϵ_2)-BDDHP-Angreifer der von Gen erzeugten Gruppen

$\mathcal{B}_3$ ein (t_3, ϵ_3)-BDDHP-Angreifer der von Gen erzeugten Gruppen

$\mathcal{B}_4$ ein (t_4, ϵ_4)-BDDHP-Angreifer der von Gen erzeugten Gruppen

$\mathcal{B}_5$ ein $(t_5, 1, \epsilon_5)$-Angreifer der Fälschungssicherheit von $\mathcal{MAC}$

wobei die t_i polynomiell beschränkt sind und $\epsilon \leq 2 \cdot \epsilon_1 + 4 \cdot \epsilon_2 + 8 \cdot \epsilon_3 + 8 \cdot \epsilon_4 + 2q \cdot \epsilon_5$.

Beweis (Skizze)

Der Beweis lässt sich ähnlich führen wie der von Satz 3.2. Die Beweisstruktur und -ideen können übernommen werden. Da allerdings $\mathcal{IBE}$ und $\mathcal{ENC}$ miteinander kombiniert wurden und nicht mehr isoliert angegriffen werden können, müssen alle Lemmata abgewandelt werden, welche $\mathcal{IBE}$ und $\mathcal{ENC}$ mit einbeziehen. An diesen Stellen wird statt den Sicherheitseigenschaften von $\mathcal{IBE}$ bzw. $\mathcal{ENC}$ immer direkt das BDDHP in den von Gen erzeugten Gruppen angegriffen. Diese Änderungen sollen an dieser Stelle nur kurz skizziert und am Beispiel von Lemma 3.5 verdeutlicht werden:

Die Zufallsexperimente und -ereignisse bleiben genau so bestehen. Lemma 3.4 kann exakt so übernommen werden. Lemma 3.5 muss wie folgt abgewandelt werden:

Lemma 3.5 (modifiziert)

Sei $\mathcal{A}$ ein (t, q, ϵ)-IND-CCA2-Angreifer von my-2_{nd}-$\mathcal{PKE}$, dann gibt es einen Algorithmus $\mathcal{B}_1$, der ein (t_1, q, ϵ_1)-BDDHP-Angreifer der von Gen erzeugten Gruppen ist. Dabei ist t_1 polynomiell beschränkt und

$$\epsilon_1(1^k) := Adv^{BDDHP}_{\mathcal{B}_1,\mathsf{Gen}}(1^k) = \frac{\left| Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{WIN}\right] - \frac{1}{2}\right|}{2}$$

Beweis (Skizze)

$\mathcal{B}_1$ simuliert für $\mathcal{A}$ den Herausforderer in Experiment 1 und versucht selbst dadurch das BDDHP in einer von Gen erzeugten Gruppe zu brechen. Als Eingabe bekommt $\mathcal{B}_1$ eine Gruppenbeschreibung $(p, \mathbb{G}, \mathbb{G}_T, e)$, die Gruppenelemente $g, x = g^\alpha, y = g^\beta, z = g^\gamma \in \mathbb{G}$ und ein $T_{\bar{b}} \in \mathbb{G}_T$. $\mathcal{B}_1$ muss entscheiden, ob entweder $T_{\bar{b}} = e(g, g)^{\alpha\beta\gamma}$ ist oder zufällig aus $\mathbb{G}_T$ gewählt wurde.

Für die Simulation verwendet $\mathcal{B}_1$ sowohl Ideen aus der Simulation zu Lemma 3.5 als auch aus der Simulation zum Sicherheitsbeweis von BB-$\mathcal{IBE}$. Die Interaktion von $\mathcal{B}_1$ und $\mathcal{A}$ verläuft analog zu Lemma 3.5. Es ist allerdings der Fall, dass $\mathcal{B}_1$ weder der Masterkey bekannt ist noch ein KeygenOracle zur Verfügung steht. Daher wird die Technik aus Satz 4.2 angewendet, mit der $\mathcal{B}_1$ zu fast allen Identitäten den privaten Schlüssel bestimmen kann. $\mathcal{B}_1$ ist wie folgt definiert (siehe Definition 15 auf Seite 24):

$$\mathcal{B}_1(p,\mathbb{G},\mathbb{G}_T,e,g,x,y,z,T_{\overline{b}}) := \left\{\begin{array}{l} g \text{ wird übernommen; } g_1 := x; g_2 := y; Z := e(g_1,g_2); \\ \hat{r} :=_R \mathbb{Z}_p; (\rho^*, com^*, dec^*) = (Z^{\hat{r}}, g^{\hat{r}}, \hat{r}); \\ ID^* := map(com^*); t :=_R \mathbb{Z}_p; g_3 := g^t \cdot g_1^{-ID^*}); \\ E := (p,\mathbb{G},\mathbb{G}_T,e,g,g_1,g_2,g_3,Z); \\ (m_0,m_1,s) := \mathcal{A}_{\mathrm{I}}^{SimDecryptOracle_1}(E); \\ b :=_R \{0,1\}^*; r :=_R \mathbb{Z}_p; \overline{c}^* := (m.dec^* \oplus G(T_{\overline{b}}), z, z^t); \\ \sigma^* := \mathsf{Sign}(\overline{c}^*, \rho^*).\ c^* := (com^*, \overline{c}^*, \sigma^*) \\ b' := \mathcal{A}_{\mathrm{II}}^{SimDecryptOracle_1}(c^*, s) \\ \text{Falls } b = b' \text{: } \overline{b'} := 0, \text{ sonst } \overline{b'} := 1 \\ \text{Ausgabe: } \overline{b'} \end{array}\right\}$$

Dabei arbeitet das $SimDecryptOracle_1$ genau so wie in Lemma 3.5, nur dass der Aufruf von KeygenOracle(com_i) ersetzt wird durch:

$$k' :=_R \mathbb{Z}_p; D_{com_i} := \left(g_2^{\frac{-t}{com_i - ID^*}} \cdot (g_3 \cdot g_1^{com_i})^{k'}, g_2^{\frac{-1}{com_i - ID^*}} \cdot g^{k'} \right)$$

Kombiniert man die Überlegungen aus Lemma 3.5 und Satz 4.2 zum Vorteil von $\mathcal{B}_1$, so erhält man:

$$\epsilon_1(1^k) = \frac{\left| Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{WIN}\right] - \frac{1}{2}\right|}{2}$$

Dabei stammt das $\left| Pr\left[E_1^{\mathcal{A}}(1^k) : \mathsf{WIN}\right] - \frac{1}{2}\right|$ aus den Überlegungen von Lemma 3.5 und der Faktor $\frac{1}{2}$ aus den Überlegungen zu Lemma 4.2.

Beweisskizze von Satz 4.13 (Fortsetzung)

Die Beobachtung 3.6 kann übernommen werden. Die Lemmata 3.7, 3.8 und 3.9 müssen ähnlich wie Lemma 3.5 angepasst werden. Die Idee ist immer die selbe: es wird die Simulation aus dem Beweis des jeweiligen Lemmas mit den Techniken zum Berechnen der privaten Schlüssel ohne Masterkey aus Lemma 4.2 kombiniert. Das letzte Lemma 3.10 und das Zusammenfügen zum Gesamtbeweis kann wieder übernommen werden.

kurzer Vergleich: Beide Umsetzungen unterscheiden sich nur in den vom ENC bestimmten Operationen. Diese sind bei Hash-$\mathcal{ENC}$ (Auswertungen von Hashfunktionen) effizienter als bei BDDHP-$\mathcal{ENC}$ (Exponentiationen). My-2_{nd}-$\mathcal{PKE}$ hat jedoch durch die gemeinsame Parameterwahl von BB-$\mathcal{IBE}$ und BDDHP-$\mathcal{ENC}$ einen kürzeren öffentlichen Schlüssel als BK-$\mathcal{PKE}$. Dieser enthält zusätzlich die Hashfunktionen h und H_s.

5 Fazit

Diese Studienarbeit: In dieser Arbeit werden im Wesentlichen die zentralen Ideen zweier wissenschaftlicher Arbeiten aus dem Bereich „identitätsbasierte Kryptographie“ sowie eigene Überlegungen dazu vorgestellt:
Dies ist zum einen das Verfahren aus [BCHK06], um aus beliebigen IND-sID-CPA-sicheren IBE ein IND-CCA2-sicheres PKE zu erhalten, welches in seiner allgemeinen Form in Kapitel 3 dargelegt und dann in Abschnitt 4.2 konkret angewendet wird. Zum anderen wird in Kapitel 4.1 ein effizientes, IND-sID-CPA-sicheres IBE aus [BB04] erläutert. Beim Anfertigen der gesamten Arbeit ist besonderer Wert darauf gelegt worden, Grundlagen detailliert zu erklären und keine „trivialen“ Ideen oder Beweisschritte wegzulassen.

Die Ergebnisse der eigenen Überlegungen sind in Abschnitt 4.3 dargelegt. Es wird mit BDDHP-$\mathcal{ENC}$ eines der drei verwendeten kryptographischen Primitive auf eine andere Art und Weise implementiert als von Boneh und Katz vorgeschlagen. Dabei ist zwar das Ziel erreicht worden, dieses ENC so zu konstruieren, dass man es gut mit BB-$\mathcal{IBE}$ kombinieren kann, leider musste jedoch die Hoffnung, hieraus ein effizienteres PKE zu erhalten, enttäuscht werden.
Als Vorteil dieser eigenen Ergebnisse lässt sich aber anführen, dass BDDHP-$\mathcal{ENC}$ mit den selben Grundoperationen und Sicherheitsannahmen wie BB-$\mathcal{IBE}$ auskommt – so ist diese Konstruktion sicherlich leichter nachzuvollziehen als das von Boneh und Katz vorgeschlagene Hash-$\mathcal{ENC}$, welches zusätzliche Primitive und Sätze für den Sicherheitsbeweis verwendet.

Die hier vorgestellten Verfahren: Das BB-$\mathcal{IBE}$ ist nach Wissen des Autors das erste effiziente IND-sID-CPA-sichere identitätsbasierte Kryptosystem, dessen Sicherheitsbeweis ohne die zusätzliche Heuristik der „zufälligen Orakel“ (siehe [BR93]) auskommt. Das Wegfallen dieser zufälligen Orakel ist ein wesentlicher Fortschritt, da ein Sicherheitsbeweis mit zufälligen Orakeln nicht notwendigerweise auch Sicherheit in der Realität impliziert.

Die Konstruktion von Boneh und Katz, mit der man aus beliebigem IND-sID-CPA-sicheren IBE ein IND-CCA2-sicheres PKE erhält, ist ein sehr wichtiges Ergebnis für die Public-Key-Kryptographie. Zunächst mag es verwunderlich erscheinen, weshalb man das in der Einleitung als „vereinfachend“ beschriebene Paradigma der identitätsbasierten Kryptographie aufgibt, um ein Public-Key-Verfahren zu erhalten. Die Tatsache, dass allerdings im Wesentlichen nur zwei Verfahren (nämlich [NY90, CS04] und ergänzende Arbeiten) bekannt sind, IND-CCA2-sicheres PKE zu erzeugen, löst diese Verwunderung schnell auf. Durch die Ergebnisse aus [BCHK06] ist nun eine dritte neuartige Möglichkeit bekannt.

Weiterhin ist die zugrundeliegende Idee vielseitig anwendbar. Sie lässt sich z.B. dazu verwenden, um IND-sID-CCA2-sicheres IBE (statt PKE) zu erhalten (siehe [BCHK06, §6]). In [BMW05] wird diese Idee ebenfalls angewendet, um direkt ein IND-CCA2-sicheres PKE zu erzeugen (also ohne „Umweg“ über ein identitätsbasiertes Verfahren).

Literatur

[ACG+06] Attrapadung, Nuttapong ; Cui, Yang ; Galindo, David ; Hanaoka, Goichiro ; Hasuo, Ichiro ; Imai, Hideki ; Matsuura, Kanta ; Yang, Peng ; Zhang, Rui: Relations among Notions of Security for Identity Based Encryption Scheme. In: *Latin American Theoretical Informatics Symposium (LATIN '06)* Bd. 3887, Springer-Verlag, 2006 (Lecture Notes in Computer Science), S. 130–141

[BB04] Boneh, Dan ; Boyen, Xavier: Efficient Selective-ID Secure Identity Based Encryption Without Random Oracles. In: *Advances in Cryptology–EUROCRYPT 2004* Bd. 3027, Springer-Verlag, 2004 (Lecture Notes in Computer Science), S. 223–238

[BCHK06] Boneh, Dan ; Canetti, Ran ; Halevi, Shai ; Katz, Jonathan: *Chosen-Ciphertext Security from Identity-Based Encryption.* 2006. – erscheint in: SIAM Journal on Computing

[BDPR98] Bellare, Mihir ; Desai, Adesai ; Pointcheval, David ; Rogaway, Phillip: Relations Among Notions of Security for Public-Key Encryption Schemes. In: *Advances in Cryptology–CRYPTO '98* Bd. 1462, Springer-Verlag, 1998 (Lecture Notes in Computer Science), S. 26–45

[BF01] Boneh, Dan ; Franklin, Matthew: Identity-Based Encryption from the Weil Pairing. In: *Advances in Cryptology–CRYPTO 2001* Bd. 2139, Springer-Verlag, 2001 (Lecture Notes in Computer Science), S. 213–229

[BK05] Boneh, Dan ; Katz, Jonathan: Improved Efficiency for CCA-Secure Cryptosystems Built Using Identity-Based Encryption. In: *Topics in Cryptology–CT-RSA 2005* Bd. 3376, Springer-Verlag, 2005 (Lecture Notes in Computer Science), S. 87–103

[BMW05] Boyen, Xavier ; Mei, Qixiang ; Waters, Brent: Direct Chosen Ciphertext Security from Identity-Based Techniques. In: *Proceedings of the 12th ACM conference on Computer and communications security*, ACM Press, 2005, S. 320–329

[BR93] Bellare, Mihir ; Rogaway, Phillip: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: *Proceedings of the First Annual Conference on Computer and Communications Security*, ACM Press New York, 1993, S. 62–73

[CHK04] Canetti, Ran ; Halevi, Shai ; Katz, Jonathan: Chosen-Ciphertext Security from Identity-Based Encryption. In: *Advances in Cryptology–EUROCRYPT 2004* Bd. 3027, Springer-Verlag, 2004 (Lecture Notes in Computer Science), S. 207–222

[Coc01] Cocks, Clifford: An Identity Based Encryption Scheme based on Quadratic Residues. In: *Cryptography and Coding: 8th IMA International Conference* Bd. 2260, Springer-Verlag, 2001 (Lecture Notes in Computer Science), S. 360–363

[CS04] Cramer, Ronald ; Shoup, Victor: Design and Analysis of Practical Public-Key Encryption Schemes Secure Against Adaptive Chosen Ciphertext Attack. In: *SIAM Journal on Computing* 33 (2004), Nr. 1, S. 167–226

[MOV97] Menezes, Alfred J. ; Oorschot, Paul C. van ; Vanstone, Scott A.: *Handbook of Applied Cryptography*. CRC Press, 1997

[NY89] Naor, Moni ; Yung, Moti: Universal One-Way Hash Functions and their Cryptographic Applications. In: *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, ACM Press New York, 1989, S. 33–43

[NY90] Naor, Moni ; Yung, Moti: Public-Key Cryptosystems Provably Secure Against Chosen Ciphertext Attacks. In: *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, ACM Press New York, 1990, S. 427–437

[Sha85] Shamir, Adi: Identity-Based Cryptosystems and Signature Schemes. In: *Advances in Cryptology: Proceedings of CRYPTO '84* Bd. 196, Springer-Verlag, 1985 (Lecture Notes in Computer Science), S. 47–53

[Wat05] Waters, Brent: Efficient Identity-Based Encryption Without Random Oracles. In: *Advances in Cryptology–EUROCRYPT 2005* Bd. 3494, Springer-Verlag, 2005 (Lecture Notes in Computer Science), S. 114–127

Erklärung

Hiermit versichere ich, dass diese Arbeit von mir selbstständig angefertigt wurde. Es wurden keine anderen als die angegebenen und bei Zitaten kenntlich gemachten Quellen und Hilfsmittel benutzt.

Jonas Schrieb
Paderborn, den 15. August 2006