cyberagentur

# 2023 ENCRYPTED COMPUTING COMPASS

Dr. Nico Döttling, CISPA, Saarbrücken
Prof. Dr. Jörn Müller-Quade, KASTEL, KIT, Karlsruhe
Thomas Agrikola, KIT, Karlsruhe
Laurin Benz, KIT, Karlsruhe
Anne Müller, CISPA, Saarbrücken

Version 1.0 | 18.11.22
Herausgeberin: Agentur für Innovation in der Cybersicherheit GmbH

## Disclaimer

Die hier geäußerten Ansichten und Meinungen sind ausschließlich diejenigen der Autorinnen und Autoren und entsprechen nicht notwendigerweise denjenigen der Agentur für Innovation in der Cybersicherheit GmbH oder der Bundesregierung.

Diese Studie wurde durch die Agentur für Innovation in der Cybersicherheit GmbH beauftragt und finanziert. Eine Einflussnahme der Agentur für Innovation in der Cybersicherheit GmbH auf die Ergebnisse fand nicht statt.

## Impressum

Herausgeberin: Agentur für Innovation in der Cybersicherheit GmbH
Große Steinstraße 19, 06108 Halle (Saale), Germany
E-Mail: kontakt@cyberagentur.de
Internet: www.cyberagentur.de
Twitter: https://twitter.com/CybAgBund

Die Nutzungsrechte liegen bei der Herausgeberin.

Lizenz: CC BY-NC-ND 4.0: https://creativecommons.org/licenses/by-nc-nd/4.0/
Erscheinungsdatum: 10.05.2023
Redaktion: Abteilung Schlüsseltechnologien, Referat Kryptologie Publikation als PDF auf: www.cyberagentur.de/programme/ec2/

# Encrypted Computing Compass

Dr. Nico Döttling, CISPA, Saarbrücken
Prof. Dr. Jörn Müller-Quade, KASTEL, KIT, Karlsruhe
Thomas Agrikola, KIT, Karlsruhe
Laurin Benz, KIT, Karlsruhe
Anne Müller, CISPA, Saarbrücken

Version 1.0 — 18.11.22

© Agentur für Innovation in der Cybersicherheit GmbH

# Inhaltsverzeichnis

# 1 Einführung

Daten bilden die Basis wichtiger ökonomischer oder gesellschaftlicher Entscheidungen und des wissenschaftlichen Fortschritts. Viele Daten, wie Firmengeheimnisse oder persönliche Daten, sollten aber geschützt sein. Daher wäre es wünschenswert, auf geheimen Daten rechnen zu können und Ergebnisse zu erhalten, ohne dass dafür Geheimnisse preisgegeben werden.

Techniken der modernen Kryptographie erlauben ein Rechnen auf Geheimnissen und das vorliegende Dokument, der *Encrypted Computing Compass*, soll eine Einordnung dieser Techniken bieten und die Praxistauglichkeit der Lösungen beurteilen.

Die moderne Kryptographie bietet, grob betrachtet, drei Ansätze um auf Geheimnissen zu rechnen:

1. Die *vollhomomorphe Verschlüsselung* (Fully Homomorphic Encryption, kurz FHE) ist ein Public Key Verschlüsselungsverfahren, mit dem Zahlen so verschlüsselt werden können, dass es möglich ist mit diesen verschlüsselten Zahlen zu rechnen, ohne die Zahlen selbst zu kennen. Die Ergebnisse der Berechnung bleiben verschlüsselte Zahlen und nur mit dem geheimen Schlüssel kann das Ergebnis entschlüsselt werden. FHE-Verfahren bilden den Schwerpunkt dieser Studie, da es gerade bei diese Verfahren in jüngster Zeit enorme Fortschritte gab und die Möglichkeiten und Grenzen dieser Verfahren nicht allgemein bekannt sind.

2. *Sichere Mehrparteienberechnungen (Secure Multiparty Computations, kurz MPC)*, dies sind kryptographische Verfahren, bei denen mehrere Teilnehmer jeweils geheime Eingaben haben und gemeinsam auf diesen Eingaben rechnen wollen, ohne dass mehr bekannt wird als das Ergebnis der Berechnung. In der Theorie ist seit den 1980ern bekannt, dass es solche Verfahren für beliebige Berechnungen gibt, aber bei der Effizienz der Verfahren gab es inzwischen deutliche Fortschritte. Sichere Mehrparteienberechnungen werden in diesem Dokument als eine Alternative zu FHE-Verfahren betrachtet. Je nach Anwendung können sichere Mehrparteienberechnungen besser geeignet sein als FHE-Verfahren, insbesondere, wenn der Kommunikationsaufwand groß sein darf.

3. *Sichere Enklaven* beziehungsweise *Trusted Execution Environments (TEEs)*, dies sind Hardwarebausteine, die eine sichere Berechnung so abkapseln, dass selbst der Betreiber der Hardware nicht an die geheimen Eingaben herankommt, oder die Ausgabe manipulieren kann. Enklaven und TEEs werden in diesem Dokument nur ganz am Rande betrachtet, weil sie, anders als FHE oder MPC, ein großes Vertrauen in den Hersteller der Enklave voraussetzen. Dennoch könnten Enklaven und TEEs, bei niedrigeren Anforderungen an die Sicherheit eine interessante Alternative sein, weil sie ohne Zusatzaufwand direkt auf den Geheimnissen rechen und somit eine durch andere Verfahren unerreichte Effizienz haben. Eine Kombination verschiedener Verfahren unterschiedlicher Effizienz und Sicherheit könnte ein vielversprechender Ansatz für die Zukunft sein.

Ziel dieses Dokuments ist das kompakte und verständliche Darlegen der wissenschaftlichen Grundlagen, auf denen FHE- und MPC-Verfahren beruhen, sowie eine Vorstudie der praktischen Machbarkeit für relevante Use-Cases.

Das Dokument ist in folgende Abschnitte gegliedert:

1. Gitterbasierte Kryptographie (Kapitel 2), in diesem Kapitel werden die mathematischen Grundlagen der gitterbasierten Verschlüsselungs-Verfahren motiviert und eingeführt, insbesondere werden die zugrundeliegenden Sicherheitsannahmen betrachtet, etwa, dass gitterbasierte Verfahren als sicher sogar gegen Quantencomputer eingeschätzt werden.

2. Gitterbasierte Verschlüsselungs-Verfahren (Kapitel 3), hier werden zwei konkrete Verschlüsselungs-Verfahren angegeben. Es sind Public-Key Verfahren, also Verfahren, bei denen jeder verschlüsseln, aber nur der Besitzer eines geheimen Schlüssels entschlüsseln kann. Diese gitterbasierten Verfahren erlauben schon eine Addition von verschlüsselten Geheimnissen und bilden die Grundlage der vollhomomorphen Verfahren.

3. Somewhat Homomorphic Encryption (Kapitel 4), dieses Kapitel erklärt die Grundlagen der vollhomomorphen Verschlüsselungs-Verfahren von den ersten Ansätzen bis zu den zur Zeit aktuellsten Verfahren der sogenannten vierten Generation.

4. Fully-Homomorphic Encryption (Kapitel 5), hier werden die verschiedenen vollhomomorphen Verfahren kurz vorgestellt, sowie die relevanten Forschungsgruppen und verfügbaren Bibliotheken. Dieses Kapitel spiegelt also den aktuellen Stand der Forschung und gibt die entscheidenden Referenzen für zukünftige Studien zur vollhomomorphen Verschlüsselung.

5. Fortgeschrittene Verfahren (Kapitel 6), hier werden Varianten von FHE-Verfahren vorgestellt, die zusätzliche Eigenschaften haben oder Vorteile bieten. Insbesondere Verfahren für numerische, d.h. näherungsweise Berechnungen, Verfahren, die mehr Teilnehmer erlauben, Verfahren, die Quanten-Berechnungen unterstützen oder das Obfuszieren von Schaltkreisen.

6. Sichere Mehrparteienberechnungen (MPC) (Kapitel 7), in diesem Kapitel werden Alternativen zu FHE-Verfahren vorgestellt, die für einige Anwendungen eine effizientere Berechnung auf geheimen Daten erlauben, die aber einen signifikant höheren Kommunikationsaufwand haben.

7. Sichere Hardware (Kapitel 8) haben wir in diesem Dokument nur kurz betrachtet, weil großes Vertrauen in die Hardware und den Hersteller nötig sind, solange man die sichere Hardware nicht mit anderen Ansätzen kombiniert. Betrachtet werden Trusted Platform Modules (TPMs), die sicherstellen, dass nur bestimmte, zertifizierte Programme auf dem Computer laufen und sichere Enklaven, die Berechnungen wegkapseln können.

8. Anwendungen und Use-Cases (Kapitel 9), dieses Kapitel ist ein Herzstück der Vorstudie, da konkrete relevante Anwendungsbeispiele betrachtet werden sowie ihre Machbarkeit auf verschlüsselten Daten. Als Zusammenfassung der Use-Case Studie ergibt sich, dass bei Anwendungen, bei denen es weder auf die Kommunikationskomplexität noch auf die Lastbalancierung ankommt, die FHE-basierte Ansätze derzeit nicht mit alternativen Ansätzen konkurrenzfähig sind.

9. Benchmarks (Kapitel 10), in diesem Abschnitt wird ein FHE-Estimator vorgestellt, der als wesentlicher Teil dieser Vorstudie entwickelt und implementiert wurde. Dieser FHE-Estimator ist über eine Webseite erreichbar und kann basierend auf einem gegebenen Programm, den Aufwand abschätzen, der nötig wäre, um dieses Programm als Schaltkreis mit einem FHE-Verfahren durchzurechnen.

10. Entscheidungshilfen (Kapitel 11), dieser Abschnitt bietet, aufbauend auf den vorangegangenen Kapiteln, eine systematische Einordnung der verschiedenen verfügbaren Ansätze für das Rechnen auf Geheimnissen. Anhand von Fragen über eine geplante Anwendung wird Schritt für Schritt entschieden welcher Ansatz für die Anwendung am besten geeignet ist.

11. Ontologie (Kapitel 12), dieser Abschnitt klärt die notwendigen Begriffe und erlaubt einen direkten Einstieg in relevante Literatur.

# 2 Gitterbasierte Kryptographie

## 2.1 Public-Key Verschlüsselung

Public-Key Encryption (PKE) ermöglicht es einem Sender eine Nachricht für einen Empfänger zu verschlüsseln, ohne dazu einen Schlüssel ausgetauscht zu haben. Es ist lediglich der öffentliche Schlüssel des Empfängers notwendig, welcher von jedem abgerufen werden kann. Während jede Partei, die Zugriff auf einen öffentlichen Schlüssel hat, verschlüsseln kann, ist es nur dem Besitzer des zu dem öffentlichen Schlüssel zugehörigen geheimen Schlüssel möglich, diese Chiffrate zu entschlüsseln.

Formaler besteht ein PKE-Verfahren aus drei Algorithmen: Einem Setup-Algorithmus Gen, der Paare von öffentlichen und geheimen Schlüsseln $(\mathsf{pk}, \mathsf{sk})$ generiert, einem Verschlüsselungsalgorithmus Enc, der bei Eingabe von $\mathsf{pk}$ und einer Nachricht $m$ ein Chiffrat $c$ ausgibt und einem deterministischen Entschlüsselungsalgorithmus Dec, der bei Eingabe eines geheimen Schlüssels $\mathsf{sk}$ und eines Chiffrats $c$ entweder eine Nachricht $m$ oder ein Fehlersymbol $\perp$ ausgibt.

Ein PKE-Verfahren muss zwei Eigenschaften erfüllen. *Korrektheit* stellt sicher, dass die Entschlüsselung (fast) immer die ursprünglich verschlüsselte Nachricht wiederherstellt. *Sicherheit* garantiert, dass es nicht machbar ist auch nur ein einziges Bit an Information über die verschlüsselte Nachricht zu extrahieren. Im Fall von PKE reicht die Kenntnis des öffentlichen Schlüssels aus, um beliebige Nachrichten zu verschlüsseln. Daher muss eine sinnvolle Definition der PKE-Sicherheit immer dem Chosen-Plaintext-Attack (CPA)-Bedrohungsmodell standhalten. Der de-facto Standard für PKE-Sicherheit betrachtet allerdings ein wesentlich stärkeres Bedrohungsmodell. Wird beispielsweise ein PKE-Verfahren in einem Protokoll verwendet, das es ermöglicht den Inhalt mancher Chiffrate zu entschlüsseln, reicht CPA-Sicherheit nicht aus. Daher wird in solchen Fällen das Bedrohungsmodell Chosen-Ciphertext-Attack (CCA) betrachtet, das es Angreifern ermöglicht, zusätzlich beliebige Nachrichten (ausgenommen das Chiffrat, das angegriffen wird) zu entschlüsseln.

## 2.2 Über Annahmen für Public-Key Verfahren

Alle derzeit existierenden Public-Key Verschlüsselungsverfahren beruhen auf unbewiesenen komplexitätstheoretischen Annahmen. Diese Situation wird sich auf absehbare Zeit nicht ändern, denn die beweisbare Sicherheit eines Public-Key Verfahrens ohne Annahmen würde direkt eine Lösung für das größte ungelöste Problem der theoretischen Informatik implizieren, dem P-vs-NP Problem. Daher sind, abgesehen von einigen wenigen informations-theoretisch sicheren kryptographischen Verfahren alle derzeitigen Verfahren nur relativ zu Annahmen sicher.

Während sich in der Symmetric-Key Kryptographie fast alle Verfahren (ausschließlich Hashfunktionen) aus einer minimalen Annahme konstruieren lassen, nämlich der Existenz sogenannter Einwegfunktionen, ist die Situation im Bereich Public-Key Kryptographie unübersichtlicher. [1] Generell existieren mehrere Familien unvergleichbarer Annahmen, aus welchen sich Public-Key Verfahren mit unterschiedlicher Funktionalität und Sicherheitsgarantien konstruieren lassen.

Die ersten Public-Key Verfahren, so zum Beispiel das RSA- und das ElGamal- Verfahren beruhen alle auf *zahlentheoretischen Strukturen*, und ihre Sicherheit dementsprechend auf komplexitätstheoretischen Problemen in diesen Strukturen. Beispielsweise ist die Sicherheit des RSA Verfahren abhängig von der Schwierigkeit, große Zahlen in ihre Primfaktoren zu zerlegen, und zwar in dem Sinne, dass wenn das Faktorisierungsproblem einfach wäre, RSA unsicher wäre. [2] Das ElGamal Verschlüsselungsverfahren, Basis zahlreicher kryptographischer Verfahren mit fortgeschrittener Funktionalität, basiert auf der sogenannten DDH-Annahme, welche verwandt aber nicht äquivalent zum sogenannten *diskreten Logarithmus (DLOG)* Problem ist. [3]

[1] Die Frage ob sich aus beliebigen Private-Key Verschlüsselungsverfahren bereits Public-Key Verfahren konstruieren lassen ist eines der größten offenen Probleme der theoretischen Kryptographie [82]

[2] Über die wesentlich interessantere Umkehrung dieser Aussage, also der Frage ob die Schwierigkeit des Faktorisierungsproblems bereits die Sicherheit des RSA Verfahrens impliziert ist selbst nach 35 Jahren Forschung relativ wenig bekannt, siehe e.g. [2]

[3] Falls das DLOG Problem einfach ist, ist auch das DDH Problem einfach und damit das ElGamal Verfahren unsicher. Über eine Umkehrung ist derzeit nichts bekannt.

## 2.3 Quanten Unsicherheit

Was allen *zahlentheoretischen* Verschlüsselungsverfahren gemein ist, ist dass sich ihre Sicherheit letztlich als *abelsches Hidden Subgroup Problem (HSP)* darstellen lässt. Ein solches Problem ist charakterisiert durch eine Funktion $f : \mathbb{H} \to S$ welche von einer Gruppe $\mathbb{H}$ (vereinfacht von Vektoren von ganzen Zahlen) in eine Menge $S$ abbildet und folgende Eigenschaft besitzt: Es existiert ein Element $w \in \mathbb{H}\backslash\{0\}$ sodass für jedes $x \in \mathbb{H}$ gilt dass $f(x + w) = f(x)$. Man nennt $w \in \mathbb{H}\backslash\{0\}$ auch eine *versteckte Periode*. Ziel des HSP ist es für ein gegebenes $f$ ein solches $w$ zu finden.

**Beispiel** So lässt sich beispielsweise das DLOG Problem *äquivalent* als HSP darstellen: Gegeben eine DLOG Instanz $(g, h) \in \mathbb{G} \times \mathbb{G}$, bei welcher das Ziel ist eine Zahl $x \in \mathbb{Z}$ zu finden, so dass $h = g^x$, dann lässt sich ein HSP mit folgender Funktion $f : \mathbb{Z}^2 \to \mathbb{G}$ definieren.

$$f(y_1, y_2) = g^{y_1} \cdot h^{y_2}.$$

Hat man ein $w = (w_1, w_2)$ gefunden für welches für alle $(y_1, y_2) \in \mathbb{Z}^2$ gilt, dass $f(y_1+w_1, y_2+w_2) = f(y_1, y_2)$, dann gilt

$$g^{y_1+w_1} \cdot h^{y_2+w_2} = g^{y_1} \cdot h^{y_2}$$

und damit

$$g^{w_1} \cdot h^{w_2} = 1,$$

was wiederum äquivalent ist zu

$$h = g^{-w_1/w_2}.$$

Damit gilt, dass $x = -w_1/w_2$ eine Lösung des DLOG Problems ist.

**Ein Algorithmus für alle HSP** Die Signifikanz des HSP stammt einerseits daher, dass sich hierdurch wichtige zahlentheorie-basierte Verschlüsselungsverfahren wie das RSA und das ElGamal Verfahren brechen lassen. Andererseits daher, dass ein effizienter *Quantenalgorithmus* existiert welcher alle HSP effizient lösen kann, nämlich Shor's Algorithmus und dessen Verallgemeinerungen [132]. Während Details zu Shor's Algorithmus den Umfang dieser Übersicht übersteigen, sei an dieser Stelle nur erwähnt, dass der zentrale Trick dieses Algorithmus in der Verwendung der *Quantenfouriertransformation* besteht.

## 2.4 Postquantum Verfahren

Generell bezeichnet man alle kryptographischen Verfahren welche weder klassisch unsicher noch *trivial unsicher* sind gegen Shor's Algorithmus oder eine seiner Varianten als *post-quantum*. Dies ist dem derzeitigen Forschungsstand geschuldet, da derzeit keine anderen Klassen von Quantenalgorithmen bekannt sind, welche einen *signifikanten Vorteil* gegenüber den besten bekannten klassischen Algorithmen erzielen.

## 2.5 Gitterprobleme und gitterbasierte Probleme

Eine prominente Klasse komplexitätstheoretischer Probleme, welche sich (nach derzeitigem Kenntnisstand) nicht als HSP ausdrücken lassen und dementsprechend als *post-quantum* gelten, sind sogenannte Gitterprobleme. Ziel von Gitterproblemen ist es, vereinfacht gesagt, eine *näherungsweise Lösung* eines hochdimensionalen ganzzahligen linearen Gleichungssystems zu finden. In Matrix-Vektor Schreibweise, gegeben eine Matrix $\mathbf{A} \in \mathbb{Z}^{m\times n}$, einen Vektor $\mathbf{v} \in \mathbb{Z}^m$ und eine positive Zahl $B \in \mathbb{R}$, ist es das Ziel einen Vektor $\mathbf{x}^* \in \mathbb{Z}^n$ zu finden, sodass $\|\mathbf{v} - \mathbf{A}\mathbf{x}^*\| \leq B$, das heißt der Vektor $\mathbf{A}\mathbf{x}^*$ liegt *nahe* (in der euklid'schen Norm) an $\mathbf{v}$ und ist damit eine näherungsweise ganzzahlige Lösung des linearen Gleichungssystems $\mathbf{A}\mathbf{x} = \mathbf{v}$. Derartige Approximationsprobleme treten in vielen praktisch relevanten Optimierungsproblemen auf, und ein effizienter Algorithmus um diese zu lösen hätte bedeutende praktische Konsequenzen. Allerdings sind nach über 60 Jahren Forschung trotz enormer Anstrengungen keine effizienten Algorithmen für derartige Gitterprobleme bekannt.

Im Umkehrschluss macht dies Gitterprobleme für die Kryptographie interessant, da diese damit bereits einer signifikanten Anzahl *Brechversuche* ausgesetzt waren und diese damit aus kryptographischer Sicht kryptanalytisch überlebt haben.

**Einige Gitter-Grundbegriffe** Im Folgenden werden einige mathematische Konzepte zu Gittern kurz eingeführt und erläutert.

- **Gitter** Aus mathematischer Sicht ist ein Gitter eine *diskrete additive Untergruppe* eines Vektorraums $\mathbb{R}^n$. Etwas konkreter lässt sich ein Gitter über eine sogenannte *Gitterbasis* definieren. Sei $\mathbf{B} \in \mathbb{R}^{n \times n}$ eine Matrix von vollem Rang. Dann definieren wir das von $\mathbf{B}$ erzeugte Gitter $\Lambda = \Lambda(\mathbf{B})$ durch

$$\Lambda(\mathbf{B}) = \{\mathbf{B} \cdot \mathbf{z} \mid \mathbf{z} \in \mathbb{Z}^n\},$$

  d.h. das Gitter $\Lambda(\mathbf{B})$ ist die Menge aller ganzzahligen Linearkombinationen der Spalten von $\mathbf{B}$. Zu einem gegebenen Gitter existieren unendlich viele Basen.

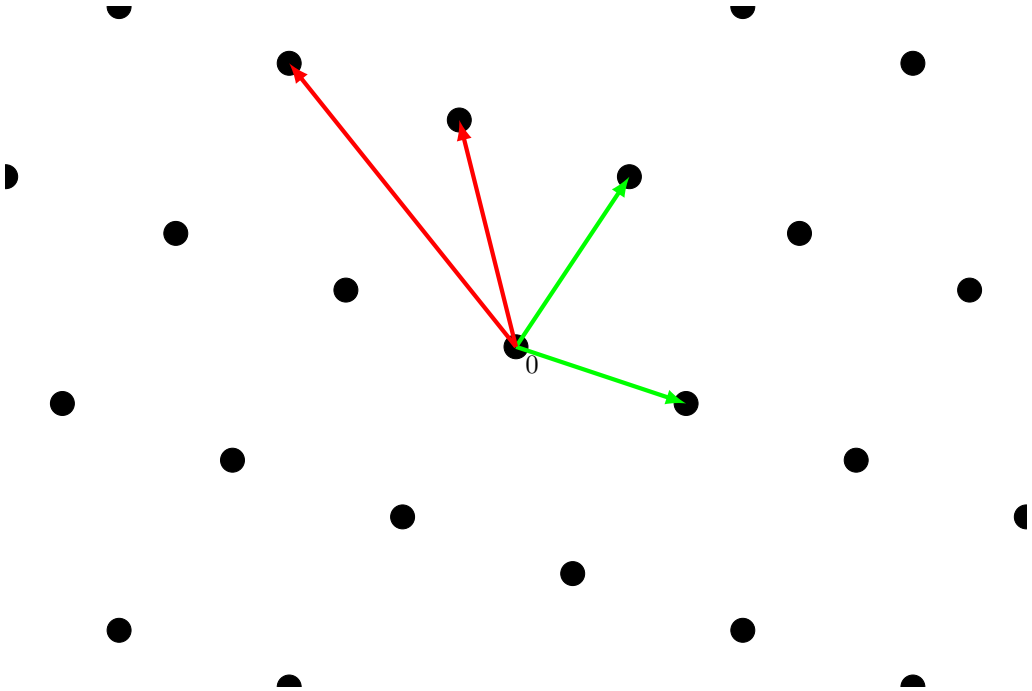



Abbildung 1: Ein zweidimensionales Gitter mit zwei unterschiedlichen Basen (in Rot und Grün)

Um die *Qualität* einer gegebene Basis zu charakterisieren betrachtet man einige geometrische Aspekte von Gittern, welche sowohl aus rein mathematischer als auch aus kryptographischer Sicht interessant sind. Zu den wichtigsten Charakteristika eines Gitters gehört die Minimaldistanz $\lambda_1$, welche als Länge des kürzesten von 0 verschiedenen Vektors in $\Lambda$ ist, d.h.

$$\lambda_1 = \lambda_1(\Lambda) = \min_{\mathbf{x} \in \Lambda \setminus \{0\}} \|\mathbf{x}\|.$$

Der Wert $\lambda_1$ charakterisiert wie gut sich ein Gitter als *fehlerkorrigierender Code* gegen euklidische Fehler eignet. Spezieller, ist $\mathbf{x} \in \Lambda$ ein Gitterpunkt und ist $\mathbf{e} \in \mathbb{R}^n$ ein *Fehlerterm* mit Länge kleiner als $\lambda_1/2$, d.h. $\|\mathbf{e}\| < \lambda_1/2$, dann lässt sich der *fehlerbehaftete Punkt* $\tilde{\mathbf{x}} = \mathbf{x} + \mathbf{e}$ eindeutig zu $\mathbf{x}$ korrigieren.

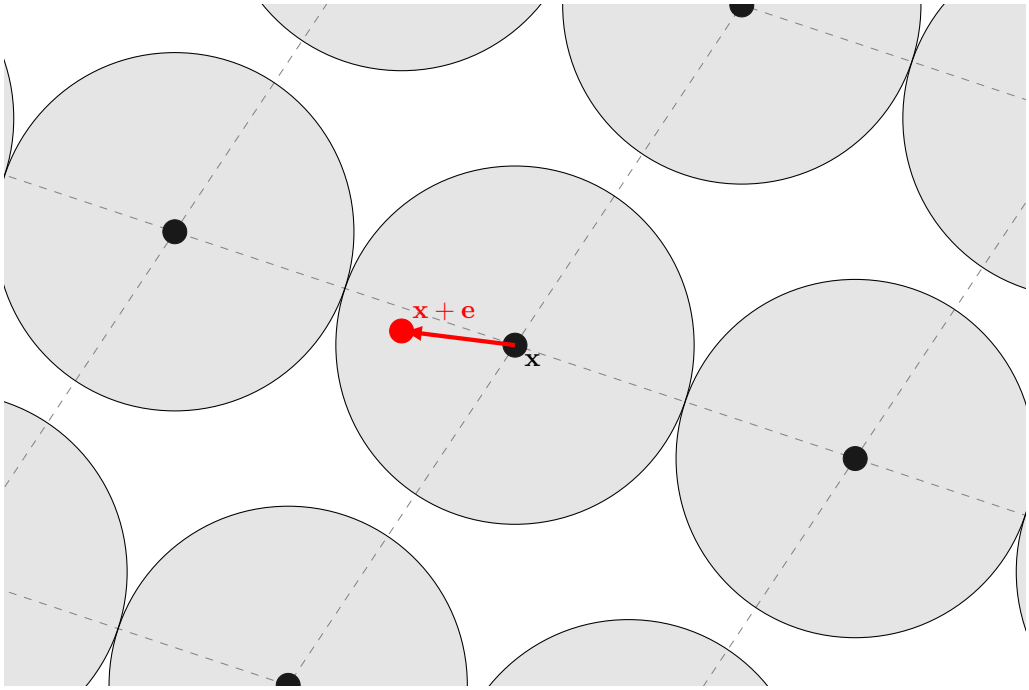



Abbildung 2: Fehler die kürzer als $\lambda_1/2$ sind lassen sich korrigieren.

Für eine gegebene Basis $\mathbf{B}$ bezeichnet man

$$\|\mathbf{B}\| = \max_{\mathbf{x}\in\mathbb{R}^n, \|\mathbf{x}\|=1} \|\mathbf{B}\mathbf{x}\|$$

als die Norm oder auch Länge der Basis. Generell sind, aus algorithmischer Sicht, algebraische Eigenschaften einfach zu berechnen, die Berechnung von geometrischen Eigenschaften dagegen recht schwierig, weshalb sich diese sehr gut als Grundlage für die Sicherheit kryptographischer Verfahren eignen. Beispielsweise ist die Berechnung von $\lambda_1$ für ein Gitter gegeben durch eine Basis $\mathbf{B}$ schwierig im *Worst-Case*. Für kryptographische Anwendungen betrachtet man typischerweise Ganzzahlgitter (engl. Integer-Lattices), also Gitter, welche von einer ganzzahligen Basis $\mathbf{B} \in \mathbb{Z}^{n\times n}$ erzeugt werden.

- **Dualgitter**: Zu einem gegebenen Gitter $\Lambda$ lässt sich ein sogenanntes *Dualgitter*

$$\Lambda^{\perp}(\Lambda) = \{\mathbf{z} \in \mathbb{R}^n \mid \forall \mathbf{x} \in \Lambda : \mathbf{z} \cdot \mathbf{x} \in \mathbb{Z}\}$$

definieren. Für ein gegebenes $\Lambda$ bezeichnet man eine Basis $\mathbf{T}$ von $\Lambda^{\perp}(\Lambda)$ als *Dualbasis* von $\Lambda$. Ist $\mathbf{B}$ eine Basis von $\Lambda$ und $\mathbf{T}$ eine Dualbasis von $\Lambda$, so gilt immer $\mathbf{T}\cdot\mathbf{B} \in \mathbb{Z}^n$ und $\mathbf{B}\cdot\mathbf{T} \in \mathbb{Z}^n$. Eine *kurze Dualbasis* lässt sich zur effizienten Fehlerkorrektur in einem Gitter verwenden, und damit beispielsweise auch als Secret-Key bei gitterbasierten Verschlüsselungsverfahren. Ist z.B. $\tilde{\mathbf{x}} = \mathbf{x} + \mathbf{e}$ für einen kurzen Fehler $\mathbf{e}$, so lässt sich

$$\tilde{\mathbf{e}} = \mathbf{T}\tilde{\mathbf{x}} = \underbrace{\mathbf{T}\mathbf{x}}_{\in\mathbb{Z}^n} + \mathbf{T}\mathbf{e}$$

berechnen. Ist $\mathbf{e}$ kurz genug sodass $\|\mathbf{T}\mathbf{e}\| < 1/2$, so lässt sich $\hat{\mathbf{x}} = \mathbf{T}\mathbf{x}$ durch *Rundung* von $\tilde{\mathbf{e}}$ auf den nächsten ganzzahligen Vektor bestimmen, und damit $\mathbf{x}$ durch Berechnung von $\mathbf{x} = \mathbf{T}^{-1}\hat{\mathbf{x}}$.

- **$q$-äre Gitter** Eine für kryptographische Anwendungen enorm wichtige Klasse von Gittern sind sogenannte $q$-äre oder $q$-periodische Gitter. Diese sind Ganzzahlgitter die dadurch charakterisiert sind, dass sie

$$q\mathbb{Z}^n = \{q\mathbf{z} \mid \mathbf{z} \in \mathbb{Z}^n\}$$

als *Teilgitter* enthalten. Ein solches Gitter $\Lambda$ lässt sich immer mittels einer $q$-Erzeugermatrix $\mathbf{A} \in \mathbb{Z}_q^{n \times k}$ beschreiben als

$$\Lambda = \Lambda_q(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^n \mid \exists \mathbf{z} \in \mathbb{Z}_q^n s.t. \mathbf{x} \mod q = \mathbf{A}\mathbf{z}\}.$$

Alternativ lässt sich ein $q$-äres Gitter $\Lambda$ auch mittels einer *Prüfmatrix* $\mathbf{H} \in \mathbb{Z}_q^{t \times n}$ beschreiben durch

$$\Lambda = \Lambda_q^{\perp}(\mathbf{H}) = \{\mathbf{x} \in \mathbb{Z}^n \mid \mathbf{H} \cdot \mathbf{x} = 0 \mod q\}.$$

Für $q$-äre Gitter betrachtet man normalerweise immer um den Faktor $q$ hochskalierte Dualbasen. Ist $\mathbf{T}'$ eine Dualbasis von $\Lambda(\mathbf{A})$ und ist $\mathbf{T} = q \cdot \mathbf{T}'$, so gilt immer $\mathbf{T} \cdot \mathbf{A} = 0$.

- **Idealgitter** Eine weitere für kryptographische Anwendungen wichtige Klasse von Gittern sind sogenannte *Idealgitter*. Während reguläre Gitter lediglich auf $\mathbb{Z}^n$ definiert sind, sind Idealgitter auf Räumen definiert welche noch eine zusätzliche Ringstruktur aufweisen. Speziell, bei derartigen Verfahren ist auf dem umgebenden Raum noch eine Multiplikationsoperation $*$ definiert, welcher dadurch mathematisch gesehen zu einem Ring $R$ wird. Ein wichtiger Aspekt hierbei ist, dass in einem solchen Ring bei Multiplikation die Länge von Ringelementen nur wenig vergrößert wird, d.h. sind beispielsweise $\mathbf{e}_1, \mathbf{e}_2 \in R$ zwei (kurze) Elemente, so soll sichergestellt sein dass $\|\mathbf{e}_1 * \mathbf{e}_2\| \leq \gamma \|\mathbf{e}_1\| \cdot \|\mathbf{e}_2\|$ für eine möglichst kleine Dehnungskonstante $\gamma$, welche nicht von den Elementen $\mathbf{e}_1, \mathbf{e}_2$ abhängig ist.

  Idealgitter sind nun Ideale des Rings $R$, also additive Untergruppen von $R$ welche unter Multiplikation mit Elementen von $R$ abgeschlossen sind.

**Worst-Case Hardness** Ein zentraler Aspekt gitterbasierter Kryptographie ist, dass diese im Gegensatz zu den meisten zahlentheoriebasierten Verfahren auf sogenannten Worst-Case-Hardness Annahmen beruht, letztere hingegen auf Average-Case-Hardness Annahmen. Ein Problem ist worst-case-hart, falls für jeden effizienten Algorithmus eine Instanz existiert auf welcher dieser scheitert. Ein Problem ist hingegen average-case-hart, wenn jeder effiziente Algorithmus auf einer zufällig gewählten Instanz scheitert.

Da es bei worst-case-harten Problemen generell nicht-trivial ist schwere Instanzen zu finden, geschweige denn *effizient zu generieren*, sind average-case-harte Probleme der Ausgangspunkt für die Konstruktion kryptographischer Verfahren, da bei diesen bereits eine zufällig erzeugte Instanz schwierig ist. Ein average-case-hartes Problem ist damit immer auch ein worst-case-hartes Problem, wohingegen in die andere Richtung *im Allgemeinen* wenig ausgesagt werden kann.

Tatsächlich basieren *alle* zahlentheoriebasierten kryptographischen Verfahren *direkt* auf average-case-harten Problemen, und kein direkter Zusammenhang mit den zugrundeliegenden worst-case-harten Problemen ist bekannt. So zum Beispiel reicht es für die Sicherheit von Verschlüsselungsverfahren welche auf dem Faktorisierungsproblem beruhen nicht aus, dass schwierige Instanzen des Faktorisierungsproblems existieren, sondern es muss explizit angenommen werden, dass zufällig generierte Instanzen dieses Problems schwierig sind.

Im Bereich der gitterbasierten Kryptographie ist die Situation eine andere: Während auch hier meist kryptographische Verfahren basierend auf average-case-harten Annahmen konstruiert werden, besitzen die hier auftretenden average-case-harten Probleme eine sogenannte *Worst-to-Average-Case Reduktion*. D.h. für diese Probleme existiert eine Transformation, welche *jeden* effizienten Algorithmus, welcher auf zufällig erzeugten Instanzen *häufig* eine Lösung findet, in einen Algorithmus umbaut, welcher für jede Instanz eine Lösung findet. Leider haben derartige Worst-to-Average-Case Reduktionen einen hohen Sicherheitsverlust, was in der Praxis bedeutet, dass typischerweise als sicher angenommene Instanziierungen von Verfahren nur eine Reduktion auf einfache Instanzen des zugrundeliegenden Problems erlauben. Möchte man Instanziierungen, welche beweisbar auf schweren Instanzen des zugrundeliegenden Problems basieren, so benötigt man aus praktischer Sicht unrealistisch große Parameterwahlen, typischerweise um mindestens einen Faktor 100 größer als bei derzeit für den praktischen Einsatz vorgeschlagenen Parameterwahlen.

**Gap-SVP** Ziel des *Gap-SVP* Problems $\gamma$-Gap-SVP ist es, gegeben ein *Gitter* $\Lambda = \Lambda(\mathbf{B})$ definiert durch eine Basis $\mathbf{B}$ und eine positive Zahl $B$, zu entscheiden ob $\Lambda$ einen Vektor der Länge $B$ besitzt, oder ob alle

Vektoren in $\Lambda$ Länge mindestens $\gamma \cdot B$ haben. Hierbei ist $\gamma \geq 1$ ein Problemparameter, welcher die "Losigkeit" des Problems festlegt. Dabei entspricht $\gamma = 1$ dem exakten Shortest Vector Problem. $\gamma$-Gap-SVP ist damit ein sogenanntes *Promise-Problem*, bei welchem garantiert wird, dass Eingaben eine von zwei Bedingungen erfüllt, welche nicht notwendigerweise alle Möglichkeiten abdecken. Für den Fall, dass die Länge des kürzesten Vektors von $\Lambda$ zwischen $B$ und $\gamma B$ liegt, wird keine Forderung gemacht.

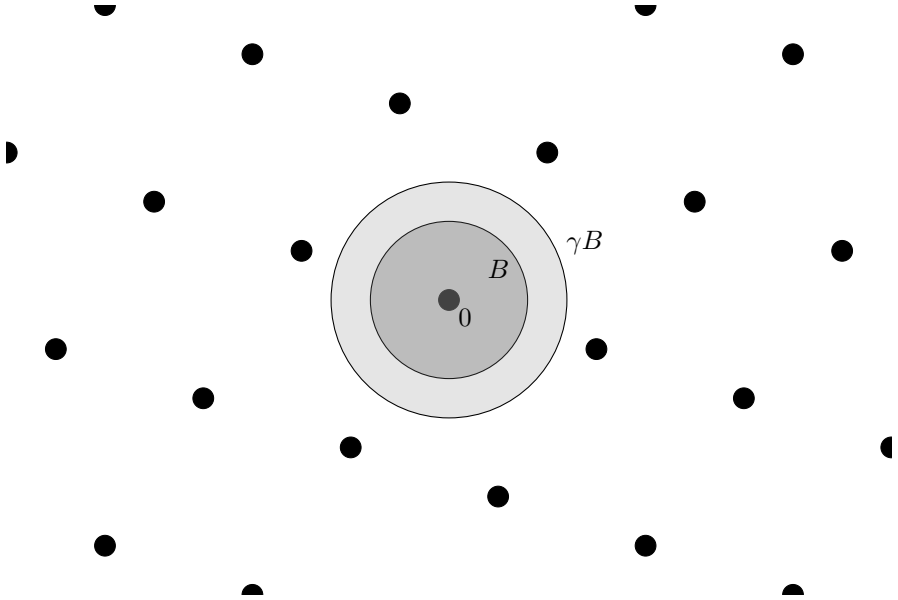



Abbildung 3: GapSVP Nein-Instanz: Alle Nichtnull-Vektoren in $\Lambda$ sind länger als $\gamma B$

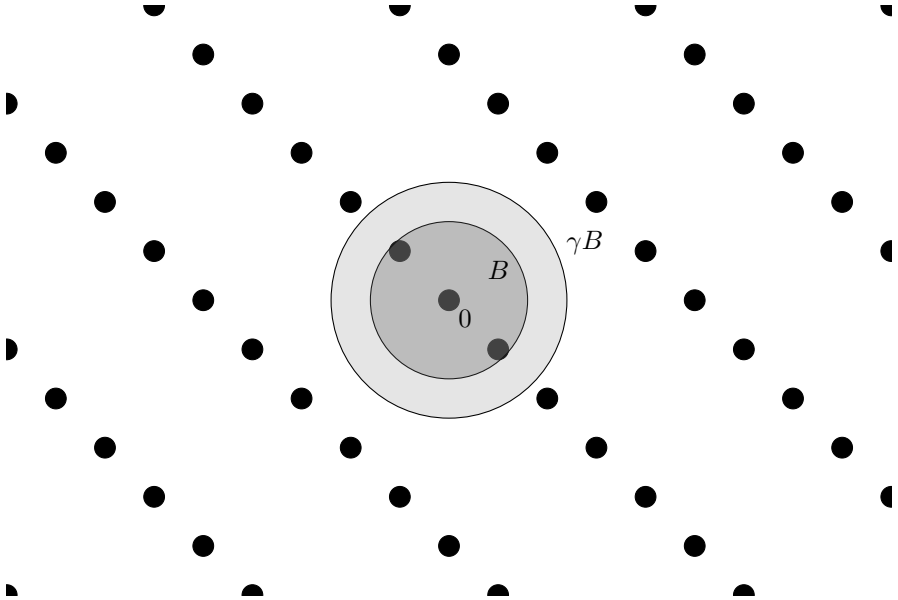



Abbildung 4: GapSVP Ja-Instanz: Es existiert ein Vektor kürzer als $B$

**Bounded Distance Decoding** Ziel des Bounded-Distance-Decoding Problems $\alpha$-BDD (wobei $0 < \alpha < 1/2$) ist es, gegeben ein Gitter $\Lambda$ mit Minimaldistanz $\lambda_1$ und einen Punkt $\mathbf{t} \in \mathbb{R}^n$ mit der Eigenschaft, dass

ein Punkt $\mathbf{x} \in \Lambda$ mit Abstand höchstens $\alpha\lambda_1$ von $\mathbf{t}$ existiert, einen solchen Punkt $\mathbf{x}$ zu finden.

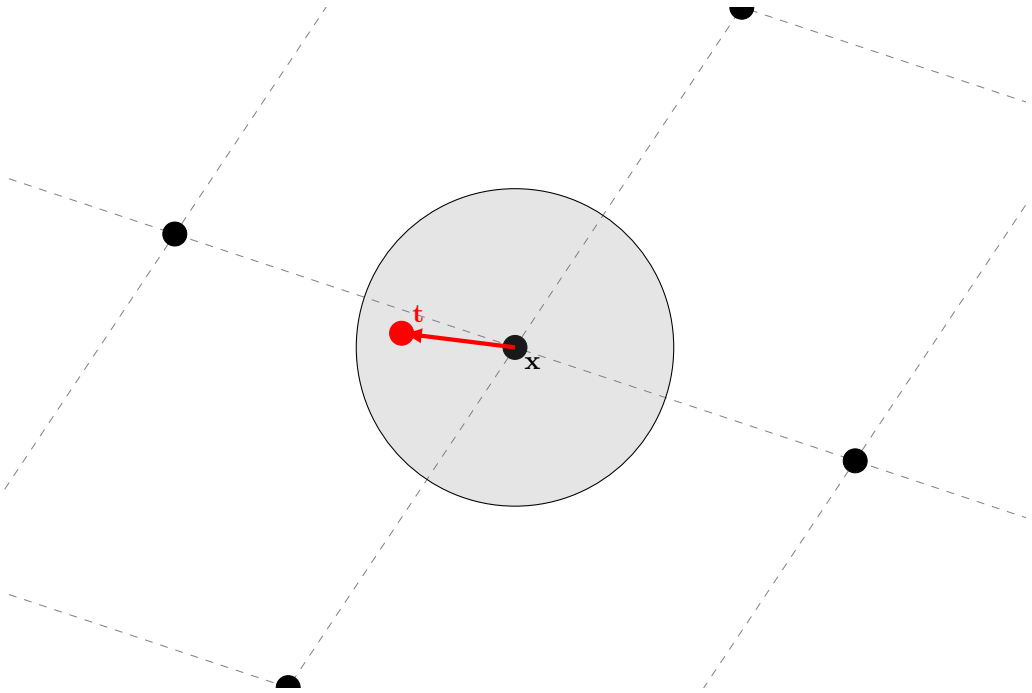



Abbildung 5: Bounded Distance Decoding: Finde $\mathbf{x}$ gegeben $\mathbf{t}$, wobei garantiert ist dass sich $\mathbf{t}$ in der grauen Kugel um genau ein $\mathbf{x}$ befindet.

**Learning With Errors** Die Gitterprobleme Gap-SVP und BDD *erben* sehr viel der algebraischen Struktur von Gittern. Um also sinnvoll über diese Probleme zu diskutieren müssen zunächst einige Grundbegriffe von Gittern eingeführt werden. Tatsächlich hat sich herausgestellt, dass diese Konzepte und Abstraktionsebenen nicht notwendig sind, um gitterbasierte kryptographische Verfahren zu konstruieren und zu analysieren.

Das Learning With Errors (LWE) Problem [124] stellt eine Art *Interface* dar, mit welchem sich gitterbasierte Hardness konzeptionell einfach zur Konstruktion kryptographischer Verfahren verwenden lässt und andererseits explizite Gitterkonzepte vermeidet.

Sei $q \in \mathbb{N}$ nun ein hinreichend großer Modulus. Ziel des LWE Suchproblems ist es, gegeben eine gleichverteilt zufällige Matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ und $\mathbf{y} = \mathbf{sA} + \mathbf{e}$ den Wert $\mathbf{s} \in \mathbb{Z}_q^n$ zu finden, wobei $\mathbf{s}$ gleichverteilt aus $\mathbb{Z}_q^n$ gezogen wurde und $\mathbf{e} \in \mathbb{Z}^m$ einer kurzen Verteilung folgt, häufig einer *diskreten Normalverteilung*.

Beim sogenannten Decision-LWE Problem ist es das Ziel, gegeben eine gleichverteilt zufällige Matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ und einen Vektor $\mathbf{y} \in \mathbb{Z}_q^m$, zu entscheiden ob $\mathbf{y}$ gleichverteilt zufällig aus $\mathbb{Z}_q^m$ gezogen wurde oder anhand von $\mathbf{y} = \mathbf{sA} + \mathbf{e}$, für ein gleichverteilt zufälliges $\mathbf{s}$ und ein $\mathbf{e}$ aus einer kurzen Verteilung.

Regev [124] konnte zeigen dass sowohl das LWE-Suchproblem als auch das Decisional-LWE Problem, welche beide Average-Case Probleme mit einfacher Instanzerzeugung sind, genauso schwierig sind wie einige worst-case Gitterproblem, so zum Beispiel das Shortest Independent Vectors Problem (SIVP) oder das Gap-SVP Problem. Leider ist diese Reduktion, wie im Paragraph *Worst-Case Hardness* oben dargelegt wenig instruktiv für praktische Parameterwahlen, was bedeutet dass diese derzeit weiterhin kryptanalytisch bestimmt werden. Allerdings garantiert die Worst-to-Average-Case Reduktion des LWE Problems dass dieses (nach derzeitigem Wissensstand) *strukturell* sicher ist, also keine vollständigen Brüche darauf basierender Verfahren zu erwarten sind.

Durch seine strukturelle Einfachheit und Versatilität hat LWE eine Revolution in der modernen Kryptographie ausgelöst. Ein fundamentaler Aspekt dabei ist, dass LWE es erlaubt direkte Berechnungen auf einer *Darstellung* von Instanzen durchzuführen, da diese schlicht Zahlen (modulo $q$) sind und nicht auf

komplizierteren algebraischen Objekten wie elliptischen Kurven definiert sind.

**Ring Learning With Errors** Das Ring Learning With Errors Problem (RLWE) [99] ist ähnlich zum LWE Problem definiert, allerdings wird hier ein *Ganzheitsring in einem algebraischen Zahlkörper* statt der ganzen Zahlen verwendet. Genauer, sei $\mathcal{R} \subseteq \mathcal{K}$ ein Ganzheitsring in einem algebraischen Zahlkörper $\mathcal{K} \subseteq \mathbb{C}$. [4] Ist $\mathcal{K}$ ein *Kreisteilungskörper* vom Grad $2^n$, also $\mathcal{K} = \mathbb{Q}[X]/(\Phi(X))$, wobei $\Phi(X) \in \mathbb{Q}[X]$ ein irreduzibles Polynom mit Nullstellen $e^{2\pi ik/2^n}$ (für $k \in \mathbb{N}$) ist, so ist der zugehörige Ganzheitsring $\mathcal{R} = \mathbb{Z}[X]/(\Phi(X))$, hat also eine relativ einfache Arithmetik.

Das RLWE Problem ist nun wie folgt definiert. Sei $q \in \mathbb{N}$ wieder ein hinreichend großer Modulus und $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$ der $q$-äre Restklassenring von $\mathcal{R}$. Ziel des RLWE Suchproblems ist es, gegeben ein gleichverteilt zufälliges $\mathbf{a} \in \mathcal{R}_q^m$ und $\mathbf{y} = s \cdot \mathbf{a} + \mathbf{e}$ das Geheimnis $s \in \mathcal{R}_q$ zu finden. Hierbei ist $s \in \mathcal{R}_q$ gleichverteilt zufällig gezogen und $\mathbf{e} \in \mathcal{R}^m$ folgt einer kurzen Verteilung, typischerweise einer Normalverteilung.

Beim Decision RLWE Problem ist es das Ziel ein solches Paar $(\mathbf{a}, s\mathbf{a} + \mathbf{e})$ von $(\mathbf{a}, \mathbf{u})$ zu unterscheiden, wobei $\mathbf{u} \in \mathcal{R}_q^m$ gleichverteilt zufällig gezogen wird.

In [99] wurde gezeigt, dass das RLWE *mindestens so schwer* ist wie das Worst-Case Shortest Vector Problem in Idealgittern in $\mathcal{R}$. Neuere Forschungen haben allerdings gezeigt, dass letzteres Problem wahrscheinlich einfacher ist als RLWE [48], daher ist diese Reduktion möglicherweise wenig aussagekräftig.

Während also RLWE möglicherweise aus kryptanalytischer Sicht ein leichteres Problem ist als LWE, so liegt der Hauptvorteil von RLWE in effizienteren arithmetischen Operationen und damit effizienteren Implementierungen. Während eine Multiplikation mit einer Matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ ungefähr $n \cdot m = O(n^2)$ arithmetische Operationen benötigt, so benötigt Multiplikation mit einem Vektor $\mathbf{a} \in \mathcal{R}_q^m$ lediglich $O(m \cdot \log(n))$ arithmetische Operationen bei Verwendung einer schnellen Fouriertransformation [98]. Damit sind Verfahren, welche auf dem RLWE Problem basieren, in Effizienzaspekten vergleichbar mit zahlentheoretischen Verfahren. Also stellt das RLWE Problem einen Kompromiss zwischen Sicherheit und Effizienz dar.

## 2.6 Angriffe auf gitterbasierte Annahmen

**Gitterreduktion** Das *standard Kryptanalysewerkzeug* in der gitterbasierten Kryptographie ist Gitterreduktion. Grundprinzip dabei ist es eine *schlechte* Gitterbasis in eine *gute* Gitterbasis zu transformieren.

Eine schlechte Basis ist im wesentlichen dadurch charakterisiert, dass ihre Basisvektoren lange sind. Mit einer schlechten Gitterbasis lassen sich zwar algebraische Fragen über Gitter effizient entscheiden, so zum Beispiel ob ein Punkt $\mathbf{x} \in \mathbb{R}^n$ in einem Gitter $\Lambda \subseteq \mathbb{R}^n$ liegt oder nicht. Andererseits sind keine Algorithmen bekannt, welche allein mittels einer schlechten Basis geometrische Probleme wie das Shortest Vector Problem, Closest Vector Problem oder Bounded Distance Decoding effizient lösen können. Die Hermite Normalform [104] $\bar{\mathbf{B}}$ einer Basis $\mathbf{B}$, welche allein vom Gitter $\Lambda(\mathbf{B})$ aber nicht der Basis $\mathbf{B}$ selbst abhängt, stellt gewissermaßen eine *schlechteste* Basis eines Gitters $\Lambda$ dar, da sie sich aus jeder anderen Basis berechnen lässt.

Eine *gute* Basis ist im Gegensatz dazu im Allgemeinen schwer zu finden. Wäre dies nicht so, so würden sich gitterbasierte Probleme nicht für Kryptographie eignen. Mit einer guten Basis lassen sich das Shortest Vector Problem, Closest Vector Problem und Bounded Distance Decoding näherungsweise lösen, wobei die Qualität der Lösung von der Qualität der Basis abhängt.

Der Lenstra-Lenstra-Lovasz (LLL) Algorithmus [92] ist in gewissem Sinne der elementare Gitterreduktionsalgorithmus und basiert auf der *Gram-Schmidt Orthogonalisierung*. Zu einer gegebenen Basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n) \in \mathbb{R}^{n \times n}$ ist die Gram-Schmidt Orthogonalisierung eine *orthogonale* Matrix $\tilde{\mathbf{B}} = (\tilde{\mathbf{b}}_1, \ldots, \tilde{\mathbf{b}}_n) \in \mathbb{R}^{n \times n}$, sodass jedes $\tilde{\mathbf{b}}_i$ jeweils im Spann von $\mathbf{b}_1, \ldots, \mathbf{b}_i$ liegt. Eine Matrix heißt dabei orthogonal, wenn je zwei ihrer Spalten ein inneres Produkt von 0 haben, das heißt es gilt $\mathbf{b}_i \cdot \mathbf{b}_j = 0$ für $i \neq j$. Die Gram-Schmidt Orthonolisierung lässt sich mittels einfacher linearer Algebra berechnen. In der Gram-Schmidt Basis $\tilde{\mathbf{B}}$ hat

[4] das bedeutet dass $\mathcal{R}$ die Menge aller Elemente in $\mathcal{K}$ ist, welche eine Ganzheitsgleichung erfüllen, d.h. die Menge aller $z \in \mathcal{K}$ für welche $f(z) = 0$ gilt für ein Polynom $f \in \mathbb{Z}[X]$ mit Leitkoeffizient 1.

die Matrix $\mathbf{B}$ Dreiecksform, die Matrix $\mathbf{B}$ lässt sich also darstellen als

$$\mathbf{B} = \begin{pmatrix} 1 & \mu_{1,2} & \mu_{1,3} & \dots & \mu_{1,n-1} & \mu_{1,n} \\ 0 & 1 & \mu_{2,3} & \dots & \mu_{2,n-1} & \mu_{2,n} \\ \vdots & & \ddots & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & 1 & \mu_{n-1,n} \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}.$$

Eine Basis $\mathbf{B}$ heißt dabei *$\delta$-LLL-reduziert*, falls

1. für alle $j > i$ gilt dass $\mu_{i,j} < 1/2$,
2. für alle $i = 1, \dots, n-1$ gilt dass $\delta \|\tilde{\mathbf{b}}_i\| \leq \|\mu_{i,i+1}\tilde{\mathbf{b}}_i + \tilde{\mathbf{b}}_{i+1}\|$.

Das zweite Kriterium hierbei stellt sicher dass $\tilde{\mathbf{b}}_{i+1}$ nicht "viel" kürzer sein kann als $\tilde{\mathbf{b}}_i$, wobei "viel" durch den Parameter $\delta$ quantifiziert wird. Der LLL-Algorithmus ist nun ein einfacher iterativer Algorithmus welcher in jedem Schritt zunächst die Gram-Schmidt Orthogonalisierung $\tilde{\mathbf{B}}$ berechnet, dann durch Ganzzahl-Spaltenoperationen sicherstellt, dass das erste Kriterium erfüllt ist, und falls das zweite Kriterium verletzt ist, die entsprechenden Spalten vertauscht und in die nächste Iteration geht. Es kann gezeigt werden, dass der LLL Algorithmus für Werte von $\delta$ in $(1/4, 1)$ eine polynomielle Laufzeit in $n$ hat.

Mit einer LLL-reduzierten Basis lassen sich nun SVP, CVP und BDD exponentiell approximieren, daher liefert LLL Schranken für die Parameterwahlen bei diesen Problemen. Weiter lassen sich ausgehend von einer LLL-reduzierten Basis alle diese Probleme in Exponentialzeit mittels Brute-Force Suche exakt lösen.

Beispielsweise lässt sich LWE mit dem LLL Algorithmus in etwa wie folgt angreifen. Gegeben eine LWE Instanz $(\mathbf{A}, \mathbf{y} = \mathbf{sA} + \mathbf{e})$ definiert man zunächst das Gitter $\Lambda^{\perp}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^m \mid \mathbf{Ax} = 0 \mod q\}$ und berechnet eine (beliebige) Basis $\mathbf{B} \in \mathbb{Z}^{m \times m}$ von $\Lambda^{\perp}(\mathbf{A})$ (eine solche Basis lässt sich mittels einfacher linearer Algebra aus $\mathbf{A}$ berechnen). Für jede Basis $\mathbf{B}$ von $\Lambda^{\perp}(\mathbf{A})$ gilt (per Definition) $\mathbf{AB} = 0 \mod q$. Nun berechnet man aus $\mathbf{B}$ eine LLL-reduzierte Basis $\mathbf{B'}$. Ist nun $\mathbf{B'}$ hinreichend kurz, so lassen sich damit wie folgt die $\mathbf{s}$ und $\mathbf{e}$ aus $\mathbf{y} = \mathbf{sA} + \mathbf{e}$ berechnen. Es gilt

$$\mathbf{yB'} = \mathbf{s}\underbrace{\mathbf{AB'}}_{=0} + \mathbf{eB'} = \mathbf{eB'} \mod q.$$

Sind nun sowohl $\mathbf{e}$ als auch $\mathbf{B'}$ hinreichend kurz, so ist auch $\mathbf{eB'}$ kurz, und damit gilt

$$\mathbf{yB'} \mod q = \mathbf{eB'},$$

es gilt also $\mathbf{yB'} \mod q = \mathbf{eB'}$ über den den ganzen Zahlen $\mathbb{Z}$ (anstatt nur über $\mathbb{Z}_q$). Nun lässt sich der Fehler $\mathbf{e}$ durch $\mathbf{e} = \mathbf{B}^{-1}(\mathbf{yB'} \mod q)$ berechnen, wobei die Inverse $\mathbf{B'} \in \mathbb{Q}^{m \times m}$ eine rationale Matrix ist. Da das Gleichungssystem $\mathbf{y} = \mathbf{sA} + \mathbf{e}$ mit hoher Wahrscheinlichkeit (über die Wahl von $\mathbf{A}$ und $\mathbf{e}$) eine eindeutige Lösung besitzt, wird diese durch die obige Prozedur gefunden. Hat man $\mathbf{e} \in \mathbb{Z}^m$ gefunden, so lässt sich $\mathbf{s} \in \mathbb{Z}_q^n$ durch lösen des überbestimmten Gleichungssystems $\mathbf{sA} = \mathbf{y} - \mathbf{e}$ finden.

**Arora-Ge Angriffe auf LWE** Der Arora-Ge Angriff [11] umschreibt eine Klasse *algebraischer* Angriffe auf das LWE Problem. Zunächst wird eine LWE Instanz $(\mathbf{A}, \mathbf{y} = \mathbf{sa} + \mathbf{e})$ in Spalten $(\mathbf{a}_1, y_1 = \mathbf{sa}_1 + e_1), \dots, (\mathbf{a}_m, y_m = \mathbf{sa}_m + e_m)$ zerlegt. Hauptbeobachtung hierbei ist, dass der Fehlerterm $\mathbf{e}$ in LWE Instanzen kurz ist. Falls $\mathbf{e}$ $B$-kurz ist, also falls für jedes $i$ gilt dass $|e_i| < B$ dann gilt trivialerweise auch dass $e_i \in (-B, B)$. Nun lässt sich mit dieser Beobachtung die Beziehung $y = \mathbf{sa} + e$ ausdrücken als

$$f_{\mathbf{a},y}(\mathbf{s}) = \prod_{z \in (-B,B)} (y - \mathbf{sa} - z) = 0.$$

Das bedeutet dass für gegebene $(\mathbf{a}_i, y_i)$ das *richtige* LWE Geheimnis $\mathbf{s}$ eine gemeinsame Nullstelle der Polynome $f_{\mathbf{a}_1,y_1}, \dots, f_{\mathbf{a}_m,y_m}$ ist. Derartige *multivariate* polynomielle Gleichungssysteme zu lösen ist im Allgemeinen

NP-hart, mit dieser Umformulierung ist daher im Allgemeinen zunächst kein Effizienzgewinn zu erwarten. Allerdings beobachten Arora und Ge nun, dass wenn $m \gg n$, also die Anzahl der verfügbaren Gleichungen sehr viel größer ist als die Anzahl der Unbekannten in $\mathbf{s}$, dann lässt sich ein derartiges Gleichungssystem *linearisieren*. Genauer bedeutet dies, dass *gemischte Monome* $\prod_{i=1}^{n} s_i^{r_i}$ in den Ausdrücken $f_{\mathbf{a},y}(\mathbf{s}) = 0$ durch neue symbolische Variablen $t_{r_1,\ldots,r_n}$ ersetzt werden, und das daraus resultierende Gleichungssystem in den $t_*$ linear wird. Falls $m$ hinreichen groß ist, genauer falls $m > \binom{n+2B}{B}$, dann hat dieses lineare Gleichungssystem garantiert eine *eindeutige* Lösung, welche damit auch eine Lösung des nichtlinearen Gleichungssystems darstellt.

Falls $B$ sehr kleine Werte annimmt, beispielsweise $B = 2$, was heißt dass die Fehlerkomponenten $e_i$ Werte auf $\{-1, 0, 1\}$ annehmen, dann bricht dieser Algorithmus erfolgreich LWE sobald mehr als $\approx n^2$ Gleichungen vorliegen. Dies bedeutet insbesondere, dass bei der Wahl der Verteilung der Fehlerterme $e_i$ keine zu kurze Verteilung verwendet werden darf.

**Decoding Angriffe auf LWE** Die Idee von Decoding Angriffen auf LWE besteht darin für eine gegebene LWE Instanz $(\mathbf{A}, \mathbf{y} = \mathbf{sA} + \mathbf{e})$ zunächst einen kurzen Vektor $\mathbf{x}$ zu finden für welchen gilt dass $\mathbf{Ax} = 0$. Hat man einen derartigen Vektor $\mathbf{x}$ gefunden, so gilt dass

$$\mathbf{yx} = (\mathbf{sA} + \mathbf{e})\mathbf{x} = \mathbf{s}\underbrace{\mathbf{Ax}}_{=0} + \mathbf{ex} = \mathbf{ex}.$$

Da sowohl $\mathbf{e}$ als auch $\mathbf{x}$ kurz sind, ist auch das innere Produkt $\mathbf{ex}$ kurz. Mit einem solchen Vektor $\mathbf{x}$ lassen sich also LWE Instanzen von zufälligen Instanzen unterscheiden, da für ein zufälliges $\mathbf{y}$ das innere Produkt $\mathbf{yx}$ mit hoher Wahrscheinlichkeit nicht kurz ist. Verschiedene Verfahren einen solchen kurzen Vektor $\mathbf{x}$ zu finden wurden unter anderem in [16, 5, 88] vorgeschlagen.

**Log-S-Unit Angriffe** Das Shortest Vector Problem in *Hauptidealgittern* (Principal Ideal Lattices) auf welchem die Sicherheit der FHE Verfahren der ersten Generation basieren, erlaubt strukturelle Angriffe welche auf allgemeinen Gittern nicht möglich sind. Eine wichtige Rolle spielt dabei das sogenannten *Log-S-Unit Lattice* [48]. Zu einem derartigen Gitter lässt sich mittels einer *logarithmischen Einbettung* in die komplexen Zahlen ein weiteres Gitter definieren. Mittels kurzer Vektoren in diesem Gitter lässt sich dann das SVP Problem im ursprünglichen Gitter lösen.

# 3 Gitterbasierte Verschlüsselungs-Verfahren

Im folgenden werden wir die im Zusammenhang mit Fully-Homomorphic-Encryption wichtigsten gitterbasierten Verschlüsselungsverfahren diskutieren. Alle diese Verfahren weisen eine begrenzte additive Homomorphie-Eigenschaft auf, welche bei der Konstruktion vollhomomorpher Verfahren ausgenützt wird. Perspektivisch gesehen sind FHE Verfahren ein Beiprodukt der Erforschung von Postquantum Verfahren, da man erst hierdurch auf die Techniken stieß, welche mit anderen Konstruktionsparadigmen, also beispielsweise im Bereich der zahlentheoriebasierten Kryptographie, nicht möglich sind.

## 3.1 Das Goldreich-Goldwasser-Halevi Verfahren

Eines der ersten *gitterbasierten* Verschlüsselungsverfahren wurde von Goldreich, Goldwasser und Halevi [73] vorgeschlagen. Im folgenden werden wir, anstatt die konkrete Instantiierung von [73] vorzustellen, die dem Verfahren zugrundeliegende Blaupause diskutieren. Die grundlegende Ideen dieses Verfahrens sind die folgenden.

- **Schlüsselerzeugung:** Ein Secret Key bei diesem Verfahren ist eine zufällig erzeugte *gute* Basis $\mathbf{B}$ eines Gitters $\Lambda$, i.e. man zieht ein $\mathbf{B}$ von einer bestimmten *guten* Verteilung und setzt $\Lambda = \Lambda(\mathbf{B})$. Den zugehörigen öffentlichen Schlüssel $\mathbf{A}$ berechnet man nun als *Normalform* von $\mathbf{B}$, typischerweise die

Hermite-Normalform [104]. Damit gilt dass $\Lambda(\mathbf{A}) = \Lambda(\mathbf{B})$. Die Inversion dieser Normalformberechnung gilt als schwierig, und ist mit dem Gitterproblem SIVP verwandt.

- **Verschlüsselung:** Zur Vereinfachung nehmen wir nun an die zu verschlüsselnde Nachricht sei ein Bitvektor $\mathbf{m} \in \{0,1\}^n$. Um $\mathbf{m}$ zu verschlüsseln zieht man sich einen zufälligen Vektor $\mathbf{e} \in \mathbb{Z}^n$ von einer *kurzen* Verteilung $\chi$. Man erhält nun ein Chiffrat $\mathbf{c}$ indem man $\mathbf{c} = \mathbf{As}+2\mathbf{e}+\mathbf{m}$ setzt. Die Skalierung des Fehlers mit dem Faktor 2 erfolgt mit dem Zweck es später bei der Entschlüsselung möglich zu machen den die Nachricht $\mathbf{m}$ wieder vom Fehler zu trennen, i.e. man kodiert $\mathbf{m}$ in das niedrigste Bit (LSB, least significant Bit) von $\check{vecm} = 2\mathbf{e} + \mathbf{m}$. Die Wahl von $\mathbf{s} \in \mathbb{Z}^n$ wurde im ursprünglichen GGH Verfahren als zufällig vorgeschlagen. Tatsächlich ist es aus Sicherheitsaspekten jedoch besser $\mathbf{s}$ deterministisch in Abhängigkeit von $\tilde{\mathbf{m}} = 2\mathbf{e} + \mathbf{m}$ zu wählen. Etwas spezifischer, mittels linearer Algebra lässt sich ein $\mathbf{s}^* \in \mathbb{Z}^n$ bestimmen, für welches $\mathbf{c} = \mathbf{As} + 2\mathbf{e} + \mathbf{m}$ ein *eindeutiger Restklassenvertreter* von $\tilde{\mathbf{m}}$ bezüglich der Basis $\mathbf{A}$ ist, gegeben dass $\tilde{\mathbf{m}}$ hinreichend kurz ist.

- **Entschlüsselung:** Zur Entschlüsselung verwendet man Kenntnis der *guten* Basis $\mathbf{B}$. Zu einer solchen Basis lässt sich beispielsweise eine *kurze Dualbasis* $\mathbf{T}$ berechnen, also eine Matrix $\mathbf{T}$ mit kurzen Einträgen für welche $\mathbf{T} \cdot \mathbf{B} = 0$ gilt. Damit lässt sich dann $\mathbf{c}$ wie folgt entschlüsseln. Man berechnet $\tilde{\mathbf{c}} = \mathbf{T} \cdot \mathbf{c}$ und es gilt
$$\tilde{\mathbf{c}} = \mathbf{T} \cdot \mathbf{c} = \mathbf{T}(\mathbf{As} + 2\mathbf{e} + \mathbf{m}) = \mathbf{T}(2\mathbf{e} + \mathbf{m}) = \mathbf{T} \cdot \tilde{\mathbf{m}}.$$
Ist nun $\tilde{\mathbf{m}} = 2\mathbf{e} + \mathbf{m}$ *hinreichend kurz*, speziell hinreichend kürzer als der kürzeste Vektor in $\Lambda = \Lambda(\mathbf{B})$, so lässt sich $\tilde{\mathbf{m}}$ durch
$$\tilde{\mathbf{m}} = \mathbf{T}^{-1} \cdot \tilde{\mathbf{c}}$$
berechnen, wobei $\mathbf{T}^{-1} \in \mathbb{Q}^{n \times n}$ die inverse Matrix von $\mathbf{T} \in \mathbb{Z}^{n \times n}$ über den rationalen Zahlen $\mathbb{Q}$ ist. Aus $\tilde{\mathbf{m}}$ erhält man nun sehr einfach $\mathbf{m}$ indem man $\tilde{\mathbf{m}}$ modulo 2 reduziert, i.e. das *Least Significant Bit* von $\tilde{\mathbf{m}}$ berechnet.

- **Additiver Homomorphismus**: Das GGH Verfahren weist eine *begrenzte additiv-homomorphe Eigenschaft* auf. Speziell, hat man zwei Chiffrate $\mathbf{c}_1$ und $\mathbf{c}_2$, so gilt
$$\begin{aligned}\mathbf{c}_3 = \mathbf{c}_1 + \mathbf{c}_2 &= \mathbf{As}_1 + 2\mathbf{e}_1 + \mathbf{m}_1 + \mathbf{As}_2 + 2\mathbf{e}_2 + \mathbf{m}_2 \\ &= \mathbf{A}(\mathbf{s}_1 + \mathbf{s}_2) + 2(\mathbf{e}_1 + \mathbf{e}_2) + \mathbf{m}_1 + \mathbf{m}_2.\end{aligned}$$
ist nun $\tilde{m}_3 = 2(\mathbf{e}_1 + \mathbf{e}_2) + \mathbf{m}_1 + \mathbf{m}_2$ hinreichend kürzer als der kürzeste Vektor in $\Lambda$, so ist $\mathbf{c}_3$ eine Verschlüsselung von $\mathbf{m}_3 = (\mathbf{m}_1 + \mathbf{m}_2) \mod 2$

- Diskussion: Die Wahl des Modulus 2 im GGH Verfahren ist nicht zwingend. Im Allgemeinen lässt sich jeder *hinreichend kleine* Modulus $q$ hier verwenden. Man bezeichnet $\tilde{\mathbf{m}}$ als *Fehlerterm*, und das GGH Verfahren codiert die Nachricht $\mathbf{m}$ in den Least-Significant-Bits des Fehlerterms.

Die Relevanz des GGH Verfahrens im Kontext von homomorpher Verschlüsselung besteht letztendlich darin, dass alle FHE Verfahren der ersten Generation Varianten dieses Verfahrens waren.

**Regev Verschlüsselung** Das Regev Verschlüsselungsverfahren [124] basiert auf der LWE Annahme und kann als LWE-Variante des Ajtai-Dwork Verfahrens angesehen werden. Die Beschreibung dieses Regev-Verfahren benötigt damit nur ein relativ geringes Maß an Gitterterminologie und lässt sich mittels modularer Arithmetik implementieren. Öffentlicher Parameter des Verfahrens ist ein (großer) Modulus $q \in \mathbb{Z}$. Im folgenden sind alle arithmetischen Operationen modulo $q$.

- **Schlüsselerzeugung**: Zur Schlüsselerzeugung generiert man sich zunächst eine zufällige Matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$. Weiter wird ein zufälliges $\mathbf{s} \in \mathbb{Z}_q^n$ gezogen und ein kurzes $\mathbf{e} \in \mathbb{Z}^m$ anhand einer kurzen Verteilung $\chi$ erzeugt. Man setzt nun $\mathbf{b} = \mathbf{sA} + \mathbf{e}$. Der Public-Key ist $\mathsf{pk} = (\mathbf{A}, \mathbf{b})$, wohingegen der Secret-Key $\mathsf{sk} = \mathbf{s}$ ist.

- **Verschlüsselung**: Verschlüsselt wird ein Bit $\mathbf{m} \in \{0,1\}$ folgendermaßen: Gegeben einen Public Key $\mathsf{pk} = (\mathbf{A}, \mathbf{b})$ generiert man sich ein gleichverteilt zufälliges $\mathbf{r} \in \{0,1\}^m$ und setzt $\mathbf{c}_0 = \mathbf{Ar}$ und $c_1 = \mathbf{b} \cdot \mathbf{r} + \frac{q}{2} \cdot \mathbf{m}$. Das Chiffrat ist $\mathbf{c} = (\mathbf{c}_0, c_1)$.
- **Entschlüsselung**: Entschlüsselt wird wie folgt: Gegeben einen Secret Key $\mathsf{sk} = \mathbf{s}$ und ein Chiffrat $\mathbf{c} = (\mathbf{c}_0, c_1)$ berechnet man zuerst $\mathbf{m}' = c_1 - \mathbf{s} \cdot \mathbf{c}_0$. Wenn $m'$ *nahe* an 0 liegt setzt man $\mathbf{m} = 0$, falls $\mathbf{m}'$ nahe an $q/2$ liegt setzt man $\mathbf{m} = 1$.

**Korrektheit** Die Korrektheit des Verfahrens, also dass Entschlüsselung wieder den ursprünglichen Klartext liefert kann folgendermaßen eingesehen werden. Setzt man $\mathbf{b} = \mathbf{sA} + \mathbf{e}$, $\mathbf{c}_0 = \mathbf{Ar}$ und $c_1 = \mathbf{br} + \frac{q}{2} \cdot \mathbf{m}$ in $\mathbf{m}' = c_1 - \mathbf{s} \cdot \mathbf{c}_0$ ein, so erhält man

$$\begin{aligned} \mathbf{m}' &= (\mathbf{sA} + \mathbf{e})\mathbf{r} + \frac{q}{2}\mathbf{m} - \mathbf{sAr} \\ &= \frac{q}{2}\mathbf{m} + \mathbf{er}. \end{aligned}$$

Da $\mathbf{r}$ ein binärer Vektor ist gilt $\|\mathbf{r}\| \leq \sqrt{m}$, also sind sowohl $\mathbf{e}$ als auch $\mathbf{r}$ sind in der euklid'schen Norm *kurz* [5].

sind, ist auch das innere Produkt $\mathbf{er}$ kurz. Man kann also $\mathbf{er}$ als einen kurzen Fehler betrachten, welcher den Term $q/2 \cdot \mathbf{m}$, der die Nachricht $\mathbf{m}$ codiert, stört. Da aber 0 und $q/2$ in $\mathbb{Z}_q$ maximal weit voneinander weg liegen, kann durch einen Abstandstest bzw. einer Rundung die ursprüngliche Nachricht dekodiert werden. Dementsprechend ist die Multiplikation mit $q/2$ eine *fehlerkorrigierende Codierung* gegen *kurze Fehler*.

**Sicherheit** Die IND-CPA Sicherheit des Regev-Verfahrens beruht auf dem LWE Problem und wird in etwa wie folgt argumentiert. Zunächst werden in einem Gedankenexperiment/Hybridexperiment die realen Public-Keys, bei welchen die Komponente $\mathbf{b}$ von der Form $\mathbf{b} = \mathbf{sA} + \mathbf{e}$ sind, durch *nicht-funktionale* Public-Keys ersetzt bei welchen $\mathbf{b}$ gleichverteilt zufällig gezogen wird. Ein derartiger Public-Key hat also keinen zugehörigen Secret-Key mehr. Für das IND-CPA Sicherheitsexperiment ist dies jedoch nicht von belang, da der Secret-Key hier nicht benötigt wird. Es lässt sich nun mittels der LWE Annahme argumentieren, dass kein effizienter Angreifer diese Ersetzung erkennen kann, da er ansonsten das LWE Problem lösen könnte. Anders formuliert bedeutet dies, dass unter der LWE Annahme funktionale Public-Keys von nicht-funktionalen Public-Keys nicht unterscheidbar sind.

Für einen derart abgewandelten Public-Key lässt sich nun mittels des Leftover-Hash-Lemmas bzw. eines Smoothings Lemmas argumentieren dass das Tupel $(\mathbf{Ar}, \mathbf{br})$ statistisch nahe an einer Gleichverteilung auf $\mathbb{Z}_q^n \times \mathbb{Z}_q$ liegt. Dementsprechend wirkt der Term $\mathbf{br}$ in $c_1 = \mathbf{br} + \frac{q}{2}\mathbf{m}$ wie ein One-Time-Pad, welcher die codierte Nachricht $\frac{q}{2} \cdot m$ *informations-theoretisch* versteckt.

## 3.2 Dual-Regev Verschlüsselung

Während bei der Regev Verschlüsselung Public-Keys mittels der LWE Annahme in einen nicht-funktionalen, aber informations-theoretisch sicheren Modus umgeschaltet werden können, ist die Idee der *Dual-Regev-Verschlüsselung* [71] die Rollen von Public-Key und Chiffrat in der Regev-Verschlüsselung zu vertauschen. Tatsächlich lässt sich das Dual-Regev-Verfahren als eine LWE-basierte Version des GGH-Verfahrens auffassen. Genau wie das Regev-Verfahren ist dieses Verfahren relativ zu einem Modulus $q$ definiert, welcher entweder Teil der öffentlichen Parameter oder des Public-Keys sein kann.

- **Schlüsselerzeugung**: Bei der Schlüsselerzeugung wird eine zufällige Matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ zusammen mit einer *kurzen Dualbasis* $\mathbf{T} \in \mathbb{Z}^{m \times m}$ gewählt, i.e. $\mathbf{T}$ hat kurze Einträge, ist invertierbar, und es gilt $\mathbf{A} \cdot \mathbf{T} = 0$. Zusätzlich wird noch eine echt zufällige Matrix $\mathbf{B} \in \mathbb{Z}_q^{n \times k}$ gewählt. Der Public-Key $\mathsf{pk}$ besteht aus den Matrizen $\mathbf{A}$ und $\mathbf{B}$, wohingegen der Secret-Key $\mathsf{sk}$ aus der Dualbasis $\mathbf{T}$ besteht.

[5] Kurz bedeutet in diesem Zusammenhang dass die Länge eines Vektors polynomiell vom Sicherheitsparameter abhängt, aber unabhängig vom Modulus $q$ ist

- **Verschlüsselung**: Verschlüsselung einer Nachricht $\mathbf{m} \in \{0,1\}^k$ erfolgt mittels folgenden Schritten. Zunächst wird eine gleichverteilt zufälliges $\mathbf{s} \in \mathbb{Z}_q^n$ gezogen sowie zwei *Fehlervektoren* $\mathbf{e}_0 \in \mathbb{Z}^m$ und $\mathbf{e}_1 \in \mathbb{Z}^k$ mittels einer kurzen Verteilung $\chi$. Man setzt nun $\mathbf{c}_0 = \mathbf{sA} + \mathbf{e}_0$ und $\mathbf{c}_1 = \mathbf{sB} + \mathbf{e}_1 + \frac{q}{2} \cdot \mathbf{m}$ und gibt das Chiffrat $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1)$ aus.
- **Entschlüsselung**: Entschlüsselung eines Chiffrates $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1)$ läuft folgendermaßen ab. Mit der Lattice-Trapdoor $\mathbf{T}$ berechnet man zunächst $\tilde{\mathbf{e}}_0 = \mathbf{c}_0\mathbf{T}$ (modulo $q$) und berechnet $\tilde{\mathbf{e}}_0 \in \mathbb{Z}^m$ dabei als *zentralen Restklassenvertreter*, i.e. alle Einträge von $\tilde{\mathbf{e}}_0$ sind im Bereich $[-q/2, q/2)$. Als nächstes berechnet man $\mathbf{e}_0 = \tilde{\mathbf{e}}_0 \cdot \mathbf{T}^{-1}$ (über $\mathbb{Z}$) und löst danach die Gleichung $\mathbf{c}_0 = \mathbf{sA} + \mathbf{e}_0$ mittels linearer Algebra über $\mathbb{Z}_q$ nach dem eindeutigen $\mathbf{s} \in \mathbb{Z}_q^n$ auf. Hat man dieses gefunden, so berechnet man $\tilde{\mathbf{m}} = \mathbf{c}_1 - \mathbf{sB} \in \mathbb{Z}_q^k$. Aus $\tilde{\mathbf{m}}$ erhält man $\mathbf{m} \in \{0,1\}^k$ indem man komponentenweise zu 0 oder $q/2$ rundet. Das heißt für jeden Index $i \in \{1, \ldots, k\}$ setzt man $m_i = 0$ falls $\tilde{m}_i$ näher an 0 als an $q/2$ liegt, wohingegen man $m_i = 1$ setzt falls $\tilde{m}_i$ näher an $q/2$ als an 0 liegt.
- **Additiver Homomorphismus**: Auch das Dual-Regev Verfahren ist additiv homomorph. Gegeben zwei Chiffrate $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1)$ und $\mathbf{c}' = (\mathbf{c}'_0, \mathbf{c}'_1)$ kann man $\mathbf{c}'' = (\mathbf{c}_0 + \mathbf{c}'_0, \mathbf{c}_1 + \mathbf{c}'_1)$ berechnen. Verschlüsselt $\mathbf{c}$ eine Nachricht $\mathbf{m} \in \{0,1\}^k$ und $\mathbf{c}'$ eine Nachricht $\mathbf{m}' \in \{0,1\}^k$ so verschlüsselt $\mathbf{c}''$ die Nachricht $\mathbf{m} + \mathbf{m}' \mod 2$.

**Korrektheit** Korrektheit wird nun ähnlich wie beim GGH Verfahren argumentiert: Für den bei der Entschlüsselung berechneten Term $\tilde{\mathbf{e}}_0$ gilt dass

$$\tilde{\mathbf{e}}_0 = \mathbf{c}_0\mathbf{T} = (\mathbf{sA} + \mathbf{e}_0)\mathbf{T} = \mathbf{s}\underbrace{\mathbf{AT}}_{=0} + \mathbf{e}_0\mathbf{T} = \mathbf{e}_0\mathbf{T}.$$

Da sowohl $\mathbf{e}$ als auch $\mathbf{T}$ *kurz* sind, ist $\tilde{\mathbf{e}}_0$ identisch zu $\mathbf{e}_0\mathbf{T}$ *über den ganzen Zahlen* $\mathbb{Z}$, obwohl die Berechnung $\tilde{\mathbf{e}}_0 = \mathbf{c}_0\mathbf{T}$ über $\mathbb{Z}_q$ durchgeführt wurde. Damit gilt dann $\mathbf{e}_0 = \tilde{\mathbf{e}}_0 \cdot \mathbf{T}^{-1}$. Man erhält somit durch Auflösen der Gleichung $\mathbf{c}_0 = \mathbf{sA} + \mathbf{e}_0$ eine eindeutige Lösung $\mathbf{s}$, da $\mathbf{e}_0$ und $\mathbf{s}$ nun bereits bekannt sind und $\mathbf{A}$ mehr Spalten als Zeilen hat, i.e. $m > n$, womit dieses Gleichungssystem überbestimmt ist. Hat man nun $\mathbf{s}$, so gilt

$$\tilde{\mathbf{m}} = \mathbf{c}_1 - \mathbf{sB} = \mathbf{sB} + \mathbf{e}_1 + \frac{q}{2}\mathbf{m} - \mathbf{sB} = \frac{q}{2}\mathbf{m} + \mathbf{e}_1.$$

Da der Fehler $\mathbf{e}_1$ kurz ist, lässt sich $\mathbf{m}$ nun komponentenweise aus $\tilde{\mathbf{m}}$ bestimmen.

Die Korrektheit der additiv homomorpher Operationen lässt sich ähnlich argumentieren. Gegeben zwei Chiffrate $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1)$ und $\mathbf{c}' = (\mathbf{c}'_0, \mathbf{c}'_1)$ bezüglich des gleichen Public-Key $\mathsf{pk} = (\mathbf{A}, \mathbf{B})$ mit

$$\begin{aligned}
\mathbf{c}_0 &= \mathbf{sA} + \mathbf{e}_0 \\
\mathbf{c}_1 &= \mathbf{sB} + \mathbf{e}_1 + \frac{q}{2}\mathbf{m} \\
\mathbf{c}'_0 &= \mathbf{s}'\mathbf{A} + \mathbf{e}'_0 \\
\mathbf{c}'_1 &= \mathbf{s}'\mathbf{B} + \mathbf{e}'_1 + \frac{q}{2}\mathbf{m}',
\end{aligned}$$

so gilt

$$\mathbf{c}''_0 = \mathbf{c}_0 + \mathbf{c}'_0 = \mathbf{sA} + \mathbf{e}_0 + \mathbf{s}'\mathbf{A} + \mathbf{e}'_0 = (\mathbf{s} + \mathbf{s}')\mathbf{A} + \mathbf{e}_0 + \mathbf{e}'_0,$$

und

$$\mathbf{c}''_1 = \mathbf{c}_1 + \mathbf{c}'_1 = \mathbf{sB} + \mathbf{e}_1 + \frac{q}{2}\mathbf{m} + \mathbf{s}'\mathbf{B} + \mathbf{e}'_1 + \frac{q}{2}\mathbf{m}' = (\mathbf{s} + \mathbf{s}')\mathbf{B} + \mathbf{e}_1 + \mathbf{e}'_1 + \frac{q}{2}(\mathbf{m} + \mathbf{m}'),$$

i.e. $\mathbf{c}''$ ist nun ein Chiffrat von $\mathbf{m} + \mathbf{m}'$ (modulo 2) für $\mathbf{s}'' = \mathbf{s} + \mathbf{s}'$, $\mathbf{e}''_0 = \mathbf{e}_0 + \mathbf{e}'_0$ und $\mathbf{e}''_1 = \mathbf{e}_1 + \mathbf{e}'_1$. Da $\mathbf{e}_0$, $\mathbf{e}'_0$, $\mathbf{e}_1$, $\mathbf{e}'_1$ kurz sind, sind auch $\mathbf{e}''_0$ und $\mathbf{e}''_1$ kurz.

**Sicherheit** Sicherheit des Dual-Regev-Verfahrens lässt sich nun wie folgt argumentieren. Zunächst stellt man sich wieder eine Gedankenexperiment vor in welchem der Public-Key *ohne* zugehörigen Secret-Key erzeugt wird. Man wählt also die Matrix $\mathbf{A}$ gleichverteilt zufällig *ohne* Kenntnis einer kurzen Dualbasis $\mathbf{T}$. Die nun in einem Chiffrat $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1)$ mit $\mathbf{c}_0 = \mathbf{sA} + \mathbf{e}_0$ und $\mathbf{c}_1 = \mathbf{sB} + \mathbf{e}_1 + \frac{q}{2}\mathbf{m}$ auftretenden Terme $\mathbf{sA} + \mathbf{e}_0$ und $\mathbf{sB} + \mathbf{e}_1$ sind LWE-Samples. Unter der LWE-Annahme lassen sich diese nun durch zufällige Terme $\mathbf{u}_0$ und $\mathbf{u}_1$ ersetzen, das modifizierte Chiffrat hat also die Form $\mathbf{c}_0 = \mathbf{u}_0$, $\mathbf{c}_1 = \mathbf{u}_1 + \frac{q}{2}\mathbf{m}$. Hier hat Addition mit $\mathbf{u}_1$ wieder den Effekt eines One-Time-Pads, das bedeutet dass $\mathbf{c}_1$ die codierte Nachricht $\frac{q}{2}\mathbf{m}$ informations-theoretisch versteckt.

# 4 Somewhat Homomorphic Encryption

Die in Kapitel 3 vorgestellten gitterbasierten Verschlüsselungsverfahren weisen alle eine additiv homomorphe Eigenschaft auf. Wir werden nun darstellen wie sich die zugrundeliegenden Ideen dieser Verfahren zu sogenannten *Somewhat Homomorphic Encryption* (SHE) Verfahren ausbauen lassen. Derartige Verfahren unterstützen nur eine begrenzte Zahl homomorpher Operation, stellen aber den kritischen Baustein bei der Konstruktion von Fully-Homomorphic-Encryption Schemata dar. Konzeptionell ist die Konstruktion eines FHE Verfahrens aus einem SHE Verfahren unabhängig von dem zugrundeliegenden SHE Verfahren, daher werden wir uns bei die Diskussion der verschiedenen Generationen von SHE/FHE Verfahren ausschließlich auf SHE Verfahren konzentrieren.

Rückblickend werden SHE und FHE Verfahren in drei Generationen eingeteilt, wobei jeder Generationsübergang einem Innovationssprung und einer konzeptionellen Vereinfachung in der Konstruktion von FHE Verfahren entspricht.

## 4.1 Verfahren der Ersten Generation

Die erste Generation von SHE Verfahren geht auf Gentry's [66] ursprüngliche Konstruktion zurück. Die Grundidee hierzu ist rückblickend gesehen relativ einfach. Basis für dieses Verfahren stellt das GGH Verschlüsselungsverfahren dar, welches bereits additiv homomorph ist. Durch eine geschickte Wahl des zugrundeliegenden Gitters $\Lambda$ lässt sich dieses Verfahren mit einer begrenzten multiplikativ homomorphen Eigenschaft ausstatten. Hauptidee hierbei ist das Gitter $\Lambda$ als zufälliges *Idealgitter* zu wählen. Während Gitter unter Addition abgeschlossen sind, so sind Idealgitter auch unter Multiplikation abgeschlossen. Etwas genauer, falls $\Lambda$ ein Gitter ist und $\mathbf{x}, \mathbf{y} \in \Lambda$, so gilt immer auch $a \cdot \mathbf{x} + b \cdot \mathbf{y} \in \Lambda$ für beliebige ganze Zahlen $a, b \in \mathbb{Z}$. Ist $\Lambda$ hingegen zusätzlich ein Idealgitter, so ist auch eine Multiplikation $*$ auf $\Lambda$ definiert, welches assoziativ, distributiv und kommutativ ist. Ferner gilt für derartige Idealgitter, dass die Multiplikation $\mathbf{e} * \mathbf{d}$ zweier kurzer Elemente $\mathbf{e}$ und $\mathbf{d}$ auch wieder kurz ist.

Gentry's SHE Verfahren ist daher tatsächlich bis auf den Schlüsselerzeugungsalgorithmus und einen zusätzlichen homomorphen Multiplikationsalgorithmus identisch zum GGH Verfahren.

- **Schlüsselerzeugung**: Der Schlüsselerzeugungsalgorithmus generiert ein *zufälliges* Idealgitter $\Lambda$ mit Basis $\mathbf{B}$ zusammen mit einer kurzen Dualbasis $\mathbf{T}$, d.h. es gilt $\mathbf{T} \cdot \mathbf{B} = 0$.
- **Homomorphe Multiplikation** Um zwei Chiffrate $\mathbf{c} = \mathbf{As} + 2\mathbf{e} + \mathbf{m}$ und $\mathbf{c}' = \mathbf{As}' + 2\mathbf{e}' + \mathbf{m}'$ zu multiplizieren, berechnet man $\mathbf{c}'' = \mathbf{c} * \mathbf{c}'$. Es gilt dann

$$\begin{aligned}\mathbf{c}'' = \mathbf{c} * \mathbf{c}' &= (\mathbf{As} + 2\mathbf{e} + \mathbf{m}) * (\mathbf{As}' + 2\mathbf{e}' + \mathbf{m}') \\ &= (\mathbf{As}) * (\mathbf{As}') + (\mathbf{As}) * 2\mathbf{e}' + (\mathbf{As}') * 2\mathbf{e} + 2(2\mathbf{e} * \mathbf{e}' + \mathbf{e} * \mathbf{m}' + \mathbf{e}' * \mathbf{m}) + \mathbf{m} * \mathbf{m}'.\end{aligned}$$

  Da $\mathbf{As}$ und $\mathbf{As}'$ im Idealgitter $\Lambda$ liegen, liegen auch $(\mathbf{As}) * (\mathbf{As}')$, $(\mathbf{As}) * 2\mathbf{e}'$ und $(\mathbf{As}') * 2\mathbf{e}$ im Idealgitter $\Lambda$, somit auch $(\mathbf{As}) * (\mathbf{As}') + (\mathbf{As}) * 2\mathbf{e}' + (\mathbf{As}') * 2\mathbf{e}$. Es existiert also ein $\mathbf{s}''$, sodass

$$\mathbf{As}'' = (\mathbf{As}) * (\mathbf{As}') + (\mathbf{As}) * 2\mathbf{e}' + (\mathbf{As}') * 2\mathbf{e}.$$

  Ferner gilt, dass weil $\mathbf{e}$, $\mathbf{e}'$, $\mathbf{m}$ und $\mathbf{m}'$ kurz sind, auch $2\mathbf{e} * \mathbf{e}'$, $\mathbf{e} * \mathbf{m}'$ und $\mathbf{e}' * \mathbf{m}$ und damit auch

$$\mathbf{e}'' = 2\mathbf{e} * \mathbf{e}' + \mathbf{e} * \mathbf{m}' + \mathbf{e}' * \mathbf{m}$$

  kurz ist. Damit ist also

$$\mathbf{c}'' = \mathbf{As}'' + 2\mathbf{e}'' + \mathbf{m} * \mathbf{m}'$$

  ein wohl-geformtes Chiffrat von $\mathbf{m} * \mathbf{m}'$, welches zum Produkt $\mathbf{m} * \mathbf{m}'$ entschlüsselt.

Auf dieser Abstraktionsebene ist noch nicht definiert wie genau die Klartexte in $\mathbf{m}$ codiert werden. In Gentry's ursprünglichem Verfahren [66] war der Vorschlag schlicht, ein Bit $m$ mittels $m \cdot \mathbf{v}$ zu kodieren, wobei $\mathbf{v}$ ein kurzer Vektor mit der Eigenschaft $\mathbf{v} * \mathbf{v} = \mathbf{v}$, $\mathbf{v}$ ist also das multiplikativ neutrale Element im Ring $\mathsf{R}$.

Hinsichtlich der Ciphertextrate ist dieses Verfahren recht ineffizient, ein Ciphertext besteht aus einem Gittervektor, verschlüsselt letztendlich aber nur ein einziges Bit.

### 4.1.1 Bootstrapping

Eine der Hauptinnovationen von Gentry's Arbeit [66] liegt in der Einführung einer Technik namens *Bootstrapping*. Wie oben erklärt *degradieren* homomorphe Operationen die Qualität von Chiffraten indem sie zu einem Fehlerwachstum führen. Die Idee des Bootstrapping ist es, den Homomorphismus eines Verfahrens selbst zu verwenden um Chiffrate *aufzufrischen* und damit Fehler zu korrigieren.

Die Grundidee hierbei ist es dem öffentlichen Schlüssel eines SHE Verfahrens eine Verschlüsselung $\mathbf{c}^* = \mathsf{Enc}(\mathsf{pk}, \mathsf{sk})$ des geheimen Schlüssels $\mathsf{sk}$ hinzuzufügen. Hiermit lassen sich wie folgt Chiffrate $\mathbf{c}$ auffrischen. Zunächst wir das Chiffrat $\mathbf{c}$ in den Entschlüsselungsalgorithmus $\mathsf{Dec}$ fest-verdrahtet, und die daraus resultierende Funktion $\mathsf{Dec}(\cdot, \mathbf{c})$ homomorph auf dem Chiffrat $\mathbf{c}^* = \mathsf{Enc}(\mathsf{pk}, \mathsf{sk})$ ausgewertet. Dies liefert ein Chiffrat

$$\mathbf{c}' = \mathsf{Eval}(\mathsf{pk}, \mathsf{Dec}(\cdot, \mathbf{c}), \mathbf{c}^*),$$

welches aufgrund der homomorphen Korrektheit des FHE Verfahrens eine Verschlüsselung von $\mathsf{Dec}(\mathsf{sk}, \mathbf{c}) = m$ ist. Ist der Entschlüsselungsschaltkreis $\mathsf{Dec}$ so beschaffen, dass er selbst nur wenige neue Fehler erzeugt, so hat diese Prozedur den Fehleranteil in $\mathbf{c}'$ im Vergleich zu $\mathbf{c}$ verbessert.

Bootstrapping wie hier beschrieben benötigt stärkere Sicherheitsannahmen, nämlich dass das zugrundeliegende SHE Verfahren zirkulär sicher ist, also selbst dann noch sicher wenn eine Verschlüsselung des geheimen Schlüssels veröffentlicht wird. Diese Zusatzannahme lässt sich vermeiden, wenn stattdessen eine Kette von unabhängigen Schlüsseln verwendet wird, man spricht dann von *Levelled Homomorphic Encryption* [23]. Dies hat jedoch den Nachteil, dass nun die Größe des öffentlichen Schlüssels mit der maximalen Tiefe bzw. Laufzeit von homomorph auswertbaren Programmen skaliert.

**Weitere Verfahren der ersten Generation** Smart und Vercauteren [134] haben eine Variante von Gentry's Schema vorgeschlagen bei welcher mehrere Nachrichtenbits in ein Gitterelement verschlüsselt werden können. Grundidee ist dabei die Ringstruktur von $\mathsf{R}$ unter Verwendung des Chinesischen Restsatzes effizienter auszunutzen.

## 4.2 Verfahren der Zweiten Generation

Bei Verfahren der ersten Generation ist die multiplikative Struktur des zugrundeliegenden Idealgitters essentiell, da hierüber die homomorphe Multiplikation auf Chiffraten ermöglicht wird.

Bei Verfahren der zweiten Generation, angefangen mit dem Verfahren von Brakerski und Vaikuntanathan [27, 25], liegt eine diametral andere Idee zugrunde. Hauptidee dieser Verfahren ist, dass sich additiv bzw. linear homomorphe Verschlüsselungsverfahren, bei welchen die Entschlüsselungsoperation eine näherungsweise lineare Operation ist, mittels geschickter homomorpher Entschlüsselung eine homomorphe Multiplikation implementieren können. Ausgangspunkt ist folgende Beobachtung, welche wir hier am Beispiel des Regev Verfahrens mit Nachrichtencodierung im Least-Significant-Bit darstellen.

Zur Erinnerung, bei diesem Verfahren ist ein Public-Key von der Form $\mathsf{pk} = (\mathbf{A}, \mathbf{b})$ wobei $\mathbf{b} = \mathbf{sA} + 2\mathbf{e}$, und ein Chiffrat hat die Form $\mathbf{c} = (\mathbf{c}_0, c_1)$ wobei $\mathbf{c}_0 = \mathbf{Ar}$ und $c_1 = \mathbf{br} + m$, wobei $\mathbf{r} \in \{0,1\}^m$ ein zufällig gewählter Binärvektor ist. Ein derartiges Chiffrat lässt sich *näherungsweise* mit der folgenden linearen Operation entschlüsseln. Berechnet man $\tilde{m} = c_1 - \mathbf{sc}_0$ so gilt

$$\tilde{m} = c_1 - \mathbf{sc}_0 = \mathbf{br} + m - \mathbf{sAr} = (\mathbf{sA} + 2\mathbf{e})\mathbf{r} + m - \mathbf{sAr} = m + 2\mathbf{er},$$

wobei wir beobachten, dass $2\mathbf{er}$ kurz ist, da $\mathbf{e}$ und $\mathbf{r}$ jeweils kurz sind. Weiter beobachten wir, dass sich die Nachricht $m$ im Least-Significant-Bit von $\tilde{m}$ befindet.

Man nehme nun zwei Chiffrate $\mathbf{c} = (\mathbf{c}_0, c_1)$ und $\mathbf{c}' = (\mathbf{c}'_0, c'_1)$, wobei $\mathbf{c}$ eine Nachricht $m$ verschlüsselt und $\mathbf{c}'$ eine Nachricht $m'$. Dann gilt mittels obiger Beobachtung

$$\tilde{\mathbf{m}} = c_1 - \mathbf{sc}_0 = m + 2\mathbf{er}$$
$$\tilde{\mathbf{m}}' = c'_1 - \mathbf{sc}'_0 = m' + 2\mathbf{er}',$$

dass wenn man das Produkt von $\tilde{\mathbf{m}}$ und $\tilde{\mathbf{m}}'$ über den ganzen Zahlen $\mathbb{Z}$ berechnet, dass *einerseits*

$$\tilde{\mathbf{m}} \cdot \tilde{\mathbf{m}}' = m \cdot m' + 2(m\mathbf{er}' + m'\mathbf{er} + 2(\mathbf{er}) \cdot (\mathbf{er}')) = m \cdot m' + 2e^*,$$

wobei $e^* = m\mathbf{er}' + m'\mathbf{er} + 2(\mathbf{er}) \cdot (\mathbf{er}')$ kurz ist. *Andererseits* gilt

$$\begin{aligned}\tilde{\mathbf{m}} \cdot \tilde{\mathbf{m}}' &= (c_1 - \mathbf{s}\mathbf{c}_0) \cdot (c_1' - \mathbf{s}\mathbf{c}_0') \\ &= c_1 \cdot c_1' - \mathbf{s}(c_1'\mathbf{c}_0 + c_1\mathbf{c}_0') + (\mathbf{s}\mathbf{c}_0) \cdot (\mathbf{s}\mathbf{c}_0').\end{aligned}$$

Der letzte Ausdruck, das Produkt $(\mathbf{s}\mathbf{c}_0) \cdot (\mathbf{s}\mathbf{c}_0')$, lässt sich als inneres Produkt zweier *Tensorprodukte* ausdrücken, d.h. es gilt

$$(\mathbf{s}\mathbf{c}_0) \cdot (\mathbf{s}\mathbf{c}_0') = (\mathbf{s} \otimes \mathbf{s})(\mathbf{c}_0 \otimes \mathbf{c}_0').$$

Wir erhalten also, dass

$$c_1 \cdot c_1' + \mathbf{s}(-c_1'\mathbf{c}_0 - c_1\mathbf{c}_0') + (\mathbf{s} \otimes \mathbf{s})(\mathbf{c}_0 \otimes \mathbf{c}_0') = m \cdot m' + 2e^*.$$

Dies lässt sich nun konzeptionell noch einen Schritt weiter vereinfachen. Setzt man $\hat{\mathbf{s}} \in \mathbb{Z}_q^{n+n^2}$ auf $\hat{\mathbf{s}} = (\mathbf{s}, \mathbf{s} \otimes \mathbf{s})$ und $\hat{\mathbf{c}}_0 \in \mathbb{Z}_q^{n+n^2}$ auf $\hat{\mathbf{c}}_0 = (-c_1'\mathbf{c}_0 - c_1\mathbf{c}_0', \mathbf{c}_0 \otimes \mathbf{c}_0')$, so können wir $\mathbf{s}(-c_1'\mathbf{c}_0 - c_1\mathbf{c}_0') + (\mathbf{s} \otimes \mathbf{s})(\mathbf{c}_0 \otimes \mathbf{c}_0')$ als inneres Produkt

$$\mathbf{s}(-c_1'\mathbf{c}_0 - c_1\mathbf{c}_0') + (\mathbf{s} \otimes \mathbf{s})(\mathbf{c}_0 \otimes \mathbf{c}_0') = \hat{\mathbf{s}}\hat{\mathbf{c}}_0$$

ausdrücken. Damit gilt nun

$$c_1 \cdot c_1' - \hat{\mathbf{s}}\hat{\mathbf{c}}_0 = m \cdot m' + 2e^*.$$

Dies lässt sich nun so interpretieren, dass wenn man das *expandierte Regev Chiffrat* $\hat{\mathbf{c}} = (\hat{\mathbf{c}}_0, c_1 \cdot c_1')$ mit dem *expandierten Regev Secret-Key* $\hat{\mathbf{s}}$ entschlüsselt, so erhält man (näherungsweise) $m \cdot m'$.

Mit dieser Idee allein ist scheinbar noch nichts gewonnen, da das neue Chiffrat $\hat{\mathbf{c}}$ weitaus größer ist als die beiden ursprünglichen Chiffrate $\mathbf{c}$ und $\mathbf{c}'$, es hat Länge $n + n^2$ statt $n$. Was also benötigt wird ist ein Mechanismus ein derartiges Chiffrat wieder auf Länge $n$ zu schrumpfen.

Die Idee hierzu ist eine Technik welche Brakerski und Vaikuntanathan als *Key-Switching* bezeichnen. Die Idee dabei ist, die näherungsweise Entschlüsselung von $\hat{\mathbf{c}}$ als Additionen auszudrücken und diese *homomorph* auf einer Verschlüsselung von $\hat{\mathbf{s}}$ auszuführen. Zunächst einmal problematisch hierbei ist, dass zur näherungsweisen Entschlüsselung von $\hat{\mathbf{c}}$ das innere Produkt $\hat{\mathbf{s}} \cdot \hat{\mathbf{c}}_0$ berechnet werden muss, welche statt Additionen auch Multiplikationen mit öffentlich bekannten Werten (sogenannte Skalarmultiplikationen) benötigt. Die Hauptidee von [27, 25] Skalarmultiplikationen mittels lediglich Additionen zu realisieren besteht darin eine *Binärzerlegung* von $\hat{\mathbf{c}}_0$ vorzunehmen. Speziell bedeutet dies, dass wir einerseits $\hat{\mathbf{c}}_0 \in \mathbb{Z}_q^{n+n^2}$ in einen längeren Binärvektor $\bar{\mathbf{c}}_0 \in \{0,1\}^{(n+n^2)\log(q)}$ umkodieren können, und andererseits $\hat{\mathbf{s}}$ in ein $\bar{\mathbf{s}} \in \mathbb{Z}_q^{(n+n^2)\log(q)}$ umkodieren können, sodass der Wert des inneren Produktes der codierten Werte gleich dem Wert des inneren Produktes der unkodierten Werte ist:

$$\hat{\mathbf{c}}_0 \cdot \hat{\mathbf{s}} = \bar{\mathbf{c}}_0 \cdot \bar{\mathbf{s}}.$$

Der Vorteil hierbei ist nun, dass sich das innere Produkt auf der rechten Seite lediglich mit Additionen realisieren lässt, da $\bar{\mathbf{c}}_0$ ein Binärvektor ist, und das innere Produkt $\bar{\mathbf{c}}_0 \cdot \bar{\mathbf{s}}$ sich dementsprechend als Summe von Komponenten von $\bar{\mathbf{s}}$ darstellen lässt. Sei nun $t = (n + n^2)\log(q)$ und $\bar{\mathbf{s}} = (z_1, \ldots, z_t) \in \mathbb{Z}_q^t$.

Um das Chiffrat $\hat{\mathbf{c}}$ zu *schrumpfen* benötigt man nun *unkodierte* Regev Verschlüsselungen der Werte $z_1, \ldots, z_t$, welche sich aus dem Secret-Key $\mathbf{s}$ berechnen lassen. Nehmen wir nun an, dass derartige Chiffrate $\mathbf{d}_1, \ldots, \mathbf{d}_t$ mit $\mathbf{d}_i = (\mathbf{d}_{i,0}, d_{i,1})$ mit

$$\mathbf{d}_{i,0} = \mathbf{A}\mathbf{r}_i \tag{1}$$

$$d_{i,1} = \mathbf{b}\mathbf{r}_i + z_i \tag{2}$$

gegeben sind. Dann lässt sich aus den *öffentlich bekannten Chiffraten* $\hat{\mathbf{c}}$ und $\mathbf{d}_1, \ldots, \mathbf{d}_t$ allein mittels additiv homomorpher Operationen folgendes Chiffrat $\mathbf{c}^* = (\mathbf{c}_0^*, c_1^*)$ berechnen. Sei dazu $\bar{\mathbf{c}}_0 = (\bar{c}_1, \ldots, \bar{c}_t)$.

$$\begin{aligned}\mathbf{c}_0^* &= \sum_{i=1}^{t} \bar{c}_i \mathbf{d}_{i,0} \\ c_1^* &= \sum_{i=1}^{t} \bar{c}_i d_{i,1}.\end{aligned}$$

Zunächst ist zu beobachten, dass $\mathbf{c}^* \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ ist und damit syntaktisch die korrekte Form eines Regev Chiffrates hat. Wir behaupten nun, dass $\mathbf{c}^*$ unter dem Secret-Key $\mathbf{s}$ zu $m \cdot m'$ entschlüsselt. Dies lässt sich folgendermaßen einsehen. Es gilt, dass

$$\begin{aligned}\mathbf{c}_0^* &= \sum_{i=1}^{t} \bar{c}_i \mathbf{d}_{i,0} \\ &= \sum_{i=1}^{t} \bar{c}_i \mathbf{A}\mathbf{r}_i \\ &= \mathbf{A}(\sum_{i=1}^{t} \bar{c}_i \mathbf{r}_i) \\ &= \mathbf{A}\mathbf{r}^*,\end{aligned}$$

wobei $\mathbf{r}^* = \sum_{i=1}^{t} \bar{c}_i \mathbf{r}_i$ kurz ist, da die $\bar{c}_i$ und die $\mathbf{r}_i$ kurz sind. Weiter gilt, dass

$$\begin{aligned}c_1^* &= \sum_{i=1}^{t} \bar{c}_i d_{i,1} \\ &= \sum_{i=1}^{t} \bar{c}_i (\mathbf{b}\mathbf{r}_i + z_i) \\ &= \mathbf{b}(\sum_{i=1}^{t} \bar{c}_i \mathbf{r}_i) + \sum_{i=1}^{t} \bar{c}_i z_i \\ &= \mathbf{b}\mathbf{r}^* + \bar{\mathbf{c}} \cdot \bar{\mathbf{s}} \\ &= \mathbf{b}\mathbf{r}^* + \hat{\mathbf{c}} \cdot \hat{\mathbf{s}} \\ &= \mathbf{b}\mathbf{r}^* + m \cdot m' + 2e^*.\end{aligned}$$

Entschlüsselt man nun $\mathbf{c}^*$ mit $\mathbf{s}$, so berechnet man zunächst

$$\begin{aligned}\tilde{m}^* &= c_1^* - \mathbf{s}\mathbf{c}_0^* \\ &= \mathbf{b}\mathbf{r}^* + m \cdot m' + 2e^* - \mathbf{A}\mathbf{r}^* \\ &= \mathbf{s}\mathbf{A}\mathbf{r}^* + 2\mathbf{e}\mathbf{r}^* + m \cdot m' + 2e^* - \mathbf{A}\mathbf{r}^* \\ &= m \cdot m' + 2(\mathbf{e}\mathbf{r}^* + e^*).\end{aligned}$$

Ist der Fehlerterm $2(\mathbf{e}\mathbf{r}^* + e^*)$ nun hinreichend klein, d.h. kleiner als $q/2$, dann ist das Least-Significant-Bit von $\tilde{m}^*$ genau $m \cdot m'$. Durch die oben beschriebene Prozedur erhalten wir also gegeben die Eingabechiffrate $\mathbf{c}$ und $\mathbf{c}'$ *allein durch öffentlich durchführbare Operationen* ein neues Chiffrat $\mathbf{c}^*$ welches $m \cdot m'$ verschlüsselt.

Ein kritischer Punkt bezüglicher der Anzahl sequentieller homomorpher Operationen, welche dieses Verfahren erlaubt, ist das Wachstum des Fehlers bei sequentiellen Multiplikationen. Bei genauerer Betrachtung stellt sich heraus, dass $e^* = m\mathbf{e}\mathbf{r}' + m'\mathbf{e}\mathbf{r} + 2(\mathbf{e}\mathbf{r}) \cdot (\mathbf{e}\mathbf{r}')$ der kritische Term beim Fehlerwachstum ist. Hierbei ist

der am stärksten wachsende Term $2(\mathbf{er}) \cdot (\mathbf{er}')$. Um den *absoluten Einfluss* dieses Terms bei Fehlerwachstum zu begrenzen, schlägt [27, 25] eine *Modulus Reduktion* genannte Technik vor. Die Idee hierbei ist das Chiffrat $\mathbf{c}^*$ so von Modulus $q$ zu einem neuen Modulus $q'$ *herunterzuskalieren*, sodass der herunterskalierte Fehler $\frac{q'}{q}e^*$ eine ähnliche Größenordnung wie der ursprüngliche Fehler $\mathbf{e}$ hat. Während diese *Renormalisierung* einerseits zur Folge hat, dass die absolute Fehlergröße im Chiffrat nicht zunimmt, hat sie andererseits zur Folge, dass bei jeder Modulus-Reduktion der neue Modulus kleiner wird, die Größe des Ursprungsmodulus also eine obere Schranke für die maximale Anzahl sequentieller homomorpher Multiplikationen vorgibt.

## 4.3 Verfahren der Dritten Generation

Während Verfahren der zweiten Generation einem recht drastischen Fehlerwachstum unterliegen, schaffen es Verfahren der dritten Generation ein sehr moderates Fehlerwachstum zu realisieren. Was Verfahren der dritten Generation mit jenen der zweiten Generation gemein haben, ist die trickreiche Verwendung von Binärzerlegungen um das Fehlerwachstum zu begrenzen. In Verfahren der dritten Generation wird dies jedoch sehr viel wirkungsvoller realisiert.

Das erste Verfahren der dritten Generation, das Verfahren von Gentry, Sahai und Waters [72], nutzt Techniken, welche ursprünglich für die Konstruktion effizienter gitterbasierter CCA-Verschlüsselungsverfahren entwickelt wurden, speziell eine von Micciancio und Peikert [105] entwickelten Technik namens *Gadget-Matrix*. Verkürzt hat die Gadget-Matrix $\mathbf{G} \in \mathbb{Z}_q^{n \times n \log(q)}$ für $\mathbb{Z}_q$ folgende Eigenschaft: Zu jedem Vektor $\mathbf{x} \in \mathbb{Z}_q^n$ lässt sich *effizient* ein Binärvektor $\mathbf{w} \in \{0,1\}^{n \log(q)}$ finden, so dass

$$\mathbf{x} = \mathbf{G} \cdot \mathbf{w}$$

gilt. Die Mächtigkeit dieser Operation stammt daher, dass sie einen geometrisch *langen* Vektor $\mathbf{x}$ in einen geometrisch *kurzen* Vektor $\mathbf{w}$ (in der euklid'schen Norm) transformiert, aus welchem sich mittels einer einfachen linearen Operation, nämlich einer Matrix-Vektor Multiplikation der ursprüngliche Vektor $\mathbf{x}$ dekodieren lässt. Für die Berechnung des Vektors $\mathbf{w}$ hat sich hernach die Schreibweise $\mathbf{w} = \mathbf{G}^{-1}(\mathbf{x})$ etabliert. Zu beachten dabei ist, dass $\mathbf{G}^{-1}(\mathbf{x})$ *nicht* die Multiplikation mit einer inversen Matrix ist, sondern eine nichtlineare Operation. Gentry, Sahai und Waters [72] schlagen eine SHE Schema vor, welches eine einfache Variante der Regev Verschlüsselung ist, die sich letztlich nur durch zusätzliche Redundanz in der Codierung der verschlüsselten Nachricht auszeichnet. Das GSW Schema kann wie folgt beschrieben werden.

- **Schüsselerzeugung**: Die Schlüsselerzeugung ist identisch zur normalen Regev Verschlüsselung. D.h. der Public-Key besteht aus einer zufälligen Matrix $\mathbf{A}' \in \mathbb{Z}_q^{(n-1) \times n \log(q)}$ und einem Vektor $\mathbf{b} = \mathbf{s}'\mathbf{A}' + \mathbf{e}$, wobei $\mathbf{s}' \in \mathbb{Z}_q^{n-1}$ zufällig gezogen wird und $\mathbf{e} \in \mathbb{Z}^{n \log(q)}$ mittels einer kurzen Verteilung $\chi$ gezogen wird. Üblicherweise werden für dieses Schema $\mathbf{A}'$ und $\mathbf{b}$ in einer einzigen Matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n \log(q)}$ zusammengefasst, welche man dadurch erhält, dass man $\mathbf{b}$ als letzte Zeile zu $\mathbf{A}'$ hinzufügt. Public-Key ist nun die Matrix $\mathbf{A}$, wohingegen der Secret-Key aus einem Vektor $\mathbf{s} = (-\mathbf{s}', 1)$ besteht, d.h. man erhält $\mathbf{s}$ indem man $\mathbf{s}'$ mit $-1$ multipliziert und eine zusätzliche Komponente 1 an diesen Vektor anhängt. Mit dieser Notation erhält man die Relation $\mathbf{sA} = \mathbf{e}$.

- **Verschlüsselung**: Um ein Bit $m \in \{0,1\}$ zur verschlüsseln, zieht man sich eine zufällige Matrix $\mathbf{R} \in \{0,1\}^{n \log(q) \times n \log(q)}$ und setzt das Chiffrat auf eine Matrix $\mathbf{C} = \mathbf{AR} + m \cdot \mathbf{G}$. Man kann sich dieses Chiffrat alternativ nicht als Matrix sondern als *Vektor von Regev Chiffraten* vorstellen.

- **Entschlüsselung**: Um ein Chiffrat $\mathbf{C}$ zu entschlüsseln setzt man zunächst $\mathbf{v} = (0, \ldots, 0, \frac{q}{2}) \in \mathbb{Z}_q^n$, d.h. $\mathbf{v}$ ist ein Vektor welcher überall 0 ist, aber in der letzten Komponente $\frac{q}{2}$. Man berechnet nun $\tilde{m} = \mathbf{sCw}$ und setzt die Nachricht $m$ auf 0, wenn $\tilde{m}$ näher an 0 liegt, und setzt $m$ auf 1, wenn $\tilde{m}$ näher an $q/2$ liegt.

- **Homomorphe Additionen**: Gegeben zwei Chiffrate $\mathbf{C}$ und $\mathbf{C}'$ welche $m$ respektive $m'$ bezüglich des gleichen Public-Keys $\mathbf{A}$ verschlüsseln, lässt sich ein neues Chiffrat $\mathbf{C}'' = \mathbf{C} + \mathbf{C}'$ durch Matrizenaddition berechnen welches $m + m' (\mod 2)$ verschlüsselt.

- **Homomorphe Multiplikationen**: Gegeben zwei Chiffrate $\mathbf{C}$ und $\mathbf{C}'$, welche $m$ respektiv $m'$ bezüglich des gleichen Public-Keys $\mathbf{A}$ verschlüsseln, können wir ein neues Chiffrat $\mathbf{C}^*$ berechnen, welches $m \cdot m'$ verschlüsselt. Dazu berechnen wir $\mathbf{C}^* = \mathbf{C} \cdot \mathbf{G}^{-1}(\mathbf{C}')$, wobei $\mathbf{G}^{-1}(\mathbf{C}')$ die $\mathbf{G}^{-1}(\cdot)$ Operation auf jede Spalte von $\mathbf{C}'$ anwendet.

**Korrektheit** Wir betrachten zunächst wieder die Korrektheit des Verfahrens. Gegeben ein Chiffrat $\mathbf{C} = \mathbf{AR} + m\mathbf{G}$ gilt

$$\begin{aligned}
\tilde{m} &= \mathbf{sCG}^{-1}(\mathbf{v}) \\
&= \mathbf{s}(\mathbf{AR} + m\mathbf{G})\mathbf{G}^{-1}(\mathbf{v}) \\
&= \underbrace{\mathbf{sA}}_{=\mathbf{e}}\mathbf{R} + m \cdot \mathbf{s}\underbrace{\mathbf{GG}^{-1}(\mathbf{v})}_{=\mathbf{v}} \\
&= \mathbf{eR} + m \cdot \underbrace{\mathbf{sv}}_{q/2} \\
&= m\frac{q}{2} + \mathbf{eR}.
\end{aligned}$$

Hier ist einfach nachzurechnen, dass das innere Produkt $\mathbf{sv} = q/2$ ist, da die letzte Komponente von $\mathbf{s}$ 1 ist, alle bis auf die letzte Komponente von $\mathbf{v}$ 0 sind und die letzte Komponente von $\mathbf{v}$ genau $q/2$ ist. Da nun weiter $\mathbf{eR}$ ein kurzer Fehler ist, folgt die Korrektheit genau wie beim regulären Regev Verfahren.

**Homomorphe Korrektheit** Betrachten wir als nächstes die homomorphe Korrektheit von Additionen. Wenn $\mathbf{C} = \mathbf{AR} + m\mathbf{G}$ eine Verschlüsselung von $m$ und $\mathbf{C}' = \mathbf{AR}' + m'\mathbf{G}$ eine Verschlüsselung von $m'$ ist, so ist lässt sich $\mathbf{C}'' = \mathbf{C} + \mathbf{C}'$ ausdrücken als

$$\mathbf{C}'' = \mathbf{C} + \mathbf{C}' = \mathbf{AR} + m\mathbf{G} + \mathbf{AR}' + m'\mathbf{G} = \mathbf{A}(\mathbf{R} + \mathbf{R}') + (m + m')\mathbf{G} = \mathbf{AR}'' + (m + m')\mathbf{G},$$

wobei $\mathbf{R}'' = \mathbf{R} + \mathbf{R}'$ die zu $\mathbf{C}''$ gehörige Zufallswahl ist. Dementsprechend verdoppelt sich durch eine homomorphe Addition die Größe des Fehlers.

Bezüglich homomorpher Multiplikationen, sei nun $\mathbf{C}^* = \mathbf{C} \cdot \mathbf{G}^{-1}(\mathbf{C}')$ das Ergebnis einer homomorphen Multiplikation. Somit gilt

$$\begin{aligned}
\mathbf{C}^* &= \mathbf{CG}^{-1}('\mathbf{C}') \\
&= (\mathbf{AR} + m\mathbf{G})\mathbf{G}^{-1}(\mathbf{C}') \\
&= \mathbf{ARG}^{-1}(\mathbf{C}') + m\underbrace{\mathbf{GG}^{-1}(\mathbf{C}')}_{=\mathbf{C}'} \\
&= \mathbf{ARG}^{-1}(\mathbf{C}') + m\mathbf{C}' \\
&= \mathbf{ARG}^{-1}(\mathbf{C}') + m(\mathbf{AR}' + m\mathbf{G}) \\
&= \mathbf{A}(\mathbf{RG}^{-1}(\mathbf{C}') + m\mathbf{R}') + mm'\mathbf{G} \\
&= \mathbf{AR}^* + mm'\mathbf{G},
\end{aligned}$$

wobei nun $\mathbf{R}^* = \mathbf{RG}^{-1}(\mathbf{C}') + m\mathbf{R}'$ die zu $\mathbf{C}^*$ gehörige Zufallswahl ist. Da sowohl $\mathbf{R}$ als auch $\mathbf{G}^{-1}(\mathbf{C}')$ kurz sind, ist auch $\mathbf{RG}^{-1}(\mathbf{C}')$ kurz und damit $\mathbf{R}^*$.

## 4.4 Verfahren der "Vierten Generation"

Eine Reihe Verfahren, darunter FHEW [56], TFHE [44] und weitere werden manchmal als Verfahren der "vierten Generation" bezeichnet. Dies kennzeichnet allerdings weniger einen Innovationssprung wie bei den ersten drei Generationen, sondern stellt eher eine zu Vermarktungszwecken eingeführte Eigenbezeichnung dar. Aus wissenschaftlicher Sicht sind die Verbesserungen dieser Verfahren als inkrementell zu bewerten.

Während diese Verfahren keine fundamental neuen Techniken einführen, stechen sie eher dadurch hervor dass sie die Parameter und Konstanten bei bisherigen Verfahren so weit fein-tunen, dass diese zu akzeptablen Laufzeiten führen. Dabei findet manchmal ein Tradeoff zwischen Effizienz und Sicherheit statt, bei welchem bis an die Grenze der Garantien von gitterbasierten Annahmen gegangen wird.

**FHEW** FHEW (Portmanteau aus FHE und few) [56] schlägt einen verbesserten Bootstrapping Algorithmus vor. Idee hierbei ist es den Bootstrapping-Algorithmus zur Auffrischung von Chiffraten direkt in eine homomorphe NAND-Operation einzubauen. Hierbei wird auf feingranulare Techniken zurückgegriffen um einerseits das Fehlerwachstum durch die NAND-Operation selbst zu verringern.

Die andere Beobachtung auf welcher diese Verbesserung basiert ist, dass im wesentlichen bei allen derzeitigen FHE Verfahren Entschlüsselung aus einem inneren Produkt zwischen Chiffrat und geheimen Schlüssel mit darauf folgender Rundung besteht. Die Beobachtung hierbei ist, dass sich beim Bootstrapping im Falle einer NAND-Operation der erste Schritt, also die Berechnung des inneren Produkts direkt in die NAND-Berechnung einbauen lässt. Hierdurch ist dann nur noch der homomorphe Rundungsschritt wirklich ineffizient. Durch diese Modifikation war es möglich homomorphe NAND-Gatter mitsamt Bootstrapping in weniger als einer Sekunde auf einem handelsüblichen Rechner auszuwerten.

**TFHE** TFHE (für Torus-FHE) [44] stellt eine weitere Verbesserung von FHEW dar, welche insbesondere dadurch zustande kommt, dass für das skaleninvariante FHE Verfahren GSW [72] das homomorph ausgewertete innere Produkt zwischen dem Chiffrat und dem geheimen Schlüssel durch die spezielle Struktur von GSW weiter vereinfacht werden kann. Die Bezeichnung *Torus-FHE* kommt daher, dass die Autoren die übliche modulo $q$ Arithmetik von LWE-basierten Verfahren durch eine modulo 1 Arithmetik also durch Gruppenoperationen auf einem $n$-dimensionalen Torus ersetzen. Aus rein mathematischer Sicht stellt sich dies jedoch nur als Reskalierung dar und hat daher weder auf die Effizienz noch auf dies Sicherheit des Verfahrens tatsächlich Einfluss.

## 4.5 Sicherheit und Kryptanalyse

Im Wesentlichen wurde alle Verfahren der ersten Generation aufgrund von sogenannten *überdehnten* Gitterannahmen auf welchen diese basieren gebrochen [15]. Daher werden die diesen Verfahren zugrundeliegenden Ansätze in der neueren Forschung nicht mehr weiter verfolgt. Verfahren der zweiten und dritten Generation basieren ausnahmslos auf dem LWE Problem, ein struktureller Bruch dieser Verfahren würde daher einen kryptanalytischen Durchbruch mit weitreichenden Folgen darstellen. Bisher wurden nur konkrete Parametervorschläge einiger Verfahren als unsicher identifiziert, da hierbei schwache LWE Parameterwahlen zugrunde lagen [57, 59].

# 5 Fully-Homomorphic Encryption

Fully-Homomorphic Encryption (FHE) ist eine Erweiterung von Public-Key Encryption, die die Möglichkeit bietet, Berechnungen auf verschlüsselten Daten auszuführen. Ein FHE-Verfahren stellt zusätzlich zu den PKE-Algorithmen einen Evaluierungsalgorithmus Eval bereit, welcher bei Eingabe eines Schaltkreises (boolesch oder arithmetisch) und Chiffraten $c_1, \ldots, c_k$ den Schaltkreis $C$ homomorph auf die Inhalte der Chiffrate $c_1, \ldots, c_k$ anwendet und ein Chiffrat des Ergebnisses ausgibt. Unter anderem ermöglicht FHE es, gegeben Chiffraten von zwei Zahlen, ein Chiffrat von einer komplett beliebigen Funktion dieser zwei Zahlen zu produzieren; wählt man zum Beispiel die Addition als Funktion, enthält das entstehende Chiffrat die Summe der zwei Zahlen. Ein FHE-Verfahren ist korrekt, wenn die Wahrscheinlichkeit hoch ist, dass das Entschlüsseln eines homomorph ausgewerteten Chiffrats das gleiche Ergebnis liefert wie das Auswerten des Schaltkreises $C$ auf den Inhalten der Chiffrate. Genauer muss gelten, dass die Wahrscheinlichkeit

$$\Pr_{(\mathsf{pk},\mathsf{sk})\leftarrow \mathrm{Gen}(1^n)} \left[\mathrm{Dec}\left(\mathsf{sk}, \mathrm{Eval}\left(C, \mathrm{Enc}(\mathsf{pk}, m)\right)\right) = C(m)\right]$$

nahe bei 1 liegt, wobei die Wahrscheinlichkeit über die Wahl von $(\mathsf{pk}, \mathsf{sk})$ und den Zufall von Enc ist. Für formale Definitionen verweisen wir auf Anhang A.

**Sicherheit** Wie auch bei der Public-Key Encryption wird zum Definieren der Sicherheit eines Fully-Homomorphic Encryption Verfahrens Passive Sicherheit (IND-CPA) verwendet. Dies betrifft die Sicherheit der verschlüsselten Daten. Bei einigen Anwendungen ist es außerdem erforderlich, Sicherheitsgarantien für den Evaluierer zu definieren. Wenn die auf den Daten ausgewertete Funktion geheim bleiben soll muss Circuit Privacy garantiert sein. Dies bedeutet, dass nach dem Entschlüsseln des Ergebnis-Ciphertextes keine Informationen über den verwendeten Schaltkreis lernbar sind außer dem offensichtlichen Eingabe-Ausgabe Paar.

**CCA-Sicherheit ist für FHE nicht möglich.** Aufgrund der Homomorphie werden Chosen-Ciphertext-Angriffe ermöglicht. Gegeben ein Chiffrat $c$ kann ein CCA-Angreifer den Schaltkreis, der einfach eine 1 addiert, homomorph auf $c$ auswerten. Das Resultat ist ein Chiffrat der originalen Nachricht plus 1. Dieses Chiffrat ist nicht identisch zum Chiffrat $c$ und kann entsprechend dem Bedrohungsmodell entschlüsselt werden, was es ermöglicht den Inhalt von $c$ zu rekonstruieren.

**Bootstrapping.** Viele Instantiierungen von FHE haben das technische Problem, dass der Fehlerterm in den Chiffraten bei homomorphen Operationen (Multiplikationen) stark anwächst. Wird dieser Fehlerterm zu groß ist eine Entschlüsselung nicht mehr möglich und der Inhalt des Chiffrats ist verloren. Aus diesem Grund muss regelmäßig ein sogenannter Bootstrapping-Schritt durchgeführt werden, der den Fehlerterm reduziert. Konzeptionell wird auf einem Chiffrat eine Wiederverschlüsselung (Ent- und Verschlüsselung) durchgeführt. Dieser Schritt ist vergleichsweise zu allen anderen FHE-Operationen sehr rechenaufwendig.

## 5.1 Instantiierungen

Es gibt viele Instantiierungen von FHE. Diese können, je nach zugrundeliegendem Ansatz, in sogenannte "Generationen" gruppiert werden [139]. Für den praktischen Einsatz sind hauptsächlich die zweite [26, 24, 20, 69, 70] und dritte Generation [72, 28, 6, 56, 80, 44, 41] relevant. Abbildung 6 ordnet die verschiedenen Instantiierungen zeitlich ein.

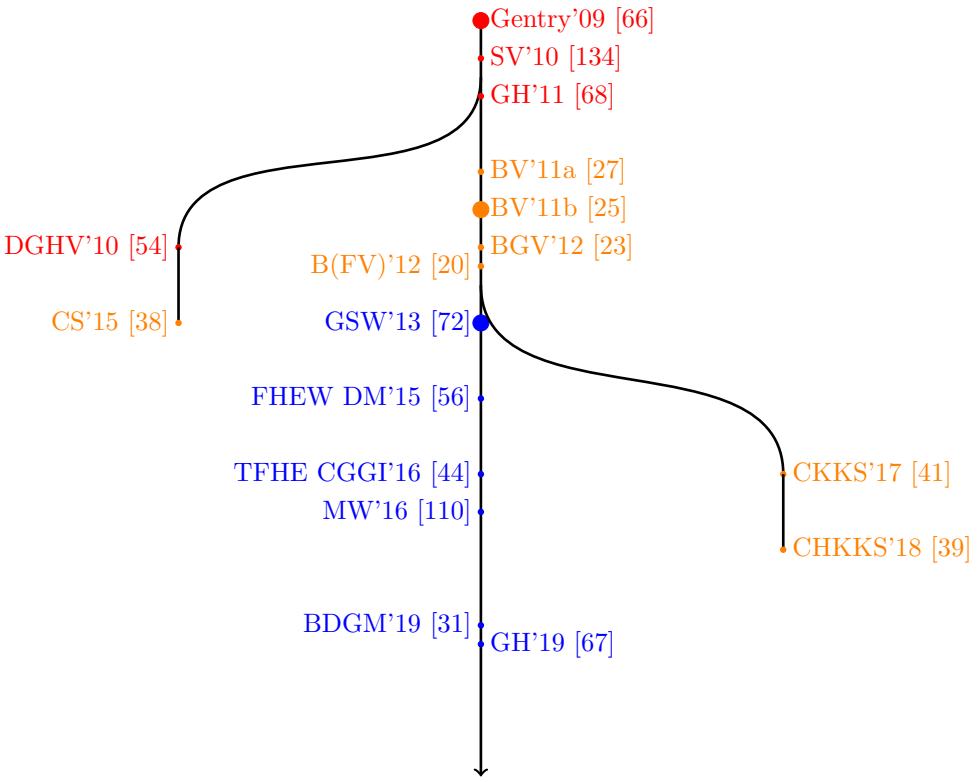



Abbildung 6: "Stammbaum" einiger FHE Verfahren in Generationen. Dabei sind Verfahren der ersten Generatio in rot, Verfahren der zweiten Generation in orange, und Verfahren der dritten Generation in blau.

Die jeweils von den Verfahren unterstützten Berechnungsmodelle sind in folgender Tabelle aufgeführt.

| Verfahren | Berechnungsmodell | Besonders geeignet für |
|---|---|---|
| BGV [24] | modulare Arithmetik | Ganzzahlarithmetik, Skalarmultiplikation |
| BFV [20, 61] | modulare Arithmetik | Ganzzahlarithmetik, Skalarmultiplikation |
| GSW [72] | boolesche Arithmetik | Zahlenvergleiche |
| FHEW [56] | boolesche Arithmetik | Zahlenvergleiche |
| TFHE [44, 46] | boolesche Arithmetik | Zahlenvergleiche |
| CKKS [41] | Gleitkommaarithmetik | Machine Learning, Polynomapproximation |

Das CKKS-Verfahren [41] hat die Besonderheit, dass es direkt mit Gleitkommazahlen umgehen kann, was in manchen Anwendungsszenarien große Vorteile bietet.

Die asymptotische Komplexität obiger Verfahren ist vergleichbar (Verfahren seit GSW haben eine leicht verbesserte asymptotische Komplexität). Die asymptotische Komplexität von GSW [72] ist für alle Operationen in $\tilde{O}(n)$ (ohne Bootstrapping) [77]. Die asymptotische Verbesserung gegenüber vorherigen Verfahren wurde durch die Vermeidung des teuren Re-Linearisierungsschritts ermöglicht. Da die Konstanten bei FHE extrem groß sind, ist für die Praxis allerdings nur die konkrete Effizienz relevant.

## 5.2 Forschungsgruppen und Implementierungen

Es ist eine Vielzahl von Open-Source Implementierungen für FHE verfügbar. Ein erster Schritt in Richtung einer globalen Standardisierung von FHE wurde in [3] unternommen.

**Concrete** Die Concrete-Bibliothek implementiert eine Variante [43] von TFHE [44, 46], was wiederum eine Variante von [72] ist. Die Bibliothek wird durch das französische Unternehmen Zama implementiert. Neben der eigentlichen FHE Implementierung bietet Zama auch ein Python Interface an, welches die Implementierung von Machine Learning Methoden erleichtern soll.

**TFHE** Die TFHE-Bibliothek [45] implementiert TFHE [44, 46]. Ausgehend von dieser Implementierung gibt es einige Bibliotheken, welche verschiedene Optimierungen implementieren.

**OpenFHE** Die OpenFHE-Bibliothek [116] implementiert BGV [24], BFV [20], FHEW (Ring-Variante von GSW [72]) [56], TFHE (Ring-Variante von GSW [72]) [44, 46], CKKS [41]. OpenFHE ist als Nachfolger des PALISADE-Projektes [117] entstanden. OpenFHE ist ein Open-Source Projekt zu dem jeder beitragen kann. Die Leitung übernimmt ein Team aus Forschern verschiedener Universitäten und Unternehmen.

**NuFHE** Die NuFHE-Bibliothek [114] implementiert TFHE [44, 46]. Die NuFHE-Bibliothek ist eine Erweiterung der cuFHE-Bibliothek[50], welche am Worcester Polytechnic Institute in Worcester, Massachusetts, USA entstanden ist.

**Microsoft Seal** Die SEAL-Bibliothek[106] von Microsoft implementiert BFV [20] und CKKS [41]. Außerdem wurde durch die chinesische Forschergruppe Alibaba Gemini Lab eine Implementierung von BGV hinzugefügt. Die Seal Bibliothek wurde verwendet, um im Microsoft Edge Browser den Password Monitor zu implementieren. Dies ist eine Anwendung der Private Set Intersection (siehe Abschnitt 9.2).

**Lattigo** Die Lattigo-Bibliothek [89, 109] implementiert sowohl BFV [20] als auch CKKS [41] inklusive CKKS-Bootstrapping. Die Bibliothek wurde zuerst durch eine an der Schweizer Universität EPFL ansässigen Gruppe implementiert und wird ab Version 3.0.0 durch das Schweizer Start-Up Tune Insight implementiert. Eines der Projekte des Unternehmens ist der Austausch sensibler Daten zwischen Krankenhäusern (LinkedIn post).

**HELib** Die HELib-Bibliothek [79] implementiert BGV [24] und CKKS [41]. Die Bibliothek wird durch das US-amerikanische Unternehmen IBM entwickelt. Eine brasilianische Forschergruppe hat mit Hilfe der Bibliothek Privacy-Preserving Machine Learning Methoden (siehe Abschnitt 9.4) für den Finanzsektor implementiert [108].

**HeaAn** Die HeaAn-Bibliothek [78, 40] implementiert CKKS [41] inklusive Bootstrapping. Die Bibliothek wird durch das Cryptography LAB an der Seoul National University entwickelt.

**FV-NFLLib** Die FV-NFLLib-Bibliothek [95] implementiert BFV [20]. Die Bibliothek wird durch das französische Unternehmen CryptoExperts entwickelt. Die Bibliothek ist Teil des europäischen Forschungsprojekts HEAT(Homomorphic Encryption Applications and Technology), welches sich unter anderem mit dem Anwendungsfall Smart Grids (siehe Abschnitt 9.3) beschäftigt hat.

**FHEW** Die FHEW-Bibliothek [62] implementiert TFHE mit einer verbesserten Bootstrapping Methode. Die Bibliothek wird laut den Autoren nicht weiter aktualisiert.

| Bibliothek | BGV | BFV | FHEW | TFHE | CKKS |
|---|---|---|---|---|---|
| Concrete [47] | | | | ✓ | |
| TFHE [45] | | | | ✓ | |
| OpenFHE [116] | ✓ | ✓ | ✓ | ✓ | ✓ |
| NuFHE [114] | | | | ✓ | |
| Microsoft SEAL [106] | | ✓ | | | ✓ |
| Lattigo-Bibliothek [89, 109] | | ✓ | | | ✓[6] |
| HELib [79] | ✓ | | | | ✓ |
| HeaAn [78, 40] | | | | | ✓[6] |
| FV-NFLLib [95] | | ✓ | | | |
| FHEW [62] | | | ✓ | ✓ | |

## 5.3 Hardwarebeschleunigte Implementierungen

Zusätzlich zur Implementierung von FHE-Verfahren in Software gibt es bereits Ansätze, Hardwarebeschleunigung für FHE-Verfahren zu verwenden. Diese Ansätze reichen von der Verwendung von Grafikkarten bis hin zu eigens für ein FHE-Verfahren konstruierte Spezialhardware.

**F1** Der F1 [127] Spezialhardwarechip wurde entworfen, um Berechnungen auf mit dem BGV-Verfahren verschlüsselten Daten zu beschleunigen. Er kann jedoch auch für das CKKS- und GSW-Verfahren verwendet werden. Bei diesem Ansatz werden nicht ganze FHE-Operationen, sondern für das FHE-Verfahren benötigte primitive Operationen in der Hardware implementiert. Kombiniert mit einem FHE-Compiler, welcher u.A. das Scheduling übernimmt, ist ein theoretischer Geschwindigkeitsgewinn von Faktor ca. 1.000× bis 15.000× gegenüber einer FHE-Softwareimplementierung möglich.

**GPU accellerated FHE** In [107] wird das TFHE-Verfahren in CUDA implementiert, was die Verwendung von Grafikkarten erlaubt. Im Gegensatz zur normalen Verwendung von TFHE, welches normalerweise auf booleschen Schaltkreisen arbeitet, wird hier ein Framework für arithmetische Operationen auf TFHE-Chiffraten erzeugt. Die parallele Struktur dieser Operationen erlaubt es dann, die Verarbeitungsweise von Grafikkarten auszunutzen. Hierbei wird ein Speedup von bis zu 20× erreicht.

**FPGA-Basierte Hardwarebschleunigung** In [136] wird ein FPGA-basierter Ansatz für Hardwarebeschleunigung vorgestellt. Der Ansatz kann für FHE-Verfahren basierend auf der RLWE-Annahme, wie z.B. das BGV-Verfahren verwendet werden. Hierbei werden diejenige Operationen in einem FPGA implementiert, welche bei einer normalen homomorphen Berechnung den höchsten Rechenaufwand verursachen, wobei

der Fokus der Implementierung auf mögliches Pipelining und daraus resultierenden Durchsatz von Anweisungen liegt. Bei der Ver- und Entschlüsselung kann gegenüber einer reinen Softwareimplementierung ein Speedup von bis zu Faktor 10 erreicht werden. In [136] werden keine Vergleichswerte zur homomorphen Schaltkreisauswertung gegeben, jedoch ist hierbei ein Speedup von mindestens Faktor 20 gegenüber einer Softwareimplementierung zu erwarten.

## 5.4 Compiler

Um ein Programm homomorph evaluieren zu können muss es als boolescher oder arithmetischer Schaltkreis vorliegen. In modernen Programmiersprachen übliche Anweisungen wie zum Beispiel der Vergleich zweier Zahlen sind in Schaltkreisen nicht als Operation vorhanden. Sie können jedoch mit Mehraufwand in Schaltkreisen implementiert werden, zum Beispiel in boolschen Schaltkreisen wie in Abbildung 7 gezeigt. Die Übersetzung eines Programms in einen Schaltkreis kann entweder von Hand durch einen Experten oder durch einen automatisierten Compiler durchgeführt werden. Es gibt einige Implementierungen von Compilern, welche für die verschiedenen FHE Varianten passende Schaltkreise erstellen. Die Compiler verfügen entweder über eine eigene Implementierung eines FHE Schemas oder nutzen eine der Open-Source Bibliotheken.

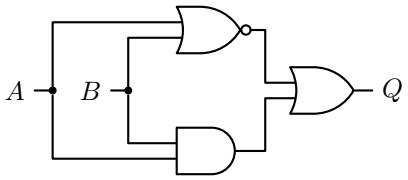



Abbildung 7: XNOR gate - Die Ausgabe dieses boolschen Schaltkreises ist $A$ XNOR $B$.

| Bibliothek | Eingabesprache | Zielimplementierung |
|---|---|---|
| Cingulata [36] | C++ | eigene BFV Implementierung, TFHE |
| HECO [142] | Python | SEAL |
| Alchemy [49] | Haskell | eigene BGV Implementierung |
| $E^3$ [42] | C++ | TFHE, FHEW, HElib, PALISADE, SEAL |
| EVA(CHET) [52] | Python | SEAL |
| Marble [141] | C++ | SEAL, HElib |
| Ramparts [9] | Julia | PALISADE |
| nGraph-HE [17] | Python | SEAL |
| SEALion [60] | Pytohn | SEAL |
| Google Transpiler [76] | C++ | TFHE |
| Hecate [91] | Python | SEAL |

Die aufgelisteten Bibliotheken übersetzen einfache Anweisungen aus der Eingabesprache in einen Schaltkreis, der dann durch eine der FHE Implementierungen ausgeführt werden kann. Die Compiler EVA/CHET, $E^3$ und nGraph-HE sind speziell auf die Anwendung mit neuronalen Netzen ausgelegt.

Ein weiterhin offenes Problem ist die Umsetzung von dynamischen Programmierelementen. Dies scheint ein schweres Problem zu sein, da zum Beispiel die Anzahl an Loop-Iterationen etwas über verschlüsselte Variablen preisgeben würde [36]. Gelöst werden könnte dies über konservative Abschätzungen der Loop-Iterationen, die allerdings manuell gemacht werden müsste.

# 6 Fortgeschrittene Verfahren

In diesem Abschnitt werden wir einige Varianten von FHE mit fortgeschrittener Funktionalität beziehungsweise fortgeschrittenen Sicherheitseigenschaften diskutieren.

## 6.1 FHE Hybridverschlüsselung

**Rate von Verschlüsselungsverfahren** Ein kritisches Merkmal aller kryptographischen Verfahren ist deren Kommunikationseffizienz. Im Normalfall vergleicht man den Kommunikationsaufwand kryptographischer Protokolle mit dem bestmöglichen unsicheren Protokoll. Im Falle von einfacher Public-Key Verschlüsselung beispielsweise vergleicht man die Größe von Chiffraten mit der Größe der entsprechenden Klartextnachrichten. Dies führt zum Konzept der *Rate* eines Verschlüsselungsverfahrens, welche für ein Chiffrat $c$ mit zugehörigem Klartext $m$ definiert ist durch

$$\rho = \frac{|m|}{|c|},$$

wobei $|\cdot|$ die Bitdartellungsgröße einer Nachricht ist. Da ein Chiffrat aus einfachen informations-theoretischen Gründen nicht kleiner sein kann als der zugehörige Klartext, ist die Rate ein Wert zwischen 0 und 1. Umgekehrt bezeichnet man $\theta = 1/\rho = |c|/|m|$ als den Chiffratexpansionsfaktor, welcher eine Zahl größer oder gleich 1 ist.

**Hybridverschlüsselung** Hybridverschlüsselung ist eine klassische Technik um die Rate von Public-Key Verschlüsselungsverfahren zu verbessern und basiert auf folgender Idee. Wenn eine lange Nachricht $m$ unter einem öffentlichen Schlüssel $\mathsf{pk}$ verschlüsselt werden soll, so generiert man zunächst einen frischen symmetrischen geheimen Schlüssel $K$. Dieser wird nun mittels des Public-Key Verfahrens zu $\mathbf{c}_0 = \mathsf{Enc}(\mathsf{pk}, K)$ verschlüsselt. Nun wird die eigentlichen Nachricht mittels eines symmetrischen Verschlüsselungsverfahrens $\mathsf{ENC}$ (beispielsweise AES) unter dem Schlüssel $K$ zu $\mathbf{c}_1 = \mathsf{ENC}(K, m)$. Das Gesamtchiffrat $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1)$ besteht nun aus den Komponenten $\mathbf{c}_0$ und $\mathbf{c}_1$. Da $\mathbf{c}_0$ nur den kurzen Schlüssel $K$ verschlüsselt, fällt die Größe von $\mathbf{c}_0$ bei der Größe des Gesamtchiffrates $\mathbf{c}$ nicht weiter ins Gewicht. Symmetrische Verfahren wie AES haben üblicherweise keine oder so gut wie keine Chiffratexpansion, daher ist $\mathbf{c}_1$ im wesentlichen genauso groß wie $m$. Die Rate des Hybridverfahren tendiert damit bei hinreichend langer Nachrichtenlänge gegen 1.

Um ein solches Chiffrat $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1)$ zu entschlüsseln wird zunächst $\mathbf{c}_0$ mittels des zu $\mathsf{pk}$ gehörigen geheimen Schlüssels $\mathsf{sk}$ zu $K$ entschlüsselt. Danach lässt sich $\mathbf{c}_1$ mit Hilfe des symmetrischen Schlüssels $K$ wieder zu $m$ entschlüsseln.

Da symmetrische Verschlüsselungsoperationen einen Faktor 100-1000 schneller sind als Public-Key Operationen hat dies zusätzlich den Vorteil dass das Hybridverfahren weitaus effizienter ist als wenn die gesamte Nachricht $m$ unter dem Public-Key Verfahren verschlüsselt würde.

**Hybridverschlüsselung mit FHE** Da FHE Verfahren Public-Key Verfahren sind, lassen sich diese direkt mittels Hybridverschlüsselung zu einem FHE Verfahren mit Rate 1 für *frische* Chiffrate umbauen. Dabei ist ein Chiffrat *frisch* falls es direkt vom Verschlüsselungsalgorithmus erzeugt wurde. Chiffrate die aus homomorphen Berechnungen hervorgehen sind damit nicht mehr frisch.

Um auf einem Hybridchiffrat homomorph rechnen zu können, muss dieses zunächst in ein reguläres Chiffrat des FHE Verfahrens expandiert werden. Dies lässt sich erreichen indem das symmetrische Chiffrat $\mathbf{c}_1$ in den Entschlüsselungsalgorithmus $\mathsf{DEC}$ des symmetrischen Verfahrens fest-verdrahtet wird, und die daraus resultierende Funktion $\mathsf{DEC}(\cdot, \mathbf{c}_1)$ homomorph auf dem FHE Chiffrat $\mathbf{c}_0 = \mathsf{Enc}(\mathsf{pk}, K)$ ausgewertet wird. Dies liefert ein Chiffrat

$$\mathbf{c}' = \mathsf{Eval}(\mathsf{pk}, \mathsf{DEC}(\cdot, \mathbf{c}_1), \mathbf{c}_0),$$

welches aufgrund der homomorphen Korrektheit des FHE Verfahrens eine Verschlüsselung von $\mathsf{DEC}(K, \mathbf{c}_1) = m$ ist. Mit diesem expandierten Chiffrat $\mathbf{c}'$ lässt sich nun wie gewohnt weiter rechnen, allerdings hat dieses Chiffrat nun die selbe (schlechte) Rate wie das FHE Verfahren.

**FHE-freundliche symmetrische Verschlüsselungsverfahren** Der kritische Schritt in der im letzten Paragraphen beschriebenen Konstruktion ist die *homomorphe Entschlüsselung* von $\mathbf{c}_1$, also die homomorphe Auswertung von $\mathsf{DEC}(\cdot, \mathbf{c}_1)$. Daher ist es nun erstrebenswert ein symmetrisches Verfahren zu verwenden für welches sich diese homomorphe Entschlüsselung mit möglichst geringem Aufwand realisieren lässt. Genauer werden symmetrische Verfahren benötigt bei welchen der Entschlüsselungsschaltkreis eine möglichst geringe *Tiefe* hat, sich also gut parallelisieren lässt damit nur wenige sequentielle Operationen benötigt werden, und welches weiterhin möglichst wenige *nicht-XOR* Gatter benötigt. XOR Gatter lassen sich bei der homomorphen Auswertung mittels Addition von Chiffraten realisieren, verursachen also keine (relative teuren) homomorphen Multiplikationen oder Bootstrapping Schritte. Derartige FHE-freundliche Verschlüsselungsverfahren wurden in [4] vorgeschlagen.

## 6.2 FHE mit hoher Rate

Bei FHE ist das Erreichen einer möglichst hohen Rate besonders kritisch: Werden FHE Chiffrate beispielsweise in einer Datenbank zwischengespeichert bevor sie weiterverarbeitet werden, dann führt eine geringe Rate/hoher Chiffratexpansionsfaktor zu einer hohen Speicherbelastung. Hat man beispielsweise einen Expansionsfaktor $\theta = 10000$, so führen 1 MB Klartextdaten zu 10 GB Chiffratdaten.

Bis vor kurzem hatten alle bekannten FHE eine derart hohe Chiffratexpansionsrate, was diese Verfahren für viele Anwendungsfälle wie beispielsweise verschlüsselte Datenbanken mit homomorphem Schreibzugriff völlig untauglich macht, da die Zusatzkosten durch Chiffratexpansion nahezu alle Vorteile des Einsatzes homomorpher Verfahren zunichte machen.

### 6.2.1 Rate-1 FHE via Homomorpher Entschlüsselung

In einer vor kurzem vorgestellten Arbeit [31] wurde das erste FHE Verfahren vorgeschlagen welches (asymptotisch) keine Chiffratexpansion hat. Dies bedeutet konkret, dass sobald die Menge verschlüsselter Klartextdaten groß genug ist, ist das Chiffrat im wesentlichen genauso groß wie der Klartext. Dies wird durch eine neuartige *Chiffratkompressionstechnik* möglich, welche wir im folgenden beschreiben werden.

Grundidee dieser Technik ist es zwei verschiedene homomorphe Verfahren mit unterschiedlichen Eigenschaften so miteinander zu kombinieren dass das resultierende Schema die Vorteile beider Verfahren in sich vereint. Benötigt wird hierzu:

1. Ein FHE Verfahren $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ mit *näherungsweise linearen Entschlüsselung* über $\mathbb{Z}_q$, also den ganzen Zahlen modulo $q$. Alle in Kapitel 4 diskutierten Verfahren haben diese Eigenschaft. Konkret lässt sich bei diesen Verfahren für ein Chiffrat $\mathbf{c}$ und einen geheimen Schlüssel $\mathbf{s}$ die näherungsweise Entschlüsselung schreiben als
$$\mathsf{Dec}_{\mathbf{s}}(\mathbf{c}) \approx L_{\mathbf{c}} \cdot \mathbf{s} = \mathbf{m} + \mathbf{e},$$
wobei $L_{\mathbf{c}}$ eine Matrix ist welche ausschließlich von dem Chiffrat $\mathbf{c}$ abhängt, $\mathbf{m}$ die entschlüsselte Nachricht ist, und $\mathbf{e}$ ein kurzer *additiver* Fehlerterm ist. Der Fehlerterm $\mathbf{e}$ ist der Grund weshalb man dies als näherungsweise Entschlüsselung bezeichnet. Durch eine entsprechende fehlerkorrigierende Codierung von $\mathbf{m}$ und einer weiteren *nichtlineare* Operation wie beispielsweise Runden lässt sich der Fehler $\mathbf{e}$ korrigieren.

2. Ein *linear homomorphes Verfahren* $(\mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}', \mathsf{Eval}')$ mit hoher Rate und Klartextraum $\mathbb{Z}_q$. Ein Beispiel für ein derartiges Verfahren ist das Damgård-Jurik Verfahren welches auf (quantenunsicheren) zahlentheoretischen Annahmen basiert. Als alternative dazu wurden in [31] linear homomorphe Verfahren mit hoher Rate von LWE konstruiert. Diese basieren auf einer neuartigen Chiffratkompressionstechnik mittels Rundung.

Beide Verfahren lassen sich nun wie folgt zu einem neuen Verfahren $(\mathsf{KeyGen}^*, \mathsf{Enc}^*, \mathsf{Dec}^*, \mathsf{Eval}^*$ kombinieren.

- Der Schlüsselerzeugungsalgorithmus $\mathsf{KeyGen}^*$ generiert zunächst ein Schlüsselpaar $(\mathsf{pk}, \mathsf{sk})$ für das FHE Verfahren mittels $\mathsf{KeyGen}$, danach ein Schlüsselpaar $(\mathsf{pk}', \mathsf{sk}')$ mittels $\mathsf{KeyGen}'$. Weiter erzeugt der Algorithmus ein Chiffrat $\mathbf{c}_{\mathsf{sk}'} = \mathsf{Enc}(\mathsf{pk}, \mathsf{sk}')$ des geheimen Schlüssels $\mathsf{sk}'$ unter $\mathsf{pk}$ sowie ein Chiffrat $\mathbf{c}'_{\mathsf{sk}} = \mathsf{Enc}'(\mathsf{pk}', \mathsf{sk})$des geheimen Schlüssels $\mathsf{sk}$ unter $\mathsf{pk}'$. Der öffentliche Schlüssel $\mathsf{pk}^*$ besteht nun aus $\mathsf{pk}$, $\mathsf{pk}'$, $\mathbf{c}_{\mathsf{sk}'}$ und $\mathbf{c}'_{\mathsf{sk}}$. Der geheime Schlüssel $\mathsf{sk}^*$ besteht aus $\mathsf{sk}$ und $\mathsf{sk}'$.

- Der Verschlüsselungsalgorithmus $\mathsf{Enc}^*$ verschlüsselt eine gegebene Nachricht $\mathsf{m}$ unter $\mathsf{pk}$ durch Verwendung von $\mathsf{Enc}'$ und gibt das entsprechende Chiffrat $\mathbf{c}'$ aus.

- Um eine solches Chiffrat zu entschlüsseln wird der Entschlüsselungsalgorithmus $\mathsf{Dec}'$ mit dem geheimen Schlüssel $\mathsf{sk}'$ auf ein solches Chiffrat angewandt.

- Der homomorphe Evaluationsalgorithmus $\mathsf{Eval}^*$ geht nun wie folgt vor um eine Funktion $f$ homomorph auszuwerten. Gegeben ein Chiffrat $\mathbf{c}'$ welches eine Nachricht $\mathbf{m}$ verschlüsselt, wird dieses zunächst homomorph zu einem Chiffrat $\mathbf{c}$ umverschlüsselt. Genauer wird wie folgt vorgegangen. Das Chiffrat $\mathbf{c}'$ wird in den Verschlüsselungsalgorithmus $\mathsf{Dec}'$ fest-verdrahtet, man erhält also eine Funktion $\mathsf{Dec}'(\cdot, \mathbf{c}'$, welcher als Eingabe noch einen geheimen Schüssel annimmt. Diese Funktion wird dann homomorph auf $\mathbf{c}_{\mathsf{sk}'}$ ausgewertet, man berechnet also

  $$\mathbf{c}^* = \mathsf{Eval}(\mathsf{pk}, \mathrm{Dec}'(\cdot, \mathbf{c}), \mathbf{c}_{\mathsf{sk}'}).$$

  Anhand der homomorphen Korrektheit des FHE Verfahrens ist $\mathbf{c}^*$ nun eine Verschlüsselung von $\mathsf{Dec}'(\mathsf{sk}', \mathbf{c}') = \mathbf{m}$. Man hat hiermit also effektiv des Chiffrat $\mathbf{c}'$ (unter $\mathsf{pk}'$) zu $\mathbf{c}^*$ (unter $\mathsf{pk}$) umverschlüsselt. Als nächstes wird ein Chiffrat

  $$\mathbf{c}^{**} = \mathsf{Eval}(\mathsf{pk}, f, \mathbf{c}^*)$$

  berechnet, wir werten also $f$ homomorph auf $\mathbf{c}^*$ aus. Anhand der homomorphen Korrektheit des FHE Verfahrens stellen wir fest dass $\mathbf{c}^{**}$ den Wert $f(\mathbf{m})$ verschlüsselt. Im letzten Schritt wird nun das Chiffrat $\mathbf{c}^{**}$ in ein Chiffrat $\mathbf{c}''$ unter $\mathsf{pk}'$ umverschlüsselt. Da der Entschlüsselungsalgorithmus $\mathsf{Dec}$ näherungsweise linear ist, existiert wie oben diskutiert eine Matrix $\mathbf{L}_{\mathbf{c}^{**}}$ für welche gilt

  $$\mathbf{L}_{\mathbf{c}^{**}} \cdot \mathsf{sk} = \mathrm{Dec}(\mathsf{sk}, \mathbf{c}^{**}) + \mathbf{e}$$

  für einen kurzen Fehlerterm $\mathbf{e}$. Nun wird $\mathbf{L}_{\mathbf{c}^{**}}$ homomorph auf $\mathbf{c}'_{\mathsf{sk}}$ ausgewertet und man erhält

  $$\mathbf{c}'' = \mathsf{Eval}'(\mathsf{pk}', \mathbf{L}_{\mathbf{c}^{**}}, \mathbf{c}'_{\mathsf{sk}}).$$

  Anhand der homomorphen Korrektheit des linear homomorphen Verfahrens ist nun $\mathbf{c}''$ eine Verschlüsselung von $\mathrm{Dec}(\mathsf{sk}, \mathbf{c}^{**}) + \mathbf{e}$. Ist der Fehlerterm $\mathbf{e}$ kurz genug und die Ausgabe von $\mathrm{Dec}(\mathsf{sk}, \mathbf{c}^{**})$ so codiert dass in den unteren Bits keine Daten enthalten sind, so kann der Fehler $\mathbf{e}$ später korrigiert werden.

Ist der Fehler $\mathbf{e}$ kurz genug, so beeinflusst dieser die Rate des Gesamtverfahrens nur marginal. Damit hat dieses im wesentlichen die selbe Rate wie das linear homomorphe Verfahren.

## 6.3 FHE on Approximate Numbers

Typischerweise sind die unter FHE Verfahren verschlüsselten Daten in Binärform, das heißt sie liegen als Bits vor. Dies hat zur Konsequenz, dass alle homomorphen Operationen mittels "Binärgattern" erfolgen müssen, also mittels Operationen wie logischem AND, OR, XOR usw. Allerdings werden für jede homomorphe Operation eine große Zahl arithmetischer Berechnungen benötigt. Tatsächlich unterstützen viele FHE prinzipiell auch arithmetische Operationen auf ganzzahligen Klartexten. Genau diesen Ansatz verfolgen "FHE on Approximate Numbers" Schemata. Dadurch, dass die Klartext-Daten bereits als Ganzzahlen vorliegen, lassen sich teils enorme Effizienzgewinne erzielen. Wird weiterhin die Korrektheitsanforderung solcher Verfahren

abgeschwächt, so lassen sich sehr effiziente Verfahren wie beispielsweise das CKKS Verfahren [41] konstruieren. Diese Verfahren wurden insbesondere vorgeschlagen für homomorphe Berechnungen auf Daten welche üblicherweise als numerische Daten (statt als Bits) vorliegen, so beispielsweise im Bereich des maschinellen Lernens.

### 6.3.1 Das CKKS Verfahren

Basis des CKKS Verfahrens [41] ist das Verfahren von Brakerski, Gentry und Vaikuntanathan (BGV) [24], welches selbst eine Variante des Verfahrens von Brakerski und Vaikuntanathan [25] darstellt.

Wie bereits in Kapitel 4.2 dargestellt ist die Entschlüsselung bei diesen Verfahren eine *lineare Operation*, das heißt Entschlüsselung eines Chiffrates $\mathbf{c}$ lässt sich als näherungsweise als inneres Produkt mit dem geheimen Schlüssel $\mathbf{s}$ darstellen und es gilt

$$\mathbf{c} \cdot \mathbf{s} = \frac{q}{2}m + e,$$

wobei $q/2 \cdot m$ eine Kodierung der Nachricht $m \in \{0, 1\}$ im obersten Bit eines $\mathbb{Z}_q$ Elements ist und $e$ ein additiver Fehlerterm ist. Die Grundidee beim CKKS Verfahren ist es nun den Klartextraum dieses Verfahrens von $\{0, 1\}$ auf $\{0, \ldots, 2^k - 1\}$ zu ändern. Das heißt dass nun anstatt Bits *hinreichend kleine* Zahlen verschlüsselt werden. Weiterhin wird auf die Codierung der Nachricht $m$ verzichtet, was bedeutet dass die Entschlüsselungsoperation bei diesem Verfahren nicht mehr korrekt ist, das heißt es gilt nun

$$\mathbf{c} \cdot \mathbf{s} = m + e.$$

Insbesondere bedeutet dies dass homomorphes Rechnen und Klartextoperationen zu unterschiedlichen Ergebnissen führen. Da der Fehlerterm $e$ *kurz* ist, führt dies allerdings nur zu einem kleinen Korrektheitsfehler. Die Argumentation in [41] warum dies hinnehmbar sei, basiert darauf dass selbst arithmetische Operationen auf Gleitkommazahlen im Klartext *nicht korrekt* sind. Dies ist tatsächlich notwendig, da reale Rechner Zahlen nur mit endlicher Präzision darstellen können, tatsächliche Berechnungsergebnisse allerdings im Allgemeinen reelle oder komplexe Zahlen sein können und daher nicht mit endlicher Präzision als numerischer Wert dargestellt werden können.

Die homomorphen Eigenschaften des BGV Verfahrens (siehe Abschnitt 4.2) bleiben erhalten; für zwei Chiffrate $(\mathbf{c}_0, c_1)$ und $(\mathbf{c}'_0, c'_1)$ und den expandierten geheimen Schlüssel $\hat{\mathbf{s}} = (\mathbf{s}, \mathbf{s} \otimes \mathbf{s})$ gilt weiterhin

$$c_1 \cdot c'_1 - \hat{\mathbf{s}}\hat{\mathbf{c}}_0 = m \cdot m' + 2e^*,$$

wobei $\hat{\mathbf{c}}_0 = (-c'_1\mathbf{c}_0 - c_1\mathbf{c}'_0, \mathbf{c}_0 \otimes \mathbf{c}'_0)$ und $e^* = mer' + m'er + 2(er) \cdot (er')$.

Der Zweck dieser Modifikation ist ein besseres Ausnutzen des Chiffratraums. Beim BGS Verfahren verschlüsselt ein Chiffrate in $\mathbb{Z}_q^{n+1}$ lediglich ein einziges Bit. Beim CKKS Verfahren wird hier nun eine Zahl in der Menge $\{0, \ldots, 2^k - 1\}$ verschlüsselt. Weiterhin wird in CKKS eine Modifikation vorgeschlagen den Ring der ganzen Zahlen $\mathbb{Z}$ durch einen Ganzheitsring $\mathcal{R}$ in einem Kreisteilungskörper zu ersetzen. Diese Modifikation erlaubt es die Packungsdichte zu verbessern, da nun im Wesentlichen jedes $\mathbb{Z}_q$ Element eine Zahl in $\{0, \ldots, 2^k - 1\}$ verschlüsselt.

In einer Folgearbeit [39] wurde ein Bootstrapping Algorithmus für dieses Verfahren vorgestellt. Allerdings sehen derzeit alle Anwendungsfälle vor das CKKS Verfahren ohne Bootstrapping, also nur auf *flachen* Berechnungen zu verwenden, da das Bootstrapping auch bei diesem Verfahren zu einem vergleichsweise exorbitanten Effizienzverlust führt.

### 6.3.2 Korrektheitsfehler von CKKS wird zu Sicherheitsproblem

Das Verfahren von CKKS selbst ist als homomorphes Verschlüsselungsverfahren unter der Learning with Errors (LWE) Annahme sicher. Allerdings ist dies bei der Konstruktion von sicheren Zwei- oder Mehrparteienberechnungsprotokollen (siehe Abschnitt 7) allein nicht ausreichend. Bei sicheren Mehrparteienberechnungen wird Sicherheit im Normalfall gegen einen Angreifer modelliert welcher bereits Kenntnis aller Ein-

und Ausgaben hat. Aufgabe des Angreifers ist es lediglich eine Ausführung des realen Protokolls von einer *Simulation* zu unterscheiden. Dies erlaubt es vernünftige Sicherheitsgarantien selbst dann abzugeben, wenn ein Angreifer bereits partielle Informationen zu Ein- und Ausgaben.

Hieraus ergibt sich nun ein fundamentales Sicherheitsproblem wenn CKKS in MPC Anwendungen eingesetzt wird, da hierbei über den Fehlerterm im Berechnungsergebnis Information über den Secret Key abfließt [93]. Wie oben beschrieben gilt beim CKKS Verfahren für ein Chiffrat $\mathbf{c}$ mit zugehörigem Klartext $m$ dass

$$\mathbf{sc} = m + e.$$

Im Allgemeinen lässt sich nicht verhindern dass ein Angreifer $\mathbf{c}$ lernt, da Chiffrate als öffentlich behandelt werden. Weiterhin muss davon ausgegangen werden, dass ein Angreifer das entschlüsselte Berechnungsergebnis $y = m + e$ lernt. Sobald dies jedoch der Fall ist besitzt der Angreifer eine *linear Gleichung* $\mathbf{sc} = y$ über den geheimen Schlüssel $\mathbf{s}$, wobei $\mathbf{c}$ und $y$ bekannt und $\mathbf{s}$ unbekannt ist. Sind hinreichend viele derartige Gleichungen gegeben, was der Fall ist wenn ein und derselbe Schlüssel mehrfach verwendet wird, so erhält man ein lineares Gleichungssystem von vollem Rang und kann dies beispielsweise mittels Gaußelimination effizient nach $\mathbf{s}$ auflösen. Tatsächlich war diese Problematik von *rein linearer Entschüsselbarkeit* seit Längerem bekannt [22], wurde aber scheinbar von den Autoren von CKKS übersehen.

**Gegenmaßnahmen** Als Gegenmaßnahmen für diese Problematik kommen folgende Modifikation von CKKS in Betracht.

- **Einmalige Verwendung eine Public-Secret Key Paars** Wird ein Paar $(\mathsf{pk}, \mathsf{sk})$ nur für eine einzige homomorphe Berechnung verwendet, sprich wird jedes mal wenn eine Berechnung einer Funktion $f$ ausgelagert wird ein neues Schlüsselpaar $(\mathsf{pk}, \mathsf{sk})$ erzeugt, so kann der oben beschriebene Angriff nicht durchgeführt werden. Tatsächlich kann das Verfahren unter einer geeigneten Variante des LWE Problems, nämlich des *Extended LWE* Problems [30] immer noch als sicher bewiesen werden.

- **Noising der Ausgabe** Der oben beschriebene Angriff stützt sich darauf, dass der Angreifer eine hinreichende Menge *exakte lineare Gleichungen* der Form $\mathbf{sc} = y$ lernt. Fügt man dem Entschlüsselungsergebnis nun einen *zusätzlichen* Fehlerterm $e'$ hinzu, zu lässt sich diese lineare Beziehung brechen. Hierbei wird der Entschlüsselungsalgorithmus durch

  $$\mathsf{Dec}(\mathbf{s}, \mathbf{c}) = \mathbf{sc} + e'$$

  definiert, wobei $e'$ einen frischen kurzen Fehlerterm darstellt, welcher durch den Entschlüsselungsalgorithmus gewählt wird. Ist die Verteilung von $e'$ hinreichend *breit*, so gilt

  $$\mathsf{Dec}(\mathbf{s}, \mathbf{c}) = \mathbf{sc} + e' = m + e + e' \approx m + e'.$$

  Der Fehlerterm $e'$ verschluckt also sozusagen den kürzeren Fehler $e$. Durch diese als *Drowning* bekannte Technik [8] fließt keine Seiteninformation mehr über den geheimen Schlüssel durch das Entschlüsselungsergebnis ab. Allerdings wirkt sie den Effizienzgewinnen von CKKS diametral entgegen und hebt diese auf, da der zusätzliche Fehlerterm die Rechengenauigkeit senkt. Einer kürzlich vorgeschlagenen Alternative [94] gelingt es diesen Ansatz für bestimmte Anwendungen mit weniger aggressiven Rauschparametern umzusetzen.

## 6.4 Multikey FHE

FHE ist zunächst einmal eine Primitive für zwei Parteien, welche es einem Delegator ermöglicht eine Berechnung sicher an einen Server auszulagern. Für den Fall, dass mehrere sich gegenseitig misstrauende Parteien gemeinsam eine sichere Berechnung durchführen möchten, reichen FHE Verfahren alleine nicht aus, sondern es sind zusätzliches Protokolle notwendig welche Schlüssel generieren und das Berechnungsergebnis verteilt entschlüsseln. Bei Multikey FHE Verfahren [97] ist ein derartiger Operationsmodus bereits im FHE Verfahren

angelegt. Dies bedeutet insbesondere, dass homomorphe Operationen auf Chiffraten durchgeführt werden können, die mit verschiedenen Schlüsseln erzeugt wurden.

Konkret bedeutet dies, dass bei derartigen Verfahren der Evaluationsalgorithmus $\mathsf{Eval}$ öffentliche Schlüssel $\mathsf{pk}_1, \ldots, \mathsf{pk}_k$ sowie Chiffrate $\mathbf{c}_1, \ldots, \mathbf{c}_k$ entgegen nimmt, wobei $\mathbf{c}_i$ jeweils eine Verschlüsselung einer Nachricht $m_i$ unter $\mathsf{pk}_i$ ist. Dabei ist zu betonen dass die Schlüssel $\mathsf{pk}_1, \ldots, \mathsf{pk}_k$ *unabhängig* voneinander erzeugt werden, das heißt es ist insbesondere nicht möglich mit dem zu $\mathsf{pk}_i$ gehörigen geheimen Schlüssel $\mathsf{sk}_i$ ein Chiffrat $\mathbf{c}_j$ (für $j \neq i$) zu entschlüsseln. Es lässt sich also ein Multikey-Chiffrat $\mathbf{c}^*$ erzeugen durch

$$\mathbf{c}^* = \mathsf{Eval}(\mathsf{pk}_1, \ldots, \mathsf{pk}_k, f, \mathbf{c}_1, \ldots, \mathbf{c}_k),$$

wobei $f(m_1, \ldots, m_k)$ die auszuwertende Funktion ist. Darüber hinaus unterstützen Multikey-FHE Verfahren eine *verteilte Entschlüsselung* $\mathsf{Dec}^*$. Gegeben ein Multikey-Chiffrat $\mathbf{c}^*$ und einen der geheimen Schlüssel $\mathsf{sk}_i$ erzeugt $\mathsf{Dec}^*$ eine *partielle Entschlüsselung* $t_i = \mathsf{Dec}^*(\mathsf{sk}_i, \mathbf{c}^*)$, aus welcher sich noch nichts über den verschlüsselten Wert ableiten lässt. Es gibt nun zusätzlich eine *öffentliche Rekonstruktionsfunktion* $R(t_1, \ldots, t_k)$ für welche gefordert wird dass

$$R(t_1, \ldots, t_k) = f(m_1, \ldots, m_k),$$

wobei $t_i = \mathsf{Dec}^*(\mathsf{sk}_i, \mathbf{c}^*)$ und $\mathbf{c}^* = \mathsf{Eval}(\mathsf{pk}_1, \ldots, \mathsf{pk}_k, f, \mathbf{c}_1, \ldots, \mathbf{c}_k)$. Die Funktion $R$ erlaubt es also partiell entschlüsselte Werte so zusammenzuführen dass der final rekonstruierte Wert dem korrekten Funktionsergebnis entspricht.

Multikey FHE Verfahren gehören in der Theorie zu den effizientesten Ansätzen für sichere Mehrparteienberechnungen (Abschnitt 7), für praktische Anwendungen sind sie derzeit aufgrund des recht großen konkreten Aufwandes noch nicht mit anderen alternativen Verfahren konkurrenzfähig.

## 6.5 Quanten FHE

Das Standardszenario von FHE ist die Ausführung komplexer Berechnungen auf nicht-vertrauenswürdigen Servern. Quanten FHE [102, 21, 37] ermöglicht das sichere Auslagern von Quantenberechnungen an einen Quantenserver durch einen rein klassischen Delegator (Abbildung 8).

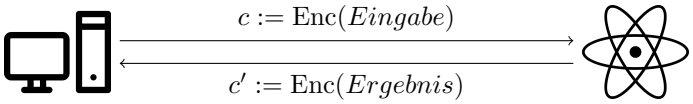



Abbildung 8: Delegation von Quantenberechnungen durch Quanten FHE

Der Quantenserver benötigt hierfür einen *universellen Quantencomputer*, welcher derzeit noch weit jenseits der Fähigkeiten verfügbarer Quantentechnologie liegt. Das dadurch ermöglichte Szenario lässt sich als *Quantum-Computing-as-a-Service* in einer Post-Quantum Welt vorstellen: Eine Institution welche einen möglicherweise teure Quantenrechner besitzt, kann für andere Parteien Quantenberechnungen durchführen, ohne dabei etwas über die Ein- oder Ausgabe der Berechnung zu lernen. Nach derzeitigem Forschungsstand wären Quantencomputer, wenn es denn möglich wird sie physikalisch zu realisieren, prinzipbedingt hoch-komplexe Anlagen welche ein hohes Ausmaß an Kühlung und Abschirmung benötigen um die empfindlichen Quantenüberlagerungszustände welche für Quantenberechnungen nötig sind zu erhalten. Daher könnte das Konzept der Quantum FHE hochrelevant werden sobald Quantenrechner zur Verfügung stehen. Die Forschung in diesem Bereich hat kürzlich gezeigt, dass sich viele fortgeschrittene Entwicklungen, wie zum Beispiel geringe Chiffratexpansion, auch für Quantum FHE verwirklichen lassen [37].

Bei Quanten FHE können sowohl Klartexte als auch Berechnungsergebnisse selbst Quantenzustände sein. Das bedeutet konkret, dass bei der Verschlüsselung die Nachricht $m$ ein Quantenzustand sein kann, also eine *Überlagerung* von klassischen Basiszuständen. Wir werden im Folgenden nicht auf die Besonderheiten

der Quanteninformationstheorie eingehen da dies den Rahmen dieser Übersicht überschreiten würde. Wichtig hierbei ist nur, dass sich Quanteninformation im Unterschied zu klassischer Information *nicht* beliebig kopieren und manipulieren lässt, sondern nur *kohärente* Informationsverarbeitung möglich ist.

Alle derzeitigen Konstruktionen von Quantum FHE [102, 21, 37] folgen dabei dem Prinzip der FHE Hybridverschlüsselung. Ein Quantenzustand $|\Phi\rangle$ wird zunächst mittels eines Quanten-One-Time-Pads (QOTP) mit klassischem Schlüssel $K$ verschlüsselt. Nun bezeichne $|\gamma\rangle$ den Quantenzustand des Chiffrates. Nun wird zusätzlich der klassische One-Time-Pad Schlüssel $K$ mithilfe eines *rein klassischen* FHE Verfahrens zu einem klassischen Chiffrat $\mathbf{c}$ verschlüsselt. Das Quantenchiffrat ist nun gegeben durch $|\gamma\rangle$ und $\mathbf{c}$. Ein derartiges Chiffrat lässt sich nun dadurch entschlüsseln dass zunächst $\mathbf{c}$ zu $K$ entschlüsselt wird, und danach $|\gamma\rangle$ mittels $K$ zu $|\Phi\rangle$ entschlüsselt wird.

Der interessante Aspekt an diesen Verfahren sind nun homomorphe Quantenoperationen auf derartigen Chiffraten. Bereits seit längerem bekannt war dass der QOTP homomorphe Operationen in der sogenannten Cliffordgruppe [32] unterstützt. Dies bedeutet dass derartige Operationen direkt auf $|\gamma\rangle$ ausgeführt werden können und nur der unter der FHE verschlüsselte QOTP Schlüssel $K$ angepasst werden muss, was bei einer FHE direkt mittels des klassischen homomorphen Auswertungsalgorithmus möglich ist. Es stellt sich heraus, dass die einzig fehlende Komponente um alle möglichen (*unitären*) Quantenoperationen zu unterstützen ein sogenanntes *klassisch gesteuertes homomorphes CNOT Gatter* ist. Etwas spezifischer, gegeben einen Quantenüberlagerungszustand $\sum_{a,b} \alpha_{a,b}|a\rangle|b\rangle$ und ein klassisches FHE Chiffrat $\mathsf{Enc}(x)$ möchte man diese in einen neuen Zustand $\sum_{a,b} \alpha_{a,b}|a\rangle|ax+b+K'\rangle$ und ein Chiffrat $\mathsf{Enc}(K')$ überführen. Es stellt sich heraus, dass dies unter bestimmten Voraussetzungen mittels des klassischen Evaluationsalgorithmus möglich ist, nämlich genau dann, wenn das klassische FHE Verfahren Circuit-Privacy besitzt, also die Ergebnisse homomorpher Berechnungen nichts über den Rechenweg verraten. Alle diese Verfahren [102, 21, 37] können unter der LWE Annahme als sicher bewiesen werden.

Da derzeit noch keine universellen Quantencomputer zur Verfügung stehen, kann Quanten-FHE derzeit noch nicht praktisch genutzt werden und stellt bisher ein rein theoretisches Konzept dar.

## 6.6 Multilineare Abbildungen, Split-FHE und Obfuscation

FHE Verfahren kommt bei einer Zahl neuer Ansätze zur Konstruktion kryptographischer Obfuscation eine zentrale Bedeutung zu. Dies war insbesondere der Fall bei einigen Durchbrüchen im Bereich *kryptographisch sichere Obfuscation* in den letzten 10 Jahren.

**Kryptographische Obfuscation** Kurz umschrieben ist ein Obfuskator eine Programmtransformation beziehungsweise ein Compiler, welcher ein gegebenes Programm $\mathcal{P}$ so zu einem neuen Programm $\mathcal{O}$ verschlüsselt, dass dieses *immer noch ausführbar bleibt* und Klartextausgaben erzeugt. Im Vergleich zu FHE lässt sich mit Obfuscation also erreichen, dass das Ergebnis einer verschlüsselten Berechnung öffentlich zugänglich wird, und damit kein geheimer Schlüssel notwendig ist um dieses zu entschlüsseln. Obfuscation ist ein weitaus stärkeres kryptographisches Primitiv als FHE, tatsächlich lässt sich FHE mittels Obfuscation und einigen elementaren kryptographischen Zusatzannahmen konstruieren [35].

Formal gesehen ist ein Obfuskator $\mathsf{iO}$ eine Programmtransformation welches eine Programm gegeben in Form eines Schaltkreises $\mathsf{C}$ als Eingabe nimmt, und ein funktional äquivalentes Programm $\tilde{\mathsf{C}}$ ähnlicher Größe ausgibt. Dabei heißt funktional äquivalent, dass wenn $\mathsf{C}$ Eingaben aus der Menge $\{0,1\}^n$ nimmt, dass dann für jedes $x \in \{0,1\}^n$ gilt dass $\mathsf{C}(x) = \tilde{\mathsf{C}}(x)$. *Ähnlicher Größe* bedeutet dabei dass die Darstellung des Schaltkreises $\tilde{\mathsf{C}}$ nur polynomiell größer ist als die von $\mathsf{C}$. Die Sicherheit eines solchen Obfuskators $\mathsf{iO}$ wird folgendermaßen definiert. Gegeben zwei funktional äquivalente Schaltkreise $\mathsf{C}_0$ und $\mathsf{C}_1$, fordern wir dass

$$\mathsf{iO}(\mathsf{C}_0) \approx_C \mathsf{iO}(\mathsf{C}_1),$$

die obfuszierten Schaltkreise $\mathsf{iO}(\mathsf{C}_0)$ und $\mathsf{iO}(\mathsf{C}_1)$ also für jeden effizienten Angreifer ununterscheidbar sind. Man nennt einen Obfuskator $\mathsf{iO}$ welcher diese Sicherheitseigenschaft erfüllt *Indistinguishability Obfuscator* [13, 65].

Wie bereits angedeutet, war die Entdeckung der ersten Obfuscation-Kandidaten eng verwoben mit der Entwicklung von FHE Verfahren.

**Multilineare Abbildungen** Dies kristallisiert sich insbesondere im Begriff der *kryptographischen multilinearen Abbildungen* [64]. Ohne tiefer in Details einzusteigen kann man sich eine derartige multilineare Abbildung als ein FHE Verfahren vorstellen bei welchem es einen zusätzlichen Algorithmus, einen sogenannten Nulltest $\mathsf{ZeroTest}$ gibt, welcher öffentlich feststellen kann ob ein gegebenes Chiffrate eine Verschlüsselung von 0 ist. Dieser Algorithmus *bricht* damit ein Stück weit die Sicherheit des zugrundeliegenden FHE Verfahren, dies ist aber essentiell für die Konstruktion von Obfuscation Verfahren mittels dieses Ansatzes.

Zwar bestand lange Zeit die Hoffnung, dass es *irgendwie* möglich sein sollte mittels dieses Ansatzes sichere Obfuscation Schemata zu konstruieren. Leider wurden jedoch im wesentlichen alle derartigen Ansätze mithilfe des Nulltests gebrochen, weshalb diese Forschungsrichtung nicht mehr weiter verfolgt wird.

**Split-FHE** Ein dazu alternativer Ansatz wurde kürzlich initiiert [29]. Hierbei wird, grob gesagt, ein spezielles FHE Verfahren konstruiert (Split-FHE), bei welchem Chiffrate welche einen *langen Klartext* verschlüsseln mittels eines *kurzen Hinweises* entschlüsselt werden können. Es stellt sich heraus, dass ein derartiger Mechanismus bereits hinreichend ist um allgemeine kryptographische Programm-Obfuscation zu konstruieren [96].

Ein Split-FHE Verfahren ist eine gewöhnliches FHE Verfahren welches um zwei Algorithmen, die sogenannte Split-Decryption erweitert wird.

- Der erste Algorithmus $\mathsf{PDec}$, auch partielle Entschlüsselung genannt, nimmt als Eingabe einen geheimen Schlüssel $\mathsf{sk}$ und ein Chiffrat $\mathbf{c}$ (welches die Ausgabe einer homomorphen Berechnung sein kann), und gibt einen *Hinweis* $\rho$ aus.
- Der zweite Algorithmus $\mathsf{Rec}$ (für Recovery), nimmt ein Chiffrat $\mathsf{c}$ und einen Hinweis $\rho$ und gibt einen Klartext $m$ aus.

Zwei Aspekte sind hierbei essentiell damit ein derartiges Verfahren hilfreich bei der Konstruktion von Obfuscation ist:

1. Der Hinweis $\rho$ soll in seiner Bitdarstellung signifikant kürzer sein als der verschlüsselte Klartext $m$.
2. Der Hinweis $\rho$ soll nicht mehr über das Chiffrat $\mathbf{c}$ verraten als den Klartext $m$. Das heißt insbesondere, dass $\mathbf{c}$ nicht verraten soll, ob $\mathbf{c}$ das Ergebnis einer homomorphen Berechnung ist.

In [29] wird nun, basierend auf [31] ein Split-FHE Verfahren basieren auf GSW und dem Damgård-Jurik (DJ) Verfahren konstruiert (vergleiche Abschnitt 6.2). Die zugrundeliegende Idee dabei ist, dass das linear homomorphe DJ Verfahren bereits über Split-Decryption verfügt und sich mittels GSW zu einem FHE Verfahren mit dieser Eigenschaft erweitern lässt. Bisher ist es allerdings noch nicht gelungen die Sicherheit dieser Konstruktion auf die Sicherheitseigenschaften ihrer Komponenten zu reduzieren.

Da die tatsächliche Konstruktion von Obfuscation mittels Split-FHE eine Reihe hochgradig ineffizienter generischer Transformationsschritte enthält, beschreiben wir im Folgenden nur den Hauptschritt, mit welchem sich die Funktionstabelle eines Schaltkreises mittels Split-FHE komprimieren lässt, ohne zusätzliche Details über den Schaltkreis zu verraten.

Dieser *schwache Obfuskator* geht nun wie folgt vor. Gegeben ein Schaltkreis $\mathsf{C}$ welcher Eingaben $x \in \{0,1\}^k$ nimmt, kann dieser in seine Funktionstabelle expandiert werden indem $\mathsf{C}$ auf jeder Eingabe $x \in \{0,1\}^k$ ausgewertet wird. Man erhält also eine Liste von Paaren von Eingaben und Funktionswerten $T = (x, \mathsf{C}(x))_{x\in\{0,1\}^k}$. Ziel ist es nun $T$ zu *komprimieren*, das heißt eine Darstellung $\tilde{T}$ von $T$ zu finden, welche signifikant kleiner als $T$ ist, d.h. kleiner als $2^k$ Bits (für den generischen Fall das $\mathsf{C}$ nur ein einzelnes Bit ausgibt) und weiterhin keine Zusatzinformation über $\mathsf{C}$ verrät. Mittels Split-FHE lässt sich dies wie folgt erreichen. Zunächst wird der Schaltkreis $\mathsf{C}$ mittels Split-FHE in zu einem Chiffrat $\mathbf{c}$ verschlüsselt. Nun wird der verschlüsselte Schaltkreis $\mathbf{c}$ homomorph auf allen Eingaben $x \in \{0,1\}^k$ ausgewertet, man erhält also eine Verschlüsselung $\mathbf{c}^*$ der Funktionstabelle $T$. Dieses Chiffrat wird nun mittels des partiellen Entschlüsselungsalgorithmus und dem geheimen Schlüssel zu einem Hinweis $\rho$ entschlüsselt. Die komprimierte Funktionstabelle $\tilde{T}$ besteht nun aus $\mathbf{c}$ und $\rho$.

Da $\mathbf{c}$ nur geringfügig größer ist als $\mathsf{C}$ und damit signifikant kleiner als $T$, und auch $\rho$ signifikant kleiner ist als $T$ (aufgrund von obigem Punkt 1), ist damit auch $\tilde{T}$ signifikant kleiner als $T$. Somit ist $\tilde{T}$ eine komprimierte Version von $T$.

Um $\tilde{T}$ öffentlich zu dekomprimieren geht man wie folgt vor. Zunächst wird wird wieder $\mathbf{c}$ homomorph an allen Eingaben $x \in \{0,1\}^k$ ausgewertet, und man erhält somit wieder das Chiffrat $\mathbf{c}^*$ welches die Funktionstabelle $T$ verschlüsselt. Dieses Chiffrat lässt sich nun mithilfe des Recovery-Algorithmus $\mathsf{Rec}$ und des Hinweises $\rho$ zu $T$ entschlüsseln.

# 7 Sichere Mehrparteienberechnungen (MPC)

Die meisten mittels FHE realisierbare Anwendungen können ebenfalls mit sicherer Mehrparteienberechnung (Secure Multi-Party Computation, MPC) umgesetzt werden. MPC-basierte Lösungen erfordern meist eine geringere Berechnungskomplexität, allerdings auf Kosten einer höheren Kommunikationskomplexität.

MPC erlaubt es einer Menge von Parteien $P_1 \ldots, P_n$, die sich gegenseitig misstrauen, gemeinsam eine Funktion $f$ auf ihren geheimen Eingaben $x_1, \ldots, x_n$ auszuwerten, und eine Ausgabe $y_1, \ldots, y_n$ zu erhalten.

Im Allgemeinen wird dazu angenommen, dass die Parteien $P_i$ über paarweise Kommunikationskanäle verfügen, die *authentifizierte* oder *sichere* (also authentifizierte und geheime) Kommunikation ermöglichen. Zum Berechnen der Funktion $f$ führen die Parteien ein Protokoll $\pi$ aus und schicken sich gegenseitig Nachrichten zu. Am Ende der Protokollausführung soll eine Partei $P_i$ ihre Ausgabe $y_i$ kennen, aber durch die Protokollausführung „nicht mehr“ gelernt haben.

Wir nehmen dabei an, dass eine Teilmenge der Parteien korrumpiert ist, also gezielt versuchen, durch die Teilnahme am Protokoll $\pi$ mehr als intendiert zu erlernen. Dies schließt den Fall mit ein, dass mehrere korrumpierte Parteien miteinander kooperieren, um einen Vorteil zu erlangen. Formal nehmen wir die Existenz eines *Angreifers* an, der für die korrumpierten Parteien am Protokoll teilnimmt.

Für eine solche Protokollausführung werden bestimmte Eigenschaften gewünscht, die wir zunächst informell skizzieren möchten:

- Korrektheit: Nach der Protokollausführung besitzen die Parteien die korrekte Ausgabe $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$.
- Privatheit: Eine Partei lernt nichts aus der Protokollausführung, das sie nicht aus ihrer Eingabe $x_i$, der zu berechnenden Funktion $f$ und der Ausgabe $y_i$ lernen kann[8].
- Unabhängigkeit der Eingaben: Eine korrumpierte Partei $P_i$ soll ihre Eingabe nicht in Abhängigkeit der Eingabe einer ehrlichen Partei $P_j$ wählen können[9]

Diese Liste ist nicht abschließend. Je nach Anwendungsfall können weitere Eigenschaften notwendig sein.

Für sichere Mehrparteienberechnungen gibt es eine Reihe von etablierten Anwendungen wie private Schnittmengenberechnungen, Berechnung des *market clearing price*, elektronische Wahlen, aber auch Berechnungen auf Genomdaten.

Seit den 80er Jahren sind darüber hinaus sehr starke Ergebnisse bekannt: Für (fast[10]) jede (in Polynomialzeit berechenbare) Funktion $f$ gibt es ein Protokoll $\pi$, das $f$ sicher berechnet [145, 144, 75].

## 7.1 Varianten

Bei sicheren Mehrparteienberechnungen gilt es eine Reihe von Varianten zu unterscheiden.

**Angreifermodell.** Bei *passiver Korruption*, auch *honest-but-curious* genannt, halten sich die korrumpierten Parteien an die Protokollausführung. Der Angreifer kennt dabei lediglich ihre Ein- und Ausgaben, das Zufallsband sowie die empfangenen Nachrichten. Dieser Begriff ist beispielsweise im Umfeld von Cloud-Computing etabliert.

Ein stärkerer Fall stellt die *aktive Korruption*, auch *malicious corruption* genannt, dar. Hier darf der Angreifer beliebig vom Protokoll abweichen um einen Vorteil zu erlangen. Im Allgemeinen sind Protokolle,

[8] Auch eine sichere Mehrparteienberechnung kann einen solchen trivialen Informationsfluss nicht ausschließen. Berechnen zwei Parteien beispielsweise das exklusive Oder auf ihren Eingaben $x_1$ und $x_2$, so ist unter Kenntnis von $x_i$ und $y_i = x_1 \oplus x_2$ per Definition schon $x_{2-i}$ berechenbar.

[9] $P_i$ darf also beispielsweise bei einer Online-Auktion nicht in der Lage sein, ein Chiffrat $c_j$, das das Gebot einer Partei $P_j$ enthält, in ein Chiffrat $c_i$ mit einem um 1 € erhöhten Gebot umzuformen. Je nach Verschlüsselungsverfahren muss $P_i$ den Inhalt von $c_j$ dafür gar nicht kennen. Insbesondere wird dieser Angriff vom etablierten Begriff der IND-CPA Sicherheit nicht verhindert und von homomorpher Verschlüsselung explizit unterstützt.

[10] Die Einschränkung ist technischer Natur und in der Praxis nicht relevant.

die vor aktiver Korruption schützen, aufwendiger und ineffizienter, da spezielle Techniken notwendig sind um den Angreifer zu einem ehrlichen Verhalten zu „zwingen“.

Neben der Art der Korruption gibt es auch Unterschiede bezüglich des Korruptionszeitpunkts. So bestimmen *statische* Angreifer zu Beginn der Protokollausführung, welche Teilmenge der Parteien unter ihrer Kontrolle ist. Dem gegenüber dürfen *adaptive* Angreifer vor, während und sogar nach der Ausführung Parteien korrumpieren, beispielsweise in Abhängigkeit der Protokollnachrichten oder aufgrund von Informationen, die bei der Korruption anderer Parteien gelernt wurden.

Es gibt noch eine ganze Reihe von weiteren Angreifermodellen. Hier zu erwähnen sind beispielsweise *mobile adversaries*, die Parteien auch wieder „entkorrumpieren“ und dadurch nacheinander alle Parteien korrumpiert haben können. Ebenfalls interessant sind *covert adversaries*, die ähnlich wie aktive Angreifer zwar vom Protokoll abweichen können, die aber ehrlich bleiben, wenn es eine gewisse Wahrscheinlichkeit gibt, entlarvt zu werden.

**Sicherheitsziele.** Bei den Sicherheitszielen unterscheiden wir oft zwischen *komplexitäts-theoretischer* und *informations-theoretischer* Sicherheit.

Bei informations-theroretischer Sicherheit benötigt das MPC-Protokoll keine Komplexitätsannahmen. Ein solches Protokoll behält seine Sicherheit also unabhängig von jeglichem technischen Fortschritt wie beispielsweise der Verfügbarkeit von universellen Quantencomputern. Leider benötigt dieser starke Sicherheitsbegriff sehr starke Annahmen, was die Anzahl der ehrlichen Parteien angeht, beispielsweise, dass die Mehrheit der Parteien unkorrumpiert bleibt[11] (*honest majority*).

Dem gegenüber steht die komplexitäts-theoretische Sicherheit, die Komplexitätsannahmen wie beispielsweise die RSA-Annahme gemacht. Gilt die Annahme nicht, geht in der Regel jegliche Sicherheit verloren. Dafür braucht komplexitäts-theoretische Sicherheit im Allgemeinen weniger Annahmen über die Anzahl der korrumpierten Parteien (es sind sogar Protokolle möglich, die in einem gewissen Sinn Sicherheit bieten, wenn *alle* Parteien korrumpiert sind). Auch kann mithilfe von Komplexitätsannahmen oftmals eine bessere Effizienz erreicht werden.

## 7.2 Definition von Sicherheit

Um die Sicherheit eines MPC-Protokolls zu analysieren ist eine präzise mathematische Definition von Sicherheit nötig. Diese umfasst beispielsweise das Maschinen- und Kommunikationsmodell, aber auch Regeln für die Ausführung (welche Entität wird wann aktiviert, wer darf mit wem reden, wie sind Ein- und Ausgaben definiert, ...).

Ein etablierter Sicherheitsbegriff für Protokolle zur sicheren Mehrparteienberechung ist das so genannte Real-Ideal-Paradigma. Dabei wird die *reale Ausführung* des Protokolls $\pi$ einer *idealen Ausführung* mit einer idealen Funktionalität $\mathcal{F}$ gegenübergestellt.

**Reale Ausführung.** In der realen Ausführung wird das Protokoll $\pi$ ausgeführt. Zuerst bekommen alle Parteien „von außen“ ihre geheime Eingabe $x_i$, woraufhin sie sich die im Protokoll $\pi$ spezifizierten Nachrichten zuschicken. Die korrumpierten Parteien werden von einem Angreifer $\mathcal{A}$ kontrolliert, der die an die korrumpierten Parteien adressierten Nachhrichten enthält und ihre Nachrichten (an ehrliche Parteien) generiert. Er kennt dabei insbesondere die Eingaben der korrumpierten Parteien. In der Regel ist der Angreifer auch für die Auslieferung der Nachrichten der ehrlichen Parteien zuständig. Er darf diese beliebig aufhalten oder unterdrücken, in der Regel aber nicht modifizieren oder neue Nachrichten „erfinden“. Am Ende der Protokollausführung geben die ehrlichen Parteien ihre Ausgaben $y_i$ aus. Auch der Angreifer hat eine Ausgabe. Ohne Beschränkung der Allgemeinheit ist dies seine *Sicht*, also sein Zufallsband, seine Eingabe sowie alle empfangenen und gesendeten Nachrichten.

[11] Falls ein Broadcast-Kanal zur Verfügung steht.

**Ideale Ausführung.** In der idealen Ausführung wird *kein* Protokoll ausgeführt. Stattdessen gibt es eine per Definition vertrauenswürdige Partei, die *ideale Funktionalität* $\mathcal{F}$ genannt wird. Die (ehrlichen) Parteien können direkt mit $\mathcal{F}$ ohne Kenntnis oder Einflussmöglichkeit des Angreifers kommunizieren und ihre Eingaben $x_i$ an $\mathcal{F}$ senden. Statt dem Angreifer $\mathcal{A}$ gibt es einen „Idealweltangreifer“, genannt *Simulator* Sim. Er sendet im Namen der korrumpierten Parteien Eingaben $x_i'$ an $\mathcal{F}$. Hat $\mathcal{F}$ alle Eingaben erhalten, wertet es die zu berechnende Funktion $f$ auf den Eingaben aus. In der Regel erhält zuerst der Simulator alle Ausgaben, die für die korrumpierten Parteien bestimmt sind. Daraufhin darf er Ausgaben für die ehrlichen Parteien unterdrücken. (Dies muss möglich sein, da der Angreifer im Realen Protokollnachrichten unterdrücken und so erzwingen kann, dass ehrliche Parteien keine Ausgabe geben.). Anschließend geben alle ehrlichen Parteien ihre Ausgaben aus. Auch der Simulator gibt eine Ausgabe aus.

Die Aufgabe des Simulators ist es dafür zu sorgen, dass die gemeinsame Verteilung aus den Ausgaben der ehrlichen Parteien und der Ausgabe des Angreifers ununterscheidbar zwischen der realen und der idealen Ausführung ist.

Wir bemerken, dass die ideale Ausführung *per Definition* sicher ist. Insbesondere ist dort garantiert, dass der Angreifer (in diesem Fall der Simulator) die Eingaben der ehrlichen Parteien nicht kennt und auch nicht manipulieren kann. Bis auf das Unterdrücken von Ausgaben und der Wahl der Eingaben der korrumpierten Parteien (die er auch nicht in Abhängigkeit der ihm unbekannten ehrlichen Eingaben wählen kann) gibt es keine Möglichkeit, Einfluss auf die ideale Ausführung zu nehmen. Sind nun die reale und die ideale Ausführung ununterscheidbar so bedeutet das, dass das reale Protokoll „so sicher“ wie die reale Ausführung ist. Könnte der Angreifer im Realen einen „Angriff“ durchführen, müsste dies auch der Simulator können, weil sonst die reale und die ideale Welt unterscheidbar wären. Per Definition sind im Idealen aber keine „Angriffe“ bis auf die genannten möglich.

**Definition 1** (Real-Ideal-Sicherheit, informell)**.** *Wir sagen, dass ein Protokoll $\pi$ sicher die Funktion $f$ berechnet, wenn für alle (Realwelt-)Angreifer $\mathcal{A}$ ein Simulator* Sim *existiert, sodass die Ausgaben der ehrlichen Parteien und die Ausgabe des Angreifers in der realen Ausführung mit $\pi$ und $\mathcal{A}$ und der idealen Ausführung mit der idealen Funktionalität $\mathcal{F}$, die $f$ berechnet, und dem Simulator* Sim*, ununterscheidbar sind.*

Real-Ideal-Sicherheit kann einfach auf probabilistische Algorithmen statt (deterministischer) Funktionen erweitert werden.

## 7.3 Protokollkomposition

Das Real-Ideal-Paradigma erlaubt es (mit Einschränkungen), kryptographische Bausteine wie beispielsweise Zero-Knowledge-Beweissysteme ideal zu definieren, diese modular in größeren Protokollen zu verwenden und später ohne Sicherheitsverlust durch ein Protokoll zu ersetzen.

Der Protokollkomposition sind jedoch Grenzen gesetzt, da *immer nur ein* Protokoll gleichzeitig ausgeführt werden darf: Man kann Protokolle angeben, die in einer einzelnen Ausführung sicher sind, aber schon bei paralleler Ausführung alle Sicherheit verlieren, siehe zum Beispiel [74].

Da in der Praxis jedoch oftmals viele kryptographische Protokolle gleichzeitig ausgeführt werden (mehrere Browser-Tabs mit TLS, verschiedene Apps und Hintergrunddienste auf Smartphones) ist es oftmals notwendig, Sicherheitsbegriffe zu betrachten, die unter Komposition abgeschlossen sind. Ein Beispiel hierfür ist die so genannte *Universal Composability*, kurz UC-Sicherheit.

Ist ein Protokoll UC-sicher, so ist garantiert, dass es seine Sicherheit in beliebigen Ausführungskontexten behält. UC-Sicherheit ist dabei sehr ähnlich wie das Real-Ideal-Paradigma definiert, erweitert dieses aber in vielerlei Hinsicht.

Wie schon beim Übergang von passiver zu aktiver Sicherheit geht der starke Begriff der UC-Sicherheit beweisbar mit bestimmten starken Annahmen einher[12], ohne die er nicht erfüllbar ist. Ein Thema der kryp-

[12]Genauer benötigen UC-sichere Protokolle ein so genanntes *Setup*, das vertrauenswürdig zur Verfügung gestellt werden muss. Typische Beispiele sind eine Public-Key-Infrastruktur, ein so genannter *Common Reference String*, der von allen Parteien ausgelesen werden kann, oder aber ein *Random Oracle*, das eine (über)idealisierte Hashfunktion darstellt. Ist das Setup nicht vertrauenswürdig gewählt, kann alle Sicherheit verloren gehen. Die Herausforderung in der Praxis besteht darin, dass es im Allgemeinen keine Entität gibt, der alle Protokollteilnehmer hinreichend vertrauen, dass sie dieses Setup zur Verfügung stellen

tographischen Forschung ist deshalb die Entwicklung von Sicherheitsbegriffen, die „fast so gut“ wie UC-Sicherheit sind, aber nur vergleichbare Annahmen wie „normale“ MPC-Protokolle benötigen.

## 7.4 Implementierungen

Es existieren verschiedene sehr effiziente Implementierungen für MPC.

- Das ABY- [53] und ABY-2.0-Protokoll-Framework [120] ermöglicht das Erstellen von sehr effizienten Protokollen, allerdings nur mit passiver Sicherheit.
- Das GAZELLE-Framework [84] spezialisiert sich auf das Training von neuronalen Netzen und bietet ebenfalls nur passive Sicherheit. Im Vergleich zu FHE ist es mehrere Größenordnungen schneller bei der Auswertung von neuronalen Netzen (Neural Network Inference).
- Das SPDZ-Framework [51, 86] und Erweiterungen wie das MP-SPDZ-Framework [85] oder das SCALE-MAMBA-Framework [133] bieten die Möglichkeit zwischen passiver und aktiver Sicherheit zu wählen. Der Preis für aktive Sicherheit ist eine etwas höhere Laufzeit. Das SPDZ-Framework [51] hat insgesamt eine Komplexität (Offline- und Online-Phase) von $O(n \cdot |C| + n^3)$ für $n$ Parteien und $|C|$ die Größe des Schaltkreises gemessen in elementaren Operationen in $\mathbb{F}_{p^k}$.
- MPyC [128] ist ein Python-Framework, das auf einfache Benutzung und Einbindung spezialisiert ist, allerdings auf Kosten der Effizienz.

könnte.

# 8 Sichere Hardware

Zur besseren Einordnung betrachten wir hier kurz sichere Hardware. Neben den klassischen, protokollbasierten Ansätzen für sichere Mehrparteienberechnungen gibt es Lösungen, die auf vertrauenswürdiger Hardware aufbaut. Diese kann „klassische“ Protokolle ergänzen [33], aber auch fast gänzlich ersetzen [118].

Hardwarebasierte Ansätze versprechen eine größere Effizienz, da oftmals nur kleine Teile der Berechnung kryptographisch gesichert werden müssen. Der Großteil der Berechnung wird stattdessen in der per Annahme vertrauenswürdigen Umgebung durchgeführt, oftmals ohne oder nur mit vernachlässigbarem Geschwindigkeitsverlust.

Dem Versprechen einer hohen Effizienz stehen jedoch meistens starke Vertrauensannahmen entgegen: Ist die Hardware bzw. die durch sie gesicherte Umgebung nicht vertrauenswürdig, ist oftmals alle Sicherheit verloren. Klassische MPC-Techniken versuchen gerade, Vertrauensannahmen mithilfe von kryptographischen Techniken zu reduzieren.

## 8.1 TPM-basierte Ansätze

Einen ersten und gleichzeitig ältesten Ansatz zu sicheren Berechnungen stellt die Verwendung eines *Trusted Platform Module* (TPM) dar, das mittlerweile auf der meisten Desktop- und Server-Hardware verfügbar ist. Ein TPM ermöglicht es, beim Start des Systems einen kryptographischen „Fingerabdruck“ der Soft- und Firmware des Systems zu erstellen, von dessen korrekter Erstellung (relativ zu bestimmten Sicherheitsannahmen) eine dritte Partei überzeugt werden kann. Dieser Fingerabdruck kann dann mit einem bekannten guten Systemzustand verglichen werden.

Ein typisches Problem bei TPM-basierten Ansätzen ist die Größe der so genannten *trusted computing base*, also der Menge von Hard- und Software, die als vertrauenswürdig angenommen werden muss: Dies umfasst in der Regel die gesamte Hardware. Diese könnte jedoch manipuliert worden sein oder auch ab Werk Sicherheitslücken erhalten. Aufgrund der Tatsache, dass gängige PC-Hardware hoch komplex ist und vielfach Kommunikation und Speicherzugriff auch am Betriebssystem vorbei erlaubt, ist dies eine sehr starke Annahme. Ein ähnliches Problem setzt sich im Software-Stack fort. Hier umfasst die Trusted Computing Base in der Regel die System-Firmware, das gesamte Betriebssystem, Systembibliotheken usw. Auch hier erscheint es unplausibel, eine korrekte Funktion ohne Sicherheitslücken anzunehmen, zumal einige Komponenten *closed source* sind.

Die Gründe für die genannten Nachteile sind jedoch gleichzeitig auch Vorteile: Durch die sehr breite Verfügbarkeit steht eine große Auswahl an Hard- und Software, die mit diesem Ansatz gesichert werden kann, zur Verfügung. Ebenso können in diesem Kontext existierende Programme ohne die Notwendigkeit einer Anpassung und mit nativer Performance wiederverwendet werden.

Mittlerweile gibt es Nachfolgetechnologien wie AMD SEV, die TPM-basierte Ansätze auf virtuelle Maschinen übertragen. Dabei sollen bei Verwendung von Virtualisierung ähnliche Sicherheitsgarantien wie bei der Ausführung auf dedizierter Hardware geboten werden. Insbesondere sollen (beispielsweise mithilfe von Speicherverschlüsselung) einzelne virtuelle Maschinen voneinander isoliert werden. Auch ein kompromittierter Hypervisor soll nicht in der Lage sein, die Ausführung einer virtuellen Maschine zu beeinflussen.

## 8.2 Sichere Enklaven

Eine Weiterentwicklung des bereits skizzierten TPM-Ansatzes stellen so genannte *sichere Enklaven* dar. Erklärtes Ziel dabei ist es, die *trusted computing base* zu reduzieren. Bei einer Enklave wird nicht mehr der gesamte Softwarestack attestiert, sondern eine wesentlich kleinere, sichere Umgebung geschaffen, in der Anwendungen ausgeführt werden können. Diese Umgebung soll selbst dann sicher sein, wenn beispielsweise das Betriebssystem unter Angreiferkontrolle ist. Dies wird vor allem dadurch erreicht, dass die Hardware weitere Sicherheitsmechanismen bereit stellt. Beispiele hierfür sind die Verschlüsselung des Hauptspeichers sowie die Unterstützung von Systemzuständen für die Unterscheidung von Ausführungen inner- und außerhalb von Enklaven. Auch hier stehen wieder Mechanismen zur Verfügung, mit denen Dritte von der Identität des ausgeführten Codes überzeugen können. Gleichzeitig wird es ermöglicht, einen sicheren Kanal in die Enklave

hinein aufzubauen. Dieser Kanal kann dann genutzt werden, um beispielsweise Geheimnisse am Betriebssystem vorbei in die Enklave zu laden, die darauf vertrauenswürdige Berechnungen durchführen kann, [113, 140, 111, 129].

Auch hier ist wieder die Notwendigkeit des Vertrauens in die Hardware notwendig, da ihre korrekte Funktionsweise in der Regel nicht überprüft werden kann. Im Gegenteil ist es so, dass aktuelle Implementierungen von Intel SGX (Intel Software Guard Extensions), der vielleicht bekannteste Implementierung von sicheren Enklaven, über eine Reihe von nicht patchbaren Sicherheitslücken verfügt. Diese erlauben es einem Angreifer mit Hardwarezugriff (wie beispielsweise einem Cloud-Provider) nicht nur, Geheimnisse aus Enklaven auszulesen. Ebenso ist es einem Angreifer möglich, über die Identität des ausgeführten Codes zu lügen. Die Sicherheit darf damit als vollständig gebrochen angenommen werden.

Weitere Nachteile sind die (momentan) geringere Performanz und die Notwendigkeit, Anwendungen auf die Ausführung anzupassen, da gerade nicht voller Zugriff auf alle Betriebssystem- und Bibliotheksfunktionen zur Verfügung steht.

Die Nachteile hinsichtlich bekannter Sicherheitslücken und Performanz sind Merkmale aktueller Implementierungen, die grundsätzlich behoben werden können. Für die Zukunft ist geplant, Open-Source-Prozessorarchitekturen (RISC V) mit sicheren Enklaven auszustatten.

# 9 Anwendungen und Use-Cases

In diesem Abschnitt werden wir nun einige Anwendungsszenarien von FHE diskutieren, ihre Alleinstellungsmerkmale sowie deren Vor- und Nachteile im Vergleich zu alternativen Ansätzen.

Generell liegt die Stärke von FHE darin, einen generischen Ansatz zu liefern, welcher Protokolle mit zusätzlichen Sicherheitsgarantien ausstattet. Genauer gesagt lassen sich mit FHE prinzipiell alle verteilten Berechnungsprobleme im Zweiparteienfall mit minimaler Interaktion und Kommunikationskomplexität sicher lösen. Dies ist das Hauptunterscheidungsmerkmal zu alternativen Ansätzen, welche entweder signifikant höhere Runden- und Kommunikationskomplexität haben oder speziell auf den Anwendungsfall zugeschnitten werden müssen.

## 9.1 Auslagern von Berechnungen auf sensiblen Daten/ Private Cloud Computing

Mittels FHE ist es möglich, Berechnungen auf den eigenen Daten an ein nicht-vertrauenswürdiges Rechenzentrum auszulagern, ohne dass dabei das Rechenzentrum Zugriff auf die vertraulichen Daten erhält (siehe Abbildung 9). Darüber hinaus ist es sogar möglich, dass selbst die auszuwertende Funktion, also der vom Server auszuführende Algorithmus geheim bleibt (siehe Abbildung 10).

Dies lässt sich wie folgt mit FHE erreichen. Ein Client mit einer Eingabe $x$ erzeugt ein Paar $(pk, sk)$ von öffentlichen und geheimen Schlüsseln und verschlüsselt $x$ unter $pk$, erhält also ein Chiffrat $c$. Dieses schickt er zusammen mit einer Funktion $f$, also einem Programm oder Algorithmus, an den Server. Dieser kann nun $f$ homomorph auf $c$ auswerten und erhält so eine Verschlüsselung $c'$ von $f(x)$. Er schickt nun $c'$ an den Client zurück, welcher $c'$ zu $f(x)$ entschlüsselt. Damit wurde die Berechnung von $f(x)$ an den Server ausgelagert, ohne dass dieser nicht-triviale Information über $x$ lernen konnte. Zu trivialer Information über $x$, welche sich nicht durch FHE verstecken lässt, gehört beispielsweise die Länge der Bitdarstellung von $x$, da die Größe des Chiffrates $c$ mit dieser skaliert.

$$\mathit{Client} \xrightarrow{c := \mathrm{Enc}(x), f(\cdot)} \mathit{Server}$$
$$\mathit{Client} \xleftarrow{\mathrm{Eval}(c, f(\cdot))} \mathit{Server}$$

Abbildung 9: Auslagern von Berechnungen auf sensiblen Daten mittels FHE.

$$\mathit{Client} \xrightarrow{c := \mathrm{Enc}(f, x)} \mathit{Server}$$
$$\mathit{Client} \xleftarrow{\mathrm{Eval}(c, \mathcal{U}(\cdot,\cdot))} \mathit{Server}$$

Abbildung 10: Auslagern von Berechnungen auf sensiblen Daten mittels FHE unter Geheimhaltung der zu berechnenden Funktion. Der Schaltkreis $\mathcal{U}$ bezeichnet den universellen Schaltkreis, der bei Eingabe eines Schaltkreises $C$ und einer Eingabe $x$ den Schaltkreis $C$ an $x$ auswertet.

Es ist zu beachten, dass FHE an sich nur *passive Sicherheit* garantiert. Dies bedeutet, dass sie nicht sicherstellt, dass eine Berechnung auch tatsächlich korrekt durch den Server ausgeführt wurde. Um eine dahingehend stärkere *aktive Sicherheit* zu erreichen, welche auch Sicherheit gegen Server bietet, die versuchen, ein inkorrektes Berechnungsergebnis abzuliefern, sind zusätzliche Techniken notwendig. Beispielsweise lassen sich derartige Sicherheitsgarantien mit *Delegation of Computation* Verfahren oder *Efficient Arguments* erreichen. Diese Verfahren verursachen allerdings selbst auch enormen Zusatzaufwand, vor allem wenn sie in Verbindung mit FHE eingesetzt werden.

Im Allgemeinen ist zunächst nicht sichergestellt, dass die Berechnung, also das Programm oder der Algorithmus, welcher auf einer geheimen Eingabe ausgeführt werden soll, vor dem Server versteckt bleibt. Um

dies zu erreichen, kann auf ein universelles Berechnungsmodell zurückgegriffen werden. Dies bedeutet schlicht nichts anderes, als den Algorithmus/das Programm zu einem Bestandteil der geheimen Eingabe zu machen. Dies lässt sich in Analogie dazu verstehen, wie herkömmliche Computer Programme ausführen: Programme sind nicht fest in der Hardware verbaut, sondern selber Daten, also Software, welche von einem Hardware-Universalrechner ausgeführt werden. Ein derartiges, für kryptografische Verfahren geeignetes universelles Berechnungsmodell ist beispielsweise das Modell der *universellen Schaltkreisen*. Diese nehmen als ihre eigene Eingabe die Beschreibung eines beliebigen boolschen Schaltkreises sowie einer Eingabe für diesen entgegen und *emulieren* die Berechnung dieses Schaltkreises auf seiner Eingabe. Durch diese Emulation entsteht allerdings wieder ein beträchtlicher Zusatzaufwand, welcher derartige Verfahren aus praktischer Sicht häufig unattraktiv macht. Genauer gesagt erzeugt ein universeller Schaltkreis für eine Berechnung mit Laufzeit $T$ einen *Slowdown*, also Effizienzverlust von einem Faktor der Größenordnung $\log(T)$. Daher ist es im Anwendungsfall vorzuziehen, keine universellen Berechnungen auszulagern, sondern nur spezielle Berechnungen. In einem konkreten Beispiel könnte man sich vorstellen, dass ein Cloudserver ein Machine-Learning Modell auf geheimen Trainingsdaten anlernen soll. Dabei ist es im Normalfall hinnehmbar, dass der Server erfährt, dass er gerade einen Lernalgorithmus ausführt, allerdings bleiben ihm die geheimen Trainingsdaten verborgen.

Die Abwesenheit derartiger Programmsicherheit kann unter Umständen auch von Vorteil sein, um Sicherheitseigenschaften *gegen* einen möglicherweise bösartigen Client zu erreichen. Beispielsweise kann ein Server dadurch verhindern oder zumindest erschweren, dass er illegale oder unerwünschte Berechnungen durchführt, wie beispielsweise dem Erstellen von *Rainbow Tables* und dem Aufbrechen von gestohlenen Passwortdatenbanken.

Aus rein theoretischer Sicht löst FHE das Problem des sicheren Auslagerns von Berechnungen vollständig. Asymptotisch gesehen verrichtet nur der Server Arbeit, und die Größe der Kommunikation skaliert nur mit der Größe der Ein- und Ausgaben. Sicheres Auslagern von Berechnungen war tatsächlich der Anwendungsfall, welcher die Erforschung von FHE Verfahren motiviert hat.

Ein wichtiger hierbei zu berücksichtigender Aspekt ist, dass im Szenario des sicheren Auslagerns von Berechnungen der Server keinerlei (geheime) Eingaben beisteuert; er stellt lediglich dem Client seine Rechenkraft zur Verfügung. Dies ist ein prinzipieller Unterschied zwischen sicherem Outsourcing und sicheren Zwei-/Mehrparteienberechnungen.

**Konkrete Beispiele**

- **Ultrathin Clients und Edge-Computing** Ein vorstellbares Anwendungsszenario von private Outsourcing sind *Ultrathin Clients*, also Endgeräte, welche praktisch über keine eigene Rechenleistung verfügen und lediglich Eingaben verschlüsseln und Ausgaben entschlüsseln können. Alle tatsächlichen Berechnungen des Gerätes würden an einen Cloud-Server ausgelagert, welcher diese homomorph verschlüsselt durchführt und das verschlüsselte Berechnungsergebnis an den Client zurückschickt, welcher dieses wiederum entschlüsselt. Das Dateisystem liegt dabei homomorph verschlüsselt auf dem Cloudserver. Weiterhin werden alle Funktionen und Programme des Endgerätes von Kommunikationsprogrammen bis hin zum Rendern eines graphischen Nutzerinterfaces homomorph durch den Cloudserver ausgeführt. Da der Cloudserver nicht vertrauenswürdig sein muss, kann, um beispielsweise die Latenz zu minimieren, dieser mittels *Edge-Computing* realisiert werden [147]. Dabei werden Cloud-Resourcen dynamisch alloziert, also beispielsweise in einer dem Endgerät räumlich naheliegenden Mobilfunkbasisstation. Dabei würde also, wenn ein Endgerät sich in eine andere Mobilfunkzelle bewegt, das verschlüsselte Dateisystem sich mit dem Endgerät auf die zugehörige Basisstation *mitbewegen* (Abbildung 11).

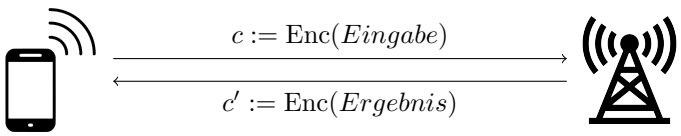



Abbildung 11: Ultrathin Clients mittels FHE.

Ein besonderer Aspekt dieses Computing-Paradigmas ist dabei, dass ein derartiger Ultrathin Client nicht programmierbar sein, also kein vollwertiger Universalcomputer sein müsste. Die Ver- und Entschlüsselungsalgorithmen könnten damit direkt in Hardware implementiert werden. Darüber hinaus bräuchte ein solches Endgerät lediglich eine Bedienschnittstelle sowie Kommunikationshardware.

Ein Vorteil dieses Paradigmas ist, dass der Ultrathin Client keine Angriffsoberfläche für Malware oder sonstige direkte physische Angriffe bietet; selbst wenn ein derartiges Gerät gestohlen oder konfisziert wird, erlangt der Angreifer keinen direkten Zugriff auf die Daten des Nutzers, da das Gerät lediglich ein Interface darstellt.

Für dieses Anwendungsszenario gibt es keine Alternativansätze zu FHE; nur vollwertige FHE erlaubt es, dass alle Berechnungen asymmetrisch auf den Cloudserver ausgelagert werden und mittels Kompaktheit über die zeitliche dynamische Entwicklung eines derartigen Systems kein serverseitiger *Ciphertext-Blowup* entsteht.

Allerdings bleibt zu bedenken, dass dieses Einsatzszenario die Möglichkeiten derzeit existierender FHE Verfahren weit überfordert.

- **Verschlüsselte Verarbeitung von Bilddaten** in einem ähnlich gelagerten Szenario kann man sich ein Kamerasystem vorstellen, welches Bilddaten homomorph verschlüsselt und an einen Bildverarbeitungsserver weitergibt. Die verarbeiteten Daten werden daraufhin an einen Entschlüsselungsserver weitergeleitet, welcher das Ergebnis der Berechnung entschlüsselt.

  In diesem Anwendungskontext ist es sinnvoll wenn das Sensormodul, also ein mit in der Kamera verbauter Rechner, bereits Vorberechnungen auf den Bilddaten vornimmt, beispielweise Segmentierung von Szenen, Objekt-Tracking sowie Gesichtserkennung. Diese Berechnungsschritte sind häufig die berechnungsintensivsten in der Bildverarbeitung, weshalb es aus Effizienzgründen Sinn macht, diese auf den Klartextbildern durchzuführen. In diesem Anwendungsfall werden damit nicht alle Berechnungen vom Client, also der Kamera ausgelagert, sondern nur solche, welche auf zusätzliche sensible Daten Rückgriff machen.

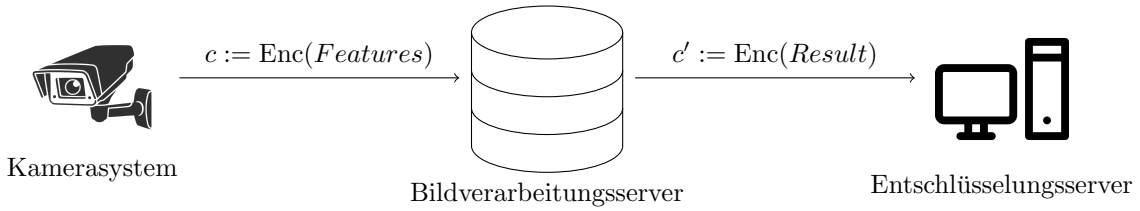



Abbildung 12: Homomorphe Bildverarbeitung mittels FHE.

Um dieses Szenario weiter zu konkretisieren, stellen wir uns vor, dass die Kamera eine Gesichtserkennung durchführt, und erkannte Gesichter auf einen kleinen Merkmalsatz reduziert (zum Beispiel in einer *Eigenface-Basis*). Diese Merkmale werden dann homomorph verschlüsselt und an den Bildverarbeitungsserver weitergegeben. Der nicht-vertrauenswürdige Bildverarbeitungsserver könnte in diesem Szenario Zugriff auf eine ebenfalls homomorph verschlüsselte Datenbank haben, in welcher Merkmale verdächtiger Person gespeichert sind. Aufgabe des Bildverarbeitungsservers wäre es, homomorph festzustellen, ob sich eine von der Kamera erkannte Person in dieser Datenbank befindet. Das Ausgabechiffrat des Bildverarbeitungsservers kann im Extremfall z.B. nur ein einziges Bit verschlüsseln, welches signalisiert, ob eine verdächtige Person detektiert wurde oder nicht. Dieses Ausgabechiffrat wird dann an einen Entschlüsselungsserver gesendet, welcher die Ausgabe entschlüsselt und entsprechend auf die so erhaltene Information eine Reaktion auslöst, beispielsweise einen menschlichen Operator über dieses Ereignis informiert (Abbildung 12).

In diesem Szenario sind die Vertrauensannahmen wie folgt verteilt:

- Die Kamera beobachtet die tatsächliche Szene, welche hierbei als geheime Eingabe betrachtet wird. Sie ist allerdings nur im Besitz des Public-Keys, und weiter *nicht* im Besitz der verschlüsselten Datenbank. Wird also lediglich die Kamera kompromittiert, so lassen sich hierdurch weder Informationen über den Secret-Key, noch über die Verdächtigendatenbank erlangen.
- Der Bildverarbeitungsserver ist lediglich im Besitz der verschlüsselten Datenbank und erhält Chiffrate von der Kamera. Er selbst hat damit direkt keine geheime Eingabe. Es sollte in diesem Szenario allerdings sichergestellt werden, dass das Ausgabechiffrat des Bildverarbeitungsservers keine Zusatzinformation über die verschlüsselten Eingaben der Kamera oder die verschlüsselte Datenbank enthält. Um dies sicherzustellen, ist ein FHE Verfahren mit Circuit-Privacy nötig. Bei einem derartigen Verfahren enthalten homomorph berechnete Ausgabechiffrate informations-theoretisch nur Information über die Ausgabe der Klartextberechnung, nicht jedoch Residualinformation über die Eingaben oder die Berechnung selbst. Wird allein der Bildverarbeitungsserver kompromittiert, erhält ein Angreifer nur verschlüsselte Information, welche für sich genommen nutzlos ist.
- Der Entschlüsselungsserver ist lediglich im Besitz des Secret-Keys, welcher als geheime Eingabe behandelt werden muss, und erhält lediglich die Ausgabechiffrate des Bildverarbeitungsservers. Wird der Entschlüsselungsserver korrumpiert, erhält ein Angreifer den Secret-Key und kann damit sowohl die verschlüsselten Ausgaben der Kamera alsauch die verschlüsselten Ausgaben des Bildverarbeitungsservers entschlüsseln. Aufgrund der oben beschriebenen Circuit-Privacy Eigenschaft lernt ein derartiger Angreifer allerdings nichts über die Datenbank. Dazu müsste noch zusätzlich der Bildverarbeitungsserver korrumpiert werden, was eine zusätzliche Hürde darstellt.

Dieses Anwendungsszenario hat sowohl Ähnlichkeit mit Private Set Intersection, speziell Private Membership Testing (Abschnitt 9.2), lässt sich aber auch als Aggregationsproblem auffassen (Abschnitt 9.3). Daher existieren für diesen Anwendungsfall ähnliche Alternativansätze zu FHE wie die dort diskutierten.

**Technology Readiness** Wir schätzen dieses Anwendungsszenario derzeit mit TRL 1/2 ein. Derzeit ist nicht absehbar in wie vielen Jahren sich dieses Anwendungsszenario kompetitiv realisieren lässt.

## 9.2 Private Set Intersection

**Private Set Intersection** Im Gegensatz zu Private Outsourcing ist Private-Set-Intersection (PSI) ein tatsächliches Zwei-/Mehrparteienberechnungsproblem. Zwei Parteien, $A$ und $B$ sind jeweils im Besitz von Datenbanken, $DB_A$ und $DB_B$. Ziel von PSI ist es, gemeinsame Datenbankeinträge in $DB_A$ und $DB_B$ zu finden, aber alle anderen Datenbankinhalte vor der jeweils anderen Partei geheim zu halten. Dabei gibt es symmetrische PSI Verfahren, bei welchen beide Parteien die gemeinsamen Einträge lernen, als auch asymmetrische Verfahren, bei welchen nur eine Partei, im Normalfall $A$, die Ausgabe lernt. Effizienzziel bei sicheren PSI Verfahren ist es gemeinhin sowohl die Kommunikationskomplexität als auch die Berechnungskomplexität für beide Parteien zu optimieren (Abbildung 13).

Aus informationstheoretischen Gründen ist es notwendig, dass die Kommunikationskomplexität mindestens mit der Größe einer der beiden Datenbanken skaliert. FHE-basierte PSI Protokolle sind vor allem dann interessant, wenn eine der beiden Datenbanken signifikant kleiner ist als die andere, im Normalfall $DB_A \ll DB_B$.

**Realisierung mittels Circuit-Private FHE** PSI lässt sich mittels Circuit-Privacy FHE wie folgt realisieren. $A$ verschlüsselt ihre komplette Datenbank $DB_A$ *eintragsweise* mittels eines FHE Verfahrens unter einem selber erzeugten Public-Key und behält den Secret-Key. Etwas genauer, ist $DB_A = \{a_1, \ldots, a_n\}$, so erzeugt $A$ Chiffrate $c_1, \ldots, c_n$, wobei $c_i$ den Datenbankeintrag $a_i$ verschlüsselt. $A$ sendet nun $c_1, \ldots, c_n$ and $B$, und $B$ führt für jedes Chiffrat $c_1$ folgende Berechnung homomorph durch: Die Funktion $f_{DB_B}(x)$ testet mittels eines linearen Scans für jeden Datenbankeintrag $b_j \in DB_B$ ob $x = b_i$. Falls ein solches $b_i$ existiert gibt sie 1 aus, ansonsten 0. Das heißt $B$ berechnet für jedes $c_i$ eine $c'_i = \text{Eval}(f_{DB_B}, c_i)$ und sendet die Chiffrate

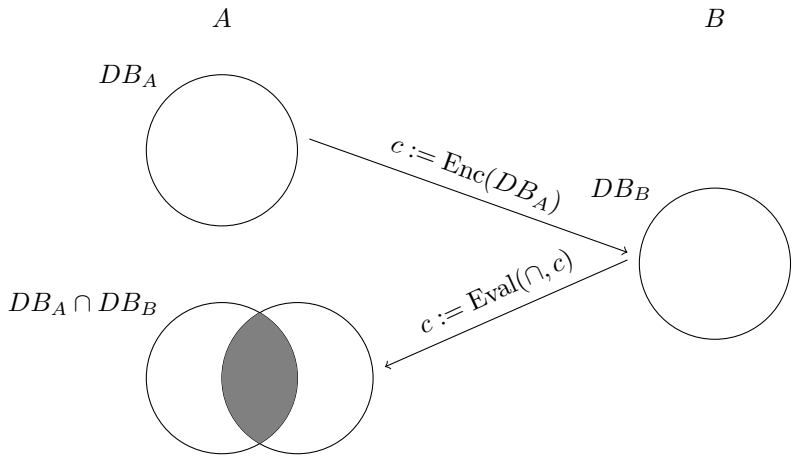



Abbildung 13: Private Set Intersection mittels FHE.

$c'_1, \ldots, c'_n$ zurück an $A$. $A$ entschlüsselt nun alle $c'_i$. Wenn $c'_i$ zu 1 entschlüsselt dann lernt sie dass $a_i \in DB_B$. Ansonsten, falls $c'_i$ zu 0 entschlüsselt gilt $a_i \notin DB_B$.

Falls die Datenbank $DB_A = \{a\}$ nur aus einem einzigen Element besteht, so spricht man auch von *Private Membership Testing*.

**Fuzzy Private Set Intersection** Fuzzy Private Set Intersection (Fuzzy PSI) ist eine Variante von PSI bei welcher nicht nur identische Datenbankeinträge identifiziert werden, sondern auch ähnliche. Ähnlichkeit wird hierbei über ein Abstandsmaß $\Delta$ charakterisiert. Dabei sind zwei Datenbankeinträge $a \in DB_A$ und $b \in DB_B$ ähnlich, falls $\Delta(a, b) < \delta$ für einen Schwellwertparameter $\delta$. Das heißt ein Fuzzy PSI filtert aus zwei Datenbanken die jeweils zueinander ähnlichen Elemente heraus. Der Nutzen von Fuzzy PSI stammt daher, dass bei vielen Anwendungsszenarien *unscharfe* Daten vorliegen, beispielsweise verschiedene, aber ähnliche Schreibweisen desselben Namens existieren oder verschiedene Scans von biometrischen Merkmalen zwar ähnlich, aber nicht identisch sind.

**Konkrete Beispiele**

- **Vergleich von Beobachtungslisten**: Ein für Private Set-Intersection vorgeschlagenes Anwendungsszenario sieht wie folgt aus. Zwei oder mehr Sicherheitsbehörden führen Listen von zu beobachtenden Subjekten, beispielsweise Personen mit vermutetem Bezug zu terroristischen Organisationen. Mittels PSI können nun besagte Sicherheitsbehörden ihrer Beobachtungslisten vergleichen. Die Sicherheitsgarantie von PSI gewährleistet nun, dass keine Behörde etwas über Subjekte lernt, welche sich in der Datenbank der anderen Behörde befinden, aber nicht in der eigenen. Damit lässt sich beispielweise sicherstellen, dass möglicherweise zu Unrecht beobachtete Personen nicht in die Datenbestände der jeweils anderen Behörde aufgenommen werden.

- **Contact Tracing** Mittels PSI lässt sich ein datensparsames verteiltes Contact-Tracing Konzept implementieren [7]. Dazu hält ein Tracing-Server eine Datenbank mit Pseudonymen aller Nutzer, bei welchen kürzlich eine Infektion festgestellt wurde. Die mobilen Endgeräte von Nutzern wechseln, wenn sich zwei Nutzer begegnen, ihre jeweiligen Pseudonyme aus. Das heißt, jeder Nutzer hat eine Liste mit Pseudonymen von anderen Nutzern, welchen er kürzlich begegnet ist. In regelmäßigen Abständen stellen Endgeräte nun PSI Anfragen an den Tracing-Server und lernen dadurch, ob einer ihrer kürzlichen Kontakte positiv getestet wurde. Durch die Sicherheit des PSI Protokolls lernt der Server nichts über die Kontakte der Nutzer, wohingegen Nutzer nichts über infizierte Nutzer lernen, welchen sie nicht begegnet sind.

- **Sichere Deduplikation von Datenbanken** Mit PSI lassen sich weiterhin verteilte Datenbanken sicher deduplizieren [1]. Nehmen wir an eine Datenbank sei auf zwei Server $A$ und $B$ verteilt. Beide Server behandeln Datenbankanfragen und Datenbankeinfügungen unabhängig voneinander. Um Speicherverschwendung zu minimieren, sollen in regelmäßigen Abständen Datensätze, welche auf beiden Servern vorkommen, durch einen der beiden Server aus seinem jeweiligen Datenbestand gelöscht werden. Allerdings kann es vorkommen, dass einer der beiden Server durch einen Angreifer korrumpiert wurde. Um die Daten auf dem jeweils anderen Server zu schützen, kann also ein PSI Verfahren eingesetzt werden, um Datensätze zu identifizieren, welche doppelt vorkommen und sie sodann von einem der beiden Server zu löschen. Durch PSI ist sichergestellt, dass selbst ein korrumpierter Server nur die gedoppelten Einträge lernt, aber nichts über die übrigen Daten auf dem jeweils anderen Server.
- **Privacy-Preserving Biometric Matching** Für das Testen auf biometrische Übereinstimmungen kann die minimale euklidische Distanz zwischen Vektoren berechnet werden. Beteiligt sind zwei Parteien: eine Datenbank mit $m$ biometrischen Samples, $(\mathbf{s_1}, \ldots, \mathbf{s_m})$, und eine Partei mit ihrem eigenen biometrischen Sample $\mathbf{d}$. Ziel ist es, die minimale Distanz von $\mathbf{d}$ zur Datenbank zu berechnen, d.h. gesucht ist $\min_{i\in\{1,\ldots,m\}} \Delta(\mathbf{s_i}, \mathbf{d})$, wobei $\Delta(\cdot,\cdot)$ ein geeignetes Abstandsmaß ist. Häufig wird die quadrierte euklidische Distanz als Abstandsmaß verwendet, also $\Delta(\mathbf{a}, \mathbf{b}) = \sum_{j=1}^{n}(a_j - b_j)^2$ für $n$-dimensionale Vektoren $\mathbf{a}$, $\mathbf{b}$. In Tabellen 1 und 2 sind die Laufzeiten und der Kommunikationsaufwand von einer Umsetzung von Biometric Matching bei verschiedenen Datenbankgrößen mit MP-SPDZ bei aktiver und passiver Sicherheit aufgeführt. Die relativ hohe Laufzeit bei kleiner Instanzgröße ist darauf zurückzuführen, dass MP-SPDZ kaum Optimierungen enthält, die vergleichbar mit FHE-Implementierungen wären [123]. Allerdings skaliert der Ressourcenverbrauch sehr gut bei steigender Datenbankgröße, was bei FHE nicht zu erwarten ist.

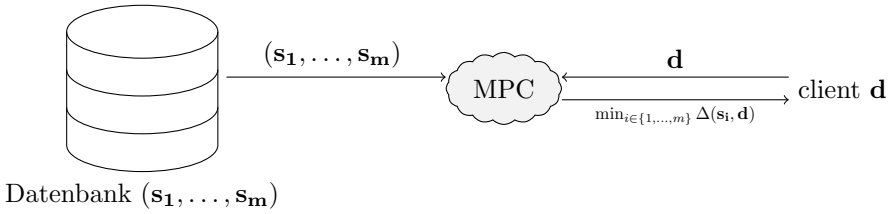



Abbildung 14: Darstellung von Privacy-Preserving Biometric Matching mittels MPC.

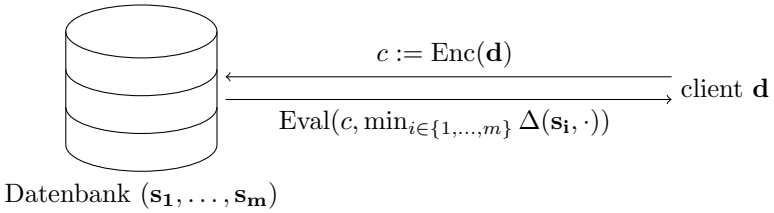



Abbildung 15: Darstellung von Privacy-Preserving Biometric Matching mittels FHE.

| Datenbankgröße | Laufzeit | gesendete Daten | Kommunikationsrunden |
|---:|---:|---:|---:|
| 2 | 355 s | 12510 MB | 23253 |
| 100 | 401 s | 12511 MB | 23353 |
| 1000 | 422 s | 12861 MB | 23647 |
| 10000 | 490 s | 16072 MB | 26175 |

Tabelle 1: Laufzeiten für Biometric Matching mit MP-SPDZ bei aktiver Sicherheit ohne Optimierungen in einer virtualisierten Umgebung mit 2 Kernen bei 3 GHz.

| Datenbankgröße | Laufzeit | gesendete Daten | Kommunikationsrunden |
|---|---|---|---|
| 2 | 1.1 s | 39.6 MB | 198 |
| 100 | 1.1 s | 39.8 MB | 294 |
| 1000 | 1.8 s | 75 MB | 350 |
| 10000 | 15.9 s | 726 MB | 2061 |

Tabelle 2: Laufzeiten für Biometric Matching mit MP-SPDZ bei passiver Sicherheit ohne Optimierungen in einer virtualisierten Umgebung mit 2 Kernen bei 3 GHz.

**Vergleich mit alternativen Verfahren** Für PSI existieren zahlreiche alternative Protokolle, welche nicht auf FHE basieren, und diese sind im Normalfall um Größenordnungen effizienter als FHE-basierte Protokolle [119, 122]. Eine Ausnahme stellt der bereits oben diskutierte Extremfall dar: Falls $DB_B \ll DB_A$, die Datenbank $DB_B$ also weitaus größer ist als $DB_A$, so kommen die Stärken von FHE zum Zuge, insbesondere dass der Berechnungsaufwand asymmetrisch verteilt ist, das heißt rechenintensive Operationen nur von $B$ durchgeführt werden müssen, und dass weiter die Kommunikationskomplexität nur mit der Größe der relativ kleineren Datenbank $DB_A$ skaliert.

**Technology Readiness** Wir schätzen dieses Anwendungsszenario derzeit mit TRL 3 ein. Es existieren prototypische Implementierungen welche bis zu einem gewissen Skalierungsgrad ausgebaut im Rahmen derzeit existierender Verfahren ausgebaut werden können.

## 9.3 Private Data Aggregation

In vielen Anwendungen müssen die Daten einzelner Nutzer zusammengeführt werden, ohne dass die Nutzer ihre Daten offenlegen. Oftmals gibt es eine zentrale Einheit, welche die Daten aller Nutzer empfängt und die Aggregation durchführt. Es gibt aber auch Szenarien, in denen kein zentraler Server zur Verfügung steht und die Teilnehmer nur durch Kommunikation untereinander die Daten zusammenführen.

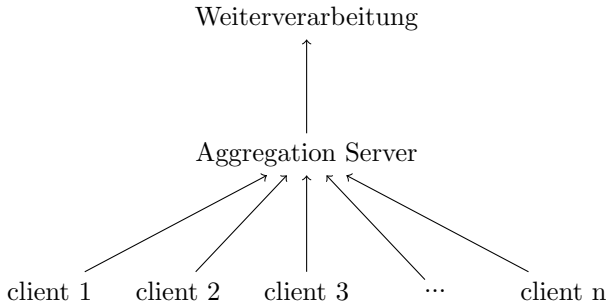



Abbildung 16: Mehrere Nutzer geben private Daten an einen zentralen Server weiter.

**Realisierung mittels FHE** Private Data Aggregation mit einem zentralen Server lässt sich durch FHE lösen. Der für die Weiterverarbeitung zuständige Server generiert die Schlüssel für das FHE Verfahren und veröffentlicht den für die Verschlüsselung benötigten Public-Key. Die Teilnehmenden verschlüsseln ihre privaten Daten und senden sie an den Aggregationsserver. Dort werden die Daten durch eine homomorphe Berechnung zusammengeführt und an den für die Weiterverarbeitung zuständigen Server weitergeleitet. Bei großen Netzwerken gibt es möglicherweise mehrere Stufen zur Aggregation der Daten. Der Weiterverarbeitungsserver entschlüsselt die aggregierten Daten.

**Konkrete Beispiele**

- **Smart Grid Networks**

  Ein Smart Grid ist ein elektrisches Netzwerk, welches Daten nutzt, um die Stromversorgung intelligent zu steuern [103]. Die Stromerzeugung und der Stromverbrauch können stark schwanken, wodurch es zu einem Überschuss oder Mangel an Energie im Stromnetz kommen kann. Ein Smart Grid steuert die Produktion und Speicherung des Stroms, um eine stabile Versorgung zu gewährleisten. In einem Smart Grid können E-Autos dann laden, wenn ein Stromüberschuss vorhanden ist und Strom abgeben, wenn erhöhter Strombedarf besteht. Auch andere private oder industrielle Endnutzer können die Nutzung von Geräten mit hohem Stromverbrauch auf das Stromnetz anpassen. Aus den dafür benötigten Daten können potenziell sensitive Informationen hergeleitet werde, zB wann ein Nutzer zu Hause ist [87]. Um dies zu vermeiden, werden Daten von lokalen Netzwerken gesammelt und aggregiert weiter gegeben. Damit die Daten auch vor dem für die Aggregation zuständigen Gerät geschützt sind, können die Daten mittels FHE verschlüsselt werden und dann homomorph aggregiert werden. Tonyali et. al [137] implementieren zwei solche Verfahren und testen diese in einem AMI(Advanced Metering Infrastructur) Netzwerk. In der Arbeit wird ein additiv homomorphes Verschlüsselungssystem und ein FHE Verfahren gegenübergestellt und insbesondere die Größe der verschlüsselten Nachricht analysiert. Tabelle 3 zeigt, dass bei dem additiven Verfahren nur ein geringer Overhead entsteht, wobei bei der Nutzung von FHE wesentlich größere Nachrichten entstehen.

| Geräte | FHE | PHE |
|---|---|---|
| 6 | 69092.8 | 15316.0 |
| 8 | 75171.2 | 19930.1 |
| 10 | 81460.9 | 24514.5 |
| 12 | 87459.3 | 29116.3 |
| 14 | 93678.9 | 33694.9 |

Tabelle 3: Jedes der Geräte im Netzwerk verschlüsselt mehrere 16 bit Nachrichten und schickt sie an einen zentralen Punkt. In der Tabelle ist die durchschnittliche Größe des resultierenden Geheimtextes in bit angegeben.

- **Federated Machine Learning**

  Beim Federated Machine Learning haben mehrere Parteien Daten, mit deren Hilfe ein Machine Learning Modell trainiert werden soll. Um die Daten nicht mit den anderen Parteien teilen zu müssen, werden die Modellparameter von jeder Partei auf Basis der eigenen Daten aktualisiert. Um die lokal trainierten Modelle zu aggregieren, kann entweder ein zentraler Server oder direkte Kommunikation zwischen den Teilnehmern genutzt werden. Diese Methode garantiert nicht die Geheimhaltung der Daten. Aufgrund der Modellparameter lassen sich potenziell Rückschlüsse auf die verwendeten Daten ziehen. Sichere Mehrparteienberechnungsverfahren können genutzt werden, um die Geheimhaltung der Daten zu garantieren. Verschiedene Arbeiten nutzen FHE zu diesem Zweck. [138, 135, 130]. Bei diesem Ansatz stellt sich die Frage, wer die Schlüssel des FHE Verfahren generiert und somit die Möglichkeit hat, alle Daten zu entschlüsseln. Ma et. al [101] löst dieses Problem, indem ein Multikey FHE (siehe 6.4) Verfahren verwendet wird. Alle Nutzer sind im Besitz ihrer eigenen Schlüssel, aber die verschlüsselten Daten können noch homomorph verarbeitet werden. Um die resultierenden Daten nutzen zu können, müssen alle Nutzer dazu beitragen, den Ciphertext zu entschlüsseln.

Andere Ansätze zum Lösen des Private Data Aggregation Problems sind Differential Privacy und Functional Encryption. Dabei gilt es zu beachten, dass diese Methoden andere Sicherheitsgarantien liefern als FHE.

**Technology Readiness** Wir schätzen dieses Anwendungsszenario derzeit mit TRL 2/3 ein. Es gibt einige prototypische Implementierungen. Allerdings sind diese derzeit auf eher geringe Eingabegrößen und Laufzeiten beschränkt.

## 9.4 Privacypreserving Machine Learning (PPML)

Unter Privacypreserving Machine Learning versteht man das Auslagern von Machine Learning bzw. gemeinsames Machine-Learning-Training. Um aufwendiges Machine Learning auf sensiblen Daten auszulagern, verschlüsselt der Dateninhaber die (Trainings-)Daten und lagert das Training eines neuronalen Netzes auf einen Serviceanbieter aus. Dieser berechnet homomorph die Modellparameter für das neuronale Netz, ohne etwas über die Daten oder die Modellparameter zu erfahren, und liefert die Modellparameter verschlüsselt dem Dateninhaber zurück, der diese entschlüsseln kann. Siehe Abbildung 17 für eine Darstellung von kooperativem Training und Abbildung 18 für eine Darstellung von ausgelagertem Training.

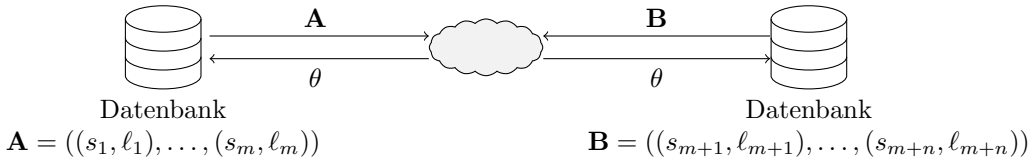



Abbildung 17: Darstellung von privacy-preserving Machine Learning.

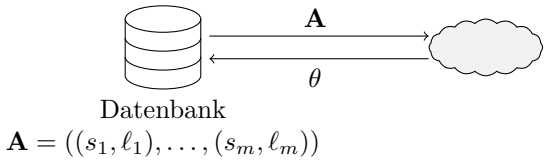



Abbildung 18: Darstellung von ausgelagertem Machine Learning.

**Konkrete Beispiele**

- **Test mithilfe des MNIST-Datensatzes** Auf einer Abwandlung des MNIST-Benchmark-Datensatzes für handgeschriebene Zahlenerkennung (bestehend aus 60.000 Trainingsdatensätzen und 10.000 Testdatensätzen, jeweils 28x28 Graustufenbilder) hatte die HELib-Bibliothek trotz Optimierungen eine Laufzeit, die um einen Faktor $10^6$ langsamer war als das direkte Ausführen auf unverschlüsselten Daten [112]. Die erreichte Sicherheit betrug 80 Bit. Da Machine-Learning meist nur auf sehr großen Datenmengen zum Einsatz kommt und die Berechnungskomplexität aus diesem Grund ohnehin sehr hoch ist, ist die auf diese Weise durch FHE entstehende Berechnungskomplexität wenig praktikabel.

  Eine Implementierung mit CKKS könnte aufgrund der besseren Unterstützung von Gleitkommazahlen, was mit einer effizienteren Auswertung der Sigmoid-Funktion einhergeht, zu besseren Ergebnissen führen.

  Interessante Anwendungsfälle beinhalten beispielsweise Machine-Learning auf gesammelten Patientendaten verschiedener Krankenhäuser oder auf Genmaterial zum Zweck wissenschaftlicher Studien.

## 9.5 Neuronal Network Inference (Machine-Learning-as-a-Service, MLaaS)

MLaaS ermöglicht es Daten auf einem neuronalen Netz zu klassifizieren, welches dem Besitzer der Daten selbst nicht vorliegt. Der Besitzer der Daten hat ein Interesse daran die Daten geheim zu halten und der Anbieter des neuronalen Netzes möchte dieses nicht preisgeben. Siehe Abbildung 19 für eine Darstellung.

Das CryptoNets-Projekt [34] löst dieses Problem mittels FHE. Hier ist es wichtig anzumerken, dass es lediglich um die Auswertung eines bereits trainierten neuronalen Netzes geht, was im Vergleich zu dessen Training erheblich effizienter ist. Der Nutzer verschlüsselt seine Daten und das neuronale Netz kann homomorph ausgewertet werden. Dies garantiert allerdings nicht die Geheimhaltung des neuronalen Netzes.

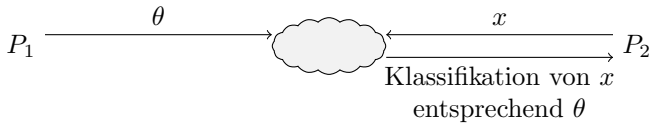



Abbildung 19: Darstellung von MLaaS.

Es ist nicht auszuschließen, dass die ausgewerteten Ciphertexte Rückschlüsse auf die Parameter und Architektur des neuronalen Netzes schließen lassen. Dies könnte durch die Verwendung eines FHE Schemas mit Circuit-Privacy sicher gestellt werden. Das CryptoNets-Projekt, das intern die FHE-Implementierung der SEAL-Bibliothek [106] benutzt, hat bei der Auswertung eines auf dem MNIST-Datensatz [90] trainierten neuronalen Netzes eine End-to-End-Latency von 297 Sekunden, wohingegen das GAZELLE-Protokoll eine End-to-End-Latency von lediglich 30 Millisekunden hat [84]. Zum Vergleich siehe die Benchmarks zum Auswerten von neuronalen Netzen mittels Concrete in Tabelle 4.[13]

| | Laufzeit | Genauigkeit |
|---|---|---|
| NN-20 | $0,17$ ms | $97,5\%$ |
| NN-50 | $0,20$ ms | $95,4\%$ |
| NN-100 | $0,33$ ms | $95,2\%$ |

| | Laufzeit | Genauigkeit |
|---|---|---|
| NN-20 | $30,04$ s | $97,4\%$ |
| NN-50 | $71,71$ s | $95,1\%$ |
| NN-100 | $108,73$ s | $91,9\%$ |

| | Laufzeit | Genauigkeit |
|---|---|---|
| NN-20 | $115,52$ s | $97,5\%$ |
| NN-50 | $233,55$ s | $95,4\%$ |
| NN-100 | $481,61$ s | $90,5\%$ |

Tabelle 4: Dauer der Auswertung neuronaler Netze der Größe 20, 50 und 100 im Klartext (links), mit FHE und 80 Bit Sicherheit (mitte) und mit FHE und 128 Bit Sicherheit (rechts) auf einem PC mit $2,6$ GHz 6-Core Intel® Core™ i7 Prozessor, siehe [43]. Bei diesen Zahlen geben wir jedoch zu bedenken, dass es sich um hochgradig optimierte Werte handelt und sie daher nicht direkt mir naiven Implementierungen vergleichbar sind.

**Technology Readiness** Wir schätzen dieses Anwendungsszenario derzeit mit TRL 2/3 ein. Derzeitige Implementierungen bewältigen eher Instanzen von Beispielgröße und es sind weitere Innovationen nötig um diese Verfahren auf Instanzen von praktisch relevanter Größe auszuführen.

## 9.6 Private Information Retrieval, Private Database Queries und Private Databases mit Schreibzugriffen

**Private Information Retrieval** Private Information Retrieval (PIR) ermöglicht es einem Client Einträge einer Datenbank eines anzufragen. Dabei kann die Datenbank entweder auf einem einzelnen Server liegen oder auf mehreren Servern verteilt sein. Ziel hierbei ist es, dass keiner der Server erfährt, welchen Eintrag der Client angefragt hat. Zudem soll die Größe der von Server an den Client übertragenen Daten unabhängig von der Größe der Datenbank sein und lediglich von der Größe eines Eintrages abhängen, siehe Abbildung 20.

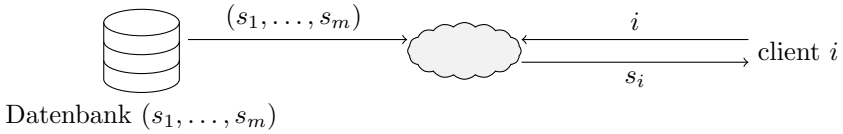



Abbildung 20: Darstellung von Private Information Retrieval.

Klassische Ansätze für PIR, wie z.B. [115], verwenden mehrere Server, die per Vertrauensannahme nicht alle zusammen korrupiert sind.

[13] Wir geben hier zu bedenken, dass es sich um hochgradig für diesen Benchmark optimierten Code handelt.

Betrachtet man den Fall mit nur einem Server, so werden hier oft Ansätze mittels FHE verwendet, die aufgrund der Nicht-Interaktivität einer Schaltkreisauswertung eine geringere Kommunikationskomplexität haben als Alternativen mittels MPC. Zwei solche Ansätze PIR mittels FHE durchzuführen werden in [146, 55] vorgestellt.

In Tabellen 5 und 6 sind Benchmnarks für eine naive Implementierung mit dem MP-SPDZ-Framework gegeben.

| Datenbankgröße | Laufzeit | gesendete Daten | Kommunikationsrunden |
|---:|---:|---:|---:|
| 10 | 0.9 s | 39.6 MB | 340 |
| 100 | 0.96 s | 39.7 MB | 1780 |
| 1000 | 1.5 s | 40.7 MB | 16180 |
| 10000 | 14 s | 379 MB | 161729 |

Tabelle 5: Laufzeiten für Private Information Retrieval mit MP-SPDZ bei passiver Sicherheit ohne Optimierungen in einer virtualisierten Umgebung mit 2 Kernen bei 3 GHz.

| Datenbankgröße | Laufzeit | gesendete Daten | Kommunikationsrunden |
|---:|---:|---:|---:|
| 10 | 390 s | 12479 MB | 23129 |
| 100 | 391 s | 12480 MB | 24569 |
| 1000 | 399 s | 12597 MB | 39073 |
| 10000 | 425 s | 13602 MB | 184137 |

Tabelle 6: Laufzeiten für Private Information Retrieval mit MP-SPDZ bei aktiver Sicherheit ohne Optimierungen in einer virtualisierten Umgebung mit 2 Kernen bei 3 GHz.

**Geheime Datenbankanfragen** Ähnlich wie bei PIR geht es hierbei darum, Datenbankanfragen an einen oder mehrere Server zu stellen und entsprechende Datenbankeinträge als Antwort zu erhalten, ohne dass der Server die Anfrage des Clients lernt. Anders als bei PIR jedoch werden Anfragen nicht durch den Index des gesuchten Datenbankeintrages, sondern durch eine Bedingung an den Eintrag selbst, siehe Abbildung 21 für eine Darstellung.

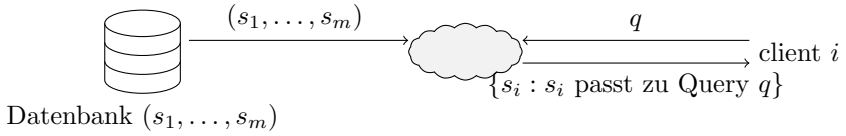



Abbildung 21: Darstellung von geheimen Datenbankabfragen, wobei $q$ eine Datenbankanfrage ist.

[18, 63] stellen einen Ansatz vor, wie Datenbankanfragen der Form `SELECT * FROM db WHERE a1=v1 AND a2=v2 AND ...` beantwortet werden können. Der erste der beiden Ansätze liefert nicht direkt die passenden Einträge der Datenbank, sondern deren Indizes, sodass eine weitere PIR-Anfrage benötigt wird, um die passenden Einträge der Datenbank zu erhalten. Auswertung von verteilten Datenbankinhalten: haben zwei oder mehr Parteien eine eigene Sammlung von (beispielsweise medizinischen) Daten, so ist es möglich auf diesen Daten eine Analyse durchzuführen, ohne diese zu teilen. Ein nützliches Beispiel ist die Erforschung von (neuen) Krankheiten. Sammeln mehrere Institute unabhängig voneinander Daten von gesunden und kranken Individuen, so kann mittels MPC oder FHE herausgefunden werden, welche dieser Werte bei der erkrankten Kohorte statistisch signifikant von der gesunden Kohorte abweicht. In Tabellen 7 und 8 sind Benchmarks für eine naive Implementierung des Mann-Whitney-U-Tests zum Finden statistisch signifikanter Abweichungen mit dem MP-SPDZ-Framework gegeben.

| Datenbankgrößen (Partei 1, Partei 2) | Laufzeit | gesendete Daten | Kommunikationsrunden |
|---|---|---|---|
| $5 + 5$ | 701 s | 23309 MB | 45367 |
| $10 + 10$ | 808 s | 23484 MB | 45743 |
| $25 + 25$ | 887 s | 26217 MB | 48303 |
| $50 + 50$ | 1930 s | 61207 MB | 110075 |

Tabelle 7: Laufzeiten für die Auswertung des Mann-Whitney-U-Tests mit MP-SPDZ bei aktiver Sicherheit ohne Optimierungen in einer virtualisierten Umgebung mit 2 Kernen bei 3 GHz.

| Datenbankgrößen (Partei 1, Partei 2) | Laufzeit | gesendete Daten | Kommunikationsrunden |
|---|---|---|---|
| $5 + 5$ | 1.6 s | 59 MB | 528 |
| $10 + 10$ | 2.6 s | 98 MB | 861 |
| $25 + 25$ | 11.3 s | 401 MB | 2720 |
| $50 + 50$ | 39.2 s | 1503 MB | 8674 |

Tabelle 8: Laufzeiten für die Auswertung des Mann-Whitney-U-Tests mit MP-SPDZ bei passiver Sicherheit ohne Optimierungen in einer virtualisierten Umgebung mit 2 Kernen bei 3 GHz.

**Private Databases mit Schreibzugriffen** In den oben diskutierten Settings werden die Datenbanken selbst nicht verschlüsselt, sondern nur zur Beantwortung verschlüsselter Anfragen verwendet. Dies ist anders gelagert im folgenden Szenario. Hierbei liegt eine Datenbank $DB$ homomorph verschlüsselt auf einem Server $S$. Nur der legitime Besitzer, welcher im Besitz des zugehörigen Secret-Keys ist, kann geheime Datenbank-Leseanfragen wie oben diskutiert stellen. Allerdings wird in diesem Szenario einer beliebigen weiteren Anzahl von Clients Schreibzugriff auf die Datenbank gewährt. Speziell kann ein Client $C$ dem Server $S$ einen verschlüsselten Schreibzugriff $f$ senden. $S$ wertet $f$ homomorph auf der ihm vorliegenden Verschlüsselung $c_{DB}$ der Datenbank $DB$ aus und erhält ein neues Chiffrat $c_{DB'}$, welches die neue Datenbank $DB'$ nach dem Schreibzugriff verschlüsselt. Dabei lernt der Server weder etwas über die neue Datenbank $DB'$ noch über den Schreibzugriff $f$.

Diese Anwendung unterscheidet sich von den vorhergehenden darin, dass der Server $S$ eine durchgehend verschlüsselte Datenbank bereithält und neue Updates in die Datenbank einpflegt. Für diesen Anwendungsfall ist FHE essenziell. Alternative Ansätze führen zu nicht hinnehmbaren Nachteilen. Nicht-kompakte Techniken wie Garbled Circuits würden dazu führen, dass die verschlüsselte Datenbank bei jedem Schreibzugriff anwächst und damit anschwillt, bis Datenbankanfragen nicht mehr effizient beantwortet werden können. Weiterhin ist hierbei auch der asymmetrische Aufwand von FHE essenziell, speziell, dass bei Lese- und Schreibanfragen nur der Server $S$ Berechnungen, welche mit der Größe der Datenbank skalieren durchführen muss, nicht aber die (möglicherweise schwachen) Clients.

Weiterhin ist bei dieser Anwendung der Einsatz eines FHE Verfahrens mit hoher Rate essenziell [31]. Ein derartiges Verfahren stellt sicher dass die verschlüsselte Datenbank nur unwesentlich größer ist als die Klartextdatenbank. Herkömmliche FHE Verfahren haben Ciphertextexpansionsraten in den Größenordnungen 100.000 - 1.000.000. Konkret würde dies bedeuten dass eine Klartextdatenbank der Größe 1 Gigabyte nach Verschlüsselung 1 Petabyte groß würde, eine extrem unpraktisch große Datenmenge.

**Technology Readiness** Wir schätzen dieses Anwendungsszenario derzeit mit TRL 3 ein. Es existieren Beispielimplementierungen und das Anwendungsszenario passt sehr gut zu den Stärken von FHE. Es sind aber in den nächsten 5-10 Jahren noch größere wissenschaftliche Innovationen zu erwarten, daher könnte es unter Umständen nicht sinnvoll sein Anwendungen mit derzeit existierenden Verfahren zu realisieren.

# 10 Benchmarks

Um die Praxistauglichkeit von FHE Anwendungen zu prüfen, implementieren wir einen FHE-Estimator. Dieser berechnet eine Vorhersage für die Laufzeit eines Programms, wenn es innerhalb eines FHE Schemas ausgeführt würde. Mithilfe eines Compilers wird ein Programm in einen Schaltkreis umgewandelt. Statt den Schaltkreis tatsächlich auszuführen, berechnen wir die zu erwartende Laufzeit. Da diese schnell sehr hoch wird, kann man so die Laufzeit potenzieller Anwendungen vorab einschätzen und abwägen, ob eine Implementierung sich lohnt.

Im FHE-Estimator Tool können einfache C++ Programme eingegeben werden. Nicht die komplette C++ Programmiersprache steht zur Verfügung, sondern lediglich die Basisfunktionen, wie Addition, Multiplikation, Schleifen und Vergleichsoperatoren. Diese werden mit Hilfe des Compilers Cingulata in Schaltkreise umgewandelt. Basierend auf der durchschnittlichen Laufzeit einzelner Operationen können wir für einen gegebenen Schaltkreis die voraussichtliche Laufzeit bestimmen. Um dies zu testen haben wir kurze Programme in Schaltkreise umgewandelt und ausgeführt. Grafik 22 zeigt die tatsächliche Laufzeit, ausgeführt mit der in Cingulata integrierten TFHE Bibliothek, und die von uns berechnete Laufzeit. Die Tests wurden auf einem handelsüblichen Laptop ausgeführt.

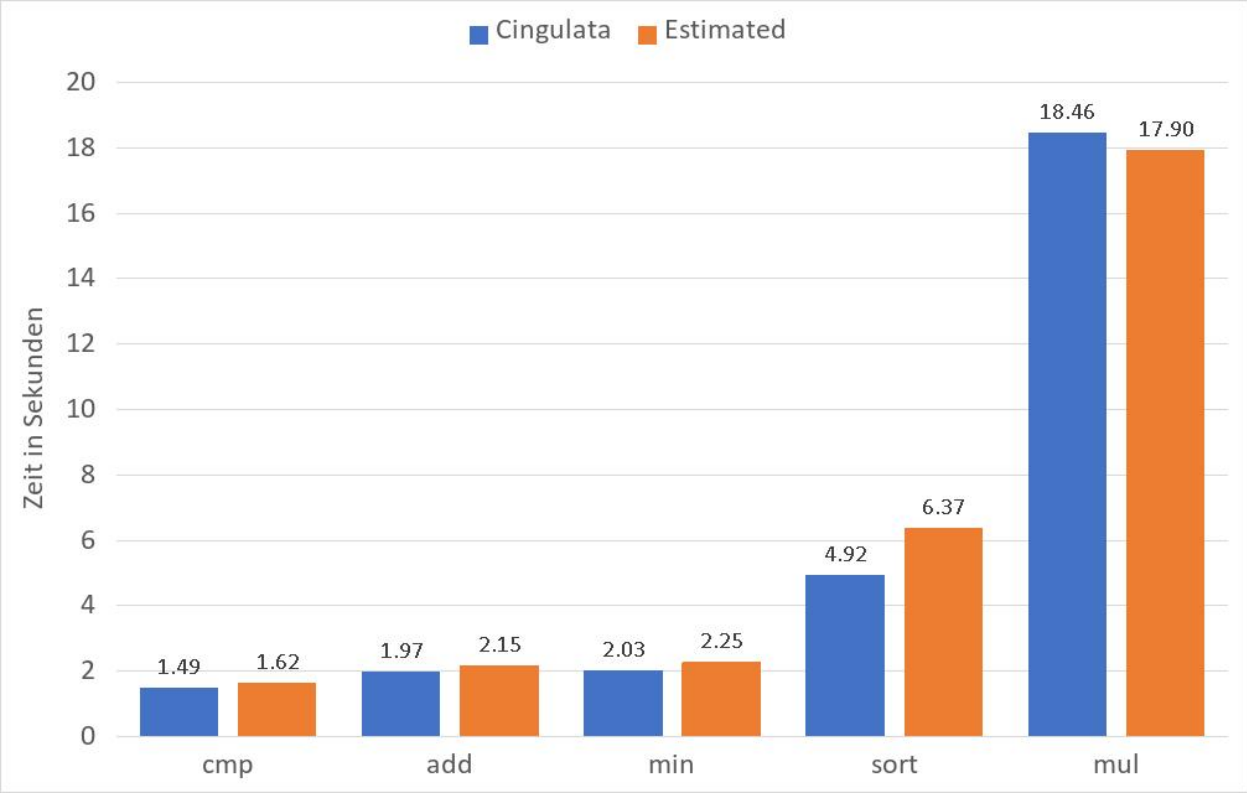



Abbildung 22: Wir haben folgende Programme als Test ausgeführt, wobei jede Anwendung mit je 4 8-bit Integers ausgeführt wurde: cmp - Vergleiche zwischen Zahlen, add - Addition, min - Minimum aus einer Liste an Zahlen, sort - sortiert 4 Zahlen, mul - multipliziert 4 Zahlen

Außerdem ist zu beachten, dass man beim Erstellen eines Circuit schon die maximale Größe der Zahlen kennen muss, die bei der Berechnung entstehen können. Je nach der Länge der Zahlen kann der Circuit sehr viel mehr Gates benötigen. In Grafik 23 wird das gleiche Programm für verschieden lange Integers compiliert.

Interessante Anwendungen für die Ausführung mit FHE sind Private Information Retrieval und Private Set Intersection. Wir haben beide Anwendungen im FHE-Estimator Tool implementiert und für verschiedene

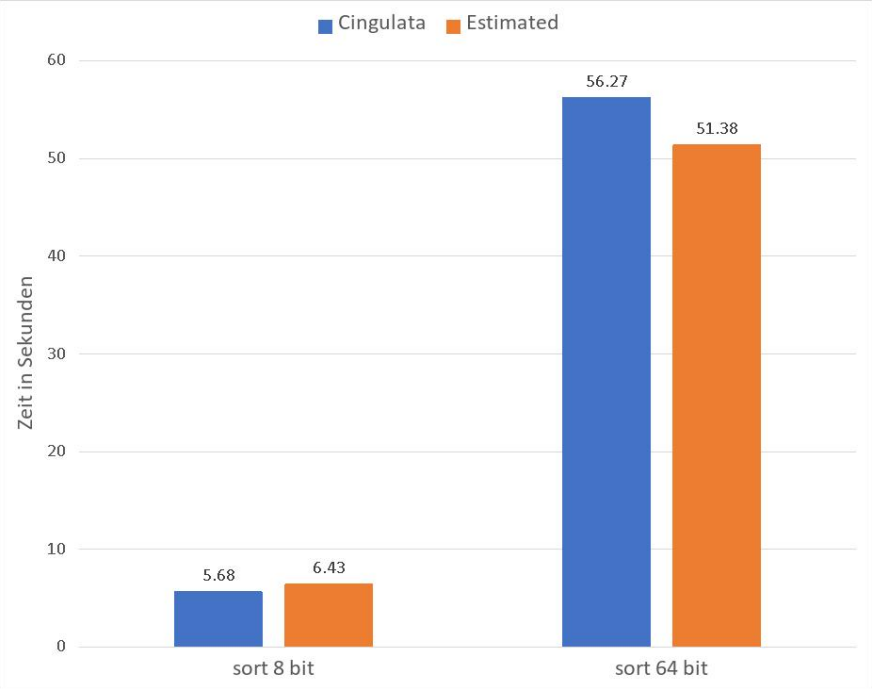



Abbildung 23: Wir haben das sort Programm mit 8-bit Zahlen und mit 64-bit Zahlen ausgeführt. Die Grafik zeigt die Zeit der tatsächlichen Ausführung (Cingulata) und die vorausgesagte Zeit (Estimated) in Sekunden.

Datenbankgrößen Schätzungen ausgeführt. Der FHE-Estimator ist auf Git zugänglich und kann genutzt werden um weitere Tests durchzuführen.

## 10.1 Private Set Intersection Benchmark

Bei Private Set Intersection werden gleiche Elemente in 2 Datenbanken gesucht. Wir haben für verschiedene Größen von Datenbank 1 und Datenbank 2 Vorhersagen erstellt und graphisch dargestellt. Grafik 24 zeigt die voraussichtlichen Laufzeiten von Datenbanken mit 10 bis 1000 Einträgen. Bei der kleinsten Größe (jede Datenbank hat 10 Einträge) beträgt die Laufzeit nur 28 Sekunden. Bei zwei Datenbanken mit jeweils 1000 Einträgen beträgt die Laufzeit etwas mehr als 3 Tage.

Ein möglicher Anwendungsfall für FHE bei Private Set Intersection besteht, wenn die vorhandenen Datenbanken unterschiedlich groß sind. Zum Beispiel liegt ein kleiner Datensatz bei einem Client vor und ein größerer Datensatz auf einem Server. Dann kann der Server die laufzeitintensive Berechnung durchführen. Grafik 25 zeigt Laufzeiten für dieses Szenario.

## 10.2 Private Information Retrieval Benchmark

Bei Private Information Retrieval wird ein Element in einer Datenbank gesucht. Grafik 27 zeigt die Vorhersagen für verschiedene Datenbankgrößen. Bei einer Datenbank von bis zu 100 Elementen befindet sich die Laufzeit unter 30 Sekunden, bei 1000 Elementen sind es knapp 5 Minuten und bei einer Größe von 100000 Elementen wird eine Laufzeit von 26810 Sekunden, also etwa 7 Stunden berechnet.

## 10.3 FHE und Garbled Circuits im Vergleich

Im Folgenden geben wir die Laufzeiten der möglichen Anwendungen ausgeführt mit FHE sowie mit Garbled Circuits wieder. Es ist zu beachten, dass dies dennoch kein direkter Vergleich ist, da in verschiedenen

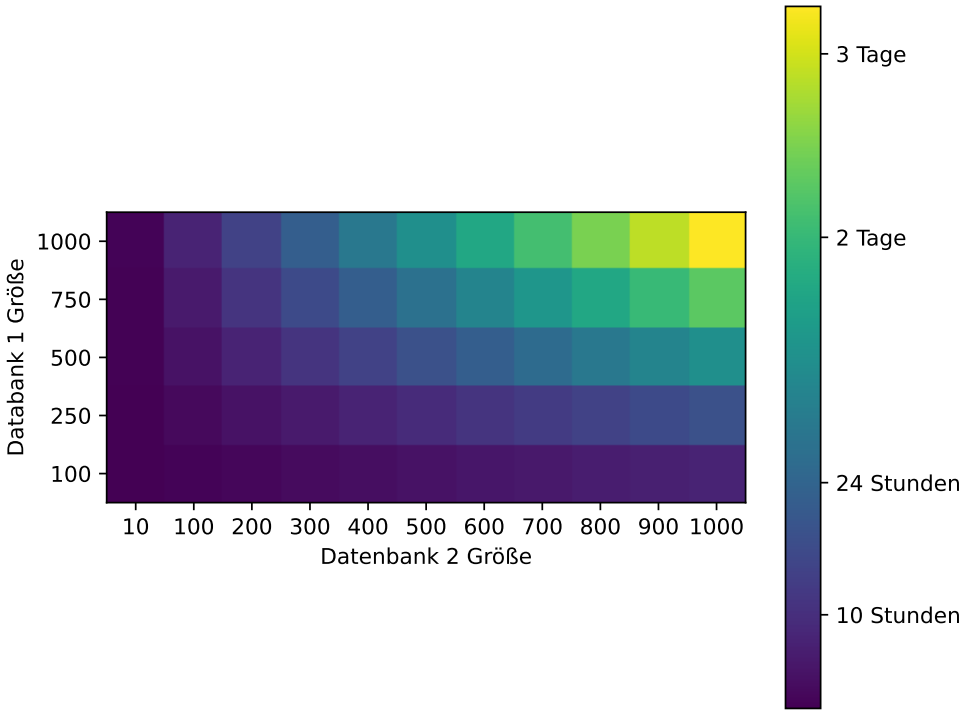



Abbildung 24: Test der Private Set Intersection Anwendung mit variablen Größen der beiden Datenbanken.

Veröffentlichungen zu den beiden Methoden nicht dieselben Benchmarks ausgeführt worden sind. Trotzdem lässt sich beim Vergleich ähnlich großer Instanzen die Laufzeitdifferenz erkennen.

| Anwendung | FHE | Garbled Circuits |
|---|---|---|
| MLaaS (Neuronales Netz der Größe 100) | 100-400s | 0.38-2.27s[12] |
| PIR (Datenbank der Größe 1000) | 200-400s | $< 1$s |
| PSI (Datenbanken der Größe je 100) | 44min | 200-400s[81] |

Tabelle 9: Die Werte der FHE Spalte sind von unserem Estimator Tool und aus [86] entnommen. Die Werte der Garbled Circuit Spalte sind aus den jeweils zitierten Papieren entnommen. Die Varianz der Werte innerhalb eines Anwendungsfalles kommt durch die Ausführung mit verschiedenen Sicherheitsparametern.

# 11 Entscheidungshilfen

## 11.1 Entscheidungshilfe für MPC vs. FHE

Im Folgenden versuchen wir einen groben Entscheidungsleitfaden anzugeben, der eine Entscheidung zwischen der Verwendung von MPC und FHE erleichtern soll. Wir beziehen die folgenden Kriterien ein:

1. *Welche Funktion soll realisiert werden?*
   FHE eignet sich dafür die Klasse SFE mit einseitiger Output-Delivery zu realisieren. Bei Funktionen, die nicht in dieser Klasse liegen, ist FHE alleine nicht anwendbar weswegen MPC zu verwenden ist.

2. *Wie viele Parteien sind an dem Szenario gleichzeitig beteiligt?*

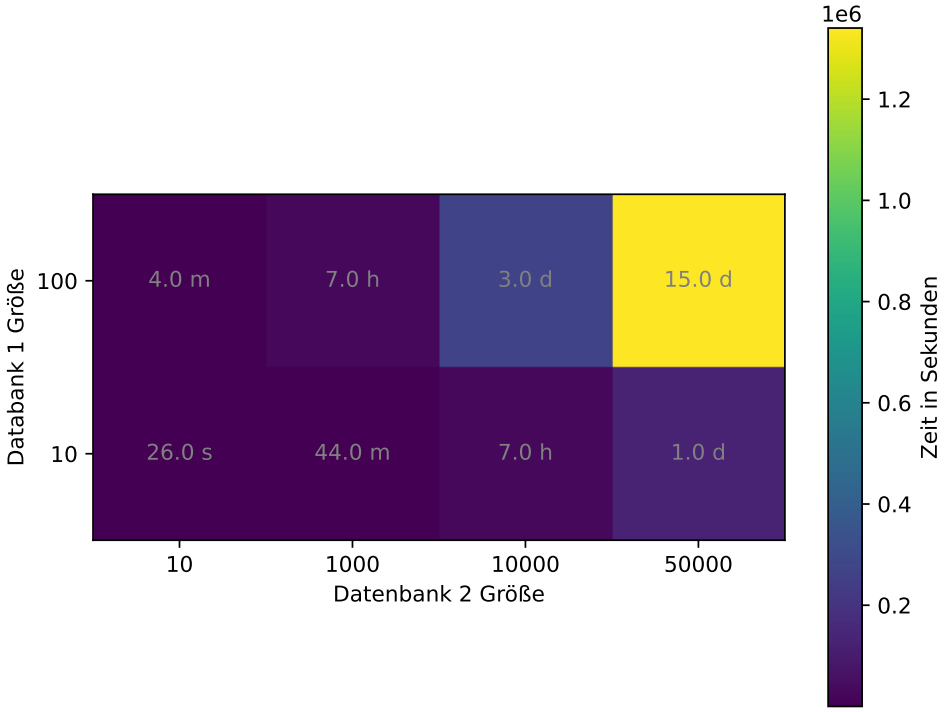



Abbildung 25: Test der Private Set Intersection Anwendung mit kleinen Größen bei Datenbank 1.

Falls mehr als zwei Parteien gleichzeitig an der Berechnung beteiligt sein sollen, ist eine Lösung mit MPC zu bevorzugen, da solche Probleme nicht ohne weiteres mit FHE gelöst werden können.

3. *Ist Netzwerk-Interaktion eingeschränkt oder unerwünscht?*
   Falls keine Netzwerk-Interaktion erwünscht ist oder diese nur eingeschränkt zur Verfügung steht, ist eine Lösung mit FHE, falls möglich, zu bevorzugen.

4. *Haben beide beteiligten Parteien genügend Berechnungsressourcen?*
   Falls eine der beteiligten Parteien nur über sehr wenig Ressourcen verfügt, ist MPC eventuell weniger geeignet als ein FHE-basierter Ansatz, da bei MPC-Lösungen, alle beteiligten Parteien einen Teil der Berechnung stemmen müssen.

5. *Welche Sicherheitsgarantien werden benötigt?*
   Wird SFE mit einseitiger Output-Delivery mittels FHE realisiert, liegt die erreichte Sicherheit zwischen passiv sicherem und aktiv sicherem MPC.

In Abbildung 28 stellen wir die Entscheidungshilfe graphisch dar. Wenn sowohl FHE und auch MPC für die Umsetzung in Frage kommen, sollte anhand eines Tradeoffs zwischen Effizienz und notwendiger Sicherheit entschieden werden.

**Sicherheitsstufen.** MPC die Möglichkeit zwischen passiver Sicherheit und aktiver Sicherheit zu wählen. Vergleicht man die erreichten Sicherheitsstufen ergibt sich folgende Hierarchie: Die geringste Sicherheit bietet passiv sichere Mehrparteienberechnung. Die Sicherheitsgarantien gelten lediglich für Teilnehmer, die sich an das Protokoll halten. Weicht ein Teilnehmer aktiv vom Protokoll ab, kann nicht nur das Ergebnis verfälscht werden, sondern unter Umständen können die geheimen Eingaben der beteiligten Parteien gelernt werden. Eine Lösung mit FHE bietet eine höhere Sicherheit, da der Empfänger des FHE-Chiffrats zwar das Ergebnis

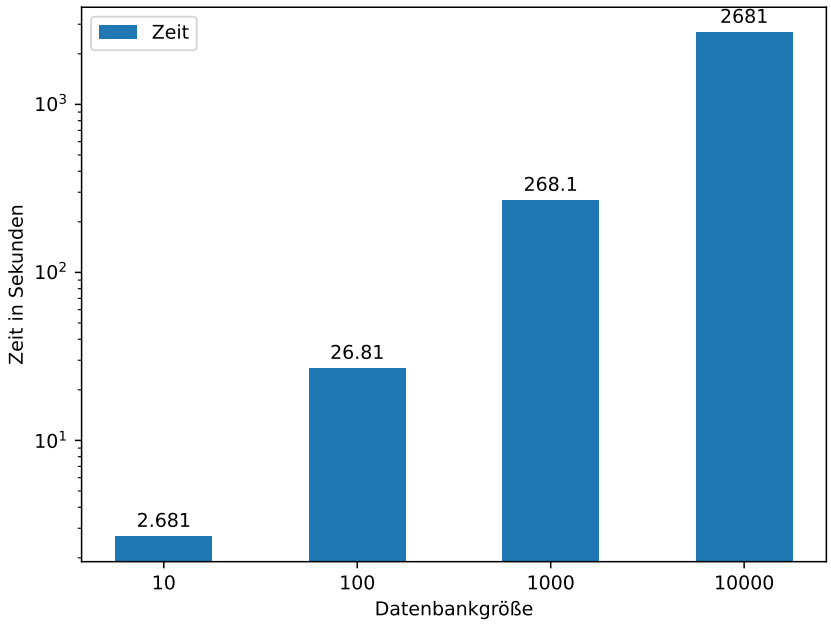



Abbildung 26

Abbildung 27: Die Graphik zeigt die Laufzeit von Private Information Retrieval bei einer Datenbankgröße von 10 bis 100000. Man beachte, dass die y-Achse logarithmisch skaliert wurde.

der Berechnung bei aktiver Manipulation gezielt beeinflussen kann, er allerdings die Eingabe des Gegenüber (d.h. den Inhalt des Chiffrats) nicht ermitteln kann. Die höchste Sicherheitsstufe bietet aktiv sichere Mehrparteienberechnung. Diese stellt sicher, dass selbst bei aktiver Abweichung vom Protokoll sowohl alle Daten geheim bleiben als auch das Ergebnis des Protokolls nicht manipuliert wird.

**Effizienz.** Unter Vernachlässigung des Kommunikationsaufwands bietet MPC aus theoretischer Sicht oft die effizientere Lösung, selbst bei aktiver Sicherheit. Allerdings sind verfügbare Implementierungen von FHE wesentlich ausgereifter und hinsichtlich Effizienz optimiert, was diesen Nachteil ausgleichen kann.

## 11.2 Entscheidungshilfe für FHE-Instantiierung

Falls FHE verwendet werden soll, sollte die Auswahl der zu verwendeten Implementierung in erster Linie in Abhängigkeit der benötigten Operation erfolgen. Die Größe des auszuwertenden Schaltkreises hat beträchtliche Auswirkungen auf die zu erwartende Laufzeit. Daher ist es empfehlenswert ein geeignetes Berechnungsmodell zu wählen, sodass die auszuwertenden Schaltkreise möglichst kompakt bleiben.

Wird hauptsächlich Ganzzahlarithmetik benötigt, eignet sich modulare Arithmetik als Berechnungsmodell gut, da diese Operationen vergleichsweise effizient als arithmetischer Schaltkreis dargestellt werden kann. Damit sind in diesem Fall Verfahren wie BGV oder BFV gut geeignet. Werden allerdings viele Vergleichsoperationen verwendet, ist boolesche Arithmetik besser geeignet, um einen möglichst kompakten Schaltkreis zu erhalten. In diesem Fall sind Verfahren wie GSW, FHEW oder TFHE gut geeignet. Benötigt die Anwendung neuronale Netze oder Polynomapproximationen, ist hingegen Gleitkommaarithmetik am besten geeignet, wodurch das CKKS-Verfahren am besten geeignet ist.

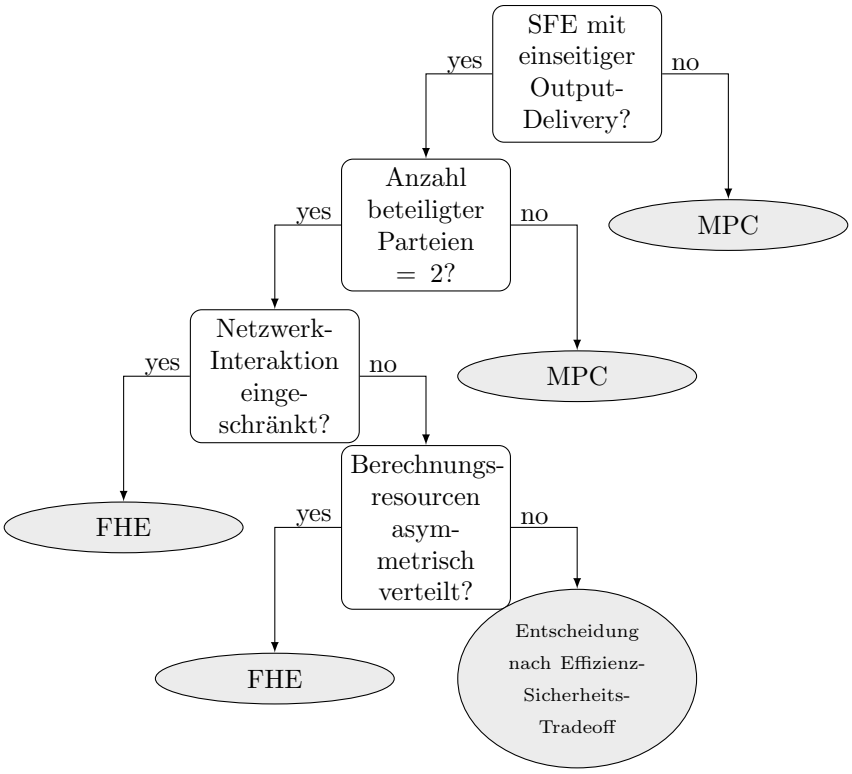



Abbildung 28: Darstellung der Entscheidungshilfe zwischen FHE und MPC.

## 11.3 Entscheidungshilfe für MPC-Implementierung

Bestimmte MPC-Verfahren unterstützen entweder boolesche Schaltkreise (im Fall von Yao-basierten Verfahren) oder arithmetische Schaltkreise (im Fall von Secret-Sharing-basierten Verfahren, insbesondere SPDZ und dessen Varianten). Dennoch bleiben, falls aktive Sicherheit gefordert ist, SPDZ-basierte Verfahren sehr kompetitiv.

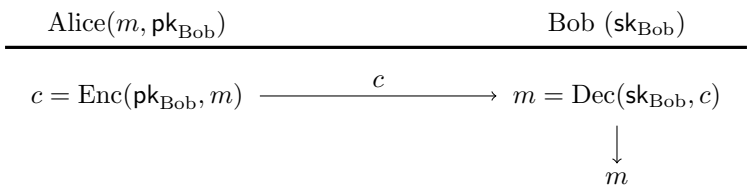
Hybride MPC-Protokolle, wie beispielsweise das ABY-Protokoll, erlauben im Gegensatz zu FHE einen Wechsel zwischen booleschen und arithmetischen Schaltkreisen während des Protokolls, was die Berechnungen erheblich beschleunigen kann, da die Anzahl auszuwertender Gatter erheblich reduziert werden kann.

# 12 Ontologie

## 12.1 Homomorphe Verschlüsselung

### 12.1.1 Varianten

**Public-Key Encryption.** Asymmetrische Verschlüsselung (Public-Key Encryption, PKE) ermöglicht es verschlüsselte Nachrichten auszutauschen, ohne dass dafür ein gemeinsames Geheimnis notwendig ist, wie das bei symmetrischer Verschlüsselung der Fall ist. Zum Verschlüsseln einer Nachricht ist nur der öffentliche Schlüssel des Empfängers notwendig. Siehe auch Abschnitt 2.1.




**(Partially) Homomorphic Encryption.** Homomorphe Verschlüsselung ist ein Spezialfall von Public-Key Encryption bei der es möglich ist, bestimmte Operationen (wie beispielsweise Addition oder Multiplikation) auf Chiffraten anzuwenden, ohne diese zu entschlüsseln. Ein Beispiel für multiplikativ homomorphe Verschlüsseung ist die RSA-Verschlüsselung und die ElGamal-Verschlüsselung Beispiele für additiv homomorphe Verschlüsselungen sind die Paillier-Verschlüsselung oder die Regev-Verschlüsselung.

**Somewhat-Homomorphic Encryption.** Somewhat-Homomorphic Encryption (SHE) ist ein weiterer Spezialfall von Public-Key Encryption. Hierbei können beliebige Operationen auf Chiffraten ausgeführt werden, also insbesondere Addition und Multiplikation. Damit lassen sich auch komplexere Berechnungen durchführen. Allerdings können dabei potentiell Informationen über den ausgeführten Schaltkreis gelernt werden, insbesondere über dessen Tiefe, da es zulässig ist, dass Chiffrate mit jeder ausgeführten homomorphen Operation wachsen.

**Leveled-Homomorphic Encryption.** Leveled-Homomorphic Encryption (LHE) erweitert die vorangegangene Definition von Somewhat-Homomorphic Encryption um zusätzliche Privatsphäre für die Partei, die die auszuwertende Funktion bestimmt: ein Evaluationsschritt ändert hier zwar (potentiell) den Inhalt eines Chiffrats, die Größe des Chiffrates ist jedoch unabhängig von der Tiefe des auszuwertenden Schaltkreises[14]. Wie bei SHE ist es möglich, beliebige Operationen auf dem Chiffrat auszuführen, allerdings nur bis zu einer festen maximalen Anzahl, die bei der Schlüsselerzeugung bekannt sein muss.

**Fully-Homomorphic Encryption.** Vollhomomorphe Verschlüsselung (Fully Homomorphic Encryption, FHE) ist ein Spezialfall von LHE, die es ermöglicht, beliebige Berechnungen auf Chiffraten auszuführen. Bei dieser Variante existiert kein Limit mehr für die maximale Tiefe der Schaltkreise, dennoch gilt (anders als bei Somewhat Homomorphic Encryption), dass die Chiffratlänge unabhängig von der Anzahl angewandter homomorpher Operationen bleibt und man somit beim Entschlüsseln des Chiffrats (abgesehen vom Ergebnis der Berechnung) nichts über die berechnete Funktion lernt. Eine grafische Darstellung einer solchen Auswertung auf einem Schaltkreis ist in Abbildung 29 zu sehen. Mittels Bootstrapping kann aus vielen SHE- oder LHE-Verfahren ein FHE-Verfahren konstruiert werden. Siehe Abschnitte 4 bis 6 für weitere Informationen.

[14] Mit Schaltkreisen sind entweder Boolesche oder arithmetische Schaltkreise gemeint.

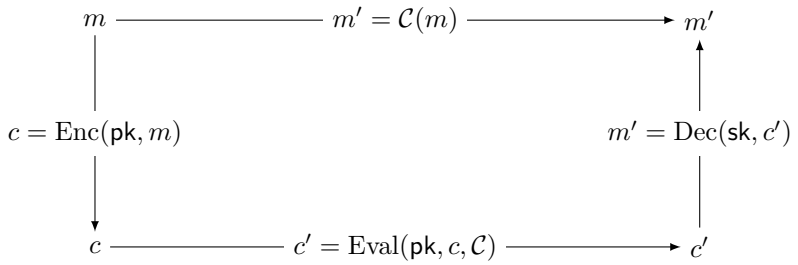



Abbildung 29: Auswertung eines Schaltkreises $\mathcal{C}$ auf einer Nachricht $m$ mittels vollhomomorpher Verschlüsselung.

**Bootstrapping.** Viele Instantiierungen von Somewhat-Homomorphic Encryption haben das technische Problem, dass ein sogenannter Fehlerterm in Chiffraten bei bestimmten homomorphen Operationen stark anwächst, sodass eine korrekte Entschlüsselung nach einigen homomorphen Operationen nicht mehr möglich ist. Bei FHE reduziert Bootstrapping diesen Fehlerterm, um eine korrekte Entschlüsselung zu ermöglichen. Siehe auch Abschnitt 5 für weitere Informationen.

### 12.1.2 Sicherheitsbegriffe

**Passive Sicherheit (IND-CPA).** Mit IND-CPA-Sicherheit bezeichnet man den schwächsten etablierten Sicherheitsbegriff für Public-Key Encryption-Verfahren. Ein Verschlüsselungsverfahren ist IND-CPA-sicher, wenn ein polynomiell beschränkter Angreifer nicht entscheiden kann, welche aus zwei von ihm ausgewählten (identisch langen) Nachrichten ein gegebenes Chiffrat verschlüsselt. Beispiele für IND-CPA-sichere Verfahren sind die ElGamal-Verschlüsselung, Paillier-Verschlüsselung und Regev-Verschlüsselung. Siehe Anhang A für eine formale Definition.

**Aktive Sicherheit (IND-CCA).** IND-CCA-Sicherheit ist ein weiterer etablierter Sicherheitsbegriff für Public-Key Encryption-Verfahren und der de-facto Standard für PKE-Sicherheit. IND-CCA ist definiert wie IND-CPA, wobei sich ein Angreifer außerdem noch beliebige Chiffrate (außer dem Challenge-Chiffrat) entschlüsseln lassen kann. Dies ist ein strikt stärkerer Begriff als IND-CPA. Jedes Verfahren, welches IND-CCA-sicher ist, ist ebenfalls IND-CPA-sicher. Homomorphe Verschlüsselungsverfahren können nicht IND-CCA-sicher sein, denn ein Angreifer kann die Challenge homomorph verändern, sich das veränderte Chiffrat entschlüsseln lassen und darüber Rückschlüsse auf die ursprüngliche Nachricht ziehen. Ein Beispiel für ein Verschlüsselungsverfahren, welches den IND-CCA-Sicherheitsbegriff erfüllt, ist das RSA-OAEP-Verfahren. Siehe Anhang A für eine formale Definition.

**Circuit Privacy.** Eine homomorphes Verschlüsselungsverfahren heißt *Circuit Private*, falls ein homomorph erzeugtes Chiffrat ununterscheidbar von einem frisch verschlüsselten Chiffrat der selben Nachricht ist. Circuit Privacy ist damit eine Eigenschaft des homomorphen Evaluationsalgorithmus. Etwas genauer, sei $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ ein homomorphes Verschlüsselungsverfahren. Sei $(\mathsf{pk}, \mathsf{sk})$ ein von $\mathsf{KeyGen}$ erzeugtes Schlüsselpaar, sei $m$ ein Nachricht und $C(\cdot)$ ein Schaltkreis/Algorithmus. Wenn dieses Verfahren Circuit-Privacy besitzt, so gilt

$$\mathsf{Eval}(\mathsf{pk}, C, \mathsf{Enc}(\mathsf{pk}, m)) \approx \mathsf{Enc}(\mathsf{pk}, C(m)),$$

das heißt man sieht einem Chiffrat nicht an ob dieses aus einer homomorphen Evaluation des Schaltkreises $C$ auf der Verschüsselung einer Nachricht $m$ hervorging, oder eine Verschlüsselung von $C(m)$ ist.

### 12.1.3 Konstruktionen

**ElGamal-Verschlüsselung.** Die ElGamal-Verschlüsselung [58] ist ein multiplikativ homomorphes Public-Key Encryption-Verfahren. Es erreicht Passive Sicherheit (IND-CPA) unter der DDH-Annahme.

**RSA-Verschlüsselung.** Die RSA-Verschlüsselung [126] ist ein multiplikativ homomorphes Public-Key Encryption-Verfahren. Es ist in seiner naiven Version deterministisch und erreicht somit keine Passive Sicherheit (IND-CPA). Erst unter Verwendung des Padding-Verfahrens RSA-OAEP erreicht es Aktive Sicherheit (IND-CCA), verliert dann aber die Homomorphieeigenschaft. Die Sicherheit der RSA-Verschlüsselung basiert auf der RSA-Annahme.

**Paillier-Verschlüsselung.** Das Paillier-Verschlüsselungsverfahren ist ein additiv homomorphes Public-Key Encryption-Verfahren. Es ist IND-CPA-sicher unter der RSA-Annahme.

**Regev-Verschlüsselung.** Regev-Verschlüsselung [125] ist das gitterbasierte Analogon zu der bekannten ElGamal-Verschlüsselung. Die Regev-Verschlüsselung ist additiv homomorph und die Sicherheit basiert auf der LWE-Annahme.

**BGV.** Das BGV-Verfahren [24] ist ein FHE-Verfahren der zweiten Generation (siehe auch Abschnitt 4) [139]. Es basiert auf der RLWE-Annahme. Siehe auch Abschnitt 5.1.

**BFV.** Das BVF-Verfahren [20, 61] ist ein FHE-Verfahren der zweiten Generation (siehe auch Abschnitt 4) [139] und basiert auf der RLWE-Annahme. Siehe auch Abschnitt 5.1.

**GSW.** Das GSW-Verfahren [72] ist ein FHE-Verfahren der sogenannten dritten Generation[15] [139]. Im Gegensatz zu BGV und BFV bietet es einen anderen Effizienz-Tradeoff. Ein wichtiger Unterschied zu BGV und BFV besteht darin, dass Bootstrapping im GSW-Verfahren eine schwächere Annahme benötigt,[16] was es ermöglicht, effizientere Parameter zu wählen. Zudem vermeidet das GSW-Verfahren den rechenaufwendigen Re-Linearisierungsschritt, wodurch es asymptotisch effizienter ist als BGV und BFV. Siehe auch Abschnitt 5.1.

**FHEW.** Bei FHEW [56] handelt es sich um eine Ring-Variante von GSW. Siehe auch Abschnitt 5.1.

**TFHE.** THFE [44, 46] ist eine weitere Ring-Variante von GSW. Siehe auch Abschnitt 5.1.

**CKKS.** Das CKKS-Verfahren [41] ist ein FHE-Verfahren der dritten Generation (siehe auch Abschnitt 4) [139]. Ein wichtiges Alleinstellungsmerkmal dieses Verfahrens ist die Unterstützung von Gleitkommaarithmetik. Aus diesem Grund ist es für Anwendungen wie Machine Learning sehr gut geeignet. Insbesondere können Rundungsoperationen im verschlüsselten Zustand vergleichsweise sehr effizient durchgeführt werden. Siehe auch Abschnitt 5.1.

**CHIMERA.** CHIMERA [19] ist eine Kombination von TFHE, BFV und CKKS. Spezifischer erlaubt CHIMERA es, effizient zwischen diesen drei FHE-Verfahren zu wechseln, ohne eine komplette Neuverschlüsselung zu benötigen. So können die Stärken dieser Verfahren kombiniert werden.

## 12.2 Sichere Mehrparteienberechnung

**MPC.** Sichere Mehrparteienberechnung (oder Multiparty Computation, MPC) erlaubt es sich gegenseitig misstrauenden Parteien gemeinsam Funktionen auszuwerten, ohne ihre Eingaben öffentlich zu machen. Für viele Anwendungen im Bereich Encrypted Computing, bietet MPC eine solide – häufig auch effizientere – Alternative zu FHE auf Kosten von mehr Kommunikationsaufwand. Für eine genauere Beschreibung siehe Abschnitt 7.

[15] Siehe auch Abschnitt 4.

[16] Bootstrapping bei BGV und BFV benötigt die Annahme, dass es schwer ist Gitterprobleme auf superpolynomielle Faktoren genau zu approximieren, wobei GSW annimmt, dass es schwer ist Gitterprobleme auf polynomielle Faktoren genau zu approximieren.

### 12.2.1 Sicherheitsbegriffe

**Passive Sicherheit.** Passive Sicherheit im Kontext von MPC fordert, dass die Sicherheitsgarantien gegen Angreifer bestehen, die sich an die Protokollbeschreibung halten. Diese Klasse von Angreifern nennt man auch "lauschende" oder "honest-but-curious" Angreifer. Siehe auch Abschnitte 7.1 und 7.2.

**Aktive Sicherheit.** Aktive Sicherheit fordert, dass die Sicherheitsgarantien auch gelten, wenn korrumpierte Parteien beliebig vom Protokoll abweichen dürfen. Man nennt diese Angreifer auch "malicious" Angreifer. Diese Form von Sicherheit ist stärker als die Passive Sicherheit und führt meist zu ineffizienteren Protokollen. Es existieren verschiedene Techniken um passiv sichere Protokolle zu aktiv sicheren Protokollen umzuwandeln, wie beispielsweise der GMW-Compiler, der IPS-Compiler oder mittels homomorpher Verschlüsselung [51, 86]. Siehe auch Abschnitte 7.1 und 7.2.

**Statische Sicherheit.** In diesem Modell darf der Angreifer nur einmalig vor der Protokollausführung (beliebig viele) Parteien korrumpieren. Siehe auch Abschnitte 7.1 und 7.2.

**Adaptive Sicherheit.** In diesem Modell kann der Angreifer während der Protokollausführung adaptiv weitere Parteien korrumpieren. Diese bleiben danach für den Rest der Ausführung korrumpiert. Siehe auch Abschnitte 7.1 und 7.2.

**Sicherheit bei ehrlicher Mehrheit.** Einige Protokolle erreichen ihre Sicherheitseigenschaften nur, wenn die Mehrheit (mehr als die Hälfte) der am Protokoll beteiligten Parteien ehrlich (also nicht korrumpiert) sind. Analog gibt es auch Sicherheitsmodelle für z. B. eine ehrliche Zweidrittel-Mehrheit, bei der die Sicherheit nur gegeben ist, wenn ein Angreifer weniger als ein Drittel der Parteien kontrollieren darf. Siehe auch Abschnitte 7.1 und 7.2.

### 12.2.2 Konstruktionen

**Garbled-Circuit-basiert.** Bei Garbled-Circuit-basiertem MPC [144] handelt es sich um ein Protokoll, welches unter Verwendung von Garbled Circuits sichere Mehrparteienberechnung für zwei (oder mehrere) Parteien ermöglicht. Zunächst wird die zu berechnende Funktion als Schaltkreis dargestellt. Dieser Schaltkreis wird von Partei $A$ (zusammen mit ihrer Eingabe) zu einem Garbled Circuit umgewandelt und an Partei $B$ zusammen mit allgemeinen Eingabe-Labels gesendet. Partei $B$ wählt entsprechend ihrer Eingabe Eingabe-Labels aus und ist so in der Lage den Schaltkreis auszuwerten. Dieses Protokoll erreicht passive Sicherheit. Garbled-Circuit-basierte MPC-Protokolle eignen sich gut zur Auswertung Boolescher Operationen. Dieses Protokoll hat wenig Kommunikationsrunden, allerdings müssen sehr große Daten übertragen werden.

**Secret-Sharing-basiert.** Secret-Sharing-basierte MPC-Protokolle ermöglichen sichere Mehrparteienberechnung für beliebig viele Parteien. Hierbei werden die Eingaben aller Parteien mit einem linearen Secret Sharing Verfahren auf alle Parteien aufgeteilt. Lineare Operationen können bei jeder Partei auf ihren Shares lokal durchgeführt werden. Nichtlineare Operationen werden beispielsweise durch eine interaktive Phase [75] oder durch Offenlegen von zuvor präparierten Share-Tupeln umgesetzt [14]. Dieses Vorgehen erreicht passive Sicherheit. Manche Secret-Sharing-basierte MPC-Protokolle erreichen sogar informationstheoretische Sicherheit. Secret-Sharing-basierte MPC-Protokolle eignen sich gut zur Auswertung arithmetischer Operationen. Bei diesem Protokoll müssen im Gegensatz zu Garbled-Circuit-basiertem MPC wesentlich weniger Daten übertragen werden, allerdings benötigt es dafür deutlich mehr Kommunikationsrunden (abhängig von der Tiefe des Schaltkreises), was das Protokoll anfällig für eine hohe Kommunikationslatenz macht. In Anhang B.5 ist dieses Verfahren noch einmal genauer beschrieben.

### 12.2.3 Implementierungen

**ABY und Varianten.** Das ABY- [53] und ABY-2.0-Protokoll-Framework [120] ermöglicht das Erstellen von sehr effizienten Protokollen, allerdings nur mit passiver Sicherheit. Dabei wird je nach Berechnungsoperation zwischen Secret-Sharing-basierten und Garbled-Circuit-basierten Konstruktionen gewechselt. Siehe auch Abschnitt 7.4.

**GAZELLE.** Das GAZELLE-Framework [84] spezialisiert sich auf das Training von neuronalen Netzen (Neuronal Network Inference) und bietet ebenfalls nur passive Sicherheit. Nach eigener Aussage ist GAZELLE bei der Auswertung von neuronalen Netzen im Vergleich zu bestimmten Implementierungen von Fully-Homomorphic Encryption um einiges schneller. Siehe auch Abschnitt 7.4.

**SPDZ und Varianten.** Das SPDZ-Framework [51, 86] und Erweiterungen wie das MP-SPDZ-Framework [85] oder das SCALE-MAMBA-Framework [133] bieten die Möglichkeit zwischen passiver und aktiver Sicherheit zu wählen. Für passive Sicherheit wird eine Secret-Sharing-basierte Variante ohne Interaktion verwendet. Zum Erreichen aktiver Sicherheit wird homomorphe Verschlüsselung verwendet. Der Preis für aktive Sicherheit ist daher eine etwas höhere Laufzeit. Das SPDZ-Framework [51] hat insgesamt eine Komplexität (Offline- und Online-Phase) von $O(n \cdot |C| + n^3)$ für $n$ Parteien und Schaltkreisgröße $|C|$ (siehe Boolescher Schaltkreis (gemessen in elementaren Operationen in $\mathbb{F}_{p^k}$). Siehe auch Abschnitt 7.4.

**MPyC.** MPyC [128] ist ein Python-Framework, das auf einfache Benutzung und Einbindung spezialisiert ist. Dies wird allerdings mit Einbußen in der Effizienz erkauft. Siehe auch Abschnitt 7.4.

## 12.3 Verwandte Begriffe

**Informationstheoretische Sicherheit.** Unter informationstheoretischer Sicherheit versteht man im Allgemeinen, dass eine Sicherheitseigenschaft uneingeschränkt gilt. Dies ist der Fall, wenn keinerlei zum Brechen der Sicherheitseigenschaft nötige Information in den dem Angreifer zur Verfügung gestellten Nachrichten vorhanden ist. Ein schwächerer Sicherheitsbegriff hierzu ist die Komplexitätstheoretische Sicherheit. Meist können nicht alle Sicherheitseigenschaften eines Protokolles informationstheoretisch sein.

**Komplexitätstheoretische Sicherheit.** Eine Sicherheitseigenschaft eines Protokolls ist komplexitätstheoretisch erreicht, wenn ein effizienter Angreifer unter der Annahme, dass ein bestimmtes Problem schwer zu lösen ist, diese Sicherheitseigenschaft nicht brechen kann. Die Annahme, dass ein bestimmtes Problem schwer zu lösen ist, ist damit die Komplexitätsannahme. Beispiele für solche Komplexitätsannahmen sind die RSA-Annahme, die DDH-Annahme oder die LWE-Annahme.

**Boolescher Schaltkreis.** Ein Boolescher Schaltkreis ist ein gerichteter azyklischer Graph. Knoten ohne Eingangskante fungieren als Eingabeknoten, Knoten ohne Ausgangskante fungieren als Ausgabeknoten. Alle Knoten mit Ein- und Ausgangskanten sind mit einer Booleschen Funktion (üblicherweise **AND**, **OR** und **NOT**) gelabelt und geben an, ob es sich dabei um ein AND, OR oder NOT-Gatter handelt. Die Anzahl an Knoten eines Schaltkreises $C$ wird auch als Schaltkreisgröße $|C|$ bezeichnet. Jede in Polynomialzeit berechenbare Funktion (insbesondere jede jede Art von Kontrollflüssen polynomieller Komplexität) lässt sich als Boolescher Schaltkreis darstellen. Zur Anschauung wird in Abbildung 30 ein einfacher Boolescher Schaltkreis dargestellt.

**Arithmetischer Schaltkreis.** Ein Arithmetischer Schaltkreis ist ähnlich wie ein Boolescher Schaltkreis definiert. Der Unterschied ist, dass Gatter mit Addition, Multiplikation (modulo einer festen Zahl $q$) und Negation gelabelt sind, und als Ein- und Ausgabe anstelle von Bits Zahlen modulo $q$ verwendet werden. Arithmetische Schaltkreise sind gleichmächtig wie Boolesche Schaltkreise. So kann ebenfalls jede in Polynomialzeit berechenbare Funktion als arithmetischer Schaltkreis dargestellt werden.

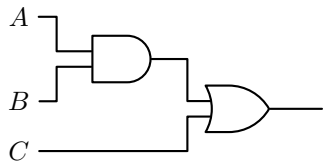



Abbildung 30: Darstellung eines Schaltkreises für die Berechnung von $f(A, B, C) = (A \wedge B) \vee C$.

Arithmetische Schaltkreise eignen sich gut für die Berechnung von Ganzzahlarithmetik. Beispielsweise kann die Addition zweier Zahlen $A$ und $B$ (je nach Größe der Zahlen und des Modulus) meist mittels eines einzigen Gatters erledigt werden. Bei Booleschen Schaltkreisen hingegen, ist dafür eine Zerlegung der Eingabe in Binärzahlen $A = \sum_{i=0}^{\lceil \log |\mathbb{F}| \rceil - 1} a_i 2^i$ und $B = \sum_{i=0}^{\lceil \log |\mathbb{F}| \rceil - 1} b_i 2^i$ notwendig. Die Addition kann mittels XOR- und AND-Gattern realisiert werden. Der Wert $c_i$ der Darstelung der Summe $C = \sum_{i=0}^{\lceil \log |\mathbb{F}| \rceil - 1} c_i 2^i$ entspricht $a_i \mathsf{XOR} b_i \mathsf{XOR} r_{i-1}$, wobei der Übertrag $d_i = (\neg x \wedge y \wedge r_{i-1}) \vee (x \wedge \neg y \wedge r_{i-1}) \vee (x \wedge y)$. Damit sind für diesen Anwendungsfall bei Booleschen Schaltkreisen wesentlich mehr Gatter notwendig.

**Symmetrische Verschlüsselung.** Symmetrische Verschlüsselung (Secret-Key Verschlüsselung, SKE) ermöglicht den Austausch verschlüsselter Nachrichten, sofern sich zuvor beide Parteien auf einen gemeinsamen Schlüssel geeinigt haben.

**Secret Sharing.** Secret Sharing bezeichnet ein Verfahren, eine Nachricht an mehrere Parteien aufzuteilen, sodass die Nachricht nur von allen oder von einer bestimmten Mindestanzahl an Parteien rekonstruiert werden kann. Jede Teilmenge, die kleiner als die zur Rekonstruktion notwendige Schwelle ist, hat keine Information über die geteilte Nachricht. Bei den meisten Verfahren ist diese Sicherheit sogar informationstheoretisch. Zwei der bekanntesten Secret-Sharing-Verfahren sind Shamir-Secret-Sharing und additives Secret-Sharing.

**Shamir-Secret-Sharing.** Shamir's Secret Sharing [131] ist ein Secret-Sharing-Verfahren, welches es erlaubt, eine Nachricht in $N$ Shares aufzuteilen, sodass man mit einer beliebigen Teilmenge der Shares der Größe $n \leq N$ die Nachricht wieder rekonstruieren kann. Hierbei wird die Nachricht in einem Koeffizienten eines Polynomes von Grad $n - 1$ kodiert. Die Shares bestehen aus zufälligen Stützstellen des Polynomes. Beliebige $n$ Parteien können mit ihren Stützstellen durch Polynominterpolation das Polynom (und somit die kodierte Nachricht) rekonstruieren.

**Additives Secret-Sharing.** Additives Secret Sharing ist ein Secret Sharing Verfahren, welches eine Nachricht $m$ so in beliebig viele Shares aufteilt, sodass die Nachricht nur mit allen Shares zusammen rekonstruiert werden kann. Die Shares $s_i$ werden dabei zufällig gleichverteilt aus der entsprechenden Gruppe gezogen, sodass $\sum_i s_i = m$. Die Nachricht wird wieder rekonstruiert, indem alle Shares aufsummiert werden.

**Oblivious Transfer.** Oblivious Transfer (OT) ist ein Baustein, der es einem Sender erlaubt, eine von mehreren möglichen Nachrichten an einen Empfänger zu übermitteln, ohne zu lernen, welche Nachricht übermittelt wurde. Der Empfänger kann auswählen welche Nachricht er empfangen will, lernt aber nichts über die anderen Nachrichten. Siehe Anhang B.4 für mehr Details.

**Garbled Circuit.** Bei einem Garbled Circuit wird ein Boolescher Schaltkreis durch Wertetabellen der einzelnen Gatter dargestellt. Jedem der logischen Werte "0" und "1" wird auf jedem Draht des Schaltkreises ein Label zugeordnet. Die Wertetabellen werden mit Symmetrischer Verschlüsselung so verschlüsselt, dass man mit den gegebenen Labels genau denjenigen Eintrag der Wertetabelle entschlüsseln kann, der das entsprechende Label des Ausgangsdrahtes des Gatters enthält. Siehe Anhang B.5 für mehr Details.

**GMW Compiler.** Der GMW-Compiler [75] erlaubt Protokolle, welche Passive Sicherheit erfüllen in ein aktiv sicheres Protokoll zu konvertieren. Dies wird dadurch erreicht, dass jede Partei zusätzlich zum Protokoll einen Zero-Knowledge Beweissystem führt, dass sie sich an das Protokoll gehalten hat. Voraussetzung dafür ist, dass das Protokoll mit Zero-Knowledge Beweisen kompatibel ist.[17] Eine detaillierte Beschreibung des GMW-Compilers ist in Anhang B.3.

**Cut-and-Choose.** Cut-and-Choose stellt eine Alternative zum häufig ineffizienten GMW Compiler dar um aktive Sicherheit für das Garbled-Circuit-basierte Protokoll zu erreichen. Der Garbler "garbelt" $n$ mal den auszuwertenden Schaltkreis und committet sich darauf. Der Auswerter lässt sich vom Garbler $n-1$ Commitments öffnen, lässt sich die zugehörigen Geheimnisse schicken und prüft, ob diese den erwarteten Schaltkreis garbeln. Nur falls dies der Fall ist, führt der Auswerter den verbleibenden Schaltkreis aus. Dies erreicht keine asymptotische Sicherheit, sondern ermöglicht es mit einer nicht-vernachlässigbaren Wahrscheinlichkeit von $1/n$ zu betrügen. Daher stellt diese Technik eine Möglichkeit eines Trade-offs zwischen aktiver Sicherheit und Effizienz dar.

**IPS Compiler.** Der IPS Compiler [83] ist ebenfalls in der Lage passiv sichere in aktiv sichere Protokolle umzuwandeln. Genauer stellt der Compiler ein Protokoll zur Verfügung, mit dem zwei vergleichsweise einfach zu konstruierende Protokolle – ein Protokoll mit aktiver Sicherheit gegen eine ehrliche Mehrheit und ein Protokoll mit passiver Sicherheit gegen beliebig viele korrumpierte Parteien – zu einem mit herkömmlichen Mitteln vergleichsweise schwierig zu konstruierenden Protokoll mit Sicherheit gegen beliebig viele aktiv korrumpierte Parteien kombiniert werden können. Dafür simulieren die $n$ "physischen" Parteien $m(\approx n^2)$ viele "virtuelle" Parteien, die das passiv sichere Protokoll auf einer modifizierten Funktion ausführen. Mit dem aktiv sicheren Protokoll gegen eine korrumpierte Minderheit verteilen die physischen Parteien ihre Eingaben an die $m$ virtuellen Parteien mittels Secret Sharing, so dass jede virtuelle Partei einen Share von jeder echten Partei besitzt, und verteilen Schlüsselmaterial über Oblivious Transfer. Durch das verteilte Schlüsselmaterial kann jede physische Partei die Ausführung von einer Teilmenge der virtuellen Parteien kontrollieren, so dass Betrugsversuche erkannt werden, jedoch auch Angreifer die zusammen arbeiten nie genug Informationen erhalten, um die Eingaben zu rekonstruieren.

**Zero-Knowledge Beweissystem.** Ein Zero-Knowledge Beweissystem erlaubt es einer Partei, einer anderen Partei zu beweisen, dass eine Aussage gültig ist, ohne weitere Informationen zu veröffentlichen. Beispielsweise lässt sich so beweisen, dass man in Besitz eines geheimen Schlüssels ist, ohne Informationen über diesen preiszugeben.

**Hashfunktionen.** Eine Hashfunktion ist eine Funktion, die eine Eingabe beliebiger Länge bekommt, und eine Ausgabe mit einer festen Länge konstruiert. Für kryptographische Anwendungszwecke wird meistens gefordert, dass es schwierig ist, zwei Eingaben zu finden, die auf den selben Wert abgebildet werden (*Kollisionsresistenz*), und dass es schwer ist, ein Urbild zu einem gegeben Wert im Bild der Funktion zu finden (*pre-image resistance*).

**Random-Oracle Modell.** Das Random Oracle Model ist ein idealisiertes Modell, bei dem Hashfunktionen als idealisiertes Orakel modelliert werden, das bei Anfrage eines noch nicht gesehenen Wertes eine zufällig gleichverteilte Ausgabe erzeugt und sich bei Anfrage bereits angefragter Werte mit vorherigen Antworten konsistent verhält.

**MNIST-Datensatz.** Der MNIST-Benchmark-Datensatz [90] für Machine Learning enthält 60.000 Trainingsdaten für handgeschriebene Ziffernerkennung von 0 bis 9 (und 10.000 Testdatensätzen). Es handelt sich jeweils um 28x28 Graustufenbilder.

[17] Im Random-Oracle Modell ist dies beispielsweise nicht möglich.

**Functional Encryption.** Functional Encryption bezeichnet eine Erweiterung von PKE-Verfahren, bei der es dem Besitzer der geheimen Schlüssels möglich ist, eingeschränkte geheime Schlüssel zu erzeugen (und zu verteilen). Jeder dieser eingeschränkten Schlüssel wird zu einer beliebigen Funktion $f$ erzeugt. Die Entschlüsselung eines Chiffrats mittels eines eingeschränkten Schlüssel, der zu Funktion $f$ erzeugt wurde, liefert nicht wie üblich den Inhalt des Chiffrats, sondern lediglich die Ausgabe der Funktion $f$ angewandt auf den Inhalt des Chiffrats. Im Gegensatz zu FHE ist functional Encryption nicht dazu geeignet Berechnungen auszulagern oder zu verteilen. Allerdings eröffnet es die Möglichkeit feingranular zu entscheiden, was bestimmte Nutzer selbständig entschlüsseln dürfen und was nicht. Functional Encryption verallgemeinert attribute-based Encryption und identity-based Encryption.

**Differential Privacy.** Oftmals möchte man Datensätze veröffentlichen ohne sensible Daten über eine einzelne Person preiszugeben. Das Konzept der Differential Privacy besagt, dass diese Ziel erreicht ist, wenn das Hinzufügen oder Löschen eines Datenpunktes nur zu einer minimalen Veränderung der berechneten Statistiken führt. Dies erreicht man zum Beispiel durch das Hinzufügen von Rauschen.

**Hybrid Homomorphic Encryption.** Bei Hybrid Homomorphiv Encryption wird ein FHE-Verfahren zusammenn mit einem herkömmlichen symmetrisches Verschlüsselungsverfahren verwendet, um zu vermeiden, dass die eigentlichen Daten mit dem ineffizienten FHE-Verfahren verschlüsselt werden müssen. Dazu wird zunächst ein geheimer Schlüssel $\mathsf{sk}'$ für das SKE-Verfahren mit dem FHE-Verfahren verschlüsselt und $c = FHE.\text{Enc}(\mathsf{pk}, \mathsf{sk}')$ an den Server gesendet. Zusätzlich werden die Daten $x$, auf denen die Berechnungen ausgeführt werden sollen, mit dem SKE-Verfahren verschlüsselt und $c' = SKE.\text{Enc}(\mathsf{sk}', x)$ zum Server gesendet. Die homomorphe Evaluierung kann nun mittels $FHE.\text{Eval}(c, C)$, wobei der Schaltkreis $C_{c'}(c)$ zunächst $c'$ entschlüsselt (mittels $c$) und anschließend die gewünschte Funktion auf dem Ergebnis auswertet. Dies ermöglicht einen Tradeoff zwischen Rechenaufwand und Kommunikationsaufwand zugunsten des Kommunikationsaufwands.

### 12.3.1 Annahmen

Alle hier erklärten Annahmen haben eine formale Definition in Anhang A.3.

**DDH-Annahme.** Die DDH-Annahme für eine zyklische Gruppe $\mathbf{G}$ mit Ordnung $q$ und Erzeuger $g$ besagt, dass für jeden effizienten Angreifer und für alle $a, b \in \mathbb{Z}_q$ das Element $g^{ab}$ wie ein zufälliges Element aussieht, selbst wenn dieser $g^a$ und $g^b$ (aber nicht $a$ und $b$) kennt.

**RSA-Annahme.** Die RSA-Annahme [126] besagt, dass kein effizienter Angreifer zu einem zufälligem Produkt $N$ zweier großer Primzahlen, zufälligen Werten $z$ und $r$, die $r$-te Wurzel von $z$ modulo $N$ berechnen kann.

**LWE-Annahme.** Die LWE-Annahme [125] besagt, dass kein effizienter Angreifer ein $s \in \mathbb{Z}_q^n$ rekonstruieren kann, selbst wenn er viele lineare Gleichungen in $s$ bekommt, die allerdings alle um einen kleinen, zufälligen Fehler verfälscht sind.

**RLWE-Annahme.** Die RLWE-Annahme [100] ist eine Spezialisierung der LWE-Annahme. Hier wird die Gruppe $\mathbb{Z}_q^n$ ersetzt durch den Polynomring $\mathbb{Z}_q[X]/\langle x^n + 1\rangle$, was die Repräsentation der linearen Gleichungen verkleinert. Die RLWE-Annahme ist eine echt stärkere Annahme als die LWE-Annahme.

## 12.4 Anwendungen

**Secure Function Evaluation.** Secure Function Evaluation bezeichnet eine Klasse von Anwendungen, bei denen sich gegenseitig misstrauende Parteien eine Funktion auf gemeinsamen Eingaben ausführen können, siehe Abbildung 31 für eine schematische Darstellung.


Abbildung 31: Schematische Darstellung von Secure Function Evaluation.

Nach der Auswertung erhalten üblicherweise beide Parteien das Ergebnis. In manchen Fällen ist es wünschenswert, dass nur eine Partei die Ausgabe erhält. Dies kann mit MPC realisiert werden, indem eine Partei einen Schlüssel zu einem SKE-Verfahren eingibt und die Ausgabe mit diesem Schlüssel verschlüsselt an beide Parteien ausgegeben wird. Zudem kann SFE mit einseitiger Output-Delivery mittels FHE realisiert werden, siehe Abbildung 32. Beachte, dass in Abbildung 32 Partei $P_2$ die Möglichkeit hat, ein beliebiges Chiffrat zu erzeugen und es anstelle von $\mathrm{Enc}(C, f(\cdot, x_2))$ an $P_1$ zu senden. Daher ist die so erreichte Sicherheit echt schwächer als aktive Sicherheit im Sinne von MPC.

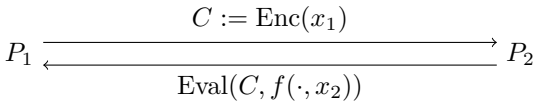



Abbildung 32: Umsetzung von SFE mit einseitiger Output-Delivery mittels FHE.

**Private Cloud Computing.** Private Cloud Computing ermöglicht es einem Client aufwendige Berechnungen auf sentitiven Daten auf einen Server beziehungsweise in die Cloud auszulagern. Dazu sendet der Client die Daten mit FHE verschlüsselt an den Server, der den gewünschten Schaltkreis auf dem Chiffrat auswertet und das erhaltene Chiffrat zurücksendet. Siehe auch Abbildung 32.

**Biometric Matching.** Testen auf biometrische Übereinstimmungen. Je nach Anwendungsfall kann es wünschenswert sein, die gemessenen biometrischen Daten und/oder die gespeicherten authentischen biometrischen Daten zu schützen. Ein Schaubild hierfür ist in Abbildung 33 dargestellt.

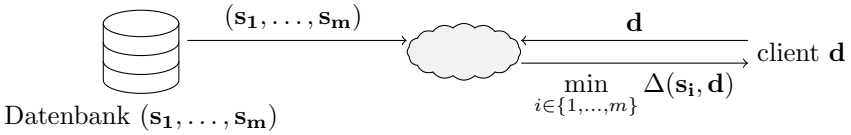



Abbildung 33: Darstellung von privacy-preserving Biometric Matching. Hierbei bezeichnet $\Delta$ eine beliebige Metrik.

**Machine Learning.** Anhand eines manuell klassifizierten Trainingsdatensatzes kann ein neuronales Netz so trainiert werden, dass es möglichst auch bisher ungesehene Daten korrekt klassifizieren kann. Die Trainingsdaten werden benutzt, um die Gewichte des neuronalen Netzes zu lernen. Diese Gewichte bezeichnet man auch als Modell $\theta$, welches dann benutzt werden kann, um bisher unklassifizierte Daten zu klassifizieren. Je nach Anwendungsfall ist es wünschenswert, die eventuell verteilt vorliegenden Trainingsdaten, das Modell und/oder die zu klassifizierenden Daten zu schützen. Siehe Abbildung 34 für verteilt vorliegenden Trainingsdaten. Als Benchmark wird häufig der sogenannte MNIST-Datensatz verwendet.

**Neuronal Network Inference (Machine-Learning-as-a-Service, MLaaS).** Um sensible Daten anhand eines aufwendig trainierten neuronalen Netzes zu klassifizieren ohne das Modell offenzulegen kann die

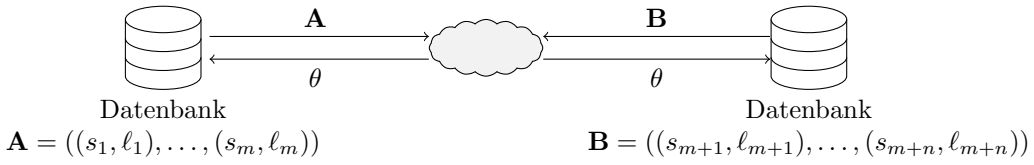



Abbildung 34: Darstellung von privacy-preserving Machine Learning.

Klassifizierung von Daten als Service angeboten werden. Es kann wünschenswert sein die zu klassifizierenden Daten zu schützen.

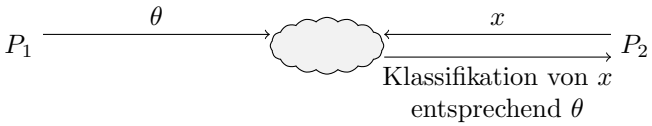



Abbildung 35: Darstellung von MLaaS.

**Private Information Retrieval.** Private Information Retrieval (PIR) ermöglicht es einem Client Informationen aus einer Datenbank des Servers abzufragen, ohne dass der Server erfährt, welchen Eintrag der Client angefragt hat, siehe auch Abbildung 36. Dies ist eine schwächere Variante von Oblivious Transfer, bei der der Client zusätzlich nichts über die anderen Einträge lernen darf.

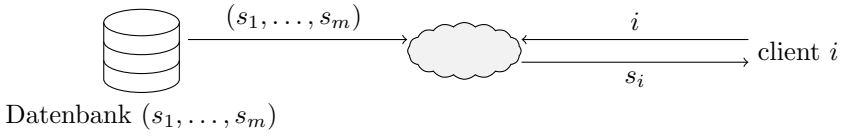



Abbildung 36: Darstellung von Private Information Retrieval.

**Geheime Datenbankanfragen.** Geheime Datenbankabfragen sind ein ähnlicher Anwendungsfall wie Private Information Retrieval mit dem Unterschied, dass der angefragte Datensatz nicht durch einen Index identifiziert wird, sondern durch Bedingungen an den Eintrag, siehe auch Abbildung 37.

## 12.5 Theoretische Anwendungen von FHE

Zusätzlich zu den praktischen Anwendungen von FHE wurde FHE auch als Baustein in anderen kryptografischen Konstruktionen genutzt. Diese sind bisher hauptsächlich von theoretischem Interesse.

**Code Obfuscation.** Code Obfuscation bezeichnet das Compilieren eines Programms auf eine Weise, die Rückschlüsse auf die interne Implementierung des Programms verhindern. Code Obfuscation hat unzählige Anwendungen in der Kryptographie, insbesondere FHE. Umgekehrt unter einer neuen Annahme über die Möglichkeit *vergesslich* LWE-Parameter generieren zu können, konnte gezeigt werden, dass eine Variante von FHE bereits ausreicht um Code Obfuscation zu konstruieren [143]. Für die Praxis hat Code Obfuscation allerdings kaum Relevanz, da bisherige Kandidaten sehr ineffizient sind.

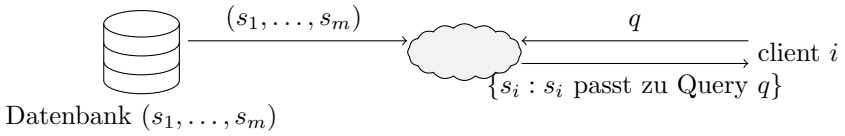



Abbildung 37: Darstellung von geheimen Datenbankabfragen, wobei $q$ eine Datenbankanfrage ist.

**Non-Interactive Zero Knowledge.** Das Protokoll eines Zero Knowlegde Beweises wird zwischen einem Prover und einem Verifier ausgeführt. Dabei versucht der Prover den Verifier von einer Aussage (in Bezug auf eine NP-harte Sprache) zu überzeugen. Der Verifier entscheidet ob er akzeptiert oder ablehnt. Die Variante Non-Interaktive Zero Knowlegde fordert außerdem, dass nur eine Nachricht vom Prover zum Verifier gesendet wird und somit keine Interaktion nötig ist. Solche Protokolle waren bisher nur für nicht quantensichere Annahmen bekannt. Durch die Nutzung von FHE in der Konstruktion konnte ein NIZK Protokoll basierend auf der 'post-quantum' Annahme LWE konstruiert werden.

# Literatur


[1] Allon Adir, Ehud Aharoni, Nir Drucker, Eyal Kushnir, Ramy Masalha, Michael Mirkin und Omri Soceanu. „Privacy-preserving record linkage using local sensitive hash and private set intersection“. In: *arXiv preprint arXiv:2203.14284* (2022).

[2] Divesh Aggarwal und Ueli Maurer. „Breaking RSA Generically Is Equivalent to Factoring“. In: *Advances in Cryptology – EUROCRYPT 2009*. Hrsg. von Antoine Joux. Bd. 5479. Lecture Notes in Computer Science. Cologne, Germany: Springer, Heidelberg, Germany, 2009, S. 36–53. DOI: 10.1007/978-3-642-01001-9_2.

[3] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Travis Morrison Dustin Moody, Amit Sahai und Vinod Vaikuntanathan. http://homomorphicencryption.org/wp-content/uploads/2018/11/HomomorphicEncryptionStandardv1.1.pdf.

[4] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen und Michael Zohner. „Ciphers for MPC and FHE“. In: *Advances in Cryptology – EUROCRYPT 2015, Part I*. Hrsg. von Elisabeth Oswald und Marc Fischlin. Bd. 9056. Lecture Notes in Computer Science. Sofia, Bulgaria: Springer, Heidelberg, Germany, 2015, S. 430–454. DOI: 10.1007/978-3-662-46800-5_17.

[5] Martin R. Albrecht, Jean-Charles Faugère, Robert Fitzpatrick und Ludovic Perret. „Lazy Modulus Switching for the BKW Algorithm on LWE“. In: *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*. Hrsg. von Hugo Krawczyk. Bd. 8383. Lecture Notes in Computer Science. Buenos Aires, Argentina: Springer, Heidelberg, Germany, 2014, S. 429–445. DOI: 10.1007/978-3-642-54631-0_25.

[6] Jacob Alperin-Sheriff und Chris Peikert. „Faster Bootstrapping with Polynomial Error“. In: *Advances in Cryptology – CRYPTO 2014, Part I*. Hrsg. von Juan A. Garay und Rosario Gennaro. Bd. 8616. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2014, S. 297–314. DOI: 10.1007/978-3-662-44371-2_17.

[7] Nick Angelou, Ayoub Benaissa, Bogdan Cebere, William Clark, Adam James Hall, Michael A Hoeh, Daniel Liu, Pavlos Papadopoulos, Robin Roehm, Robert Sandmann u. a. „Asymmetric private set intersection with applications to contact tracing and private vertical federated machine learning“. In: *arXiv preprint arXiv:2011.09350* (2020).

[8] Benny Applebaum, Yuval Ishai und Eyal Kushilevitz. „How to Garble Arithmetic Circuits“. In: *52nd Annual Symposium on Foundations of Computer Science*. Hrsg. von Rafail Ostrovsky. Palm Springs, CA, USA: IEEE Computer Society Press, 2011, S. 120–129. DOI: 10.1109/FOCS.2011.40.

[9] David W Archer, José Manuel Calderón Trilla, Jason Dagit, Alex Malozemoff, Yuriy Polyakov, Kurt Rohloff und Gerard Ryan. „Ramparts: A programmer-friendly system for building homomorphic encryption applications“. In: *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 2019, S. 57–68.

[10] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A. Reuter und Martin Strand. *A Guide to Fully Homomorphic Encryption*. Cryptology ePrint Archive, Report 2015/1192. https://eprint.iacr.org/2015/1192. 2015.

[11] Sanjeev Arora und Rong Ge. „New Algorithms for Learning in Presence of Errors“. In: *ICALP 2011: 38th International Colloquium on Automata, Languages and Programming, Part I*. Hrsg. von Luca Aceto, Monika Henzinger und Jiri Sgall. Bd. 6755. Lecture Notes in Computer Science. Zurich, Switzerland: Springer, Heidelberg, Germany, 2011, S. 403–415. DOI: 10.1007/978-3-642-22006-7_34.

[12] Marshall Ball, Brent Carmer, Tal Malkin, Mike Rosulek und Nichole Schimanski. „Garbled neural networks are practical“. In: *Cryptology ePrint Archive* (2019).

[13] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan und Ke Yang. „On the (Im)possibility of Obfuscating Programs“. In: *Advances in Cryptology – CRYPTO 2001*. Hrsg. von Joe Kilian. Bd. 2139. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2001, S. 1–18. DOI: 10.1007/3-540-44647-8_1.

[14] Donald Beaver. „Foundations of Secure Interactive Computing“. In: *Advances in Cryptology – CRYPTO'91*. Hrsg. von Joan Feigenbaum. Bd. 576. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1992, S. 377–391. DOI: 10.1007/3-540-46766-1_31.

[15] Jean-François Biasse und Fang Song. „Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields“. In: *27th Annual ACM-SIAM Symposium on Discrete Algorithms*. Hrsg. von Robert Krauthgamer. Arlington, VA, USA: ACM-SIAM, 2016, S. 893–902. DOI: 10.1137/1.9781611974331.ch64.

[16] Avrim Blum, Adam Kalai und Hal Wasserman. „Noise-tolerant learning, the parity problem, and the statistical query model“. In: *32nd Annual ACM Symposium on Theory of Computing*. Portland, OR, USA: ACM Press, 2000, S. 435–440. DOI: 10.1145/335305.335355.

[17] Fabian Boemer, Yixing Lao, Rosario Cammarota und Casimir Wierzynski. „nGraph-HE: a graph compiler for deep learning on homomorphically encrypted data“. In: *Proceedings of the 16th ACM International Conference on Computing Frontiers*. 2019, S. 3–13.

[18] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang und David J. Wu. „Private Database Queries Using Somewhat Homomorphic Encryption“. In: *ACNS 13: 11th International Conference on Applied Cryptography and Network Security*. Hrsg. von Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel und Reihaneh Safavi-Naini. Bd. 7954. Lecture Notes in Computer Science. Banff, AB, Canada: Springer, Heidelberg, Germany, 2013, S. 102–118. DOI: 10.1007/978-3-642-38980-1_7.

[19] Christina Boura, Nicolas Gama, Mariya Georgieva und Dimitar Jetchev. *CHIMERA: Combining Ring-LWE-based Fully Homomorphic Encryption Schemes*. Cryptology ePrint Archive, Report 2018/758. https://eprint.iacr.org/2018/758. 2018.

[20] Zvika Brakerski. „Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP“. In: *Advances in Cryptology – CRYPTO 2012*. Hrsg. von Reihaneh Safavi-Naini und Ran Canetti. Bd. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2012, S. 868–886. DOI: 10.1007/978-3-642-32009-5_50.

[21] Zvika Brakerski. „Quantum FHE (Almost) As Secure As Classical“. In: *Advances in Cryptology – CRYPTO 2018, Part III*. Hrsg. von Hovav Shacham und Alexandra Boldyreva. Bd. 10993. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2018, S. 67–95. DOI: 10.1007/978-3-319-96878-0_3.

[22] Zvika Brakerski. „When Homomorphism Becomes a Liability“. In: *TCC 2013: 10th Theory of Cryptography Conference*. Hrsg. von Amit Sahai. Bd. 7785. Lecture Notes in Computer Science. Tokyo, Japan: Springer, Heidelberg, Germany, 2013, S. 143–161. DOI: 10.1007/978-3-642-36594-2_9.

[23] Zvika Brakerski, Craig Gentry und Vinod Vaikuntanathan. „(Leveled) fully homomorphic encryption without bootstrapping“. In: *ITCS 2012: 3rd Innovations in Theoretical Computer Science*. Hrsg. von Shafi Goldwasser. Cambridge, MA, USA: Association for Computing Machinery, 2012, S. 309–325. DOI: 10.1145/2090236.2090262.

[24] Zvika Brakerski, Craig Gentry und Vinod Vaikuntanathan. „(Leveled) Fully Homomorphic Encryption without Bootstrapping“. In: *ACM Trans. Comput. Theory* 6.3 (2014), 13:1–13:36. DOI: 10.1145/2633600. URL: https://doi.org/10.1145/2633600.

[25] Zvika Brakerski und Vinod Vaikuntanathan. „Efficient Fully Homomorphic Encryption from (Standard) LWE“. In: *52nd Annual Symposium on Foundations of Computer Science*. Hrsg. von Rafail Ostrovsky. Palm Springs, CA, USA: IEEE Computer Society Press, 2011, S. 97–106. DOI: 10.1109/FOCS.2011.12.

[26] Zvika Brakerski und Vinod Vaikuntanathan. „Efficient Fully Homomorphic Encryption from (Standard) $\mathsf{LWE}$“. In: *SIAM J. Comput.* 43.2 (2014), S. 831–871. DOI: 10.1137/120868669. URL: https://doi.org/10.1137/120868669.

[27] Zvika Brakerski und Vinod Vaikuntanathan. „Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages“. In: *Advances in Cryptology – CRYPTO 2011*. Hrsg. von Phillip Rogaway. Bd. 6841. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2011, S. 505–524. DOI: 10.1007/978-3-642-22792-9_29.

[28] Zvika Brakerski und Vinod Vaikuntanathan. „Lattice-based FHE as secure as PKE“. In: *ITCS 2014: 5th Conference on Innovations in Theoretical Computer Science*. Hrsg. von Moni Naor. Princeton, NJ, USA: Association for Computing Machinery, 2014, S. 1–12. DOI: 10.1145/2554797.2554799.

[29] Zvika Brakerski, Nico Döttling, Sanjam Garg und Giulio Malavolta. „Candidate iO from Homomorphic Encryption Schemes“. In: *Advances in Cryptology – EUROCRYPT 2020, Part I*. Hrsg. von Anne Canteaut und Yuval Ishai. Bd. 12105. Lecture Notes in Computer Science. Zagreb, Croatia: Springer, Heidelberg, Germany, 2020, S. 79–109. DOI: 10.1007/978-3-030-45721-1_4.

[30] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev und Damien Stehlé. „Classical hardness of learning with errors“. In: *45th Annual ACM Symposium on Theory of Computing*. Hrsg. von Dan Boneh, Tim Roughgarden und Joan Feigenbaum. Palo Alto, CA, USA: ACM Press, 2013, S. 575–584. DOI: 10.1145/2488608.2488680.

[31] Zvika Brakerski, Nico Döttling, Sanjam Garg und Giulio Malavolta. „Leveraging Linear Decryption: Rate-1 Fully-Homomorphic Encryption and Time-Lock Puzzles“. In: *TCC 2019: 17th Theory of Cryptography Conference, Part II*. Hrsg. von Dennis Hofheinz und Alon Rosen. Bd. 11892. Lecture Notes in Computer Science. Nuremberg, Germany: Springer, Heidelberg, Germany, 2019, S. 407–437. DOI: 10.1007/978-3-030-36033-7_16.

[32] Anne Broadbent und Stacey Jeffery. „Quantum Homomorphic Encryption for Circuits of Low T-gate Complexity“. In: *Advances in Cryptology – CRYPTO 2015, Part II*. Hrsg. von Rosario Gennaro und Matthew J. B. Robshaw. Bd. 9216. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2015, S. 609–629. DOI: 10.1007/978-3-662-48000-7_30.

[33] Brandon Broadnax, Alexander Koch, Jeremias Mechler, Tobias Müller, Jörn Müller-Quade und Matthias Nagel. „Fortified Multi-Party Computation: Taking Advantage of Simple Secure Hardware Modules“. In: *Proc. Priv. Enhancing Technol.* 2021.4 (2021), S. 312–338. DOI: 10.2478/popets-2021-0072. URL: https://doi.org/10.2478/popets-2021-0072.

[34] Alon Brutzkus, Oren Elisha und Ran Gilad-Bachrach. „Low Latency Privacy Preserving Inference“. In: *International Conference on Machine Learning*. 2019.

[35] Ran Canetti, Huijia Lin, Stefano Tessaro und Vinod Vaikuntanathan. „Obfuscation of Probabilistic Circuits and Applications“. In: *TCC 2015: 12th Theory of Cryptography Conference, Part II*. Hrsg. von Yevgeniy Dodis und Jesper Buus Nielsen. Bd. 9015. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Heidelberg, Germany, 2015, S. 468–497. DOI: 10.1007/978-3-662-46497-7_19.

[36] Sergiu Carpov, Paul Dubrulle und Renaud Sirdey. „Armadillo: a compilation chain for privacy preserving applications“. In: *Proceedings of the 3rd International Workshop on Security in Cloud Computing*. 2015, S. 13–19.

[37] Orestis Chardouvelis, Nico Döttling und Giulio Malavolta. „Rate-1 Quantum Fully Homomorphic Encryption“. In: *TCC (1)*. Bd. 13042. Lecture Notes in Computer Science. Springer, 2021, S. 149–176.

[38] Jung Hee Cheon und Damien Stehlé. „Fully Homomophic Encryption over the Integers Revisited“. In: *Advances in Cryptology – EUROCRYPT 2015, Part I*. Hrsg. von Elisabeth Oswald und Marc Fischlin. Bd. 9056. Lecture Notes in Computer Science. Sofia, Bulgaria: Springer, Heidelberg, Germany, 2015, S. 513–536. DOI: 10.1007/978-3-662-46800-5_20.

[39] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim und Yongsoo Song. „Bootstrapping for Approximate Homomorphic Encryption“. In: *Advances in Cryptology – EUROCRYPT 2018, Part I*. Hrsg. von Jesper Buus Nielsen und Vincent Rijmen. Bd. 10820. Lecture Notes in Computer Science. Tel Aviv, Israel: Springer, Heidelberg, Germany, 2018, S. 360–384. DOI: 10.1007/978-3-319-78381-9_14.

[40] Jung Hee Cheon, Andrey Kim, Miran Kim und Yongsoo Song. „Homomorphic encryption for arithmetic of approximate numbers“. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2017, S. 409–437.

[41] Jung Hee Cheon, Andrey Kim, Miran Kim und Yong Soo Song. „Homomorphic Encryption for Arithmetic of Approximate Numbers“. In: *Advances in Cryptology – ASIACRYPT 2017, Part I*. Hrsg. von Tsuyoshi Takagi und Thomas Peyrin. Bd. 10624. Lecture Notes in Computer Science. Hong Kong, China: Springer, Heidelberg, Germany, 2017, S. 409–437. DOI: 10.1007/978-3-319-70694-8_15.

[42] Eduardo Chielle, Oleg Mazonka, Homer Gamil, Nektarios Georgios Tsoutsos und Michail Maniatakos. „E3: A framework for compiling C++ programs with encrypted operands“. In: *Cryptology ePrint Archive* (2018).

[43] Ilaria Chillotti, Marc Joye und Pascal Paillier. „Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks“. In: *Cyber Security Cryptography and Machine Learning - 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8-9, 2021, Proceedings*. Hrsg. von Shlomi Dolev, Oded Margalit, Benny Pinkas und Alexander A. Schwarzmann. Bd. 12716. Lecture Notes in Computer Science. Springer, 2021, S. 1–19. DOI: 10.1007/978-3-030-78086-9_1. URL: https://doi.org/10.1007/978-3-030-78086-9_1.

[44] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva und Malika Izabachène. „Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds“. In: *Advances in Cryptology – ASIACRYPT 2016, Part I*. Hrsg. von Jung Hee Cheon und Tsuyoshi Takagi. Bd. 10031. Lecture Notes in Computer Science. Hanoi, Vietnam: Springer, Heidelberg, Germany, 2016, S. 3–33. DOI: 10.1007/978-3-662-53887-6_1.

[45] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva und Malika Izabachène. *TFHE: Fast Fully Homomorphic Encryption Library*. https://tfhe.github.io/tfhe/. August 2016.

[46] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva und Malika Izabachène. „TFHE: Fast Fully Homomorphic Encryption Over the Torus“. In: *Journal of Cryptology* 33.1 (Jan. 2020), S. 34–91. DOI: 10.1007/s00145-019-09319-x.

[47] *Concrete*. https://github.com/zama-ai.

[48] Ronald Cramer, Léo Ducas, Chris Peikert und Oded Regev. „Recovering Short Generators of Principal Ideals in Cyclotomic Rings“. In: *Advances in Cryptology – EUROCRYPT 2016, Part II*. Hrsg. von Marc Fischlin und Jean-Sébastien Coron. Bd. 9666. Lecture Notes in Computer Science. Vienna, Austria: Springer, Heidelberg, Germany, 2016, S. 559–585. DOI: 10.1007/978-3-662-49896-5_20.

[49] Eric Crockett, Chris Peikert und Chad Sharp. „Alchemy: A language and compiler for homomorphic encryption made easy“. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, S. 1020–1037.

[50] *cuFHE*. https://github.com/vernamlab/cuFHE.

[51] Ivan Damgård, Valerio Pastro, Nigel P. Smart und Sarah Zakarias. „Multiparty Computation from Somewhat Homomorphic Encryption“. In: *Advances in Cryptology – CRYPTO 2012*. Hrsg. von Reihaneh Safavi-Naini und Ran Canetti. Bd. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2012, S. 643–662. DOI: 10.1007/978-3-642-32009-5_38.

[52] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine und Madan Musuvathi. „EVA: An encrypted vector arithmetic language and compiler for efficient homomorphic computation“. In: *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2020, S. 546–561.

[53] Daniel Demmler, Thomas Schneider und Michael Zohner. „ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation“. In: *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015. URL: https://www.ndss-symposium.org/ndss2015/aby---framework-efficient-mixed-protocol-secure-two-party-computation.

[54] Marten van Dijk, Craig Gentry, Shai Halevi und Vinod Vaikuntanathan. „Fully Homomorphic Encryption over the Integers“. In: *EUROCRYPT*. Bd. 6110. Lecture Notes in Computer Science. Springer, 2010, S. 24–43.

[55] Changyu Dong und Liqun Chen. „A Fast Single Server Private Information Retrieval Protocol with Low Communication Cost“. In: *ESORICS 2014: 19th European Symposium on Research in Computer Security, Part I*. Hrsg. von Miroslaw Kutylowski und Jaideep Vaidya. Bd. 8712. Lecture Notes in Computer Science. Wroclaw, Poland: Springer, Heidelberg, Germany, 2014, S. 380–399. DOI: 10.1007/978-3-319-11203-9_22.

[56] Léo Ducas und Daniele Micciancio. „FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second“. In: *Advances in Cryptology – EUROCRYPT 2015, Part I*. Hrsg. von Elisabeth Oswald und Marc Fischlin. Bd. 9056. Lecture Notes in Computer Science. Sofia, Bulgaria: Springer, Heidelberg, Germany, 2015, S. 617–640. DOI: 10.1007/978-3-662-46800-5_24.

[57] Kirsten Eisenträger, Sean Hallgren und Kristin E. Lauter. „Weak Instances of PLWE“. In: *Selected Areas in Cryptography*. Bd. 8781. Lecture Notes in Computer Science. Springer, 2014, S. 183–194.

[58] Taher ElGamal. „A public key cryptosystem and a signature scheme based on discrete logarithms“. In: *IEEE transactions on information theory* 31.4 (1985), S. 469–472.

[59] Yara Elias, Kristin E. Lauter, Ekin Ozman und Katherine E. Stange. „Provably Weak Instances of Ring-LWE“. In: *Advances in Cryptology – CRYPTO 2015, Part I*. Hrsg. von Rosario Gennaro und Matthew J. B. Robshaw. Bd. 9215. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2015, S. 63–92. DOI: 10.1007/978-3-662-47989-6_4.

[60] Tim van Elsloo, Giorgio Patrini und Hamish Ivey-Law. „SEALion: A framework for neural network inference on encrypted data“. In: *arXiv preprint arXiv:1904.12840* (2019).

[61] Junfeng Fan und Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Report 2012/144. https://eprint.iacr.org/2012/144. 2012.

[62] *FHEW*. https://github.com/lducas/FHEW.

[63] Youssef Gahi, Mouhcine Guennoun und Khalil El-Khatib. „A secure database system using homomorphic encryption schemes“. In: *arXiv preprint arXiv:1512.03498* (2015).

[64] Sanjam Garg, Craig Gentry und Shai Halevi. „Candidate Multilinear Maps from Ideal Lattices“. In: *Advances in Cryptology – EUROCRYPT 2013*. Hrsg. von Thomas Johansson und Phong Q. Nguyen. Bd. 7881. Lecture Notes in Computer Science. Athens, Greece: Springer, Heidelberg, Germany, 2013, S. 1–17. DOI: 10.1007/978-3-642-38348-9_1.

[65] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai und Brent Waters. „Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits“. In: *54th Annual Symposium on Foundations of Computer Science*. Berkeley, CA, USA: IEEE Computer Society Press, 2013, S. 40–49. DOI: 10.1109/FOCS.2013.13.

[66] Craig Gentry. „Fully homomorphic encryption using ideal lattices“. In: *41st Annual ACM Symposium on Theory of Computing*. Hrsg. von Michael Mitzenmacher. Bethesda, MD, USA: ACM Press, 2009, S. 169–178. DOI: 10.1145/1536414.1536440.

[67] Craig Gentry und Shai Halevi. „Compressible FHE with Applications to PIR“. In: *TCC 2019: 17th Theory of Cryptography Conference, Part II*. Hrsg. von Dennis Hofheinz und Alon Rosen. Bd. 11892. Lecture Notes in Computer Science. Nuremberg, Germany: Springer, Heidelberg, Germany, 2019, S. 438–464. DOI: 10.1007/978-3-030-36033-7_17.

[68] Craig Gentry und Shai Halevi. „Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits“. In: *52nd Annual Symposium on Foundations of Computer Science*. Hrsg. von Rafail Ostrovsky. Palm Springs, CA, USA: IEEE Computer Society Press, 2011, S. 107–109. DOI: 10.1109/FOCS.2011.94.

[69] Craig Gentry, Shai Halevi und Nigel P. Smart. „Fully Homomorphic Encryption with Polylog Overhead“. In: *Advances in Cryptology – EUROCRYPT 2012*. Hrsg. von David Pointcheval und Thomas Johansson. Bd. 7237. Lecture Notes in Computer Science. Cambridge, UK: Springer, Heidelberg, Germany, 2012, S. 465–482. DOI: 10.1007/978-3-642-29011-4_28.

[70] Craig Gentry, Shai Halevi und Nigel P. Smart. „Homomorphic Evaluation of the AES Circuit“. In: *Advances in Cryptology – CRYPTO 2012*. Hrsg. von Reihaneh Safavi-Naini und Ran Canetti. Bd. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2012, S. 850–867. DOI: 10.1007/978-3-642-32009-5_49.

[71] Craig Gentry, Chris Peikert und Vinod Vaikuntanathan. „Trapdoors for hard lattices and new cryptographic constructions“. In: *40th Annual ACM Symposium on Theory of Computing*. Hrsg. von Richard E. Ladner und Cynthia Dwork. Victoria, BC, Canada: ACM Press, 2008, S. 197–206. DOI: 10.1145/1374376.1374407.

[72] Craig Gentry, Amit Sahai und Brent Waters. „Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based“. In: *Advances in Cryptology – CRYPTO 2013, Part I*. Hrsg. von Ran Canetti und Juan A. Garay. Bd. 8042. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2013, S. 75–92. DOI: 10.1007/978-3-642-40041-4_5.

[73] Oded Goldreich, Shafi Goldwasser und Shai Halevi. „Eliminating Decryption Errors in the Ajtai-Dwork Cryptosystem“. In: *Advances in Cryptology – CRYPTO'97*. Hrsg. von Burton S. Kaliski Jr. Bd. 1294. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1997, S. 105–111. DOI: 10.1007/BFb0052230.

[74] Oded Goldreich und Hugo Krawczyk. „On the Composition of Zero-Knowledge Proof Systems“. In: *ICALP90*. Hrsg. von Mike Paterson. Bd. 443. Lecture Notes in Computer Science. Springer, 1990, S. 268–282. DOI: 10.1007/BFb0032038.

[75] Oded Goldreich, Silvio Micali und Avi Wigderson. „How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority“. In: *19th Annual ACM Symposium on Theory of Computing*. Hrsg. von Alfred Aho. New York City, NY, USA: ACM Press, 1987, S. 218–229. DOI: 10.1145/28395.28420.

[76] Shruthi Gorantala, Rob Springer, Sean Purser-Haskell, William Lam, Royce Wilson, Asra Ali, Eric P Astor, Itai Zukerman, Sam Ruth, Christoph Dibak u. a. „A general purpose transpiler for fully homomorphic encryption“. In: *arXiv preprint arXiv:2106.07893* (2021).

[77] Shai Halevi. *Homomorphic Encryption*. https://shaih.github.io/pubs/he-chapter.pdf.

[78] *HeaAn*. https://github.com/snucrypto/HEAAN.

[79] *HElib*. https://github.com/homenc/HElib.

[80] Ryo Hiromasa, Masayuki Abe und Tatsuaki Okamoto. „Packing Messages and Optimizing Bootstrapping in GSW-FHE“. In: *PKC 2015: 18th International Conference on Theory and Practice of Public Key Cryptography*. Hrsg. von Jonathan Katz. Bd. 9020. Lecture Notes in Computer Science. Gaithersburg, MD, USA: Springer, Heidelberg, Germany, 2015, S. 699–715. DOI: 10.1007/978-3-662-46447-2_31.

[81] Yan Huang, David Evans, Jonathan Katz und Lior Malka. „Faster Secure {Two-Party} Computation Using Garbled Circuits“. In: *20th USENIX Security Symposium (USENIX Security 11)*. 2011.

[82] Russell Impagliazzo und Steven Rudich. „Limits on the Provable Consequences of One-Way Permutations“. In: *21st Annual ACM Symposium on Theory of Computing*. Seattle, WA, USA: ACM Press, 1989, S. 44–61. DOI: 10.1145/73007.73012.

[83] Yuval Ishai, Manoj Prabhakaran und Amit Sahai. „Founding Cryptography on Oblivious Transfer - Efficiently“. In: *Advances in Cryptology – CRYPTO 2008*. Hrsg. von David Wagner. Bd. 5157. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2008, S. 572–591. DOI: 10.1007/978-3-540-85174-5_32.

[84] Chiraag Juvekar, Vinod Vaikuntanathan und Anantha Chandrakasan. „GAZELLE: A Low Latency Framework for Secure Neural Network Inference“. In: *USENIX Security 2018: 27th USENIX Security Symposium*. Hrsg. von William Enck und Adrienne Porter Felt. Baltimore, MD, USA: USENIX Association, 2018, S. 1651–1669.

[85] Marcel Keller. „MP-SPDZ: A Versatile Framework for Multi-Party Computation“. In: *ACM CCS 2020: 27th Conference on Computer and Communications Security*. Hrsg. von Jay Ligatti, Xinming Ou, Jonathan Katz und Giovanni Vigna. Virtual Event, USA: ACM Press, 2020, S. 1575–1590. DOI: 10.1145/3372297.3417872.

[86] Marcel Keller, Dragos Rotaru, Peter Scholl und Nigel P. Smart. *Multiparty computation with SPDZ, MASCOT, and Overdrive offline phases*. https://github.com/bristolcrypto/SPDZ-2. 2018.

[87] Himanshu Khurana, Mark Hadley, Ning Lu und Deborah A Frincke. „Smart-grid security issues“. In: *IEEE Security & Privacy* 8.1 (2010), S. 81–85.

[88] Paul Kirchner und Pierre-Alain Fouque. „An Improved BKW Algorithm for LWE with Applications to Cryptography and Lattices“. In: *Advances in Cryptology – CRYPTO 2015, Part I*. Hrsg. von Rosario Gennaro und Matthew J. B. Robshaw. Bd. 9215. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2015, S. 43–62. DOI: 10.1007/978-3-662-47989-6_3.

[89] „Lattigo“. In:

[90] Yann LeCun und Corinna Cortes. *MNIST handwritten digit database*. http://yann.lecun.com/exdb/mnist/. 2010.

[91] Yongwoo Lee, Seonyeong Heo, Seonyoung Cheon, Shinnung Jeong, Changsu Kim, Eunkyung Kim, Dongyoon Lee und Hanjun Kim. „Hecate: performance-aware scale optimization for homomorphic encryption compiler“. In: *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE. 2022, S. 193–204.

[92] A.K. Lenstra, H.W.jun. Lenstra und László Lovász. „Factoring polynomials with rational coefficients“. In: *Math. Ann.* ().

[93] Baiyu Li und Daniele Micciancio. „On the Security of Homomorphic Encryption on Approximate Numbers“. In: *Advances in Cryptology – EUROCRYPT 2021, Part I*. Hrsg. von Anne Canteaut und François-Xavier Standaert. Bd. 12696. Lecture Notes in Computer Science. Zagreb, Croatia: Springer, Heidelberg, Germany, 2021, S. 648–677. DOI: 10.1007/978-3-030-77870-5_23.

[94] Baiyu Li, Daniele Micciancio, Mark Schultz und Jessica Sorrell. „Securing Approximate Homomorphic Encryption Using Differential Privacy“. In: *CRYPTO (1)*. Bd. 13507. Lecture Notes in Computer Science. Springer, 2022, S. 560–589.

[95] *Library implementing the Fan-Vercauteren homomorphic encryption scheme*. https://github.com/CryptoExperts/FV-NFLlib.

[96] Huijia Lin, Rafael Pass, Karn Seth und Sidharth Telang. „Indistinguishability Obfuscation with Non-trivial Efficiency“. In: *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part II*. Hrsg. von Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano und Bo-Yin Yang. Bd. 9615. Lecture Notes in Computer Science. Taipei, Taiwan: Springer, Heidelberg, Germany, 2016, S. 447–462. DOI: 10.1007/978-3-662-49387-8_17.

[97] Adriana López-Alt, Eran Tromer und Vinod Vaikuntanathan. „On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption“. In: *44th Annual ACM Symposium on Theory of Computing*. Hrsg. von Howard J. Karloff und Toniann Pitassi. New York, NY, USA: ACM Press, 2012, S. 1219–1234. DOI: 10.1145/2213977.2214086.

[98] Vadim Lyubashevsky, Chris Peikert und Oded Regev. „A Toolkit for Ring-LWE Cryptography“. In: *Advances in Cryptology – EUROCRYPT 2013*. Hrsg. von Thomas Johansson und Phong Q. Nguyen. Bd. 7881. Lecture Notes in Computer Science. Athens, Greece: Springer, Heidelberg, Germany, 2013, S. 35–54. DOI: 10.1007/978-3-642-38348-9_3.

[99] Vadim Lyubashevsky, Chris Peikert und Oded Regev. „On Ideal Lattices and Learning with Errors over Rings“. In: *Advances in Cryptology – EUROCRYPT 2010*. Hrsg. von Henri Gilbert. Bd. 6110. Lecture Notes in Computer Science. French Riviera: Springer, Heidelberg, Germany, 2010, S. 1–23. DOI: 10.1007/978-3-642-13190-5_1.

[100] Vadim Lyubashevsky, Chris Peikert und Oded Regev. „On Ideal Lattices and Learning with Errors over Rings“. In: *J. ACM* 60.6 (2013), 43:1–43:35. DOI: 10.1145/2535925. URL: https://doi.org/10.1145/2535925.

[101] Jing Ma, Si-Ahmed Naas, Stephan Sigg und Xixiang Lyu. „Privacy-preserving federated learning based on multi-key homomorphic encryption“. In: *International Journal of Intelligent Systems* (2022).

[102] Urmila Mahadev. „Classical Homomorphic Encryption for Quantum Circuits“. In: *59th Annual Symposium on Foundations of Computer Science*. Hrsg. von Mikkel Thorup. Paris, France: IEEE Computer Society Press, 2018, S. 332–338. DOI: 10.1109/FOCS.2018.00039.

[103] Anthony R Metke und Randy L Ekl. „Security technology for smart grid networks“. In: *IEEE Transactions on Smart Grid* 1.1 (2010), S. 99–107.

[104] Daniele Micciancio. „Improving Lattice Based Cryptosystems Using the Hermite Normal Form“. In: *CaLC*. Bd. 2146. Lecture Notes in Computer Science. Springer, 2001, S. 126–145.

[105] Daniele Micciancio und Chris Peikert. „Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller“. In: *Advances in Cryptology – EUROCRYPT 2012*. Hrsg. von David Pointcheval und Thomas Johansson. Bd. 7237. Lecture Notes in Computer Science. Cambridge, UK: Springer, Heidelberg, Germany, 2012, S. 700–718. DOI: 10.1007/978-3-642-29011-4_41.

[106] *Microsoft SEAL (release 4.0)*. https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA. März 2022.

[107] Toufique Morshed, Md Momin Al Aziz und Noman Mohammed. „CPU and GPU Accelerated Fully Homomorphic Encryption“. In: *2020 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2020, San Jose, CA, USA, December 7-11, 2020*. IEEE, 2020, S. 142–153. DOI: 10.1109/HOST45689.2020.9300288. URL: https://doi.org/10.1109/HOST45689.2020.9300288.

[108] Katia Moskvitch. *Top Brazilian Bank Pilots Privacy Encryption Quantum Computers Can't Break*. Techn. Ber. 2020.

[109] Christian Mouchet, Jean-Philippe Bossuat, Juan Troncoso-Pastoriza und J Hubaux. „Lattigo: A multiparty homomorphic encryption library in go“. In: *WAHC 2020–8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. Bd. 15. 2020.

[110] Pratyay Mukherjee und Daniel Wichs. „Two Round Multiparty Computation via Multi-key FHE“. In: *Advances in Cryptology – EUROCRYPT 2016, Part II*. Hrsg. von Marc Fischlin und Jean-Sébastien Coron. Bd. 9666. Lecture Notes in Computer Science. Vienna, Austria: Springer, Heidelberg, Germany, 2016, S. 735–763. DOI: 10.1007/978-3-662-49896-5_26.

[111] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss und Frank Piessens. „Plundervolt: Software-based Fault Injection Attacks against Intel SGX“. In: *2020 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, 2020, S. 1466–1482. DOI: 10.1109/SP40000.2020.00057.

[112] Karthik Nandakumar, Nalini Ratha, Sharath Pankanti und Shai Halevi. „Towards Deep Neural Network Training on Encrypted Data“. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2019, S. 40–48. DOI: 10.1109/CVPRW.2019.00011.

[113] Alexander Nilsson, Pegah Nikbakht Bideh und Joakim Brorsson. „A Survey of Published Attacks on Intel SGX“. In: *CoRR* abs/2006.13598 (2020). arXiv: 2006.13598. URL: https://arxiv.org/abs/2006.13598.

[114] *NuCypher fully homomorphic encryption (NuFHE) library implemented in Python*. https://github.com/nucypher/nufhe.

[115] Femi G. Olumofin und Ian Goldberg. „Revisiting the Computational Practicality of Private Information Retrieval“. In: *FC 2011: 15th International Conference on Financial Cryptography and Data Security*. Hrsg. von George Danezis. Bd. 7035. Lecture Notes in Computer Science. Gros Islet, St. Lucia: Springer, Heidelberg, Germany, 2012, S. 158–172.

[116] *OpenFHE*. https://github.com/openfheorg/openfhe-development.

[117] *PALISADE*. https://git.njit.edu/groups/PALISADE.

[118] Rafael Pass, Elaine Shi und Florian Tramèr. „Formal Abstractions for Attested Execution Secure Processors“. In: *Advances in Cryptology – EUROCRYPT 2017, Part I*. Hrsg. von Jean-Sébastien Coron und Jesper Buus Nielsen. Bd. 10210. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, 2017, S. 260–289. DOI: 10.1007/978-3-319-56620-7_10.

[119] Arpita Patra, Thomas Schneider, Ajith Suresh und Hossein Yalame. „{ABY2. 0}: Improved {Mixed-Protocol} Secure {Two-Party} Computation“. In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, S. 2165–2182.

[120] Arpita Patra, Thomas Schneider, Ajith Suresh und Hossein Yalame. „ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation“. In: *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. Hrsg. von Michael Bailey und Rachel Greenstadt. USENIX Association, 2021, S. 2165–2182. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/patra.

[121] Chris Peikert. *A Decade of Lattice Cryptography*. Cryptology ePrint Archive, Report 2015/939. https://eprint.iacr.org/2015/939. 2015.

[122] Benny Pinkas, Thomas Schneider und Michael Zohner. „Faster private set intersection based on {OT} extension“. In: *23rd USENIX Security Symposium (USENIX Security 14)*. 2014, S. 797–812.

[123] Gaëtan Pradel und Chris Mitchell. „Privacy-Preserving Biometric Matching Using Homomorphic Encryption“. In: *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE. 2021, S. 494–505.

[124] Oded Regev. „On lattices, learning with errors, random linear codes, and cryptography“. In: *37th Annual ACM Symposium on Theory of Computing*. Hrsg. von Harold N. Gabow und Ronald Fagin. Baltimore, MA, USA: ACM Press, 2005, S. 84–93. DOI: 10.1145/1060590.1060603.

[125] Oded Regev. „On lattices, learning with errors, random linear codes, and cryptography“. In: *J. ACM* 56.6 (2009), 34:1–34:40. DOI: 10.1145/1568318.1568324. URL: http://doi.acm.org/10.1145/1568318.1568324.

[126] Ronald L. Rivest, Adi Shamir und Leonard M. Adleman. „A Method for Obtaining Digital Signatures and Public-Key Cryptosystems“. In: *Commun. ACM* 21.2 (1978), S. 120–126. DOI: 10.1145/359340.359342. URL: http://doi.acm.org/10.1145/359340.359342.

[127] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald G. Dreslinski, Christopher Peikert und Daniel Sánchez. „F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption“. In: *MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event, Greece, October 18-22, 2021*. ACM, 2021, S. 238–252. DOI: 10.1145/3466752.3480070. URL: https://doi.org/10.1145/3466752.3480070.

[128] Berry Schoenmakers. *MPyC*. https://github.com/lschoe/mpyc.

[129] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher und Daniel Gruss. „ZombieLoad: Cross-Privilege-Boundary Data Sampling“. In: *ACM CCS 2019: 26th Conference on Computer and Communications Security*. Hrsg. von Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang und Jonathan Katz. ACM Press, 2019, S. 753–768. DOI: 10.1145/3319535.3354252.

[130] Arnaud Grivet Sébert, Renaud Sirdey, Oana Stan und Cédric Gouy-Pailler. „Protecting Data from all Parties: Combining FHE and DP in Federated Learning“. In: *arXiv preprint arXiv:2205.04330* (2022).

[131] Adi Shamir. „How to Share a Secret“. In: *Communications of the Association for Computing Machinery* 22.11 (Nov. 1979), S. 612–613.

[132] Peter W Shor. „Algorithms for quantum computation: discrete logarithms and factoring“. In: *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, S. 124–134.

[133] Nigel Smart, Ben Hamlin, Draos Rotaru und Asif Mallik. *SCALE-MAMBA MPC system*. https://github.com/KULeuven-COSIC/SCALE-MAMBA.

[134] Nigel P. Smart und Frederik Vercauteren. „Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes“. In: *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*. Hrsg. von Phong Q. Nguyen und David Pointcheval. Bd. 6056. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, 2010, S. 420–443. DOI: 10.1007/978-3-642-13013-7_25.

[135] Dimitris Stripelis, Hamza Saleem, Tanmay Ghai, Nikhil Dhinagar, Umang Gupta, Chrysovalantis Anastasiou, Greg Ver Steeg, Srivatsan Ravi, Muhammad Naveed, Paul M Thompson u. a. „Secure neuroimaging analysis using federated learning with homomorphic encryption“. In: *17th International Symposium on Medical Information Processing and Analysis*. Bd. 12088. SPIE. 2021, S. 351–359.

[136] Yang Su, Bailong Yang, Chen Yang und Luogeng Tian. „FPGA-Based Hardware Accelerator for Leveled Ring-LWE Fully Homomorphic Encryption“. In: *IEEE Access* 8 (2020), S. 168008–168025. DOI: 10.1109/ACCESS.2020.3023255. URL: https://doi.org/10.1109/ACCESS.2020.3023255.

[137] Samet Tonyali, Nico Saputro und Kemal Akkaya. „Assessing the feasibility of fully homomorphic encryption for smart grid ami networks“. In: *2015 Seventh International Conference on Ubiquitous and Future Networks*. IEEE. 2015, S. 591–596.

[138] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang und Yi Zhou. „A hybrid approach to privacy-preserving federated learning“. In: *Proceedings of the 12th ACM workshop on artificial intelligence and security*. 2019, S. 1–11.

[139] Vinod Vaikuntanathan. *Homomorphic Encryption References*. https://people.csail.mit.edu/vinodv/FHE/FHE-refs.html.

[140] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom und Raoul Strackx. „Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution“. In: *USENIX Security 2018: 27th USENIX Security Symposium*. Hrsg. von William Enck und Adrienne Porter Felt. Baltimore, MD, USA: USENIX Association, 2018, S. 991–1008.

[141] Alexander Viand und Hossein Shafagh. „Marble: Making fully homomorphic encryption accessible to all“. In: *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 2018, S. 49–60.

[142] Alexander Viand, Patrick Jattke, Miro Haller und Anwar Hithnawi. „HECO: Automatic Code Optimizations for Efficient Fully Homomorphic Encryption“. In: *arXiv preprint arXiv:2202.01649* (2022).

[143] Hoeteck Wee und Daniel Wichs. „Candidate Obfuscation via Oblivious LWE Sampling“. In: *Advances in Cryptology – EUROCRYPT 2021, Part III*. Hrsg. von Anne Canteaut und François-Xavier Standaert. Bd. 12698. Lecture Notes in Computer Science. Zagreb, Croatia: Springer, Heidelberg, Germany, 2021, S. 127–156. DOI: 10.1007/978-3-030-77883-5_5.

[144] Andrew Chi-Chih Yao. „How to Generate and Exchange Secrets (Extended Abstract)“. In: *27th Annual Symposium on Foundations of Computer Science*. Toronto, Ontario, Canada: IEEE Computer Society Press, 1986, S. 162–167. DOI: 10.1109/SFCS.1986.25.

[145] Andrew Chi-Chih Yao. „Protocols for Secure Computations (Extended Abstract)“. In: *23rd Annual Symposium on Foundations of Computer Science*. Chicago, Illinois: IEEE Computer Society Press, 1982, S. 160–164. DOI: 10.1109/SFCS.1982.38.

[146] Xun Yi, Mohammed Golam Kaosar, Russell Paulet und Elisa Bertino. „Single-Database Private Information Retrieval from Fully Homomorphic Encryption“. In: *IEEE Transactions on Knowledge and Data Engineering* 25.5 (2013), S. 1125–1134. DOI: 10.1109/TKDE.2012.90.

[147] Jiale Zhang, Bing Chen, Yanchao Zhao, Xiang Cheng und Feng Hu. „Data security and privacy-preserving in edge computing paradigm: Survey and open issues“. In: *IEEE access* 6 (2018), S. 18209–18237.

# Anhang

# A Formale Definitionen

## A.1 Effizienz und vernachlässigbare Funktionen

Der Begriff der Effizienz baut auf polynomiellen Schranken auf, die garantieren, dass die Zeit bis ein Algorithmus eine Ausgabe gibt höchstens polynomiell in der Größe der Probleminstanz wächst. Effiziente Algorithmen dürfen zusätzlich Zufall verwenden. Wir bezeichnen die Klasse aller effizienten Algorithmen als PPT (probabilistic polynomial time).

Aus kryptographischer Sicht ist wichtig, zu quantifizieren wann ein Algorithmus keinen, beziehungsweise nur vernachlässigbaren Erfolg bei der Lösung eines Problems hat.

**Definition 2** (Vernachlässigbare Funktion)**.** *Eine Funktion* $\nu\colon \mathbb{N} \to \mathbb{R}$ *heißt* vernachlässigbar *(*negligible*), falls sie schneller als jedes Polynom fällt. Das heißt, falls für jedes* $c \in \mathbb{N}$ *ein* $n_0 \in \mathbb{N}$ *existiert, sodass für alle* $n > n_0$ *gilt:*

$$|\nu(n)| < n^{-c}$$

*Wir nennen* $1 - \nu(n)$ *dann* überwältigend*. Wir nutzen für eine nicht spezifizierte in* $n$ *vernachlässigbare Funktion auch die Bezeichung* $\mathsf{negl}(n)$*.*

*Eine Funktion* $f\colon \mathbb{N} \to \mathbb{R}$ *ist* nicht-vernachlässigbar *(*non-negligible*), falls sie nicht vernachlässigbar ist, d.h., falls für ein* $c \in \mathbb{N}$ *gilt, dass* $f(n) \geq n^{-c}$ *für unendlich viele* $n \in \mathbb{N}$*.*

**Definition 3** (Ununterscheidbarkeit)**.** *Zwei Wahrscheinlichkeitsverteilungen* $A$ *und* $B$ *sind ununterscheidbar, falls für alle PPT-Angreifer* $\mathcal{A}$ *gilt, dass*

$$|\Pr[\mathcal{A}(A) = 1] - \Pr[\mathcal{A}(B) = 1]|$$

*vernachlässigbar ist.*

## A.2 Formale Definitionen zu Verschlüsselungsverfahren

**Definition 4** (PKE-Verfahren)**.** *Ein PKE-Verfahren (für Nachrichtenraum* $\mathcal{M}$*) ist ein Tupel* $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ *von PPT-Algorithmen.*

$\mathrm{Gen}(1^n)$ *generiert bei Eingabe des unären Sicherheitsparameters* $1^n$ *ein Schlüsselpaar bestehend aus einem öffentlichem und geheimem Schlüssel* $(\mathsf{pk}, \mathsf{sk})$*.*

$\mathrm{Enc}(\mathsf{pk}, m)$ *erzeugt bei Eingabe eines öffentlichen Schlüssels* $\mathsf{pk}$ *und einer Nachricht* $m$ *ein Chiffrat* $c$*.*

$\mathrm{Dec}(\mathsf{sk}, c)$ *gibt bei Eingabe eines geheimen Schlüssels* $\mathsf{sk}$ *und eines Chiffrats* $c$ *entweder eine Nachricht* $m$ *oder ein Fehlersymbol* $\perp$ *aus.*

*Für alle Nachrichten* $m$ *in* $\mathcal{M}$ *ist die Wahrscheinlichkeit*

$$\Pr\left[\mathrm{Dec}(\mathsf{sk}, \mathrm{Enc}(\mathsf{pk}, m)) = m \mid (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{Gen}(1^n)\right]$$

*überwältigend, wobei die Wahrscheinlichkeit über den Zufall von* Gen *und* Enc *gemeint ist.*

**Definition 5** (IND-CPA-Sicherheit)**.** *Sei* $\mathsf{PKE} = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ *ein PKE-Verfahren.* $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ *ist IND-CPA-sicher, wenn für alle PPT-Angreifer* $\mathcal{A}$ *die Wahrscheinlichkeit*

$$\Pr[Exp^{\text{IND-CPA}}_{\mathsf{PKE},\mathcal{A}}(n) = 1]$$

*vernachlässigbar in* $n$ *ist, wobei das Sicherheitsspiel* $Exp^{\text{IND-CPA}}_{\mathsf{PKE},\mathcal{A}}(n)$ *in Abbildung 38 definiert ist.*

$\underline{Exp^{\text{IND-CPA}}_{\mathsf{PKE},\mathcal{A}}(n)}$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \text{Gen}(1^n)$
$(m_0, m_1) \leftarrow \mathcal{A}(1^n, \mathsf{pk})$
$b \leftarrow \{0,1\}$
$c \leftarrow \text{Enc}(\mathsf{pk}, m_b)$
$b' \leftarrow \mathcal{A}(1^n, c)$
**return** $b \stackrel{?}{=} b'$

Abbildung 38: Das IND-CPA-Sicherheitsspiel.

$\underline{Exp^{\text{IND-CCA}}_{\mathcal{A}}(n)}$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \text{Gen}(1^n)$
$(m_0, m_1) \leftarrow \mathcal{A}^{\text{Dec}(\mathsf{sk},\cdot)}(1^n, \mathsf{pk})$
$b \leftarrow \{0,1\}$
$c \leftarrow \text{Enc}(\mathsf{pk}, m_b)$
$b' \leftarrow \mathcal{A}^{\text{Dec}(\mathsf{sk},\cdot)}(1^n, c)$
**return** $b \stackrel{?}{=} b'$

Abbildung 39: Das IND-CCA-Sicherheitsspiel.

**Definition 6** (IND-CCA-Sicherheit)**.** *Sei* $\mathsf{PKE} = (\text{Gen}, \text{Enc}, \text{Dec})$ *ein PKE-Verfahren.* $(\text{Gen}, \text{Enc}, \text{Dec})$ *ist IND-CCA-sicher, wenn für alle PPT-Angreifer* $\mathcal{A}$ *die Wahrscheinlichkeit*

$$\Pr[Exp^{\text{IND-CCA}}_{\mathsf{PKE},\mathcal{A}}(n) = 1]$$

*vernachlässigbar ist, wobei das Sicherheitsspiel* $Exp^{\text{IND-CCA}}_{\mathsf{PKE},\mathcal{A}}(n)$ *in Abbildung 39 definiert ist.*

**Definition 7** (Somewhat Homomorphic Encryption, [10])**.** *Sei* $\mathcal{C}$ *eine Menge von Schaltkreisen. Ein SHE-Verfahren (für Nachrichtenraum* $\mathcal{M}$*) ist ein Tupel* $(\text{Gen}, \text{Enc}, \text{Dec}, \text{Eval})$ *von PPT-Algorithmen, sodass* $(\text{Gen}, \text{Enc}, \text{Dec})$ *ein IND-CPA-sicheres PKE-Verfahren ist und* Eval *bei Eingabe eines öffentlichen Schlüssels* $\mathsf{pk}$*, eines* $k$*-ären Schaltkreises* $C \in \mathcal{C}$ *und* $k$ *Chiffraten* $c_1, \ldots, c_k$ *ein Chiffrat* $c$ *ausgibt, sodass folgendes gilt:*

**Korrektheit.** *Für alle Nachrichten* $m$ *in* $\mathcal{M}$ *und Schaltkreise* $C \in \mathcal{C}$ *ist*

$$\Pr_{(\mathsf{pk},\mathsf{sk}) \leftarrow \text{Gen}(1^n)} [\text{Dec}(\mathsf{sk}, \text{Eval}(C, \text{Enc}(\mathsf{pk}, m))) = C(m)]$$

*überwältigend, wobei die Wahrscheinlichkeit über die Wahl von* $(\mathsf{pk}, \mathsf{sk})$ *und den Zufall von* Enc *gemeint ist.*

**Definition 8** (Leveled Homomorphic Encryption, [10])**.** *Ein LHE-Verfahren (für Nachrichtenraum* $\mathcal{M}$*) ist ein Tupel* $(\text{Gen}, \text{Enc}, \text{Dec}, \text{Eval})$ *von PPT-Algorithmen, sodass* $(\text{Gen}, \text{Enc}, \text{Dec})$ *ein IND-CPA-sicheres PKE-Verfahren ist,* Gen *eine zusätzliche Eingabe* $d$ *erwartet, und* Eval *bei Eingabe eines öffentlichen Schlüssels* $\mathsf{pk}$*, eines beliebigen* $k$*-ären polynomiell großen Schaltkreises* $C$ *maximaler Tiefe* $d$ *und* $k$ *Chiffraten* $c_1, \ldots, c_k$ *ein Chiffrat* $c$ *ausgibt, sodass folgendes gilt:*

**Korrektheit.** *Für alle* $d \in \mathbb{N}$*, für alle Nachrichten* $m$ *in* $\mathcal{M}$ *und alle* $k$*-ären Schaltkreise* $C$ *polynomieller Größe und maximaler Tiefe* $d$ *ist*

$$\Pr_{(\mathsf{pk},\mathsf{sk}) \leftarrow \text{Gen}(1^n, d)} [\text{Dec}(\mathsf{sk}, \text{Eval}(C, \text{Enc}(\mathsf{pk}, m))) = C(m)]$$

*überwältigend, wobei die Wahrscheinlichkeit über die Wahl von* $(\mathsf{pk}, \mathsf{sk})$ *und den Zufall von* Enc *gemeint ist.*

**Kompaktheit.** *Es existiert ein Polynom $p$, so dass für jedes Schlüsselpaar $(\mathsf{pk}, \mathsf{sk})$, das von $\mathrm{Gen}(1^n, d)$ ausgegeben wird, für jeden Schaltkreis $C$ maximaler Tiefe $d$ und für alle Chiffrate $c_1, \ldots, c_k$ die Größe der Ausgabe von $\mathrm{Eval}(\mathsf{pk}, C, c_1, \ldots, c_k)$ höchstens $p(n)$ Bits groß ist, unabhängig von der Größe des ausgewerteten Schaltkreises und unabhängig von der maximalen Tiefe $d$.*

Der Unterschied zwischen SHE und LHE besteht darin, dass die Schaltkreistiefe, mit der ein SHE-Verfahren umgehen kann, durch Parameterwahl verändert werden kann. Mit steigender Schaltkreistiefe steigt üblicherweise die Chiffratgröße. Ein LHE-Verfahren kann alle Schaltkreise einer bei Schlüsselerzeugung festgelegten Tiefe auswerten. Die Chiffratgröße ist unabhängig von der maximale auswertbaren Schaltkreistiefe.

**Definition 9** (Fully Homomorphic Encryption, [10])**.** *Ein FHE-Verfahren (für Nachrichtenraum $\mathcal{M}$) ist ein Tupel* $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec}, \mathrm{Eval})$ *von PPT-Algorithmen, sodass* $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ *ein IND-CPA-sicheres PKE-Verfahren ist und* Eval *bei Eingabe eines öffentlichen Schlüssels* $\mathsf{pk}$*, eines* beliebigen *polynomiell großen $k$-ären Schaltkreises $C$ und $k$ Chiffraten $c_1, \ldots, c_k$ ein Chiffrat $c$ ausgibt.*

**Korrektheit.** *Für alle Nachrichten $m$ in $\mathcal{M}$ und alle $k$-ären Schaltkreise $C$ polynomieller Größe ist*

$$\Pr_{(\mathsf{pk},\mathsf{sk}) \leftarrow \mathrm{Gen}(1^n)} \left[\mathrm{Dec}\left(\mathsf{sk}, \mathrm{Eval}\left(C, \mathrm{Enc}(\mathsf{pk}, m)\right)\right) = C(m)\right]$$

*überwältigend, wobei die Wahrscheinlichkeit über die Wahl von* $(\mathsf{pk}, \mathsf{sk})$ *und den Zufall von* Enc *gemeint ist.*

**Kompaktheit.** *Es existiert ein Polynom $p$, so dass für jedes Schlüsselpaar $(\mathsf{pk}, \mathsf{sk})$, das von $\mathrm{Gen}(1^n)$ ausgegeben wird, für jeden Schaltkreis $C$ polynomieller Größe und für alle Chiffrate $c_1, \ldots, c_k$ die Größe der Ausgabe von $\mathrm{Eval}(\mathsf{pk}, C, c_1, \ldots, c_k)$ höchstens $p(n)$ Bits groß ist, unabhängig von der Größe des ausgewerteten Schaltkreises.*

**Definition 10** (Circuit Privacy)**.** *Ein SHE-, LHE- oder FHE-Verfahren erfüllt Circuit Privacy, falls für jedes von $\mathrm{Gen}(1^n, \cdot)$ erzeugtes Schlüsselpaar $(\mathsf{pk}, \mathsf{sk})$, für alle von dem jeweiligen Verfahren unterstützten Schaltkreise $C$, für alle Nachrichten $m_i \in \mathcal{M}$ und alle möglichen Chiffrate $c_i \in \mathsf{supp}(\mathrm{Enc}(\mathsf{pk}, m_i))$ $(1 \leq i \leq k)$ die Verteilungen*

$$\{\mathrm{Eval}(\mathsf{pk}, C, c_1, \ldots, c_k)\} \quad \textit{und} \quad \{\mathrm{Enc}(\mathsf{pk}, C(m_1, \ldots, m_k)\}$$

*ununterscheidbar sind.*

## A.3 Sicherheitsannahmen

**Definition 11** (DDH-Annahme)**.** *Sei $\mathrm{Gen}_G$ ein PPT-Generatoralgorithmus, der bei Eingabe $1^n$ eine endliche zyklische Gruppe $G$ der Ordnung $q = q(n)$ mit Erzeuger $g$ ausgibt. Die DDH-Annahme gilt relativ zu $\mathrm{Gen}_G$, wenn die Verteilungen $\{(g^a, g^b, g^c) \mid (G, q, g) \leftarrow \mathrm{Gen}_G(1^n), a, b, c \leftarrow \mathbb{Z}_q\}$ und $\{(g^a, g^b, g^{ab}) \mid (G, q, g) \leftarrow \mathrm{Gen}_G(1^n), a, b \leftarrow \mathbb{Z}_q\}$ ununterscheidbar sind.*

**Definition 12** (RSA-Annahme)**.** *Sei $\mathrm{Gen}_{\mathsf{RSA}}$ ein PPT-Generatoralgorithmus, welcher bei Eingabe $1^n$ zwei zufällige Primzahlen $2^n \leq p, q \leq 2^{n+1}$ erzeugt, eine zufällige Zahl $2 \leq e < N := p \cdot q$ zieht und $(N, e, d := e^{-1} \bmod \varphi(N))$ ausgibt.*
*Die* RSA-Annahme *gilt relativ zu $\mathrm{Gen}_{\mathsf{RSA}}$, wenn für jeden PPT-Angreifer $\mathcal{A}$ die Wahrscheinlichkeit*

$$\Pr[x^e = y \bmod N \mid (N, e, d) \leftarrow \mathrm{Gen}_{\mathsf{RSA}}(1^n), y \leftarrow \mathbb{Z}_N, \mathcal{A}(N, e, y) = x]$$

*vernachlässigbar ist.*

Seien $n, q$ natürliche Zahlen und $\chi$ eine Gaußverteilung über $\mathbb{Z}$.

**Definition 13** (LWE-Annahme, [121]). *Für einen Vektor $\mathbf{s} \in \mathbb{Z}_q^n$ zieht die LWE-Verteilung $A_{\mathbf{s},\chi}$ über $\mathbb{Z}_q^n \times \mathbb{Z}_q$ zunächst einen Vektor $\mathbf{a} \in \mathbb{Z}_q^n$ zufällig gleichverteilt und einen Fehler $e \leftarrow \chi$ und gibt $(a, b = \langle s, a \rangle + e \bmod q)$ aus.*
*Die LWE-Annahme gilt relativ zu $(n, q, \chi)$, wenn die zwei Verteilungen*

$$\{(\mathbf{a_1}, b_1), \ldots, (\mathbf{a_m}, b_m) \mid \mathbf{s} \leftarrow \mathbb{Z}_q^n, (\mathbf{a_i}, b_i) \leftarrow A_{\mathbf{s},\chi}\}$$
$$\{(\mathbf{a_1}, b_1), \ldots, (\mathbf{a_m}, b_m) \mid (\mathbf{a_i}, b_i) \leftarrow \mathbb{Z}_q^n \times \mathbb{Z}_q\}$$

*ununterscheidbar sind.*

Sei $R$ ein Ring von Grad $n$ über $\mathbb{Z}$. Sei $q$ eine natürliche Zahl, $R_q := R/qR$ und $\chi$ eine Fehlerverteilung über $R$.

**Definition 14** (RLWE-Annahme, [121]). *Für ein $s \in R_q$ zieht die RLWE-Verteilung $A_{s,\chi}$ über $R_q \times R_q$ zunächst ein Element $a \in R_q$ zufällig gleichverteilt und einen Fehler $e \leftarrow \chi$ und gibt $(a, b = s \cdot a + e \bmod q)$ aus.*
*Die RLWE-Annahme gilt relativ zu $(R, q, \chi)$, wenn die zwei Verteilungen*

$$\{(a_1, b_1), \ldots, (a_m, b_m) \mid s \leftarrow R_q, (\mathbf{a_i}, b_i) \leftarrow A_{s,\chi}\}$$
$$\{(a_1, b_1), \ldots, (a_m, b_m) \mid (a_i, b_i) \leftarrow R_q \times R_q\}$$

*ununterscheidbar sind.*

# B Formale Definitionen zu MPC

## B.1 Setting, n-Parteien-Funktionen, Protokolle

Sei $f\colon \mathbb{N} \times (\{0,1\}^*)^n \times \{0,1\}^* \to (\{0,1\}^*)^n$ eine in Polynomialzeit berechenbare Funktion. Wir interpretieren $f$ als $n$-Parteien-Funktion: Der erste Parameter ist der Sicherheitsparameter, die weiteren $n$ Parameter sind die Eingaben der $n$ Parteien, der letzte Parameter ist der Zufall von $f$. Die Ausgabe der $i$-ten Partei ist das $i$-te Element der Ausgabe von $f$. (Alternativ können wir $f$ auch als $n$-Tupel $f = (f_1, \ldots, f_n)$ betrachten.)

Wir möchten nun Aussagen über Protokolle $\pi$ machen, die $f$ berechnen. Ein $n$-Parteien-Protokoll $\pi$ ist eine Menge aus $n$ PPT-ITMs, wobei $P_i$ die $i$-te Partei (PPT-ITM) bezeichnet. Neben den $n$ Protokollparteien gibt es mit dem Angreifer $\mathcal{A}$ eine weitere PPT-ITM. Der Angreifer $\mathcal{A}$ kontrolliert eine Teilmenge $C \subseteq \mathbb{Z}_n$ der Parteien $P_i$ (die korrumpierten Parteien) und nimmt an ihrer Stelle am Protokoll teil. Je nach Korruptionsmodell muss sich der Angreifer dabei an die Protokollbeschreibung halten (passive Korruption, *semi-honest*), oder darf beliebig davon abweichen (aktive Korruption, *malicious*). Steht die Menge $C$ der korrumpierten Parteien vor Beginn der Ausführung fest, so heißt $\mathcal{A}$ *statischer* Angreifer. Kann $\mathcal{A}$ zur Protokolllaufzeit Parteien korrumpieren, heißt $\mathcal{A}$ *adaptiver* Angreifer. Wir betrachten hier nur statische Angreifer. Weiterhin kontrolliert $\mathcal{A}$ das nicht vertrauenswürdige Kommunikationsnetzwerk, kann also je nach Modell beispielsweise Nachrichten unterdrücken oder verändern.

## B.2 Sicherheitsdefinitionen

Wir beginnen zunächst mit der Frage, was Sicherheit im Kontext von Mehrparteienberechnungen bedeutet.

### B.2.1 Sicherheit als Listeneigenschaften

Oftmals wird Sicherheit als eine Liste von Eigenschaften mit jeweils eigenen Sicherheitsspielen definiert. Wir könnten für die Sicherheit eines Protokolls $\pi$ für eine $n$-Parteien-Funktion $f$ folgende Kriterien fordern (informell):

1. Korrektheit: $\pi$ muss $f$ für ehrliche Parteien korrekt berechnen.

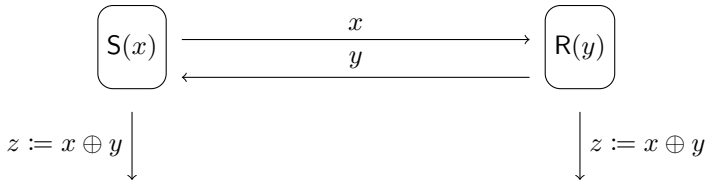



Abbildung 40: (Un-)„Sichere“ Mehrparteienberechnung eines XORs

2. Input-Privacy: $\pi$ soll einer Partei $P_i$ nichts über die Eingaben der anderen Parteien verraten, das nicht aus der Ausgabe und der eigenen Eingabe von $P_i$ berechnet werden kann.

Für beispielsweise ein sicheres 1-aus-2-OT scheinen diese Eigenschaften hinreichend zu sein. Betrachten wir nun das Zwei-Parteien-Protokoll aus Abbildung 40 zur Berechnungen des bitweisen XOR, also $f(n, x_1, x_2) = (x_1 \oplus x_2, x_1 \oplus x_2)$.

Dieses Protokoll erfüllt offensichtlich die beiden Forderungen (wobei Input-Privacy trivial ist). Es ist jedoch nicht *sicher*, da Bob seine Eingabe in Abhängigkeit von Alices wählen kann. Damit kann Bob die Ausgabe frei bestimmen. Es ist klar, dass ein sicheres Protokoll also auch *Unabhängigkeit der Eingaben* (engl. *independence of inputs*) garantieren sollte. Es stellt sich nun die Frage ist, ob wir nicht noch eine weitere nötige Sicherheitseigenschaft übersehen haben. Die Sicherheit als Liste von Eigenschaften zu definieren birgt also die Gefahr, wichtige Eigenschaften zu übersehen – insbesondere bei komplexen Funktionalitäten.

### B.2.2 Das Real-Ideal-Paradigma

Im Folgenden führen wir zuerst die Ausführungssemantik für die reale sowie die ideale Ausführung in Anwesenheit eines aktiven Angreifers ein und definieren anschließend Sicherheit.

#### Reale Ausführung

- **Eingaben:** Jede Partei $P_i$ hat (private) Eingabe $x_i \in \{0,1\}^*$ und Zufall $r_i$. Der Angreifer $\mathcal{A}$ hat *auxiliary input* $z$ (mit der Menge der korrumpierten Parteien $C$ und deren Eingaben $\{x_i \mid i \in C\}$ beginnend) und Zufall $r$. Alle Parteien haben gemeinsame (erste) Eingabe $1^n$. Wir definieren $\mathbf{x} = (x_1, \ldots, x_n)$ und $\mathbf{r} = (r, r_1, \ldots, r_n)$.

- **Nachrichtenberechnung:** Die Nachrichten der ehrlichen Parteien werden nach Vorgabe des Protokolls $\pi$ erstellt. Die Nachrichten der korrumpierten Parteien werden vom Angreifer (im Rahmen seiner Laufzeit und des Angreifermodells) beliebig gewählt. Betrachten wir einen semi-honest Angreifer, so werden auch die Nachrichten der korrupierten Parteien nach Vorgabe von $\pi$ gewählt.

- **Nachrichtenübertragung:** Wir nehmen ein *asynchrones, ideal authentifiziertes* Netzwerk mit *Punkt-zu-Punkt-Verbindungen* als Kommunikationsmodell an: Nachrichten der ehrlichen Parteien werden *zuerst an den Angreifer geschickt*, der Sender, Empfänger sowie Inhalt lernt. Für jede Nachricht kann der Angreifer entscheiden, ob die Nachricht *zugestellt* wird oder nicht. Nicht sofort zugestellte Nachrichten können auf Wunsch des Angreifers zu einem späteren Zeitpunkt zugestellt werden. Der Angreifer kann jedoch *keine* Nachrichten der ehrlichen Parteien *verändern* oder in ihrem Namen *erstellen*. (Man könnte hier auch andere Sachen annehmen, z. B. ein Broadcast-Channel, sichere Kanäle, synchrone Ausfürhung, etc. Hier ergeben sich teilweise unterschiedliche Sicherheitsbegriffe)

- **Ausgabe:** Am Ende der Berechnung erzeugen alle Parteien lokal ihre Ausgabe. Die unkorrumpierten Parteien geben aus, was das Protokoll $\pi$ spezifiziert. Die korrumpierten Parteien geben ein spezielles Symbol $\top$ aus. $\mathcal{A}$ gibt eine beliebige PPT-berechenbare Funktion seiner Sicht aus, wobei wir im folgenden immer o. B. d. A. davon ausgehen, dass der Angreifer direkt seine Sicht ausgibt.

Sei $\mathrm{EXEC}_{\pi,\mathcal{A}}(n,z,\mathbf{x},\mathbf{r})_i$ die Ausgabe der Partei $P_i$ im Protokoll $\pi$ mit Angreifer $\mathcal{A}$, Sicherheitsparameter $n$, *auxiliary input* $z$ und Eingaben $\mathbf{x}$. Sei $\mathrm{EXEC}_{\pi,\mathcal{A}}(n,z,\mathbf{x},\mathbf{r})_{\mathcal{A}}$ Ausgabe des Angreifers $\mathcal{A}$ im Protokoll $\pi$ mit Sicherheitsparameter $n$, *auxiliary input* $z$, Eingaben $\mathbf{x}$. Wir definieren

$$\mathrm{EXEC}_{\pi,\mathcal{A}}(n,z,\mathbf{x},\mathbf{r}) \coloneqq (\mathrm{EXEC}_{\pi,\mathcal{A}}(n,z,\mathbf{x},\mathbf{r})_{\mathcal{A}},\ \mathrm{EXEC}_{\pi,\mathcal{A}}(n,z,\mathbf{x},\mathbf{r})_1,\ \ldots,\ \mathrm{EXEC}_{\pi,\mathcal{A}}(n,z,\mathbf{x},\mathbf{r})_n).$$

$\mathrm{EXEC}_{\pi,\mathcal{A}}(n,z,\mathbf{x})$ bezeichnet die zu $\mathrm{EXEC}_{\pi,\mathcal{A}}(n,z,\mathbf{x},\mathbf{r})$ gehörende Zufallsvariable, wobei $\mathbf{r}$ gleichverteilt zufällig gewählt ist. $\mathrm{EXEC}_{\pi,\mathcal{A}}$ bezeichnet das *uniforme Wahrscheinlichkeitsensemble* $\{\mathrm{EXEC}_{\pi,\mathcal{A}}(n,z,\mathbf{x});\ (z,\mathbf{x}) \leftarrow \mathsf{G}(1^n)\}_{n\in\mathbb{N}}$ für eine PPT-Eingabemaschine $\mathsf{G}$.

**Ideale Ausführung.** In der idealen Ausführung betrachten wir kein Protokoll, bei dem sich die Protokollparteien $P_1,\ldots P_n$ Nachrichten zuschicken. Stattdessen schicken alle Parteien ihre Eingabe zu einer per Definition vertrauenswürdigen PPT-ITM $\mathcal{F}$, der idealen Funktionalität. Nach Erhalt aller Eingaben wertet $\mathcal{F}$ die Funktion $f$ aus und gibt jeder Partei eine Ausgabe. Dem Realweltangreifer $\mathcal{A}$ steht der Idealangreifer $\mathsf{Sim}$, auch Simulator genannt, gegenüber. Er wählt die Eingaben der korrumpierten Parteien, interagiert mit $\mathcal{F}$ und gibt am Ende eine Ausgabe aus.

- **Eingaben:** Jede Partei $P_i$ hat (private) Eingabe $x_i \in \{0,1\}^*$ und *keinen Zufall*[18]. Der Angreifer $\mathsf{Sim}$ hat *auxiliary input* $z$ (mit der Menge der korrumpierten Parteien $C$ und deren Eingaben $\{x_i \mid i \in C\}$ beginnend) und Zufall $r$. Die ideale Funktionalität $\mathcal{F}$ hat keine private Eingabe und Zufall $r_{\mathcal{F}}$. Alle Parteien (auch die ideale Funktionalität) haben gemeinsame (erste) Eingabe $1^n$. Wir definieren $\mathbf{x} \coloneqq (x_1,\ldots,x_n)$ und $\mathbf{r} = (r, r_{\mathcal{F}})$.
- **Eingaben an $\mathcal{F}$:** Die ehrlichen Parteien schicken ihre Eingabe an die ideale Funktionalität $\mathcal{F}$. Für jede korrumpierte Partei $P_i$ schickt der Angreifer entweder einen speziellen Wert `abort` zum Abbrechen, die Eingabe $x_i$ oder einen anderen Wert $x_i'$. Sei $\mathbf{x}' = (x_1',\ldots,x_n')$ der resultierende Eingabevektor, wobei $x_i' = x_i$ für jede ehrliche Partei $P_i$.
- **Frühes Abbrechen:** Wenn $\mathcal{F}$ eine Eingabe der Form `abort` von einer korrumpierten Partei erhält, sendet sie $\perp$ an alle Parteien als Ausgabe. Die Ausführung hält an. Anderenfalls geht die Ausführung mit dem nächsten Schritt weiter.
- **$\mathcal{F}$ sendet Ausgabe an $\mathsf{Sim}$:** $\mathcal{F}$ berechnet nun $(y_1,\ldots,y_n) = f(n,\mathbf{x}',r_{\mathcal{F}})$. Für alle korrumpierten Parteien $P_i$ schickt $\mathcal{F}$ die Ausgabe $y_i$ an den Angreifer.
- **$\mathsf{Sim}$ instruiert $\mathcal{F}$:** Der Angreifer sendet entweder `continue` oder `abort` an $\mathcal{F}$. Im Fall von `continue` schickt $\mathcal{F}$ jeder ehrlichen Partei $P_i$ ihre Ausgabe $y_i$. Sendet $\mathsf{Sim}$ `abort`, bekommen die ehrlichen Parteien die Ausgabe $\perp$.
- **Ausgaben:** Die ehrlichen Parteien geben immer die von $\mathcal{F}$ erhaltene Ausgabe aus. Korrumpierte Parteien geben immer $\top$ aus. Der Simulator $\mathsf{Sim}$ gibt eine beliebige (PPT-berechenbare) Funktion der echten Eingaben der korrumpierten Parteien, seines Zufalls $r$, seines *auxiliary inputs* $z$ und der Ausgaben der korrumpierten Parteien aus.

Sei $\mathrm{IDEAL}_{\mathcal{F},\mathsf{Sim}}(n,z,\mathbf{x},\mathbf{r})_i$ die Ausgabe der Partei $P_i$ in der idealen Ausführung mit idealer Funktionalität $\mathcal{F}$, Angreifer $\mathsf{Sim}$, Sicherheitsparameter $n$, *auxiliary input* $z$, Eingaben $\mathbf{x}$ und Zufall $\mathbf{r}$. Ebenfalls sei $\mathrm{IDEAL}_{\mathcal{F},\mathsf{Sim}}(n,z,\mathbf{x},\mathbf{r})_{\mathsf{Sim}}$ die Ausgabe des Angreifers $\mathsf{Sim}$ in der idealen Ausführung mit idealer Funktionalität $\mathcal{F}$, Sicherheitsparameter $n$, *auxiliary input* $z$, Eingaben $\mathbf{x}$ und Zufall $\mathbf{r}$. Wir definieren

$$\mathrm{IDEAL}_{\mathcal{F},\mathsf{Sim}}(n,z,\mathbf{x},\mathbf{r}) \coloneqq \\ (\mathrm{IDEAL}_{\mathcal{F},\mathsf{Sim}}(n,z,\mathbf{x},\mathbf{r})_{\mathsf{Sim}}, \mathrm{IDEAL}_{\mathcal{F},\mathsf{Sim}}(n,z,\mathbf{x},\mathbf{r})_1,\ldots,\mathrm{IDEAL}_{\mathcal{F},\mathsf{Sim}}(n,z,\mathbf{x},\mathbf{r})_n).$$

Die ideale Ausgabe $\mathrm{IDEAL}_{\mathcal{F},\mathsf{Sim}}(n,z,\mathbf{x})$ bezeichnet die zu $\mathrm{IDEAL}_{\mathcal{F},\mathsf{Sim}}(n,z,\mathbf{x},\mathbf{r})$ gehörende Zufallsvariable, wobei $\mathbf{r}$ gleichverteilt zufällig gewählt ist. $\mathrm{IDEAL}_{\mathcal{F},\mathsf{Sim}}$ bezeichnet das *uniforme Wahrscheinlichkeitsensemble* $\{\mathrm{IDEAL}_{\mathcal{F},\mathsf{Sim}}(n,z,\mathbf{x});(z,\mathbf{x}) \leftarrow \mathsf{G}(1^n)\}_{n\in\mathbb{N}}$ für eine PPT-Eingabemaschine $\mathsf{G}$.

[18] Aufgrund des Ablaufs der idealen Ausführung benötigen die Parteien keinen Zufall.

**Definition 15.** *Sei $f$ eine $n$-Parteien-Funktion, $\mathcal{F}$ die dazugehörige ideale Funktionalität und sei $\pi$ ein Protokoll, das $f$ berechnet. Wir sagen, dass $\pi$ die Funktion $f$* sicher mit Abort in Anwesenheit von statisch korrumpierenden aktiven Angreifern *berechnet, wenn für alle PPT-Angreifer $\mathcal{A}$ für die reale Ausführung ein PPT-Simulator* $\mathsf{Sim}$ *für die ideale Ausführung existiert, sodass*

$$\mathrm{EXEC}_{\pi,\mathcal{A}} \overset{c}{\approx} \mathrm{IDEAL}_{\mathcal{F},\mathsf{Sim}}$$

*gilt. In diesem Fall schreiben wir auch $\pi \geq \mathcal{F}$ und sagen „$\pi$ realisiert $\mathcal{F}$“.*

Die Standard-Vorgehensweise eines Simulators besteht darin, den Realweltangreifer $\mathcal{A}$ intern auszuführen und mit passenden Protokollnachrichten für die ehrlichen Parteien zu versorgen (deren Eingaben der Simulator jedoch nicht notwendigerweise kennt). Sind die Nachrichten an $\mathcal{A}$ korrekt verteilt, ist auch die Sicht von $\mathcal{A}$, die der Simulator simulieren muss, korrekt verteilt. Der Simulator muss weiterhin die ideale Funktionalität $\mathcal{F}$ so mit Eingaben für die korrumpierten Parteien bedienen, dass die ehrlichen Parteien eine korrekt verteilte Ausgabe erhalten. In vielen Fällen gibt es genau eine passende Eingabe für die korrumpierten Parteien, die zur korrekten Ausgabe der ehrlichen Parteien führt. Ein sicheres Protokoll $\pi$ wird (in der Regel) die (impliziten) Eingaben der korrumpierten Parteien vor den ehrlichen Parteien verstecken. Der Simulator $\mathsf{Sim}$ muss diese also *extrahieren*. Als Vorteil gebenüber dem Realweltangreifer dient ihm dabei die Fähigkeit, diesen zu *rewinden*.

### B.2.3 Protokollkomposition

Anstatt ein Protokoll als eine große Einheit zu entwerfen und als sicher zu beweisen bietet es sich oftmals an, Protokolle *modular* zu entwerfen: Das zu lösende Problem wird in Teil-Probleme aufgeteilt, für die sichere Protokolle entworfen werden. Ein Rahmenprotokoll kann nun diese Teilprotokolle verwenden, um das große Problem zu lösen. Das Rahmenprotokoll soll dabei die Sicherheit der Teilprotokolle „erben“: Lösen diese ein Teilproblem sicher, so soll auch das Gesamtprotokoll sicher sein. Die Sicherheit der Teilprotokolle und des Rahmenprotokolls kann also jeweils einzeln betrachtet werden.

Wir betrachten im Folgenden eine Variante der realen Ausführung, in der die Protokollparteien ideale Funktionalitäten $\mathcal{F}_1, \ldots, \mathcal{F}_m$ als Subroutinen aufrufen können (Hybridausführung). Gilt $\pi_i \geq \mathcal{F}_i$ für ein Protokoll $\pi_i$ und erfolgen Aufrufe von $\mathcal{F}_i$ sequentiell, können wir die Ausführung von $\mathcal{F}_i$ durch $\pi_i$ ersetzen. Dies liefert uns ein (einfaches) Kompositionstheorem.

#### Das Hybridmodell

Seien $\mathcal{F}_1, \ldots, \mathcal{F}_m$ ideale $n$-Parteien-Funktionalitäten. Die Ausführung eines Protokolls $\pi$ im $\mathcal{F}_1, \ldots, \mathcal{F}_m$-Hybridmodell ist, bis auf folgende Änderungen, wie die reale Ausführung in Anhang B.2.2 definiert:

- Wenn $\pi$ (möglicherweise in Abhängigkeit vom Sicherheitsparameter) den Aufruf von $\mathcal{F} \in \{\mathcal{F}_1, \ldots, \mathcal{F}_m\}$ vorsieht, wird das Protokoll für $\mathcal{F}$ ausgeführt.
- Alle ehrlichen Partien führen die Ausgabe von $\pi$ erst dann fort, wenn sie Ausgabe von $\mathcal{F}$ erhalten haben.

Es wird also zu jedem Zeitpunkt entweder $\pi$ oder *genau eine* Instanz von $\mathcal{F}$ ausgeführt.

Sei $\mathrm{EXEC}^{\mathcal{F}_1,\ldots,\mathcal{F}_m}_{\pi,\mathcal{A}}(n, z, \mathbf{x})$ die Ausgabe der Berechnung im $\mathcal{F}_1, \ldots, \mathcal{F}_m$-Hybridmodell mit Protokoll $\pi$, Angreifer $\mathcal{A}$, Sicherheitsparameter $n$, Eingaben $\mathbf{x}$ und *auxiliary input* $z$. $\mathrm{EXEC}^{\mathcal{F}_1,\ldots,\mathcal{F}_m}_{\pi,\mathcal{A}}$ bezeichnet das *uniform probability ensemble* $\{\mathrm{EXEC}^{\mathcal{F}_1,\ldots,\mathcal{F}_m}_{\pi,\mathcal{A}}(n, z, \mathbf{x})\}_{n\in\mathbb{N},(z,\mathbf{x})\leftarrow G(1^n)}$.

Wir wollen nun die Ausführung einer Funktionalität $\mathcal{F} \in \{\mathcal{F}_1, \ldots, \mathcal{F}_m\}$ durch die Ausführung von $\rho$ ersetzen, wobei $\rho \geq \mathcal{F}$ ($\rho$ realisiert $\mathcal{F}$) gilt.

#### Ersetzung von idealen Funktionalitäten

Im Folgenden beschreiben wir die Ersetzung des idealen Protokolls $\mathcal{F}$ durch das reale Protokoll $\rho$, das $\mathcal{F}$ realisiert:

1. Jede Partei $P_i$ speichert vor Aufruf von $\mathcal{F}$ ihren internen Zustand (relativ zum Protokoll $\pi$) auf einem speziellen Band. Sei $\sigma_i$ dieser Zustand.

2. Der Aufruf von $\mathcal{F}$ wird mit einem Aufruf des Codes von $P_i$ für das Protokoll $\rho$ ersetzt. Die Eingabe $x_i^\rho$ ist dabei der Wert, den $P_i$ an $\mathcal{F}$ laut Protokoll $\pi$ geschickt hätte. Der Zufall $r_i^\rho$ wird gleichverteilt zufällig gewählt.

3. Sei $v_i^\rho$ die Ausgabe der Partei $P_i$ in Protokoll $\rho$. $P_i$ fährt mit der Ausführung von $\pi$ im Zustand $\sigma_i$ fort, wobei die Ausgabe von $\mathcal{F}$ durch $v_i^\rho$ ersetzt wird.

Sei $\pi^{\rho_1,\ldots,\rho_m}$ das Protokoll $\pi$ (ursprünglich im $\mathcal{F}_1,\ldots,\mathcal{F}_m$-Hybridmodell), bei dem Aufrufe von $\mathcal{F}_i$ durch Aufrufe von $\rho_i$ ersetzt wurden.

**Theorem 1** (Kompositionstheorem)**.** *Seien $\mathcal{F}_1,\ldots,\mathcal{F}_m$ $n$-Parteien-Funktionalitäten. Sei $\pi$ ein Protokoll im $\mathcal{F}_1,\ldots,\mathcal{F}_m$-Hybridmodell, wobei nie mehr als eine ideale Funktionalität gleichzeitig aufgerufen wird. Seien $\rho_1,\ldots,\rho_n$ Protokolle, sodass $\rho_i \geq \mathcal{F}_i$ $(i=1,\ldots,m)$ gilt. Dann gibt es für jeden Angreifer $\mathcal{A}$ einen Angreifer $\mathcal{A}_\pi$ im $\mathcal{F}_1,\ldots,\mathcal{F}_m$-Hybridmodell, dessen Laufzeit polynomiell in der Laufzeit von $\mathcal{A}$ ist, sodass*

$$\mathrm{EXEC}_{\pi^{\mathcal{F}_1,\ldots,\mathcal{F}_m},\mathcal{A}_\pi} \approx \mathrm{EXEC}_{\pi^{\rho_1,\ldots,\rho_m},\mathcal{A}}.$$

### B.2.4 Passiv-sichere Zwei-Parteien-Berechnung

Im Folgenden betrachten wir nur Boolesche Schaltkreise mit AND- und NOT-Gattern.

**Construction 1** (Protokoll zur passiv-sicheren Auswertung von Schaltkreisen)**.** *Sei o. B. d. A. $|x_1| = |x_2| = |y_1| = |y_2| = n$, wobei $x_i$ (bzw. $y_i$) die Eingabe (bzw. Ausgabe) von $P_i$ ist. Beide Parteien haben als gemeinsame Eingabe einen Schaltkreis $C$, dessen erste $n$ Eingabeknoten (Ausgabeknoten) der Partei $P_1$ und die nächsten $n$ Eingabeknoten (Ausgabeknoten) der Partei $P_2$ zugeordnet sind.*

**2-aus-2-Secret-Sharing der Eingabe.** *Sei $x_i^k$ die zum Eingabeknoten $n_j$ gehörende Eingabe. $P_1$ berechnet für $x_1^k$ einen Share $s_1^k \leftarrow \{0,1\}$ und setzt $s_2^k := x_1^k \oplus s_1^k$ und schickt $s_2^k$ an Partei $P_2$. (Analog für $P_2$.)*

**Auswertung des Schaltkreises.** *Sei $c_j$ das $j$-te* Gatter *von $C$ (wobei wir beginnend von der ersten Ebene der Gatter in jeder Ebene von links nach rechts zählen). Parteien $P_1$ und $P_2$ werten die Gatter des Schaltkreises* auf Shares *nacheinander wie folgt aus:*

- *Ist $c_j$ ein NOT-Gatter, hat $P_i$* einen *Eingabe-Share $s_i^j$. Die Partei $P_1$ berechnet ihren Ausgabe-Share $t_1^j$ als $1 \oplus s_i^j$, die Partei $P_2$ berechnet $t_2^j = s_2^j$. (Es gilt offensichtlich $t_1^j \oplus t_2^j = \neg(s_1^j \oplus x_2^j)$.)*
- *Ist $c_j$ ein AND-Gatter, hat $P_i$* zwei *Eingabe-Shares $s_i^{j,1}, s_i^{j,2}$. Beide Parteien $P_1$ und $P_2$ rufen die Funktionalität $\mathcal{F}$[MULT] auf, die folgende Funktion $f$ berechnet:*

$$f(n, \underbrace{(s_1^{j,1}, s_1^{j,2})}_{\text{Eingabe } P_1}, \underbrace{(s_2^{j,1}, s_2^{j,2})}_{\text{Eingabe } P_2}, z_1) \mapsto (z_1, z_2), \text{ wobei } z_2 = (s_1^{j,1} \oplus s_2^{j,1}) \wedge (s_1^{j,2} \oplus s_2^{j,2}) \oplus z_1$$

  *Die Ausgaben $z_1, z_2$ von $f$ sind also (zufällige) Shares auf $(s_1^{j,1} \oplus s_2^{j,1}) \wedge (s_1^{j,2} \oplus s_2^{j,2})$*

**Rekonstruktion der Ausgabe.** *Für jeden Ausgabeknoten $n_j$ von $C$, der das $k$-te Bit der Ausgabe $y_1$ von $P_1$ darstellt, schickt $P_2$ den Share $s_2^j$ an $P_1$. $P_1$ rekonstruiert $y_1^k = s_1^j \oplus s_2^j$. Sind alle Ausgaben rekonstruiert, gibt $P_1$ den rekonstruierten Wert $y_1$ aus. (Analog $P_2$.)*

**Theorem 2.** *Konstruktion 1 realisiert die $n$-Parteien-Funktion*

$$f_C(n, x_1, x_2) \mapsto (C(x_1, x_2)_1, C(x_1, x_2)_2)$$

*im $\mathcal{F}$[MULT]-Hybridmodell mit passiver Sicherheit.*

**Theorem 3** (informell)**.** *Sei $f$ eine $n$-Parteien-Funktion. Dann existiert ein Protokoll $\pi$, das $f$ passiv sicher berechnet.*

Im Folgenden geben wir eine Konstruktion für die Multiplikationsfunktion $f$ aus einem 1-aus-4-OT an.

**Construction 2.** *Durch Ausmultiplizieren können wir $(s_1^1 \oplus s_2^1) \wedge (s_1^2 \oplus s_2^2)$ zu $s_1^1 \wedge s_1^2 \oplus (s_2^1 \wedge s_1^2 \oplus s_1^1 \wedge s_2^2) \oplus s_2^1 \wedge s_2^2$ umformen. Die Partei $P_i$ kann offensichtlich $s_i^1 \wedge s_i^2$ selbst berechnen. Die Parteien müssen also nur noch $g(n, (s_1^1, s_1^2), (s_2^1, s_2^2), t_1) \to (t_1, t_2)$ berechnen, wobei $t_2 = (s_2^1 \wedge s_1^2) \oplus (s_1^1 \wedge s_2^2) \oplus t_1$.*

*Dafür nutzen wir ein 1-aus-4-OT wie folgt: die Partei $P_1$ agiert als OT-Sender und zieht gleichverteilt zufällig $t_1' \leftarrow \{0,1\}$. Die OT-Nachrichten $m_1, \ldots, m_4$ werden wie folgt gewählt:*

| $k$ | *Entsprechender Wert von* $(s_2^1, s_2^2)$ | $m_k$ |
|---|---|---|
| *1* | $(0,0)$ | $t_1'$ |
| *2* | $(0,1)$ | $t_1' \oplus s_1^1$ |
| *3* | $(1,0)$ | $t_1' \oplus s_1^2$ |
| *4* | $(1,1)$ | $t_1' \oplus s_1^1 \oplus s_1^2$ |

*$P_2$ wählt $k$ nach der Formel $2s_2^1 + s_2^2 + 1$ und erhält $t_2' = t_1' \oplus (s_2^1 \wedge s_1^2) \oplus (s_1^1 \wedge s_2^2)$. $P_1$ gibt $t_1 = t_1' \oplus (s_1^1 \wedge s_1^2)$, $P_2$ gibt $t_2 = t_2' \oplus s_2^1 \wedge s_2^2$ aus.*

**Theorem 4.** *Konstruktion 2 realisiert die Funktionalität $\mathcal{F}$[MULT] im $\mathcal{F}$[OT]-Hybridmodell mit passiver Sicherheit.*

## B.3 Von passiver zu aktiver Sicherheit: Der GMW-Compiler

Mit Konstruktion 1 lassen sich beliebige Zwei-Parteien-Funktionen $f$ passiv sicher auswerten. Mithilfe einer generischen Konstruktion, dem *GMW-Compiler*, ist es möglich, die Konstruktion aktiv sicher zu machen. Der Ansatz ist generisch und auf beliebige passiv sichere Protokolle anwendbar. Ebenso kann er kanonisch auf $n$ Parteien erweitert werden.

**Construction 3** (Der GMW-Compiler, informell)**.** *Sei $\pi$ ein Zwei-Parteien-Protokoll. Wir konstruieren daraus ein Protokoll $\pi'$ wie folgt:*

**Vorberechnung.**

1. *Input-Commit-Phase: Sei* COM *ein sicheres Commitment-Verfahren. $P_1$ und $P_2$ rufen zuerst die Hybridfunktionalität $\mathcal{F}_1$ auf, die folgende Funktion $f$ berechnet: $f(n, x_1, \varepsilon, r) \mapsto (d, c)$, wobei $(c, d) =$ $\mathsf{Commit}(1^n, x_1; r)$. Die Partei $P_2$ hat als Ausgabe also ein Commitment auf die Eingabe $x_1$ von $P_1$. $P_1$ hat die dazugehörige Unveil-Information $d$. Die Input-Commit-Phase wird analog mit vertauschten Rollen ein zweites Mal durchgeführt.*

2. *Münzwurf-Phase: Um ein gleichverteilt zufälliges Zufallsband zu haben, rufen $P_1$ und $P_2$ die Hybridfunktionalität $\mathcal{F}_2$ auf, die folgende Funktion $f$ berechnet: $f(n, \varepsilon, \varepsilon, (r, s)) \mapsto ((r, d), c)$, wobei $(c, d) =$ $\mathsf{Commit}(1^n, r; s)$ (*augmented coin tossing*). Die Partei $P_2$ hat als Ausgabe also ein Commitment auf einen Zufallsstring $r$. Partei $P_1$ hat als Ausgabe den Zufallsstring $r$ und die Unveil-Information $d$. Die Münzwurf-Phase wird analog mit vertauschten Rollen ein zweites Mal durchgeführt.*

**Protokollausführung.** *$P_1$ und $P_2$ führen $\pi$ aus. Statt Zufall vom eigenen Zufallsband wird der Zufall aus der Münzwurf-Phase (auf den die andere Partei ein Commitment hat) verwendet. Das Protokoll $\pi$ wird normal ausgeführt. Nach dem Senden einer Protokollnachricht wird die ideale Zero-Knowledge-Funktionalität $\mathcal{F}$[ZK] von der sendenden Partei aufgerufen. Bewiesen wird dabei (im Wesentlichen) die Korrektheit der gesendeten Protokollnachricht relativ zum Protokoll $\pi$, zum Zufall $r$, auf den die andere Partei ein Commitment hat und zur eigenen Eingabe $x_i$, auf die die andere Partei ebenfalls ein Commitment hat. Zeuge ist (im Wesentlichen) die eigene Eingabe $x_i$, die zum Commitment auf $x_i$ gehörende Unveil-Information, der Zufall $r$, sowie die zum Commitment auf den Zufall gehörende Unveil-Information.*

**Theorem 5** (Goldreich, Micali und Wigderson [75], informell)**.** *Sei $\pi$ ein Protokoll, das eine ideale Funktionalität $\mathcal{F}$ passiv sicher realisiert. Konstruktion 3 transfomiert $\pi$ in ein Protokoll $\pi'$, das $\mathcal{F}$ aktiv sicher realisiert.*

## B.4 Oblivious Transfer

Oblivious Transfer (OT) ist ein Primitiv, das einem Sender $\mathsf{S}$ erlaubt, einen (vom Empfänger kontrollierten) Teil seiner Nachrichten an einen Empfänger $\mathsf{R}$ zu senden, sodass $\mathsf{S}$ nicht lernt, welche Nachrichten $\mathsf{R}$ erhalten hat.

### B.4.1 Definition von Eins-aus-zwei-OT

Dieser Abschnitt definiert und erläutert $\binom{2}{1}$-$\mathsf{OT}$. Inspiriert von Zero-Knowledge ist unsere Definition *simulationsbasiert*, d.h. die Sicherheit wird mittels „Ununterscheidbarkeit“ einer „echten Ausführung“ und einer „simulierten Ausführung“ definiert. Ähnlich wie bei Zero-Knowledge sagen wir, der Sender bei einem OT „lernt nichts“, bzw. der Empfänger „lernt nichts“ außer $m_b$, wenn man sich den Protokollablauf auch selbst simulieren hätte können (also ohne die Kenntnis der anderen Informationen). In diesem Fall ist die eigene Sicht der echten Ausführung und die der Simulation computationally ununterscheidbar. Die simulierte Ausführung ist per Definition kein Problem, da sie per PPT-TM ohne die „geheimen“ Informationen erzeugt wurde. Die Ununterscheidbarkeit beider Ausführungen zeigt wie bei Zero-Knowledge, dass ein echter Angriff nicht mehr Informationen extrahieren kann (als der simulierte).

**Definition 16** (1-aus-2-OT)**.** *Ein* (1-aus-2-)Oblivious Transfer-Protokoll *(kurz $\binom{2}{1}$-OT) mit Nachrichtenraum $\mathcal{M} = \{\mathcal{M}_n\}_n$ ist ein interaktives Zwei-Parteien-Protokoll, d. h. ein Tupel aus zwei PPT-ITMs $\mathsf{OT} = (\mathsf{S}, \mathsf{R})$, wobei $\mathsf{S}$ als Eingabe $1^n$ und zwei Nachrichten $m_0$, $m_1$ aus $\mathcal{M}_n$ erhält, und $\mathsf{R}$ die Eingaben $1^n$ und ein Choice-Bit $b$. Die Ausgabe von $\mathsf{R}$ ist aus $\mathcal{M}_n \cup \{\bot\}$, also eine Nachricht oder ein Fehlersymbol. Wir fordern:*

- Korrektheit: $\forall n \in \mathbb{N}, m_0, m_1 \in \mathcal{M}, b \in \{0,1\}\colon \operatorname{out}_{\mathsf{R}} \langle \mathsf{S}(m_0, m_1), \mathsf{R}(b) \rangle (1^n) = m_b$.

*Für Sicherheit gegen passive Angreifer verlangen wir:*

- Empfängersicherheit (computational): *(„Choice-Hiding“) Es existiert eine PPT-TM $\mathsf{Sim}$, sodass für jede PPT-TM $\mathsf{G}$ (auch „Eingabemaschine“ genannt), die bei Eingabe $1^n$ ein 4-Tupel $(m_0, m_1, b, \mathsf{state})$ mit $m_0, m_1 \in \mathcal{M}_n$, $b \in \{0,1\}$ ausgibt, gilt:*

$$\begin{aligned} &\{(v, \mathsf{state}) \mid v \leftarrow \operatorname{view}_{\mathsf{S}} \langle \mathsf{S}(m_0, m_1), \mathsf{R}(b) \rangle (1^n); (m_0, m_1, b, \mathsf{state}) \leftarrow \mathsf{G}(1^n)\}_n \\ \overset{c}{\approx}\ &\{(v, \mathsf{state}) \mid v \leftarrow \mathsf{Sim}(1^n, m_0, m_1); (m_0, m_1, b, \mathsf{state}) \leftarrow \mathsf{G}(1^n)\}_n \end{aligned}$$

- Sendersicherheit (computational): *(„Message-Hiding“) Es existiert eine PPT-TM $\mathsf{Sim}$, sodass für jede PPT-Eingabemaschine $\mathsf{G}$ (wie oben) gilt:*

$$\begin{aligned} &\{(v, \mathsf{state}) \mid v \leftarrow \operatorname{view}_{\mathsf{R}} \langle \mathsf{S}(m_0, m_1), \mathsf{R}(b) \rangle (1^n); (m_0, m_1, b, \mathsf{state}) \leftarrow \mathsf{G}(1^n)\}_n \\ \overset{c}{\approx}\ &\{(v, \mathsf{state}) \mid v \leftarrow \mathsf{Sim}(1^n, b, m_b); (m_0, m_1, b, \mathsf{state}) \leftarrow \mathsf{G}(1^n)\}_n \end{aligned}$$

Einige weitere Anmerkungen zu dieser Definition:

- Sicherheit ist nur für *passive* Angreifer definiert und gewährleistet. Da sich solch ein passiver Angreifer an das Protokoll hält und alle Nachrichten und Werte wie im Protokoll beschrieben berechnet, ist die Sicht der ehrlichen Parteien ($\mathsf{R}$ bzw. $\mathsf{S}$) genug, um alles was gelernt wird zu beinhalten.
- Die Simulatoren hängen von keinem Angreifer ab, denn ein passiver Angreifer führt genau den Protokollablauf durch. nämlich als „die Sicht von $\mathsf{S}$ für $b = 0$ und $b = 1$ ist ununterscheidbar“. Stattdessen haben wir beide Sicherheitseigenschaften simulationsbasiert definiert.

- Analog zu den Defintionen von Zero-Knowledge, erlauben wir $\mathsf{G}$ noch ein $\mathsf{state}$ auszugeben, was dem Unterscheider helfen kann.
- Es lassen sich analoge Definitionen für statistische und perfekte Empfänger- und Sendersicherheit angeben.

## B.5 Garbled Circuits

Mit Garbled Circuits lässt sich ein passiv sicheres 2-Parteien-Protokoll zur Auswertung von Booleschen Schaltkreises konstruieren. Im Folgenden werden Garbled Circuits definiert und gezeigt, wie daraus sichere Mehrparteienberechnung entsteht.

Ein Boolescher Schaltkreis kann ausgewertet werden, indem nach und nach der Wert am Ausgang eines Gatters (Knotens) berechnet wird, sofern die Werte aller Eingänge des Gatters feststehen, wobei der Wert 0 oder 1 ist. Bei Garbled Circuits geht man dazu über, nicht mehr direkt die Werte 0 und 1 zu betrachten, sondern Labels aus $\{0,1\}^n$. Jeder Draht $w$ hat somit Labels $l_w^0$ für den Wert 0 und $l_w^1$ für den Wert 1. Hat eine Partei nun für einen Draht ein Label $l_w^b$, so kann sie nicht zuordnen, ob dies ein Label für $b = 0$ oder für $b = 1$ ist, da beide Labels zufällig gleichverteilt gezogen wurden.

Eine Auswertung eines Gatters erfolgt indem für die Labels am Eingang des Gatters neue (Ausgangs-)Labels berechnet werden. Da ein einfaches Auswerten eines Gatters durch die Labels nicht mehr möglich ist, gibt es zu jedem Gatter eine Liste von Ausgangslabels, die jeweils mit den Eingangslabels symmetrisch verschlüsselt sind.

| Eingangslabel $w_1$ | Eingangslabel $w_2$ | Ausgangslabel | Garbled Auswertungsliste |
|---|---|---|---|
| $l_{w_1}^0$ | $l_{w_2}^0$ | $l_{w_{out}}^0$ | $\mathrm{Enc}_{l_{w_1}^0, l_{w_2}^0}(l_{w_{out}}^0)$ |
| $l_{w_1}^0$ | $l_{w_2}^1$ | $l_{w_{out}}^0$ | $\mathrm{Enc}_{l_{w_1}^0, l_{w_2}^1}(l_{w_{out}}^0)$ |
| $l_{w_1}^1$ | $l_{w_2}^0$ | $l_{w_{out}}^0$ | $\mathrm{Enc}_{l_{w_1}^1, l_{w_2}^0}(l_{w_{out}}^0)$ |
| $l_{w_1}^1$ | $l_{w_2}^1$ | $l_{w_{out}}^1$ | $\mathrm{Enc}_{l_{w_1}^1, l_{w_2}^1}(l_{w_{out}}^1)$ |

Tabelle 10: Garbled Circuit Wertetabelle für ein UND-Gatter mit Eingangsdrähten $w_1$ und $w_2$ und Ausgangsdraht $w_{out}$.

Eine Auswertung erfolgt nun, indem jeder Eintrag der garbled Auswertungsliste mit den am Gatter anliegenden Eingangslabels entschlüsselt wird. Im Allgemeinen wird nur einer der Einträge korrekt entschlüsseln.[19] Die entschlüsselte Nachricht ist das Ausgangslabel des Gatters. Es ist hiermit nur möglich entweder das 0-Label oder das 1-Label zu erhalten, jedoch nicht beides. Wird die Reihenfolge der Einträge in der garbled Auswertungsliste randomisiert, so ist es beim Auswerten nicht möglich aus der Position des erhaltenen Ausgangslabels Rückschlüsse auf den tatsächlichen Wert der Ein- und Ausgänge des Gatters zu ziehen.

**Definition 17** (Garbled Circuit)**.** *Ein Garbled Circuit ist ein Boolescher Schaltkreis, wobei an den Ausgangsdrähten jedes Eingangsknotens Labels ausgegeben werden und jedes Gatter eine garbled Auswertungsliste hat, welche den Eingangslabels Ausgangslabels zuweist. Die Labels für die logische 0 und 1 werden für jeden Draht zufällig gleichverteilt aus* $\{0,1\}^n$ *gezogen.*

Die Sicherheit eines Garbled Circuit-Protokolls basiert darauf, dass keine sichtbare Verbindung zwischen den Labels und den tatsächlichen Werten, die sie repräsentieren besteht. Dies setzt jedoch voraus, dass auch die Eingabelabels für den Schaltkreis so verteilt werden, dass der Zusammenhang zwischen Label und logischem Wert nicht bekannt wird und dass für jeden Draht nur eines der beiden Labels bekannt wird.

Gegeben genau ein Label pro Eingangsdraht des Schaltkreises, dann ist es für einen PPT-Angreifer bei der Auswertung eines Garbled Circuit nur mit vernachlässigbarer Wahrscheinlichkeit möglich den zugehörigen Wert eines Labels an einem Ausgangsdraht eines Gatters zu bestimmen.

[19] Alternativ können wir beim erstellen der garbled Auswertungstabelle den Wert $0^n | l_{w_{out}}^b$ verschlüsseln und beim Entschlüsseln die Struktur der Nachricht überprüfen.

Hat eine Partei ein Label $l_w^b$ eines Drahtes $w$, so kann sie nicht entscheiden, ob dies ein Label für $b = 0$ oder für $b = 1$ ist, da beide Labels zufällig aus derselben Wahrscheinlichkeitsverteilung gezogen wurden. Hält eine Partei für jeden Eingangsdraht eines Gatters nur ein Label, so kann sie auch nur (bis auf mit vernachlässigbarer Wahrscheinlichkeit) eines der Labels des Ausgangsdrahtes bestimmen. Durch die zufällige Reihenfolge der garbled Auswertungstabelle erhält sie keine Informationen darüber, ob die Eingangslabels oder die erhaltenen Ausgangslabels zu $b = 0$ oder $b = 1$ gehören.

Hat eine Partei zu einem Draht $w$ beide Labels $l_w^0$ und $l_w^1$, so kann sie mehrere Einträge der garbled Auswertungsliste entschlüsseln und (je nach Eingabe an den anderen Drähten) mehrere Ausgangslabels lernen. Weiß sie nun zusätzlich, um welches Gatter es sich handelt (UND, ...), so kann sie außerdem aus den erhaltenen Ausgangslabels (und daraus, ob diese für beide $l_w^b$ gleich sind oder nicht) einigen Labels den logischen Wert 0 oder 1 zuordnen.

Garbled Circuits „verstecken“ Zwischenergebnisse also nur solange die Auswertende Partei zu keinem Eingangsdraht eines Gatters beide Labels lernt.

Daraus konstruiert man ein Protokoll zur Sicheren Mehrparteienberechnung, welches als zentralen Bestandteil Garbled Circuits verwendet.

**Construction 4** (Yao's Garbled Circuit Zwei-Parteien-Protokoll)**.** *Seien im folgenden die beiden Parteien* A *und* B *und die zu berechnende Funktion $f$. Seien weiterhin $x_{\mathrm{A},i}$ und $x_{\mathrm{B},i}$ das $i$-te Bit der Eingabe von* A *und* B*, sowie $y_{\mathrm{A},i}$ und $y_{\mathrm{B},i}$ das $i$-te Bit der Ausgabe von* A *und* B*. Dann ist folgendes Protokoll ein passiv sicheres Zwei-Parteien-Protokoll:*

1. A *überführt $f$ in einen Booleschen Schaltkreis und dann in einen Garbled Circuit.* A *merkt sich dabei alle Labels.*
2. A *schickt den Garbled Circuit, sowie die zu* A*'s Eingabe passenden Labels $l_{\mathrm{A},i}^{x_{\mathrm{A},i}}$ an* B*.*
3. A *und* B *führen für jedes Bit in* B*'s Eingabe einen Eins-aus-Zwei String-OT durch.* A *nimmt dabei als Sender mit den Nachrichten $l_{\mathrm{B},i}^0$ und $l_{\mathrm{B},i}^1$ teil,* B *nimmt als Empfänger mit dem Choice-Bit $x_{\mathrm{B},i}$ teil.*
4. B *hat nun zu allen Eingabeknoten Labels und wertet den Garbled Circuit aus.*
5. B *sendet die zu* A*'s Ausgabe gehörenden berechneten Labels $l_{\mathrm{A}',i}^{y_{\mathrm{A},i}}$ zu* A*.*
6. A *sendet die zu* B*'s Ausgabe gehörenden Labels $l_{\mathrm{B}',i}^0$ und $l_{\mathrm{B}',i}^1$zu* B*.*
7. *Beide Parteien haben nun sowohl beide zu ihren Ausgabe-Bits gehörenden Labels, als auch ihre berechneten Labels zu ihren Ausgaben aus dem Garbled Circuit und berechnen daraus ihre Ausgabe-Bits $y_{\mathrm{A},i}$, bzw. $y_{\mathrm{B},i}$.*

Da jede Partei ihre Ein- und Ausgabe, sowie die Zuordnung ihrer Ausgabelabels zu den logischen Werten kennt, lässt dies bedingt Rückschlüsse auf Ein- und Ausgabewerte von Gattern zu, z.B. weil der Ausgabewert eines UND-Gatters 1 ist und somit die Eingabewerte auch 1 sein müssen. Es ist möglich, dass eine Partei somit Informationen über die Eingabe der anderen Partei erhält. Dies steht jedoch nicht im Widersprich zur Definition von Sicherer Mehrparteienberechnung, da diese Informationen mit dem Schaltkreis bereits aus der eigenen Ein- und Ausgabe berechnet werden können, ohne die Eingabe der anderen Partei zu kennen. Ein Simulator könnte diese Zwischenergebnisse in der idealen Welt auch berechnen.