Secret-Sharing – Sicherheitsbetrachtungen und Tools

Veronika Pachatz

Alpen-Adria-Universität Klagenfurt
veronikapa@edu.aau.at

Zusammenfassung

Secret-Sharing-Protokolle ermöglichen die sichere Verarbeitung von vertraulichen Daten durch die Verteilung der Daten auf mehrere Instanzen. Damit sind Secret-Sharing-Verfahren neben der Verschlüsselung eine technische Maßnahme, um vertrauliche Daten während der Verarbeitung vor unbefugten Zugriffen zu schützen. Verschlüsselungsverfahren zur sicheren Datenverarbeitung können unter bestimmten Voraussetzungen durch Chosen-Instruction-Angriffe gebrochen werden. Dadurch kann die Möglichkeit zur beliebigen Datenverarbeitung zur Erlangung von unbefugten Zugriffen auf vertrauliche Daten ausgenutzt werden. In diesem Beitrag wird die Anwendbarkeit von Chosen-Instruction-Angriffen auf Secret-Sharing-Protokolle, ähnlich zur Vorgehensweise bei Verschlüsselungen, diskutiert. Neben der theoretischen Diskussion von Angriffsszenarien auf Secret-Sharing-Protokolle werden zwei Werkzeuge zur Ausführung von verteilten Berechnungen auf geheimen Werten vorgestellt: Eine Java-Bibliothek stellt Methoden und Konstrukte zur Verfügung, um verteilte Berechnungen von arithmetischen Ausdrücken durchzuführen. Ein Prototyp integriert die Java-Bibliothek und ermöglicht die Ausführung von verteilten Berechnungen zwischen beliebig vielen Instanzen innerhalb eines Netzwerks.

1 Einleitung

In einer zunehmend datengetriebenen Welt gewinnt die Geheimhaltung von vertraulichen Informationen stetig an Bedeutung. Die Verarbeitung von vertraulichen Informationen setzt technische Maßnahmen voraus, um den unbefugten Zugriff auf vertrauliche Daten effektiv zu unterbinden. Eine technische Maßnahme ist der Einsatz von Verschlüsselungen, deren Sicherheit auf der Geheimhaltung der verwendeten Schlüssel beruht. Jedoch können auf verschlüsselten Daten bisweilen nur einfache und spezifische Operationen durchgeführt werden, die unter bestimmten Voraussetzungen sogar Chosen-Instruction-Angriffe (CI-Angriffe) [RS16] ermöglichen. Durch den Einsatz von Multiparty-Computations, anstelle einer Verschlüsselung, können die vertraulichen Informationen einer Berechnung auf unterschiedliche Instanzen verteilt werden, sodass keine Instanz Zugriff auf die vollständigen Informationen hat. Die Verteilung der Daten erfolgt mit Hilfe von sogenannten Secret-Sharing-Protokollen. Secret-Sharing-Protokolle ermöglichen die Verteilung von Geheimnissen auf eine beliebige Anzahl an Teilnehmern, wobei nur eine qualifizierte Teilmenge der Teilnehmer das Geheimnis kooperativ rekonstruieren kann.

Der Fokus dieses Beitrags liegt in der Darstellung von Implementierungen und in der Sicherheitsanalyse von Protokollen zur Durchführung von verteilten Berechnungen mittels Secret-Sharing. Im Rahmen der Sicherheitsbetrachtungen wird die Anwendbarkeit von CI-Angriffen auf Secret-Sharing-Protokolle, ähnlich zur Vorgehensweise bei Verschlüsselungen [RS16], diskutiert. Dabei werden zwei konkrete Möglichkeiten eines solchen Angriffs auf Secret-Sharing-

P. Schartner · N. Pohlmann (Hrsg.) · D•A•CH Security 2018 · syssec (2018) 105-116.

Protokolle und dafür notwendige Voraussetzungen erläutert. In beiden Szenarien wird Angreifern der unbefugte Zugriff auf das Ergebnis verteilter Berechnungen ermöglicht, indem vorhandene Sicherheitsmechanismen zweckentfremdet werden. Eine Variante verwendet die „Verifizierbarkeit“ von speziellen Secret-Sharing-Protokollen, um den Angriff durchzuführen. Die zweite Variante nutzt die Verschlüsselung des Datentransfers zwischen den beteiligten Instanzen als Angriffspunkt.

Neben der theoretischen Diskussion von Angriffsszenarien auf Secret-Sharing-Protokolle werden zwei Werkzeuge zur Ausführung von verteilten Berechnungen vorgestellt. Die beschriebene Java-Bibliothek stellt alle notwendigen Methoden und Klassen zur Verfügung, um verteilte Berechnungen von arithmetischen Ausdrücken durchzuführen. Unterstützte Operationen sind Additionen und Multiplikationen. Der Prototyp integriert die Java-Bibliothek und ermöglicht die Ausführung verteilter Berechnungen zwischen beliebig vielen Instanzen innerhalb eines Netzwerks. Für die Ausführung von speziellen, mathematischen Operationen innerhalb algebraischer Strukturen wurde die Open-Source Bibliothek Sunset/FFapl [SSRG17] verwendet.

2 Grundlagen

Nachfolgend werden grundlegende Begriffe und Definitionen diskutiert, die in den Folgekapiteln verwendet werden. Ein kurzer Überblick über Secret-Sharing-Protokolle, Commitments und verifizierbare Secret-Sharing-Protokolle soll die grundlegende Funktionsweise von Secret-Sharing und Multiparty-Computation erläutern. Zusätzlich wird die homomorphe Verschlüsselung behandelt, da dieser Mechanismus für die Ausführung von CI-Angriffen genutzt wird.

2.1 Secret Sharing

Das Ziel von Secret-Sharing ist die Verteilung eines Geheimnisses unter beliebig vielen Teilnehmern mit der Einschränkung, dass nur qualifizierte Teilmengen an Teilnehmern die Rekonstruktion des Geheimnisses durchführen können. Ein Secret-Sharing-Protokoll besteht aus einer Verteilungs- und einer Rekonstruktionsphase. In der Verteilungsphase erstellt ein Dealer, welcher das Geheimnis kennt, Teile des Geheimnisses, sogenannte „shares“, und verteilt diese vertraulich unter den Teilnehmern. Die Rekonstruktionsphase wird genutzt, um das Geheimnis durch die Kombination einer Teilmenge von Teilgeheimnissen zu rekonstruieren. Teilnehmer, die in der Rekonstruktionsphase kooperieren, werden auch als Kombinierer bezeichnet. [CDN15]

Bei einem (k, n)-Threshold-Secret-Sharing-Protokoll wird das Geheimnis s in n-Teile aufgeteilt. Der Schwellwert $k \leq n$ definiert die minimale Anzahl von Teilgeheimnissen, die kombiniert werden müssen, um das Geheimnis s zu rekonstruieren. Für ein (k, n)-Threshold-Schema gilt, dass das Geheimnis s leicht berechnet werden kann, wenn beliebige k aus n Teilgeheimnissen bekannt sind. Es können jedoch keine Informationen über das Geheimnis s abgeleitet werden, wenn weniger als k Teilgeheimnisse bekannt sind. Eines der ersten (k, n)-Threshold-Secret-Sharing-Protokolle wurde von Shamir [Sha79] veröffentlicht. Das Geheimnis s kann in n-Teile aufgeteilt werden, indem man ein zufälliges Polynom mit dem Grad $k - 1$ wählt: $g(x) = a_0 + a_1 \cdot x + ... + a_{k-1} \cdot x^{k-1}$ mit $a_0 = s$ und $x = 1, 2, ..., n$. Die Koeffizienten sollten so gewählt werden, dass $a_0, a_1, ..., a_{k-1} \in \mathbb{Z}_p$ mit $p \in \mathbb{P}$ und $p \geq max(t, n + 1)$ gilt. Jeder Teilnehmer T_i erhält einen Teil des Geheimnisses $s_i = g(i)$ für $1 \leq i \leq n$. Mit k-Teilgeheimnissen ist es möglich, die Koeffizienten von $g(x)$ mittels Lagrange-Polynominterpolation zu ermitteln

und damit $s = g(0)$ zu berechnen:

$$g(x) = \sum_{i=1}^{k} g(x_i) \cdot \prod_{\substack{1 \leq j \leq k \\ j \neq i}} \frac{x - x_j}{x_i - x_j} = a_0 + a_1 \cdot x + ... + a_{k-1} \cdot x^{k-1}$$

Die Idee dieses Schemas ist, dass mindestens k Punkte auf einem Polynom mit dem Grad $k-1$ bekannt sein müssen, sodass das Polynom eindeutig bestimmt werden kann. Daher ist es möglich, $s \in \mathbb{Z}_p$ zu rekonstruieren, wenn mindestens k Teilgeheimnisse eines Polynoms mit dem Grad $k-1$ angegeben werden. Die Kenntnis von weniger als k-Teilgeheimnissen reicht nicht aus, um eine Lagrange-Polynominterpolation durchzuführen, und es können keine Informationen über s abgeleitet werden.

Secret-Sharing ist unter anderem eines der wichtigsten Werkzeuge für die sichere Ausführung von Multiparty-Computations, da es zur gemeinsamen Verteilung der Eingabewerte zwischen den Teilnehmern und zur Rekonstruktion des Endergebnisses verwendet wird [CDN15]. Das Problem der Multiparty-Computation (MPC) ist die kollektive Berechnung einer multivariaten Funktion $y = f(x_1, ..., x_n)$ durch n Teilnehmer. In einem Multiparty-Computation-Protokoll vereinbaren die Teilnehmer $T_1, ..., T_n$ eine Funktion f, die sie berechnen wollen und jeder Teilnehmer hat einen geheimen Eingabewert x_i mit $1 \leq i \leq n$ [CCD88]. In der Verteilungsphase agiert jeder Teilnehmer als Dealer und verteilt Teile seines Geheimnisses an die anderen Teilnehmer. In der Berechnungsphase wird die Berechnung der multivariaten Funktion f durchgeführt. Abschließend geben die Teilnehmer in der Rekonstruktionsphase ihren Teilwert des Funktionsergebnisses bekannt, so dass jeder Teilnehmer das Ergebnis rekonstruieren kann. [Noj12]

Ein Szenario aus der Praxis, bei dem vertrauliche Daten aus verschiedenen Quellen innerhalb einer Berechnung verwendet werden können, ist E-Voting. In einem sicheren E-Voting-System stellt jede teilnehmende Partei seine Stimmabgabe als geheimen Eingabewert zur Verfügung. In der Rekonstruktionsphase kann das korrekte Abstimmungsergebnis aus den abgegebenen Stimmen berechnet werden, während die Geheimhaltung der einzelnen Stimmen gewahrt bleibt. [Hir01]

2.2 Verifizierbares Secret-Sharing & Commitments

Bei der Ausführung von Secret-Sharing-Protokollen wird angenommen, dass sich sowohl der Dealer als auch alle Teilnehmer während der gesamten Protokollausführung exakt an das Protokoll halten. Diese Annahme ist nicht realistisch, da der Dealer oder die Teilnehmer vom Protokoll abweichen könnten, um das Secret-Sharing zu manipulieren. Beispielsweise könnte der Dealer falsche Teilgeheimnisse an die Teilnehmer senden, damit die Teilnehmer ein falsches Geheimnis rekonstruieren. Weiters könnten Teilnehmer ein verfälschtes Teilgeheimnis während der Rekonstruktionsphase weitergeben. Dies würde zur Rekonstruktion eines falschen Geheimnisses führen oder könnte die Rekonstruktion des Geheimnisses sogar unterbinden. [Ped91]

Diese Manipulationen können erkannt werden, wenn ein verifizierbares Secret-Sharing-Schema (VSS) verwendet wird. In VSS-Protokollen wird ein Verifikations-Unterprotokoll zwischen der Verteilungs- und der Rekonstruktionsphase hinzugefügt. Dieses Unterprotokoll ermöglicht es den Teilnehmern zu überprüfen, ob sie einen gültigen Teil des Geheimnisses erhalten haben, ohne Informationen über das Geheimnis selbst zu erhalten. Daher ermöglicht ein VSS-Schema,

dass alle Teilmengen von Teilnehmern dasselbe Geheimnis rekonstruieren, wenn alle Teilgeheimnisse gültig sind. Zur Überprüfung der Richtigkeit von Teilgeheimnissen werden meist Commitments eingesetzt. [Sta96]

Ein Commitment-Schema ist ein zweiphasiges, kryptografisches Protokoll zwischen einem Committer und einem Verifizierer. In der Commit-Phase definiert ein Committer eine Nachricht m, zu der er sich verpflichten will. Der Committer führt die Funktion $\mathcal{C}_m = Commit(m)$ aus und veröffentlicht $\mathcal{C}_m$. Das Commitment bindet den Committer an die Nachricht m, hält die Nachricht m jedoch geheim. In der Öffnungsphase, überprüft der Verifizierer, ob die Nachricht m und der dazugehörige Commitment-Wert $\mathcal{C}_m$ konsistent sind. Dazu kann der Verifizierer die Commit-Funktion ein zweites Mal ausführen und den resultierenden Commitment-Wert $\mathcal{C}'_m$ mit $\mathcal{C}_m$ vergleichen. Sofern der Committer die Nachricht nicht nachträglich verändert hat, müssen die zwei Commitment-Werte ident sein. [BKP11, Noj12]

Bei verifizierbaren Secret-Sharing-Protokollen erstellt ein Dealer zusätzlich zu den Teilgeheimnissen, Commitment-Werte zu jedem Teilgeheimnis und zum Geheimnis s. Dadurch können die Teilnehmer die Gültigkeit ihrer Teilgeheimnisse überprüfen, sowie die Korrektheit des rekonstruierten Geheimnisses verifizieren.

Eine für das nachfolgende Kapitel wichtige Unterscheidung ist die Differenzierung zwischen deterministischen und probabilistischen Commitment-Protokollen. Beispielsweise das Diskrete-Logarithmus-Commitment ist ein deterministisches Schema. Es gibt einen eindeutigen Wert $\mathcal{C}_m = g^m$ für jede geheime Nachricht $m \in \mathbb{Z}_{p-1}$. Im Gegensatz dazu ist das Pedersen-Commitment ein probabilistisches Commitment-Schema, da der Wert $r \in \mathbb{Z}_q$ zufällig gewählt wird und daher das Commitment $\mathcal{C}(m, r) = g^m \cdot h^r$ auch zufällig ist. Ein Verifizierer kann also keine Informationen über m erhalten, ohne r zu kennen. [Sla08]

2.3 Homomorphe Verschlüsselung

Es gibt unzählige Verschlüsselungsverfahren, um die Vertraulichkeit geheimer Informationen sicherzustellen. Herkömmliche Verschlüsselungsverfahren erlauben keine Ausführung von Operationen auf verschlüsselten Daten. Die Chiffrate müssen für jede Manipulation der zugrunde liegenden Klartexte entschlüsselt werden. Dies führt zu einem Sicherheitsrisiko, da der Klartext während der Durchführung der Operationen für einen Angreifer zugänglich sein könnte. Der notwendige Entschlüsselungsschritt bringt zudem einen zusätzlichen Rechenaufwand mit sich. Die homomorphe Verschlüsselung hingegen erlaubt die Ausführung von mathematischen Operationen auf den Chiffraten, die mathematischen Operationen auf dem Klartext entsprechen. Folglich ist es möglich, den zugrunde liegenden Klartext durch mathematische Operationen auf den zugehörigen Chiffraten zu modifizieren, ohne eine Entschlüsselung durchzuführen. Die Abhängigkeit zwischen Operationen auf den Chiffraten und Operationen auf dem Klartext besteht aufgrund der homomorphen Eigenschaften solcher Verschlüsselungsverfahren. [RAD78]

3 Sicherheitsbetrachtungen

In diesem Kapitel wird die mögliche Anwendung eines Chosen-Instruction-Angriffs (CI-Angriffs) auf Secret-Sharing-Protokolle diskutiert. Aufgrund der Verwendung homomorpher Verschlüsselung in verschiedenen Teilen von Secret-Sharing-Protokollen schien es plausibel, die homomorphen Eigenschaften zu nutzen, um unbefugten Zugriff auf geheime Informationen zu erhalten, ohne Schlüssel oder lokale Teilgeheimnisse von Teilnehmern zu kennen. Zwei

mögliche Szenarien, in denen ein CI-Angriff unter bestimmten Voraussetzungen auf Secret-Sharing-Protokolle ausgeführt werden kann, sind in den Abschnitten 3.2 und 3.3 beschrieben. Zuvor erfolgt eine allgemeine Beschreibung des CI-Angriffs.

3.1 Der Chosen-Instruction-Angriff

Der Chosen-Instruction-Angriff [RS16] wurde konzipiert, um die vertrauliche Datenverarbeitung (Secure-Function-Evaluation) anzugreifen. Homomorphe Verschlüsselungen erlauben die Ausführung von mathematischen Operationen auf verschlüsselten Daten, welche mathematischen Operationen auf den entschlüsselten Daten entsprechen. Dadurch ist es möglich verschlüsselte Daten weiterzuverarbeiten, ohne eine vorherige Entschlüsselung der Daten vorzunehmen. Ein CI-Angriff nutzt diese mathematische Eigenschaft aus und ermöglicht die Ermittlung des Klartextes durch die Ausführung von Berechnungen auf dem Chiffrat. Das Ziel des CI-Angriffs ist die Berechnung des zugrunde liegenden Klartextes m aus einem gegebenen Chiffrat c unter Verwendung der Homomorphie-Eigenschaften solcher Verschlüsselungen. Dabei es ist nicht notwendig, den genauen Aufbau des verwendeten Verschlüsselungsschemas zu kennen. Der Angreifer muss lediglich $c = E(m, k)$ kennen. Dann kann der Angreifer den Klartext m berechnen, ohne c zu entschlüsseln.

Dieser Angriff geht davon aus, dass ein Universalrechner alle seine internen Signale und den Speicher verschlüsselt, aber in der Lage ist, beliebige Operationen auf den verschlüsselten Daten auszuführen. Ein Universalrechner ist eine Maschine die einen Assembler-Befehlssatz (RISC oder CISC) versteht und beliebige Programme ausführen (Universalität) kann, die als Klartext gespeichert sind (mit Ausnahme von verschlüsselten Daten im Code wie zum Beispiel Konstanten). Zudem arbeitet ein Universalrechner nur mit Chiffraten und interne Signale werden nie entschlüsselt. Es wird weiters implizit davon ausgegangen, dass der Programmzähler als Klartext verfügbar bleibt, da es notwendig ist die nächste Anweisung abzurufen nachdem die vorhergehende Anweisung abgeschlossen wurde.

Für einen CI-Angriff ist es notwendig, dass die Addition und Subtraktion von Chiffraten, Division von Chiffraten, bitweise, logische Operationen sowie bedingte Sprünge ausgeführt werden können. Somit kann ein CI-Angriff mit einem einfachen Assemblerprogramm unter Ausnutzung der Standardbefehle eines Assemblers durchgeführt werden.

Der CI-Angriff besteht aus folgenden Schritten (siehe auch Abbildung 1):

1. Zuerst muss geprüft werden, dass $m \neq 0$ gilt, um eine Division-durch-Null in den Folgeschritten zu vermeiden. Daher berechnet der Angreifer $c_0 = c - c$. Aufgrund der Homomorphie-Eigenschaften von c gilt $c_0 = E(0, k)$. Da der Angreifer nun $E(0, k)$ kennt, ist es möglich zu entscheiden, ob $m = 0$ zutrifft. Wenn $c_0 = c$ dann gilt, dass $c = E(0, k)$ ist und $m = 0$ ermittelt wurde. In diesem Fall war der CI-Angriff bereits erfolgreich. Ansonsten ist es offensichtlich, dass $m \neq 0$ gilt und der Angriff mit Schritt 2 fortgesetzt wird.
2. Nach der Sicherstellung von $m \neq 0$ ist es möglich, c durch sich selbst zu teilen, ohne in eine Division-durch-Null-Ausnahme zu laufen. Der Angreifer berechnet $c_1 = \frac{c}{c} = E(1, k)$.
3. Nun kennt der Angreifer $c_0 = E(0, k)$ und $c_1 = E(1, k)$. Der Klartext m kann berechnet werden, indem $c_k = E(1, k), c_k = E(2, k), ..., c_k = E(n, k)$ durch bis zu k-Additionen und Vergleiche von c und c_k ermittelt werden. Der Angreifer beginnt mit der Berechnung von $c_k = c_0 + c_1 = c_1 = E(1, k)$. Dann prüft der Angreifer, ob $E(1, k) = c$ gilt. Wenn

diese Gleichung zutrifft, muss der zugrundeliegende Klartext $m = 1$ sein. Ansonsten berechnet der Angreifer $c_k = c_k + c_1 = c_1 + c_1 = E(2, k)$ und vergleicht erneut c_k mit c. Falls diese Gleichung zutrifft, muss der zugrundeliegende Klartext $m = 2$ sein. Dieser Additions- und Vergleichsschritt wird solange durchgeführt, bis eine Übereinstimmung zwischen dem Additionsergebnis c_k und c gefunden wird.

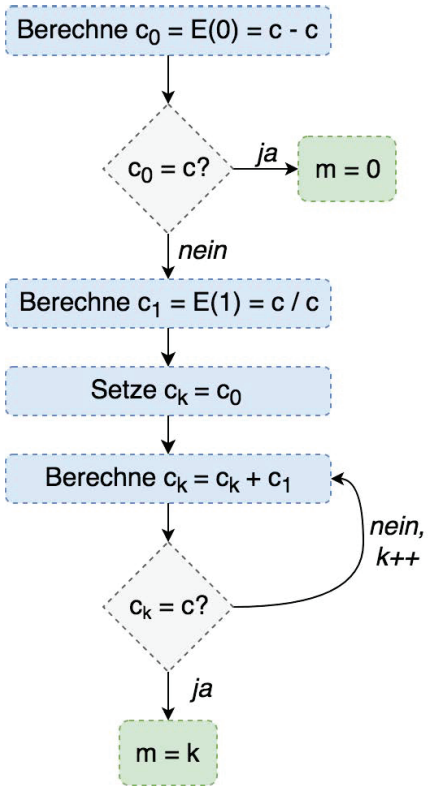

Abb. 1: Chosen-Instruction-Angriff auf ein gegebenes Chiffrat $c = E(m)$

Zu beachten ist, dass Schritt 1 die Verwendung von bedingten Sprüngen erfordert, welche in einem Universalrechner allgemein verfügbar sind. Handelt es sich bei der Implementierung der homomorphen Verschlüsselung jedoch um einen fixen Schaltkreis, dann wäre die Ausführung dieses Schrittes nicht möglich und der gesamte Angriff könnte nicht durchgeführt werden. Weiters ist es notwendig, dass der Programmzähler beobachtet werden kann, um den entsprechenden Klartext m zu ermitteln.

Aufgrund der Einfachheit dieses Angriffs kann der CI-Angriff auf jedem Universalrechner durchgeführt werden und bislang sind keine Gegenmaßnahmen bekannt. Dennoch erscheint die Konstruktion von Universalrechnern basierend auf homomorpher Verschlüsselung aus heutiger Sicht unwahrscheinlich.

3.2 CI-Angriffe auf verifizierbare Secret-Sharing-Protokolle

Die Grundidee dieser CI-Angriff-Variante ist, dass ein Angreifer mit den entsprechenden, übermittelten Commitment-Werten $\mathcal{C}_s = Commit(s)$, unbefugten Zugriff auf ein Geheimnis s erhalten kann. Angenommen, ein Angreifer kann die Teilnehmer erfolgreich auffordern willkürliche Operationen auf den Teilgeheimnissen auszuführen, dann ist es möglich einen CI-

Angriff durchzuführen, indem der Angreifer die zum Geheimnis dazugehörigen Commitment-Werte von den Teilnehmern erhält. Die Commitment-Werte können dann verwendet werden, um das Geheimnis s hinter einem Commitment-Wert $\mathcal{C}_s$ zu bestimmen, indem der CI-Angriff ähnlich wie in Abschnitt 3.1 beschrieben ausgeführt wird. Die wesentliche Erkenntnis dabei ist, dass ein Commitment für den CI-Angriff gleich gut wie eine Verschlüsselung, in der ursprünglichen Beschreibung des CI-Angriffs, fungiert. Dies liegt daran, dass für einen CI-Angriff Operationen innerhalb der Verschlüsselung, beziehungsweise alternativ innerhalb der Commitments, welche für die Verifikationen während der Protokollausführungen benötigt sein können, stattfinden müssen. Operationen wirken sich aufgrund der Verifizierbarkeits-Eigenschaft solcher Secret-Sharing-Protokolle auch auf die Commitment-Werte aus. Dadurch ist diese Voraussetzung für einen CI-Angriff erfüllt.

Zwei weitere Voraussetzungen für einen CI-Angriff sind die Möglichkeit das Programmverhalten zu beobachten und Werte miteinander zu vergleichen. Verifizierbare Secret-Sharing-Protokolle erfüllen auch diese beiden Voraussetzungen: Wenn ein Angreifer in der Lage ist, die Ausführung willkürlicher Operationen auszulösen, kann die Bestimmung der verschiedenen Commitment-Werte, die zur Durchführung des CI-Angriffs erforderlich sind, z.B. $\mathcal{C}_0 = \mathcal{C}_s - \mathcal{C}_s$, an die Teilnehmer ausgelagert werden. Eine spezielle Operation, die für den CI-Angriff notwendig ist, ist der Vergleich zweier Commitment-Werte, um zu entscheiden, ob der entsprechende Wert s zu einem gegebenen Commitment-Wert $\mathcal{C}_s$ gefunden wurde. Für die Vergleichsoperation ist es wichtig zu erwähnen, dass zwischen deterministischen und probabilistischen Commitment-Protokollen unterschieden werden muss. Wenn ein deterministisches Commitment-Protokoll, wie das Diskrete-Logarithmus-Commitment (siehe Abschnitt 2.2), verwendet wird, ist ein direkter Vergleich von zwei Commitment-Werten $\mathcal{C}_{s_i}$ und $\mathcal{C}_{s_j}$ möglich, da die Gleichung $g^{s_i} = g^{s_j}$ nur gilt, wenn die Gleichung $s_i = s_j$ auch gilt. Falls ein probabilistisches Commitment-Protokoll, wie das Pedersen-Commitment (siehe Abschnitt 2.2), verwendet wird, können zwei Commitment-Werte $\mathcal{C}_{s_i}$ und $\mathcal{C}_{s_j}$ gleich sein, obwohl $s_i \neq s_j$ gilt oder die Commitment-Werte können unterschiedlich sein, obwohl $s_i = s_j$ gilt. Ein direkter Vergleich der probabilistischen Commitment-Werte ist daher nicht möglich. Ein Angreifer müsste ein Unterprogramm ausführen lassen, damit er das Ergebnis des Vergleichs am Verhalten der Teilnehmer erkennen kann. Zum Beispiel könnte ein Angreifer die Teilnehmer veranlassen, ein Programm ähnlich dem Algorithmus 1 auszuführen; dieser Algorithmus stellt eine Form eines Seitenkanalangriffs dar. Wenn die Gleichung $s_i = s_j$ gilt, beginnen die Teilnehmer miteinander zu kommunizieren, um die Multiplikation durchzuführen. Da jede Multiplikation eine Interaktion zwischen den Teilnehmern erfordert. Kommunizieren die Teilnehmer nicht miteinander, dann gilt $s_i \neq s_j$. Da jede Addition durch eine lokale Addition der Teilgeheimnisse, ohne Interaktion mit den anderen Teilnehmern, durchgeführt werden kann. Die Beobachtung eines der beiden Verhaltensweisen durch Beobachtung der Kommunikationsmuster zwischen den Teilnehmern würde es einem Angreifer ermöglichen, das Ergebnis des Vergleichs zwischen zwei Commitment-Werten zu bestimmen.

Algorithmus 1 Feststellung der Gleichheit von probabilistischen Commitment-Werten

if $s_i = s_j$ **then**
  $r = s_i \cdot s_j$ {Massive Kommunikation (aufgrund des Multiplikationsprotokolls)}
else
  $r = s_i + s_j$ {Keine Kommunikation (da die gesamte Berechnung lokal erfolgt)}
end if

3.3 CI-Angriffe durch homomorphe Verschlüsselung

Wird eine homomorphe Verschlüsselung verwendet, um eine sichere Kommunikation zwischen dem Dealer und den Teilnehmern herzustellen, kann ein CI-Angriff durchgeführt werden. Sei $E(s_i)$ die homomorphe Verschlüsselung eines Teilgeheimnisses s_i. Dann könnte ein Angreifer alle Teilgeheimnisse s_i bestimmen, indem er einen CI-Angriff auf jedes verschlüsselte Teilgeheimnis $E(s_i)$ ausführt, das an einen Teilnehmer übertragen wird.

Angenommen, ein Dealer teilt mehrere Geheimnisse $s_1, ..., s_n$ und die Teilnehmer berechnen einen Ausdruck bestehend aus mehreren Operationen, z.B. $s = (s_1 + s_2 + ... + s_{n-1}) \cdot s_n$. Die Geheimnisse $s_1, ..., s_n$ werden vom Dealer geteilt und das Ergebnis s sollte von den Teilnehmern gemeinsam berechnet werden. Sei s_i das Teilgeheimnis des Teilnehmers T_i nach der Ausführung aller Operationen. Dann kann das Geheimnis s unbefugt ermittelt werden, indem ein CI-Angriff auf den gesammelten, homomorph-verschlüsselten, finalen Teilgeheimnissen $E(s_i)$ angewendet wird. Diese verschlüsselten Teilgeheimnisse werden vor der Rekonstruktion zwischen den Kombinierern ausgetauscht und beinhalten bereits alle notwendigen, lokalen Berechnungen. Die Teilgeheimnisse s_i können berechnet werden, indem ein CI-Angriff, ähnlich wie in Abschnitt 3.1 beschrieben, ausgeführt wird. Wenn ein Angreifer den Modulo p und den entsprechenden x-Wert zu jedem Teilgeheimnis s_i kennt, kann er die s_i-Werte interpolieren und das Geheimnis s berechnen. Die Interpolation könnte äquivalent zu einer Interpolation durchgeführt werden, die von einem Kombinierer in der Rekonstruktionsphase durchgeführt wird (siehe Abschnitt 2.1). Wenn das verwendete Protokoll verifizierbar ist und ein Angreifer Zugriff auf die Commitment-Werte hat, kann zusätzlich die Korrektheit der ermittelten Teilgeheimnisse s_i und die Korrektheit des rekonstruierten Geheimnisses s anhand dieser Commitment-Werte sichergestellt werden.

4 Tools zur Ausführung verteilter Berechnungen

Im Rahmen dieses Beitrags werden zudem zwei Werkzeuge zur Ausführung verteilter Berechnungen präsentiert. Beide Werkzeuge sind Java-Programme, welche die verteilte Berechnung von arithmetischen Ausdrücken bestehend aus beliebig vielen Additionen und Multiplikationen erlauben.

4.1 Bibliothek

Die Java-Bibliothek stellt alle notwendigen Konstrukte zur Verfügung, damit $2, ..., n$ Teilnehmer Geheimnisse teilen können und Additionen sowie Multiplikationen auf den verteilten Daten ausführen können. Die Bibliothek verwendet Teile der Open-Source-Software Sunset / FFapl [SSRG17], um die notwendigen, mathematische Berechnungen innerhalb der entsprechenden, algebraischen Strukturen durchzuführen.

Die Bibliothek basiert auf Sitzungen, innerhalb derer ein einziges Geheimnis geteilt werden kann oder kombinierte Ausdrücke mehrerer Geheimnisse berechnet werden können. Zwei Hauptobjekte dieser Bibliothek sind `Dealer` und `Participant`. Ein `Participant` (Teilnehmer) kann als Spieler oder Kombinierer an einer Sitzung teilnehmen. Ein Dealer kann eine neue Sitzung beginnen, geheime Eingabewerte und ein Multiparty-Computation-Programm definieren. Sobald ein Teilnehmer einer Sitzung beitritt, kann er entscheiden, ob er als Spieler oder als Kombinierer agiert, wobei mindestens zwei Kombinierer an einer Sitzung teilnehmen müssen. Teilnehmer, die als Spieler agieren, können ihre lokalen Teilergebnisse berechnen und

diese an Kombinierer weitergeben. Kombinierer sind Spieler, welche zusätzlich an der Rekonstruktionsphase teilnehmen. Kombinierer erhalten die Teilergebnisse anderer Teilnehmer und können das berechnete Geheimnis ermitteln.

Mittels der Eigenschaft `commitmentType` ist es dem Dealer möglich, das verwendete Commitment-Protokoll für eine Sitzung auszuwählen. Die Bibliothek unterstützt die beiden Commitment-Protokolle SHA-256 und das Pedersen-Commitment [Ped91]. Wobei für praktische Anwendung der Einsatz des Pedersen-Commitments empfohlen ist. Andere Objekte, die für den Bibliotheksbenutzer sichtbar sind, sind die Klassen `MultipartyOperationObject` und `MultipartyOperation`. Ein Dealer kann mithilfe des `MultipartyOperation`-Objektes ein Multiparty-Computation-Programm definieren. Dadurch ist es möglich, zu berechnende, kombinierte Ausdrücke zu definieren. Der Dealer berechnet zu Beginn der Sitzung alle Teilgeheimnisse und Commitments für das gesamte Multiparty-Computation-Programm. Danach wird das Multiparty-Computation-Programm Operation für Operation von den Teilnehmern abgearbeitet, indem pro Operation Teilgeheimnisse, Commitment-Werte und auszuführenden Berechnungen an die Teilnehmer gesendet werden. Die Bibliothek unterstützt drei verschiedene Operationstypen: Um die Rekonstruktion eines einzelnen Geheimnisses durchzuführen, ist der Operationstyp `None` anzugeben. Wird der Operationstyp `Sum` angegeben, so wird die Summe der zwei definierten Eingabewerte berechnet. Wird der Operationstyp `Product` angegeben, so wird das Produkt der zwei definierten Eingabewerte berechnet.

Die Verteilung eines Geheimnisses und die Berechnung von Summen basiert auf dem verifizierbaren Secret-Sharing-Protokoll von Gennaro et al. [GRR98]. Multiplikationen werden basierend auf dem vereinfachten Multiplikationsprotokoll von Gennaro et al. und Lory [GRR98, LW11] durchgeführt. Die Berechnung von Multiplikationen ist aufgrund der Protokollwahl nicht verifizierbar. Aus diesem Grund ist eine Sitzung nicht mehr verifizierbar, sobald innerhalb einer Sitzung eine Multiplikation berechnet wird.

In Programmcode 1 wird exemplarisch dargestellt, wie die Bibliothek genutzt werden kann, um die verteilte Berechnung von $(a + b) \cdot c$ mit $a = 5$, $b = 2$ und $c = 3$ zu definieren. Als `commitmentType` wird hier das Pedersen-Commitment eingesetzt. Mit dem Methodenaufruf `defineSecrets` kann der Dealer eine neue Sitzung anlegen. Damit wird die Berechnung der Protokollkonstanten, der Teilgeheimnisse sowie die Ermittlung der dazugehörigen Commitment-Werte gestartet.

```
Dealer dealer = new Dealer();
HashMap<String, BigInteger> eingabeWerte = new HashMap<>();
eingabeWerte("a",BigInteger.valueOf(5));
eingabeWerte("b",BigInteger.valueOf(2));
eingabeWerte("c",BigInteger.valueOf(3));

ArrayList<MultipartyOperation> mpcProgramm = new ArrayList<>();
mpcProgramm(new MultipartyOperation("r1","a","b",Operation.Sum));
mpcProgramm(new MultipartyOperation("r2","r1","c",Operation.Product));

CommitmentType commitmentTyp = CommitmentType.Pedersen;
dealer.defineSecrets(eingabeWerte, mpcProgramm, commitmentTyp);
```

Programmcode 1: Definition von Eingabewerten und einem Multiparty-Computation-Programm

4.2 Prototyp

Zusätzlich zur Bibliothek wurde ein Java-Prototyp für die verteilte Berechnung von Ausdrücken zwischen mehreren Instanzen implementiert. Der Prototyp ist eine Java-8-Anwendung mit ei-

ner Befehlsschnittstelle, welche die im vorherigen Abschnitt beschriebene Bibliothek integriert. Wenn nur die Bibliothek verwendet wird, muss der Ablauf der Methodenaufrufe abhängig vom zu berechnenden Ausdruck und den Teilnehmern manuell adaptiert werden. Wird beispielsweise ein zusätzlicher Teilnehmer benötigt oder ändert sich der arithmetische Ausdruck, dann müssen Änderungen im Ablauf der Methodenaufrufe vorgenommen werden. Im Gegensatz dazu erlaubt der Prototyp einer beliebigen Anzahl von Teilnehmern an einer Sitzung teilzunehmen und beliebige Ausdrücke zu berechnen, ohne Anpassungen im Code vorzunehmen. Jede Instanz der Prototyp-Anwendung ist ein einzelner Client im Netzwerk und jeder Client kann als Dealer, Spieler, Kombinierer oder Angreifer agieren. Die Clients können einerseits direkt mit einem einzelnen Client kommunizieren und andererseits Broadcast-Nachrichten an eine Gruppe von Clients senden. Für die Kommunikation wird die Open-Source-Bibliothek `JGroups` [Ban18] verwendet. `JGroups` ist ein Toolkit für zuverlässige Netzwerkkommunikation, bei dem Cluster erstellt werden können und Knoten eines Clusters Nachrichten an jeden anderen Knoten im Cluster senden können.

Bei Programmstart und am Ende einer Sitzung kann ein Benutzer entscheiden, ob er als Dealer, Teilnehmer oder Angreifer agieren möchte. Ein Angreifer empfängt alle Broadcast-Nachrichten die zwischen dem Dealer und den Teilnehmern, sowie zwischen den Teilnehmern untereinander gesendet werden. Dadurch können Informationen über die Sitzung gewonnen werden, welche für die Ausführung von Angriffen genutzt werden können.

Der Prototyp kann verwendet werden, um die verteilte Berechnung eines gegebenen Ausdrucks auszulösen. Wenn ein Dealer die Ausführung einer verteilten Berechnung starten möchte, muss eine Menge geheimer Eingabewerte und eine Menge von Berechnungsoperationen definiert werden. Das auszuführende Multiparty-Computation-Programm kann aus mehreren Zeilen bestehen, wobei jede Zeile eine auszuführende Operation definiert. Die Definition eines Multiparty-Computation-Programms muss in einer Datei mit der Endung `.mpc` gespeichert sein und folgender Syntax entsprechen:

```
Zeichen := "A" ... "Z" | "a" ... "z" .
Zahl := "0" ... "9" .
Bezeichnung := Zeichen | Zahl { Zeichen | Zahl } .
Operator := "+" | "*" | " " .
KeineOperation := Bezeichnung "=" Bezeichnung ";" .
Summe := Bezeichnung "=" Bezeichnung "+" Bezeichnung ";" .
Produkt := Bezeichnung "=" Bezeichnung "*" Bezeichnung ";" .
MpcOperation = KeineOperation | Summe | Produkt .
```

Programmcode 2: Syntax für einen Multiparty-Computation-Ausdruck

Die für die Berechnung verwendeten vertraulichen Eingabewerte müssen in einer Datei mit der Endung `.secrets` gespeichert sein. Die Datei kann mehrere Eingabewerte definieren, wobei jede Zeile genau einer Eingabedefinition mit folgender Syntax entsprechen muss:

```
Zeichen := "A" ... "Z" | "a" ... "z" .
Zahl := "0" ... "9" .
Bezeichnung := Zeichen | Zahl { Zeichen | Zahl } .
Wert := Zahl { Zahl } .
Definition := Bezeichnung "=" Wert ";" .
```

Programmcode 3: Syntax für die Definition von geheimen Eingabewerten

Beispielsweise erlaubt der Prototyp die verteilte Berechnung des Ausdrucks $((a \cdot b) + (c + d)) \cdot d$ mittels folgendem Multiparty-Computation-Programm:

```
r1 = a * b;
r2 = c + d;
r3 = r1 + r2;
result = r3 * d;
```

Programmcode 4: Beispiel-Definition eines Multiparty-Computation-Programms

Dazu müssen alle verwendeten Eingabewerte in einer `.secrets` Datei definiert werden. Eine mögliche Definition der geheimen Eingabewerte kann beispielsweise wie folgt aussehen:

```
a = 2;
b = 7;
c = 5;
d = 13;
```

Programmcode 5: Beispiel-Definition von Eingabewerten für eine verteilte Berechnung

5 Resümee und Ausblick

Secret-Sharing-Protokolle ermöglichen die Verteilung von vertraulichen Informationen auf mehrere Instanzen, wobei nur eine qualifizierte Teilmenge der Instanzen die vertraulichen Informationen rekonstruieren kann. Secret-Sharing ist unter anderem ein wichtiges Werkzeug für die sichere Ausführung von Multiparty-Computation. Dabei werden vertraulichen Informationen für eine Berechnung auf unterschiedliche Instanzen verteilt, damit keine Instanz Zugriff auf die vollständigen Informationen erhält. In der Praxis können solche Protokolle beispielsweise für die sichere Durchführung von E-Voting eingesetzt werden.

In diesem Beitrag wurde die Anwendbarkeit von Chosen-Instruction-Angriffen [RS16] auf Secret-Sharing-Protokolle, ähnlich zur Vorgehensweise bei homomorphen Verschlüsselungen, diskutiert. Dabei wurden zwei, konkrete Angriffsszenarien auf Secret-Sharing-Protokolle erläutert. Beide Szenarien erlauben Angreifern unbefugten Zugriff auf das Ergebnis einer Multiparty-Computation zu erhalten, indem vorhandene Sicherheitsmechanismen zweckentfremdet werden. Außerdem wurden zwei Werkzeuge zur Ausführung von verteilten Berechnungen vorgestellt: Die dargestellte Java-Bibliothek stellt Methoden und Konstrukte zur Verfügung, um die verteilte Berechnung von arithmetischen Ausdrücken durchzuführen. Der Prototyp integriert die Java-Bibliothek und ermöglicht die Ausführung von verteilter Datenverarbeitung zwischen beliebig vielen Instanzen innerhalb eines Netzwerks. Sowohl die Bibliothek als auch der Prototyp erlauben einer beliebigen Anzahl an Teilnehmern die verteilte Berechnung von arithmetischen Ausdrücken bestehend aus beliebig vielen Additionen und Multiplikationen.

Die praktische Implementierung besitzt einige, offene Erweiterungsmöglichkeiten: Die Java-Bibliothek unterstützt aktuell nur das verifizierbare Addieren von Werten, jedoch nicht das verifizierbare Multiplizieren von Werten. Sobald das zu berechnende Multiparty-Computation-Programm eine Multiplikation beinhaltet, ist die weitere Berechnung nicht mehr verifizierbar. Durch den Austausch des vereinfachten Multiplikationsprotokolls von Gennaro et al. und Lory [GRR98, LW11] mit einem verifizierbaren Multiplikationsprotokoll, z.B. durch das Fast-Track-Multiplikations-Protokoll von Gennaro et al. [GRR98] könnten alle Berechnungen verifizierbar gemacht werden. Eine weitere Verbesserung des Prototyps für den praktischen Einsatz wäre die Erweiterung der Netzwerkkommunikation, um einen Ver- und Entschlüsselungsalgorithmus, da derzeit alle Broadcast-Nachrichten und Punkt-zu-Punkt-Nachrichten zu Analysezwecken im Klartext übertragen werden. Zudem stellt der Prototyp zwar die Grundlagen bereit um alle für

einen CI-Angriff erforderlichen Operationen auszuführen, diese sind jedoch derzeit noch nicht alle implementiert. Etwa wäre es möglich mit dem Prototyp eine Division auszuführen, jedoch ist dies aktuell noch ein sehr komplexer Vorgang, der nicht direkt vom Prototyp angeboten wird.

Literatur

[Ban18] B. Ban: JGroups – The JGroups Project, 2018. http://www.jgroups.org, Letzter Zugriff 21.04.2018.

[BKP11] M. Backes, A. Kate, A. Patra: Computational verifiable secret sharing revisited. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 590–609. Springer, 2011.

[CCD88] D. Chaum, C. Crépeau, I. Damgård: Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19. ACM, 1988.

[CDN15] R. Cramer, I.B. Damgård, J.B. Nielsen: *Secure Multiparty Computation: An information-theoretic Approach*. Cambridge University Press, 2015.

[GRR98] R. Gennaro, M.O. Rabin, T. Rabin: Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 101–111. ACM, 1998.

[Hir01] M. Hirt: *Multi-Party Computation: Efficient Protocols, General Adversaries, and Voting*. PhD thesis, ETH Zürich, 2001.

[LW11] P. Lory, J. Wenzl: A note on secure multiparty multiplication. 2011.

[Noj12] M. Nojoumian: *Novel secret sharing and commitment schemes for cryptographic applications*. University of Waterloo, 2012.

[Ped91] T.P. Pedersen: Non-interactive and information-theoretic secure verifiable secret sharing. In *Crypto*, volume 91, pages 129–140. Springer, 1991.

[RAD78] R.L. Rivest, L. Adleman, M.L. Dertouzos: On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[RS16] S. Rass, P. Schartner: On the Security of a Universal Cryptocomputer: the Chosen Instruction Attack. *IEEE Access*, 4:7874–7882, 2016.

[Sha79] A. Shamir: How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Sla08] D. Slamanig: *Efficient and Practical Protocols for Privacy in Cloud Computing*. PhD thesis, Alpen-Adria-Universität Klagenfurt, Klagenfurt, November 2008.

[SSRG17] Alpen-Adria-Universität Klagenfurt – Forschungsgruppe Systemsicherheit: Sunset/ffapl. Website, 2017. https://github.com/stefan-rass/sunset-ffapl, Letzter Zugriff 21.04.2018.

[Sta96] M. Stadler: Publicly verifiable secret sharing. In *Eurocrypt*, volume 96, pages 190–199. Springer, 1996.