Unser erstes Dokument

Latex Tutorial

08. Februar 2013

Inhaltsverzeichnis

1 Einleitung

Secure Multiparty Computation ein großes Forschungsfeld in der Kryptographie, in dem zwar schon in den 1980er Jahren die Forschung begann, das aber seit den 2000er Jahren mehr Aufmerksamkeit bekommt. [1] In diesem Forschungsfeld werden Methoden erforscht, mit denen gemeinsam Funktionen von Eingabedaten berechnet werden können, ohne dass dabei die anderen teilnehmenden Parteien die Eingabedaten erhalten.
Nachdem die Forschungsergebnisse in den 80ern eher theoretisch waren, sind die neueren Forschungen eher praktisch. Denn durch mehrere Effekte wurden die Berechnungen erst in breiteren Anwendungsbereichen sinnvoll nutzbar. Einerseits wurden die berechnenden Computer stärker, beispielsweise hat sich die CPU Geschwindigkeit in PCs ungefähr verdoppelt. Das allein kann aber noch nicht die mehr als 60.000 fache Beschleunigung der Berechnungsgeschwindigkeit erklären. [1] GRAPH EINFÜGEN.
Der größte Teil des Tempogewinns liegt also an neuen oder verbesserten Protokollen, die entwickelt wurden. Eine Funktion, die in vielen Bereichen interessant sein kann, ist die Schnittmenge der Eingabemengen der teilnehmenden Parteien. 2021 wurde [2] veröffentlicht. In diesem Paper wurde ein neues Protokoll vorgestellt, das genau diese Funktion erfüllt. Durch eine Implementierung des Protokolls, kann nun festgestellt werden, ob dieses Protokoll funktioniert und effizient ist.

1.1 Anwendungsbeispiel

Der Browser Tor, den Menschen benutzen können, um in das sogenannte Darknet zu gelangen legt einen sehr großen Wert auf die Sicherheit der Nutzer. Das macht die Analyse bestimmter Statistiken sehr schwierig. Deshalb gibt es auch vom Anbieter selbst keine genauen Angaben über beispielsweise die Nutzerzahl, sondern nur Schätzungen. Um genauere Schätzungen über die Nutzerzahl zu ermöglichen, könnte man nun Daten von mehreren "Knoten"des Tor Browsers kombinieren. In diesem Fall möchte man gerne herausfinden, wie viele Überschneidungen es in den Daten der "Knoten"gibt, um Mehrfachzählungen zu vermeiden. Da viele Nutzer des Tor Browsers aber einen sehr großen Wert auf Sicherheit legen, möchte man natürlich tunlichst vermeiden, Daten von mehreren "Knoten"miteinander zu kombinieren, weil so möglicherweise Informationen gewonnen werden können, die geheim sein sollten.
Um zu garantieren, dass die Daten beispielsweise nur zur Ermittlung der Größe der Schnittmenge genutzt werden, können spezielle Protokolle genutzt werden, die die Daten nur verschlüsselt benutzen und nur ganz bestimmte Analysen auf den verschlüsselten Daten erlauben. So können Statistiken über den Tor Browser erstellt werden, ohne dass die Gefahr besteht, dass Informationen der Nutzer zugänglich werden.

1.2 Zielsetzung

Das Ziel der Arbeit ist es, die Neuerungen in dem Paper [2] zu implementieren. Einerseits, um zu demonstrieren, dass die vorgestellten Protokolle und Funktionalitäten sicher und effizient sind. Andererseits, um die neuen Funktionalitäten und ?Verschnel-

lerungen? allen zugänglich zu machen.
Um die Funktionalitäten wie im Paper [2] beschrieben zu implementieren, werden auch Implementierungen von anderen Veröffentlichungen benötigt ([**?**]). Da zu diesem Zeitpunkt noch keine derartigen Implementierungen veröffentlicht sind, kann auch nicht auf diese Zurückgegriffen werden. Um trotzdem zu demonstrieren, dass das im [2] vorgestellte Protokoll funktioniert und effizient ist, habe ich die Unterprotokolle, die auf solche externen Paper zurückgreifen auf unsichere Weise implementiert. Dadurch funktioniert das Protokoll, und die unsicheren Unterprotokolle können ohne viel Zeitaufwand durch sichere Implementierungen ersetzt werden, wenn die Funktionalitäten der anderen Veröffentlichungen implementiert sind.

2 Grundlagen

2.1 Homomorphe Verschlüsselungen

Homomorphe Verschlüsselung ist eine Form der Verschlüsselung, die bestimmte Berechnungen auf verschlüsselten Daten erlauben, und dann ein verschlüsseltes Ergebnis zurückgibt. Wenn dieses dann wieder entschlüsselt wird, ist das Ergebnis das gleiche, wie wenn diese Arbeitsschritte auf dem unverschlüsselten Anfangswert ausgeführt worden wären. [3]
In dieser Arbeit wird im speziellen additiv homomorphe Verschlüsselung benutzt. Diese erlaubt das addieren von zwei verschlüsselten Zahlen und das multiplizieren einer verschlüsselten mit einer unverschlüsselten Zahl.
Diese Funktionalitäten sind für das Protokoll wichtig, denn in jedem Unterprotokoll wird mit verschlüsselten Daten gerechnet. Und nur so können wir trotz einer sicheren Verschlüsselung mit den Daten rechnen, ohne den Inhalt der Verschlüsselung zu kennen.

3 Multiparty Cardinality Testing

3.1 Neues am Paper

Diese Arbeit basiert auf dem Protokoll, das in [2] vorgestellt wurde. Es gab schon vorher einige Veröffentlichungen zum Thema Private Set Intersections, auf denen das Paper aufbauen kann. Zuerst wurden Protokolle für zwei Parteien entworfen, dann auch für mehrere Parteien. Das Paper [4] ist die letzte Veröffentlichung, auf der das Protokoll aufbaut.
Die große Neuerung in der Veröffentlichung [4] ist, dass die Kommunikations-Komplexität nun vor allem von O(t) und nur noch logarithmisch von O(n) abhängt. [4]
Das von Gosh vorgeschlagene Cardinality Testing ist jedoch noch nicht für mehrere Parteien optimiert. Daher ist das neue Protokoll zur Bestimmung der Schnittmengen-Größe die wichtigste Neuerung des Papers [2]

3.2 Abwandlungen zum Paper

Das Protokoll wurde offensichtlich entworfen um auch mit mehreren teilnehmenden Parteien effizient zu sein. Das Paper [2] nutzt also in vielen Unterprotokollen Broadcast-Funktionen.

BEISPIEL: secRank, auch OMM, auch MPCT

Ich habe versucht, nah an die Spezifizierungen des Papers ran zu kommen, doch um die Implementierung des Protokolls zu vereinfachen und um das Testen der Implementierung zu erleichtern, habe ich das System etwas abgewandelt. In meiner Implementierung gibt es einen Koordinator. Dieser Koordinator erleichtert das testen enorm, denn durch ihn kann sicher gestellt werden, dass alle "verschicktenÏnformationen immer zum richtigen Zeitpunkt am richtigen Ziel ankommen.
Diese Änderung wird die Sicherheit des Protokolls nicht schwächen, weil alle Informationen, per Broadcast verschickt werden immer verschlüsselt sind. Auch wenn Daten entschlüsselt werden, sind sie immer für einen Angreifer, der Informationen über die Eingabemengen erhalten will nutzlos. Ein gutes Beispiel dafür ist das Unterprotokoll OInv
HIER BEISPIEL EINFÜGEN

Hier wird zwar eine Matrix entschlüsselt, diese ist aber randomisiert, also nutzlos, außer, man besitzt eine der geheimen Matrizen, die ja nicht verschickt werden.
Das Protokoll ist also auf eine Art und Weise aufgebaut, dass kein passiver Zuhörer irgendwelche nützlichen Informationen aus den Nachrichten, die zwischen den Parteien verschickt werden, gewinnen kann. Also kann auch ein Angreifer, der alle Nachrichten zwischen den Teilnehmern des Protokolls keine Informationen extrahieren. Also können wir auch sicher sein, dass, selbst wenn der Koordinator von einem Angreifer kontrolliert wird, die verschlüsselten Eingabedaten sicher sind.
Die neue Struktur, die einfacher zu implementieren und testen ist, macht das Protokoll also nicht weniger sicher. Besonders deshalb, weil das Protokoll so konstruiert ist, dass selbst alle verschickten Nachrichten zusammen keine geheimen Daten preisgeben.

Die meisten der Unterprotokolle bestehen aus sich abwechselnden Teilen von Kommunikation zwischen den Parteien und Berechnungen der Parteien. Also habe ich die Unterprotokolle implementiert, indem die Berechnungen der Parteien in Abschnitte aufgeteilt sind, die dann von dem Koordinator aufgerufen werden können.
BEISPIEL MPCT Das ist die einfachste Möglichkeit, um zu erreichen, dass alle Parteien zum richtigen Zeitpunkt das richtige Berechnen, denn einige der Berechnungen hängen auch von den gesendeten Nachrichten der anderen Parteien ab.

3.3 andere Paper

4 Funktion der Software

4.1 Verschlüsselungen

4.1.1 Die Wahl der Verschlüsselung

Im Paper [2] wird kein Verschlüsselungsverfahren genannt. Jedes additiv homomorphe Verschlüsselungsverfahren kann theoretisch genutzt werden. Es werden jedoch einige Vorschläge gemacht. Einer dieser Vorschläge ist das Paillier-Kryptosystem. Das Damgard-Jurik System, für das ich mich entschieden habe, ist eine Verallgemeinerung des Paillier-Kryptosystems. Unter anderem ist die Damgard-Jurik Verschlüsselung besser für Berechnungen mit mehreren Beteiligten geeignet, denn es gibt auch eine Threshold-Variante dieses Verschlüsselungs-Systems. [5] Ich habe mich bei der Verschlüsselung für das Damgard-Jurik Cryptosystem entschieden, weil es gut zu den Anforderungen, die wir haben, passt. Es ist speziell für den Anwendungsfall mit mehreren Parteien entworfen und ist deshalb auch in unserem Fall effizient.

4.1.2 Damgard-Jurik Verschlüsselung

Indem wir Shamirs Secret Sharing benutzen, können können wir einen public Key und eine beliebige Anzahl an Private Key-Shares erstellen. Shamirs Secret Sharing basiert auf Polynominterpolation und kann beispielsweise auch genutzt werden, um dem System später noch neue Parteien hinzuzufügen.[6]
Durch die Nutzung der Damgard-Jurik Verschlüsselung kann jeder, der den public Key besitzt, Daten verschlüsseln. Entschlüsseln von Daten ist jedoch komplexer. Um Daten zu entschlüsseln, müssen genügend Parteien mit Private Key-Shares die Daten teilweise entschlüsseln. Danach müssen die partiellen Entschlüsselungen kombiniert werden. Hat man aber weniger als die benötigte Anzahl an partiellen Entschlüsselungen, werden keine Informationen sichtbar. [5]
Einer der Hauptgründe, wegen dem ich mich für die Damgard-Jurik-Verschlüsselung entschieden habe ist, das man zum Entschlüsseln von Daten den Private Key nicht wiederherstellen muss. Dadurch kann der Schlüssel nicht missbraucht werden, um auch andere Dinge zu entschlüsseln. Dadurch müssen die Parteien keiner anderen Partei trauen und die Parteien können entscheiden, was entschlüsselt werden soll und was nicht. [5]
Die Verschlüsselung erfüllt natürlich auch die andere große Anforderung, die wir haben. Die Verschlüsselung ist additiv homomorph, was wir für die Funktionalität des Protokolls brauchen. [5]

4.1.3 Übertragung der Python Implementierung

Für die Implementierung der Verschlüsselung habe ich mich an der Python Implementierung [7] orientiert. Da die Sicherheit der Verschlüsselung auf mathematischen

Annahmen basiert [8], muss mit großen Zahlen (mehr als 32 Bits) gerechnet werden, um sicher gegen Brute-Force-Angriffe zu sein. Daher entstehen bei der Übertragung der Python Implementierung nach Java einige Probleme. Da in Python Ganzzahlen beliebig groß werden können, aber in Java Ganzzahlen auf 32 Bit begrenzt sind, muss ich auf BigInteger zurückgreifen, um verschlüsselte Zahlen korrekt darstellen zu können. Ein ähnliches Problem ist beispielsweise das Erstellen von zufälligen Primzahlen von bestimmter Länge. In Python stellt das kein Problem dar, aber in Java ist das erstellen von beliebig langen Primzahlen komplizierter. Trotz dieser Probleme habe ich eine funktionierende Implementierung der Damgard-Jurik Verschlüsslung geschaffen, die ich dann nutzen kann, um die Implementierung der komplexeren Teil-Protokolle zu testen.

4.2 Architektur

4.3 Ring Realisierung

Da in vielen der Unterprotokolle (OLS, ODT, OMM, secRank, OInv, SUR) Berechnungen über endlichen Körpern stattfinden, um Korrektheit und Sicherheit zu gewährleisten, benötigen wir eine robuste und effiziente Möglichkeit, mit endlichen Körpern zu arbeiten.
Da wir unter anderem lineare Gleichungssysteme über endlichen Körpern lösen müssen, und es nur wenige Java-Bibliotheken gibt, die diese Funktionalität anbieten, ist die Open-Source Bibliothek JLinalg [?] gut für unseren Fall geeignet. Die Bibliothek bietet aber standardmäßig nur eingeschränkte Funktionalitäten für beliebige Körper an. Das Lösen von linearen Gleichungssystemen ist eines der Dinge, die aber nur in den Rationalen Zahlen oder dem Körper {0;1} möglich sind. Daher war es augenscheinlich am einfachsten, die Bibliothek zu erweitern, indem ich eine neue Klasse erstellt habe, wodurch die Bibliothek jetzt auch andere Funktionalitäten über beliebig großen Körpern berechnen kann. Unter anderem kann so auch das lineare Gleichungssystem gelöst werden.

AUSSCHNITT DER NEUEN KLASSE

4.4 sichere Matrix Berechnungen

Das grundlegendste Teil-Protokoll, das im Paper [2] vorgestellt wird, ist das secMult Protokoll, das genutzt werden kann, um verschlüsselte Matrizen, also Matrizen mit verschlüsselten Einträgen miteinander zu multiplizieren. Das ist ein wichtiger Bestandteil des Protokolls, denn diese Funktion wird in vielen Teil-Protokollen genutzt.
BEISPIEL FÜR secMULT Interessanter Weise lassen sich dadurch auch verschlüsselte Zahlen multiplizieren, indem Matrizen mit nur einem Eintrag erstellt werden und diese dann multipliziert werden. Dadurch, dass wir nun verschlüsselte Zahlen addieren können (durch die additiv homomorphe Verschlüsselung) und wir jetzt auch verschlüsselte Zahlen multiplizieren können, erreichen wir ähnliche Funktionalität, wie bei voll-homomorpher Verschlüsselung. Um Zahlen zu multiplizieren brauchen wir zwar die

Mithilfe der anderen Parteien, aber nicht das Vertrauen der anderen Parteien. Denn die anderen Parteien müssen zwar ihren Teil des Secret-Keys benutzen, um ihren Teil zur Matrix Multiplikation beizutragen, aber durch das gewählte Verschlüsselungsverfahren müssen sie weder ihren Secret-Key-Share verschicken, noch werden Informationen über ihren Secret-Key-Share bekannt.
Durch die nun praktisch voll-homomorphe Verschlüsselung bekommen wir viele mathematische Möglichkeiten, die wir in den anderen Teil-Protokollen nutzen können. BEISPIEL: PolynomMultiplikation, in secDT

5 Sicherheit der Software

6 Probleme der Software

6.1 Nicht alle Protokolle sicher implementiert

Da einige der Protokolle auch auf anderen Veröffentlichungen basieren, für die keine Implementierungen zu finden waren, hat die Zeit nicht ausgereicht, um alle Protokolle sicher zu implementieren. Deshalb musste ich einige Teil-Protokolle unsicher implementieren, um die Funktionalität des Protokolls zeigen zu können.
BEISPIEL FÜR UNSICHERE IMPLEMENTIERUNG: OLS, braucht SUR, was auf dem Paper [9] basiert.

Wenn dann die relevanten Teile der anderen Paper implementiert sind, können diese dann die unsicheren Teil-Protokolle ersetzen und diese Implementierung damit sicher machen. Die Teil-Protokolle, bei denen eine sichere Implementierung möglich war, habe ich wie im Paper gezeigt sicher implementiert. Dazu zählen unter anderem die grundlegenden Protokolle, wie ßecure Matrix Multiplikation", und die interessantesten Protokolle, wie ßecure Cardinality Testing".

6.2 Bei großen Zahlen interferieren von Ring/Verschlüsselung

7 Fazit

Das Ziel war es, zu zeigen, dass die Neuerungen aus dem Paper [2] funktionieren und effizient sind. Durch meine Implementierung wird deutlich, dass das vorgestellte Protokoll funktioniert. Leider konnte ich durch die Zeitbegrenzung keine komplette, sichere Implementierung des ganzen vorgestellten Protokolls liefern, da zu viele der Teil-Protokolle auf andere Implementierungen zurückgreifen, die es zu diesem Zeitpunkt noch nicht gibt. Das Protokoll ist so nicht direkt in Anwendungen nutzbar, die auf der Datensicherheit der Eingabemengen bestehen. Dennoch habe ich gezeigt, dass das Protokoll funktioniert. Dadurch kann weitere Forschung auf dem Protokoll aufbauen und es erweitern oder verbessern.
Das Ziel dieser Arbeit ist also erreicht, auch wenn bis zu einer sicheren Implementierung noch einige Arbeit Fehlt.

8 Quellen

Literatur

[1] Digma Kogan. A few lessons from the history of multiparty computation. https://theorydish.blog/2021/05/26/few-lessons-from-the-history-of-multiparty-computation, May 2021. Stand: 17.7.2021.

[2] Sihang Pu Pedro Branco, Nico Döttling. Multiparty cardinality testing for threshold private setintersection. 2021.

[3] X Yi, R Paulet, and E Bertino. Homomorphic encryption. in: Homomorphic encryption and applications. springerbriefs in computer science, 2014.

[4] Satrajit Ghosh and Mark Simkin. The communication complexity of threshold private set intersection. Cryptology ePrint Archive, Report 2019/175, 2019. `https://eprint.iacr.org/2019/175`.

[5] Mads Jurik Ivan Damgard. A generalization of paillier's public-key systemwith applications to electronic voting. 2004.

[6] Adi Shamir. How to share a secret. 1979.

[7] swansonk14. Damgard-jurik. https://github.com/cryptovoting/damgard-jurik.

[8] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[9] Eike Kiltz, Payman Mohassel, Enav Weinreb, and Matthew K. Franklin. Secure linear algebra using linearly recurrent sequences. In *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*, pages 291–310. Springer, 2007.