

# Unser erstes Dokument

Latex Tutorial

08. Februar 2013

## Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>2</b>
1.1	Anwendungsbeispiel . . . . .	2
1.2	Zielsetzung . . . . .	2
<b>2</b>	<b>Grundlagen</b>	<b>2</b>
2.1	Homomorphe Verschlüsselungen . . . . .	2
<b>3</b>	<b>Multiparty Cardinality Testing</b>	<b>3</b>
3.1	Neues am Paper . . . . .	3
3.2	Abwandlungen zum Paper . . . . .	3
3.3	andere Paper . . . . .	4
<b>4</b>	<b>Funktion der Software</b>	<b>4</b>
4.1	Verschlüsselungen . . . . .	4
4.1.1	DDH/Damgard-Jurik . . . . .	4
4.1.2	Damgard-Jurik Verschlüsselung . . . . .	4
4.1.3	Übertragung der Python Implementierung . . . . .	5
4.2	Architektur . . . . .	5
4.3	Ring Realisierung . . . . .	5
4.4	sichere Matrix Berechnungen . . . . .	5
<b>5</b>	<b>Sicherheit der Software</b>	<b>5</b>
<b>6</b>	<b>Probleme der Software</b>	<b>5</b>
6.1	Nicht alle Protokolle sicher implementiert . . . . .	5
6.2	Bei großen Zahlen interferieren von Ring/Verschlüsselung . . . . .	5
<b>7</b>	<b>Fazit</b>	<b>5</b>
<b>8</b>	<b>Literatur</b>	<b>5</b>

# 1 Einleitung

## 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 [1] 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 [1] 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 [1] 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. [2]

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

### 3.2 Abwandlungen zum Paper

Das Protokoll wurde offensichtlich entworfen um auch mit mehreren teilnehmenden Parteien effizient zu sein. Das Paper [1] 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 "verschickten" Informationen 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 DDH/Damgard-Jurik

##### 4.1.2 Damgard-Jurik Verschlüsselung

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.

Indem wir Shamirs Secret Sharing benutzen, 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.[3]

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. [4]

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. [4]

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. [4]

#### **4.1.3 Übertragung der Python Implementierung**

### **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**

## **5 Sicherheit der Software**

## **6 Probleme der Software**

### **6.1 Nicht alle Protokolle sicher implementiert**

### **6.2 Bei großen Zahlen interferieren von Ring/Verschlüsselung**

## **7 Fazit**

## **8 Literatur**

### **Literatur**

- [1] Sihang Pu Pedro Branco, Nico Döttling. Multiparty cardinality testing for threshold private setintersection. 2021.
- [2] X Yi, R Paulet, and E Bertino. Homomorphic encryption. in: Homomorphic encryption and applications. springerbriefs in computer science, 2014.
- [3] Adi Shamir. How to share a secret. 1979.

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