

Orsum occulendi

AES in Java | PKN

Lukas Bischof, Philipp Fehr

Inhaltsverzeichnis

[Einleitung 2](#_Toc481613811)

[Das Problem 2](#_Toc481613812)

[Planung 2](#_Toc481613813)

[Verwendete Lösung 3](#_Toc481613814)

[Einleitung 3](#_Toc481613815)

[Funktionsweise AES 4](#_Toc481613816)

[Reflexion 6](#_Toc481613817)

# Einleitung

Philipp Fehr und Lukas Bischof haben in dem Projekt „Orsum Occulendi“ versucht,  
den AES-Algorithmus in Java umzusetzen. Damit die Kommunikation in unserem bereits vorhandenen Java Vier-Gewinnt verschlüsselt werden kann.  
Während der Entwicklung wurde das Projekt zusätzlich von Herrn Tromsdorff  
und Herrn Veselcic betreut.

# Das Problem

Das zentrale Problem des Projekt ist, dass die Verschlüsselung unseres Java Vier-Gewinnt Spieles nicht verschlüsselt übertragen wird und somit eine erhebliche Sicherheitslücke aufweist, da man mit einer Man-In-The-Middle Attacke die Spiele verändern sowie die Spieler getäuscht werden können.

# Planung

Zuerst haben wir uns einen groben Überblick über die Funktionsweise von AES verschafft. Dabei hat uns eine Aufnahme von einer Präsentation von Christof Paar sehr geholfen. Während der Umsetzung untersuchten wir dann jeder einzelne Schritt und setzten diesen dann um, sobald wir die Mechanik verstanden hatten.

Als wir dann mit dem Algorithmus fertig waren, implementierten wir diesen in unser Server/Client Modell des Spiels.

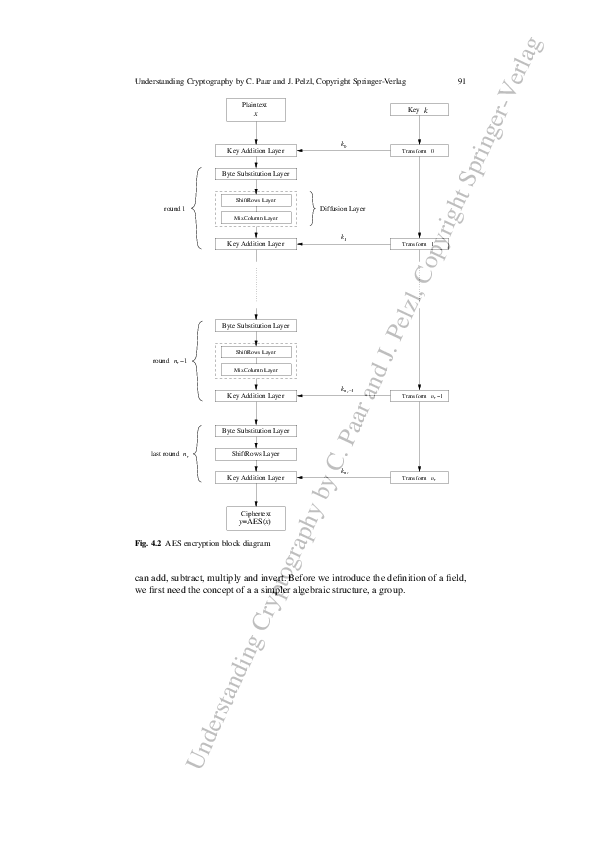
# Verwendete Lösung

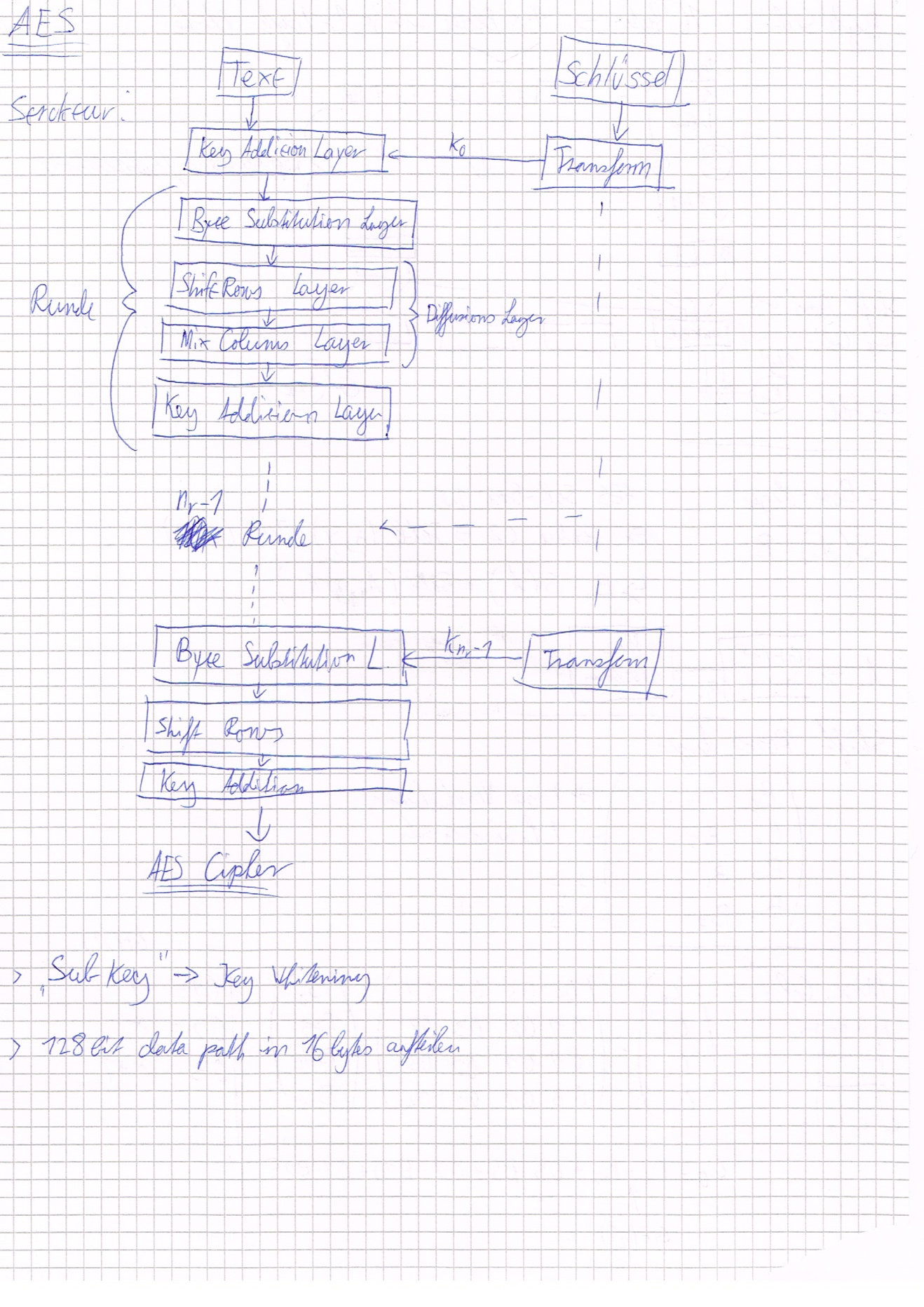
## Einleitung

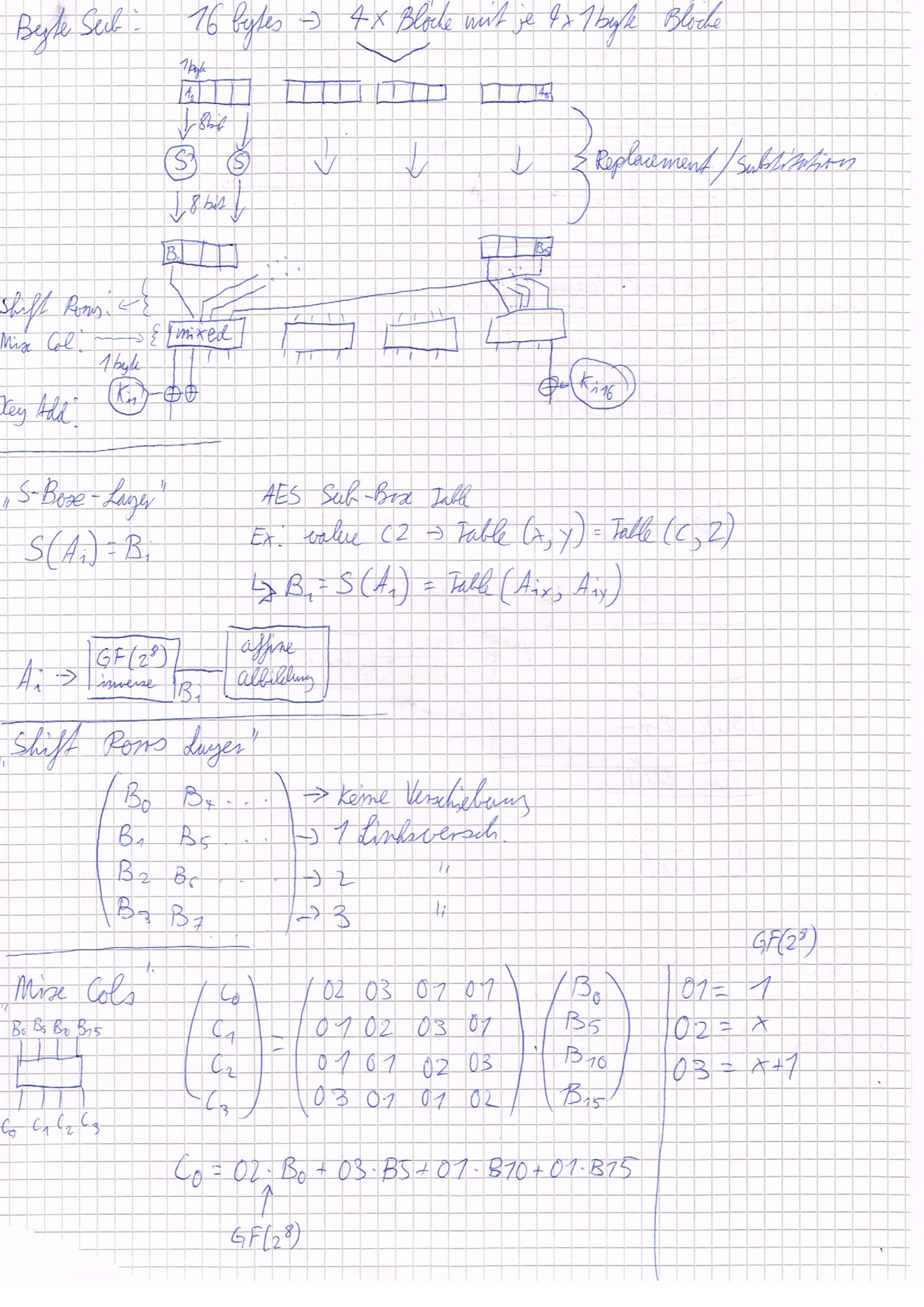
Da eine fertige AES-Library zu verwenden langweilig wäre und keine Herausforderung bieten würden, haben wir uns dazu entschieden AES selber zu umzusetzen. Um zudem vollumfänglich eine sichere Kommunikation zu ermöglichen haben wir unser Krypto System mit den eingebauten Java-Libraries mit einem RSA Schlüsselaustausch erweitert.

## Funktionsweise AES

AES besteht eigentlich nur aus fünf Methoden; KeyExpansion, ByteSubsitution, ShiftRow, MixColumn und AddRoundKey. Diese werden in unserem Fall bis zu 10 Mal auf den zu verschlüsselnden Text ausgeübt. Die KeyExpansion ist dafür verantwortlich aus dem Schlüssel sogenannte Rundenschlüssel zu generieren. Diese werden in den 10 Runden die durchlaufen werden verwendet. Die ByteSubsitution sorgt dafür, dass sich Bytes verändern und die ShiftRow und MixColumn sorgen dafür, dass sich diese Veränderungen auch über den gesamten Text verteilen und nicht nur an einem Ort sind. Die AddRoundKey Methode fügt, wie es der Name schon sagt, den Rundenschlüssel bitweise zu dem zu verschlüsselnden Text hinzu. Die erste und letzte Runde sind etwas anders als die «mittleren». Dies sieht man auch in den folgenden Bildern.

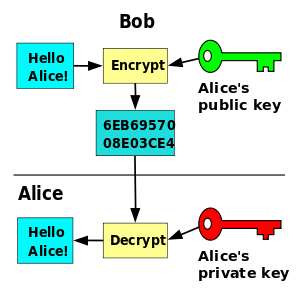






## Funktionsweise RSA

RSA funktioniert mir Public und Privat Keys. Der Public Key wird, wie es der Name bereits sagt, der Öffentlichkeit zugänglich gemacht. Mit diesem kann der andere Gesprächspartner nun eine Nachricht verschlüsseln und an, in diesem Fall, Alice schicken. Alice kann die Nachricht, dann mit ihrem Private Key entschlüsseln.



# Reflexion

Nachwort Philipp

Die Teamarbeit hat sehr gut funktioniert und wir haben uns die Arbeit sinnvoll und fair aufgeteilt. Leider haben sich ein paar unnötig dumme Fehler eingeschlichen. Doch konnten wir diese, auch wenn mit erheblichem Zeitverlust beheben.

Ich finde, dass es ein erfolgreiches Projekt ist, welches so funktioniert, wie wir uns das gewünscht haben.

Nachwort Lukas

In dem Projekt haben wir den Algorithmus sehr gut und koordiniert umgesetzt. Jedoch schlichen sich manchmal dumme Fehler ein, welche wir teilweise über eine Woche lang suchten und uns nach dem Finden dann dementsprechend über die Banalität aufregten.

Ich finde trotzdem, dass dies ein gelungenes Projekt ist, welches restlos nach unseren Wünschen funktioniert.