Inhaltsverzeichnis

1 Einführung und Grundlagen

1.1 Einleitung

Moderne Verschlüsselungsalgorithmen sind im Allgemeinen sehr rechenintensiv und werden oft als Bestandteil des Betriebssystems ausgeführt. Da gewöhnliche CPUs für diese Art von Operationen nicht ausgelegt sind, wird das gesamte System durch die entstehende Auslastung gebremst. Es liegt also nahe, eine geeignetere Plattform für die Berechnung von Verschlüsselungen zu nutzen.

Im Rahmen dieses Praktikums wird daher evaluiert, inwiefern sich moderne Grafikkarten bzw. FPGAs zur optimierteren Ausführung nutzen lassen. Die Verschlüsselungen sollen transparent in den Linux-Kernel eingebunden und systemweit zur Verfügung gestellt werden.

Im Folgenden wird mithilfe des CUDA-Frameworks von NVIDIA der AES-Algorithmus auf einer GPU vom Typ "GTS 8800" aus dem Hause NVIDIA implementiert und durch eine Reihe von Tests auf eine eventuelle Verbesserung der Datendurchsatzrate hin überprüft.

1.2 Grundlagen

1.2.1 AES im Kurzüberblick

Der Advanced Encryption Standard (AES) ist ein symmetrisches Kryptosystem. Es wurde von Joan Daemen und Vincent Rijmen im Rahmen eines international ausgeschriebenen Wettbewerbes des National Institute of Standards and Technology (NIST) entwickelt. Als Nachfolger von DES und 3DES, gilt AES seit 2000 als De-facto Verschlüsselungsstandard, welcher Dank seiner starken Verschlüsselung selbst höchsten Sicherheitsansprüchen genügt.

Bei AES handelt es sich um ein Blockverschlüsselungssystem, auch Blockchiffre genannt, also ein Verschlüsselungsverfahren, bei dem der Klartext in eine Folge gleichgroßer Blöcke zerlegt wird. Diese Blöcke werden anschließend unabhängig voneinander mit einem aus einem Schlüsselwort berechneten Blockschlüssel chiffriert. Somit werden auch Chiffretextblöcke mit einer festen Länge erzeugt und letzendlich zum endgültigen Chiffretext aneinandergereiht.

AES schränkt die Blocklänge auf 128 Bit ein. Die Schlüssellänge kann jedoch zwischen 128, 192 und 256 Bit gewählt werden, weshalb zwischen den drei AES-Varianten AES-128, AES-192 und AES-256 unterschieden wird. AES bietet ein sehr hohes Maß an Sicherheit und ist in den USA sogar für staatliche Dokumente mit höchster Geheimhaltungsstufe zugelassen. Der Algorithmus ist frei verfügbar und darf ohne Lizenzgebühren eingesetzt sowie in Software bzw. Hardware implementiert werden.

1.2.2 CUDA Framework

Das „Compute Unified Device Architecture Software Developer Kit“ (CUDA SDK) wurde von NVIDA am 15. Februar 2007 erstmals der Öffentlichkeit vorgestellt. Intention dieses SDKs ist, die Programmierung aktueller Grafikkarten unter einer einheitlichen und standardisierten Schnittstelle zu ermöglichen.

Die Architektur moderner GPUs ist aufgrund ihrer Geschichte als reine Berechnungseinheit für Bildschirmausgaben für den Zweck ausgelegt, Operationen parallel auszuführen. Als Co-Prozessor können GPUs somit Dank der CUDA-API dazu genutzt werden, bestimmte Programmteile signifikat schneller abzuarbeiten.

CUDA basiert auf einer optimierten Variante von C (“C for CUDA”) und ist damit weitestgehend plattformunabhängig. So ist es möglich, CUDA-Anwendungen unter Windows, Linux und Mac OS X auszuführen - eine kompatible Grafikkarte vorausgesetzt.

2 GPU Architektur

2.1 Architektur

2.1.1 Prozessoraufbau

Moderne Grafikarten sind über die PCI-E Schnittstelle an die CPU angebunden. Über diesen Bus werden die Daten und Prozesse an die berechnenden Einheiten der GPU übertragen.

Im Folgenden ist die Architektur der GPU abgebildet. Hierbei ist zu beachten, dass die Anzahl der Multiprozessoren auf der GPU, sowie die Anzahl der Streaming-Prozessoren (SP) je nach Modell unterschiedlich sind.

2.1.2 Multithreaded Instruction Unit (MT IU)

Die MT IU verwaltet die Ausführung von Threads auf dem Multiprocessor. Hierbei werden einzelne Threads zu einem Block, mehrere Blöcke zu einem Grid zusammengefasst.

2.1.3 Streaming Multiprocessor

Auf jeden Streaming Multiprocessor wird genau ein Block abgebildet. Die in dem Block befindlichen Threads werden -soweit möglich- parallel abgearbeitet. Hierbei hat jeder Thread in dem Block eine eindeutige ID, auf welche auch in dem Thread zugegriffen werden kann.

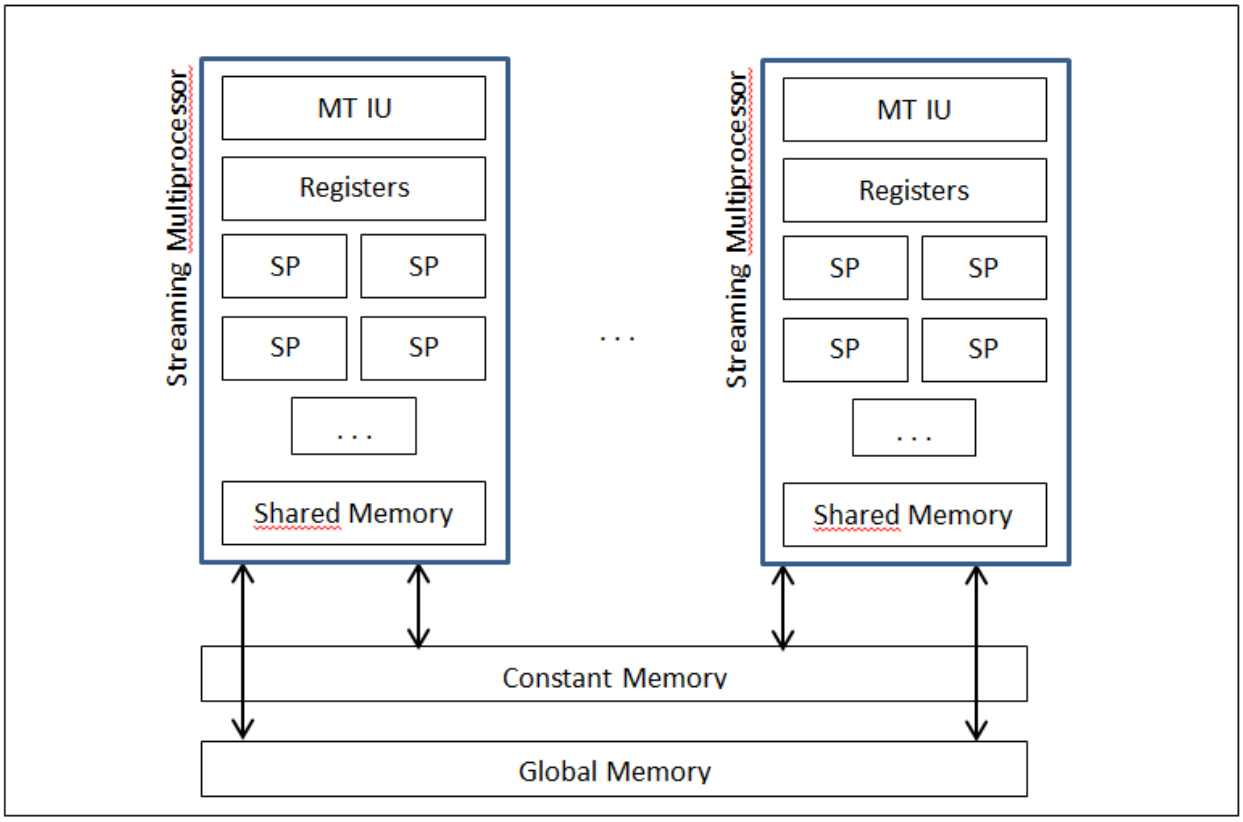

Abbildung 1: GPU-Architecture

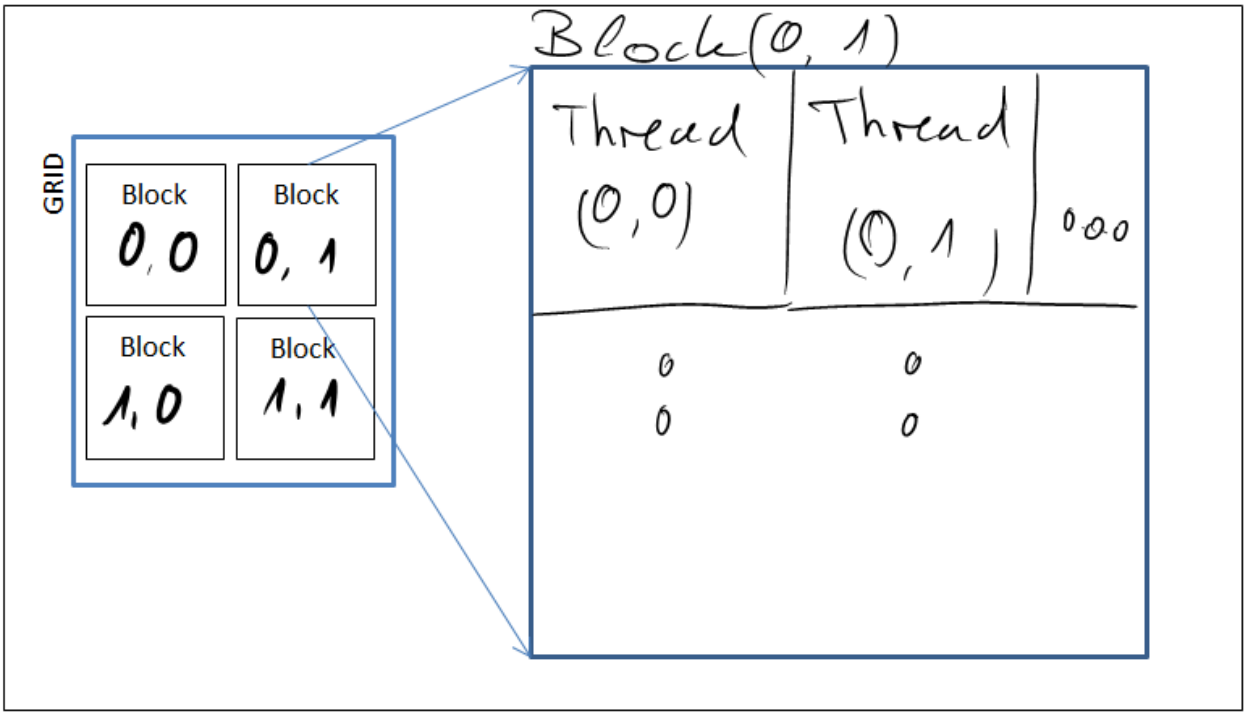

Abbildung 2: Streaming Processor

2.1.4 Streaming Processor (SP)

Jeder SP führt genau einen Tread aus.

2.2 Speicherhierarchie

2.2.1 Global Memory

Der Global Memory (RAM) ist der Größte Bereich und sowohl von CPU als auch von der GPU schreib- und lesbar. Dieser Speicher ermöglicht den Austausch von Daten zwischen GPU und CPU. Dieser Speicher hat die größte Kapazität, ist jedoch der langsamste der Speicherhierarchie bezüglich der GPU.

2.2.2 Constant Memory

Der Constant Memory ist physikalisch auf 64KB beschränkt.

2.2.3 Register

Threads eines Bocks teilen sich gemeinsame Register.

Shared Memory

Für jeden Streaming-Multiprozessor-Prozessor ist ein Shared Memory vorgesehen, welches der schnellste -mit 16 KB jedoch auch der kleinste- Speicher in der Hierarchie der GPU ist. Der Multiprozessor teilt diesen verfügbaren Speicher und seinen Streamingprozessoren auf. Dieser Speicher kann nur von der GPU gelesen und geschrieben werden.

3 AES-Implementierung

3.1 Algorithmus im Detail

Im diesem Kapitel wird der Ablauf des AES Algorithmus vorgestellt. Er besteht aus mehreren Phasen, auf die im Folgenden genauer eingegangen wird.

Jeder Block wird zunächst in eine zweidimensionale Tabelle mit vier Zeilen und 4 Spalten geschrieben, deren Zellen ein Byte groß sind. Jeder Block wird nun nacheinander bestimmten Transformationen unterzogen. Anstatt jeden Block einmal mit dem Schlüssel zu verschlüsseln, wendet Rijndael verschiedene Teile des erweiterten Originalschlüssels nacheinander auf den Klartext-Block an. Die Anzahl r dieser Runden variiert und ist von der Schlüssellänge k abhängig:

Schlüssellänge k	Runden r
128	10
192	12
256	14

Tabelle 1: Rundenlängen

Der Ablauf jeder einzelnen Block-Verschlüsselung entspricht folgendem Schema:

- Schlüsselexpansion (entfällt bei ECB-Modus)
- Vorrunde
  - AddRoundKey (Rundenschlüssel[0])
- Verschlüsselungsrunden (runde $<$ R)
  - Substitution
  - ShiftRow
  - MixColumn
  - AddRoundKey (Rundenschlüssel[runde])
- Schlussrunde

Abbildung 3: Berechnung der Rundenschlüssel

- Substitution
- ShiftRow
- AddRoundKey (Rundenschlüssel[R])

3.1.1 Schlüsselexpansion

Der Benutzerschlüssel wird in r+1 Teilschlüssel aufgeteilt, die sogenannten Rundenschlüssel. Diese müssen dieselbe Länge wie die Blöcke haben, was bedeutet, dass der Benutzerschlüssel zunächst auf die Länge 128*(r+1) expandiert werden muss. Aus diesem werden die Rundenschlüssel erzeugt und ebenfalls in Tabellen mit 4 Spalten und 4 Zeilen gespeichert. Für die Erzeugung der ersten Spalte des jeweils nächsten Rundenschlüssels wird zunächst die letzte Spalte des vorherigen Schlüssels, am Anfang also des Benutzerschlüssels, genommen und um eine Zeile nach oben rotiert. Die oberste Zelle wird unten wieder eingefügt. Nun erfolgt eine Substitution der vier Werte mit Hilfe der Substitutionsbox. Sie ist meist als Array aufgebaut und gibt an, wie jedes Byte durch einen anderen Wert zu ersetzen ist.

Die Konstruktion der S-Box unterliegt Designkriterien, die die Anfälligkeit für die Methoden der linearen und der differentiellen Kryptoanalyse sowie für algebraische Attacken minimieren sollen. Mit Hilfe der S-Box wird jedes Byte des Blocks durch ein Äquivalent ersetzt und die Daten somit monoalphabetisch verschlüsselt. Diese neuerzeugte Spalte wird mit der drei Spalten zurückliegenden Spalte und mit der ersten Spalte der sogenannten Rcon-Tabelle XOR-verknüpft.

Die Rcon-Tabelle ist ebenfalls in Form eines Arrays mit 4 Zeilen und einer Spalte für jeden Rundenschlüssel aufgebaut. Sie enthält konstante Werte die auf einem mathematischen System beruhen. Die aus der XOR-Verknüpfung erzeugten Werte ergeben die erste Spalte des nächsten Rundenschlüssels. Die restlichen drei Spalten ergeben sich jeweils aus einer XOR-Verknüpfung der davor liegenden und der zu dieser drei zurückliegenden Spalte. Für alle folgenden Rundenschlüssel läuft die Berechnung analog ab. Abbildung 3 veranschaulicht dieses Vorgehen.

3.1.2 Vorrunde

Bei AddRoundKey erfolgt eine XOR-Verknüpfung zwischen dem zu verschlüsselnden Block und dem ersten Rundenschlüssel (siehe Abb.4). Nur an dieser Stelle ist der Algorithmus vom Benutzerschlüssel abhängig.

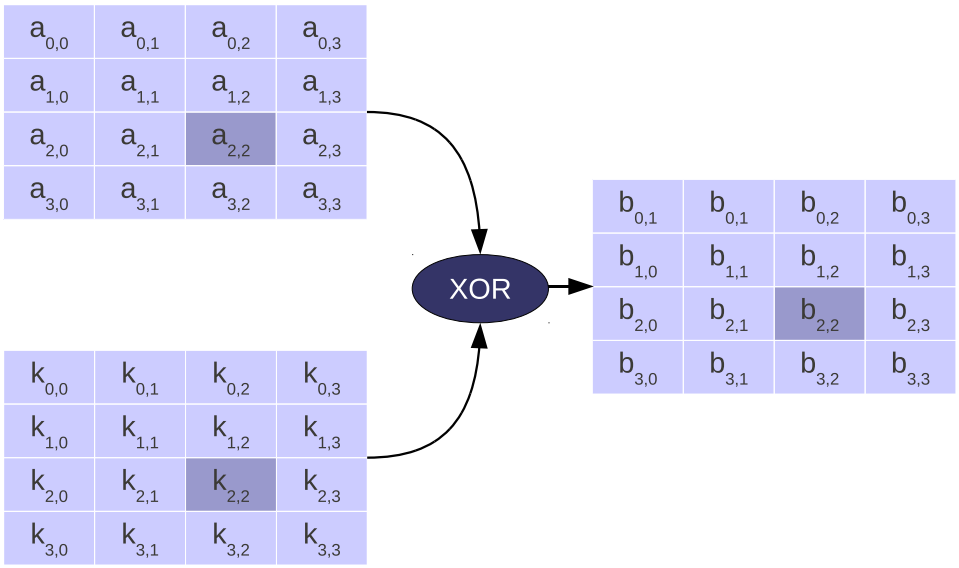

Abbildung 4: AddRoundKey

3.1.3 Verschlüsselungsrunden

In den folgenden Verschlüsselungsrunden wird zunächst eine Substitution durchgeführt.

Dazu wird die Substitutionsbox verwendet. Im darauffolgenden Schritt, genannt ShiftRow, werden die Zeilen um eine bestimmte Anzahl von Spalten nach links verschoben und links hinausgeschobene Zellen rechts wieder angefügt. Die erste Zeile bleibt konstant, die zweite wird um eine, die dritte um zwei und die vierte um drei Spalten verschoben. Abbildung 5 veranschaulicht dieses Vorgehen.

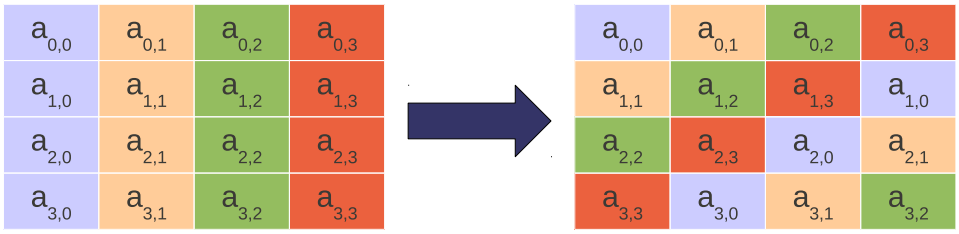

Abbildung 5: ShiftRow

Im MixColumn-Schritt wird zunächst jede Zelle mit einer Konstanten multipliziert und dann die Spalten mit den Ergebnissen der Multiplikation XOR verknüpft. Durch eine geschickte Analyse dieser Operation, vereinfacht sich die Rechnung zu einer simplen Matrixmultiplikation.

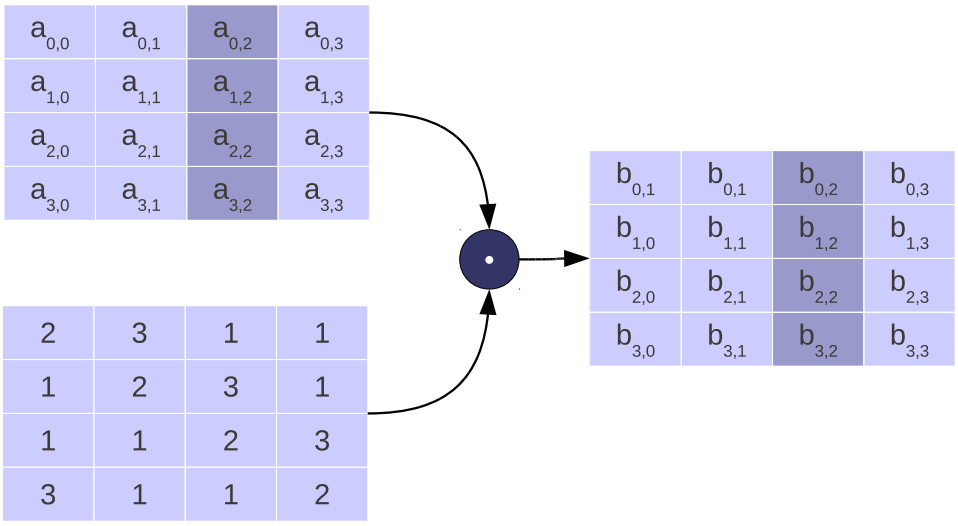

Abbildung 6: MixColumn

Nun folgt zum Abschluss jeder Verschlüsselungsrunde noch ein Mal AddRoundKey. Abgeschlossen wird die Verschlüsselung mit der Schlussrunde, in der noch ein Mal die Schritte Substitution, ShiftRow und AddRoundKey angewendet werden.

Die Schlussrunde verläuft identisch zu den übrigen Verschlüsselungsrunden mit der Ausnahme, dass keine MixColumn-Funktion ausgeführt wird.

3.1.4 Entschlüsselung

Bei der Entschlüsselung von Daten wird genau rückwärts vorgegangen. Die Daten werden zunächst wieder in zweidimensionale Tabellen gelesen und die Rundenschlüssel generiert. Allerdings wird nun mit der Schlussrunde angefangen und alle Funktionen in jeder Runde in der umgekehrten Reihenfolge aufgerufen. Durch die vielen XOR-Verknüpfungen unterscheiden sich die meisten Funktionen zum Entschlüsseln nicht von denen zum Verschlüsseln. Jedoch muss eine andere S-Box genutzt werden (die sich aus der originalen S-Box berechnen lässt) und die Zeilenverschiebungen erfolgen in die andere Richtung.

3.2 C++ Implementierung

Ausgangspunkt der weiteren Bearbeitung ist eine unfertige C++ Implementierung des Algorithmus von Paulo S. L. M. Barreto. Der Autor hat bereits erste Ansätze zur Datenverschlüsselung auf einer GPU programmiert, hat das Projekt jedoch aus unbekannten Gründen eingestellt, sodass noch einige Ergänzungen und Änderungen vorgenommen werden müssen.

Die Implementierung ist Dank der Ausnutzung komplexerer Galois-Feld-Operationen vergleichsweise schnell. Hierbei werden die Berechnungen durch Nutzung vorkalkulierter

Tabellen auf Kosten der Speichereffizienz beschleunigt. Der Aufwand der Verschlüsselung verringert sich somit auf eine Reihe (schneller) Binäroperationen, welche sich auf der GPU vorteilhaft parallel ausführen lassen können.

Zusätzlich ist anzumerken, dass zugunsten der Parallelität der sogenannte "Electronic Code Book Mode" (ECB) verwendet wird, also jeder Block mit dem gleichen expandierten Schlüssel chiffriert wird.

3.3 CUDA-spezifische Veränderungen

3.3.1 Prozessaufteilung

3.3.2 Speichernutzung

4 Tests und Benchmarks

4.1 Testumgebung

4.2 Ergebnisse

5 Ausblick