Praktikum 3 – AES-Algorithmus in Software auf dem ARM Cortex-A53 und dem ARM Cortex-R5

Lernziele

- Sie können auf einem System mit Embedded Linux die Performance Ihrer Software messen.
- Sie können auf Ihrem Host-Rechner einen Verschlüsselungsalgorithmus für Ihre Embedded-Plattform in Software für verschiedene CPUs implementieren.
- Sie kennen das Konzept, wie ein Algorithmus als Remote-Procedure auf einem zweiten Prozessor ausgeführt werden kann.
- Sie verstehen, welche Schritte nötig sind, um eine Remote-Procedure bei Linux anzumelden und diese zu benutzen.

Grundlagen

Verschlüsselung

In vielen Embedded-Anwendungen ist es notwendig, die Kommunikationswege und Daten gegen unbefugten Zugriff zu schützen. Hierzu kommen Verschlüsselungsverfahren zur Anwendung. Eine der wichtigsten Anforderungen an ein Verschlüsselungsverfahren ist, den Aufwand, es zu knacken, möglichst hoch zu halten. Eine brute-force-Attacke, also das Durchprobieren aller möglichen Schlüssel, soll so umständlich und teuer wie möglich gemacht werden.

Der DES (Data Encryption Standard) mit einer Schlüssellänge von 56 Bit wurde 1997 zum ersten Mal im Rahmen eines Wettbewerbs (DESCHALL, https://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/) geknackt. Dafür benötigte man 1998 noch 56 Stunden. Im Jahr 2021 konnte DES innert 5 Minuten geknackt werden, da riesige, parallele Rechenkapazität sehr günstig geworden war.

Die USA beauftragten 1997 das National Institute of Standards and Technology (NIST) damit, einen neuen Verschlüsselungsalgorithmus auszuwählen, der schwieriger zu knacken ist. Mit dem AES-Algorithmus wird dies gewährleistet:

- Mit 128- bis 256-Bit langen Schlüsseln bietet AES generell hohe Sicherheit und mit der wählbaren Schlüssellänge eine gewisse Flexibilität bezüglich des Rechenaufwands.
- Je länger der Schlüssel, desto höher wird auch der Rechenaufwand für Ver- und Entschlüsselung.
- AES kann sowohl in Software als auch in Hardware effizient implementiert werden.
- AES hat verschiedene Modi mit unterschiedlichen Stärken und Schwächen.

In diesem Praktikum werden Sie eine Softwareimplementierung von AES im Cipher-Block-Chaining-Modus (CBC) verwenden. In diesem Modus wird der verschlüsselte Block (Ciphertext) wie in einer Kette mit dem nächsten unverschlüsselten Block (Plaintext) in einer XOR-Operation verrechnet, um repetitives Verhalten des Algorithmus zu verunmöglichen. So besteht keine 1:1-Beziehung mehr zwischen dem Plaintext und dem Ciphertext, weil dem Algorithmus bei jedem Block des Plaintexts garantiert ein anderer Input übergeben wird. Der erste Block des Plaintexts wird ausserdem mit einem Initialisierungsvektor (IV) XOR verrechnet. Das Verschlüsselungsverfahren ist in Abbildung 1 gezeigt, die Entschlüsselung in Abbildung 2.

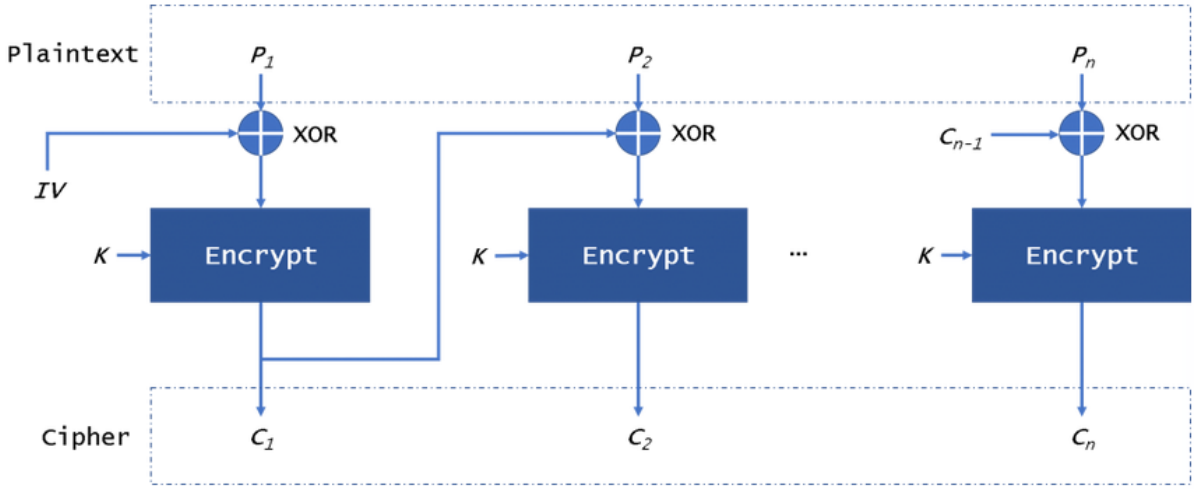

Abbildung 1 Cipher-Block-Chaining-Verschlüsselung mit dem AES-Algorithmus. Durch die Verwendung des verschlüsselten vorangehenden Blocks bei der Verschlüsselung eines weiteren Blocks kann die Verschlüsselung nur single-threaded ausgeführt werden. (Bild: https://www.highgo.ca/2019/08/08/the-difference-in-five-modes-in-the-aes-encryption-algorithm/)

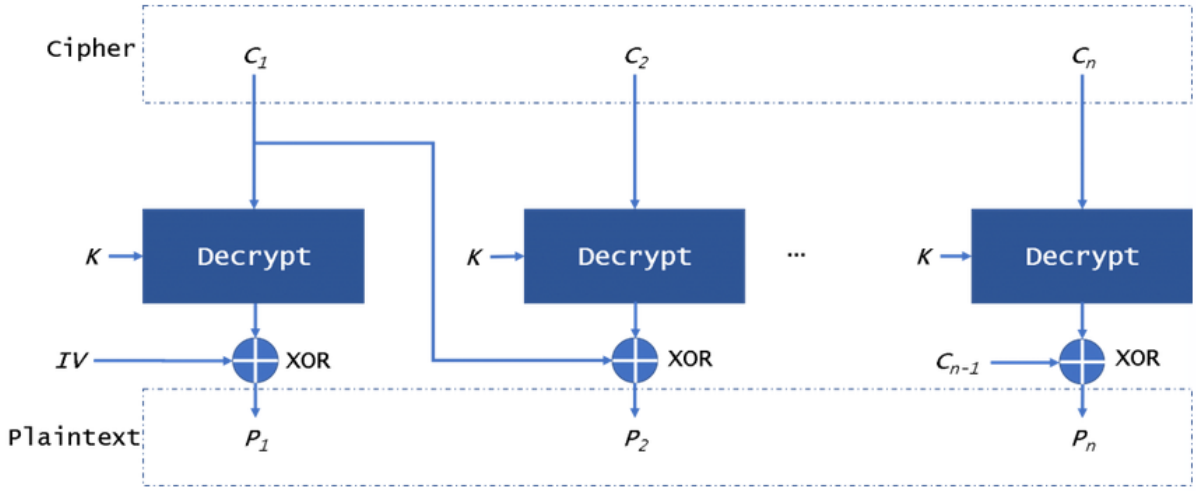

Abbildung 2 Cipher-Block-Chaining-Entschlüsselung mit AES. Für die Entschlüsselung werden nur der Schlüssel und die beiden benachbarten Blöcke des Ciphertexts benötigt. Das ermöglicht die Entschlüsselung parallel durchzuführen. (Bild: https://www.highgo.ca/2019/08/08/the-difference-in-five-modes-in-the-aes-encryption-algorithm/)

Remote Procedures

Remote Procedures sind Funktionen, die auf einem separaten, externen Prozessor ausgeführt werden. Damit lagert man den Rechenaufwand auf zusätzliche Prozessorressourcen aus. Dies erfordert Kommunikation zwischen der internen Applikation und dem externen Prozessor: Die Remote-Funktion muss auf dem zweiten Prozessor aufgesetzt werden, Input- und Output-Daten müssen transferiert werden, und die aufrufende Applikation muss informiert werden, wenn die Ausführung beendet ist.

Diese Kommunikation ist in der Abbildung 3 gezeigt. Ihre Applikation läuft auf dem ARM Cortex-A53 unter Linux. Im Kernel stehen die Treiber *rpmsg* und *rproc* zur Verfügung, die mit ihrem Gegenstück im Cortex-R5 kommunizieren. Die Applikation benutzt den Kommunikationskanal, um die Remote Procedure im Cortex-R5 zu initialisieren und aufzurufen.

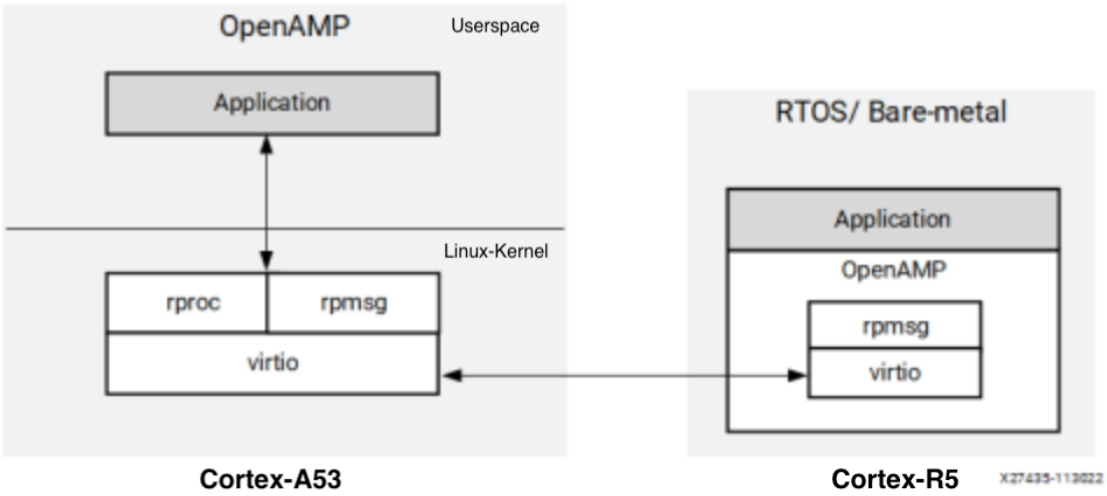

Abbildung 3 Remote-procedure setup mit Kommunikation über den Linux-Kernel. Die Applikation läuft auf dem Cortex-A53 und kommuniziert über den Treiber im Linux-Kernel mit der Remote Procedure im Cortex-R5. Bild: Xilinx Libmetal and OpenAMPUser Guide

Der Ablauf von Setup bis Shutdown ist in Abbildung 4 ersichtlich. Im rot gestrichelten Pfeil sind folgende Schritte enthalten:

1. Lade die Firmware auf den Remote-Processor.
2. Erstelle eine remoteproc-struct mit der Resource Table der Firmware.
3. Registriere die RPMsg-Callback-Funktion. (Callback-Funktion: Wird einer anderen Funktion als Argument mitgegeben, um zum Beispiel nach Erledigung einer Aufgabe dies dem Aufrufer aktiv zu melden.)
4. Erstelle ein virtuelles File für die "RPMsg virtio", das als Kommunikationskanal dient. (siehe Abbildung 3)

Danach folgen die weiteren Schritte wie in Abbildung 4 gezeigt.

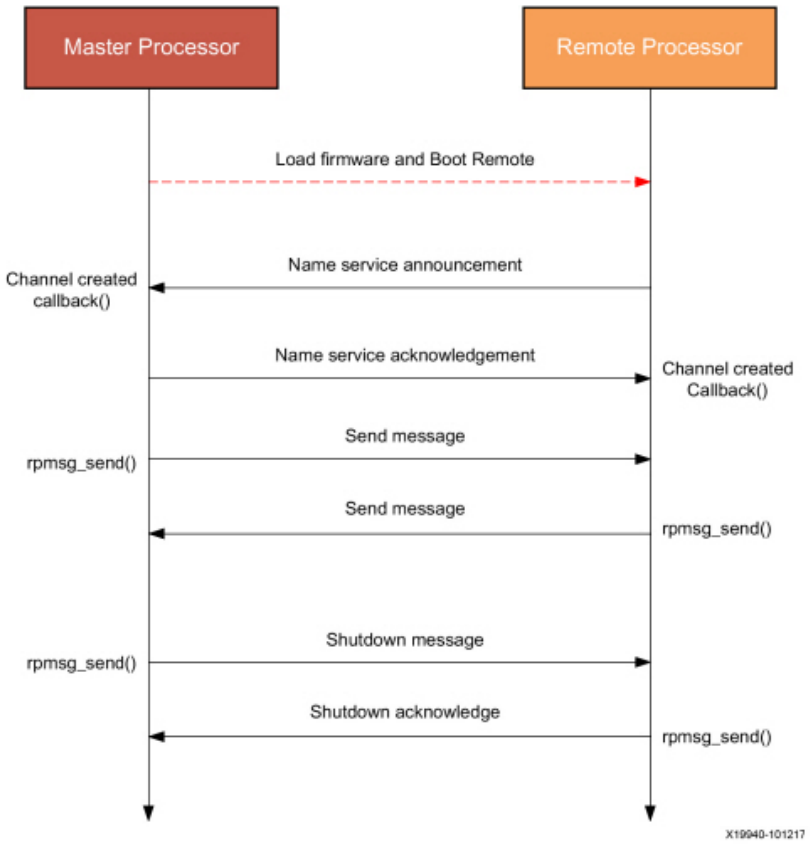

Abbildung 4 Ablauf der Kommunikation zwischen Master-Processor und Remote-Processor. Zu Beginn muss die Firmware für die Remote Procedure in den Remote-Processor geladen werden. Die Firmware meldet dann die Callback-Funktion beim Master an, welcher den sog. Name-Service bestätigt. Danach steht ein Kommunikationskanal für den Aufruf der Remote-Procedure zur Verfügung, bis die Shutdown-Message ausgetauscht wird.

Ziel des Praktikums

Da Verschlüsselungsalgorithmen prinzipbedingt einen hohen Rechenaufwand benötigen, lohnt es sich, den Energie- und Zeitaufwand ihrer Ausführung zu messen. Diese Messungen helfen bei der Entscheidung, welche Prozessorressource für einen Algorithmus am besten geeignet ist.

Insbesondere bei Embedded-Systemen, die unabhängig betrieben werden (Batteriebetrieb, Energy-Harvesting), kann eine solche Analyse für verschiedene Implementierungen angebracht sein.

Am ultra96-Board werden Sie die (berechtigte) Kritik anbringen, dass das Gerät bereits im Ruhezustand ordentlich warm wird. Die Leistung, die für den Crypto-Algorithmus aufgewendet wird, ist im Verhältnis zum ständigen Verbrauch verschwindend klein. Das Board bietet sich trotzdem für die Performancemessungen an, weil es über unterschiedliche Ressourcen für das Processing verfügt. Diese werden Sie im Laufe des Semesters ausprobieren.

Im heutigen Versuch werden Sie einen vorbereiteten AES-Algorithmus auf einem ARM Cortex-A53 und auf einem ARM Cortex-R5 implementieren und eine Zeitmessung für dessen Ausführung durchführen.

Aufgabenstellung

Die Source Files für den Versuch liegen wieder im Git-Repository https://github.zhaw.ch/InES/ESE_HS25_stud, im Verzeichnis *P3*.

Ausführung in der Application Processing Unit (APU), ARM Cortex-A53

Das Programmgerüst für diesen Teil liegt im Unterverzeichnis 'APU'. Sie finden hier wieder ein vorbereitetes *Makefile* und ein *src*-Verzeichnis. In der Datei *main.c* finden Sie das Programmgerüst, das Sie an einigen Stellen ergänzen sollen. In *aes_apu.c* sind die Funktionen definiert, welche die eigentlichen Schritte des AES-Algorithmus (in *aes.c*) aufrufen. Wir verwenden diese Modul-Struktur, damit wir den Algorithmus in späteren Praktika einfach auf mehreren Prozessoren vergleichen können.

1) Machen Sie sich zuerst mit der Programmstruktur vertraut.
   Welche Funktionen müssen aufgerufen werden, bevor die eigentliche Verschlüsselung oder Entschlüsselung stattfinden kann?
   Was machen diese Funktionen (gesucht: zwei Schritte)?
   AES_APU_init_ctx_iv(key, iv);
   1. Schlüssel vorbereiten
   2. Initialisierungsvektor setzten

2) In *time.h* sind Funktionen deklariert, die auf Zeitdienste des Betriebssystems zugreifen. Mit der Realtime-Clock von Linux kann in einem Programm eine für unsere Zwecke genügend genaue Zeitmessung gemacht werden.
   Setzen Sie die Funktion *clock_gettime(…)* im Modus *CLOCK_REALTIME* ein, um an sinnvollen Punkten in der Main-Funktion Zeitstempel (*time_start, time_stop*) zu erfassen. (Manual: https://linux.die.net/man/3/clock_gettime)
   Es ist sicher sinnvoll, das Setzen der Zeitstempel so nah wie möglich an den zu messenden Funktionsaufrufen zu platzieren. **Daher bietet es sich an, die Zeitstempel unmittelbar vor der Initialisierung und nach der Verschüsselung zu nehmen.**
3) Am Ende der *main*-Funktion ist die Auswertung der Zeitstempel bereits vorbereitet. Builden Sie die Software mit *make*.

4) Wenn die Software erfolgreich gebaut wurde, starten Sie das Ultra96-Board und verbinden es mit Ihrem Rechner. Im Makefile ist ein Target *install* vorbereitet. Mit

make install kann das *apu.elf* auf das Ultra96-Board kopiert werden und mit den Rechten für die Ausführung versehen werden.

5) Öffnen Sie eine Windows Powershell und verbinden Sie sich mit *ssh* auf das Board:
   > ssh ese@10.0.0.1
   Password for user ese: ese

6) Führen Sie das Programm *apu.elf* nun auf dem Board aus:
   > ./apu.elf

   Prüfen Sie mit dem Befehl *ls -l*, dass *apu.elf* nun "*rwx*" Permissions besitzt.
   ```
   -rwxr-xr-x 1 ese  ese  37848 Nov 19 18:00
   ```

7) Im File *main.c* sind verschieden grosse Nachrichten vorbereitet (16 Bytes, 64 Bytes und 240 Bytes). Diese Nachrichten werden mit dem von NIST zur Verfügung gestellten Test-Schlüssel ver- und wieder entschlüsselt. Die erwarteten verschlüsselten Strings sind ebenfalls bereitgestellt, um das Resultat automatisiert zu prüfen.
   Wiederholen Sie nun Ihre Zeitmessungen für die drei Nachrichtengrössen je mindestens fünf Mal und berechnen Sie jeweils die durchschnittliche Ausführungszeit. Protokollieren Sie diese in einer Tabelle und bewahren Sie die Werte auf bis Ende des Semesters. Sie werden diese Werte im letzten Praktikum benötigen, wenn Sie die verschiedenen Prozessoreinheiten miteinander vergleichen werden.

Ausführung in der Realtime Processing Unit (RPU), ARM Cortex-R5

Für die Ausführung einer Remote-Procedure sind mehrere Komponenten notwendig, die separat gebaut werden müssen. Das Main-Programm läuft auf dem ARM Cortex-A53 unter Linux. Die Remote-Procedure mit dem Verschlüsselungsalgorithmus läuft auf dem ARM Cortex-R5 unter einem Real-Time OS (RTOS), einem schlanken Betriebssystem speziell für Anwendungen mit harten Realtime-Anforderungen. Für den Verschlüsselungsalgorithmus gibt es zwar keine solchen Anforderungen. Wir nutzen den R5 Realtime-Prozessor aber, weil er sich von den A53-Cores unterscheidet und separat von diesen betrieben wird. Linux hat also keinen direkten Zugriff auf die Ressourcen des Cortex-R5-Cores.

Firmware mit dem Verschlüsselungsalgorithmus

1) Für das Bauen der Firmware für die Remote-Procedure wird ein zweiter Cross-Compiler benötigt. Wir ersparen Ihnen den Aufwand, diese zu installieren, und stellen Ihnen die Firmware *aes_rpu_rtos.elf* bereits kompiliert zur Verfügung.

2) Trotzdem untersuchen Sie den Sourcecode, um das Konzept von Remote-Procedure Calls zu verstehen: Öffnen Sie die Source-Datei "rmpsg_aes.c", welche die main-Funktion enthält. VS Code wird sich beklagen, dass die Headerfiles nicht gefunden werden, bietet dafür aber einen Quick Fix an, den Pfad zu den Headerfiles anzuge-ben. Setzen Sie folgende Pfade, um die Unterstützung der C/C++-Erweiterung zu

nutzen:
Include path
C:\<ihrPfad>**cross-compile**\freertos10_xilinx_psu_cortexr5_0\bspinclude\include
C:\ <ihrPfad>**cross-compile**\standalone_psu_cortexr5_0\bspinclude\include

3) Untersuchen Sie die Datenstruktur `aes_datatype` der remote procedure message (rpmsg). Wie viele Bytes an Daten werden zur remote procedure übertragen? Welche Daten sind enthalten?
   50 bytes + Text
   decrypt-flag, Schlüssel, Initialisierungsvektor, Textlänge und Text

   Welche Daten schickt die remote procedure an den caller zurück?
   ent- oder verschlüsselter Text und Textlänge

4) Die Callback-Funktion `rpmsg_endpoint_cb` ist der Teil, der den AES-Algorithmus effektiv ausführt/aufruft. Die Callback-Funktion wird beim Programm-Setup auf der A53-Seite registriert und steht ab dann für den Aufruf zur Verfügung.
   Wie wird unterschieden, ob gerade decryption oder encryption ausgeführt werden soll?
   "dec" flag

   Wie kann die remote procedure wieder entfernt werden?
   Shutdown-message senden

5) Kopieren Sie nun die Firmware ("aes_rpu_rtos.elf") wie gewohnt auf das Ultra96-Board.

6) Auf dem Ultra96-Board muss die Firmware noch ausführbar gemacht werden (chmod +x) und unter /lib/firmware/ abgelegt werden, damit der Kernel-Treiber die Firmware findet. Sie benötigen dafür sudo-Rechte:
   ```
   > sudo mv aes_rpu_rtos.elf /lib/firmware
   ```
   Damit ist der Remote-Teil der Software bereit.

A53-Software

Nun brauchen wir noch das Gegenstück auf der Linux-Seite, das den Treiber für die rpmsg lädt, die Firmware auf den R5 lädt und dort startet, um die remote procedure zur Verfügung zu stellen.

1) Damit Sie den Setup-Ablauf nachvollziehen können, führen Sie die folgenden Schritte einmal manuell auf dem ultra96-Board aus.

   a. Sie benötigen Superuser-Rechte dafür:
      ```
      > sudo su
      ```

b. Der Remote-Processor wird als Gerät im Device Tree dargestellt. Den Device Tree können Sie als virtuelles Filesystem erkunden. Wechseln Sie ins Verzeichnis des Remote-Processors:
```
> cd /sys/class/remoteproc/remoteproc0
> ls
coredump  device  firmware  name  power  recovery
state  subsystem  uevent
```

c. Im File "firmware" wird der Name der Firmware, die auf dem Remoteprozessor ausgeführt wird, definiert. Benutzen Sie die zur Verfügung gestellte Firmware aes_rpu_rtos.elf.
Mit dem Befehl `echo` und dem Umleitungsoperator `>` können Sie auf Linux ohne einen Editor in ein File schreiben:
```
> echo aes_rpu_rtos.elf > firmware
```
Über die Datei "state" kann die firmware geladen und gestartet werden:
```
> echo start > state
```

d. Nun wäre der Remote Processor bereit, den Kommunikationskanal aufzubauen, indem der rpmsg-Treiber im Master-Prozessor mit jenem auf dem Remote Processor verbunden werden. Dafür sind einige Schritte notwendig, deren Verständnis über unsere Ziele hinausgeht. Die Schritte können bei Interesse im Sourcefile aes_rpu.c der A53-Software nachvollzogen werden in der Funktion AES_RPU_xcrypt_buffer(…).

e. Stoppen Sie die Firmware im Remote Procesor wieder:
```
> echo stop > state
```
(Bei der Ausführung des RPU-Programms wird die Firmware durch das Programm geladen.)

2) Öffnen Sie den Ordner *RPU* in VS Code. Falls VS Code die Header-Dateien nicht findet, passen Sie auch hier in den VS-Code-Einstellungen den Pfad zu den Header-Dateien in den Sysroots an.

3) Die AES-Applikation soll den benötigten Treiber und die Firmware automatisch gemäss dem Ablauf in Punkt 1 laden. Ergänzen Sie die fehlenden Schritte in den Funktionen AES_RPU_start und AES_RPU_stop. Damit Sie die Software kompilieren können, müssen Sie die gegebenen Vorbereitungen für die Zeitmessung vorerst auskommentieren.
Um die benötigten Treiber zu laden, benötigt die Software Superuser-Rechte (die Sie bereits haben sollten, wenn Sie die vorangegangenen Schritte ausgeführt haben).
Wenn Sie die Software nun ausführen, sollten Sie die Remote Procedure nutzen können.

4) Überlegen Sie bei der Zeitmessung, an welchen Punkten in der Software Sie einen Timestamp nehmen möchten, um die Performance mit jener der AES-Software auf dem A53 vergleichen zu können. Schauen Sie sich dazu die Funktionen AES_RPU_start und AES_RPU_stop etwas genauer an. Wo entsteht eine grosse Verzögerung?

sleep(2) in stop

5) Das Setup der Remote Procedure macht unsere Software etwas komplexer. Beginnend bei der main-Funktion folgen Sie dem Programmablauf:
Was macht der Funktionsaufruf "AES_RPU_start()"?
setzt RPU auf

Schauen Sie auf dem ultra96-Board unter dem Pfad, der in AES_RPU_start() angegeben ist, ob die Firmware bereits installiert/gestartet ist.
(z.B. `> cat firmware`)

Die Funktion AES_RPU_encrypt_buffer() ruft die Remote Procedure auf. Sie finden zu Beginn der Funktion einige Schritte zur Vorbereitung der zu verschlüsselnden Daten. Wie lautet der effektive Aufruf? Würde es Sinn machen, die Zeitmessung hierhin zu legen?
write(fd_glob, aes_data, data_size)
nein ich denke nicht, dass kopieren von key etc ist auch teil des Ablaufs

6) Messen Sie an geeigneter Stelle die Performance der Remote Procedure auf dem R5 für die gleichen Fälle (16-Byte-, 64-Byte-, 240-Byte-Nachrichten) wie für die Ausführung auf dem A53 und vergleichen Sie die gemessenen Zeiten. Welche Schritte (Start, Encrypt, Stop) benötigen wie viel Zeit?

7) Diskutieren Sie die Messresultate mit dem Dozenten. Wo sehen Sie Optimierungsmöglichkeiten für die Performance?

Bewertungskriterien:

- Sie erhalten korrekte Testresultate bei der Ver- und Entschlüsselung, sowohl auf dem A53 als auch auf dem R5.
- Sie wählen sinnvolle Punkte im Programm für Ihre Zeitmessungen.
- Sie haben vertrauenswürdige Messwerte für die Ausführungszeiten des AES-Algorithmus für verschiedene Nachrichtengrössen auf den beiden Prozessoren protokolliert.
- Sie können erklären, welche Schritte notwendig sind, um einen Remote Procedure Call vorzubereiten und auszuführen.