Praktikum 4, Teil 2 – AES-CBC in der Configuration Security Unit (CSU)

Lernziele

- Sie verstehen die Abläufe, wie ein Socket geöffnet wird und wie man über einen Socket kommunizieren kann.
- Sie können mit einem Programm über einen Socket mit der CSU kommunizieren, um die Crypto-Funktionen zu nutzen.

Configuration Security Unit

Die Hersteller von Mikrocontrollern und System-on-Chip-Produkten treiben grossen Aufwand, um den Anwendern ihrer Produkte starke Instrumente zum Schutz vor unbefugtem Zugriff auf die Firmware und private Schlüssel zu geben. Im Xilinx Zynq UltraScale+ MPSoC des Ultra96-Boards steht zu diesem Zweck eine **Configuration Security Unit (CSU)** zur Verfügung, die ab Beginn des Boot-Vorgangs verfügbar ist und z.B. die Signatur eines Bootloaders überprüfen oder diesen entschlüsseln kann.

Die CSU besteht aus zwei Hauptteilen, dem Secure Processor Block (in Abbildung 1 links), der einen dreifach redundanten Prozessor mit eigenem, speziell geschütztem Speicher (RAM und ROM) enthält. Der rechte Teil stellt die Hardwareblöcke (ähnlich wie eine ALU) für verschiedene Krypto-Algorithmen (SHA, AES, RSA, …) sowie geschützten Speicher für private Schlüssel dar, die in einem Gerät abgelegt werden müssen. Solche Schlüssel werden zum Beispiel für Lizenzierungsanwendungen verwendet, bei denen die Signatur eines Lizenzfiles geprüft werden soll. Dieser Speicher kann nur wenige Male beschrieben werden, z.B. um einen privaten Schlüssel auszutauschen, wenn dieser geleakt wurde. Sind die Schreibzyklen aufgebraucht, ist das Gerät nicht mehr für Sicherheitszwecke nutzbar. Deshalb ist ein starker Schutz dieses Speichers besonders wichtig.

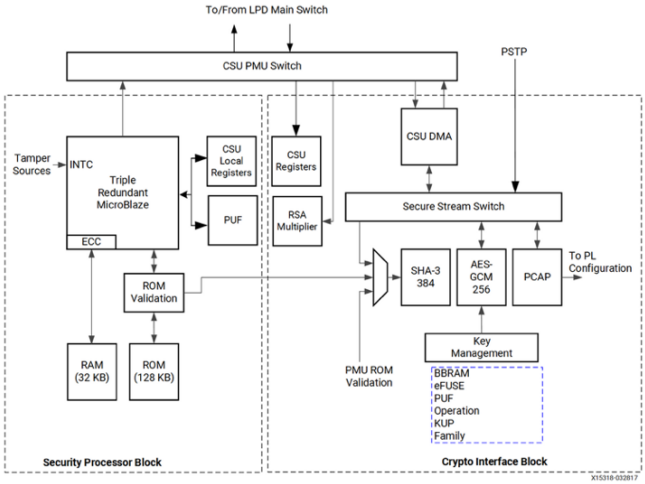


Abbildung 1 Architektur der Configuration and Security Unit. Linke Seite: Secure Processor, der für Secure-Boot-Anwendungen verwendet werden kann. Rechte Seite: Crypto Interface Block (CIB), der unter anderem AES, SHA und RSA unterstützt. Bild: AMD: Zynq UltraScale+ MPSoC Software Developer Guide (ug1137)

Socket

Die CSU ist ein schützenswerter Block, weshalb die Zugriffe auf die Status- und Kontrollregister nicht im Detail öffentlich dokumentiert werden. Stattdessen stellt das Betriebssystem eine Bibliothek zur Verfügung, über welche wir die CSU benutzen können. Diese Bibliothek stellt einen **Socket** zur Verfügung, auf den unsere Software verbinden kann.

Ein Socket ist grundsätzlich ein Verbindungspunkt für die Kommunikation zwischen zwei Knoten, häufig in einem Netzwerk, aber auch innerhalb eines Systems.

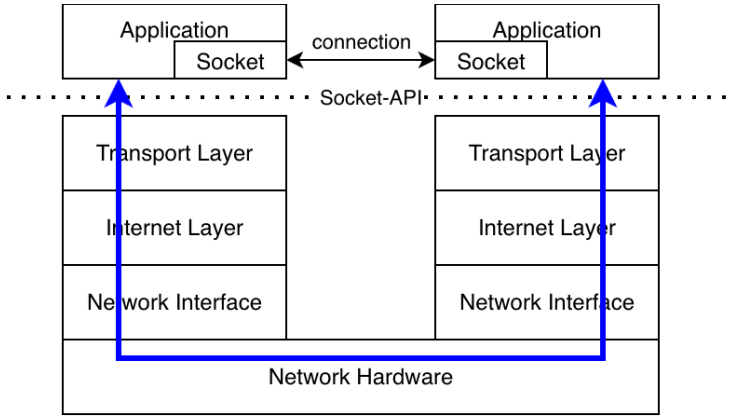


Abbildung 2 Schematische Darstellung einer Verbindung zwischen zwei Applikationen über einen Socket durch das Netzwerk. Die detaillierten Aspekte der Kommunikation (Adressierung, Paketisierung, …) werden durch die verschiedenen Layer abstrahiert. Die Applikation schreibt an und liest von einem Socket wie von einer Datei.

Beim Erzeugen eines Sockets wird dieser konfiguriert, unter anderem durch Angabe einer Adresse, unter welcher der Socket erreichbar ist. Die Übermittlung der Daten über das Netzwerk wird vom Transport Protocol erledigt. Somit ist der Socket ein Endpunkt eines Kommunikationskanals, der Protokoll- und Adressierungsdetails abstrahiert.

Um einen Socket zu verwenden, muss dieser korrekt konfiguriert werden. Dazu sind folgende Schritte nötig:

1) Socket erzeugen:

   ```
   sd = socket(AF_ALG, SOCK_SEQPACKET, 0);
   ```

   Der Return-Wert sd ist der socket descriptor, der wie ein File handle verwendet wird.

2) Anbindung an die Gegenseite:

   ```
   ret = bind(sd, (struct sockaddr *) &sa, sizeof(sa));
   ```

   Die `struct sa` enthält die Konfiguration des Structs, also welche Art Socket es sein soll (`salg_family`), die zu verwendenden Funktionen (`salg_type`) und den Namen des AES-Modus (`salg_name`).

   ```
   struct sockaddr_alg sa = {
     .salg_family = AF_ALG,
   ```

```
    .salg_type = "skcipher",
    .salg_name = "cbc(aes)"
};
```

3) Über die Socket Options wird der AES-Schlüssel übergeben:

```
ret = setsockopt(sd, SOL_ALG, ALG_SET_KEY, key,
AES_KEY_LENGTH);
```

Der Pointer `key` zeigt auf das uint8_t-Array mit dem Key. Das Macro `AES_KEY_LENGTH` muss die korrekte Schlüssellänge in Bytes enthalten.

4) Ein Filehandle für den Socket beziehen, über welches die Kommunikation abläuft:

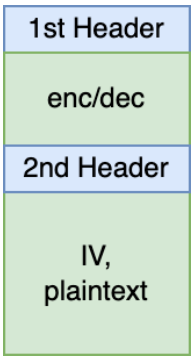
```
fd = accept(sd, NULL, 0);
```

Nach der Konfiguration kann der Socket verwendet werden, indem Meldungen gesendet und empfangen werden:

1) Senden:

```
ret = sendmsg(fd, &msg, 0);
```

Die Struct `msg` enthält alle nötigen Informationen für die CSU, also z.B. den Plaintext oder Ciphertext, dessen Länge sowie den Initialisierungsvektor und weitere Informationen. Die Details finden Sie im Bedarf im Headerfile.



2) Empfangen:

```
ret = read(fd, buf, len);
```

Der Pointer `buf` zeigt auf das uint8_t-Array für den verarbeiteten Text, der Pointer `len` bietet Platz für die Länge des Texts.

Aufgabenstellung

Die Source Files für den Versuch liegen im Git-Repository ESE_HS25_stud, im Verzeichnis P4. Sie benötigen den Cross-Compiler für den A53-Prozessor.

1. Die Kommunikation mit der CSU aus Linux funktioniert über einen Socket. Stellen Sie im File aes_csu.c das Öffnen, Konfigurieren und Verwenden des Sockets fertig. Konfigurieren Sie einen 256-Bit-Key und betreiben Sie die CSU im AES (CBC)-Modus.

2. Messen Sie die Ausführungszeiten für die Verschlüsselung und Entschlüsselung einer Nachricht. Wiederholen Sie die Messungen mit unterschiedlichen Längen der Nachricht. Um aussagekräftige Werte zu erhalten, messen Sie sowohl die Zeit inklusive des Aufbaus des Sockets als auch die reinen Verschlüsselungs-/Entschlüsselungszeiten.

Bewertungskriterien:

- Ihre Software öffnet einen Socket zur CSU und kommuniziert erfolgreich darüber, um Nachrichten zu ver-/entschlüsseln.
- Sie wählen sinnvolle Punkte im Programm für Ihre Zeitmessungen.
- Sie haben vertrauenswürdige Messwerte für die Ausführungszeiten des AES-Algorithmus in der CSU protokolliert.