ECOLE
POLYTECHNIQUE
DE BRUXELLES

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

Implementation of High-Level Cryptographic Protocols using a SoC Plateform

Mémoire présenté en vue de l'obtention du diplôme
d'Ingénieur Civil en informatique à finalité spécialisée

Quentin Delhaye

Directeur
Professeur Frédéric Robert

Superviseur
Sébastien Rabou (Barco Silex)

Service
BEAMS

Année académique
2014 - 2015

Abstract

by Quentin Delhaye, Arnaud Dumont, Camille Giaux, Pierre Lasbleis and Gauthier Roig; Université Libre de Bruxelles, 2011–2012.

Résumé

par Quentin Delhaye, Arnaud Dumont, Camille Giaux, Pierre Lasbleis et Gauthier Roig ; Université Libre de Bruxelles, 2011–2012.

Aknowledgements

Thanks a bunch of people here : - Frédéric Robert for his support - Sébastien Rabou for his insight - Batien Heneffe for his extensive help

Contents

Chapter 1

Introduction

1.1 Challenge

1.2 Network security

Chapter 2

Presentation

Here we talk about the protocols, the platform. Show The OS stack (kernel/user)

2.1 Experimental setup

The experimental environment is build around a standard x86 host and an ARM Cortex-A9 alongside an Altera Cyclone V FPGA as the target.

2.1.1 x86 host

OS Ubuntu 12.04 LTS, kernel 3.16

CPU Intel Core-i3 ... (two logical core out of four)

RAM 1.5GB DDR3

2.1.2 Altera Socrates SoCFPGA

OS Yocto project, kernel 3.14

CPU Dual core ARM Cortex-A9, 800MHz

RAM ...GB DDR3

FPGA Altera Cyclone V

Chapter 3

Implementation

Chapter 4

Conclusion

4.1 Future work

Although the present work presents some promissing results, the implementation can certainly be improved in several ways and some further experiments should be conducted.

Driver improvement The driver of the BA411E can be made less ressources hungry by improving the initialisation of the descriptors and their linking, but the gain would not be significant enough to justify the time investment at this point. A better alternative would be to avoid descriptors altogether by modifiying the interface with the IP so that it can use the scatterlist directly. We would then spare a lot of DMA mapping instruction and thus some precious cycles on the software side.

As we already remarked in **??**, the use cases involving IPSec were conducted using a previous revision of the driver still activelly polling the IP for its results. A better and cleaner way to proceed is to use interruption routines, as shown in **??**. However, the kernel does not support their current implementation and panics upon usage. If one were to be willing to spend the time replacing the active polling by clean asynchronous interruptions, he should be aware of the overhead imposed by an interruption. In some cases, when the operation is just a few clock cycle long for the IP, an active polling could still be the better way to go. A more thorough comparison of the mutual trade-off deserves some investigation, and as a starting point, the packet size could be treated as a branching point between the two solutions.

Registering public key verification with the crypto API As we saw in **??**, the driver is already capable of offloading a large portion of public key operations

to the IP, but only with very specific libraries at the time being – openSSL in our case. The next step is to register the very same operations with the crypto API so it can be used without having to rely on a custom openSSL engine.

Conditional offloading in cryptodev The figure **??** clearly shows a threshold on the packet size from which the hardware has a clear advantage, and below which the user mode software implementation is to go for. Using this tipping point, one could set a conditional branch as shown in listing 4.1

```
1 int some_function() {
2   if(packet_size < 1024*1024) {
3     callback_function();
4   }
5 }
```

Listing 4.1: cryptodev conditional offloading

He should however be aware that as the tipping point is around 1024kB, the performance for a network application should very close to those of a full software implementation, knowing that the ethernet frame size, the MTU, is set by default at 1500kB.

Appendix A

Cross-compilation

A.1 OpenSSL

A.2 OpenVPN

A.3 nginx

A.4 Strongswan