HARDWARE-ACCELERATED CRYPTOGRAPHY FOR SOFTWARE-DEFINED NETWORKS

Peter Cíbik
Doctoral Degree Programme (1st), FEEC BUT
E-mail: xcibik00@vutbr.cz

Supervised by: Lukáš Malina and Jan Hajný (BUT), Jakub Hrabovský (Netcope Technologies a.s.)
E-mail: malina@feec.vutbr.cz, hajny@feec.vutbr.cz, hrabovsky@netcope.com

Abstract: This paper presents a Software-Defined Network (SDN) cryptographic solution targeted on high-speed smart Network Interface Cards (NICs) with an FPGA chip. This solution provides a fast alternative method to develop network-oriented data processing cryptography applications for an accelerator. A high-level programming language – Programming Protocol-independent Packet Processor (P4) – is used to avoid a complex and time-consuming hardware development. The solution consists of two main parts: a library of mainly used cryptographic primitives written in VHSIC Hardware Description Language (VHDL) i.e. a symmetric cipher (AES-GCM-256), a hash function (SHA-3), a SHA-3-based Hash-based Message Authentication Code (HMAC), a digital signature scheme (EdDSA) and a post-quantum digital signature scheme (Dilithium), and a compiler P4/VHDL with the support for these cryptographic components as external objects of $P4_{16}$.

Keywords: Cryptography, Hardware acceleration, FPGA, VHDL, Software-Defined Networks, P4

1 INTRODUCTION

Nowadays trends of a centralized architecture in Information Communications Technology (ICT) solutions deploy central servers that receive messages from many end nodes. In order to process a huge amount of transactions and still keep high quality of services, many functions require a speed-up of their execution in the form of their offloading into hardware accelerators. The cryptography primitives, addressed in this paper, are a good example of such functions. The main problem of design and development of accelerator applications is their cost, primarily caused by the complexity and time and knowledge demand of the whole process.

This paper presents a solution for fast development of hardware-accelerated cryptographic applications. The fast development is reached by usage of a high-level P4 language for description of a packet-processing cryptographic application in the intelligible form. The P4 code is automatically compiled to a configuration file (*bitstream*) via the Netcope P4/VHDL compiler and directly loadable into an FPGA-based accelerator.

The solution consists of a standalone library of VHDL implementations of the basic cryptographic components, such as AES-GCM-256, SHA-3, HMAC, EdDSA and Dilithium, and an extended compiler P4/VHDL with the support for these cryptographic components as external objects of $P4_{16}$.

2 TECHNOLOGIES

- **VHDL** - Very High Speed Integrated Circuits (VHSIC) Hardware Description Language, the most popular Hardware Description Language (HDL) in Europe, is primarily used for an FPGA programming and simulation [1],
- **FPGA** - Field Programable Gate Array, shortly FPGA, is a programmable logic circuit, which

consists of three main elements: Input and Output Block (IOB), Configurable Logic Block (CLB) and programable horizontal and vertical interconection. Each of these elements is fully reconfigurable (programable) to achieve an expected functionality. High performance of FPGA chips is reached by the usage of Look-Up Tables (LUT) [1].

- **P4** - a high-level language designed for Programming Protocol-independent Packet Processors is mainly used in SDN's. Its key features are: reconfigurability, protocol independence, target independence and in the newest version ($P4_{16}$) external objects. The external objects in $P4_{16}$ bring support for target dependent features, in this solution the cryptographic external objects [2, 3].

3 CHOSEN PLATFORM FOR THE ACCELERATOR

In the presented solution, a high-speed smart NIC – NFB200G2QL, produced by *Netcope Technologies a.s.*, is used in the role of an accelerator. The smart NIC contains a Xilinx FPGA chip providing the transmition speed up to 200 Gb/s. The targeted FPGA frequency is 200 MHz. Primary parameters of the card are summarized in the Table 1.

Table 1: The parameters of NFB-200G2QL card [4]

Network interface	2×QSFP28
PCI Express	Gen 3 x16 + x16 (128 + 128 Gb/s)
FPGA chip	Xilinx Virtex UltraScale+ (xcvu7p-flvb2104-2-i)
On-board memory	3×72Mb QDRIIIe SRAM / 3×288Mb QDRIIIe SRAM

4 CRYPTOGRAPHIC COMPONENTS

Our cryptographic components library consists of five specific implementations. The Table 2 summarizes the resource utilization of each of the components.

HASH FUNCTION SHA-3

The Secure Hash Algorithm 3 (SHA-3) is based on the Keccak algorithm that uses a sponge construction. Our SHA-3 component implementation consists of two subcomponents: the **Padding Block** (the input message alignment) and the **Hash Block** (the Keccak function core). The input (a message) and output data width (a hash) are 512 bits [5, 7].

HMAC BASED ON SHA-3

The Hash-based Message Authentication Code (HMAC) component is implemented as a sequence of two separate SHA-3 instances, described in the previous section. Our HMAC component is designed according to the standard FIPS198-1, has the same input (a message) and output (a HMAC stamp) width – 512 bits. The input key (*KEY_0*) length is 576 bits [6]. The HMAC stamp is calculated like: `HMAC(K, message) = H((K0 XOR opad)||H((K0 XOR ipad)||message).`

DIGITAL SIGNATURE EDDSA

The digital signature algorithm EdDSA is a particular version of the ECDSA that considers curves in a twisted Edwards form. This change positively affects the performance, has smaller key requirements and the better resiliency to side-channel attacks, specifically *Ed25519*, which uses SHA-512 (SHA-2) and *Curve25519*. The implementation incorporates all main EdDSA phases: a public key generation, a signature creation and a signature verification [7].

SYMMETRIC CIPHER AES-GCM-256

The Advanced Encryption Standard (AES) in this implementation has a key length of 256 bits for a higher security level. It operates in the Galoise-Counter Mode (GCM), which provides authenticated encryption. A chosen symmetric cipher, key length and cipher mode are in line with current NIST recommendations. Our implementation of the AES-GCM-256 consists of these components: **Expansion** (used for a key expansion for per-round keys generation), **Encryption** (provides a message encryption), **Decryption** (provides a message decryption) and **GCM** (Galoise-Counter Mode operation mode wrapper) [7].

POST-QUANTUM DIGITAL SIGNATURE DILITHIUM

The Dilithium is a post-quantum (quantum-resistant) lattice-based digital signature algorithm and is one of a few candidates for the NIST Standard. The implementation consists of all underlying functions, such as SHAKE-128, SHAKE-256, ExpandA$_q$, ExpandMask$_q$, PowerToRound$_q$, MakeHint$_q$ and Number-Theoretic Transform (NTT) functions. All of them are integrated into the main Dilithium algorithms: key generation, signing and sign verification [8].

Table 2: The hardware resources utilization of the implemented cryptographic components after an RTL synthesis

Component	LUTs	Flip-Flops	Max. Frequency [MHz]
SHA-3			
SHA-3	19 341	4 829	237
HMAC			
HMAC	17 892	11 499	218
EdDSA [7]			
Public key generation	25 830	12 317	307
Signature	31 024	16 706	307
Verification	45 420	24 762	207
AES-GCM-256 [7]			
AES-256	28 389	2 596	222
AES-GCM-256	37 362	3 834	205
Dilithium [8]			
Key Generation	54 183	25 236	350
Signing	68 461	86 295	333
Verification	50 460	28 354	225

5 THE NETCOPE P4$_{16}$/VHDL COMPILER

The Netcope P4$_{16}$/VHDL (NP4) compiler is a central part of the proposed cryptography acceleration. The NP4 compiler translates a P4 source code to a VHDL code and maps it on an FPGA platform, as shown in the Figure 1. Integration of the support for cryptographic external objects to the NP4 compiler consist of several steps: the NP4 pipeline extension, a generation of templates for various wrappers and support components, and an instantiation of the components based on the content of source *.p4* files. The steps are described in detail in the following subsections.

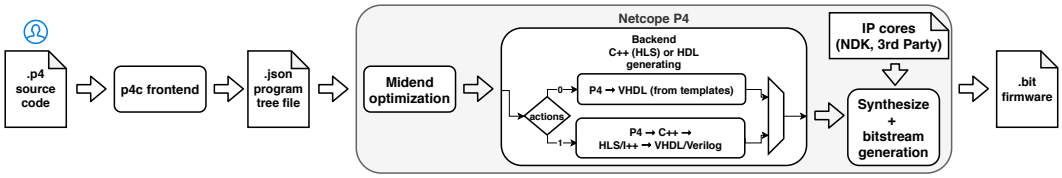


Figure 1: Compilation process

THE NP4 PIPELINE EXTENDED WITH C2 CONTROL BLOCK

The extended NP4 pipeline consists of the Parser (parses packet headers and a payload to separate fields), the Match+Action (implements all the tables and performs the actions, specified in the source code), the **C2 control block** (instantiates cryptographic external objects) and the Deparser (joins all modified headers and a payload back to a resulting packet), as shown in the Figure 2.

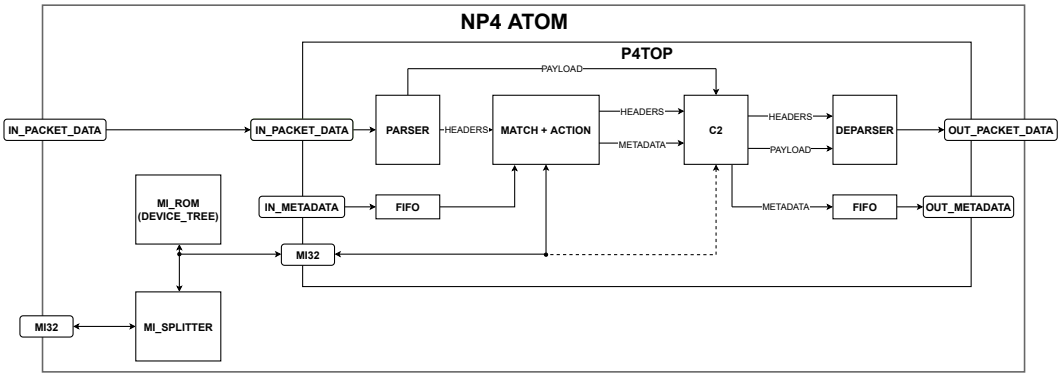
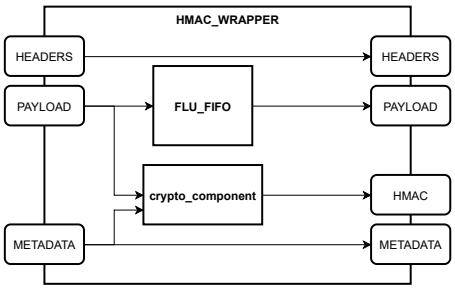


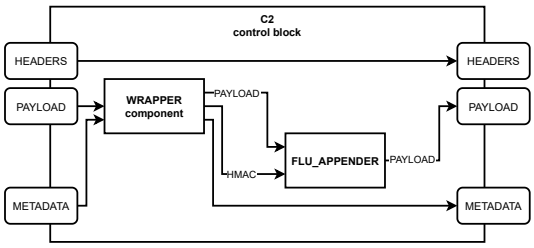
Figure 2: The architecture of the extended NP4 pipeline

WRAPPERS FOR CRYPTOGRAPHIC COMPONENTS

All cryptographic components, integrated into the NP4 pipeline, have their own wrappers to ensure the compatibility with the input and output interface of the pipeline. A general architecture of the wrapper is shown in the Figure 3a. A wrapper integration to the C2 control block is shown in the Figure 3b also with a FLU_APPENDER component, which appends computed results to the end of the packet payload.



(a) The architecture of wrapper components



(b) The architecture of the C2 control block with a wrapper component

Figure 3: Architecture schemes

THE NP4 COMPILER EXTENSION

The NP4 Compiler was extended to support cryptographic external objects with these functionalities: processing of a *.json* tree file extracted from a *.p4* source code, generation of wrappers for cryptographic external objects, instantiation of external objects to the C2 control block (the architecture generation) and other supporting functions.

REFERENCE USAGE OF CRYPTOGRAPHIC EXTERNAL OBJECTS IN A .P4 FILE

The NP4 provides a source code `externs.p4` with declaration of currently supported external objects shown in Listing 1.

Listing 1: The declaration of cryptographic external objects `externs.p4`

```
extern HMACOverPayload {
    HMACOverPayload();
    void compute();
}

extern HashOverPayload {
    HashOverPayload();
    void compute();
}
```

6 CONCLUSION

This work describes a hardware-accelerated cryptographic solution based on a smart NIC with an FPGA chip. It employs a high-level P4 language for a fast and flexible development of accelerated cryptographic applications. The VHDL component library contains implementations of AES-GCM-256, SHA-3, HMAC, EdDSA and Dilithium. The NP4 compiler was extended by a set of functions to generate supported cryptographic components and their wrappers, and to place them inside the new C2 control block in the pipeline based on the processed input *.p4* files. Two of these components are already implemented in the NP4 compiler as the cryptographic external objects. In future work, the NP4 compiler will be extended by the support for more cryptographic external objects.

ACKNOWLEDGEMENT

This work is supported by the Ministry of the Interior of the Czech Republic under grant VI20192022126.

REFERENCES

[1] PINKER, Jiří a Martin POUPA. *Číslicové systémy a jazyk VHDL*. Praha: BEN - technická literatura, 2006. ISBN 80-7300-198-5.

[2] BOSSHART, P.; DALY, D.; GIBB, G.; aj.: *P4: Programming Protocol-Independent Packet Processors. ACM SIGCOMM Computer Communication Review*, year 3, Num. 44, 2014: s. 87–95, ISSN 0146-4833

[3] *P4 16 Language Specification*. Version 1.2.0-rc. The P4 Language Consortium, 2019. Available from: `https://p4.org/p4-spec/docs/P4-16-v1.2.0.pdf`

[4] Netcope products: *Netcope FPGA boards (NFB)* [online]. [cit. 2019-11-11]. Available from: `https://www.netcope.com/en/products/fpga-boards`

[5] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: *The keccak sha-3 submission. Submission to NIST (Round 3)* 6 (7), 16 (2011)

[6] D. H. Krawczyk, M. Bellare a R. Canetti, *HMAC: Keyed-Hashing for Message Authenti-cation, RFC 2104, feb. 1997*. doi: 10.17487/RFC2104. url: `https://rfc-editor.org/rfc/rfc2104.txt`

[7] MALINA, Lukáš, David SMÉKAL, Sara RICCI, Peter CÍBIK and Jakub HRABOVSKÝ. *Hardware-Accelerated Cryptography for Software-Defined Networks with P4*. SECITC, 2020.

[8] RICCI, Sara, Lukáš MALINA, Petr JEDLIČKA, Jan HAJNÝ, Peter CÍBIK and Patrik DOBIÁŠ. *Implementing CRYSTALS-Dilithium Signature Scheme on FPGAs*