Simple AEAD Hardware Interface (SÆHI) in a SoC: Implementing an On-Chip Keyak/WhirlBob Coprocessor

Markku-Juhani O. Saarinen
Norwegian University of Science and Technology
mjos@item.ntnu.no

ABSTRACT

Simple AEAD Hardware Interface (SÆHI) is a hardware cryptographic interface aimed at CAESAR Authenticated Encryption with Associated Data (AEAD) algorithms. Cryptographic acceleration is typically achieved either with a coprocessor or via instruction set extensions. ISA modifications require re-engineering the CPU core, making the approach inapplicable outside the realm of open source processor cores. Our proposed hardware interface is a memory-mapped cryptographic coprocessor, implementable even on very low end FPGA evaluation platforms. Algorithms complying to SÆHI must also include C language API drivers that directly utilize the memory mapping in a "bare metal" fashion. This can also be accommodated on MMU systems.

Extended battery life and bandwidth resulting from dedicated cryptographic hardware is vital for currently dominant computing and communication devices: mobile phones, tablets, and Internet-of-Things (IoT) applications. We argue that these should be priority hardware optimization targets for AEAD algorithms with realistic payload profiles.

We demonstrate a fully integrated implementation of WhirlBob and Keyak AEADs on the FPGA fabric of Xilinx Zynq 7010. This low-cost System-on-Chip (SoC) also houses a dual-core Cortex-A9 CPU, closely matching the architecture of many embedded devices. The on-chip coprocessor is accessible from user space with a Linux kernel driver. An integration path exists all the way to end-user applications.

Categories and Subject Descriptors

D.4.7 [**Organization and Design**]: Real-time systems and embedded systems; D.4.6 [**Security and Protection**]: Cryptographic Controls; E.3 [**Data Encryption**]: Standards

Keywords

Cryptographic coprocessor, System-on-Chip, Keccak, Keyak, Whirlpool, WhirlBob, StriBob, CAESAR.

1. INTRODUCTION

An Authenticated Encryption with Associated Data (AEAD) algorithm provides both confidentiality and integrity protection in a single processing pass. AEADs are more effective alternatives to traditional encryption methods such as CBC block cipher mode [10] or stream ciphers such as RC4 [22], which had to be paired with suitable Message Authentication Codes (such as HMAC [17]) in transport protocols.

A typical AEAD algorithm $C = \text{Æ}(K, N, A, P)$ produces an authenticated ciphertext message C from following inputs:

- K Secret key.
- N Initialization vector, nonce, or message sequence number. Does not have to be secret. Integrity protected.
- A Authenticated data, only integrity protection. Transmitted in clear (or implicit to both parties).
- P Plaintext. Confidentiality and integrity protection.

The inverse algorithm $\text{Æ}^{-1}(K, N, A, C)$ returns either the original plaintext P or FAIL if the integrity of K, N, A, C is violated.

Advanced Encryption Standard (AES) in the Galois / Counter Mode (GCM) is an AEAD standardized and promoted by U.S. NIST and NSA for governmental, military, and public networks [7, 12, 15, 16, 28]. AES-GCM has rapidly replaced older bulk encryption algorithms; the majority of `https` connections to popular web services services such as facebook, gmail, and twitter are protected by AES-GCM at the time of writing (Q3/2014). In governmental and military use, AES-GCM is approved up to Top Secret level when appropriately implemented and used [20]. However, the security of AES-GCM is widely seen as brittle [5, 6, 21, 23, 29].

1.1 Motivation: CAESAR Hardware Testing

CAESAR (Competition for Authenticated Encryption: Security, Applicability, and Robustness) is a NIST - Sponsored international effort to find one or more algorithms to replace or complement AES-GCM [19].

The CAESAR competition runs from 2014 until 2017 and consists of multiple stages or "elimination rounds." By the March 2014 deadline, 57 candidate algorithms had been submitted [8]. Cryptographic weaknesses forced some of these to be withdrawn shortly thereafter. Selection for Round 2 must be made from almost 50 candidates.

An AEAD can be implemented in many ways:

1. **AEAD Block Cipher Modes**, like AES-GCM. About half (28) of the CAESAR first round candidates are based on block ciphers.
2. **Sponge-based AEADs.** There were eight proposals based on unkeyed Sponge permutations and theory, including Keyak [4], which is directly based on SHA3's Keccak-P sponge [18]. Another proposal derived from a hash function was StriBob/WhirlBob[1] [25, 26].
3. **Custom and ad-hoc designs.** There were various AEAD proposals that are were either entirely new or recycled some components from existing cipher designs without relying on theoretical frameworks.

Each one of these approaches represents different challenges and mechanisms for hardware implementation and performance evaluation.

Measurements should be made with realistic usage profiles that account for latency, key agility, concurrent streams, and other real-life requirements (Section 4). For mobile devices we propose the $e = \frac{R}{W}$ metric (Section 2.3). For network infrastructure and cloud data centers the focus is on terabits per dollar (total cost of ownership). For smart cards the focus is mostly in side channel resistance. RFID systems have additional latency and power considerations [27].

1.2 Hardware Reality: SoC Coprocessors

Confidentiality and integrity of transmitted and stored data is even more relevant to mobile devices than to "PC" systems. Mobile devices should be able to efficiently secure speech, streaming media, browsing, remote access, emails, and other messaging. There is an increased need to secure the private storage of these devices as they are more at risk of hostile examination after being seized or lost. We see mobile devices as the priority target for hardware optimization of bulk AEAD algorithms. Detailed profiling of hardware bottlenecks is therefore needed.

System-on-Chip (SoC) designs integrate all the necessary components of a computing application on a single chip. SoCs are dominant in mobile electronics such as (smart) phones and tablets, and are also found in mass-produced electronic appliances such as modems, routers, home media, and Internet-of-Things (IoT) applications.

Mobile devices are sensitive to power consumption and have restricted general computational resources. In SoC designs specific tasks are offloaded to coprocessors, primarily to minimize power consumption. Audio and video codecs, RF processing, display rendering, and 3D acceleration are typical functions handled by coprocessors present in many mobile architectures. SoC *design flow* typically employs use of packaged IP components such as CPU Cores (often ARM or MIPS ISA) and peripheral systems that are interfaced through on-chip buses such as those specified by ARM Advanced Microcontroller Bus Architecture [1] (AMBA). Our proposal fits into this industry-standard design model.

[1]WhirlBob is based on Whirlpool [2]. If StriBob is accepted to the second round of the CAESAR competition, the WhirlBob variant (used as an example in this paper) will be included as a tweak. The designs are closely related.

1.3 ISA Extensions vs. Coprocessors

As almost half of the first round CAESAR proposals are AES-based, we note that Intel (AES-NI) and ARM (ARMv8-A) have published optional ISA extensions that offer direct, specific support for the AES algorithm. These instructions compute either a single AES encryption or decryption round, or assist in computation of round keys. Generic ISA extensions have also been proposed for various cryptographic tasks [3].

Availability of these cryptographic instructions far from universal. Current mobile devices are based on 32/64-bit CPU cores that generally do not offer AES instructions. Lower-end embedded systems such as IoT appliances are unlikely to ever employ 128-bit or even 64-bit register sets, making such extensions unworkable. Non-AES proposals tend to have state sizes of 512 bits or more. Implementations of full rounds would be difficult on Sponge-based algorithms such as SHA3 and Keyak. The state size of SHA3 is $25 \times 64 = 1600$ bits, with full mixing occurring rapidly.

An instruction set extension requires modifications to the CPU core and compiler toolchains, while a coprocessor requires only a driver. On operating systems such as Linux (Android), the same hardware device drivers are commonly used across different CPU and even bus architectures. A coprocessor design is therefore vastly more portable.

Hardware acceleration should ideally be located where it is used. We will probably eventually see integration of AEAD cores within network interfaces and storage controllers.

2. INTERFACE

A simple C API was specified by the CAESAR secretariat for reference software implementations of the first round candidates [19]. The main emphasis was intended to be in portability and readability – the code is used to generate test vectors. Hardware implementation is required only for second round candidates (tentatively by April 15, 2015).

$N =$ `npub[CRYPTO_NPUBBYTES]`, $K =$ `k[CRYPTO_KEYBYTES]`, $P =$ `m[mlen]`, and $A =$ `ad[adlen]`. `*nsec` is a pointer to a "secret nonce" which was unused by most proposals.

```
int crypto_aead_encrypt(
  uint8_t *c, uint64_t *clen,
  const uint8_t *m, uint64_t mlen,
  const uint8_t *ad, uint64_t adlen,
  const uint8_t *nsec,
  const uint8_t *npub,
  const uint8_t *k
);
```

Decryption and integrity verification can be performed with `crypto_aead_decrypt()`, which has an equivalent interface.

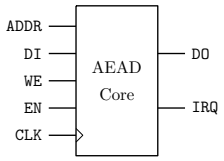
This API is not well suited for raw performance evaluation as it lacks a context pointer that could hold state information across calls. A context structure facilitates incremental operations on long messages and storage of reusable temporary variables such as round keys, avoiding duplication of effort. However, we assume that the two-function API will remain as the baseline portable interface of candidates.

Proposals such as Keyak [4] and StriBob's BLNK mode [24, 26] support MAC-and-continue "sessions" and other advanced protocol support features that are not available through this interface. Furthermore, Keyak utilizes the same sponge permutation as Keccak and therefore a hardware module can be easily adopted to support SHA3 too. The same is true for StriBob, which has the same fundamental transform as GOST R 34.11-2012 Streebog hash [11], and WhirlBob which utilizes the transform ISO/IEC 10118-3:2004 Whirlpool hash [2]. A hardware coprocessor can make these hashing functionalities available as well.

2.1 Proposed Hardware-Software Interface

Our cryptographic coprocessor has a simple, almost universal memory-mapped interface. The module or hardware PIN interface is the same as for generic single port RAM (with optional interrupt request line). Minimally this contains the following elements:

Signal	Dir	Purpose	Diagram
`ADDR`	In	Address	
`DI`	In	Data Write	
`WE`	In	Write enable	
`EN`	In	Enable/Select	
`CLK`	In	Clock	
`DO`	Out	Data Read	
`IRQ`	Out	Interrupt Req.	



HDL modules should be accompanied by baseline C API software interfaces that take care of necessary initialization and load/store operations. Peripheral's address is passed as an additional argument to `crypto_aead_encrypt()` and `crypto_aead_decrypt()` functions, which will now have the functionality of a "driver." The memory block, when defined the `volatile` keyword in C, will offer a simple and flexible interface to the hardware. The output of the hardware-assisted version must match with the reference software. The reference software driver should be entirely platform-independent and should not use any system calls.

Memory mapping mechanism depends on the target. On bare metal embedded systems without MMU this is trivial to do – one just assigns a physical address range for the coprocessor and implements appropriate chip selection logic. We first used this mechanism, but also interfaced our design through a more elaborate Linux kernel interface (Sect. 3).

It is up to hardware designers to specify the coprocessors' internal register layout, and to implement the necessary mechanisms of usage (for key and nonce set-up, padding, MAC verification) in the C driver. Implementation should work without `IRQ` – it is an entirely optional "look at me" flag.

It is possible to pipeline the operation so that data transfers to and from the hardware module can occur concurrently while a cryptographic operation is running. In continuous operation, one reads the result of the preceding operation and writes next plain/ciphertext on the coprocessor while a cryptographic transform is running. This requires some additional logic to implement the particularities of the Sponge/Block Cipher mode of operation in hardware.

Bus width (`DI` and `DO` sizes) should be 32, 64, or 128 bits. We suggest using large data paths – multiplexing is easy.

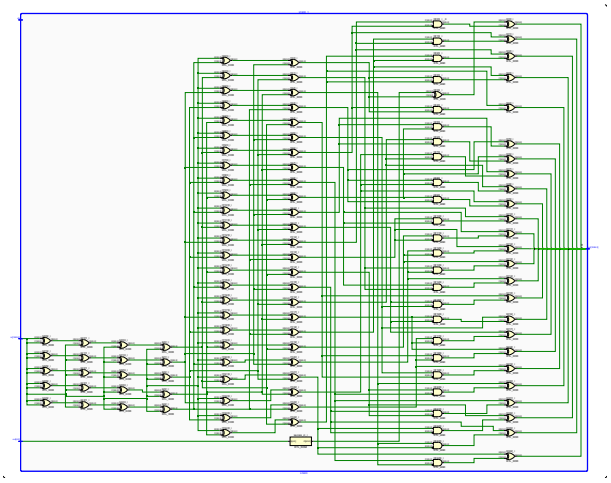


Figure 1: k1600.v, Keccak SHA3 and Keyak round (with 64-bit gates) and round constant generator. We suggest implementing individual rounds as stateless modules that only have combinatorial logic.

2.2 Discussion

We only specified a basic interface that should be useable even with the simplest FPGA development kits and low-end embedded SoCs. Actual data transfer speeds will not reach the numbers attainable via AMBA High-performance Bus [1] or other Direct Memory Access (DMA) mechanisms.

Our first generation AEAD modules utilize `0x400` bytes of address space, with one address pin simply assigned as a `STAT/CMD` indicator. Reading from `STAT/CMD` address range returns the status (and revision) of the coprocessor. Writing a word to this range is interpreted as a `GO` (start operation) instruction, a `RST` reset pulse, etc., depending on bits. 16 or 24 address pins should be sufficient for all implementations.

We urge implementors to separate the core cryptographic transform (round or subround function) as a separate HDL module (Figure 1). Such an module should be stateless and contain only combinatorial logic. This way double-steping and pipelining become easier to implement. For our algorithms, it was sufficient to interface the Sponge permutation rounds with `BLK_IN` (512/1600 wires), 4/5-bit round number input `RND`, and `BLK_OUT` (512/1600 wires) as the sole output.

2.3 On Measurements for Mobile Devices

We hope to interface multiple CAESAR candidates with various implementation testbenches for measurement of:

A **Area.** FPGA Slices or ASIC Gate Equivalents.

W **Power.** Power consumption (Watts).

R **Speed.** Ideal throughput (Bits / Second).

One key goal is to maximize $e = \frac{R}{W}$ in realistic usage. We note that doubling A for factor 2 parallelism will approximately double both R and W and e will remain constant. The same is true for doubling the clock frequency since power consumption is almost linearly dependent on clock frequency for most low-power (CMOS) circuits.

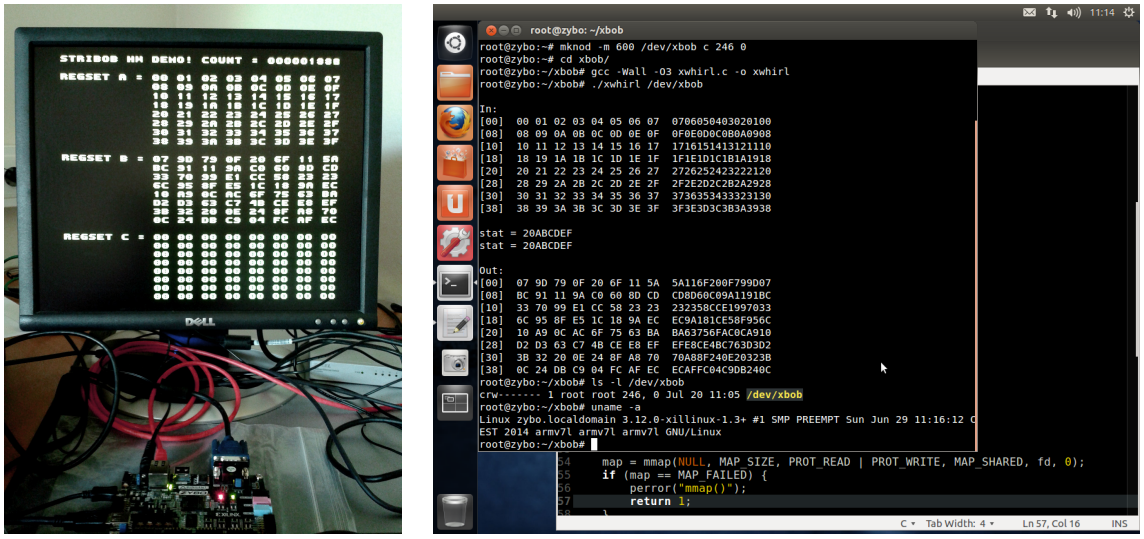


Figure 2: Left: A homemade VGA test module is driving the WhirlBob core directly in FPGA, while the CPU is inactive. Right: The Zynq CPU Cores are running a Linux kernel, with WhirlBob π, Keccak-f[1600] and other peripherals (display, networking, etc) in the FPGA Logic Fabric. The cryptographic module has been memory mapped with a Linux device file system driver at `/dev/xbob` and is accessible from user space.

3. EXPERIMENTAL IMPLEMENTATION

All hardware components were written in Verilog. We first implemented the basic WhirlBob π and Keccak-P functions and debugged those directly with a homemade 40×30 - character VGA test module (Figure 2, left). In this "test and demo" set-up the display module drives the cryptographic module directly and is controlled via switches on the board. CPU is unused; SÆHI also works in pure hardware projects.

We then wrote a Xillybus-lite based interface which allows the cryptographic module to be memory-mapped to Linux user space, emulating bare metal environment [30]. Xillinux is a full-featured single-chip port of Ubuntu Linux for Xilinx Zynq (Figure 2, right). The basic kernel device driver simply maps the peripheral address space to the device file system (we used `/dev/xbob`), from where it can be mapped to user process address space with `mmap(2)` system call.

Executing π can be as simple as (error handling omitted):

```
int fd, i;
volatile uint64_t *xbob;     // *volatile*

fd = open("/dev/xbob");      // devfs
xbob = mmap(NULL, 0x400, PROT_READ |
    PROT_WRITE, MAP_SHARED, fd, 0);

for (i = 0; i < 8; i++) {    // Input vec
    xbob[i] = 0x0706050403020100 +
        i * 0x0808080808080808;
}
xbob[0x40] = 1;              // GO pulse!
while (xbob[0x40] != 0)      // Wait
      ;

for (i = 0; i < 8; i++) {    // Result
    printf("%d␣%016lX\n", i, xbob[i]);
}
```

3.1 Implementation Metrics

Code lines in our rudimentary WhirlBob (StriBob) and Keyak reference implementations:

Component	WhirlBob	Keyak
Interface Verilog	99	114
Round Verilog	228	129
Driver C	60	60
API Interace C	261	*w.i.p.*

Post synthesis and route utilization within Artix-7 FPGA fabric of Xilinx Zynq 7010:

Logic	WhirlBob	Keyak
LUTs	3,795	4,574
Flip-Flops	1,060	3,237
MUXs	90	159
Other	1	2
Total logic	**4,946**	**7,972**

With one "extra" reloading cycle per block, the maximum throughput of these implementations is:

Parameter	WhirlBob	Keyak
Rounds	12	12
Cycles	13	13
Rate (bits)	256	1344
Speed (bit/clk)	**19.7**	**103.4**

Processing speeds are significantly slower when the Keccak core is used in SHA3 hashing mode [18]. Speed ranges from 23.0 (SHA3-512) to 47.5 (SHA3-224) bits/cycle as SHA3 uses 24 rounds of the round transform.

The 10-round Whirlpool hash standard [2] would be slightly faster than WhirlBob with this core, but would require roughly double the amount of logic due to its two-track design.

3.2 Implementation Resources

We used Digilent's Zybo Zynq 7010 board and Vivado 14.1 and 14.2 design tools with Ubuntu Linux 14.04. We used the free `arm-linux-gnueabi` toolchain for cross-compiling the Linux Kernel, drivers, and test applications to the target ARM7 / Cortex-A9 platform.

For testing, we adopted Xillinux kernel patches and FPGA Display and connectivity firmware from Xillybus Ltd [31] for the target device. These are open source and free for non-commercial use. However the final packaged, portable IP Cores and drivers are not dependent on any external (potentially proprietary or non-free) intellectual property.

The development environment – Vivado Design Edition for Linux academic license, hardware development kit, and cabling but excluding the generic PC components used – cost under $150 in June 2014.

The project took about one calendar month to complete by the author. The main tasks were installation of hardware and software (including kernel customization to target), writing the Verilog modules and rudimentary C drivers, testing, and writing this report.

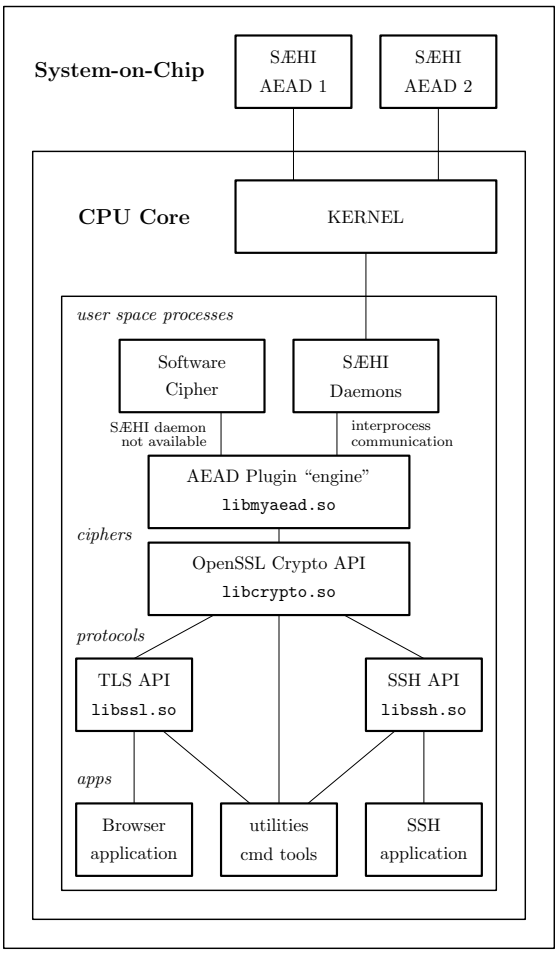


Figure 3: Integrating the cryptographic coprocessors with algorithm APIs, protocol libraries and applications via SÆHI daemon processes.

4. OPERATING SYSTEM INTEGRATION

Beyond the basic API, there is a path for integrating the cryptographic coprocessors with the Linux (Android) operating system and applications. For encrypted communications, protocol extensions will be required to accommodate new cipher suites. The most straightforward way to do this is to adopt the AES-GCM AEAD keying methods from TLS [9, 28], SSH [13], and IPSec [14] with appropriate experimental algorithm identifiers and key/nonce sizes.

A real-life generic SoC coprocessor architecture will have m CPU cores and m cryptographic coprocessors that may or may not have DMA capability. Furthermore some coprocessors may be outside the realm of CPU, being directly controlled by and contained within network interfaces or storage controllers ("acceleration where it is used").

Such a cryptographic resource can be managed by a dedicated scheduler system process (daemon). This process handles AEAD requests essentially in FIFO fashion. One may also integrate key management functions with this process.

Figure 3 shows how to integrate a new cipher to various levels of standard Linux (Android) cryptographic architecture via an engine plug-in for OpenSSL's `libcrypto` (or its derivatives).

5. CONCLUSIONS

We have described SÆHI, a simple memory-mapped testing interface primarily for hardware reference implementations of AEAD Algorithms in the CAESAR competition.

SÆHI HDL modules should be accompanied by C "driver software" that utilizes the cryptographic module as it is a bare metal coprocessor. Ideally the driver should be able to work on any CPU architecture with suitable datapath size. The external software API interface of this basic driver is almost equivalent to the API interface of first round software reference implementations to facilitate easy verification.

There are about 50 first round CAESAR candidates and hardware reference implementations are required for the second round. As expect about a dozen proposals in the second round, an uniform hardware testing interface is needed. We suggest that hardware evaluation should have special focus on most common computing and communication devices: mobile devices (cost and power efficiency), followed by smart cards (cost and side channel security), RFID (cost, power, area, latency, side channels), embedded Internet-of-Things devices (cost), network infrastructure and cloud data centers (terabits per dollar – total cost of ownership).

We have implemented WhirlBob and Keccak AEAD transforms in Verilog and proven them with Artix-7 FPGA Logic Fabric that is available on the Xilinx Zynq 7000-series System-on-Chip. This SoC also houses a dual-core Cortex-A9 and is able to run Linux (with Android), making it a realistic profiling tool for mobile devices, embedded, and IoT targets. The implementations utilize the SÆHI interface.

We described kernel and user space drivers that utilize the hardware. We also discussed integration of the coprocessors in the kernel, operating system, and application level.

Acknowledgments

This work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme.

6. REFERENCES

[1] ARM. AMBA Open Specifications. `www.arm.com/products/system-ip/amba`, 2014.

[2] P. S. L. M. Barreto and V. Rijmen. The Whirlpool hashing function. NESSIE Algorithm `www.larc.usp.br/~pbarreto/WhirlpoolPage.html`, 2000, Revised May 2003.

[3] S. Bartolini, R. Giorgi, and E. Martinelli. Instruction set extensions for cryptographic applications. In Ç. K. Koç, editor, *Cryptographic Engineering*, pages 191–233. Springer, 2009.

[4] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche, and R. V. Keer. CAESAR submission: Keyak v1. `competitions.cr.yp.to/round1/keyakv1.pdf`, March 2014.

[5] A. Biryukov, D. Khovratovich, and I. Nikolić. Distinguisher and related-key attack on the full AES-256. In S. Halevi, editor, *CRYPTO '09*, volume 5677 of *LNCS*, pages 231–249. Springer, 2009.

[6] A. Bogdanov, D. Khovratovich, and C. Rechberger. Biclique cryptanalysis of the full AES. In D. Lee and X. Wang, editors, *ASIACRYPT '11*, volume 7073 of *LNCS*, pages 344–371. Springer, 2011.

[7] K. Burgin and M. Peck. Suite B Profile for Internet Protocol Security (IPsec). IETF RFC 6380, October 2011.

[8] CAESAR. CAESAR submissions. `competitions.cr.yp.to/caesar-submissions.html`, March 2014.

[9] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. IETF RFC 5246, August 2008.

[10] M. Dworkin. Recommendation for block cipher modes of operation. NIST Special Publication 800-38A, December 2001.

[11] GOST. *Information Technology. Cryptographic Protection of Information, Hash Function*. Number R 34.11-2012 in GOST. Gosudarstvennyi Standard of Russian Federation, 2012. (In Russian).

[12] K. Igoe. Suite B Cryptographic Suites for Secure Shell (SSH). IETF RFC 6239, May 2011.

[13] K. Igoe and J. Solinas. AES Galois Counter Mode for the Secure Shell Transport Layer Protocol. IETF RFC 5647, August 2009.

[14] D. McGrew and J. Viega. The Use of Galois Message Authentication Code (GMAC) in IPsec ESP and AH. IETF RFC 4543, May 2006.

[15] NIST. Advanced Encryption Standard (AES). FIPS 197, 2001.

[16] NIST. Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC. NIST Special Publication 800-38D, 2007.

[17] NIST. The keyed-hash message authentication code (HMAC). FIPS 198-1, July 2008.

[18] NIST. DRAFT SHA-3 standard: Permutation-based hash and extendable-output functions. DRAFT FIPS 202, May 2014.

[19] NIST and D. Bernstein. CAESAR call for submissions. `competitions.cr.yp.to/caesar-call.html`, January 2014.

[20] NSA. Suite B Cryptography. `www.nsa.gov/ia/programs/suiteb_cryptography`, June 2014.

[21] G. Procter and C. Cid. On weak keys and forgery attacks against polynomial-based MAC schemes. In S. Moriai, editor, *FSE '13*, volume 8424 of *LNCS*, pages 287–304. Springer, 2013.

[22] R. Rivest. The RC4 encryption algorithm. Proprietary Specification, March 1992.

[23] M.-J. O. Saarinen. Cycling attacks on GCM, GHASH and other polynomial MACs and hashes. In A. Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 216–225. Springer, 2012.

[24] M.-J. O. Saarinen. Beyond modes: Building a secure record protocol from a cryptographic sponge permutation. In J. Benaloh, editor, *CT-RSA 2014*, volume 8366 of *LNCS*, pages 270–285. Springer, 2014.

[25] M.-J. O. Saarinen. Lighter, Faster, and Constant-Time: WHIRLBOB, the Whirlpool variant of STRIBOB. IACR ePrint 2014/501, `eprint.iacr.org/2014/501`, June 2014.

[26] M.-J. O. Saarinen. The STRIBOBr1 authenticated encryption algorithm. CAESAR First Round Candidate, `www.stribob.com`, March 2014.

[27] M.-J. O. Saarinen and D. Engels. A Do-It-All-Cipher for RFID: Design Requirements (Extended Abstract). DIAC 2012, 05-06 July 2012, Stockholm SE. IACR ePrint 2012/317, `eprint.iacr.org/2012/317`, June 2012.

[28] M. Salter and R. Housley. Suite B Profile for Transport Layer Security (TLS). IETF RFC 6460, January 2012.

[29] VCAT and NIST. NIST Cryptographic Standards and Guidelines Development Process: Report and Recommendations of the Visiting Committee on Advanced Technology of the National Institute of Standards and Technology, July 2014.

[30] Xillybus. The guide to xillybus lite, version 2.0. `xillybus.com/downloads/doc/xillybus_lite.pdf`, March 2014.

[31] Xillybus. Xillinux: A Linux distribution for Zedboard, ZyBo, MicroZed and SocKit. `xillybus.com/xillinux`, 2014.

Version 20140725010500.