This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2017.2741464, IEEE Design and Test

FASTEN: An FPGA-based Secure System for Big Data Processing

Boeui Hong, Han-Yee Kim, Minsu Kim, Lei Xu, Weidong Shi, and Taeweon Suh

Abstract— Security is a major concern in adopting cloud computing especially for processing private and/or sensitive data. FASTEN is an FPGA-based secure system for processing big data. It leverages the security features in modern FPGAs such as crypto engines and PUF. In FASTEN, the kernel functions for data processing are translated into hardware using a High Level Synthesis and are executed inside the FPGA fabric. Thus, the plaintext data and security keys are not exposed to main memory and secondary storage, keeping security-critical data from malicious insiders and outsiders. We have constructed a 24-node cluster using Zynq-7000 FPGA-SoCs, and conducted the performance evaluation with K-means, LWLR, and Sobel filter applications using Hadoop MapReduce on Linux. Experiment outcomes and security analysis reveal that the FASTEN provides both performance and security advantages over the system with Hadoop's native security support, at the expense of additional hardware.

Index Terms—FPGA, Security, Big Data, Cloud Computing, and Hadoop MapReduce

1 Introduction

Cloud computing platforms have emerged as ideal venues for data analytics services due to its processing time and cost advantages. They provide scalable, flexible and pay-per-use services without maintaining customer's own infrastructures. Even with its convenience, the customers with the privacy-sensitive data are reluctant to utilize the services from cloud service providers (CSPs) due to security concerns. For this reason, clouds adopt some security features to protect the sensitive information, where cryptography is typically applied for safeguarding the clients' data.

Nevertheless, the security threats could come from both internal and external sources. In a multi-tenant cloud environment, there is a potential risk of attacks from its neighbors because clients share the same physical infrastructure. For example, Ristenpart [1] showed that a client was able to get information of its neighbor virtual machine (VM) through sharing physical resources (e.g., the CPU's data caches). Then, such information was used to enable a key stroke timing attack or a side-channel attack for stealing the cryptographic key. In addition, even if the CSPs guarantee the data security from external sources, malicious insider (MI) always poses a potential threat as seen in the Edward Snowden case.

Responding to the security concerns, major processor companies are introducing security features in their products: ARM TrustZone, Intel Software Guard Extensions (SGX) and AMD's Secure Encrypted Virtualization (SEV). ARM's TrustZone supports two execution environments: secure world and normal world. The secure world is supposed to execute security sensitive applications while the normal world runs software that is not fully trusted. When the processor is in normal world, the access to the memory, cache and hardware devices is restricted. Intel SGX provides a way to protect memory access. Specifically, it introduces a set of new instructions to allocate private regions of memory for code and data, which are called enclave. Its concept is similar to TrustZone, and the enclave is accessible only by the authorized software. The AMD's SEV integrates the memory encryption capability with virtualization to provide encrypted VMs. It attempts to protect VMs from the other VMs, or even the hypervisor.

In this paper, we propose FASTEN: an FPGA-based system as a new possibility of providing the most security-tightened alternative for big data processing. The FASTEN was motivated by the fact that there are inevitable security vulnerabilities in systems with software-based big data processing. In FASTEN, the kernel code for data processing is ported into the FPGA fabric, and the execution is performed inside the hardware isolated from the host computer. Since the data processing and crypto-work takes place inside the hardware, there are no practically feasible paths, through which the plaintext data may be exposed. It also takes advantage of tamper resistant features of modern FPGAs to cope with side-channel attacks such as differential power analysis (DPA). These security features are combined with the reconfigurable characteristic of FPGAs to be utilized in the cloud. We have constructed a 24-node Linux system using Xilinx Zynq-7000 devices. Experiment outcomes with MapReduce applications show that FASTEN provides not only the tightened security but the performance advantage over the latest Hadoop release for security, at the expense of additional hardware.

2 Related Works

There are several studies on accelerating workload and/or enhancing security in cloud environment. Byma [2] proposed a hardware and software framework to virtualize FPGAs as cloud resources on OpenStack, which is called virtualized FPGA resources (VFR). VFRs are dynamically allocated to cloud users. It allows users to take advantage of the power of reconfigurable hardware, while maintaining the scalability and flexibility of cloud computing. Zerfos [3] proposed a secure distributed file system (SDFS) for Hadoop to provide data-at-rest protection. SDFS is able to read and write secured input

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2017.2741464, IEEE Design and Test

Table 1. Encryption and authentication supports of modern FPGA-SoCs.

FPGA vendors	Xilinx		Altera		Microsemi
Devices	*Spartan-7, Artix-7, Kintex-7, Virtex-7*	*Kintex Ultrascale, Virtex Ultrascale*	*Arria 10*	*Stratix 10*	*SmartFusion2*
Symmetric Encryption	AES-CBC 256	AES-GCM 256	AES-CTR 256	AES-GCM 256	AES-128/256 (ECB, CTR, CBC, OFB mode)
Asymmetric Encryption	RSA	RSA	RSA, ECC	RSA, ECC	RSA, ECC
Integrity/ Authenticity Hash Function	HMAC-SHA-256	AES-GCM 256	SHA-256/384 ECDSA	HMAC-SHA-256 SHA-256/384 ECDSA	HMAC-SHA-256 SHA-256 ECDSA
Key Storage	BBRAM eFUSE	BBRAM eFUSE	BBRAM eFUSE	BBRAM eFUSE SRAM-PUF	FLASH SRAM-PUF
Side-channel Protection	N/A	Yes	Yes	Yes	Yes

data on behalf of MapReduce applications using keys that are distributed and controlled by customers. Schuster [4] proposed VC3 system using Intel SGX. It allows users to run distributed MapReduce computations in the cloud while keeping their code and data secret. The VC3 ensures that the MapReduce applications are confidentially and verifiably executed under untrusted cloud environments. Eguro and Venkatesan [5] proposed a system similar to our work in a way that FPGAs are used to run the sensitive portions of applications. They adopted cryptography for the FPGA configuration. However, the proposal relies on the Trusted Authority for key management, whom was simply assumed in the paper. In addition, a prototype system was not constructed for evaluation. Our paper differs from [5] in terms of the key management and a full system demonstration. Our prior work [6] introduced an FPGA-based MapReduce framework for security. This paper expands our prior work to propose a security-tightened complete system, constructing a working cluster system with 24 slaves, and demonstrating the performance and security advantages with MapReduce applications.

3 Security Features of Modern FPGA-SoCs

FPGA devices are configured by bitstreams. Although the bitstream structure is usually quite complex and proprietary to the vendors, it is still possible to reverse-engineer or modify bitstreams maliciously. Thus, FPGA vendors provide bitstream encryption and authentication capabilities for confidentiality, integrity and Intellectual Property (IP) protection. Table 1 compares security features of the latest devices from the three major FPGA vendors: Xilinx, Altera and Microsemi.

3.1 Bitstream Protection

As shown in Table 1, all the devices provide AES with different modes and key sizes to protect IPs in bitstream. All the modes provide a similar level of security except ECB. The encrypted bitstream is either directly downloaded to an FPGA device, or stored in non-volatile flash storage such as SD-card or QSPI on board. The encrypted bitstream is then decrypted by the hardware AES engine inside FPGA for configuration.

For the bitstream integrity, all the devices employ HMAC or digital signature mechanisms listed in Table 1. Xilinx 7-series supports HMAC-SHA-256 for the integrity verification, which generates the bitstream authentication code with a HMAC key. Then, both the authentication code and HMAC key are loaded to FPGA as part of the encrypted bitstream. After the bitstream decryption, the HMAC engine inside the device generates bitstream authentication code with the HMAC key and compares it against the received one. If these two codes are equivalent, the FPGA believes the loaded bitstream is not tampered.

Altera's Arria 10 supports the integrity verification through SHA-256, HMAC and elliptic curve signature (ECDSA). SHA-256 is applied to the bitstream and the hash value is attached to the encrypted bitstream. HMAC-SHA-256 and ECDSA are also available for the integrity verification, but they require extra keys. HMAC requires the symmetric key shared between a sender and a receiver. The ECDSA uses a private-public key pair. The private key is used by the client to generate the bitstream signature. The public key is embedded inside the device for checking the signature to determine whether the bitstream is generated by a trusted party.

For Microsemi's Smartfusion2 and Altera's Stratix-10, SHA, HMAC and ECDSA hardware engines are available to be selected for the bitstream integrity.

3.2 Key Storage and Management

All the devices in Table 1 provide volatile and non-volatile storages for keys: Battery-backed RAM (BBRAM), eFUSE and flash storage. Both Xilinx and Altera support BBRAM and eFUSE. BBRAM is a type of volatile memory that is writable but not externally readable. In Xilinx, if an attempt to extract the key from BBRAM is detected, the FPGA configuration and key in BBRAM is erased. eFUSE is a non-volatile one-time programmable storage that does not need an external battery. Once the key is stored in eFUSE, it cannot be cleared. Different vendors have their own proprietary mechanism to protect information stored in eFUSE but it is less secure than BBRAM in general. Microsemi supports the flash storage as a key memory, where the key is stored in encrypted form. Microsemi provides temporal or permanent lock bits to prevent any further changes to the stored key. The physical tamper detection nullifies the FPGA configuration or even makes the board unusable.

Some keys should be programmed by the client using the vendor's proprietary CAD tool via JTAG. For Altera and Xilinx devices, it is mandatory to program the key at a trusted place. However, certain FPGA models from Microsemi provide a mechanism that allows the client to set up secret keys remotely. For example, Smartfusion2 has a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2017.2741464, IEEE Design and Test

Physical Unclonable Function (PUF) that generates a device-specific ECC private key based on the process variation in the SRAM fabrication. This private key never leaves the device, and the Smartfusion2 generates a corresponding public key to be released to the public. With this public/private key pair, the client transmits a symmetric key to FPGA device through ECDH protocol [7].

4 FASTEN Architecture and Security Analysis

This section describes the threat model and FASTEN architecture.

4.1 Threat Model

Three parties are involved in FASTEN: Client (or Customer), FPGA vendor, and Cloud Service Provider (CSP). The client has private and security-sensitive data and is totally trusted. The FPGA vendor supplies FPGA boards to CSP, and we assume that the vendor is trusted with a careful monitoring of the device manufacturing process. A private key from PUF is embedded inside each FPGA in the manufacturing phase of the devices, such as Smartfusion2, and it is not disclosed to any third party. The corresponding public key is released to the client. The CSP is not trusted. The data processing work is delegated to the CSP, but there may be some malicious outsiders and insiders including rogue system administrator. The security goal of FASTEN is to protect the privacy and security of the client's data while allowing the CSP to process the data.

4.2 FASTEN Architecture

FASTEN setup and operation are divided into three phases: ① FPGA vendor: private-public key pair generation; ② Client: bitstream generation; and ③ CSP: FPGA configuration, data processing, and output generation.

Each party's responsibility is depicted in Figure 1 with an example, and is elaborated as follows.

FPGA vendor's responsibility: FPGA vendors are responsible for creating a database of FPGA identifications (IDs) and their corresponding public keys, for the FPGAs deployed in CSP. FPGA should support PUF inside its fabric such as SRAM-PUF [8] in the latest FPGAs. For a FPGA $device_i$, a PUF inside the FPGA internally generates a unique private key sk_i. The corresponding public key pk_i is generated inside the FPGA using the crypto engine. The internally generated private key is not accessible from the external world, as implemented in Microsemi's FPGAs. The database of public keys and FPGA IDs is then shared with the client. Besides managing the key information, the FPGA vendors also provide proprietary CAD tools (such as Xilinx's Vivado High Level Synthesis (HLS)) for converting applications written in high level language to hardware.

Client's responsibility: Clients are responsible for generating own FPGA bitstreams. The client first communicates with CSP for available FPGA boards for data processing. Assuming that k FPGAs are assigned to the client, $image_{0\ldots(k-1)}$ will be generated for $device_{0\ldots(k-1)}$. Each image is composed of the encrypted hardware bitstream, optional authentication code for the bitstream, and encrypted bitstream key, as shown in Figure 1. To deliver the bitstream encryption key securely to the FPGAs on CSP, the client should use the public key encryption such as RSA or ECC, where the public keys ($pk_{0\ldots(k-1)}$) are obtained from the database.

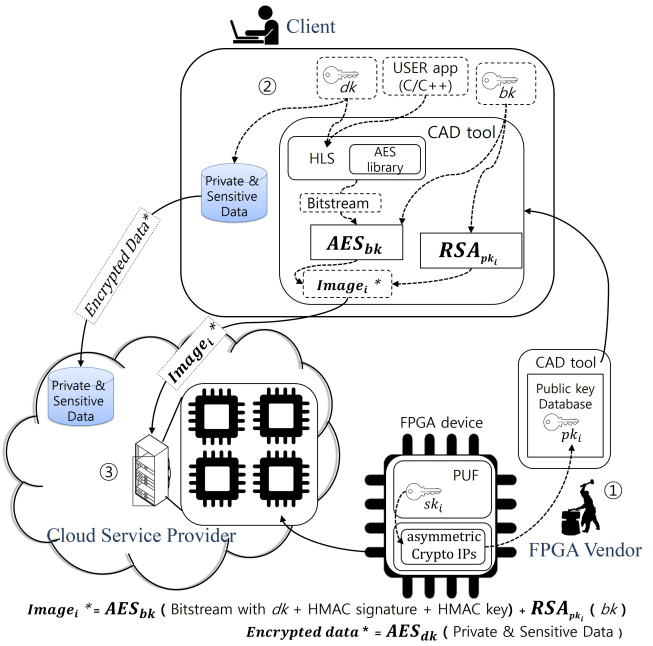


Figure 1. A FASTEN example. Three parties are involved for processing client's private & sensitive data securely.

For the bitstream generation, the client utilizes the CAD tool from FPGA vendors. The data decryption key (dk) is embedded in the application code for the data handling. The generated bitstream is encrypted with bitstream encryption key (bk). Finally, the client encrypts bk with ($pk_{0\ldots(k-1)}$). For the bitstream integrity, the client may also use HMAC or digital signature to add a tag to the bitstreams.

CSP's responsibility: The CSP is responsible for deploying images from client to appropriate FPGA boards, where the preselected k FPGAs will be configured. The configuration process may also verify the authenticity and integrity of the bitstream. It is exactly the reverse of the image creation process. For the device i, it evokes the internal PUF to regenerate the private key sk_i to decrypt the ciphertext of bk. Then, bk is used for the bitstream decryption. The authentication code is compared against the hash value of bitstream, which is internally generated by a hashing engine (such as SHA or HMAC).

The CSP is also responsible for compiling and executing the pure software portion of the data processing, which interacts with the hardware kernel downloaded to FPGAs. The software portion simply orchestrates encrypted data distribution to appropriate FPGAs for actual processing. For example, we used the Hadoop MapReduce framework. The data processing kernels are converted to hardware, while the tasks for managing slaves and distributing workload remain in software.

Finally, the CSP is responsible for handing over the en-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2017.2741464, IEEE Design and Test

crypted outcome to the client. The data processing outcome is encrypted using *dk* inside FPGA and returned to the client.

4.3 Security Analysis

The conventional software-based attacks such as memory dumping for plaintext data and/or key extraction are not feasible against FASTEN. It is because data is always present in encrypted form in memory, and the kernel processing takes place inside FPGAs. However, potential security threats could come from three ways: The first one is attempting to read a private key (sk_i) embedded inside the FPGA. The second one is attempts to read-back the configured bitstream from FPGA. The last one is side channel attacks (SCAs) against AES engines.

The private key (sk_i) is the root of trust in FASTEN. Its leakage leads to severe consequences on data security as well as image integrity. As discussed in Section 3.2, it is guaranteed to be safe inside the FPGA through companies' proprietary technologies. For example, Xilinx nullifies the key storage upon a tamper detection. Altera utilizes the scrambling key where the key is stored in altered form using internal logic.

Reading-back the configured FPGA can reveal the hardware design and data since the key (*dk*) and hardware logic is contained in the configured fabric. The read-back capability is typically implemented in the vendor's CAD tools using debugging ports (e.g. JTAG) for verifying the configured bitstream. The CAD tools also provide an option to disable the read-back feature. Thus, the read-back attempt can be blocked simply by tweaking the CAD tools by FPGA vendors or by disabling the JTAG connection.

The SCA toward the crypto IPs is another threat regarding the key confidentiality. The SCA takes advantage of the circuit's physical characteristics such as power consumption, electromagnetic measurement or even sound to reconstruct the secret key. In FASTEN, there are two AES engines used for processing: a fused AES for the bitstream decryption and a soft AES for data protection. For the fused AES, the DPA-resistant technology in recent FPGAs could block SCAs. For instance, Xilinx Ultrascale has the facility to block the decryption of invalid bitstreams [9], which is tried for measuring power characterics in SCA. In case of the soft AES, to the best of our knowledge, there have not been published attacks against the configured soft AES mixed with considerable logic in FPGAs. Nevertheless, DPA resistant AES core [10] can be used in the hardware design for prevention.

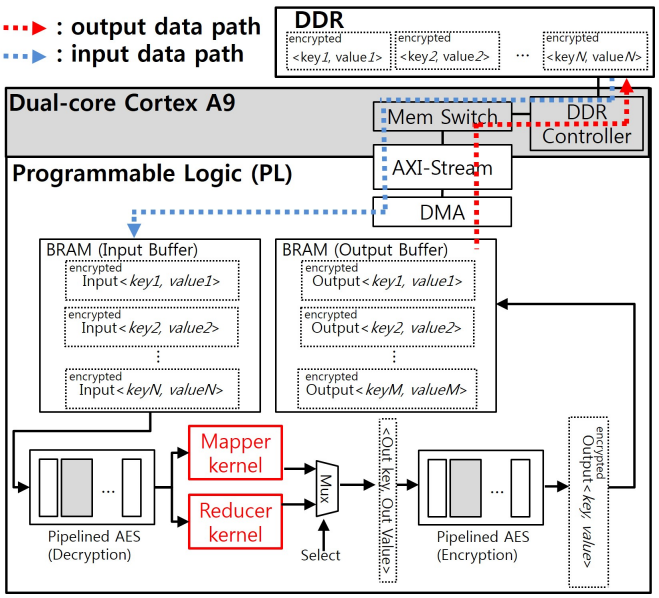
5 Experiments and Evaluations

We constructed a Hadoop MapReduce cluster as a proof of concept platform. This section details the experiment infrastructure and performance evaluation.

5.1 Cluster Setup

Figure 2 (a) is a picture of the constructed 24-node cluster, which has two x86 machines with Core i7 Haswell and twenty-four ZedBoards. ZedBoard is mounted with a Zynq-7000 (XC7Z02), where dual-core Cortex-A9s are integrated with programmable logic (PL). The operating

(a) Constructed cluster with 24 Zedboards.



(b) Detailed schematic diagram of execution data flow on a slave node's FPGA; Mapper and reducer kernel functions are downloaded to PL. Input and output buffers contain only the encrypted data.

Figure 2. Constructed cluster picture and hardware schematic diagram inside FPGA of a slave node.

clock of Cortex-A9s and PL is 667MHz and 100MHz, respectively. The Zedboard has a 512MB DDR as main memory and 32GB SD card as a secondary storage. Each Zedboard represents one slave node. The 24 slave nodes are connected via a switch. We installed Hadoop 2.7.2 on top of Ubuntu 14.04.

5.2 Hadoop MapReduce Implementation

The execution of the Hadoop MapReduce applications is divided into two parts: *mapper* and *reducer*. For a data chunk, a *mapper* in slave node outputs <*key*, *value*> pairs. These pairs from multiple *mappers* are merged and sent to the corresponding *reducers*, based on the key value. This intermediate stage is called *shuffle*. Finally, the *reducers* generate the output.

For FASTEN, we implemented the software and hardware portions of MapReduce applications, and the Linux device driver. The software portion of MapReduce applications is only responsible for passing the encrypted data to the *mappers* or *reducers* implemented inside PL. It also

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2017.2741464, IEEE Design and Test

takes the final output from *reducers*, which is also in encrypted form. For using a low-level *I/O* library in Java-based Hadoop MapReduce, we used the Hadoop Streaming. It allows the use of C language. The data processing takes place inside the FPGA fabric, as depicted in red boxes of Figure 2 (b). The hardware portion is divided into two sections: application-specific section and common section; The application-specific section contains both *mapper* and *reducer* kernel functions, and either of these two functions are selected via memory-mapped register. The common section includes communication modules and AES security modules. The communication modules are for passing data between DDR and FPGA. The security modules are to decrypt data for processing and to encrypt the processed data. This way, input and output buffers in Figure 2 (b) always contain the encrypted data. Thus, even the operating system is not able to access the key or the plaintext.

A page-sized data (4KB) is passed from DDR to an input buffer and from an output buffer to DDR via DMA through the communication module. For performance, we exploited parallelism in the AES code and reutilized the AES round keys by storing them into registers; Some loops in the AES algorithm are executed independently. Thus, we applied the loop unrolling technique with the *LOOP_UNROLL* Vivado directive. The *PIPELINE* and/or *ARRAY_PARTITION* directives were applied as well to improve performance. The round keys, which are changed only when the AES key is different, are stored after decrypting the first block and reused without recomputation. Compared to a naive HLS translation of C++ AES code, the performance optimization reduces the execution clock cycles from 89,698 to 10,642 (1/8x) and from 61,212 to 8,718 (1/7x) for 4KB data encryption and decryption, respectively. However, the hardware resource utilization was increased from 11 to 44 (4x) for BRAMs, from 3,264 to 7,070 (2.1x) for LUTs, and from 11,770 to 21,850 (1.8x) for FFs, as shown in Table 2.

5.3 Applications

For the performance evaluation, we have used three applications: K-means, Locally weighted linear regression (LWLR), and Sobel filter. The implementation of K-means and LWLR algorithm is based on [11], where the *mapper* and *reducer* division is detailed. The randomly generated floating point numbers were used as input for the 3-dimensional space. Sobel filter uses a 3x3 kernel, where the original image is convolved with the kernel to compute the derivative approximations. Sobel filter was implemented with mapper-only operations because of the application characteristic. In the Sobel filter, each mapper filters a portion of the image, and the edge-detected image is generated by combining the mappers' outputs. We used the face images from CK+ [12], which were preprocessed to display grayscale images with 256 steps.

The applications in C++ were translated to the HDL code using Vivado HLS (2015.4). For the hardware translation of application kernels, we assumed that user is not familiar with hardware design with HLS. Therefore, application portion written in software is directly translated into hardware without any HLS directives. The performance gain using hardware accelerators on our cluster is discussed in Section 5.4.1. Table 2 shows the hardware resource utilizations. Sobel filter requires many BRAMs because it needs to store a portion of an image and process for edge detection. For processing a 4KB data, the comsumed clock cycles of each application kernel are 16,897 for Sobel filter, 8,192 for LWLR and 12,592 for K-means.

Table 2. Resource utilization and power consumption in programmable logic (PL) of Zynq-7000 (ZC702).

Resources Utilization		Hardware Components in FPGA Used # (Utilizaton %)			
		#BRAMs (18Kbit)	*#DSPs (48E)*	*#FFs*	*#LUTs*
Application Kernels	**K-means kernel**	2 (1%)	4 (2%)	1290 (1%)	1532 (3%)
	LWLR kernel	2 (1%)	4 (2%)	1918 (2%)	2942 (6%)
	Sobel filter kernel	18 (3%)	4 (2%)	3356 (3%)	1691 (3%)
Common Components	**AXI-DMA**	2 (~0%)	0 (0%)	1821 (1%)	1277 (2%)
	AES	44 (16%)	8 (3%)	11700 (11%)	7070 (13%)
	AXI4 memory Interconnection	0 (0%)	0 (0%)	1163 (1%)	996 (~0%)
Power Consumption		**Security Modules**			
		w/o AES (baseline)	*w/ AES (FASTEN)*	*w/ AES ($FASTEN_{dup}$)*	
Application Kernels	**K-means kernel**	1.772W	1.892W	2.13W	
	LWLR kernel	1.845W	1.984W	2.348W	
	Sobel filter kernel	1.846W	2.02W	2.449W	

5.4 Evaluations

For the performance evaluation, we have experimented with software or hardware implementation of the application kernel section and security section. The software implementation of the security section in this paper means the Hadoop's native security support: transparent encryption (TE) [13] and encrypted shuffle (ES) [13]. The TE is for protecting data stored in distributed file system, and the ES is to hide the plaintexts during the shuffle stage. The hardware implementation of the security section is the AES hardware modules embedded inside FPGA, as in FASTEN. In Figure 3, *Software* means the software implementation of applications without security support. *$Hadoop_{TE\_ES}$* implements the application kernel with hardware, but utilizes the software implementation for the security section. *FASTEN* implements both application kernel and security sections in hardware. *$FASTEN_{dup}$* has the same configuration as *FASTEN* with duplicated hardware modules for performance: AXI-DMA, AXI4 memory interconnection, AES module and application kernel. To utilize the duplicated hardware, the code was parallelized with Pthread. *Baseline* is the hardware implementation of application kernels without the security support. The experiments were performed with three different input data sizes: 10GB, 20GB, and 40GB.

5.4.1 Hardware Accelerator Effect

Figure 3 shows the normalized execution times of applications over the *Baseline*. The hardware acceleration impact is measured by comparing the execution times of *Software*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2017.2741464, IEEE Design and Test

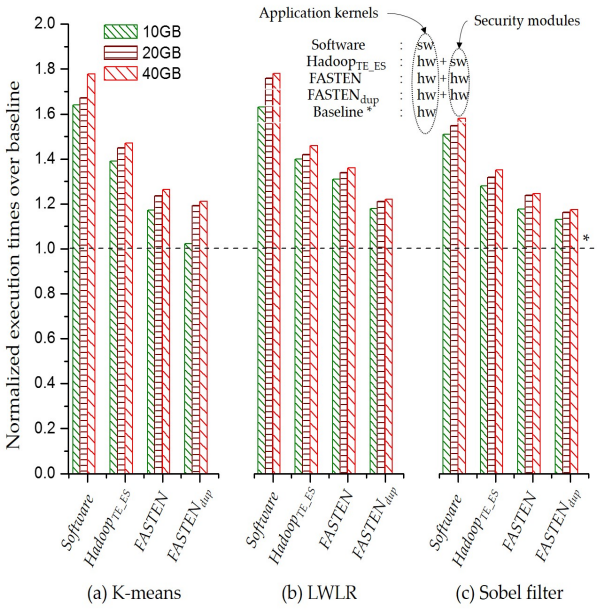


Figure 3. Normalized execution times of applications over baseline.

and *Baseline*. With 40GB data, K-means, LWLR and Sobel filter show roughly 55%, 56% and 63% improvement in execution times, respectively. On average, the performance was improved roughly 58% throughout the benchmarks. Even a naïve HLS translation brings a significat impact on performance.

5.4.2 FASTEN Performances and Overhead

Compared with the *Baseline*, the *Hadoop*$_{TE\_ES}$ shows the increase in execution times by 46%, 47% and 35% for K-means, LWLR, and Sobel filter, respectively, for 40GB data. In *FASTEN*, the additional crypto engines entail 36%, 26% for and 24% increases in execution time for K-means, LWLR, and Sobel filter, respectively, over the *Baseline* for 40GB input data. However, the *FASTEN* provides a superior performance to *Hadoop*$_{TE\_ES}$, by 6%, 14% and 7% for K-means, LWLR, and Sobel filter, respectively. Table 2 shows the hardware resource utilization of each application. The AES engines in *FASTEN* utilize additional 44 BRAMs, 12 DSPs, 11,700 FFs and 7,070 LUTs for the tightened security, compared with the *Baseline*. The power consumption is listed in Table 2, as well. Compared with *Baseline*, the static power analysis in Vivado reports that the AES modules increase the power consumption by 0.144W, on average. The *FASTEN*$_{dup}$ reports 1 ~ 10% performance enhancement over the *FASTEN* at the cost of additional hardware.

The FASTEN implemented by HLS already shows the performance advantage over the other alternatives, as shown in Figure 3. However, there is still some room for enhancement in AES and kernel implementations. The AES engines can be replaced with the one from opencores.org, which reports a superior performance with HDL-based design. The performance of application kernels could also be enhanced if proper directives are applied in Vivado according to application characteristics. For a wide adoption of FASTEN, popular application kernels could be pre-implemented and provided in the HLS library.

7 Conclusions

This paper presented *FASTEN*: an FPGA-based security-tightened system for big data processing. *FASTEN* leverages the security features of the latest FPGAs such as crypto engines and PUF to protect private and sensitive data from malicious insiders and outsiders of CSP. The client's data are stored in encrypted form all the time in CSP, and the actual data processing takes place inside programmable logic. We have constructed a 24-node cluster with Zynq-7000 devices and experimented with three MapReduce applications on Ubuntu. The *FASTEN* provides a superior performane to the *Hadoop*$_{TE\_ES}$ by 9% on average, at the expense of additional hardware.

Acknowledgement

This work was supported by NATO Science for Peace and Security Programme (G4919) and the National Research Foundation of Korea(NRF) grant funded by the government (No. NRF-2014R1A1A2059652).

References

[1] Ristenpart, Thomas, et al. "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds." *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009.

[2] Byma, Stuart, et al. "FPGAs in the Cloud: Booting Virtualized Hardware Accelerators with Openstack." *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*. IEEE, 2014.

[3] Zerfos, Petros, et al. "SDFS: Secure distributed file system for data-at-rest security for Hadoop-as-a-service." *Big Data (Big Data), 2015 IEEE International Conference on*. IEEE, 2015.

[4] Schuster, Felix, et al. "VC3: trustworthy data analytics in the cloud using SGX." *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015.

[5] Eguro, Ken, and Ramarathnam Venkatesan. "FPGAs for trusted cloud computing." *22nd International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2012.

[6] Xu, Lei, et al. "Privacy preserving large scale dna read-mapping in mapreduce framework using fpgas." *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2014.

[7] Barker, Elaine, et al. "Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography." *Technical Report; National Institute of Standards and Technology (NIST): Gaithersburg, MD, USA, 2006. 2012*. 2007.

[8] Guajardo, Jorge, et al. "FPGA intrinsic PUFs and their use for IP protection." *International workshop on Cryptographic Hardware and Embedded Systems*. Springer Berlin Heidelberg, 2007.

[9] Ed Peterson. "Xilinx Application Note,"Developing Tamper-Resistant Designs with UltraScale and UltraScale+ FPGAs" *XAPP 1098*, Feb. 22, 2017. Version 1.2.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2017.2741464, IEEE Design and Test

[10] Available: https://www.rambus.com/security/dpa-countermeasures/dpa-resistant-core/. [Accessed: 20-July-2017].
[11] Chu, Cheng, et al. "Map-reduce for machine learning on multicore." *Advances in neural information processing systems* 19 (2007): 281.
[12] Kanade, Takeo, Jeffrey F. Cohn, and Yingli Tian. "Comprehensive database for facial expression analysis." *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*. IEEE, 2000.
[13] Spivey, Ben, and Joey Echeverria. *Hadoop Security: Protecting Your Big Data Platform*. " O'Reilly Media, Inc.", 2015.

AUTHORS

Boeui Hong received the B.S. degree in Computer Science from Korea University, Seoul, Korea in 2016 and he is working toward M.S. degree at KU. Email: boyhong@korea.ac.kr; Address: 145 Anam-Ro, Seongbuk-Gu, Seoul, Korea. 136-701.

Han-Yee Kim received the B.S. degree in Computer Science Education from Korea University, Seoul, Korea in 2012. He is currently a combined M.S and Ph.D. student at Korea University. Email: hanyeemy@korea.ac.kr; Address: 145 Anam-Ro, Seongbuk-Gu, Seoul, Korea. 136-701.

Minsu Kim received the B.S. degree in Computer Science from Kyonggi University, Korea in 2016. He is currently a M.S student at Korea University. Email: koggiri1990@korea.ac.kr; Address: 145 Anam-Ro, Seongbuk-Gu, Seoul, Korea. 136-701.

Lei Xu received the B.S degree in Applied Mathematics from Hebei University, China, in 2004, and the Ph.D. of Computer Science from Institute of Software, Chinese Academy of Sciences, in 2011. He is currently research assistant professor at University of Houston. From 2011 to 2013, he worked as a research engineer at the Central Research Institute, Huawei Technologies Co. Ltd. Email: lxu13@central.uh.edu Address: 3551 Cullen Blvd. Houston, TX 77204-3010

Weidong(Larry) Shi received his Ph.D. of Computer Science from Georgia Institute of Technology where he did research in computer architecture and computer systems. He was previously a senior research staff engineer at Motorola Research Lab, Nokia Research Center, and co-founder of a technology startup. Currently, he is employed as an associate professor by University of Houston. Email: wshi3@central.uh.edu Address: 3551 Cullen Blvd. Houston, TX 77204-3010

Taeweon Suh received the B.S. degree in Electrical Engineering from Korea University, Seoul, Korea in 1993, the M.S. degree in Electronics Engineering from Seoul National University, Seoul, Korea in 1995, and the Ph.D. degree in Electrical and Computer Engineering from the Georgia Institute of Technology, Atlanta in 2006. He is currently a Professor in the Department of Computer Science and Engineering, Korea University. Prior to joining academia, he was a Systems Engineer in Intel Corporation, Hillsboro, OR. Email: suhtw@korea.ac.kr; Address: 145 Anam-Ro, Seongbuk-Gu, Seoul, Korea. 136-701.