8-2019

# SafeDB: Spark Acceleration on FPGA Clouds with Enclaved Data Processing and Bitstream Protection

Han-Yee Kim

Rohyoung Myung

Boeui Hong

Heonchang Yu

Taeweon Suh

*See next page for additional authors*



## Recommended Citation

Chalk, C., Martinez, E., Schweller, R. et al. Optimal staged self-assembly of linear assemblies. Nat Comput 18, 527–548 (2019). https://doi.org/10.1007/s11047-019-09740-y

# SafeDB: Spark Acceleration on FPGA Clouds with Enclaved Data Processing and Bitstream Protection

Han-Yee Kim, Rohyoung Myung, Boeui Hong, Heonchang Yu, and Taeweon Suh*
Computer Science and Engineering
Korea University
Seoul, Republic of Korea
Email: {hanyeemy, mry1811, boyhong, yuhc, suhtw}@korea.ac.kr

Lei Xu
Department of Computer Science
University of Texas Rio Grande Valley
Brownsville, TX, USA
Email: xuleimath@gmail.com

Weidong Shi
Department of Computer Science
University of Houston
Houston, TX, USA
Email: wshi3@uh.edu

***Abstract*—This paper proposes SafeDB: Spark Acceleration on FPGA Clouds with Enclaved Data Processing and Bitstream Protection. SafeDB provides a comprehensive and systematic hardware-based security framework from the bitstream protection to data confidentiality, especially for the cloud environment. The AES key shared between FPGA and client for the bitstream encryption is generated in hard-wired logic using PKI and ECC. The data security is assured by the enclaved processing with encrypted data, meaning that the encrypted data is processed inside the FPGA fabric. Thus, no one in the system is able to look into clients' data because plaintext data are not exposed to memory and/or memory-mapped space. SafeDB is resistant not only to the side channel attack but to the attacks from malicious insiders. We have constructed an 8-node cluster prototype with Zynq UltraScale+ FPGAs to demonstrate the security, performance, and practicability.**

***Keywords—FPGA as a service, Bitstream protection, Enclaved data processing, Spark big data processing***

## I. INTRODUCTION

Clouds are attractive platforms for tasks that require large amounts of resources because of its scalability, elasticity, flexibility, and cost savings. Compute resources and storage can be easily scaled up and down in a pay-as-you-go manner. The clouds also provide convenient big data processing environment such as MapReduce and Spark for the broad adoption from the public. However, customers with private and/or sensitive data may be reluctant to use the clouds due to the security concern. Most big data processing frameworks incorporate some security features, which are typically based on cryptography [1]. Nonetheless, the data confidentiality is not assured once the key for the crypto operation is leaked. One extreme way of protecting data in software-only processing is to use fully homomorphic encryption [2]. It allows for directly processing cipher-text data without decryption, but it is too expensive in terms of the processing time. With the growing concerns for security, processor vendors such as Intel, AMD, and ARM are shipping their products with Trusted Execution Environment (TEE) called SGX, SME, and TrustZone. However, as revealed in Spectre and Meltdown cases [3, 4], microarchitecture weaknesses are exploited to disarm the security and extract data in main memory.

Field-Programmable Gate Arrays (FPGAs) are being emerged as one of the major resources in clouds because data processing can be accelerated with customized hardware engines. These days, renowned cloud providers are already offering FPGAs in their premises. For example, Amazon EC2 F1 and Huawei FACS provide Xilinx's Virtex UltraScale+ FPGAs, whereas Microsoft Azure offers Intel's Arria-10 GX FPGAs. FPGAs are configured with the bitstream that is a low-level representation of hardware design. The bitstream is susceptible to similar security attacks to software, including unauthorized copy, IP theft, reverse-engineering, and tampering [5]. If the hardware design is reverse-engineered and tampered from the bitstream, an attacker could peek the data being processed and transfer it through a covert channel. To cope with this problem, modern FPGAs provide the bitstream protection mechanisms mostly based on the AES encryption, and the encryption key is stored in on-board storage. In the cloud environment, the bitstream protection is susceptible to the man-in-the-middle (MITM) attack. It is because the key along with the encrypted bitstream should be sent remotely from customers to the clouds for the FPGA configuration.

In this paper, we propose SafeDB, a Spark-based comprehensive and systematic framework for security and data processing acceleration in FPGA-based clouds. For the bitstream protection, SafeDB proposes a hard-wired logic in FPGAs for the key exchange and authentication using the Public Key Infrastructure (PKI). For the data confidentiality, it implements a fully enclaved processing of application kernels inside FPGA fabric, taking and emitting only encrypted data. Thus, the plaintext data are never exposed to memory-mapped space. We estimated the hardware cost of implementing the proposed bitstream protection scheme. For the performance evaluation, we have constructed a complete SafeDB cluster system with eight Zynq UltraScale+ devices [6]. The experiments show that SafeDB achieves the performance improvement of real applications by up to 1.36x.

The remainder of this paper is organized as follows: Section II introduces background information. Section III summarizes the related works. Section IV details the proposed architecture with its security analysis. In Section V, we elaborate on the detailed implementation. Section VI shows the experimental results and their analysis. We discuss the

practicability and usability of the SafeDB in Section VII. We finally conclude our paper in Section VIII.

## II. BACKGROUND

### A. Spark Overview

Spark is one of the most prevalent frameworks for big data processing with fault tolerance support and in-memory cluster computing. It provides high-level APIs in Java, Scala, Python and R. With the APIs, it is convenient to develop applications without requiring much of customization. Spark aims at improving the MapReduce framework in terms of the flexibility via high-level APIs and the performance by minimizing accesses to the secondary storage. Spark supports batch processing and provides a micro-batch based streaming model. Its framework also offers diverse tools including Spark SQL for the structured data processing, MLlib for the machine learning, GraphX for the graph processing, and Spark streaming for real-time processing. For performance, it especially introduces a new abstract data structure called resilient distributed dataset (RDD). The RDD is a read-only multiset of data items normally residing in memory. RDD can be created from data in secondary storage or other RDDs. It can act as a working set of data processing. Various transformations such as data-accumulation-by-key can be applied to RDDs. The sequence of RDD operations is tracked in Spark for fault tolerance.

### B. Bitstream Protection in Modern FPGAs

Modern FPGAs are equipped with hard-wired bitstream protection modules where the AES, a symmetric cryptography, is typically used for the bitstream encryption. The symmetric key itself is also encrypted and stored in on-board storage. Table I summarizes the encryption methods and key storages in modern FPGAs from major vendors.

TABLE I. SECURITY FEATURES OF MODERN FPGAS

| Security Features | Xilinx | | Intel | | Microsemi |
|---|---|---|---|---|---|
| | ***Virtex / Kintex UltraScale+*** | ***Zynq UltraScale+*** | ***Arria-10 GX / SX*** | ***Stratix-10 GX / SX*** | ***SmartFusion2*** |
| Bitstream encryption | AES-GCM 256 | AES-GCM 256 | AES-GCM 256 | AES-GCM 256 | AES-GCM 128/256 with ECDH |
| Key Storage | BBRAM eFUSE | BBRAM eFUSE PUF | BBRAM eFUSE | BBRAM eFUSE PUF | FLASH PUF |

Since the bitstream itself is shielded by the AES encryption, the remaining issue is the AES key protection. As shown in Table I, Physically Unclonable Function (PUF) is especially incorporated in the major devices and used for the key protection. The PUF takes advantage of semiconductor process variations such as oxide thickness and channel length. It is used as a digital fingerprint of each device. The PUF generates a unique response (output) for each challenge (input). There are two PUF-based key protection schemes in FPGAs; The first one directly generates the encrypted key with the PUF output and stores the result (encrypted key) in on-board non-volatile memory. At the time of the FPGA configuration with the encrypted bitstream, the encrypted key in the non-volatile memory is decrypted by the PUF's output and used for the bitstream decryption. This scheme is applied to Zynq UltraScale+. The second scheme is based on the key exchange protocol with Diffie-Hellman algorithms where the 'shared secret' between user and FPGA is generated. With its own private key and the counterpart's public key, both the user and FPGA can generate the same AES key. On the FPGA side, the private key of each FPGA is the PUF output and its corresponding public key is released to the user. At the time of the FPGA configuration with the encrypted bitstream, the shared AES key is generated inside the FPGA and used for the bitstream decryption. This scheme is used in the Microsemi SmartFusion2.

## III. RELATED WORKS

There are several studies on securing big data workload. Zerfos [7] proposed a secure distributed file system (SDFS) for Hadoop to provide data-at-rest protection. SDFS can read and write encrypted data on behalf of MapReduce applications using keys that are distributed and controlled by customers. Nevertheless, software-based data-at-rest security has drawbacks in that the plaintext is inevitably exposed in memory space and the performance can negatively be affected by the cryptographic operation. Schuster [8] proposed Verifiable Confidential Cloud Computing (VC3) system using Intel SGX. The VC3 ensures that the MapReduce applications are confidentially and verifiably executed under untrusted cloud environments. However, as demonstrated in SgxPectre [9], the SGX can be compromised by exploiting the architectural vulnerability.

There are case studies for big data processing acceleration using reconfigurable hardware. Shan et al [10] implemented a MapReduce framework on FPGA, referred to as FPMR, where the whole process of the mapper and reducer including scheduler is executed inside the FPGA. Chen et.al. presented the design of FPGA-based Spark acceleration for DNA sequencing [11]. Morcel et.al [12] implemented an FPGA-based accelerator for the distributed training of deep convolutional neural network with Spark. These prior works have focused only on acceleration without considering the data confidentiality.

There are prior works regarding the bitstream protection for the remote FPGAs in clouds. Eguro et al [13] proposed a trusted third party based bitstream protection scheme. The proposal relies on the Trusted Authority (TA) for key management. More specifically, TA undertakes to register AES keys for all FPGA boards before shipping to the clouds, encrypting all the bitstreams requested from users, and sending them to the clouds. This method is susceptible to MITM attack when sending bitstreams to the TA. Genssler et al [14] proposed a secure bitstream protection scheme and its end-to-end connection protocol between virtualized FPGA and client. On the FPGA fabric, client's encrypted bitstream is decrypted and configured to a certain area of FPGA fabric via partial reconfiguration technology. To the best of our knowledge, SafeDB is the first comprehensive and systematic framework for security and acceleration for FPGA-as-a-Service (FaaS). It covers from the bitstream protection to the

data confidentiality with the real implementation and evaluation on an off-the-shelf FPGA based cluster.

## IV. SafeDB Architecture and its Security Analysis

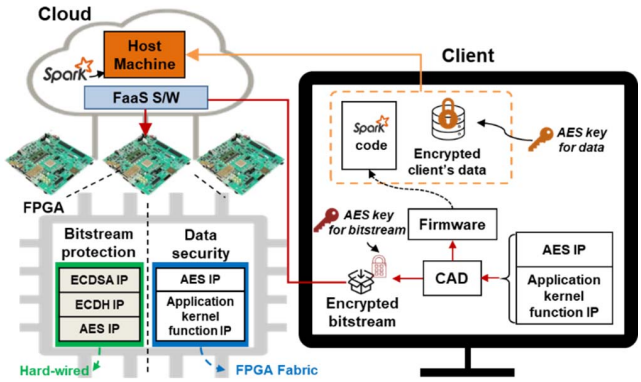


Fig 1. SafeDB overview

### A. Threat Model

The Cloud Service Provider (CSP) offers FPGAs and we assume that the FPGAs are trusted and their vendors have a complete monitoring mechanism to prevent adversaries from influencing the device manufacturing process. On the CSP side, there may be malicious insiders or outsiders who want to learn about the data being processed. They may sneak into the cloud system and try to steal the clients' private and/or sensitive information via illegal access. They may even try to tamper hardware bitstream to figure out sensitive data. The clients have large amounts of personal and/or sensitive data to manipulate and thus need a big data processing platform that guarantees no information leakage both in the perspective of data and its functional behavior. Simply speaking, no one on the CSP side should be able to learn about customers' data and its hardware design.

### B. SafeDB System Architecture

SafeDB provides a systematic hardware-based mechanism from the bitstream protection to data security. Fig. 1 shows the operational overview of SafeDB where the data and bitstream flow from a client to CSP is depicted. The client has data to process and application to run, which are sent to CSP for execution. For security, data is encrypted with the client's AES key and sent to CSP. The application is divided into two parts: house-keeping and kernel codes. The kernel code, which actually processes the client's data, is automatically translated to the hardware by CAD tools and will later be executed inside FPGA on CSP. The house-keeping code is executed in software on CSP. For security, FPGA takes encrypted data, decrypts it, executes the kernel and finally encrypts the output for passing it externally. Thus, the bitstream is prepended with the AES decryption module and appended with the AES encryption module, which is performed in the SafeDB framework. To call the hardware-translated kernel in runtime, the client should prepare firmware with the house-keeping code. The CAD tool helps generate the firmware. On the CSP side, there is a management tool called FPGA-as-a-Service (FaaS) software that mediates an end-to-end connection between FPGA and the client for the initial configuration. It performs simple

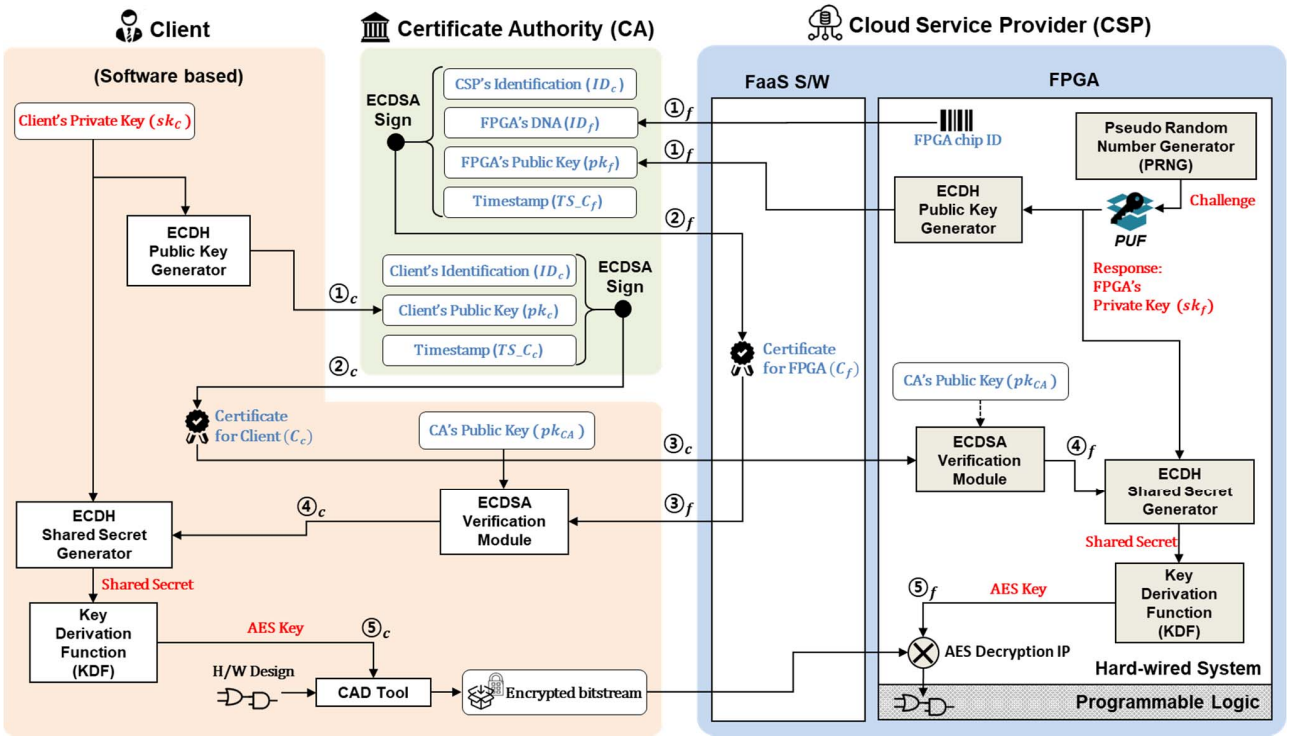


Fig 2. Basic components and operational procedure of bitstream protection in SafeDB

communication operations such as passing metadata, for bitstream protection.

### C. Bitstream Protection in SafeDB

As in modern FPGAs, the bitstream itself is encrypted by AES on the client side. Then, the AES key should be safely shared with FPGAs on the clouds. For the AES key sharing, the SafeDB utilizes the PKI and ECC algorithms. The PKI is an asymmetric key based security system. In the PKI system, a trusted party called Certificate Authority (CA) guarantees that a public key is genuine by issuing a certificate, which is signed by the CA's private key. Then, each party is able to authenticate the public key of its counterpart.

The hardware components and operation flow required for the key sharing are depicted in Fig. 2. The PRNG is used to generate diverse inputs (challenges) to PUF. The PUF's output (response) is used as a private key ($sk_f$). Then, its corresponding public key ($pk_f$) is generated with $sk_f$. Note that only public key is released to the outside world. Initially, both client and FPGA generate their own private keys ($sk_c$, $sk_f$), and their corresponding public keys ($pk_c$, $pk_f$) are computed from the ECDH public key generator. Then, the FPGA identification and its public key are transferred to CA ($①_f$), and the client also sends its public key to CA ($①_c$). The FPGA chip ID from the manufacturer can be used as the FPGA identification. Then, CA issues and sends an ECDSA certificate to each party ($②_c, ②_f$). The CA-signed certificate contains the ID, public key, and timestamp for validity. When a client wants to use FPGAs, he/she sends a request to the CSP with the CA-signed certificate ($③_c$). Then, the FaaS software assigns several boards and sends the corresponding CA-signed certificates to the requestor ($③_f$). On each party, the counterpart's certificate is verified in the ECDSA verification module ($④_c, ④_f$). With its own private key and its counterpart's public key, each party generates the same shared secret. Finally, the AES key used for the bitstream encryption is generated from the KDF with the shared secret. With the derived AES key, the client generates the encrypted bitstream ($⑤_c$) and sends it to CSP. FPGA on the CSP then decrypts the received bitstream with the same derived AES key ($⑤_f$). Because all critical information is hidden in hard macro blocks inside FPGA, the proposed scheme has no surface for rogues to steal and/or eavesdrop AES key and/or private key on the CSP side.

### D. Data Confidentiality in SafeDB

For the data confidentiality, SafeDB proposes a fully enclaved processing by migrating Spark application kernel functions to the FPGA fabric in an isolated manner. All the data-at-rest are stored in memory or secondary storage in encrypted form even it is intermediate data. To take the encrypted data for processing and emit the encrypted output after processing, one AES crypto engine is prepended on the front side of the application kernels, and another is appended on the rear side. Note that the AES key for the data protection is different from the AES key for the bitstream protection; The latter is distinct according to the FPGAs because the PUF inside each FPGA generates a unique private key. The client would want to use the same AES key for data regardless of the FPGAs. The AES key for data is included in the bitstream.

The detailed operational procedure and hardware/software co-operation for data confidentiality is depicted in Fig. 3. First, a Spark application partitions the encrypted data and calls the *firmware* to execute the application kernels ported into the FPGA fabric. The *firmware* contains the device drivers for the application kernel function. The *firmware* takes an argument, which is either a path to the input file or input data itself. Then, the partitioned encrypted data in memory is supplied to the AES decryption module in the FPGA fabric via DMA. After the decryption, the application kernel performs the data processing. Then, its output is passed to the AES encryption module. Finally, the DMA carries out the encrypted data transfer from the FPGA fabric to memory.

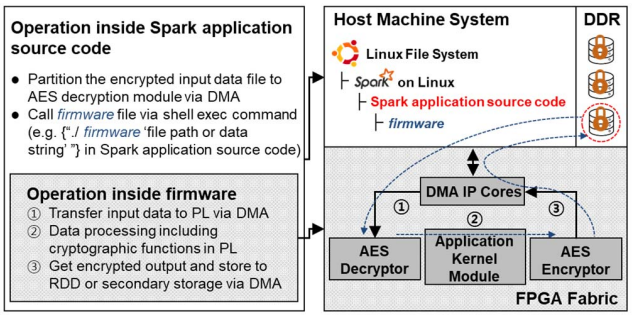


Fig 3. Basic components and operational procedure for data confidentiality in SafeDB

### E. Security Analysis

In SafeDB, it is not feasible to look into the plaintext data. It is because the data is stored in an encrypted form all the time in memory, and the data processing occurs inside the FPGA fabric. Furthermore, the bitstream is also protected by the hard-wired IPs described in Section IV-C. It means that known software attacks [15] are essentially blocked and prevented. Side channel attack (SCA) may be attempted to extract key information from AES modules inside FPGA. There are two kinds of attack scenarios in SCA: Timing analysis and Power analysis. In the timing-based attack [16], attackers measure the time to perform a certain operation. By measuring the execution times according to parameters, the attackers can speculate the stored key. However, the AES algorithm in SafeDB is implemented as hardware and there is no memory-mapped register for storing the intermediate outcome of crypto operations. In addition, the implemented AES engine consumes a constant amount of time regardless of inputs. Therefore, the timing attack is not feasible. The power analysis based SCA may be carried out if attackers know the information gathered from the power trace with knowledge about the implementation. The power trace is collected by physically connecting a power meter to the system. There is a published power analysis attack against a Virtex 800 FPGA [17]. However, the power-based SCA requires prior knowledge about the hardware implementation and its location. It is not virtually possible to perform this kind

of attacks in SafeDB because the bitstream is created and encrypted from the client.

## V. SafeDB Cluster System Implementation

### A. Experiment Environment

We used eight ZCU102 boards for cluster setup. Each board has Zynq UltraScale+ where Cortex-A53 quad-core processor with 1.5 GHz, 4GB DDR4 RAM, and a number of programmable logic cells are contained. Fig. 4 shows a picture of a constructed 8-node cluster. The eight nodes are connected through Ethernet via a switch. SDSoC 2016.4v [18], a CAD tool from Xilinx, is used for hardware and software implementation. The SDSoC provides a capability of the automated system-level integration for C/C++/OpenCL code, targeting Zynq programmable SoCs. The system-level integration includes software-to-hardware translation, device driver generation, and kernel creation. The SDSoC allows users to specify software functions to be translated to hardware. For the maximal performance and throughput that FPGA allows, it provides directives that can be annotated in C/C++ source code.

Fig 4. Cluster prototype with eight Zynq UltraScale+ boards

### B. Cluster Setup with Linux Kernel, File System, and Boot Image Creation

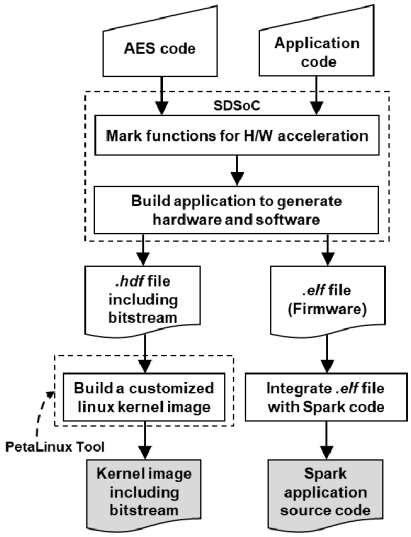


Fig 5. Hardware bitstream and firmware generation flow in SafeDB prototype with Zynq Ultrascale+

Fig. 5 shows the bitstream and software generation flow for the experiments with a prototyped SafeDB system. We implemented AES software for data protection, which will be provided to clients. For the hardware implementation with SDSoC, application kernel functions and AES are marked for the hardware translation. Then, SDSoC generates a hardware description file (*hdf*) including bitstream. With the *hdf* file, PetaLinux tool [19] can build a customized linux kernel image. The SDSoC also build an *elf* file, which is firmware. The *elf* is included in each target application program. We used the Ubuntu 16.04 ARMv8 file system from Ubuntu official repository and installed Spark 2.2.

### C. Target Applications

We used three benchmark programs to evaluate the SafeDB in terms of performance and hardware utilization: Word Count, Sobel Filter, and Logistic Regression. The benchmark details are elaborated as follows:

- Word Count (WC): WC is one of the most commonly used benchmarks for big data processing. It reads input data, splits each word, and creates a key-value pair. Then, all the intermediate values are accumulated if the key (word) is the same. The complexity of WC is $O(n)$. The WC hardware module processes a 2MB data per execution.
- Sobel filter (SF): The Sobel operator is used in image processing and computer vision, particularly for detecting edges. First, each image is split into window-sized partitions. In our implementation, each original image was split into 1024x512 partitions. Then, the edge detection is performed with the 3x3 kernel, to compute the derivative approximations of image convolution. The complexity of SF is $O(n^2)$.
- Logistic regression (LR): LR is widely used to predict a binary response. The binary logistic model is used to estimate the probability of a binary response based on one or more predictor variables. LR is used in various fields, especially machine learning. In this application, the Spark splits encrypted input data by 2MB, which has 1024 data sets. Each set has 512 floating-point data representing 512 kinds of features. In the hardware module, the weight update and regularization occur continuously. The complexity of implemented LR is $O(n^3)$.

### D. Bitstream Protection Hardware with PUF

We have implemented hardware modules for bitstream protection described in Section IV-C with Vivado. Based on the open-source software, we have made modifications for high-level synthesis and generated a PRNG, ECDH public key & shared secret generator, and ECDSA verification module. The PRNG is based on a linear feedback shift register (LFSR). B-571(sect571r1) parameters are selected for the ECDH and ECDSA implementation. The B-571 has 571-bit private key and 1142-bit public key for the ECDH handshaking. We set the same 571-bit output length for both PUF and ECDH shared secret. The ECDSA module is also fitted to take the 1142-bit public key. The KDF module takes the 571-bit shared secret and uses SHA256 to generate the 256-bit AES key. It is called the HMAC-based extract-and-expand key derivation function (HKDF). For the PUF, we used

Anderson's soft PUF [20]. It takes advantage of the timing delay variation between two carry-chained lookup tables when used as the shift register. In the implementation, the XOR-based challenge and response framework is applied to get the diverse outputs from the PUF. The PUF is designed via HDL. The hardware cost for each module is estimated in Section VI.

## VI. SAFEDB EVALUATIONS

### A. Resource Utilization and Performance of Hardware Components

Table II summarizes the resource utilization and performance of each hardware module for bitstream protection. Block RAMs (BRAMs) are used as internal memory in Xilinx FPGA. Digital Signal Processing units (DSPs) are typically used when translating signal processing algorithms with multiply-accumulation (MAC) operations. Flip-flops (FFs) and Lookup Tables (LUTs) are used for general logic designs. Overall, the bitstream protection hardware occupies 4.61% of BRAMs, 4.69% of FFs, and 17.72% of LUTs of the programmable logic. When it comes to the performance, all modules are operating at 100MHz. The PRNG and the PUF take 1 cycle and AES256 take 520 cycles for execution, whereas the execution times of ECDH, ECDSA, and KDF modules are dynamically determined according to inputs.

TABLE II. RESOURCE UTILIZATION AND EXECUTION TIME OF HARDWARE COMPONENTS FOR BITSTREAM PROTECTION

| *Hardware Components* | *#BRAMs (18Kbit)* | *#DSPs (48E)* | *#FFs* | *#LUTs* | *Execution Time* |
|---|---|---|---|---|---|
| PRNG 100MHz | 0 (0%) | 0 (0%) | 0 (0%) | 522 (0.19%) | 1 Cycle (10ns) |
| PUF 100MHz | 0 (0%) | 0 (0%) | 1,168 (0.21%) | 2,170 (0.79%) | 1 Cycle (10ns) |
| ECDH Public key generator 100MHz | 20 (1.10%) | 0 (0%) | 4,476 (0.82%) | 12,926 (4.72%) | N/A |
| ECDH Shared secret generator 100MHz | 26 (1.43%) | 0 (0%) | 4,200 (0.77%) | 13,187 (4.81%) | N/A |
| ECDSA Verification module 100MHz | 22 (1.21%) | 0 (0%) | 3,362 (0.61%) | 10,790 (3.94%) | N/A |
| KDF 100MHz | 8 (0.44%) | 0 (0%) | 1,056 (0.19%) | 3,987 (1.45%) | N/A |
| AES256 decryption 100Mhz | 8 (0.44%) | 0 (0%) | 11,457 (2.09%) | 4,990 (1.82%) | 520 cycles (5,200ns) |

Table III summarizes the resource utilization and performance of each hardware component for data processing. The WC application kernel consumes relatively low hardware resources due to the simplicity of the application. In SF and LR, BRAMs are mainly used in the hardware translation whereas DSPs, FFs, and LUTs are not much utilized in the three application kernels. After the hardware translation, we have measured the maximum operating frequency of the hardware system, which turned out to be 200MHz due to the timing constraints. The execution times of WC and SF hardware kernels are roughly 10.49ms and 80.86ms, respectively. The execution time of LR varies around 10.81sec because a certain internal loop can finish when the break condition is met. The AES128 encryption and decryption take about 0.2μs for a single execution separately.

TABLE III. RESOURCE UTILIZATION AND EXECUTION TIME OF HARDWARE COMPONENTS FOR DATA PROCESSING

| *Hardware Components* | *#BRAMs (18Kbit)* | *#DSPs (48E)* | *#FFs* | *#LUTs* | *Execution Time* |
|---|---|---|---|---|---|
| WC kernel 200MHz | 2 (0.11%) | 0 (0%) | 714 (0.13%) | 520 (0.19%) | 2,097,154 cycles (10,485,770ns) |
| SF kernel 200MHz | 1,698 (93.09%) | 16 (0.63%) | 13,367 (4.88%) | 16,446 (3.00%) | 16,172,253 cycles (80,861,265ns) |
| LR kernel 200MHz | 933 (51.15%) | 22 (0.87%) | 3,877 (0.71%) | 8,104 (2.96%) | ↑ 2,162,181,647 cycles ( ↑ 10,810,908,235ns) |
| AES128 encryption 200Mhz | 0 (0%) | 0 (0%) | 6,865 (1.25%) | 3,300 (1.20%) | 426 cycles (2,130ns) |
| AES128 decryption 200Mhz | 8 (0.44%) | 0 (0%) | 11,455 (2.09%) | 4,982 (1.82%) | 384 cycles (1,920ns) |

### B. Resource Utilization, Power Consumption, and Performance of a Single Node

Table IV shows the overall resource utilization and power consumption of final hardware systems on a single node according to the target applications. They are different from the ones in Table III. It is because the system interconnection such as AXI is always required and DMA engines are integrated for speeding up the data transfer. The FFs and LUTs are utilized by up to 8.16 %, and 12.21%, respectively.

TABLE IV. OVERALL RESOURCE UTILIZATION AND POWER CONSUMPTION OF TARGET APPLICATIONS ON ULTRASCALE+

| *Hardware Systems* | *#BRAMs (18Kbit)* | *#DSPs (48E)* | *#FFs* | *#LUTs* | *Power Consumption* |
|---|---|---|---|---|---|
| WC System 200MHz | 143 (7.84%) | 3 (0.12%) | 38,543 (7.03%) | 27,925 (10.19%) | 4.023W |
| SF kernel 200MHz | 1,771 (97.09%) | 22 (0.87%) | 42,171 (7.69%) | 32,057 (11.70%) | 4.772W |
| LR kernel 200MHz | 1,055 (57.84%) | 28 (1.11%) | 44,703 (8.16%) | 33,465 (12.21%) | 4.463W |

Fig. 6 shows the resource utilization breakdown of hardware systems in the FPGA fabric except for DSP. In the WC, the application kernel module is relatively tiny so that AXI interface modules occupy a considerable portion of hardware resources. Except for the WC system, the BRAMs were mainly utilized for application kernel functions. In the SF system, FFs and LUTs are also mainly consumed by the application kernel function. The power consumptions for each benchmark follow the allocated BRAM resources because the utilization variation of the other resources is marginal.

Fig. 7 shows normalized execution times of target applications over software-only approach, which is used as a *baseline* in the experiment. The experiments were performed by running each *elf* file once. We observed that, as the

algorithm complexity increases, the acceleration effect is relatively moderate. The WC system took the benefit of SafeDB the most by boosting performance by 25.6x over the baseline. The SF system provides roughly an 18x performance improvement, whereas the LR achieves a 2x performance over the baseline. We strongly believe that it is because LR exhibits the data locality characteristic, which takes advantage of the caches of Cortex-A53 in the baseline.

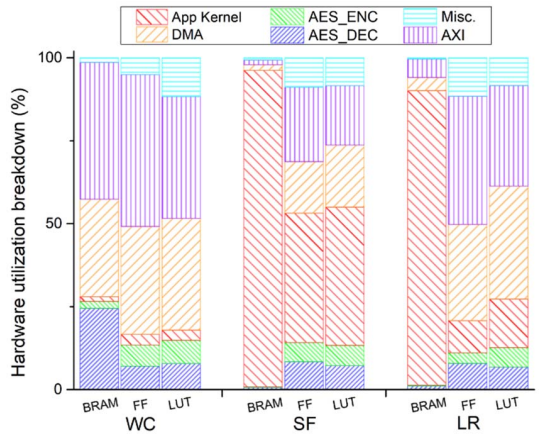


Fig 6. Resource utilization breakdown of hardware systems according to resource types in the FPGA fabric of Zynq UltraScale+

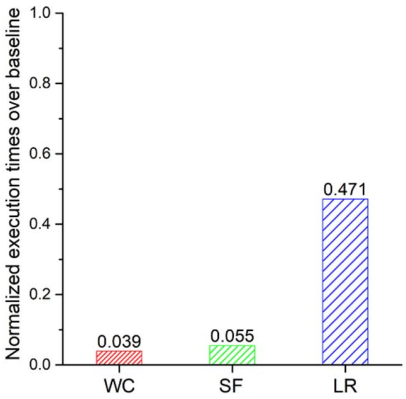


Fig 7. Normalized execution times of each applicaion over baseline without Spark environment.

To measure the performance of the organized cluster, the target applications were executed with three workload sizes: 64GB, 128GB, and 192GB. To maximize the hard-wired processing system performance, each node executes four threads. Note that the hard-wired processing system has four Cortex-A53 cores. Fig. 8 shows the normalized execution times over the baseline. The SafeDB improves performance throughout the benchmark programs. Especially, the LR shows the largest improvement of 1.36x with 64GB data. However, compared with the single execution in Fig. 7, the performance difference from the baseline is not considerable. There are two reasons: the difference in the number of threads and the task management scheme in Spark. First, by assigning the tasks to four cores, the performance of the baseline was improved dramatically by ideally up to 4x. The second reason is the overhead in Spark task management. Spark uses a method called the lazy evaluation, with which the execution is not started immediately. Thus, the *elf* execution may be delayed, and the impact of the overhead will be noticeable as the execution time of the translated application kernel in hardware becomes shorter. The pure execution times of WC and SF are around 0.3 seconds, whereas LR takes around 11 seconds per single execution. Thus, as shown in Fig. 8, the LR exhibits a better improvement in performance, relatively well absorbing its overhead.

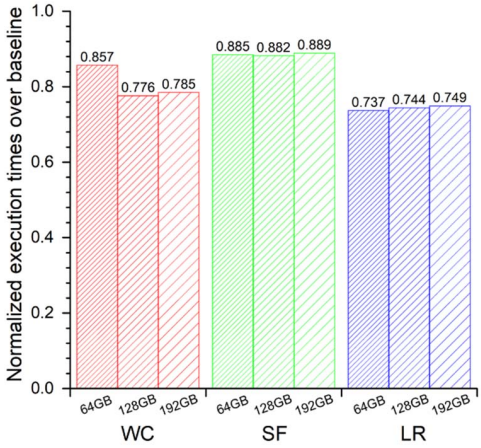


Fig 8. Normalized execution times on the organized cluster (that is , 8 UltraScale+ boards) over baseline

## VII. DISCUSSION

For security, modern FPGAs are equipped with hard-macro IPs; Xilinx integrates Chip Security Unit (CSU) and Intel has Secure Device Manager (SDM). SafeDB proposes such hard-wired bitstream protection for the cloud environment, taking advantage of ECC and PUF with the PKI system. As reported in Section VI, the overall hardware cost for bitstream protection is reasonably low. Therefore, we strongly believe that the proposed solution can be easily implemented in the FPGAs targeting for the cloud environment.

The proposed solution can be used not only for the big data processing purposes but for other application areas such as Blockchain where the security demand is high. For example, for the blockchain system with FPGAs, ECDH or ECDSA can be implemented in the FPGA fabric, and soft PUFs can be used to generate a random number for a private key. Even blockchain virtual machines can be implemented in FPGA fabric to increase security for smart contracts. It would make the transactions on Blockchain much more secure, compared to the software approach.

There were some design considerations when implementing a SafeDB hardware system with automated CAD tools such as SDSoC. While FPGAs can be used for acceleration by parallelism, too many hardware modules can lead to inefficiency and/or problems such as input and output port scarcity. Other factors to consider is the processing granularity of data and computation complexity. If a processing module is designed to take a small granule of data

or have a simple complexity, the acceleration effect would be trivial or it may even degrade the performance because of the overhead from the data transfer latency between DDR and FPGA fabric. In the execution of target applications, the data size granule is ~MB and the computation complexity is from $O(n)$ to $O(n^3)$. According to our evaluation, a complex and time-consuming task tends to take better advantage of the hardware acceleration.

## VIII. CONCLUSION

This paper presented SafeDB and demonstrated its security, performance, and practicality. By taking advantage of ECC, PUF, and PKI system, the AES key for the bitstream protection is securely shared between client and FPGA. It is because all the critical operations are performed by the hard-wired logic in FPGA without human involvement. The proposed scheme is resistant not only to the man-in-the-middle attack but to the attacks from malicious insiders. In SafeDB, the data confidentiality is guaranteed by migrating application kernel functions with AES crypto engines to the FPGA fabric. The AES key for data is included in the hardware bitstream. For the evaluation, we have constructed a cluster prototype using 8 Zynq UltraScale+ boards. Our experiments revealed that SafeDB provides the performance benefit as well, thanks to the hardware acceleration effect. In the case of the LR system with a 64GB workload, it improved the performance by up to 1.36x. The FPGA-based clouds are being emerged in the market and are already in service. We believe that our approach would expedite a broader adoption of FPGA-aided cloud systems. The proposed scheme can be applied to all the big-data platforms with FPGAs and other security-demanding sectors such as Blockchain.

## ACKNOWLEDGMENT

This work was partially supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2019-0-00533, Research on CPU vulnerability detection and validation). *Correspondence to: Taeweon Suh.

## REFERENCES

[1] Apache Spark. "Spark Security" [Online] Available: https://spark.apache.org/docs/latest/security.html

[2] Gentry, C., "Fully homomorphic encryption using ideal lattices", Stoc. Vol. 9. 2009.

[3] Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., and Yarom, Y., "Spectre attacks: Exploiting speculative execution." arXiv preprint arXiv:1801.01203, 2018.

[4] Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Mangard, S., Kocher, P., Genkin, D., Yarom, Y. and Hamburg, M., "Meltdown." arXiv preprint arXiv:1801.01207, 2018.

[5] Trimberger, S.M. and Moore, J.J., "FPGA security: Motivations, features, and applications." Proceedings of the IEEE, 102(8), pp.1248-1265, 2014.

[6] Boppana, V., Ahmad, S., Ganusov, I., Kathail, V., Rajagopalan, V., and Wittig, R., "UltraScale+ MPSoC and FPGA families." 2015 IEEE Hot Chips 27 Symposium (HCS).

[7] Zerfos, P., Yeo, H., Paulovicks, B.D. and Sheinin, V., "SDFS: Secure distributed file system for data-at-rest security for Hadoop-as-a-service." In 2015 IEEE International Conference on Big Data (Big Data), pp. 1262-1271.

[8] Schuster, F., Costa, M., Fournet, C., Gkantsidis, C., Peinado, M., Mainar-Ruiz, G., and Russinovich, M., "VC3: Trustworthy data analytics in the cloud using SGX." In 2015 IEEE Symposium on Security and Privacy, pp. 38-54.

[9] Chen, G., Chen, S., Xiao, Y., Zhang, Y., Lin, Z., and Lai, T.H., "Sgxpectre attacks: Leaking enclave secrets via speculative execution." arXiv preprint arXiv:1802.09085, 2018.

[10] Shan, Y., Wang, B., Yan, J., Wang, Y., Xu, N., and Yang, H., "FPMR: MapReduce framework on FPGA." In Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays, pp. 93-102. 2010.

[11] Chen, Y.T., Cong, J., Fang, Z., Lei, J., and Wei, P., "When Spark Meets FPGAs: A Case Study for Next-Generation DNA Sequencing Acceleration." In 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud). 2016.

[12] Morcel, R., Ezzeddine, M., and Akkary, H., "Fpga-based accelerator for deep convolutional neural networks for the spark environment." In 2016 IEEE International Conference on Smart Cloud (SmartCloud), pp. 126-133.

[13] Eguro, K. and Venkatesan, R.,. "FPGAs for trusted cloud computing." In 22nd International Conference on Field Programmable Logic and Applications (FPL), pp. 63-70. IEEE, 2012.

[14] Genssler, P.R., Knodel, O., and Spallek, R.G., "Securing Virtualized FPGAs for an Untrusted Cloud." In Proceedings of the International Conference on Embedded Systems, Cyber-physical Systems, and Applications (ESCS), pp. 3-9. 2018.

[15] LeBlanc, D. and Viega, J., "24 deadly sins of software security: programming flaws and how to fix them.", McGraw-Hill. 2010.

[16] Bernstein, D.J., "Cache-timing attacks on AES.", 2005.

[17] Kocher, P., Jaffe, J. and Jun, B., "Differential power analysis.", In Annual International Cryptology Conference (pp. 388-397). Springer, Berlin, Heidelberg. 1999.

[18] Kathail, V., Hwang, J., Sun, W., Chobe, Y., Shui, T. and Carrillo, J., "SDSoC: A higher-level programming environment for Zynq SoC and Ultrascale+ MPSoC.", In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (pp. 4-4).

[19] Xilinx. "PetaLinux Tools" [Online] Available: https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html/

[20] Anderson, J.H., "A PUF design for secure FPGA-based embedded systems.", In Proceedings of the 2010 Asia and South Pacific Design Automation Conference (pp. 1-6). IEEE Press.