Cryptographically Secure Multi-tenant Provisioning of FPGAs

Arnab Bag[1(✉)], Sikhar Patranabis[1], Debapriya Basu Roy[2], and Debdeep Mukhopadhyay[1]

[1] Indian Institute of Technology, Kharagpur, Kharagpur, India
amiarnabbolchi@gmail.com
[2] Technische Universität München, Munich, Germany

Abstract. Field-programmable gate arrays (FPGAs) have gained massive popularity today as accelerators for a variety of workloads, including big data analytics, and parallel and distributed computing. This has fueled the study of mechanisms to provision FPGAs among multiple tenants as general purpose computing resources on the cloud. Such mechanisms offer new challenges, such as ensuring IP protection and bitstream confidentiality for mutually distrusting clients sharing the same FPGA. A direct adoption of existing IP protection techniques from the single tenancy setting do not completely address these challenges, and are also not scalable enough for practical deployment.

In this paper, we propose a dedicated and scalable framework for secure multi-tenant FPGA provisioning that can be easily integrated into existing cloud-based infrastructures such as OpenStack. Our technique has *constant resource/memory overhead* irrespective of the number of tenants sharing a given FPGA, and is provably secure under well-studied cryptographic assumptions. A prototype implementation of our proposition on Xilinx Virtex-7 FPGAs is presented to validate its overheads and scalability when supporting multiple tenants and workloads. To the best of our knowledge, this is the first FPGA provisioning framework to be prototyped that achieves a desirable balance between security and scalability in the multi-tenancy setting.

Keywords: FPGAs · Security · Provisioning · Multi-tenant · Cloud computing

1 Introduction

The modern era of cloud computing has actualized the idea of ubiquitous provisioning of computational resources and services via a network. Cloud-based solutions are now marketed by all leading enterprise IT vendors such as IBM (PureApplication), Oracle (ExaData), Cisco (UCS) and Microsoft (Azure), as well as Web companies such as Amazon (AWS) and Google (Compute Engine). In the midst of this paradigm shift from traditional IT infrastructures

L. Batina et al. (Eds.): SPACE 2020, LNCS 12586, pp. 208–225, 2020.
https://doi.org/10.1007/978-3-030-66626-2_11

to the cloud, field-programmable gate arrays (FPGAs) have risen as attractive computational avenues for accelerating heavy workloads.

Modern FPGAs offer a number of advantages including reconfigurability, high throughput, predictable latency and low power consumption. They also offer *dynamic partial reconfiguration* (DPR) capabilities [21], that allow non-invasive run-time modification of existing circuitry for on-the-fly functionality enhancement. The recent trend of deploying FPGAs as computing resources on the cloud is visible in upcoming commercial applications such as Microsoft's Project Catapult [28], that integrates FPGAs with cloud-based data centers in a distributed architecture and enables using up to thousands of FPGAs to accelerate a single service.

In this paper, we examine the following question:

Can FPGAs be viewed and realized in the cloud as general purpose programmable resources that can be re-configured as on-demand devices?

There is a growing interest today into whether FPGA resources may be shared among multiple tenants and their applications, as opposed to the "all-or-nothing" philosophy where a single tenant has complete control over the FPGA [6]. It is interesting to note that the benefits of such sharing, which includes maximal resource utilization, are already being realized in the GPU domain. While GPUs were traditionally limited to only one user/tenant per host, a paradigm shift is occurring with Nvidia providing hardware support for multi-tenancy in its latest Kepler architecture GPU [1].

Very recently, Amazon has announced the addition of Xilinx FPGAs to their cloud services [8], signaling that major commercial institutions are seeing market demand in realizing FPGAs as general purpose shared computing resources in the cloud. In this work, we focus on the security challenges that arise when an FPGA accelerates multiple workloads from mutually distrusting tenants, and possible techniques to mitigate such challenges.

Security Challenges. Provisioning shared FPGAs on the cloud offers a number of challenges such as resource abstraction, ecosystem compatibility (libraries and SDKs) and, most importantly, *security*. While some of these challenges have been addressed comprehensively in the existing literature [6], security issues emerging from such a model are largely under-studied. One such security issue is *IP protection*. Multiple mutually distrusting tenants sharing a common pool of FPGA resources are likely to demand guarantees for bitstream confidentiality. Since FPGAs are inherently designed for single party access, FPGA vendors today focus on ensuring the privacy of bitstreams originating from *single users*, especially when deployed into hostile industrial/military environments.

Mitigation techniques typically used include bitstream encryption and authentication, combined with fault-tolerance. However, a direct adoption of such techniques in the multi-tenancy setting potentially blows up resource requirements, imposes significant key-management overheads, and leads to an overall lack of scalability.

In particular, a simpler solution based on traditional public key encryption would incur significant storage overheads since the secret key corresponding to each partition would have to be stored separately and securely. This motivates the need for dedicated and scalable security solutions tuned to the multi-tenancy setting where the storage required *does not grow* linearly with the number of partitions.

Existing Solutions. While a number of recent works [18,27,34] have helped develop general acceptance for FPGAs as general-purpose computing elements in portable ecosystems, security concerns regarding large-scale FPGA deployment have been discussed only in the context of specific applications. For example, the authors of [2] have looked into the security of specific applications such as building databases, where FPGAs are used as accelerators. Their security discussions are more at the application-level rather than the system-level.

Other works [6] focus on the threats originating from malicious tenants either crashing the system or attempting illegal memory accesses. Their proposed mitigations are mostly based on virtualization, in the sense that they use dedicated hypervisors and DMA units to regulate the memory access made by each tenant's bitstream file on the host FPGA node. However, they do not consider the threats posed by *co-resident VM attacks* [11,31], where data resident on a target VM can be stolen by a second malicious VM, so long as they co-exist on the same physical node. This poses a massive threat to IP security in the shared tenancy setting, and underlines the need for cryptographic security guarantees in addition to architectural barricading.

While a number of cryptographic solutions have been proposed for IP protection in the single tenancy scenario [10,16], there exist no equivalent solutions tuned to the shared tenancy setting to the best of our knowledge.

1.1 Our Contributions

In this paper, we propose a dedicated and scalable framework for secure multi-tenant FPGA provisioning on the cloud. Our framework also has following desirable features:

- Our framework guarantees bitstream confiden tiality in exchange for a constant amount of resource/memory overhead, *irrespective of the number of tenants sharing a given FPGA*. We achieve this using a novel technique known as *key-aggregation* that is provably secure under well-studied cryptographic assumptions.
- The only trusted agent in our framework is the FPGA vendor. Note that even in IP protection solutions in the single tenancy setting, the FPGA vendor is typically a trusted entity. Hence, this is a reasonable assumption. More importantly, the cloud service provider need not be trusted, which is desirable from a tenant's point of view.

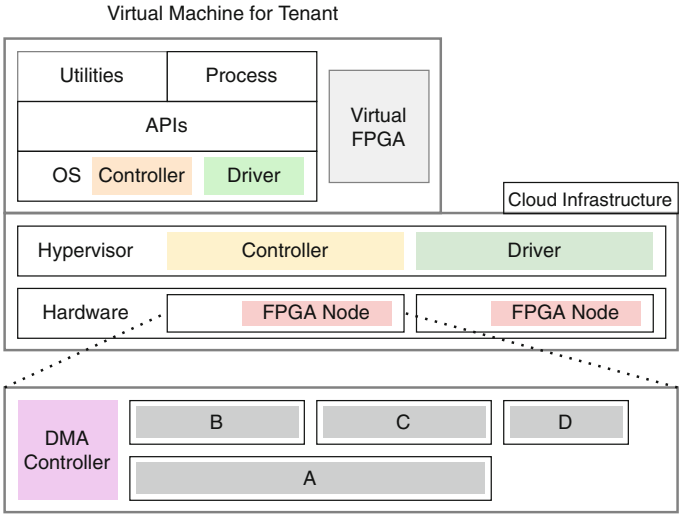

Fig. 1. FPGA provisioning on a cloud [6]

– Our framework can be easily integrated into existing cloud-based infrastructures such as OpenStack, and does not interfere with other desirable properties of an FPGA provisioning mechanism, such as resource virtualization/isolation and platform compatibility.

Prototype Implementation. We illustrate the scalability of our proposed approach via a prototype implementation on Xilinx Virtex-7 FPGAs. Our results indicate that the proposed approach has a fixed overhead of around $5-8\%$ of the available FPGA resources. This overhead remains unaltered for any number of tenants/workloads using the FPGA resources at any given point of time. Note the choice of the Virtex-7 FPGA family for our prototype is only for benchmarking, and may be extended to other FPGA vendors/families.

Applications in the Automotive Setting. FPGAs are being increasingly used as accelerators in automotive applications. In particular, the high parallel processing capabilities of FPGAs provide great advantages in applications such as ADAS, Smart Park Assist systems, and power control systems in modern vehicles. Most FPGAs also come with integrated peripheral cores that implement commonly-used functions like communication over controller area network (CAN) [12]. In an automotive setting, a single FPGA may be required to accelerate applications from multiple stakeholders, that are mutually distrusting and wish to protect their individual IPs. The core techniques underlying our proposed framework in this paper can be equivalently applied to build efficient and scalable IP protection units for such applications.

2 Secure Multi-tenant FPGA Provisioning: Our Proposition

In this section, we present our proposal for secure provisioning of FPGAs among multiple tenants on the cloud. We assume a basic FPGA provisioning setup on a cloud [6], as illustrated in Fig. 1. The idea is to abstract the FPGA resources to the client as an *accelerator pool*. Each FPGA is divided into multiple accelerator slots (e.g. A, B, C and D in Fig. 1), with one or more slots assigned to a tenant. The dynamic partial reconfiguration mechanism of modern FPGAs allows a tenant to view each such slot as a *virtual FPGA*, with specific resource types, available capacity and compatible interfaces. The DMA controller module is meant primarily for bandwidth and priority management across the various FPGA partitions. At the hypervisor layer, the controller module chooses available FPGA nodes based on their compatibility with a tenant's requirements, and helps configure them with the desired bitstream file via the service layer. The tenant essentially sees a VM, embedded with a virtual FPGA and containing the necessary APIs and controller modules to configure the FPGA. The allocation of resources to various tenants and the creation of corresponding VMs is handled by a separate controller module. More details of this basic setup can be found in [6]. Our aim is to propose an efficient and secure mechanism that ensures IP protection in this setup, without compromising on the other well-established features such as virtualization, inter-VM isolation and platform compatibility.

2.1 Bring Your Own Keys (BYOK)

The fundamental idea underlying our security proposal is as follows: each tenant encrypts her bitstream using a secret-key *of her own choice* before configuring the virtual FPGA with the same. Since bitstreams would potentially be encrypted in bulk, a symmetric-key encryption algorithm such as AES-128 is the ideal choice in this regard. Note that this approach immediately assures bitstream confidentiality. In particular, since neither the service provider nor any malicious agent can correctly guess the key chosen by a tenant (except with negligible probability), they can no longer gain access to her bitstream.

Notwithstanding its apparent benefits, the aforementioned BYOK-based bitstream encryption technique poses two major challenges in the shared FPGA setting - synchronizing bitstream encryption and decryption for different tenants, and efficient key-management. The main novelty of our proposal is in the application of *key-aggregation* [29] - a provably secure cryptographic technique - to efficiently solve both these challenges. We begin by providing a brief overview of a key-aggregate cryptosystem (KAC), along with a concrete construction for the same. We then demonstrate how KAC solves the key-management and synchronization challenges posed by the BYOK-based approach.

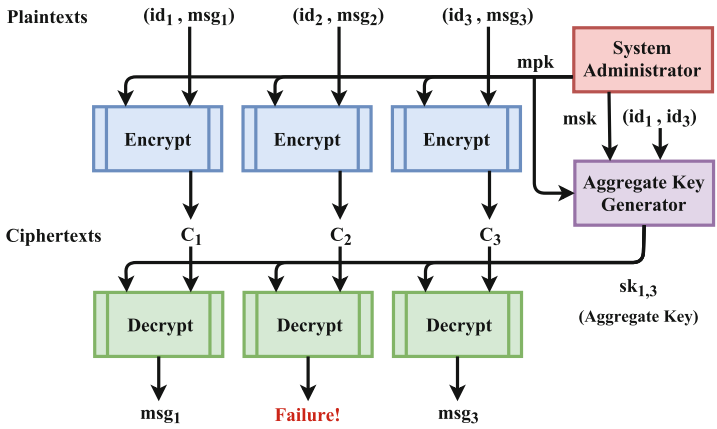

Fig. 2. An illustration of KAC over three entities

2.2 Key-Aggregate Cryptosystems (KAC)

KAC is a public-key mechanism to encapsulate multiple decryption-keys corresponding to an arbitrarily large number of independently encrypted entities into a single constant-sized entity. In a KAC, each plaintext message/entity is associated with a unique identity id, and is encrypted using a common master public-key mpk, generated by the system administrator. The system administrator also generates a master secret-key msk, which in turn is used to generate decryption keys for various entities. The main advantage of KAC is its ability to generate constant-size *aggregate decryption keys*, that combine the power of several individual decryption keys. In other words, given ciphertexts $C_1, C_2 \cdots, C_n$ corresponding to identities $\mathsf{id}_1, \mathsf{id}_2, \cdots, \mathsf{id}_n$, it is possible to generate a constant-size aggregate decryption key $\mathsf{sk}_{\mathcal{S}}$ for any arbitrary subset of identities $\mathcal{S} \subseteq \{\mathsf{id}_1, \cdots, \mathsf{id}_n\}$. In addition, the aggregate key $\mathsf{sk}_{\mathcal{S}}$ *cannot* be used to decrypt any ciphertext C_j corresponding to an identity $\mathsf{id}_j \notin \mathcal{S}$. Figure 2 illustrates the concept of a KAC scheme with a simple toy example. Observe that the individual secret-keys sk_1 and sk_3 for the identities id_1 and id_3 are compressed into a single aggregate-key $\mathsf{sk}_{1,3}$, that can be used to decrypt both the ciphertexts C_1 and C_3, *but not* C_2. Additionally, $\mathsf{sk}_{1,3}$ has the same size as either of sk_1 and sk_3, individually.

2.3 A Concrete KAC Construction on Elliptic Curves

Algorithm 1 briefly describes a provably secure construction for KAC to illustrate its key-aggregation property. The main mathematical structure used by the construction is a prime order sub-group of elliptic curve points $\mathbb{G}$, generated by a point P, and a bilinear map e that maps pairs of elements in $\mathbb{G}$ to a unique element in another group $\mathbb{G}_T$. The construction supports a maximum of

Algorithm 1. A Concrete KAC construction on Elliptic Curves

procedure KAC.Setup(n)
  Take as input the number of entities n
  Let P be an elliptic curve point of prime order q that generates a group $\mathbb{G}$ with a bilinear map $e : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$.
  Randomly choose α, γ in the range $[0, q-1]$ and output the following:

$$\mathsf{mpk} = \left(\{ \left[\alpha^j\right] P \}_{j \in [0,n] \cup [n+2, 2n]}, [\gamma] P \right)$$
$$\mathsf{msk} = \gamma$$

end procedure
procedure KAC.Encrypt(mpk, i, M)
  Take as input the master public key mpk, an entity identity $i \in [1, n]$ and a plaintext bitstream M.
  Randomly choose r in the range [0,q-1] and set:

$$c_0 = [r] P$$
$$c_1 = [r] \left([\gamma] P + \left[\alpha^i\right] P \right)$$
$$c_2 = M \oplus H \left(e \left(\left[\alpha^1\right] P, [\alpha^n] P \right)^r \right)$$

  where H is a collision-resistant hash function and $\oplus$ denotes the bit-wise XOR operation
  Output the ciphertext $C = (c_0, c_1, c_2)$
end procedure
procedure KAC.AggregateKey($\mathsf{msk}, \mathsf{mpk}, \mathcal{S}$)
  Take as input the master secret key $\mathsf{msk} = \gamma$, the master public key mpk and a subset of entities $\mathcal{S} \subseteq [1, n]$.
  Compute $a_{\mathcal{S}} = \sum_{j \in \mathcal{S}} \left[\alpha^{n+1-j}\right] P$
  Output the aggregate key $\mathsf{sk}_{\mathcal{S}} = [\gamma] a_{\mathcal{S}}$
  Also output $a_{\mathcal{S}}$ and $b_{i,\mathcal{S}} = \sum_{j \in \mathcal{S} \setminus \{i\}} \left[\alpha^{n+1-j+i}\right] P$ for each $i \in \mathcal{S}$
end procedure
procedure KAC.Decrypt($\mathsf{sk}_{\mathcal{S}}, a_{\mathcal{S}}, b_{i,\mathcal{S}}, C$)
  Take as input a ciphertext $C = (c_0, c_1, c_2)$ corresponding to an entity with identity i, an aggregate key $\mathsf{sk}_{\mathcal{S}}$ such that $i \in \mathcal{S}$, along with $a_{\mathcal{S}}$ and $b_{i,\mathcal{S}}$ as defined above.
  Output the decrypted message M as:

$$M = c_2 \oplus H \left(e \left(a_{\mathcal{S}}, c_1 \right) \cdot e \left(\mathsf{sk}_{\mathcal{S}} + b_{i,\mathcal{S}}, c_0 \right)^{-1} \right) \quad (1)$$

  where H is the same collision-resistant hash function as used in KAC.Encrypt
end procedure

n entities, and is provably secure against chosen-plaintext-attacks under a variant of the bilinear Diffie-Hellman assumption [13]. We refer the reader to [29] for more details on the correctness and security of the construction. Note that the notations $P_1 + P_2$ and $[a]P$ denote point addition and scalar multiplication operations, respectively, over all elliptic curve points P, P_1, P_2 and all scalars a.

Observe that the aggregate key $\mathsf{sk}_{\mathcal{S}}$ is a *single elliptic-curve point* (with a fixed representation size), irrespective of the size of the subset $\mathcal{S}$.

2.4 Combining BYOK with KAC

The crux of our proposal lies in combining BYOK with KAC for efficient key-management and synchronization of bitstream encryption-decryption. We achieve this via the following three-step proposal:

Step-1: Setup. In this step, the FPGA vendor sets up a KAC system by generating a master public key and a master secret key. Without loss of generality, we assume that each manufactured FPGA can be divided into a maximum of n accelerator partitions, where each partition is associated with a unique partition identity id, and represents an independent virtual FPGA from the tenant point of view. A dedicated mapping between the virtual FPGA id and the corresponding partition id may be suitably defined; we avoid details of the same for simplicity. Each FPGA contains a KAC decryption engine, that is pre-programmed to use a single aggregate decryption key $sk_{\mathcal{S}}$ corresponding to the subset $\mathcal{S}$ of partition ids it hosts. As already mentioned, the KAC aggregate key is a constant-sized entity (typically 256-320 bits), and can be securely stored in either a dedicated non-volatile RAM, or in the eFUSE[1] of certain advanced FPGA families such as Xilinx Virtex-7.

Step-2: Bitstream Encryption. In keeping with the idea behind BYOK, each tenant encrypts her bitstream using her own custom AES-128 key. This may be done using commercially available software tools such as Xilinx Vivado. In our proposal, this functionality is augmented to additionally encrypt the AES-128 key using the master public key of the KAC. The second encryption is performed under the identity id of the partition assigned to the tenant.

Step-3: Bitstream Decryption. Bitstream encryption occurs on-chip in two steps. Each FPGA is provided with a single KAC decryption core, while each individual partition is provided with its own AES-128 decryption core. The KAC decryption engine is first used to recover the AES-128 key chosen by the tenant. Since a single tenant is expected to use the same AES-128 key in a given session, the KAC decryption core needs to be invoked only once per tenant. The recovered key is subsequently used to decrypt any number of encrypted bitstreams and program the FPGA partition with the same.

[1] https://www.xilinx.com/support/documentation/application_notes/xapp1239-fpga-bitstream-encryption.pdf.

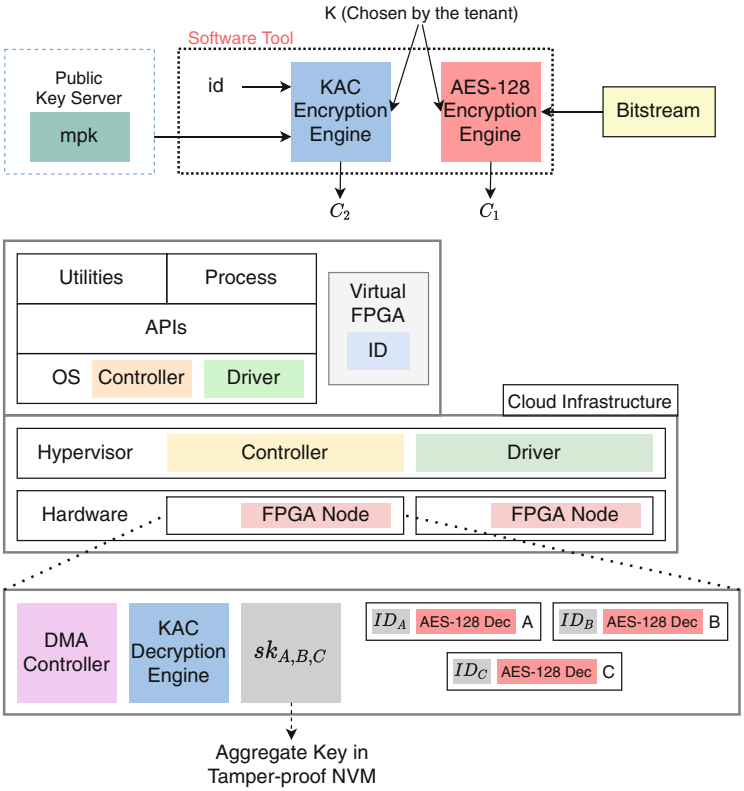

Fig. 3. Secure FPGA provisioning scheme: combining KAC with BYOK

3 Analysis of Our Proposal

3.1 Threat Models and Security

Our threat model considers the following potential IP stealing scenarios:

- **Malicious External Agents:** In this threat model, malicious external agents attempt to gain unauthorized access to a tenant's bitstream.
- **Malicious Co-tenants:** In this threat model, malicious co-tenants collude to try and expose the bitstream of an honest tenant sharing the same FPGA.
- **Malicious Service Provider:** In this threat model, the service provider itself attempts to steal the IP of a tenant (Fig. 3).

Note that in our proposal, bitstream decryption happens entirely on-chip. Hence, IP stealing is possible only if a malicious adversary gains access to the AES-128 key chosen by the tenant to encrypt her bitstream. This key is not known apriori to any third party, including the service provider. The AES-128 key is in turn encrypted using KAC, which is provably CPA secure [29], implying

Algorithm 2. Secure Multi-Tenant FPGA Provisioning

procedure INITIAL SETUP (FPGA VENDOR)
  $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{KAC.Setup}$
  Publish the master public key mpk for the KAC scheme
  for each FPGA **do**
    Partition the FPGA into accelerator slots (the maximum number of such slots may be pre-defined depending on the FPGA family)
    for each FPGA partition **do**
      Assign a unique random identity id to the partition
    end for
    Let $\mathcal{S}$ denote the set of all id-s corresponding to partitions on the same FPGA
    $(\mathsf{sk}_{\mathcal{S}}, a_{\mathcal{S}}, \{b_{\mathsf{id},\mathcal{S}}\}_{\mathsf{id}\in\mathcal{S}}) \leftarrow \mathsf{KAC.AggregateKey}(\mathsf{msk}, \mathcal{S})$
    Embed $\mathsf{sk}_{\mathcal{S}}$ in a tamper-proof non-volatile memory segment on the FPGA
    Embed $a_{\mathcal{S}}$ in a non-volatile memory segment on the FPGA (*need not be secure/tamper-proof*).
    Embed each $b_{\mathsf{id},\mathcal{S}}$ in a non-volatile memory segment of the partition with identity id (*again need not be secure/tamper proof*)
  end for
  Each FPGA is provisioned with a single KAC decryption engine, while each FPGA partition is provisioned with its own AES-128 decryption engine.
end procedure
procedure BITSTREAM ENCRYPTION($\mathsf{Bitstream}$)
  Suppose a tenant is assigned an FPGA partition with identity id.
  $K \leftarrow \mathsf{AES.KeyGen}$
  $C_1 \leftarrow \mathsf{AES.Encrypt}(K, \mathsf{Bitstream})$
  $C_2 \leftarrow \mathsf{KAC.Encrypt}(\mathsf{mpk}, \mathsf{id}, K)$
  Submit (C_1, C_2) to the framework for configuring the FPGA partition.
end procedure
procedure BITSTREAM DECRYPTION(C_1, C_2)
  $K \leftarrow \mathsf{KAC.Decrypt}(\mathsf{sk}_{\mathcal{S}}, C_2)$
  $\mathsf{Bitstream} \leftarrow \mathsf{AES.Decrypt}(K, C_1)$
end procedure

that the encrypted key cannot be accessed without the appropriate aggregate decryption key for the KAC scheme. Since this key is constant-overhead and is stored in dedicated secure storage, an external adversary cannot gain access to the aggregate decryption key. Hence, our proposal ensures that neither an external adversary nor the service provider can steal the IP of a tenant in the shared FPGA setting.

As mentioned, we also consider threats from malicious co-tenants, who could collude to try and expose the IP of an honest tenant sharing the same FPGA. Note that the malicious tenants have access to the aggregate key(s) corresponding to their own accelerator partitions, and not the aggregate key corresponding to the partitions being used by the honest tenant. However, the KAC scheme is provably collusion resistant against an unbounded number of malicious parties [29], implying that, irrespective of the number of malicious tenants colluding,

the aggregate decryption key of an honest tenant cannot be exposed. Thus, our proposal guarantees IP security in the malicious co-tenant setting.

Note that, some recent works on side-channel attacks [20,30] target multi-tenant FPGA systems. Currently, our proposal provides a secure solution for multi-tenant FPGA use on the cloud which focuses on the performance and storage (resource) efficiency, but not security against implementation-level attacks. We shall consider the possible applicability of these attacks against the proposed scheme and necessary countermeasures (if needed) in our future works.

3.2 Performance and Efficiency

Our proposal has the following desirable features from the point of view of efficiency as well as security.

Constant Secure Storage Overhead per FPGA: Each FPGA stores a single aggregate decryption key that suffices for all its partitions. As already mentioned, KAC generates constant-overhead aggregate-keys irrespective of the number of entities they correspond to. Hence, the memory requirement per FPGA for secure key storage remains the same irrespective of the maximum number of partitions n. In other words, the framework scales to any arbitrarily large n without incurring any additional overhead for secure key storage.

Constant Encryption and Decryption Latency: The encryption and decryption latencies for both KAC and AES-128 are constant, and independent of the maximum number of partitions n supported by an FPGA. In particular, the encryption and decryption sub-routines in the KAC scheme of [29] involve a constant number of elliptic curve operations, and hence require a constant amount of time.

No Leakage to the Cloud Service Provider: The new scheme achieves synchronization between the encryption and decryption engines via a public-key mechanism that is set up by the FPGA vendor. Since the entire bitstream decryption happens on-chip, the confidentiality of the bitstream as well as that of the AES-128 key from the cloud service provider (as well as any external malicious agents) are guaranteed by the security of AES-128 and the CPA security of the KAC scheme, respectively.

4 Prototype Implementation

In this section, we present a prototype implementation of our proposed protocol on the Xilinx Virtex-7 family of FPGAs. In particular, we focus on the overhead and performance results for the security-related components, namely KAC and AES-128. The results are presented in two parts. The first part focuses on the on-chip decryption engines, while the second part focuses on the software tool for generating the encrypted bitstreams and encrypted AES-128 keys.

Table 1. Implementation Details: Elliptic Curve Operations, Optimal-Ate Pairing and AES-128

Elliptic Curve Operations	**Module/ Algorithm**	**#Clock Cycles**	**Operating Frequency (in MHz)**	**Latency (in ms)**
	Point Addition	705	180.505	3.905×10^{-3}
	Point Doubling	528		2.925×10^{-3}
	Scalar Multiplication	279552		1.548
Opt-Ate Pairing Operations	**Module/ Algorithm**	**#Clock Cycles**	**Operating Frequency (MHz)**	**Latency (ms)**
	Miller's Loop	1669941	180.505	9.252
	Final Exponentiation	882403		4.888
AES-128 Operations	**Module/ Algorithm**	**#Clock Cycles**	**Operating Frequency (MHz)**	**Latency (ms)**
	Encryption/ Decryption	10	180.505	5.54×10^{-5}

4.1 On-Chip Decryption Engines

The two main components of our prototype implementation are the AES-128 and KAC decryption engines. To implement the AES-128 decryption core, we adopt a distributed look-up table (LUT)-based approach [23] for efficient and low-latency implementations. While other benchmark implementations for AES-128 using composite field based approaches (such as polynomial, normal, redundant and mixed bases) are well-known [32,35], distributed LUT-based approach is especially tuned to FPGA-based implementations, and is hence chosen.

The KAC Decryption Engine. The KAC decryption engine, adopted from Algorithm 1 [29], requires the implementation of an elliptic curve core that also supports bilinear pairing operations, such as Tate and Optimal-Ate pairing [7,9,36]. Since pairing-friendly elliptic curves with small characteristics [7,17,26] have recently been analyzed as vulnerable to DLP attacks [14,15], we choose an elliptic curve with a 256-bit characteristic prime from the family of pairing-friendly Barreto-Naehrig (BN) curves [4]. All elliptic-curve operations, including point addition, and scalar multiplication are implemented using a Montgomery ladder [5] for constant-time operations. Other constant-time approaches such as window-non-adjacent form (w-NAF) [22] may also be used. We used Miller's algorithm [24] for Optimal-Ate pairing computation with combined point operations and line function evaluation to reduce computation time.

Table 2. Overhead Comparison for Comparable Throughput: BYOK + Authenticated Key Exchange v/s BYOK + KAC for 10 partitions

Bitstream Decryption Methodology	**Resource Consumption**					**Secure Storage Requirement(in Kb) (10 partitions)**
	#Slices (%)	#LUTs (%)	#Registers (%)	#DSPs (%)	#BRAMs (%)	
BYOK+Auth. DHKE (Naive Approach)	32880 (30.30)	70380 (16.24)	85848 (9.90)	400 (11.11)	200 (13.60)	3.84
BYOK+KAC (Our Proposal)	5330 (4.91)	12747 (2.94)	11839 (1.36)	40 (1.11)	20 (1.36)	0.256

We would like to point out that the choice of curve for our prototype implementation is only for demonstration; our proposal may be easily adopted and implemented efficiently using standard curve choices, including Hessian, Edwards, NIST and Koblitz curves [3,19,33], subject to the restriction that the characteristics of these chosen curves is large.

Use of DSP Blocks. A novel feature of our implementation is the use of DSP blocks to design efficient prime-field multipliers, which in turn are used in the elliptic curve-based operational modules. Modern FPGAs such as the Xilinx Virtex-7 are inherently equipped with numerous DSP blocks, which can be used to design low-latency circuits for arithmetic operations. We exploited this fact to design a high-speed prime field multiplier, that optimally uses these DSP blocks based on an efficient high-radix Montgomery algorithm [25] for modular multiplication. The post-route timing reports for the elliptic curve operations and the AES-128 operations are summarized in Table 1. Although we have chosen the Xilinx Virtex-7 FPGA (xc7vx690tffg1761) family for the prototype implementation, our implementation may be readily ported to other FPGA families with minimal effort.

Comparison with Naïve Approach. In Table 2, we compare the resource overhead of our approach with an alternative naïve approach where each tenant chooses her own key and exchanges the same via a secure and authenticated Diffie-Hellman key exchange (DHKE) protocol. The second approach requires elliptic curve scalar multiplication operations, that required dedicated scalar multiplication units per FPGA partition. For the purpose of comparison, we assume a total of 10 partitions on a given FPGA. Quite evidently, for a comparable throughput, the area requirement of our proposed approach (BYOK+KAC) is significantly more resource-thrifty, which may be attributed to the constant-size aggregate key generation feature of the KAC scheme.

4.2 Software Encryption Engine

The software encryption engine in our prototype implementation allows a tenant to encrypt her bitstream using an AES-128 key of her own choice, and subsequently, encrypt this key under the KAC scheme. As mentioned previously,

Table 3. Implementation Details: KAC Encryption Engine

Operation	Point addition	Point doubling	Pairing	KAC encryption
Latency (ms)	1.351×10^{-2}	1.098×10^{-2}	82.025	104.333

BYOK-based bitstream encryption can be readily availed using commercial design tools such as Xilinx Vivado.

We implemented the KAC encryption engine in software using the open-source Pairing-Based Cryptography (PBC) library[2], that provides APIs to compute pairings over the BN family of elliptic curves. The only pre-requisite for using the PBC library is the open-source GNU Multiple Precision Arithmetic Library[3] (GMP). The PBC library works on a variety of operating systems, including Linux, Mac OS, and Windows (32 and 64 bits).

It is important to note that similar to the decryption operation, the latency for KAC encryption is also independent of the number of partitions a given FPGA can support.

We present implementation results for the KAC encryption engine using the PBC library in Table 3. The target platform is a standard desktop computer, with an Intel Core i5-4570 CPU, 3.8 GB RAM, and an operating frequency of 3.20GHz. Use of PBC is primarily for demonstrating the utility of the framework. Modern libraries like RELIC[4] or MIRACL[5] may be used for better performance.

5 Scalability of Our Framework

In order to elucidate the scalability of our proposed framework, we demonstrate how the following parameters of our prototype implementation scale with the maximum number of tenants/partitions per FPGA:

Secure Storage on FPGA. In Fig. 4, we compare the amount of secure key storage required per FPGA in our proposed framework (combining KAC with BYOK) against a framework that simply uses BYOK with authenticated Diffie-Hellman key exchange. The latter scheme would require to store the AES-128 key for every tenant on the corresponding FPGA partition allocated to her. Naturally, the storage requirement grows with the number of partitions that a given FPGA can support. In our proposition, the aggregation capability of KAC ensures that the tamper-resistant non-volatile storage requirement is independent of number of partitions that a given FPGA can support. In other words, our FPGA provisioning scheme has a far superior scalability in terms of secure key storage, as compared to a simple BYOK-based provisioning scheme.

[2] https://crypto.stanford.edu/pbc/.
https://crysp.uwaterloo.ca/software/PBCWrapper/.
[3] https://gmplib.org/.
[4] https://github.com/relic-toolkit/relic.
[5] https://github.com/miracl/MIRACL.

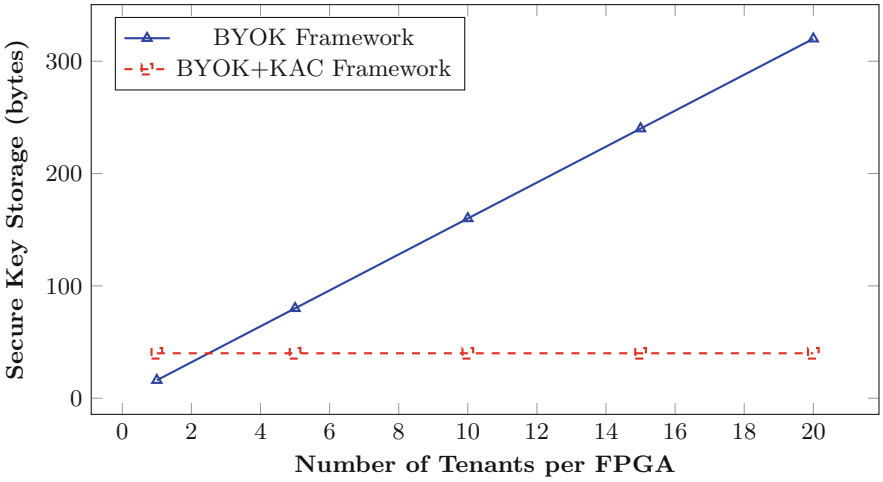

Fig. 4. On-chip secure-storage requirements

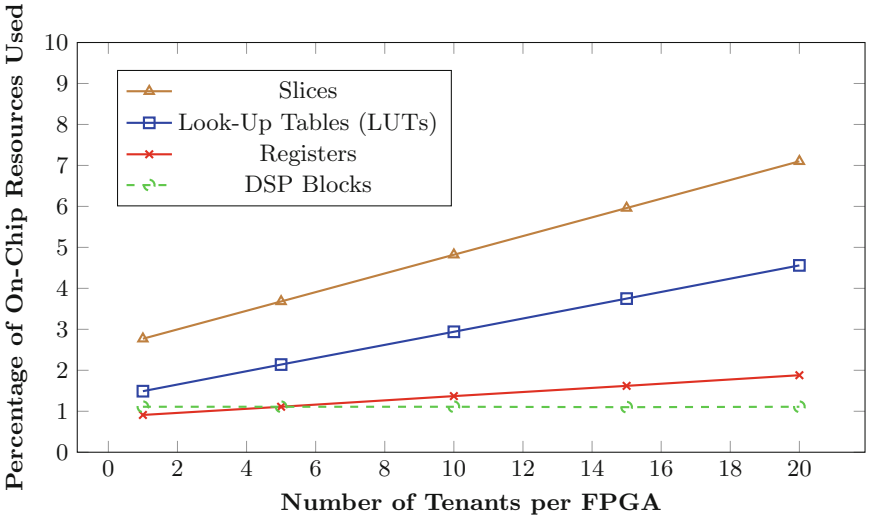

Fig. 5. On-chip resource requirements for BYOK + KAC

On-Chip Resource Overhead. Since our framework requires only a single KAC decryption engine per FPGA, the on-chip resource overhead remains almost constant with respect to the number of partitions that a given FPGA can support. This is illustrated in Fig. 5. The only slight increase is due to the presence of an AES-decryption engine in every FPGA partition. The overall on-chip resource overhead for up to 20 partitions on a single FPGA device is less than 5% for LUTs and is less than 7% for Slices, which highlights the scalability of our proposal.

Bitstream Encryption/Decryption Performance. Finally, as already mentioned, the bitstream encryption/decryption latency (both KAC and AES-128) of our framework is independent of the number of partitions that a given FPGA can support.

In summary, the incorporation of KAC plays a crucial role in ensuring that our framework retains the same levels of performance and efficiency for arbitrarily large number of tenants sharing a single FPGA node. To the best of our knowledge, this is the first FPGA provisioning framework to be prototyped that achieves a desirable balance between security and scalability in the multi-tenancy setting.

6 Conclusion

In this paper, we proposed a dedicated and scalable framework for secure multi-tenant FPGA provisioning on the cloud. Our framework guarantees bitstream confidentiality with constant amount of resource/memory overhead, *irrespective of the number of tenants sharing a given FPGA*. We achieved this using a novel technique known as *key-aggregation* that is provably secure under well-studied cryptographic assumptions. Our framework can be easily integrated into existing cloud-based infrastructures such as OpenStack, and does not interfere with other desirable properties of an FPGA provisioning mechanism, such as resource virtualization/isolation and platform compatibility. We illustrated the scalability of our proposed approach via a prototype implementation on Xilinx Virtex-7 FPGAs. Our results indicate that the proposed approach has a fixed overhead of less than 5% of the available FPGA resources (LUT). This overhead remains unaltered for any number of tenants/workloads using the FPGA resources at any given point of time.

References

1. Nvidia Inc. GRID GPUs
2. Arasu, A., Eguro, K., Kaushik, R., Kossmann, D., Ramamurthy, R., Venkatesan, R.: A secure coprocessor for database applications. In: 2013 23rd International Conference on Field Programmable Logic and Applications (FPL), pp. 1–8. IEEE (2013)
3. Azarderakhsh, R., Reyhani-Masoleh, A.: Efficient FPGA implementations of point multiplication on binary Edwards and generalized Hessian curves using Gaussian normal basis. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **20**(8), 1453–1466 (2012)
4. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006). https://doi.org/10.1007/11693383_22
5. Bernstein, D.J., Lange, T., Schwabe, P.: The security impact of a new cryptographic library. In: Hevia, A., Neven, G. (eds.) LATINCRYPT 2012. LNCS, vol. 7533, pp. 159–176. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33481-8_9

6. Chen, F., Shan, Y., Zhang, Y., Wang, Y., Franke, H., Chang, X., Wang, K.: Enabling FPGAs in the cloud. In: Proceedings of the 11th ACM Conference on Computing Frontiers, p. 3. ACMD (2014)
7. Duursma, I., Lee, H.S.: Tate pairing implementation for hyperelliptic curves ŷ 2= x̂ p-x+ d. In: ASIACRYPT, vol. 2894, pp. 111–123. Springer (2003). https://doi.org/10.1007/978-3-540-40061-5_7
8. Freund, K.: Amazon's Xilinx FPGA Cloud: Why This May Be A Significant Milestone (2016)
9. Frey, G., Muller, M., Ruck, H.G.: The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems. IEEE Trans. Inf. Theory **45**(5), 1717–1719 (1999)
10. Guajardo, J., Kumar, S.S., Schrijen, G.-J., Tuyls, P.: FPGA intrinsic PUFs and their use for IP protection. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 63–80. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74735-2_5
11. Irazoqui, G., Inci, M.S., Eisenbarth, T., Sunar, B.: Wait a minute! a fast, cross-VM attack on AES. In: Stavrou, A., Bos, H., Portokalidis, G. (eds.) RAID 2014. LNCS, vol. 8688, pp. 299–319. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11379-1_15
12. Johansson, K.H., Törngren, M., Nielsen, L.: Vehicle applications of controller area network. In: Hristu-Varsakelis, D., Levine, W.S. (eds.) Handbook of Networked and Embedded Control Systems, pp. 741–765 (2005). https://doi.org/10.1007/0-8176-4404-0_32
13. Joux, A.: A one round protocol for tripartite Diffie-Hellman. J. Cryptol. **17**(4), 263–276 (2004). https://doi.org/10.1007/s00145-004-0312-y
14. Joux, A., Pierrot, C.: Technical history of discrete logarithms in small characteristic finite fields - the road from subexponential to quasi-polynomial complexity. Des. Codes Cryptograph. **78**(1), 73–85 (2016). https://doi.org/10.1007/s10623-015-0147-6
15. Joux, A., Vitse, V.: Elliptic curve discrete logarithm problem over small degree extension fields. Application to the static Diffie-Hellman problem on E(Fq5) (2010)
16. Kean, T.: Cryptographic rights management of FPGA intellectual property cores. In: Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on FIeld-programmable Gate Arrays, pp. 113–118. ACM (2002)
17. Kerins, T., Marnane, W.P., Popovici, E.M., Barreto, P.S.L.M.: Efficient hardware for the tate pairing calculation in characteristic three. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 412–426. Springer, Heidelberg (2005). https://doi.org/10.1007/11545262_30
18. Kirchgessner, R., Stitt, G., George, A., Lam, H.: VirtualRC: a virtual FPGA platform for applications and tools portability. In: Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pp. 205–208. ACM (2012)
19. Koblitz, N.: Elliptic curve cryptosystems. Math. Comput. **48**(177), 203–209 (1987)
20. Krautter, J., Gnad, D.R., Schellenberg, F., Moradi, A., Tahoori, M.B.: Active fences against voltage-based side channels in multi-tenant FPGAS. IACR Cryptol. ePrint Arch 2019:1152 (2019)
21. Lie, W., Feng-Yan, W.: Dynamic partial reconfiguration in FPGAs. In: 2009 Third International Symposium on Intelligent Information Technology Application, IITA 2009, vol. 2, pp. 445–448. IEEE (2009)

22. Longa, P., Miri, A.: New Multibase Non-Adjacent Form Scalar Multiplication and its Application to Elliptic Curve Cryptosystems (extended version). IACR Cryptology ePrint Archive 2008:52 (2008)
23. McLoone, M., McCanny, J.V.: Rijndael FPGA implementations utilizing look-up tables. J. VLSI Signal Process. Syst. Signal Image Video Technol. **34**(3), 261–275 (2003)
24. Miller, V.S.: The Weil pairing, and its efficient calculation. J. Cryptol. **17**(4), 235–261 (2004)
25. Mukhopadhyay, D., Roy, D.B.: Revisiting FPGA implementation of montgomery multiplier in redundant number system for efficient ECC application in GF (p). In: 2018 28th International Conference on Field Programmable Logic and Applications (FPL), pp. 323–3233. IEEE (2018)
26. Oliveira, L.B., Aranha, D.F., Morais, E., Daguano, F., López, J., Dahab, R.: Tinytate: computing the Tate pairing in resource-constrained sensor nodes. In: Sixth IEEE International Symposium on Network Computing and Applications, 2007. NCA 2007, pp. 318–323, IEEE (2007)
27. Opitz, F., Sahak, E., Schwarz, B.: Accelerating distributed computing with FPGAs. Xcell J. **3**, 20–27 (2012)
28. Ovtcharov, K., Ruwase, O., Kim, J.Y., Fowers, J., Strauss, K., Chung, E.S.: Accelerating deep convolutional neural networks using specialized hardware. Microsoft Res. Whitepaper **2**(11), 1–4 (2015)
29. Patranabis, S., Shrivastava, Y., Mukhopadhyay, D.: Provably secure key-aggregate cryptosystems with broadcast aggregate keys for online data sharing on the cloud. IEEE Trans. Comput. **66**(5), 891–904 (2017)
30. Provelengios, G., Holcomb, D., Tessier, R.: Characterizing power distribution attacks in multi-user FPGA environments. In: 2019 29th International Conference on Field Programmable Logic and Applications (FPL), pp. 194–201. IEEE (2019)
31. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, pp. 199–212. ACM (2009)
32. Rudra, A., Dubey, P.K., Jutla, C.S., Kumar, V., Rao, J.R., Rohatgi, P.: Efficient Rijndael encryption implementation with composite field arithmetic. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 171–184. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44709-1_16
33. Shu, C., Gaj, K., El-Ghazawi, T.: Low latency elliptic curve cryptography accelerators for NIST curves over binary fields. In: Proceedings of the 2005 IEEE International Conference on. Field-Programmable Technology, 2005, pp. 309–310. IEEE (2005)
34. So, H.K.H., Brodersen, R.: A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH. ACM Trans. Embed. Comput. Syst. (TECS) **7**(2), 14 (2008)
35. Ueno, R., Homma, N., Sugawara, Y., Nogami, Y., Aoki, T.: Highly efficient GF (28) g f (2 8) inversion circuit based on redundant GF arithmetic and its application to AES design. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp. 63–80. Springer (2015). https://doi.org/10.1007/978-3-662-48324-4_4
36. Vercauteren, F.: Optimal pairings. IEEE Trans. Inf. Theory **56**(1), 455–461 (2009)