Secure acceleration on cloud-based FPGAs – FPGA enclaves

SUBMISSION DATE / POSTED DATE

18-12-2020 / 21-12-2020

CITATION

Lindskog, Niklas; Englund, Håkan (2020): Secure acceleration on cloud-based FPGAs – FPGA enclaves. TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.13408889.v1

DOI

10.36227/techrxiv.13408889.v1

Secure acceleration on cloud-based FPGAs - FPGA enclaves

Håkan Englund
Ericsson Research
Lund, Sweden
hakan.englund@ericsson.com

Niklas Lindskog
Ericsson Research
Lund, Sweden
niklas.lindskog@ericsson.com

***Abstract*—FPGAs are becoming a common sight in cloud environments and new usage paradigms, such as FPGA-as-a-Service, have emerged. This development poses a challenge to traditional FPGA security models, as these are assuming trust between the user and the hardware owner. Currently, the user cannot keep bitstream nor data protected from the hardware owner in an FPGA-as-a-service setting. This paper proposes a security model where the chip manufacturer takes the role of root-of-trust to remedy these security problems. We suggest that the chip manufacturer creates a Public Key Infrastructure (PKI), used for user bitstream protection and data encryption, on each device. The chip manufacturer, rather than the hardware owner, also controls certain security-related peripherals. This allows the user to take control over a predefined part of the programmable logic and set up a protected *enclave* area. Hence, all user data can be provided in encrypted form and only be revealed inside the enclave area. In addition, our model enables secure and concurrent multi-tenant usage of remote FPGAs. To also consider the needs of the hardware owner, our solution includes bitstream certification and affirming that uploaded bitstreams have been vetted against maliciousness.**

***Keywords*—Cloud security, Confidential computing, Enclaves, FPGA, Hardware security, System-on-chip**

I. Introduction and related work

Confidential computing, sometimes also referred to as secure remote computation, is the problem of deploying a calculation on a remote computer, with an untrusted owner and in an untrusted environment. The goal is to provide, to various levels, guarantees of confidentiality and integrity protection of the implementation and data for the computation. Intel Secure Guard eXtensions (SGX)[1] is an interesting solution in this area where shielded executions environments are enforced by the hardware. In this paper, we take inspiration from the SGX framework when we look at the problem of secure remote computation on FPGAs.

In recent years, Amazon [2] and other cloud providers have started to offer FPGA-as-a-service (FaaS). The rapid development for FPGA use cases has highlighted one difficulty of FaaS; the security measures which exist on contemporary FPGAs are not designed with this setup in mind. Anyone who wants to use FaaS must ultimately trust the hardware owner (HwO) and expose both FPGA configuration (bitstream) and data to this entity. In use cases involving e.g. medical records, the confidentiality of data is essential and potential exposure to HwO is unacceptable. In other cases, the bitstream is proprietary and needs to be protected. We solve this by introducing a new trust model which includes a third party, trusted by both HwO and user. Thus, the user of an FaaS device no longer has to trust the HwO. Using our model, the user does not need to expose data or bitstream to the HwO. Additionally, we enable the user to, before usage, ensure that the device is correctly programmed. Our model also takes FPGA-specific attacks, which uses its reprogrammable nature as an attack vector, into consideration. Such attacks can not only break the confidentiality and integrity of the data passing through the device but also temporarily or permanently damage the hardware, e.g. by intentionally drawing more current than the device was designed for (see [3]). Considering this, our model ensures that the HwO can trust that a bitstream deployed on an FaaS device has been vetted against maliciousness.

TABLE I
Comparison with prior work

	This paper	[4]	[5]
User data protected from HwO	✓	✓	✓
User bitstreams protected from HwO	✓	✓	✓
Attestation support	✓	✓*	✗
Does not require pre-installed logic	✓	✗	✓
Bitstream inspection support	✓	✗	✗
Enables multi-tenancy	✓	✗	✗

*Suggests user-initiated configuration readback of the user's circuit over a secure channel rather than SGX equivalent attestation.

Some related work exists in the field of FaaS security. Both Elrabaa et al. [4] and Eguru et al. [5] proposes a security model addressing FaaS where the user does not trust the HwO, but rather a trusted authority. We compare our model to prior work in Table I.

The contributions of this paper are as follows:

- A new security model where HwO and client data owner does not need to trust each other. This includes:
  - Removing HwO's possibility to control certain security-critical components.
  - Creation of secluded FPGA areas, *enclaves*, equipped with unique keys.
  - Attestation of enclave area state prior to usage.
- A novel solution enabling multi-tenancy on FPGAs.
- Enabling multiple usage scenarios, for an FaaS device, with different levels of security.

 DOI: 10.1109/IPDPSW50202.2020.00026

In this paper, we will also describe an implementation on a modern-day device. Next-generation devices from major FPGA vendors are moving towards heterogeneous computing [6], [7]. We have therefore chosen a diversified computing platform in the form of a System-on-chip (SoC) FPGA to test the practicality of our security model. These devices include a *processing subsystem*[1] which contains CPUs, power management and memory, among other components.

The outline for the rest of the paper is as follows. Background describing SGX and the security problems relating to FaaS are introduced in Section II. In Section III we present our solution to the challenges within FaaS security. We describe our trial implementation and the corresponding results in Section IV. We discuss our result and future undertakings in Section V.

II. Background

A. Intel Secure Guard eXtensions (SGX)

Intel Secure Guard eXtensions has received a large interest, especially in the research community. SGX aims to provide an execution environment in which applications, called *enclaves*, are bidirectionally shielded from both the OS and any other software running on the devices.

SGX is hardware-enforced, hence, the Trusted Computing Base (TCB) of SGX is provided by Intel, no matter who owns the device. This is a great foundation for simple trust models, even if operations are performed in a cloud environment.

SGX also supports attestation. Attestation is a procedure in which a local or remote party can establish that it is communicating with an enclave with a given identity; that the enclave has not been tampered with; that it is the intended enclave and it is running on Intel SGX enabled hardware.

B. FPGA as a service and security

As mentioned previously, FaaS is offered by several cloud providers. In this paper, the current involved parties we consider for the FaaS use case are:

- **User** - Pays to use the hardware and possibly bitstreams. Wants to use the hardware without exposing data.
- **Hardware owner (HwO)** - Owns the hardware and rents it to customers.
- **Chip manufacturer (CM)** - Manufactures or oversees manufacturing of the device.
- **Bitstream creator (BCr)** - Designs accelerators, synthesizes, routes, places, and creates bitstreams.

In this paper, we want to introduce two additional parties, presented below.

- **Bitstream Certifier (BCe)** - Inspects bitstreams and ensures that it does not contain vulnerabilities or dangerous constructs. Trusted by HwOs as well as BCr, the latter as the bitstream must be exposed to BCe.
- **Attestation Root of Trust (A-RoT)** - Acts as the root of trust for the enclave implementations. Trusted by Users and HwOs.

[1]Also known as hard processors.

In an FaaS setting, the FPGA can be described as two separate planes. The programmable plane, or "role" is the part of the device available to the end-user for deploying bitstreams and applications. The FPGA also contains a control plane, or "shell", which is controlled by HwO. It contains e.g. the bitstream programmer and security peripherals.

Development tools are provided by the HwO; this is partly a service to the user, but also a security measure. Controlling the tool-chain is a means to ensure that designs are not harming the device nor eavesdropping on the shell. While no commercial equivalent to virus scanners for FPGAs currently exist, design checks can be performed in the toolchain used for bitstream creation. For this reason, HwOs best maliciousness protection is to demand that all bitstreams are created with their toolchain, resulting in inevitable trust in the HwO.

Furthermore, FaaS solutions do not support remote attestation, nor true secure end-to-end communication. The latter as it not possible to hide any secrets or encryption keys from the HwO-controlled tool-chain.

Multi-tenancy is common in cloud environments where several users share the same physical resource by using virtual machines and containers. To our best understanding, none of the current FPGA-as-a-service providers are offering multi-tenancy or has announced any plans to do so in the near future. There are several reasons for this, one being that the current tool-chains and isolation possibilities in FPGAs have not been developed with multi-user environments in mind. Another problem is partitioning the device between the shell and the roles, as the tool-chains require all roles to have predetermined positions and resources boundaries.

III. Solution overview

To overcome the security issues raised in Section II-B, we propose a solution where we operate in our trust model, introduced in Section I. Our goal is to let the user setup a secure communication channel with a trustworthy device, in an expected state, without having to blindly trust the HwO. The user provides its accelerator as an encrypted partial bitstream, decrypted on the device but outside the reach of the HwO. All bitstreams must have been certified by a trusted third party prior to upload, thus removing the need for the HwO to inspect bitstreams to ensure that they are not malicious.

The solution can be divided into two categories, framework establishment and confidential computing procedure. We present each category below.

A. Framework establishment

There are three globally trusted asymmetric private keys in our trust model:

- $\text{Priv}_{\text{BitSK}}$, owned by bitstream certifier (BCe), used to provide bitstream certification.
- $\text{Priv}_{\text{BootSK}}$, owned by the attestation trust anchor (A-RoT), used to sign software required to securely start the Shell providing the root of trust for attestation ($\text{Shell}_{\text{A-RoT}}$).
- Priv_{EK}, owned by A-RoT, used to bootstrap devices by endorsing device unique keys.

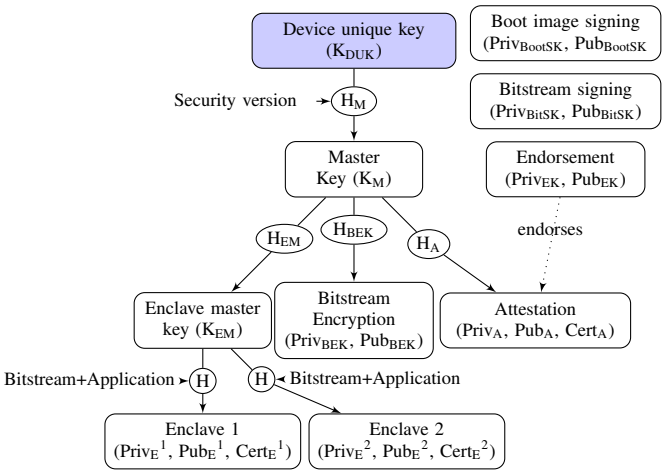

Fig. 1. Key hierarchy, highlighted key remains under A-RoT control.

On each FaaS device, a device master key (K_M), is deterministically generated from a device unique key (K_{DUK}), e.g., generated by a PUF. Using K_M, the device is able to derive different subkeys for bitstream encryption ($Priv_{BEK}$), attestation ($Priv_A$)[2], and enclave key derivation master key (K_{EM}) respectively. The keys are created using differently seeded key derivation functions (denoted H in Fig. 1)[3].

In order to establish the identity of a device on which one wishes to deploy a bitstream, the device must be bootstrapped and introduced into the PKI by the attestation trust anchor (A-RoT). The device utilizes its attestation private key ($Priv_A$) for bootstrapping and receives a device endorsement certificate ($Cert_A$) from A-RoT during manufacturing. The key hierarchy is summarized in Fig. 1.

As previously mentioned, contemporary tool chains demand that roles, i.e. partial reconfiguration (PR) regions, are predefined and do not overlap. This requires the FPGA to be configured with a static bitstream (BS_{shell}) that has isolated PR-regions adequate for enclave use. To guarantee this, we suggest that the HwO has several pre-approved shell bitstreams, each of which can include one or several PR-regions.

When a security flaw in the framework is found, the security version is updated in order to revoke all device keys. This results in a new master key, which also invalidates certificates (or rather makes the associated private key unavailable). Hence, a new attestation key pair needs to be derived, endorsed and a new certificate created.

To get past the current problem of the HwO-owned shell managing the entire control plane, the shell is divided into $Shell_{HwO}$ and $Shell_{A\text{-}RoT}$. The trusted computing base (TCB) of our solution is constituted by $Shell_{A\text{-}RoT}$ which is responsible for protection of: device unique key, JTAG, secure boot, persistent storage & bitstream loader.

Due to the risk of malicious or poorly configured bitstreams damaging the FPGA, the HwO must be able to trust every bitstream entering the device. A mutually trusted third party, i.e. the bitstream certifier (BCe), inspects the bitstream to minimize the risk that the bitstream does contain any maliciousness. The third-party inspection allows the bitstream to be sent encrypted and thereby keeping it secret, apart from exposing it to the bitstream certifier (BCe). After inspection, BCe provides a bitstream signature using $Priv_{BitSK}$.

B. Confidential computing procedure

To set up a remote session with confidential computation, the user needs to verify the device's identity ($Cert_A$) as well as its configuration. Reconfigurable regions in FPGAs are today very much dependent on the size and location of the available PR regions, hence, an additional step where the available resources are queried is introduced.

The user inspects that there is an enclave area which fits the placement and size of the bitstream.[4] If the enclave region is approved by the user, the next step is to provide the partial bitstream, encrypted with Pub_{BEK}. The user may also provide an application to run in the processing subsystem (PS). Depending on confidentiality needs, said application can also be encrypted using the same key.

$Shell_{A\text{-}RoT}$ decrypts, using $Priv_{BEK}$, and checks the bitstream signature using Pub_{BitSK}. $Shell_{A\text{-}RoT}$ also verifies that the partial bitstream fits into a non-occupied enclave area. The application is started in a separate, protected environment, and a hash of the application is stored. The partial bitstream is used to populate an enclave area, and the hash of the partial bitstream is also stored. At this point, $Shell_{A\text{-}RoT}$ supplies a signed quote, containing the computed hashes and possibly also hashed configuration data, to the user.

The hashes can be precomputed by the user and compared to the quote. Any discrepancies in this comparison will cause the user to reject the quote and disconnect from the device.

The final step of the initialization phase is to establish a common ephemeral secret key. This is done via an authenticated key agreement scheme (using Pub_E to check the authenticity). The cryptographic algorithms for key exchange and data transport protection, can be implemented in the user-provided enclave. Alternatively, the A-RoT shell could provide these crypto capabilities. Once receiving a finish-command, the $Shell_{A\text{-}RoT}$ wipes the enclave using a blanking bitstream, removing the design and any sensitive remnants.

Multiple users can perform the process described in this section without compromising security or require trust between co-tenants. By designing BS_{shell} in a way where enclaves are physically separated, combined with BCe checking all user designs for dangerous constructs, the risk of eavesdropping on signals can be significantly reduced. For each user, $Shell_{A\text{-}RoT}$ sets up a new domain, i.e. the application, the enclave and an exclusively allocated memory region. The communication channels into each domain are kept separate, either by letting $Shell_{A\text{-}RoT}$ act as ethernet gateway or by allocating separate ethernet controllers for the respective domain. $Shell_{A\text{-}RoT}$ also

[2]This requires asymmetric algorithms where the private key can be deterministically derived from a seed.

[3]A seeded key derivation function can, for example, be embodied by a hash function where a specific prefix/suffix is added to each input.

[4]Enclave areas in different BS_{shell} must be made available to user, e.g. on A-RoT homepage.

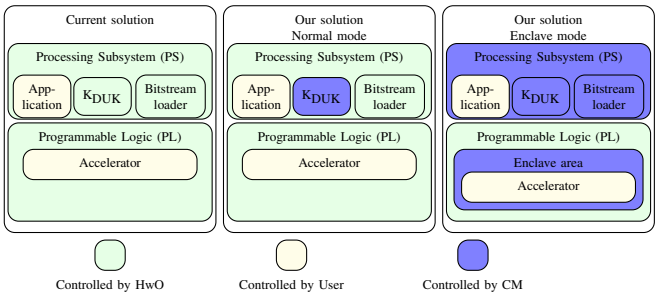

Fig. 2. Trust in currently deployed FPGAs vs suggested future FPGAs

ensures locking of active domains, i.e. a partial bitstream aimed for an already programmed enclave will be rejected.

IV. Implementation and result

In the spirit of reducing the number of trusted entities (as well as alignment with SGX), we assume that the root of trust for attestation (A-RoT) and bitstream certifier (BCe) are, in fact, the chip manufacturer (CM).

We investigated the feasibility of our security model with an experimental implementation on a Zynq Ultrascale+ MPSoC (ZCU102). In our implementation, the boot loader is under A-RoTs control and determines the signature on all loaded components. We have also chosen to let A-RoT control the Platform Management Unit (PMU), as this entity handles all requests to peripherals and specifically readout of K_{DUK}. And, as concluded in the CLKSCREW attack [8], the PMU can be considered as a very security-sensitive resource. To protect the processing subsystem and enable multi-tenancy simultaneously, we included a modified version of the Xilinx ported Xen hypervisor as a part of the boot image. The programmable logic is controlled by BS_{shell}. All the above-listed components are signed by $Priv_{BootSK}$.

To also allow for a use case with lower security guarantees on the same device, i.e. trusting the HwO, we introduce two boot modes. *Enclave mode* fully implements all functionality of $Shell_{A\text{-}RoT}$ as described in Section III-B, while in *normal mode*, we reduce $Shell_{A\text{-}RoT}$ to only protect the critical resources required to retain the security of enclave mode. This is achieved by protecting K_{DUK} and generating distinct K_M for each boot mode. Any attempt to impersonate enclave mode is thereby prevented. The difference is illustrated in Fig. 2.

In our model, the synthesized, routed and placed designs (alternatively, the netlists) are created by BCr and sent to BCe for inspection, bitstream generation and signing. Virus scanning of netlists and bitstreams is an open research problem and crucial for our model. However, the same scanning which is performed by the current HwO-controlled tool-chain can be performed by a BCe.

Our implementation shows that all components needed for remote confidential computation are implementable:

- Predefined enclave regions, populated and wiped by the $Shell_{A\text{-}RoT}$.
- Unique keys for each enclave, guaranteed to be device-unique by deriving from K_{DUK}.
- Only exposing data inside the enclave region.
- Setting up separate memory regions in both PS and PL, thereby enabling multi-tenancy.

V. Discussion & Future work

As concluded in the previous section, in any kind of security model where HwO is not trusted, K_{DUK} must be protected from local access. To reduce the amount of code needed to be owned and controlled by the chip manufacturer (CM), one option would be to incorporate support in hardware. There are already CM-controlled parts of current-day FPGAs which manages the device unique keys and makes them (or derivatives from them) available to PL and PS. We suspect that for the future generation of FPGAs, the changes needed to support the key hierarchy, illustrated in Fig. 1, in hardware would be minor.

Partial reconfiguration is currently very restrictive as it requires pre-determined and non-overlapping areas. Improving the toolchain to allow overlapping areas or dynamic allocation of partial regions during runtime would vastly increase the flexibility for FaaS multi-tenancy scenarios.

Another challenge is how to handle updates of security version that would require invalidation of all prior versions. This can be solved by using one-time programmable memory to increment the lowest allowed version number. The HwO would be responsible for using a boot image with the latest security version and the user able to inspect that the security version is up-to-date when connecting to the device.

A hypervisor is a rather invasive method to protect the processing subsystem. There could be ways of leveraging other security mechanisms, such as TrustZone and trusted firmware, to accomplish the same result.

This paper has focused on the feasibility of an SGX-inspired security model with bitstream inspection support, using a contemporary device. Further research regarding our model's performance impact and area consumption is needed.

References

[1] Matthew Hoekstra. Intel SGX design objectives. https://software.intel.com/en-us/blogs/2013/09/26/protecting-application-secrets-with-intel-sgx, Sep 2013. Accessed 2020-03-13.

[2] AWS. Amazon EC2 F1 instances. https://aws.amazon.com/about-aws/whats-new/2017/04/amazon-ec2-f1-instances-customizable-fpgas-for-hardware-acceleration-are-now-generally-available/, Apr 2017. Accessed 2020-03-13.

[3] Christian Beckhoff, Dirk Koch, and Jim Torresen. Short-circuits on FPGAs caused by partial runtime reconfiguration. In *2010 International Conference on Field Programmable Logic and Applications*, pages 596–601. IEEE, 2010.

[4] Muhammad ES Elrabaa, Mohammed Al-Asli, and Marwan Abu-Amara. Secure Computing Enclaves Using FPGAs. *IEEE Transactions on Dependable and Secure Computing*, 2019.

[5] K. Eguro and R. Venkatesan. FPGAs for trusted cloud computing. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pages 291–300. IEEE, 2012.

[6] Xilinx. Versal: The First Adaptive Compute Acceleration Platform (ACAP). https://www.xilinx.com/support/documentation/white_papers/wp505-versal-acap.pdf, Sep 2019. Accessed 2020-03-13.

[7] Martin S. Won. Intel® Agilex™. https://plan.seek.intel.com/psg_WW_psgcom3_LPCD_EN_2019_AgilexArchitectureWP, Apr 2019. Accessed 2020-03-13.

[8] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. CLKSCREW:Exposing the Perils of Security-Oblivious Energy Management. In *26th USENIX Security Symposium*, pages 1057–1074, 2017.