Optimized Secure VFPGA Provisioning in a Heterogenous Cloud Environment

A Research Report
Submitted to the Faculty of
The Department of Electrical and Computer Engineering
Villanova University

By

Stephen Donchez

In Partial Fulfillment
of the Requirements for the Degree of
Master of Science in Computer Engineering

VILLANOVA
UNIVERSITY

October 19, 2021

Copyright © 2021 by Stephen Donchez
All Rights Reserved

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Proposal

1.1 Literature Implementation

In "Cryptographically Secure Multi-Tenant Provisioning of FPGAs"[1], Bag et al. outline a comprehensive architecture for the implementation of a cloud-compatible FPGA architecture whereby multiple discrete tenants can occupy "Virtual FPGA" (VFPGA) partitions of a larger Physical FPGA (PFPGA) device in a secured environment. In their proposal, they utilize Key Aggregation Cryptography (KAC) to facilitate the encryption of a symmetric (AES) key, which in turn can be used to decrypt tenant bitstreams within their allocated partition. In this implementation, they utilize a single KAC decryption engine at the PFPGA level, which extracts the AES key and furnishes it to individual AES engines at the VFPGA level. The authors claim that by implementing their design in this manner, the only party which the tenant is required to trust is the FPGA Vendor, who provides the public and private keys used by the KAC engine to decrypt the symmetric key.

1.1.1 Implementation Evaluation

However, this claim is not completely true. By virtue of decrypting the symmetric key at the PFPGA layer, the Cloud Service Provider (CSP) is in possession of both the encrypted bitstream and the symmetric key required to decrypt it. In essence, this is equivalent to them having access to the IP contained within the bitstream itself. Therefore, the tenant must necessarily also trust the CSP, which is not a policy compatible with the security needs of many would-be tenants.

Additionally, this implementation allocates a considerable amount of resources to the multiple identical AES decryption cores, which don't actually yield any additional

security benefit since the AES key is present at the PFPGA layer alongside the bitstream it encrypts. Furthermore, this implementation requires a predefined number of VFPGAs per PFPGA, which constrains the CSP's ability to adjust the size and number (inversely) of VFPGAs per PFPGA to adapt to demand.

1.2 Proposed implementation

The author proposes modifying this implementation to eliminate the need for trust in the CSP, along with eliminating the duplication of the AES IP. To achieve this goal, the author proposes the implementation of an attestably secured partition within the PFPGA fabric, which contains both the KAC and AES engines. By isolating both of these components in an attestably secure partition, the CSP no longer has access to the decrypted symmetric key, preventing them from decrypting the bitstream and gaining access to the IP contained therein. Meanwhile, the presence of the single AES IP in this secured partition eliminates the need for identical engines on each partition, which increases available space and also offers flexibility with regards to VFPGA provisioning, as the only fixed limit remaining is the number of VFPGA device IDs allocated by the FPGA Vendor for the device. This number can simply be made arbitrarily large relative to the realistic number of simultaneous tenants that the device can support, such that the CSP is effectively free to divide the PFPGA as they see fit.

The presence of a secure partition for decryption operations is useless without the ability to securely transport the decrypted data from said partition to the VFPGA in question. The use of the Hard Processor System (HPS), and specifically a Trusted Execution Environment (TEE) implemented therein, could conceivably facilitate the secure transfer of this information between regions of the FPGA without transiting the unsecured outer region of the programable logic. However, implementation of direct connections between partitions and the HPS such as this necessitates will require further investigation, as the mechanics of such an operation are not currently understood by the author. Alternatively, the use of direct memory access could be a route to secure transfer, although assurance must exist to prevent malicious access to the memory regions in question.

The use of the HPS to facilitate such a transfer has additional advantages, as it is conceivable on larger devices that multiple such partitions could be desired to facilitate simultaneous tenant reprogramming. In either case, the use of the HPS to schedule access to the secured partition(s) could similarly prove to be a useful design component.

1.2.1 Research Objectives

The above implementation requires a number of discrete objectives be completed and integrated into a comprehensive design. Those objectives are described below.

- Obtain or design the AES and KAC cores for the secured partition.
  - Ideally these cores should be pipelined for efficiency
- Investigate direct communication between partition and HPS (without untrusted transit)
- Investigate use of DMA to facilitate secure communication between partitions securely
- Design and Implement scheduling algorithm to allocate decryption partition to tenants
  - With multiple AES cores, it may make sense to share a single KAC engine if secured link between them can be guaranteed (the KAC has to decrypt a single key of trivial size, whereas the AES engine is in much higher demand due to need to decrypt entire bitstream)
- Implement attestably secure partition and provide mechanism for attestation and verification
- Implement Trusted Execution Environment (if using HPS for secured data transfer)
- Implement data routing and verification logic within TEE (if using HPS for secured data transfer)

Chapter 2

Key Conclusions and Findings

The below sections capture brief comments relating to various findings obtained as a result of research efforts this semester. They are arranged thematically, and dates for each finding are indicated in parentheses.

2.1 Architecture

- Presence of both the decrypted AES key and the encrypted bitstream in the PFPGA outside the tenant partition necessitates (unwanted) trust of the CSP (9/20/21)

- Allocating a dedicated AES engine to each partition in conjunction with the need for a pre-determined number of unique IDs per PFPGA (for the generation of the device's aggregate key) necessarily constrains the number of tenant partitions, and prevents dynamic resizing of said partitions based on tenant needs. (9/20/21)

- Being able to securely interface a single AES core to any of the VFPGAs, in an isolated container also containing the KAC engine, would resolve both of the concerns above, as well as free up additional space on the PFPGA for tenant logic. (9/20/21)

  - such a container would need to be attestably secure.
  - Such a container would also necessitate a means of secured communication with the tenant partition. One potential solution is utilizing a Trusted Execution Environment (TEE) on the Hard Processor System (HPS) co-located on the die with the PFPGA.

2.2 Cryptocores

- The AES core utilized for last semester's research effort (sourced from [2]) is not pipelined, which greatly limits its performance. (9/27/21)
- The AES core outlined in "Implementation of the AES-128 on Virtex-5 FPGAs" [3], is made efficient by the fact that it utilizes the new (at the time of its publication) 6 bit LUTs implemented in the Virtex-5. The ZedBoard currently being used by this effort contains the same LUTs, making it an ideal choice (10/2/21)
- Initial research efforts have not produced evidence of a publicly available KAC engine implemented in HDL (10/2/21)
- The UltraScale series of SoCs have a hardware AES engine intended to facilitate DPR operations. It may be worth investigating utilizing this instead of an IP, if it can be leveraged in this manner [4].

Chapter 3

AES Engine Scheduling

Since the AES engine is decrypting a bitstream, as opposed to a simple key (as is the case in the KAC engine), it will necessarily be occupied for a considerably longer period of time than the KAC engine for each reconfiguration cycle. Furthermore, since the trend in cloud infrastructure is to implement large devices with many partitions rather than many smaller ones, it is likely that a single AES engine per PFPGA will be insufficient to accomodate the partial reconfiguration needs of said device's tenants.

To this end, this architecture proposes the implementation of multiple discrete AES cores, each of which may be attached to a partition for the purposes of facilitating a decryption operation as part of the reconfiguration process. This necessitates the scheduling and allocation of these resources in a fashion that takes into account the real-world constraints of the CSP. To this end, this architecture proposes an Earliest Deadline First (EDF) scheduling algorithm, to be implemented on the Hard Processor System (HPS) of the heterogenous system. This algorithm, including its assumptions, constraints, and implementation, is outlined in the following sections.

3.1 Assumptions

The following assumptions are made regarding the implementation of the AES core scheduling process for this architecture:

- The time required to decrypt a given partial bitstream increases linearly with a corresponding increase in file size.
- The time required to decrypt a given partial bitstream can be aproximated into a discrete number of "time units".

- Deadlines for decryption are made available as tasks are received, and are driven by real world constraints (License Agreements, Pricing Tier, etc.).
- The CSP will not allocate more decryption jobs to the PFPGA than can be scheduled within their given deadlines (meaning that the schedulability does not need to be verified).
- The time required to switch from processing one bitstream to processing another is both non-trivial and constant.
- The act of reading (ingesting) and writing (outputting) bitstream segments to/from the AES engine requires a known, constant time.
- The time required for communication between the HPS and the AES core is assumed to be both constant and trivial.
- Due to the nature of the environment in which the algorithm is being implemented, it is important to note that tasks are random and likely non-periodic in nature.

3.2 Design Constraints

The implementation outlined in subsequent sections will be subject to the following set of design constraints, imposed by the assumptions listed in the preceeding section as well as best practices:

- In the event of a tie between two task for the next time unit, where one task is the task that was operated on in the previous time unit, the task which was previously being executed shall continue to be executed, in order to elimintate overhead associated with context switching between tasks.
- As the IP core currently selected for performing the AES decryption operation is non-pipelined, and therefore not capable of transitioning seemlessly between decryption operations without first having its state machine reset, there shall be a small reset delay during context switches
- The algorithm shall be implemented for an abstract number of AES cores N, with the assumption that N shall be some small number in the real-world implementation.
- The HPS shall communicate with the AES cores via the AXI bus.
- The bitstream shall reside in the domain of the PS for purposes of this algorithm, and shall be transferred via the AXI bus.

- The atomic unit of the bitstream shall be considered 1 Kilobyte, as too small of an atomic unit will cause scheduling computations to take longer than the time unit itself.
- Tasks are placed in a queue, Q, arranged by deadline in ascending order (such that the earliest deadline is at the start of the queue).
- The deadline associated with a processor is the deadline of the task it is currently executing, or ∞ if it is idle.

3.3 Algorithm Implementation

The following algorithm implements the scheduling routine for the AES Cores, as governed by the constraints and assumptions outlined above:

Algorithm 1 EDF Scheduling Algorithm for AES Decryption Cores

```
// Variables:
// N - the set of cores to be utilized
// Q - the queue containing all tasks currently pending execution
// T_{n_i} - the task assigned to a given core
// D_t - the delay associated with a given task
// D_{n_i} - the delay associated with the task currently executing on a core (for brevity)
// D_{Q_i} - the delay associated with the task at a given position in the queue (for brevity)

Require: n(N) > 0                          ▷ There are a nonzero number of cores
  function SwapTaskToQueue(T)
    for all Q_i ∈ Q do
      if D_T < D_{Q_I} then
        Insert T into Q at index Q_i
        return pop(Q)
      end if
    end for
    Append T to the end of Q               ▷ If all currently queued tasks expire before T
  end function

  while true do                                                  ▷ Each Time Unit
    for all n_i ∈ N do
      if Q ≠ ∅ then
        if D_{n_i} == ∞ then
          T_{n_i} ← T_{Q_0}                                              ▷ Pop Q

            ▷ If the currently executing task has a later deadline than the first task
              in the queue and has the latest deadline of any currently executing task
              across all cores.
        else if (D_{n_i} > D_{Q_0}) AND D_{n_i} == max(D_N) then
          T_{n_i} ←SwapTaskToQueue(T_{n_i})
        else
          T_{n_i} ← T_{n_i}
        end if
      end if
    end for
  end while
```

References

[1] Arnab Bag, Sikhar Patranabis, Debapriya Basu Roy, and Debdeep Mukhopadhyay, "Cryptographically Secure Multi-tenant Provisioning of FPGAs," in *Security, Privacy, and Applied Cryptography Engineering*, Lejla Batina, Stjepan Picek, and Mainack Mondal, Eds., Cham, 2020, Lecture Notes in Computer Science, pp. 208–225, Springer International Publishing.

[2] Jason Tsang and Daniel Dixon, "AES File Encryption and Decryption," Tech. Rep., Simon Fraser University, 2018.

[3] Philippe Bulens, François-Xavier Standaert, Jean-Jacques Quisquater, Pascal Pellegrin, and Gaël Rouvroy, "Implementation of the AES-128 on Virtex-5 FPGAs," in *Progress in Cryptology – AFRICACRYPT 2008*, Serge Vaudenay, Ed., Berlin, Heidelberg, 2008, Lecture Notes in Computer Science, pp. 16–26, Springer.

[4] "Using Encryption and Authentication to Secure an UltraScale/UltraScale+ FPGA Bitstream," Mar. 2021.