2023 International Conference on Field Programmable Technology (ICFPT) | 979-8-3503-5911-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICFPT59805.2023.00007

# A Tenant Side Compilation Solution for Cloud FPGA Deployment

1st Maximillian Panoff
*Dept. of Electrical and Computer Engineering*
*University of Florida*
Gainesville, Florida, 32611, USA
ORCID: 0000-0003-2849-7197

2nd Muhammed Kawser Ahmed
*Dept. of Electrical and Computer Engineering*
*University of Florida*
Gainesville, Florida, USA
ORCID: 0000-0001-7389-7232

3rd Hanqiu Wang
*Dept. of Electrical and Computer Engineering*
*University of Florida*
Gainesville, Florida, 32611, USA
ORCID: 0000-0002-5947-9285

4th Shuo Wang
*Dept. of Electrical and Computer Engineering*
*University of Florida*
Gainesville, Florida, 32611, USA
ORCID: 0000-0002-1827-4355

5th Christophe Bobda
*Dept. of Electrical and Computer Engineering*
*University of Florida*
Gainesville, Florida, 32611, USA
ORCID: 0000-0002-9042-9470

***Abstract*—We propose a DecryptStrapper and Dually Leveraged Deployment (DS+DLD), a protocol to authenticate tenant-side design checks for trusted cloud FPGA deployment. Current methods require tenants to trust the cloud FPGA provider with their designs for inspection and allow for no method to confirm the confidentiality of the design check process (without Trusted Execution Environments which have a performance overhead). As a result, there is a clear need for a protocol that allows both tenants and providers to cooperate when deploying FPGA configuration files in a trusted way. DS+DLD works by combining the hash of a compiled bitstream with the hash of the design rule or virus scan results, preventing tenants from uploading bitstreams with falsified results while also never exposing an unencrypted bitstream to the Cloud FPGA Provider (CFP). The result is sent along with an encrypted version of the bitstream to the cloud provider, who recovers the hash of the bitstream and forwards the encrypted bitstream to a Root-of-Trust on the FPGA to program. This Root-of-Trust, known as *DecryptStrapper* (DS), checks the hash provided by the CFP against the recovered bitstream to confirm they match before programming the FPGA. This ensures that both parties are satisfied with the design in a process we call *Dually Leveraged Deployment*, forming DS+DLD. As DS+DLD does not use true Trusted Execution Environments (TEEs), the compilation overhead is minimal. We find that our method, while marginally slower than others that fail to provide the same level of coverage, manages to outperform the most similar method, completing 1.06x-3x faster when including compile-time and up to 1.5x faster when not, while also addressing a larger amount of potential vulnerabilities.**

***Index Terms*—Cloud FPGA Security, Bitstream Validation, Cloud FPGA Deployment**

This work was partially funded by the National Science Foundation (NSF) under Grant numbers 2007320, 1801599, 1916175, and 2019283. Maximillian Panoff is the corresponding author and is available at m.panoff@ufl.edu

## I. Introduction

FPGAs are well known for their ability to accelerate various computational loads, from search requests [1] to deep learning [2]. As a result, many current cloud users, or *Intended Tenants* (ITs) are eager to integrate FPGA solutions into their solutions without investing in the physical hardware. Many *Cloud FPGA Providers* (CFPs) have emerged in recent years with FPGAs as part of their cloud solutions using recently developed techniques [3]–[5]. However, FPGAs have unique security challenges associated with their design and deployment, in addition to those of traditional cloud computing.

One of the largest of these challenges is Remote Side Channel Analysis. In remote side-channel analysis, users upload a design to an FPGA that contains malicious circuitry such as Ring Oscillators and Time/Digital Converters, which indirectly measure voltages [6]–[8] on the FPGA. This data is then returned to the user, who can analyze it to recover sensitive information from other circuits and data such as the crypto keys used in RSA [6]. With deep learning greatly increasing the capabilities of such side-channel attacks [9], [10], this is a quickly growing threat. Other threats include trojans, which can affect the computations of other designs running on the device or even cause physical damage [11].

In general, to mitigate the risks associated with Remote Side Channel analysis and other threats, CFPs use Design Rules Checks (DRCs) and virus checkers to evaluate hardware designs and ensure that bitstreams are free of malicious circuitry before downloading them onto tenants' FPGA [11]. CFPs request that circuits be compiled on their clouds using the

979-8-3503-5911-4/23/$31.00 ©2023 IEEE
DOI 10.1109/ICFPT59805.2023.00007

vendor software they provide in a secure cloud environment.

This approach naturally raises concerns about the confidentiality of any designs made on these services, as they are compiled in the cloud provider-controlled environment. Solutions to these concerns either rely on Trusted Execution Environments (TEEs), like ARM's TrustZone [12] or Intel's SGX [13], to ensure confidentiality [14], or third parties to compile and check the code [15], and play other active roles in the protocol [14]. The main problem with TEEs is the limited throughput because the checks and compilation take significantly longer [14]. While approaches like SGX-FPGA [16] can mitigate these throughput limitations using FPGA-based TEE/enclaves, they fail to address the root of the issue, authentication of DRC results completed by another party.

To address this issue, we propose a novel protocol, DecryptStrapper and Dually Leveraged Deployment (DS+DLD) (shown at a high level in Figure 1) as a means to ensure that bitstream compiled by tenants is safe, without compromising the confidentiality of the design.

Specifically, we propose DS+DLD to enable CFPs to authenticate the design rule results of a bitstream checked by the tenant. This relies on a Root-of-Trust (RoT) being present on the tenant's device with certain capabilities, as well as a second RoT on the FPGA to program. The CFP can then leverage this RoT to bootstrap trust onto the tenant's device. The tenant then leverages DecryptStrapper, a physical RoT on the target FPGA owned by the CFP, to securely implement the client's designs on the cloud FPGA.

The contributions of this work are as follows:

- We propose a method that uses a cloud FPGA-side RoT to authenticate the results of an inspection for malicious circuitry performed by the client outside a dedicated TEE
- We additionally propose a method using two separate Root-of-Trusts to ensure complete system confidentially and integrity for cloud FPGA deployment without requiring the FPGA Manufacturer/Vendor to play an active role
- We examine the robustness of our proposed solution in a series of case studies, including novel cases.
- We examine the timing overheads of DS+DLD vs. others in this space, with ours outperforming the only other to offer similar levels of protection

The rest of this paper is organized as follows: in Section II, we provide some necessary background to understand this work, and in Section III, we introduce the threat models and introduce a few existing solutions in this area. Next, Section IV describes DS+DLD, and Section V evaluates security in a few case studies and compares it to the solutions from Section III. We further expand upon these comparisons with an analysis of timing overheads for DS+DLD in Section VI. Finally, we conclude our work in Section VII.

## II. Background

To help better understand DS+DLD, we introduce several foundational security concepts that are used throughout the protocol. These key concepts include Hash Chains, Certificate Chains, Public Key Infrastructure (PKI), Root of Trust (RoT), and the Trusted Execution Environment (TEE).

### A. Hash Chains and Hash States

A Hash chain is a state formed from a sequence of hash values. Each hash value is computed by combining the previous hash value with the new hash. Hash chains are often leveraged in secure boot solutions. During the boot process, the hash state is compared to a list of known values stored in a secure location in the system. If the hash state matches a known value, the system continues. If the hash state does not match a known value, it is assumed that the process has been compromised, and the system is halted [17].

### B. Certificate Chains and Public Key Infrastructure

Certificate chains, (*i.e.* certificate hierarchies, certificate paths) play a critical role in establishing the trustworthiness of digital certificates in a Public Key Infrastructure (PKI) [18]. A certificate chain comprises three components: the end certificate, intermediate certificates, and the root or Certificate Authority (CA) certificate. Certificates are verified starting with the end certificate and proceeding up the chain the trusted CA certificate is reached. Once each certificate in the chain is validated, each certificate within the chain is trusted.

### C. Root of Trust (RoT)

A Root of Trust (RoT) is used to establish the trust of all other components within a system. A common use of this is to assist with the boot process, where a system can rely on the RoT to authenticate a bootloader, which can then check the next component, and so on [19]. A RoT can also provide various security functions like Secure Key Storage, Attestation, Secure Communication, and Cryptographic Operations, which it uses to establish trust of other components.

### D. Trusted Execution Environment (TEE)

Trusted Execution Environments (TEEs) take many forms but are typically execution platforms separate from the rest of a system's hardware. Operations conducted in a TEE are assumed to be safe from external processes and actors. Both Intel and Xilinx/AMD devices include TEEs [13], [20], and additionally, many academic works focusing on improvements to these, especially in the context of FPGAs [16], [21], [22].

## III. Threat Model and Related Works

In this work, we examine the case of Cloud FPGA Providers (CFPs) and accelerator developers or Intended Tenants (ITs) from the FPGA Accelerated Cloud model of [23]. In this model, ITs design custom hardware logic and utilize rented FPGA resources for acceleration. As an example, Amazon provides single-tenant FPGA access through their AWS F1 EC2 service. Tenants can rent an FPGA region allocated by Amazon and design their hardware logic on Amazon servers. However, direct upload of the design to the FPGA is not permitted. Instead, AWS compiles the design, incorporating preset Design Rule Checks (DRCs), and then uploads the final bitstream to the FPGA [23]–[25]. Turan *et al.* find this to

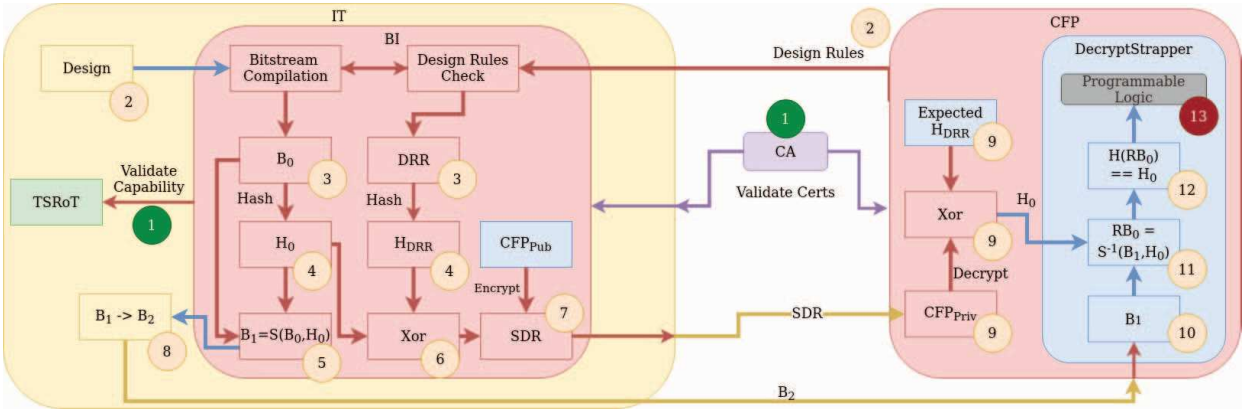



Fig. 1: A high-level overview of DS+DLD, which is explained in full in Section IV.

be the most common and modern model for Cloud FPGAs (although research for secure multi-tenancy is ongoing [26]–[28]). Importantly, they note that:

> *This trust model and base-design oriented security protects only the platform providers as they administer them.* In addition, no feature provides the developers with any protection from potential malice or abuse. Application developers have to trust the cloud service providers if they decide using their infrastructure. [23] (emphasis ours)

As this is a critical security flaw, we introduce threats that exploit this requirement and evaluate how DS+DLD mitigates threats from both the IT and CFP in Section V.

### A. Traditional Threat Models in FPGA Accelerated Cloud

In most prior works on Cloud FPGA deployment, two main threat models are studied: insertion of malicious circuitry by ITs for remote side channel or Denial of Service attacks, and CFPs stealing designs belonging to ITs, known as IP Piracy.

*1) IT Malicious Circuit Insertion:* It is possible for ITs to include circuitry in their designs that causes physical damage or breaches the integrity or confidentiality of other designs running on the same FPGA as that circuitry. To prevent this CFPs have DRCs and virus scanners which examine designs to ensure those types of circuits are not included in a design [11] or to limit interactions between multiple tenants on the same physical FPGA [26]–[28].

*2) CFP IP Piracy:* To run the design checks mentioned above CFPs must have access to the raw design or bitstream, which raises concerns about the confidentiality of these designs. If CFPs copy an IT's designs, or Intellectual Properties (IPs), they may reuse or resell them. Balancing this with the CFP's need to prevent malicious circuitry poses a challenging problem without a definite solution. So far, most solutions in this space favor the CFPs, as mentioned in [23], although recent works have begun to move beyond this [14].

### B. Novel Threat Models

However, these two cases do not cover every potential threat during cloud FPGA deployment. In particular, to the best of our knowledge, this is the first work to explicitly examine the following threat models that even industrial cloud FPGA infrastructures are vulnerable to.

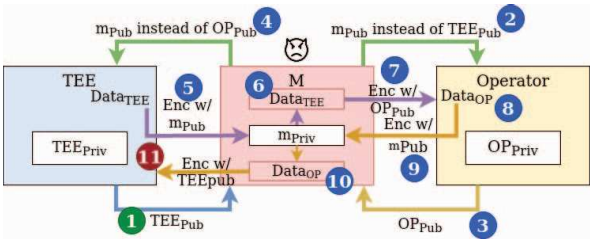



Fig. 2: An example of how MITM attacks can occur during key exchange, even when TEEs are used.

*1) CFP Man-in-the-Middle:* Prior works like [14] rely on the ability of TEEs to exchange keys with outside operators securely. However, TEEs can be vulnerable to Man-in-the-Middle (MITM) attacks when not externally confirming the other party, as shown in Figure 2. Such attacks can defeat current industrial solutions that use the following steps Compile, Sign, and Symmetrically Encrypt (CSSE) [23] as discussed below. While [14] does include a solution to this, it requires the FPGA manufacturing to maintain a service allowing users to confirm the responses of Physically Unclonable Functions (PUFs) on certain FPGAs, and thus limits practicality.

*2) CFP Malicious Circuit Insertion:* Preventing keys used to secure designs from recovery by CFPs goes beyond confidentiality concerns. In particular, CFPs may add malicious circuitry to designs that affect their performance during critical junctions. Therefore solutions must ensure IT IP is placed on the FPGA unmodified and unseen.

*3) IT Design Rule Recovery:* If given foreknowledge of the checks designs will be subject to, Malicious ITs may be able to operate around those restrictions. Thus all Design Rule knowledge the IT obtains must be minimal.

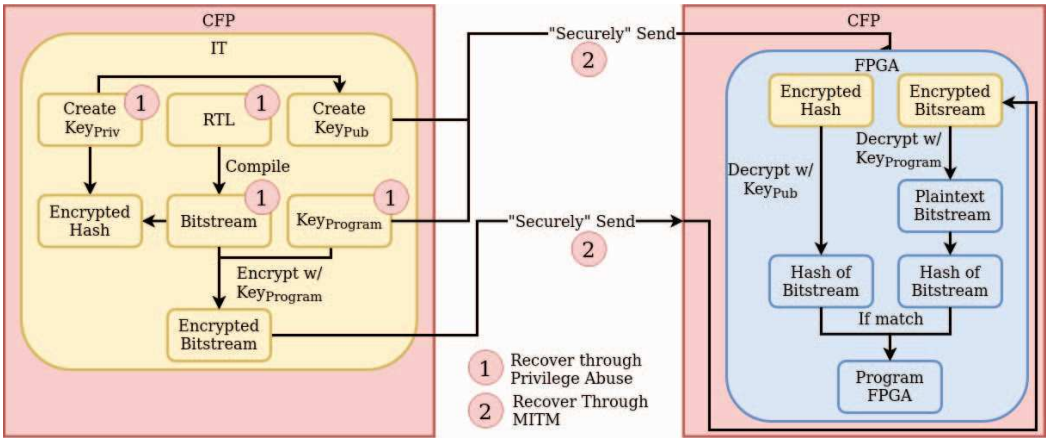



Fig. 3: An example of how CFPs can abuse the current FPGA Accelerated Cloud Model during CSSE, either through directly recovering IP as they control the hardware (1) or performing MITM (Figure 2) to intercept passwords and protected data as they control both ends of the communication (2).

### C. Limitations of Existing Solutions

To our knowledge, a single cohesive protocol that can jointly address these concerns does not yet exist. While [15] can maintain IP confidentiality and uses PKI, it requires a trusted third party to manage bitstream generation and upload while still not explicitly examining the case of malicious circuitry. More recently, TruFPGA [14] examines both aspects of the problem but can only examine the final bitstream, and not intermediate stages, for malicious circuitry. TruFPGA also requires the FPGA Manufacturer (FM) to play an active role, providing ITs with Trusted Shells, nonces (one-off random numbers used to prevent replay attacks), and Challenges for a PUF built into the FPGA. At the same time, TruFPGA has the non-insubstantial overhead of 3.21x slower compile time due to the compilation and design rules happening in an SGX [13] TEE provided by the CFP. Client-side TEE compilation has even further reduced compilation performance due to smaller TEEs. TruFPGA also does not include support for a CA/PKI, and as such, may be vulnerable to MITM as seen in Figure 2.

On the other hand, CSSE is the solution currently supported by industry, with hardware RoTs now included on several boards [20], [29]. This method works as follows: Firstly, the developer hashes the bitstream and encrypts the hash with their private key. Next, the developer encrypts the bitstream with a symmetric key. The IT then sends the symmetric key, public key, encrypted hash, and encrypted bitstream to the FPGA. The FPGA then decrypts the bitstream with the symmetric key and decrypts the encrypted hash. It then calculates the hash of the recovered bitstream to match against the decrypted hash. Should those two match, it proves the authenticity of the recovered bitstream. However, CSSE can be defeated by the CFP, as shown in Figure 3. Specifically, once the CFP has access to $Key_{Program}$ and $Key_{Priv}$, they can easily recover the bitstream and/or modify the Encrypted Hash and bitstream, or even recover the RTL design directly from their servers.

| Term | Meaning |
|---|---|
| IP | Intellectual Property |
| CFP | Cloud FPGA Provider |
| IT | Intended Tenant |
| FM | FPGA Manufacturer |
| $B_{\#}$ | Bitstream number # |
| $H_{\#}$ | Hash of Bitstream number # |
| DR | Design Rule(s) |
| EDR | Encrypted Design Rules |
| DRR | Design Rule Results |
| SDRR | Secured Design Rule Results |
| AsymEnc(D, K) | Asymmetric Encryption of data D with key K |
| AsymDec(D, K) | Asymmetric Decryption of data D with key K |
| SymEnc(D, K) | Symmetric Encryption of data D with key K |
| SymDec(D, K) | Symmetric Decryption of data D with key K |
| TSRoT | (Intended) Tenant-side Root of Trust |
| DS | DecryptStrapper (FPGA-side Root of Trust) |
| CA | Certificate Authority |
| BI | Bootable Image / Executable |

TABLE I: Common Definitions in this work.

## IV. Proposed Method

### A. Assumptions and Definitions

We make a few key assumptions as to the capabilities and responsibilities of various parties of the Cloud FPGA Deployment which we outline below and introduce some common terms in Table I.

- The TSRoT is capable of shutting down the BI should the TSRoT Hash State exit a list of allowed values
- The BI will require TSRoT authentication to execute any program, and the TSRoT was enabled at boot
- The CFP has a set of Design Rules that can detect malicious circuitry in a design along with other violations
- There is a CA trusted and used by the IT, FM, and CFP
- DS prevents reading out the current FPGA configuration
- DS and TSRoT are side-channel resistant

### B. Critical Requirements for Security

DS+DLD requires the following statements to hold true in order to protect the system. In Section V we will examine how

**Algorithm 1:** BI Methods

**Function** `BI Validate(`*Design, DR*`)`**:**
  BI Confirms TSRoT Cert and Capabilities
  BI Generates $BI_{Pub}, BI_{Priv}, BI_{Cert}$ and sends them to TSRoT to sign
  **return** $TSRoT_{Cert}$, $BI_{Cert}$, $EDR_{Req}$

**Function** `DRC(`*Design, DR*`)`**:**
  **foreach** *rule* $\in$ *DR* **do**
    **if** *rule met* **then**
      DRR$[rule] = 1$
    **else**
      DRR$[rule] = 0$
  **end**
  **return** DRR

**Function** `BI Compile(`*DR, RTL,* $EDR_{Key}$`)`**:**
  $N_{EDR}$ = AsymDec(AsymDec($EDR_{Key}$, $BI_{Priv}$), $TSRoT_{Priv}$)
  $DR$ =SymDec($EDR, ED_{Key}$)
  $DR_{RTL}$, $DR_{Syn}$, $DR_{Impl}$, $DR_{Bit}$ $\leftarrow$ DR
  DRR$_1$ $\leftarrow$ `DRC(`*RTL,* $DR_{RTL}$`)`
  Syn $\leftarrow$ Synthesis(RTL)
  DRR$_2$ $\leftarrow$ `DRC(`*Syn,* $DR_{Syn}$`)`
  Impl $\leftarrow$ Implementation(Syn)
  DRR$_3$ $\leftarrow$ `DRC(`*Impl,* $DR_{Impl}$`)`
  $B_0$ $\leftarrow$ Bitstream(Impl)
  DRR$_4$ $\leftarrow$ `DRC(`$B_0$, $DR_{Bit}$`)`
  DRR $\leftarrow$ [DRR$_1$, DRR$_2$, DRR$_3$, DRR$_4$]
  $H_0 \leftarrow$ Hash($B_0$)
  SDRR $\leftarrow$ AsymEnc(Hash(DRR) $\oplus$ $H_0$, $CFP_{Pub_{Key}}$) **//See IV-D.1**
  $B_1 \leftarrow$ SymEnc($B_0$, $H_0$) **//See IV-D.2**
  **return** SDRR, $B_1$

DS+DLD mitigates risk from potential cases in which either the CFP or the IT attempt to violate these assumptions.

1) CFP cannot recover or modify $B_0$ or the RTL code.
2) $B_0$ does not contain malicious circuitry.

As long as these statements hold true, all confidentiality and integrity requirements should be met for the system.

### C. IT Registration

The process starts with the IT downloading and booting a Bootable Image (BI) from the CFP. This BI is a minimal executable used only for compiling and checking a design, and is not intended as a full design environment. Once loaded, the BI generates its own keys, uses the TSRoT to sign them, and then signs and forwards the result and TSRoT Capabilities to the CFP. The CFP then calculates and returns EDR to the BI, which it decrypts into DR. It is important to note that DR should contain 'negative' passing rules (*e.g.* rules which, when violated, indicate a non-malicious design and, when complied with, indicate a malicious design). These will greatly increase the potential values to explore if an attacker were to brute force the system and act as a sort of session-specific

**Algorithm 2:** Main Flow of DS+DLD

**Function** `Main()`**:**
  IT boots BI
  CFP $\leftarrow$ `BI Validate()`
  CFP Validates $TSRoT_{Cert}$, $BI_{Cert}$ with the CA
  CFP sends $TSRoT_{Pub}$ to DS
  DS creates nonce $N_{DS}$
  $N_{DS\_Safe}$ $\leftarrow$AsymEnc($N_{DS}, TSRoT_{Pub}$)
  $N_{DS\_Auth}$ $\leftarrow$ AsymEnc($N_{DS\_Safe}, DS_{Priv}$)
  $N_{DS\_Full} \leftarrow [N_{DS\_Safe}, N_{DS\_Auth}]$
  CFP generates nonce $N_{EDR}$ **//See IV-D.3**
  EDR $\leftarrow$ SymEnc(DR, $N_{EDR}$)) **//See IV-D.3**
  $EDR_{Key} \leftarrow$ AsymEnc(AsymEnc($N_{EDR}$, $BI_{Pub}$), $TSRoT_{Pub}$) **//See IV-D.3**
  CFP returns EDR, $EDR_{Key}$, $NDS\_Full$
  IT Provides RTL code to BI
  SDRR, $B_1 \leftarrow$ `BI Compile(`*DR, RTL,* $EDR_{Key}$`)`
  IT and confirms $DS_{Cert}$ with CA
  $\hat{N_{DS}} \leftarrow$AsymDec($N_{DS\_Safe}, TSRoT_{Priv}$)
  IT Confirms AsymDec(AsymDec($N_{DS\_Auth}$, $DS_{Pub}$), $TSRoT_{Priv}$) == $\hat{N_{DS}}$
  $B_2 \leftarrow$ SymEnc($B_1, \hat{N_{DS}}$) **//See IV-D.4**

**Algorithm 3:** $B_0$ Deployment

**Function** `DecryptStrapper(`$\hat{H_0}$, $B_2$`)`**:**
  $\hat{B_1} \leftarrow$ SymDec($B_2, N_{DS}$)
  $\hat{B_0} \leftarrow$ SymDec($B_1$, $\hat{H_0}$)
  **if** *Hash(*$\hat{B_0}$*)* == $\hat{H_0}$ **then**
    DS Programs with $\hat{B_0}$
  **else**
    DS does not program with $\hat{B_0}$, alerts CFP
**Function** `Main(`*SDRR,* $DRR_{Pred}$, $B_2$`)`**:**
  DRR $\leftarrow$ AsymDec(SDRR, $CFP_{Priv}$)
  $\hat{H_0} \leftarrow$ DRR $\oplus$ Hash($DRR_{Pred}$)
  $DS \leftarrow B_2, \hat{H_0}$ from CFP
  `DecryptStrapper(`$\hat{H_0}$, $B_2$`)`

nonce, mitigating SDRR rehydration. At the same time, the CFP should identify an FPGA for the client and provide the onboard DS with the $TSRoT_{Pub}$, while also returning the $DSCert$ and $N_{DS_Full}$ to the IT.

### D. Bitstream Creation

Once Registration has been completed, the IT can use the BI to create the SDRR and $B_2$. This process is laid out in Algorithms 1 2. The critical aspects of this process include:

1) Line 28 of Algorithm 1, where the hash of raw bitstream $B_0$ is xored with the hash of the raw SDRR before being encrypted by the CFP's Public Key

2) Line 29 of Algorithm 1, the hash of $B_0$ is used as a symmetric key, thereby requiring DS to be provided with the hash of $B_0$ to be decrypted
3) Lines 10-12 of Algorithm 2, the CFP creates a nonce, $N_{EDR}$, which is used as a key for symmetric encryption of the DR. This nonce is then asymmetrically encrypted by the BI and TSRoT public keys and sent to the BI. This greatly increases performance over asymmetrically encrypting the whole of DR.
4) Line 19 of Algorithm 2, as the Symmetrically Encrypted bitstream $B_1$ from line 22 is encrypted by the $N_{DS}$, which is not shared with the CFP, it remains locked to a particular session as well as unrecoverable by the CFP.

### E. Bitstream Authentication and Deployment

The resulting SDRR hash and $B_2$ are then sent to the CFP. Following the process outlined in Algorithm 3, the CFP recovers an estimate hash of $B_0$, $\hat{H}_0$. Only if all rules in DR were followed or violated as expected by the CFP will $\mathbf{H_0} == \mathbf{\hat{H}_0}$. Thus the CFP and the DS using $\hat{H}_0$ to 'unlock' $B_1$ prevents altered or non-conforming designs.

## V. Case Studies

To understand the strengths and capabilities of DS+DLD, we will examine a few standard cases, as well as a few novel ones, summarized by Table II. Note that this is not a comprehensive list but focuses on the most relevant scenarios.

### A. Malicious Insertions

Modern Cloud FPGA solutions must protect against Remote Side Channel Analysis (RSCA) [6] and other malicious circuitry. These cases include a few studies of how DS+DLD handles efforts to introduce malicious circuitry by the CFP or IT. In this section, we examine a few cases, focusing on methods for the IT to defeat the CFP's checks for malicious circuitry in a design or for the CFP to alter $B_0$.

*1) Malicious Circuitry by IT:* Should there be Malicious Circuity in the RTL code provided to the BI by the IT, the Design Rule Results (DRR) will differ from the CFP's expectation. As a result, the Hash of the DRR will be different. Thus when the SDRR is used to obtain $\hat{H}_0$, the resulting value will be incorrect and the hash of the symmetrically decrypted bitstream $\hat{B}_0$ will not match $\hat{H}_0$, and DS will refuse to program the FPGA and alert CFP.

*2) Alterations to $B_2$ by IT:* Should the IT attempt to replace $B_2$/$B_1$/$B_0$ with a separate bitstream $B_{Mal}$ that is not the same as the one used to make SDRR, the Hash of $B_{Mal}$ will not equal $\hat{H}_0$, therefore DS will prevent the upload and alert CFP.

*3) Compile Time Alterations by CFP:* Attempts by BI to insert malicious circuitry should be detectable by IT. IT can create a bitstream of their own design implemented on the same type of board, $B_{Test}$. They can then do a similar process to Algorithm 3 by taking the Hash of $B_Test$ and using that to decrypt $B_1$ and confirming that the result matches $B_{Test}$.

*4) Post-Compile Alterations by CFP:* Should CFP attempt to alter $B_2$ prior to deployment on the FPGA, the hash of $B_0$ will be altered. Even if $\hat{H}_0 == H_0$ (*i.e.* the original bitstream hash is successfully recovered by the CFP), the properties of a cryptographic hash ensure that the CFP cannot use that information to defeat Decryptstrapper's check of the hash of $\hat{H}_0 =? Hash(\hat{B}_0)$. It would be possible for the CFP to alter portions of the deployed bitstream post-deployment, but as this would have to be done blind, it would be difficult to accomplish a meaningful stealthy modification of IT logic. Currently, CSSE does not protect against this case, as the CFP can recover and modify the bitstream and can modify the hash the RoT uses for authentication [20], [29].

### B. IP Piracy

Large amounts of time and effort go into creating and testing accelerator IPs, and IP Privacy (the theft of IP) is an unsolved problem in hardware security. IP Piracy in Cloud FPGA deployment has traditionally been solved through locking of IP as in [15]. However, in light of RSCA, many CFPs are unwilling to accept the risk of deploying a design they have not evaluated. As a result, ITs often must accept the risk of CFP IP Privacy to deploy their designs. DS+DLD mitigates the threat of IP Piracy as shown in the following case studies.

*1) Pre-Compile IP Recovery by CFP:* As the IT is not in control of BI and cannot verify exactly what is being performed, there is a potential threat of BI communicating $B_0$ to the CFP in an encrypted format. This threat is mitigated by the fact that although IT does not control BI, it is responsible for all communication to and from it (i.e. BI should not be directly connected to the internet). As mentioned in Section V-A4, the IT can easily confirm the contents of $B_1$ to ensure that the raw bitstream is not being provided. The only other outputs from BI after the RTL is provided should be the SDRR and certificates. As these are of fixed length and far smaller than most bitstreams, these requirements prevent BI from covertly exporting IT's IP to the CFP.

*2) Post-Compile IP Recovery by CFP:* As the CFP is never directly exposed to $B_0$ (*i.e.* only encrypted bitstreams), and DS prevents reading out implemented designs, they should be unable to recover the plaintext design. Only $B_2$ which is encrypted by $N_{DS}$, is exposed to the CFP and thus is meaningless to them, with all decryption happening inside the Decryptstrapper. Modern implementations of bitstream protection [20], [29] may be vulnerable to Man-in-the-middle (MITM) attacks. Section III-C and Figure 2 cover this in more detail. To prevent this, the IT verifies the $DS_{Cert}$ with the CA themselves, thus preventing the attack.

### C. Breach of Confidentiality Threats

DS+DLD relies on the secrecy of various keys and values. Should the CFP or IT be able to recover these, they will be able to violate the critical assumptions of DS+DLD and thus cause the system to fail. We examine a few cases in which the parties may attempt to obtain these values and how DS+DLD defeats these attempts. We exclude the cases targeting DRs/DRRs from Table II as those attributes are exclusive to DS+DLD.

| Case | Unprotected | CSSE on IT Hardware | CSSE on CFP Hardware | Eguro *et al.* 2012 [15] | Zeitouni *et al.* 2021 [14][1] | **DS+DLD** |
|---|---|---|---|---|---|---|
| V-A1 | ✗ | ✗ | ✓ | ✗[2] | ✓[3] | ✓ |
| V-A2 | ✗ | ✗ | ✓ | ✗ | ✓ | ✓ |
| V-A3 | ✗ | ✓ | ✗ | ✓ | ✓ | ✓ |
| V-A4 | ✗ | ✓ | ✗ | ✓ | ✓ | ✓ |
| V-B1 | ✗ | ✓ | ✗ | ✓ | ✓ | ✓ |
| V-B2 | ✗ | ✓ | ✗ | ✓ | ✓ | ✓ |
| V-C4 | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ |
| V-D1 | ✗ | ✗ | ✗ | ✓ | ✗[4] | ✓ |

TABLE II: A summary of how DS+DLD compares against other solutions in the FPGA Accelerated Cloud model of [23]. CSSE denotes Compile, Sign, Symmetrically Encrypt. 1) Requires active action by FM to create nonces, challenges, and trusted shells. 2) Unless using a Trusted 3rd party to compile code, which is optional. 3) Only portions of the bitstream are evaluated, not the whole design process. 4) Assuming the MITM attack from III-C.

*1) IT Man-in-the-Middle During Registration:* Should the tenant recover the plaintext DRs, they could extract the DRR, which will violate system security as with the correct DRR the tenant could tie a passing result to an arbitrary bitstream. One method that the tenant may use to recover the plaintext DRs is to perform a MITM attack between the CFP and TSRoT, similar to Figure 2. However, DS+DLD mitigates this threat through CFP confirming $TSRoT_{Cert}$ and $BI_{Cert}$ with the CA.

*2) IT Commandeering of BI:* Another method that the IT could use to recover DRR is to freeze the system or read out intermediate values as BI compiles the design. However, as we assume the TSRoT has the ability to force a system shutdown, needs to compute and record the hash state when executing any programs on the system, and receives a list of acceptable hash states from the CFP, DS+DLD defeats any such attempts. However, it may still be possible for the IT to recover this information through side channel analysis or bus piracy [9]. The CFP can mitigate these threats by using the TSRoT to verify the hardware installed on the system and require components that protect against these concerns through encrypted bus traffic and side channel mitigations.

*3) Binary Analysis of BI:* Should the IT recover $BI_{Priv}$, they could recover DR from EDR. While attempts to get the BI's private key at runtime would be handled by the same prevents as in Section V-C2, it may be possible for IT to violate this assumption of confidentiality through binary analysis. DS+DLD prevents this by having the BI generate a new private key at runtime, which is then registered with the TSRoT to form a certificate chain. This signed certificate is then provided to the CFP, which uses it to create EDR after verifying the creator is the BI. As a result, there is no secret key stored within the binary for the IT to recover.

*4) Side Channel Analysis:* Side channel analysis (SCA) is often used to recover the secret keys used by cryptographic operations, such as $TSRoT_{Priv}$ and $N_{DS}$, which would result in a violation of the critical security requirements of DS+DLD. However, SCA is a well-established threat in hardware security and many RoT components can be designed to resist it. Additionally, modern SCA attempts require thousands of encryption measurements to train a model or multiple measurements using the same key once trained [9]. As DS+DLD uses one-off session-specific nonces as keys, it mitigates this threat.

| Operation | 400kB | 4MB | 40MB | 400MB |
|---|---|---|---|---|
| SHA256 (CPU) | 8 | 25 | 115 | 1414 |
| AES128 (CPU) | 7 | 47 | 140 | 1296 |
| SHA256 [20] | 3.2 | 32 | 320 | 3200 |
| AES128 (FPGA) | 8 | 84 | 836 | 8359 |

TABLE III: A comparison of the time to complete different cryptographic operations across platforms. All values in ms.

| Test | *s.o.* $B$ | *s.o.* $CD$ | CT | s.o. BI | Fig | Case |
|---|---|---|---|---|---|---|
| 1 | 400MB | 400MB | 1min | 200GB | 4a | Least Favorable |
| 2 | 400kB | 400MB | 1hr | 20GB | 4b | Most Favorable |
| 3 | 4MB | 40MB | 5min | 20GB | 4c | Least Realistic |

TABLE IV: The timing case studies conducted. *s.o.* stands for 'size of', $B$ for the bitstream, $CD$ for configuration memory size, and CT for Compile-Time.

### D. Rehydration Threats

Rehydration or replay attacks reuse 'good' messages from prior communications to circumvent security measures. We examine how DS+DL responds to several of these.

*1) IP Rehydration by CFP:* The CFP is prevented from Rehydrating $B_2$ after receiving it in two ways. Firstly, at a more practical level, as the CFP does not know the design $B_0$ it does not have knowledge of the plaintext IP or how to interact with it. Secondly, at a more formal level, as the design is encrypted with $N_{DS}$, it is locked to that particular session.

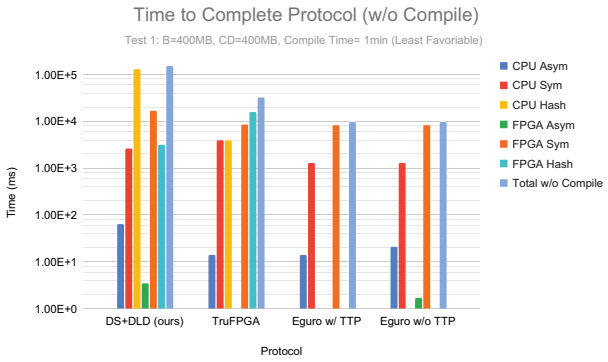
*2) SDRR Rehydration by IT:* Similarly, there are two protections against IT simply replaying a compliant SDRR and then uploading an arbitrary bitstream to the CFP. Firstly, EDR should be changed between transactions, which acts as a nonce. Should the EDR not be changed, however, only a bitstream with a hash matching the one the SDRR was created with will be programmed, which prevents IT from altering it.

## VI. Timing Analysis

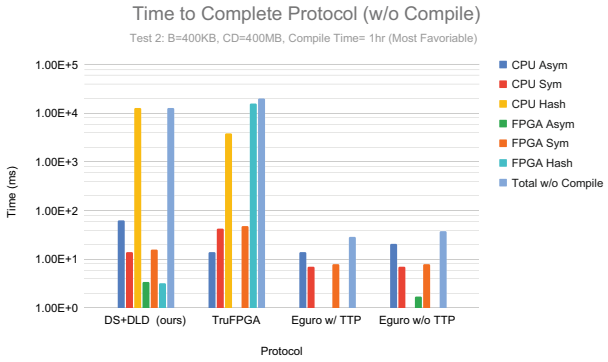
Bitstream security is essential to maintaining programmed FPGAs' integrity. Both major manufacturers, Xilinx and Intel, have implemented proprietary bitstream security solutions, which we denote as Compile, Sign, and Symmetrically Encrypt (CSSE) within their commercially available FPGAs to safeguard against unauthorized alterations and ensure authenticated use [20], [29]. Xilinx, for example, employs security measures across different generations of their FPGA models.

| Operation Type | DS+DLD | TruFPGA [14] | [15] w/ TTP | [15] w/o TTP |
|---|---|---|---|---|
| CPU Asymmetrical | 6 (256 bits) + 3 (2048 bits) | 2 (*s.o.* $FPGA_{ID}$) | 2 (2048 bits) | 1 (2048 bits) |
| CPU Symmetrical | 2 (*s.o.* $B$) | 3 (*s.o.* $B \times C_O$) + 3 (256 bits) | 1 (*s.o.* $B$) | 3 (*s.o.* $B$) |
| CPU Hash | 2 (s.o. $B$) + 2 (*s.o.* DRR) + 1 (s.o. BI) | 3 (*s.o.* CD) | 0 (N/A) | 0 (N/A) |
| FPGA Asymmetrical | 2 (128 bits) | 0 (N/A) | 0 (N/A) | 1 (2048 bits) |
| FPGA Symmetrical | 2 (*s.o.* $B$) | 1 (*s.o.* $B_0 \times C_O$) + 5 (256 bits) | 1 (*s.o.* $B$) | 3 (*s.o.* $B$) |
| FPGA Hash | 1 (*s.o.* $B$) | 5 (*s.o.* CD) | 0 (N/A) | 0 (N/A) |
| Compile Overhead | 1x | 3.21x | 1x | 1x |

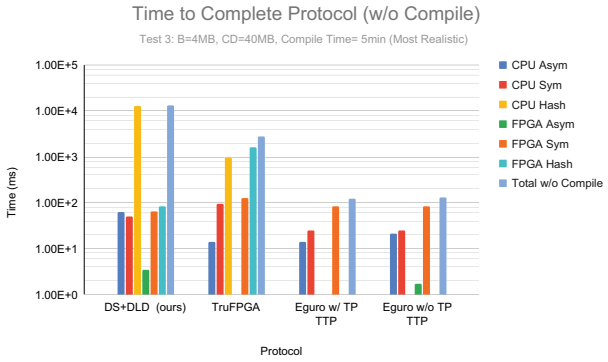
TABLE V: Number of cryptographic operations on each device for each method. $B_0$ denotes the bitstream, $C_O$ the amount of overlap between partial bitstreams, and CD is the Configuration Memory on the FPGA and TTP for Trusted Third Party.



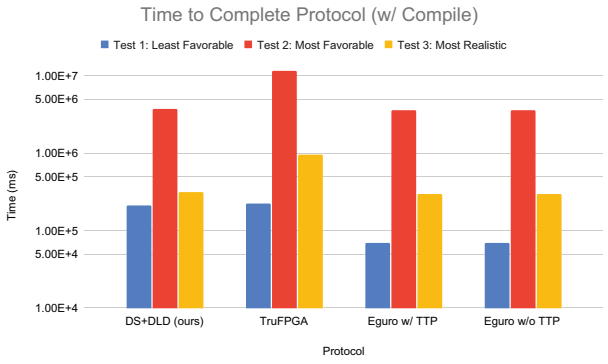

(a) Source of overheads for Test 1




(b) Source of overheads for Test 2




(c) Source of overheads for Test 3




(d) A comparison of total time to complete each Test, including compilation times (lower is better)

Fig. 4: Comparisons between our approach and other similar protocols. Note that Eguro *et al.* only protects against malicious ITs when using a Trusted Third Party.

In their 7 series FPGAs, Advanced Encryption Standard (AES) is utilized to encrypt and decrypt bitstream, which adds an extra layer of security that deters alterations to the bitstream. In the more recent Ultrascale and Ultrascale+ FPGAs, the RSA algorithm is applied to sign the segmented bitstream, providing an additional level of security that enables the identification of the sender and prevents unauthorized modifications [20]. On the other hand, Intel incorporates a Secure Device Manager module in their Stratix 10 FPGAs [29]. This comprehensive module contains an encryption core, AES, and Elliptic Curve Digital Signature Algorithm (ECDSA). The AES core is used to encrypt the bitstream, similar to the security measures implemented by Xilinx. However, Intel further enhances security by employing ECDSA to sign the bitstream, ensuring the integrity and authenticity of the bitstream data [29].

As DS+DLD relies upon a hardware RoT module permanently installed into the cloud FPGA (*i.e.* thus not using programmable logic), we did not feel it would be beneficial to compare resource utilization and thus focused our efforts on timing analysis. We examined the estimated timing delay of existing cryptographic solutions and compared these in Table III. For these tests, CPU evaluation was run on a workstation with an i7-11700KF @ 3.60GHz using OpenSSL [30]. FPGA timings come from [31] for AES and [32] for SHA256, assuming a 33MHz Clock on a Zynq UltraScale+. We also note asymmetric encryption of $\leq 400kB$ takes 7ms for CPU and 1.7ms for FPGA (determined experimentally). We compare the number of cryptographic operations DS+DLD, [14], and [15]

each require in Table V.

We examine the complete timing of each pipeline in a few generalized situations using the estimations from Tables III and V in Figure 4. We assume a DR size of 28,963 bytes, taken from the signatures used by [11]. We also assume no overlap between partial bitstreams and CPU bound TSRoT as a less favorable comparison for DS+DLD vs. TruFPGA across all Tests. We note that configuration Flash memory size on Zynq Ultrascale+ boards varies from 64MB to 512MB [33]. Under these assumptions, we create the Tests in Table IV.

From these, and a complete comparison including compile time in Figure 4d, we can see that DS+DLD performs similarly in terms of timing overhead compared to [15] when including compilation times but is slower when ignoring that. We believe this is still a favorable comparison, as [15] requires a third party to compile and inspect designs prior to programming, and without that entirely fails to prevent malicious circuitry (which is of critical concern to this threat model). At the same time, TruFPGA [14] protects against malicious ITs and CFPs (discounting MITM attacks), but DS+DLD often has lower timing overhead. This difference is largely due to TruFPGA needing to hash the entire configuration of the FPGA, which can be a considerable source of overhead (especially for smaller designs), in addition to using a TEE.

## VII. Conclusion

We present a DS+DLD, a method to authenticate tenant-side DRCs for cloud FPGA deployment without needing a TTP beyond a Certificate Authority. We find that DS+DLD, while marginally slower than others that fail to provide the same coverage [15], manages to outperform the most similar method [14], completing $\sim$1.06x-3x (3.04x under the most realistic conditions) faster when including compile-time and up to 1.5x faster when not. These values assume a CPU-based TSRoT, indicating more significant potential gains with a dedicated hardware TSRoT. Notably, this performance increase improves linearly with compilation time and the size of configuration data for the FPGA, but decreasing linearly with larger bitstreams and BIs.

## References


[1] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–13.

[2] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou, "Dlau: A scalable deep learning accelerator unit on fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 3, pp. 513–517, 2016.

[3] J. M. Mbongue, F. Hategekimana, D. T. Kwadjo, D. Andrews, and C. Bobda, "Fpgavirt: A novel virtualization framework for fpgas in the cloud," in *11th IEEE International Conference on Cloud Computing, CLOUD 2018, San Francisco, CA, USA, July 2-7, 2018*, 2018, pp. 862–865. [Online]. Available: https://doi.org/10.1109/CLOUD.2018.00122

[4] J. M. Mbongue, A. M.-I. Shuping, P. Bhowmik, and C. Bobda, "Architecture support for fpga multi-tenancy in the cloud," in *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2020, pp. 125–132.

[5] J. Mandebi Mbongue, F. Hategekimana, D. Tchuinkou Kwadjo, and C. Bobda, "Fpga virtualization in cloud-based infrastructures over virtio," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, 2018, pp. 242–245.

[6] M. Zhao and G. E. Suh, "Fpga-based remote power side-channel attacks," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 229–244.

[7] S. Moini, A. Deric, X. Li, G. Provelengios, W. Burleson, R. Tessier, and D. Holcomb, "Voltage sensor implementations for remote power attacks on fpgas," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 1, dec 2022. [Online]. Available: https://doi.org/10.1145/3555048

[8] I. Giechaskiel, S. Tian, and J. Szefer, "Cross-vm covert- and side-channel attacks in cloud fpgas," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 1, dec 2022. [Online]. Available: https://doi.org/10.1145/3534972

[9] M. Panoff, H. Yu, H. Shan, and Y. Jin, "A review and comparison of ai-enhanced side channel analysis," *J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 3, apr 2022. [Online]. Available: https://doi.org/10.1145/3517810

[10] H. Yu, S. Wang, H. Shan, M. Panoff, M. Lee, K. Yang, and Y. Jin, "Dual-leak: Deep unsupervised active learning for cross-device profiled side-channel leakage analysis," in *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2023, pp. 144–154. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/HOST55118.2023.10133491

[11] T. M. La, K. Matas, N. Grunchevski, K. D. Pham, and D. Koch, "Fpgadefender: Malicious self-oscillator scanning for xilinx ultrascale + fpgas," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, no. 3, sep 2020. [Online]. Available: https://doi.org/10.1145/3402937

[12] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin, "Trustzone explained: Architectural features and use cases," in *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*. IEEE, 2016, pp. 445–451.

[13] "Overview on Signing and Whitelisting for Intel® Software Guard Extension (Intel® SGX) Enclaves Scope." 2015.

[14] S. Zeitouni, J. Vliegen, T. Frassetto, D. Koch, A.-R. Sadeghi, and N. Mentens, "Trusted configuration in cloud fpgas," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021, pp. 233–241.

[15] K. Eguro and R. Venkatesan, "Fpgas for trusted cloud computing," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, 2012, pp. 63–70.

[16] K. Xia, Y. Luo, X. Xu, and S. Wei, "Sgx-fpga: Trusted execution environment for cpu-fpga heterogeneous architecture," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 301–306.

[17] B. Kelly, "Hardware secure boot," OCP Security workgroup, Tech. Rep. Revision 1.

[18] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. L. Rivest, "Certificate chain discovery in spki/sdsi," *Journal of Computer security*, vol. 9, no. 4, pp. 285–322, 2001.

[19] K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito, "Smart: secure and minimal architecture for (establishing dynamic) root of trust." in *Ndss*, vol. 12, 2012, pp. 1–15.

[20] Xilinx, *TrustZone Technology Support in Zynq-7000 All Programmable SoCs*, 2014. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp429-trustzone-zynq.pdf

[21] S. Kumar Saha and C. Bobda, "FPGA Accelerated Embedded System Security through Hardware Isolation," in *Proceedings of the 2020 Asian Hardware Oriented Security and Trust Symposium, AsianHOST 2020*. Institute of Electrical and Electronics Engineers Inc., 12 2020.

[22] F. Hategekimana, T. Whitaker, M. J. H. Pantho, and C. Bobda, "Shielding non-trusted IPs in SoCs," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–4.

[23] F. Turan and I. Verbauwhede, "Trust in fpga-accelerated cloud computing," *ACM Comput. Surv.*, vol. 53, no. 6, dec 2020. [Online]. Available: https://doi.org/10.1145/3419100

[24] C. Bobda, J. M. Mbongue, P. Chow, M. Ewais, N. Tarafdar, J. C. Vega, K. Eguro, D. Koch, S. Handagala, M. Leeser, M. Herbordt, H. Shahzad, P. Hofste, B. Ringlein, J. Szefer, A. Sanaullah, and R. Tessier, "The future of fpga acceleration in datacenters and the cloud," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 3, feb 2022. [Online]. Available: https://doi.org/10.1145/3506713

[25] J. M. Mbongue, D. T. Kwadjo, A. Shuping, and C. Bobda, "Deploying multi-tenant fpgas within linux-based cloud infrastructure," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 2, dec 2021. [Online]. Available: https://doi.org/10.1145/3474058
[26] A. Bag, S. Patranabis, D. B. Roy, and D. Mukhopadhyay, "Cryptographically secure multi-tenant provisioning of fpgas," 2018.
[27] B. Hong, H.-Y. Kim, M. Kim, T. Suh, L. Xu, and W. Shi, "Fasten: An fpga-based secure system for big data processing," *IEEE Design & Test*, vol. 35, no. 1, pp. 30–38, 2018.
[28] J. Vliegen, M. M. Rabbani, M. Conti, and N. Mentens, "Sacha: Self-attestation of configurable hardware," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 746–751.
[29] Intel, *Secure Device Manager*, 2023.
[30] N. I. of Standards and Technology, "Cryptographic module validation program, certificate #4282," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. 4282, 2022. [Online]. Available: https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/certificates/August%202022_010922_0715_signed.pdf
[31] L. Design Gateway Co., "Aes128 ip," Tech. Rep., Feb 2023. [Online]. Available: https://dgway.com/ASIP_E.html\#AES
[32] S. binti Suhaili and T. Watanabe, "Design of high-throughput sha-256 hash function based on fpga," in *2017 6th International Conference on Electrical Engineering and Informatics (ICEEI)*, 2017, pp. 1–6.
[33] *Zynq 7000 SoC Technical Reference Manual*, Xilinx, june 2023. [Online]. Available: https://docs.xilinx.com/r/en-US/ug585-zynq-7000-SoC-TRM