# Robust Bitstream Protection in FPGA-based Systems through Low-Overhead Obfuscation

Robert Karam*, Tamzidul Hoque*, Sandip Ray†, Mark Tehranipoor*, and Swarup Bhunia*
*Dept. of ECE, University of Florida, Gainesville, FL 32608
†NXP Semiconductor, Austin, TX 78721
Email: robkaram@ufl.edu

***Abstract*—Reconfigurable hardware, such as Field Programmable Gate Arrays (FPGAs), are being increasingly deployed in diverse application areas including automotive systems, critical infrastructures, and the emerging Internet of Things (IoT), to implement customized designs. However, securing FPGA-based designs against piracy, reverse engineering, and tampering is challenging, especially for systems that require remote upgrade. In many cases, existing solutions based on bitstream encryption may not provide sufficient protection against these attacks. In this paper, we present a novel obfuscation approach for provably robust protection of FPGA bitstreams at low overhead that goes well beyond the protection offered by bitstream encryption. The approach works with existing FPGA architectures and synthesis flows, and can be used with encryption techniques, or by itself for power and area-constrained systems. It leverages "FPGA dark silicon" – unused resources within the configurable logic blocks – to efficiently obfuscate the true functionality. We provide a detailed threat model and security analysis for the approach. We have developed a complete application mapping framework that integrates with the Altera Quartus II software. Using this CAD framework, we achieve provably strong security against all major attacks on FPGA bitstreams with an average 13% latency and 2% total power overhead for a set of benchmark circuits, as well as several large-scale open-source IP blocks on commercial FPGA.**

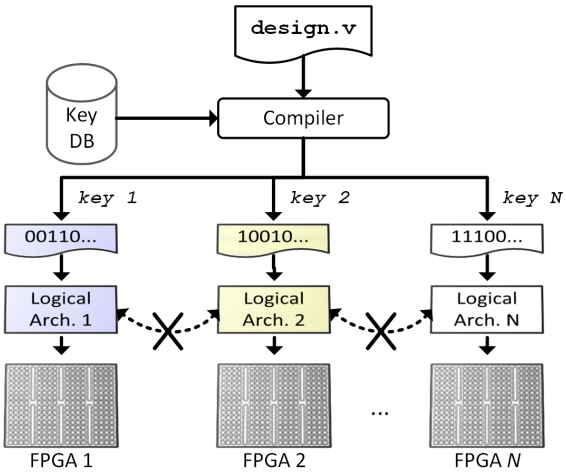


Fig. 1. The proposed key-based bitstream obfuscation generates logically-varying bitstreams for the same underlying FPGA architecture. This offers robust protection against major attacks, including known design, tampering, and physical attack, against which bitstream encryption is insufficient.

## I. Introduction

System security is becoming an increasingly important design consideration in modern computing systems, especially network-connected mobile and Internet of Things (IoT) devices. With estimates of 10s of billions of connected systems in the coming years, it is imperative to have technologies, architectures, and protocols that pair device efficiency with hardware security, data security, and privacy. The use of a reconfigurable hardware platforms, such as Field Programmable Gate Arrays (FPGA), is a common practice that helps designers to satisfy the growing demands on the area, performance, cost, and power requirements of next-generation devices [1]. In particular, FPGAs are well-suited to after-market reconfigurability, enabling them to adapt to changing requirements in functionality, energy-efficiency, and security during the lifetime of a device [2]. This is a critical design consideration in many application domains, including military, automotive, IoT, and data centers, among others.

The role of FPGAs in computing system security has recently seen significant interest from diverse sectors [3], [4]. However, existing research primarily focuses on realizing functional security primitives in FPGA. We note that in addition to serving as a hardware acceleration engine for security primitives, FPGAs are inherently more secure against supply chain attacks. In particular, the post-silicon reconfigurability of FPGAs implies that the true circuit functionality is not realized until after chip production and design details are not exposed to untrusted parties in the foundry and the supply chain. However, the security of the FPGA bitstream (or configuration file), can still be at risk, both during initial configuration in untrusted third-party system integration facilities as well as during wireless and in-field reconfiguration. These bitstreams are susceptible to various attacks, including unauthorized programming to third-party hardware, reverse-engineering of the design, malicious modifications such as hardware Trojan insertion [5], and cloning/piracy of the valuable Intellectual Property (IP) blocks. Such attacks can cause significant monetary loss for the IP designer, and unwanted, unreliable, and even potentially catastrophic operations in the field.

One existing approach to defend against these attacks is bitstream encryption [6], which is typically available for high-end FPGAs. Unfortunately, for many situations, encryption alone is not sufficient for comprehensive system security:

1) FPGAs aimed at applications constrained by aggressive low-energy and cost requirements or tiny form-factors may not include dedicated encryption blocks due to area or energy constraints.
2) During remote upgrade, encryption keys must be sent along with the bitstream, making designs vulnerable to

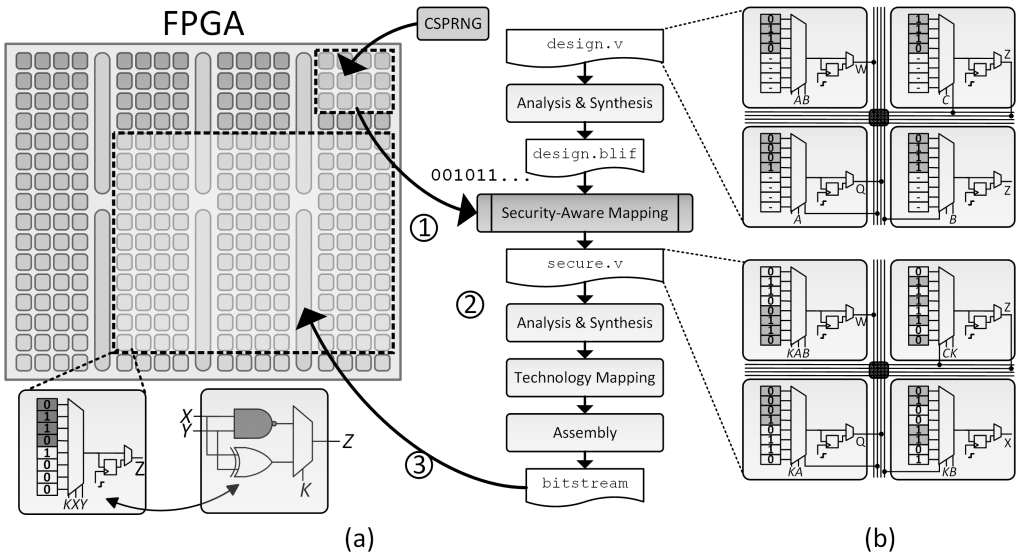


Fig. 2. (a) Security-aware mapping for FPGA bitstreams, which requires no design-time changes to the hardware. An initial synthesis run produces technology-mapped lookup tables, to which (1) the proposed key-based obfuscation is applied; (2) the obfuscated design is resynthesized and (3) mapped to the target FPGA. (b) (top) Original design consisting of 4 CLBs which support 3 input functions, but only contain 2 input functions, leaving 50% of the LUT unoccupied. (bottom) after obfuscation, these content bits are filled with other functions; correct response selection depends on the key input position and value.

leakage to an attacker with network access.

3) Devices that remain in the field for many years, such as those in long-life military, automotive, and IoT applications, are susceptible to physical attacks that can result in leaking the encryption key via side channels (e.g. power, timing) [6].

Consequently, there is a critical need for a robust, low-cost approach to FPGA bitstream security that goes beyond the protection offered by standard encryption techniques.

In this paper, we present a novel, low-overhead FPGA bitstream obfuscation solution, which maintains mathematically provable robustness against major attacks. Central to this contribution is the identification of FPGA dark silicon, i.e., unused LUT memory already available in design mapped to FPGAs, which is exploited to achieve bitstream security. It helps to drastically reduce the overhead of the obfuscation mechanism. The approach does not introduce additional complexity in design verification and incurs a low performance and negligible power penalty. In particular, the mechanism proposed here permits the creation of *logically varying architectures* for an FPGA, so that there is a unique correspondence between a bitstream and the target FPGA. Fig. 1 shows a high-level overview of our approach. Compared to existing logic obfuscation techniques [7], we do not require design-time changes to the FPGA architecture or expensive on-chip public key cryptography. Note that in addition to obfuscation of design functionality, our approach also enables locking a particular bitstream to a specific FPGA device, helping to prevent piracy of the valuable IP blocks incorporated in a design. Therefore, it goes well beyond standard bitstream encryption in FPGA security. Furthermore, it is targeted to the protection of FPGA bitstreams, rather than hardware metering of integrated circuits [7]. Finally, the procedure seamlessly integrates into existing CAD tool flows for programming FPGA devices. *To our knowledge, this is the first comprehensive technique to protect against major attacks on FPGA bitstreams.*

The paper makes the following important contributions:

- We propose a novel, low-overhead key-based bitstream obfuscation technique for FPGAs, which can be used with off-the-shelf FPGA hardware. This approach leverages unused FPGA resources within an existing design to minimize the hardware overhead required for the obfuscation.
- We analyze the properties of unused FPGA resources for various (small ($< 2000$ LUT) to large ($> 40000$ LUT)) combinational and sequential benchmarks, demonstrating that this approach can be used with almost any design.
- We introduce a custom software tool for the proposed obfuscation process. This tool is fast, processing even large FPGA designs in seconds, and is integrated into a complete CAD flow for application mapping which integrates with Altera Quartus II, a commercial software package for mapping designs to FPGA.
- We provide a detailed security analysis that shows the mathematical robustness of the approach against reverse engineering of a design using both brute-force and known design methods, as well as malicious design modification (e.g. hardware Trojan insertion). We also provide comprehensive evaluation of the approach for diverse FPGA designs for area, power, and performance.

The rest of the paper is organized as follows. Section II describes the threats to FPGA bitstream security. Section III defines the concept of *dark silicon* in FPGAs with benchmark results on a modern FPGA. Section IV explains how to exploit FPGA dark silicon for bitstream security and describes the software architecture, tool flow, and the proposed authentication protocol for remote reconfiguration. Section V gives a detailed security analysis and mapping overhead results on physical FPGA hardware. We conclude in Section VI.

## II. FPGA Bitstream Security Issues

Our approach is targeted to protect FPGA-based systems against the following two major categories of attacks.

1) *IP Piracy:* For designs implemented in FPGAs, the attacker can obtain the IP by simply converting the unencrypted bitstream to a netlist [8]. Attacks during wireless reconfiguration and through physical access to devices in the field can lead to resale of valuable IP cores and unauthorized use in third party products.
2) *Targeted Malicious Modification (TMM):* Once reverse engineered, insertion of a targeted malicious modification can give an attacker complete access, leading to reduced device lifetime, unreliable or unwanted functionality, decreased battery life, or sharing of private data. Note that we differentiate such *targeted* attacks from a *random* malicious modification, in which the attacker blindly modifies portions of the bitstream, intending primarily to render the bitstream non-functional; protecting against such random malicious modifications is outside the direct scope of our work.

In the first case, a remote attacker can intercept the communication between the vendor and FPGA during remote upgrade. Decryption keys are commonly sent along with the encrypted bitstream to enable the target device to perform the reconfiguration. For devices in field for many years, such as military or automotive systems, long-term physical access can additionally compromise system security, giving an attacker time to discover vulnerabilities in the security architecture.

In the second case, an attacker can maliciously modify a bitstream, inserting a Trojan to alter the functionality of the device. Even if the true functionality is unknown, it is sometimes possible to make a targeted malicious modification:

1) *Unused Resources:* This attack operates on unencrypted (or decrypted) bitstreams by inserting hardware Trojans in unused FPGA resources that are identified by observing sequences of zeros into the bitstream [5]. Please note that our proposed obfuscation based technique only prevents tampering attacks on partially or fully utilized FPGA resources.
2) *Mapping Rule Extraction:* Another attack that does not rely on *a priori* knowledge of the bitstream format is based on *known design attack*. This is used to reverse engineer the bitstream format, which enables a targeted malicious modification, such as leaking the secret key from a cryptographic module [9]. A number of functional variants are mapped to the device, and the attacker observes how the resultant bitstream changes, enabling the extraction of mapping rules. Once all mapping rules are determined, the knowledge can be used to both reverse-engineer and maliciously alter any bitstream generated for the same FPGA series.

## III. FPGA Dark Silicon

The typical island-style FPGA architecture consists of an array of multi-input, single-output lookup tables (LUTs). Generally, LUTs of size $n$ can be configured to implement any function of $n$ variables, and require $2^n$ bits of storage for function responses. Programmable Interconnects (PIs) can be configured to connect LUTs to realize a given hardware design. Additional resources, including embedded memories, multipliers/DSP blocks, or hardened IP blocks can be reached through the PI network and used in the design.

The nature of FPGA architecture requires that sufficient resources be available for the worst case. For example, some newer FPGAs may support 6 input functions, requiring 64 bits of storage for the LUT content. However, typical designs are more likely to use 5 or fewer inputs, while less frequently utilizing all 6. Note that each unused input results in a 50% decrease in the utilization of the available content bits. This leads to an effect that resembles *dark silicon* in multicore processors [10], where only a limited amount of silicon real estate and parallel processing can be used at a given time. To make this analogy explicit, we refer to the unused space in FPGA as "FPGA dark silicon". Note that in spite of the nomenclature the causes behind dark silicon in the two cases are different. For multicore processors, it is typically due to physical limitations or limited parallelism; for FPGAs, it is the reality of having sufficient resources available for the worst-case which may occur infrequently, if at all.

Our approach critically depends on the presence of FPGA dark silicon to be exploited for obfuscation needs. Consequently, we made a comprehensive evaluation of this phenomenon to identify the scope and scale of this phenomenon. Table I shows the result of this evaluation. Note that the evaluation uses benchmark designs of diverse scale and complexity, taken from three publicly available benchmarks, e.g., EPFL Arithmetic Benchmark Suite (http://lsi.epfl.ch/benchmarks), Opencores (http://opencores.org), and Github (http://github.org). All benchmarks were mapped to an Altera Cyclone V device [11]. The Cyclone V contains two 6-input Adaptive LUTs (ALUTs) per Adaptive Logic Module (ALM), and 10 such ALMs per Logic Array Block (LAB).

Our evaluation shows the availability of significant unused space across the diversity of benchmarks. Even for small combinational circuits (less than 2000 LUTs), roughly 50% of the LUTs mapped use 4 inputs or fewer, while 82% of the LUTs mapped use 5 inputs or fewer. The effect is more pronounced for large sequential benchmarks, where 69% of LUTs are 4 inputs or fewer, and 82% use 5 inputs or fewer.

**Remark:** Our study of FPGA dark silicon is consistent with, albeit different from, existing studies. For example, previous research has shown that the point of minimum area utilization for a typical FPGA is achieved with less than 100% logic utilization [12] due to routing constraints. However, this work refers to utilization in terms of logic, i.e. ALM usage, rather than the percentage of content bits *within the ALMs used to implement the design*. Clearly, the number and type of LUTs can vary widely among specific designs (cf. Table I). While increased content utilization can be a byproduct of more advanced algorithms, it is not necessarily the only objective, as designs can be optimized for other attributes (e.g. timing or power) during synthesis. Note also that the phenomenon of unused LUTs has been exploited in other contexts, e.g., for low-overhead insertion of scan flip flops in the context of Design for Testability (DfT) [13].

To quantify the role of dark silicon, we define a metric,

TABLE I
CUMULATIVE PERCENTAGE OF 1 - 7 INPUT LUTS

| Circuit | Cumulative % of LUTs with Inputs $n$ | | | | | | Total |
|---|---|---|---|---|---|---|---|
| Name | $\leq 2$ | 3 | 4 | 5 | 6 | 7 | LUTs |
| alu4 | 10.6 | 26.1 | 48.4 | 77.7 | 97.9 | 100 | 188 |
| apex2 | 11.4 | 26.0 | 52.3 | 91.0 | 99.1 | 100 | 669 |
| apex4 | 16.7 | 27.4 | 50.3 | 89.4 | 97.6 | 100 | 574 |
| ex5p | 41.0 | 42.1 | 58.7 | 84.5 | 98.4 | 100 | 373 |
| ex1010 | 16.9 | 24.2 | 46.4 | 84.8 | 98.3 | 100 | 711 |
| misex | 14.0 | 27.7 | 46.9 | 84.0 | 97.5 | 100 | 480 |
| pdc | 16.3 | 28.5 | 51.9 | 77.7 | 98.4 | 100 | 1588 |
| seq | 16.6 | 51.9 | 51.9 | 89.1 | 99.0 | 100 | 727 |
| spla | 17.8 | 53.1 | 53.1 | 79.9 | 98.7 | 100 | 1509 |
| **Avg.** | **17.9** | **29.0** | **51.1** | **84.2** | **98.3** | **100** | **758** |
| div | 7.8 | 13.1 | 32.7 | 60.1 | 100 | – | 12.4k |
| hyp | 0.9 | 28.8 | 42.6 | 64.0 | 100 | – | 45.3k |
| log2 | 7.0 | 17.2 | 39.5 | 59.7 | 99.0 | 100 | 7894 |
| mult | 2.5 | 25.0 | 50.5 | 59.0 | 99.2 | 100 | 5553 |
| sqrt | 5.8 | 5.0 | 43.5 | 84.5 | 100 | – | 3685 |
| square | 5.6 | 55.9 | 60.2 | 74.6 | 100 | – | 4066 |
| **Avg.** | **4.5** | **24.2** | **44.8** | **67.0** | **99.7** | **100** | **13.1k** |
| AES | 39.7 | 64.2 | 72.0 | 100 | – | – | 4112 |
| AOR32 | 20.7 | 22.9 | 31.5 | 46.8 | 97.8 | 100 | 2299 |
| BTCM | 32.5 | 95.3 | 99.8 | 100 | 100 | – | 41.0k |
| JPEGE | 45.2 | 37.6 | 48.4 | 67.0 | 99.4 | 100 | 5154 |
| Salsa20 | 59.9 | 57.4 | 93.8 | 93.9 | 100 | – | 2836 |
| **Avg.** | **39.2** | **55.5** | **69.1** | **81.5** | **99.4** | **100** | **11.1k** |

the *Occupancy* of the FPGA, as the percentage of content bits used per LUT, divided by the total number of available bits in the LUTs which are used. We use the Cyclone V device architecture as a case study. In Eqn. 1, the number of $n$-input LUTs ($\#(LUTn)$) is multiplied by the content bits used for that LUT ($2^n$); this value is divided by the LUT capacity $2^p$ times the number of LUTs used in total; the variable $p$ indicates the maximum power of the LUT, which in this case is 6. This yields the *ALUT Occupancy.* Next, *ALM Occupancy* is computed in Eqn. 2 as the average number of ALUTs per ALM; in this case, the *ALM_MAX_CAP* is 2. Finally, the *LAB Occupancy* is computed in Eqn. 3 as the average number of ALMs per LAB; *LAB_MAX_CAP* is 10 for the Cyclone V. Finally, the product of these three terms gives the overall occupancy (Eqn. 4), indicating the true percentage of fine-grained resource utilization at the content bit level for the given FPGA architecture.

$$O_{ALUT} = \frac{\sum_{n=1}^{p} \#(LUTn) \times 2^n}{\#(LUT) \times 2^p} \tag{1}$$

$$O_{ALM} = \frac{\#(ALUT)}{\textit{ALM\_MAX\_CAP} \times \#(ALM)} \tag{2}$$

$$O_{LAB} = \frac{\#(ALM)}{\textit{LAB\_MAX\_CAP} \times \#(LAB)} \tag{3}$$

$$O_{Total} = O_{ALUT} \times O_{ALM} \times O_{LAB} \tag{4}$$

We computed $O_{Total}$ for a set of 9 combinational benchmark circuits [14] and found the average occupancy to be 26% $\pm$ 4%, leaving nearly $3/4$ of the available content bits *within the used LUTs* empty. This same phenomenon extends to designs which require more resources, e.g. large arithmetic circuits [15] for which the occupancy is slightly higher (31% $\pm$ 4) and the previously listed IP cores, for which the occupancy is significantly lower with higher variance (12 %$\pm$ 8).

TABLE II
EXAMPLE LUTS WITH 2 PRIMARY INPUTS AND 1 KEY INPUT. THE TRUE FUNCTION IS $Z = X \oplus Y$, WHICH IS ONLY SELECTED WHEN $K = 0$.

| X | Y | K | **Z** |
|---|---|---|---|
| 0 | 0 | 0 | **0** |
| 0 | 0 | 1 | **0** |
| 0 | 1 | 0 | **1** |
| 0 | 1 | 1 | **0** |
| 1 | 0 | 0 | **1** |
| 1 | 0 | 1 | **0** |
| 1 | 1 | 0 | **0** |
| 1 | 1 | 1 | **1** |

(a)

| X | K | Y | **Z** |
|---|---|---|---|
| 0 | 0 | 0 | **0** |
| 0 | 0 | 1 | **1** |
| 0 | 1 | 0 | **0** |
| 0 | 1 | 1 | **0** |
| 1 | 0 | 0 | **1** |
| 1 | 0 | 1 | **0** |
| 1 | 1 | 0 | **0** |
| 1 | 1 | 1 | **1** |

(b)

| K | X | Y | **Z** |
|---|---|---|---|
| 0 | 0 | 0 | **0** |
| 0 | 0 | 1 | **1** |
| 0 | 1 | 0 | **1** |
| 0 | 1 | 1 | **0** |
| 1 | 0 | 0 | **0** |
| 1 | 0 | 1 | **0** |
| 1 | 1 | 0 | **0** |
| 1 | 1 | 1 | **1** |

(c)

## IV. BITSTREAM PROTECTION METHODOLOGY

In this section, we describe the proposed bitstream protection methodology and its integration into the design flow.

### A. Design Obfuscation

As described above, most of the LUTs used to implement a given design do not require full utilization of the available memory bits. This leaves open spaces where additional function responses can be inserted to obfuscate the true functionality of the design, which in turn makes it more difficult for an adversary to make a Targeted Malicious Modification.

For example, consider a 3-input LUT, which contains 8 content bits, used to implement a 2 input function, $Z = X \veebar Y$. A third input $K$ can be added at either position 1, 2, or 3, leaving the original function in either the top or bottom half of the truth table, or interleaved with the obfuscation function. An example of this is shown in the 4 LUT design of Fig. 2(b), as well as in Table II. In this case, the correct output is selected when $K = 0$; if $K = 1$, a response from the incorrect function ($Z = X \wedge Y$) is selected. However, if it is *not* known that this truth table is obfuscated, the function could possibly be $Z = XYK \vee \overline{X}Y\overline{K} \vee X\overline{Y}K$, $Z = XYK \vee \overline{X}\overline{Y}K \vee X\overline{Y}\overline{K}$, or $Z = XYK \vee \overline{X}\overline{Y}K + \overline{X}Y\overline{K}$ – three functions with distinctly different responses.

The security of this approach depends on the number of LUTs that are mapped for a given design; with more LUTs obfuscated in this manner, the security increases dramatically. For real-world designs, this is not likely to be a limitation, since designs will typically implement several hundred to several thousand device resources. Further analysis of this security is presented in Section V-C.

### B. Key Generation

The first step for the secure bitstream mapping is a low-overhead key generator, such as a nonlinear feedback shift register (NLFSR), which is resistant to cryptanalysis. A Physical Unclonable Function can also be used; though this requires an additional enrollment stage for each device, it has the added benefit of not requiring key storage. Various PUF-based key generators have been proposed, including PUFKY [16], which are amenable to FPGA implementation. Furthermore, using a PUF-based key generator requires that FPGA vendor tools provide floorplanning and/or enable assignment to specific

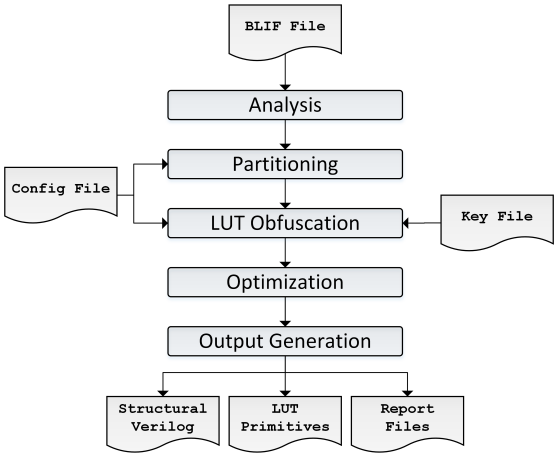


Fig. 3. Software flow leveraging FPGA dark silicon for design security through key-based obfuscation.

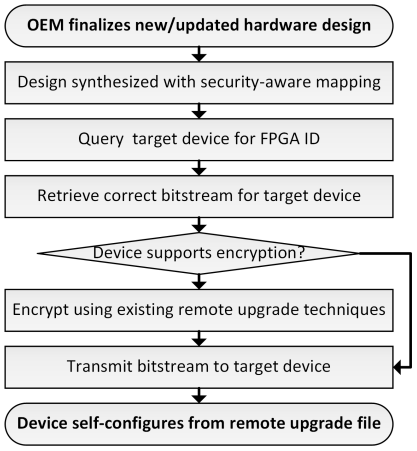


Fig. 4. Remote upgrade of secure and obfuscated bitstreams.

device resources for reproducibility. In general, we refer to the key generator as the system's *CSPRNG*, or cryptographically secure pseudorandom number generator. The specific CSPRNG used depends on the application requirements.

### C. Initial Design Mapping

The second step is the synthesis of the HDL design into LUTs. This can be performed by freely available tools (e.g. ODIN II [17]); it is also possible to configure commercial tools, e.g. Altera Quartus II, by including specific commands into the project settings file (**.qsf*) before compilation; this generates a Berkeley Logic Interchange Format (BLIF) [14] file with technology-mapped LUTs.

### D. Security-Aware Mapping

The security-aware mapping leverages FPGA dark silicon (Section IV-A) for key-based design obfuscation. The software flow is shown in Fig. 3. The following is a brief description of the processing stages:

1) **Analysis:** Inputs to this stage include the BLIF design, as well as the maximum size of LUT supported by the target technology. The circuit is parsed, analyzed, and assembled into a hypergraph data structure. The analysis also determines the current occupancy.
2) **Partitioning:** Inputs to this stage include the hypergraph data structure, as well as the key length. The hypergraph is partitioned into a set of subgraphs which share common inputs/outputs using a breadth-first traversal. Nodes are marked as belonging to a particular subgraph such that those with the greatest commonality are grouped into partitions. The number of partitions is directly proportional to the size of the key.
3) **Obfuscation:** For a device supporting $k$-input LUTs, every LUT with at most $(k-1)$-inputs is obfuscated by implementing a second function using the unoccupied LUT content bits. One additional input is added to the LUT which corresponds to the key bit used to select the correct half of the LUT during operation. The second function can be either template-derived, such as basic logic operations (NAND, NOR, XOR, etc.), or functions implemented in other LUTs in the same design.
4) **Optimization:** In this stage, individual LUTs are optimized using the Espresso Logic Minimizer [18]. The optimized Espresso output is converted back into the internal representation. This process significantly reduces both the output file size, as well as eventual compilation time in the FPGA mapping tool.
5) **Output Generation:** The output file generation can take one of two formats: (a) structural Verilog, which implements the circuit as a series of assignment statements, or (b) using device-specific LUT primitive functions. The second option is preferred because using low-level primitives ensures that the design will be mapped with the specified LUTs.

The number of LUTs per partition is an especially important metric, as it has a direct impact on both the overhead and the level of security. Furthermore, the partitioning and sharing of key bits need to be done judiciously, as a random assignment can potentially dramatically increase area overhead (see Section V-B). Thus, key sharing, when paired with the LUT output generation, is intended to (a) reduce overhead, and (b) strongly suggest to the physical placement and routing algorithms used by the commercial mapping tool to group certain LUTs in a given ALM and/or LAB, and thus minimize area overhead. Ideally, this process could be integrated into a commercial tool itself to enable technology-dependent optimizations.

### E. Communication Protocol and Usage Model

The security-aware mapping procedure creates a one-to-one association between the hardware design and a specific FPGA device, since selection of the correct LUT function responses depends on the CSPRNG output. This means that OEMs must have one unique bitstream for each key in their device database. Therefore, it is critical that the correct bitstream is used with the correct device. Modern FPGAs contain device IDs which can be used for this purpose; alternatively, if a PUF is used as the CSPRNG, the ID can be based on the PUF response. Using existing FPGA mapping software, generating a large number of bitstreams will take considerable time; however, with modifications to the CAD tools, the security-aware mapping can be done just prior to bitstream generation, so that the design does not need to be rerouted.

The initial device programming, prior to distribution in-field, may be done by a (potentially untrusted) third party. The third party is able to read the device ID, but does not require access to the key database. Similarly, device testers do not need access to the key, merely the ability to read the ID. This allows OEMs to keep the ID/key relation secret. Once the device is in field, the remote upgrade procedure differs slightly from the initial in-house programming. The typical upgrade flow is shown in Fig. 4. After finalizing the updated hardware design, it is synthesized using the security-aware mapping procedure. Target devices are queried to retrieve the FPGA ID; if the device supports encryption, the bitstream can be encrypted. Next, the bitstream is transmitted to the device, and the device reconfigures itself using its built-in reconfiguration logic.

## V. Overhead and Security Analysis

In this section, we describe the experimental setup, present the hardware overhead, and analyze the level of security.

### A. Experimental Setup

To obtain area, power, and latency overhead results, BLIF files were generated following the procedure described in Section IV-C for an Altera Cyclone V device. Note that BLIF files cannot be generated if any encrypted IP cores are used in the design. Moreover, they do not support certain hardware elements (e.g. asynchronous reset), so any design containing these signals will not be functionally equivalent to the original implementation. The Quartus mapper will produce a warning if this occurs during BLIF generation.

Therefore, the experiments are limited to those files without the offending signals or encrypted IP. Additionally, for designs which utilize hardened IP blocks within the FPGA, such as adders or shift registers, these resources are instead implemented in LUTs when written to BLIF. We refer to this as the *intermediate representation*. Note that this can result in significant overhead when mapped back to the FPGA, even before undergoing the security-aware mapping procedure. The effect is not seen in purely combinational circuits, and therefore the intermediate overhead values are not listed. However, for larger "IP" cores mapped to FPGA, the effect is significant, so the intermediate overhead values are reported; this represents the overhead due to the conversion from the original HDL code to a technology mapped BLIF file. For these cores, the security-aware mapping overhead should be compared to the intermediate mapping results, and *not* the original mapping.

Once the BLIF was generated, the circuits were then processed by the secure mapping tool to create obfuscated LUTs, and then written to file using the original (unsecured) and mapped (obfuscated) Verilog outputs. For the obfuscation, random functions were used. Both outputs, along with the original benchmarks, were simulated using Altera ModelSim and the same test vectors. All benchmarks were found to be functionally equivalent when the correct key was provided to the obfuscated versions; incorrect keys resulted in different output, as expected. The Verilog files were then mapped to the same Cyclone V device, from which we obtained the power (estimated using PowerPlay Power Analyzer), performance (estimated using TimeQuest Timing Analyzer), and area (obtained from the compilation report). These results were then compared with the other mapping results to find the overhead.

### B. Overhead Analysis

Table III lists the initial (pre-obfuscation) and resulting (post-obfuscation) technology mappings for LUT usage by function input, the number of ALUTs, ALMs, LABs, and the Occupancy, as computed by Eqn. 4. Note that in both mappings, the number of ALUTs (Column *W*) may differ slightly from the sum of LUTs in that row; this is because the number of 1 to 7 input LUTs is obtained from post-synthesis results, whereas the total number of ALUTs is post-fit. Both sets of results are shown for comparison purposes.

The addition of a single key bit input to LUTs 1 to 5 resulted in an increase of ALUT Occupancy from an average 41% to 54%. However, the ALM Occupancy decreased, from 88% to 72%. LAB Occupancy also decreased, though less significantly, from 74% to 71%. Therefore, while the ALUT Occupancy did increase, the Overall Occupancy remained nearly equivalent (Table III). The decrease in ALM and LAB Occupancy also implies a moderate area overhead, which in terms of ALUTs was under 10% for the combinational benchmarks. In terms of ALMs, however, the area overhead was higher, roughly 36%. Less dense packing also implies increased routing/interconnect delay, which manifests as an average 1.2x reduction in $f_{max}$. However, the effect on power consumption was very low, requiring an average 1.02x more power for the secured version.

For larger designs, BLIF conversion was possible for 3 of the 5 "IP" cores from Section III, e.g., AES, Salsa20, and the AltOR32. The version of AES mapped here differs from the full encryption core from Section III, in that it utilizes a number of 6-input LUTs, rather than embedded memory blocks, to hold the AES substitution box (SBOX) values. This allows us to obtain the BLIF representation, since no encrypted IP is involved. Similarly, we use a "Lite" version of the AltOR32 which does not instantiate memories for instruction and data caches, and refer to this as "AOR32-L".

Results for pre- and post-secure mapping for these three cores are shown in Table IV. Compared to the purely combinational circuits, we observed larger percent increases in the area overhead, even when comparing to the intermediate mapping result (Section V-A). For the AES core, this was 25%, 21%, and 18% for ALUTs, ALMs, and LABs, respectively. Unlike the combinational benchmarks, AES had only a 6% reduction in $f_{max}$, and 2% increase in power consumption, similar to the combinational circuits. Salsa20 and AOR32-L both had significantly higher area overhead (e.g. 54% more ALUTs in Salsa20 and 67% more ALUTs in AOR32-L), and 12% reduction in $f_{max}$ for both (Table IV).

We believe that the larger increase in ALUT usage for the large IP cores compared to the small combinational circuits is due primarily to the conversion of arithmetic mode ALUTs, which use two 4-input LUTs with dedicated hardware full adders [11] to normal mode ALUTs with adders realized in numerous other LUTs. This was not an issue in the combinational benchmarks, as they used only normal or occasionally extended mode ALUTs. Based on these results, we believe

TABLE III
ORIGINAL AND SECURE MAPPING RESULTS FOR SMALL COMBINATIONAL BENCHMARKS

| Name | Original Mapping* | | | | | | | | | | Secured Mapping* | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\leq 2$ | 3 | 4 | 5 | 6 | 7 | W | X | Y | Z (%) | $\leq 2$ | 3 | 4 | 5 | 6 | 7 | W | X | Y | Z (%) |
| alu4 | 18 | 31 | 42 | 55 | 38 | 4 | 185 | 115 | 14 | 30.3 | 12 | 17 | 38 | 61 | 63 | 0 | 191 | 128 | 16 | 33.1 |
| apex2 | 74 | 100 | 176 | 259 | 54 | 6 | 664 | 336 | 49 | 25.6 | 24 | 93 | 119 | 243 | 308 | 6 | 793 | 594 | 87 | 27.5 |
| apex4 | 78 | 79 | 132 | 224 | 47 | 14 | 569 | 325 | 45 | 24.5 | 29 | 59 | 159 | 171 | 143 | 0 | 562 | 388 | 57 | 24.3 |
| ex5p | 96 | 61 | 62 | 96 | 52 | 6 | 371 | 192 | 26 | 25.6 | 38 | 33 | 89 | 112 | 93 | 2 | 367 | 245 | 37 | 24.3 |
| ex1010 | 110 | 62 | 158 | 273 | 96 | 12 | 704 | 418 | 57 | 26.2 | 11 | 116 | 106 | 191 | 211 | 1 | 636 | 464 | 66 | 26.5 |
| misex3 | 54 | 79 | 92 | 178 | 65 | 12 | 475 | 271 | 32 | 31.5 | 24 | 46 | 105 | 134 | 122 | 1 | 432 | 283 | 37 | 30.2 |
| pdc | 241 | 212 | 371 | 410 | 328 | 26 | 1585 | 978 | 140 | 24.7 | 56 | 225 | 374 | 625 | 685 | 17 | 2002 | 1462 | 228 | 25.0 |
| seq | 93 | 125 | 159 | 271 | 72 | 7 | 723 | 396 | 53 | 25.9 | 38 | 73 | 194 | 248 | 153 | 2 | 708 | 465 | 61 | 27.8 |
| spla | 237 | 204 | 360 | 405 | 284 | 19 | 1502 | 916 | 130 | 24.4 | 46 | 234 | 310 | 543 | 663 | 14 | 1810 | 1339 | 206 | 25.7 |
| **Avg.** | **111** | **106** | **172** | **241** | **115** | **12** | **758** | **439** | **61** | **26.5** | **28** | **102** | **166** | **259** | **271** | **5** | **833** | **596** | **88** | **27.2** |

*W, X, Y, and Z represent the total number of ALUTs, ALMs, LABs, and Total Occupancy, respectively.
Columns labeled *1..7* represent the total number of LUTs into which that size function has been mapped.

that tighter integration of the secure mapping process with the mapping tool would yield further improved occupancy, full utilization of hard IP resources within the FPGA, and lower latency overhead from additional routing/interconnect delay:

- The relatively low routing resource utilization strongly implies that the reduction in ALM and LAB occupancy is not entirely due to routability constraints.
- The current hypergraph partitioning stage can only make *suggestions* to the placement tool by grouping certain nodes; ultimately this cannot be directly controlled unless the two tools are tightly integrated.
- The conversion from original design to BLIF and back can incur significant overhead and running the secure mapping process on the FPGA mapping tool's internal circuit representation could avoid this issue entirely.

### C. Security Analysis

For security analysis, we assume the attacker intends to reverse engineer the design or perform malicious modification and reprogram the device.

*1) Brute Force Attack:* A brute force attack represents the most challenging and time consuming attack. The basic requirements of the brute force attack on the obfuscated bitstream differ from a typical cipher, because the attacker needs not only the bitstream (and a means to apply various keys) but also knowledge of the bitstream structure and the test patterns with known responses. Knowledge of the bitstream structure is required to identify LUT content bits and LUT interconnection through FPGA routing. Therefore, if a design has 128 LUTs and a 128 bit key is used, the attacker must try $2^{128}$ combinations to determine the key and reverse engineer the bitstream. If bitstream encryption was used, the attacker must break the bitstream encryption and the security-aware mapping, before reverse engineering the bitstream. Furthermore, a larger number of LUTs (e.g. $>$1000) are typical for many designs, leading to a much larger search space.

If the attacker does not know the entire bitstream format (e.g. they can identify the location of the LUT content bits, but not how LUTs are connected), and therefore does not know which input represents the key bit, the search space increases dramatically. Finding the location of the LUT content bits is feasible through template attacks (as described in Section II. The difficulty lies in the number of potential combinations of the various LUT content bit organizations, which can be counted as follows:

1) For each *k*-input LUT, the ordering is important; this contributes a factor of $k!$
2) For each LUT, it is necessary to select the correct half, depending on if the value of the key bit is "0" or "1", multiplying this result by 2.
3) These factors are multiplied by the number of LUTs in the design.

However, this is not the complete picture. Recall that some fraction of LUTs which currently have 6 inputs are in fact *5-input* LUTs with 1 additional key bit. The remaining 6-input LUTs did not receive a key bit, because they were already at maximum occupancy for the given technology. This requires the attacker to differentiate between keyed and unkeyed maximally-occupied LUTs. We denote the total number of *k*-input LUTs as $\#(LUT)_k$, and the number of keyed (5+1) input LUTs as $\#(LUT)_{k-1}$, which multiply the previous factors to yield Eqn. 5.

$$TC = \binom{\#(LUT)_k}{\#(LUT)_{k-1}} \times \sum_{k=2}^{N} 2 \times k! \times \#(LUT)_k \qquad (5)$$

It follows that an attacker will have significantly more difficulty reverse engineering the design when the complete bitstream format is not known and the design is obfuscated, than it is to use template attacks to determine the format of an unobfuscated bitstream. Even if the format is known, the previous analysis shows that the difficulty of brute forcing the key depends on the key length, as long as there are sufficient LUTs on which to apply a key bit input (e.g. $\geq$ 128 LUTs).

*2) Known Design / Bitstream Tampering Attacks:* A known design attack can enable an attacker to reverse engineer the bitstream format, and potentially the IP, due to insufficient protection offered by bitstream encryption. This may lead to not only IP piracy, but also malicious modification and unauthorized system reprogramming with the tampered bitstream.

A moving target defense is known to provide robust protection against known design attacks, since they rely on an underlying assumption of consistency between subsequent trials – in this case, compilations of the bitstream. For FPGA, the mapping format is traditionally consistent over time for the same device architecture. Using the proposed technique,

TABLE IV
ORIGINAL, INTERMEDIATE, AND SECURE MAPPING RESULTS FOR THREE LARGE IP BLOCKS.

| | | $\leq 2$ [1] | 3 | 4 | 5 | 6 | 7 | W[2] | X | Y | Z (%) | $f_{max}$ (MHz) | Power (mW) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AES | Original | 181 | 78 | 53 | 66 | 905 | 22 | 1306 | 1117 | 153 | 32.5 | 168.8 | 545.8 |
| | Intermed.[3] | 169 | 76 | 58 | 76 | 914 | 33 | 1326 | 1139 | 154 | 33.1 | 167.8 | 545.8 |
| | Secured | 29 | 170 | 214 | 152 | 1067 | 21 | 1656 | 1376 | 182 | 34.1 | 158.3 | 555.5 |
| Salsa20 | Original | 1620 | 8 | 1031 | 5 | 171 | 1 | 2836 | 1656 | 224 | 11.9 | 152.6 | 550.0 |
| | Intermed. | 916 | 658 | 507 | 1254 | 1381 | 46 | 4463 | 2929 | 367 | 31.6 | 112.0 | 549.4 |
| | Secured | 289 | 732 | 1434 | 1255 | 3156 | 52 | 6869 | 5023 | 642 | 33.5 | 100.4 | 560.0 |
| AOR32-L | Original | 199 | 141 | 153 | 190 | 973 | 65 | 1624 | 1424 | 204 | 29.0 | 94.7 | 531.1 |
| | Intermed. | 337 | 234 | 221 | 355 | 995 | 64 | 2211 | 1616 | 244 | 29.8 | 82.6 | 528.8 |
| | Secured | 421 | 253 | 1091 | 518 | 1315 | 87 | 3686 | 2496 | 326 | 30.2 | 74.0 | 538.1 |

[1] Columns labeled *1..7* represent the total number of LUTs into which that size function has been mapped.
[2] *W*, *X*, *Y*, and *Z* represent the total number of ALUTs, ALMs, LABs, and Total Occupancy (%), respectively.
[3] Intermed. results show the overhead due to the HDL to BLIF conversion; this gives "Secured" a fair comparison.

we note several aspects that *do* change between compilations, violating the assumption of consistency and making known design attacks impractical:

- If a strong PUF is used as a key generator, then the key will change each time the design is compiled by issuing a different challenge vector. If an alternative CSPRNG is used, a different seed can be used each time to generate a different key sequence. Using a different key will produce drastically differing bitstreams.
- The location of the key input to each LUT also changes each time, leading to a number of different configurations of the content bits equal to the number of inputs. This information is encoded in the routing bits, and therefore its security is independent of the key generator.
- The second function mapped into spare LUT content bits also changes each time. Thus, not only do half the content bits change each time, they are also permuted in different ways. This also does not depend on the key generator, providing another independent layer of security.

In summary, the key, the LUT content bits, and their ordering will mutate each time the design undergoes the security-aware mapping procedure. This approach therefore provides robust protection against known design attacks. This also prevents targeted malicious modifications, because the meaning of individual bits changes during each compilation.

## VI. CONCLUSION

We have presented a novel, low-overhead design obfuscation technique aimed at securing FPGA bitstreams. The approach does not require any modification in FPGA architecture and hence can be readily used with existing FPGA devices – both before in-field deployment, and subsequently via standard remote upgrade procedures. It is therefore attractive for both FPGA vendors as well as system designers, who pursue system integration with FPGA devices already in the market. Moreover, the tool flow can seamlessly integrate with commercial FPGA mapping tools, such as Altera Quartus II. While the process does minimally affect the design optimization process for FPGA, and hence incurs modest performance and negligible power overhead, it provides mathematically robust security against several major threats to FPGA-based systems, which are not fully protected against by traditional bitstream encryption approaches. Nevertheless, it can still be used in conjunction with encryption for additional security. The technique capitalizes on the unoccupied space in an FPGA's lookup tables – which we call the FPGA's *dark silicon*, making area-efficient use of existing resources, rather than consuming a large number of additional logic elements for security enhancement. The approach is scalable to larger designs. Future work will focus on refining the tool to further reduce overhead, as well as built-in support for outputting low-level primitives for different FPGA platforms.

## REFERENCES


[1] P. Garcia *et al.*, "An Overview of Reconfigurable Hardware in Embedded Systems," *EURASIP Journal on Embedded Systems*, 2006.
[2] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "FPGA-oriented Security," *Introduction to Hardware Security and Trust/eds. M. Tehranipoor and C. Wang. Springer*, pp. 195–231, 2011.
[3] G. Gogniat *et al.*, "Reconfigurable Hardware for High-Security/High-Performance Embedded Systems: the SAFES perspective," *TVLSI*, vol. 16, no. 2, pp. 144–155, 2008.
[4] P. Kocher *et al.*, "Security as a New Dimension in Embedded System Design," in *DAC*. ACM, 2004, pp. 753–760.
[5] R. S. Chakraborty *et al.*, "Hardware Trojan Insertion by Direct Modification of FPGA Configuration Bitstream," *Design & Test, IEEE*, vol. 30, no. 2, pp. 45–54, 2013.
[6] A. Moradi *et al.*, "On the Vulnerability of FPGA Bitstream Encryption Against Power Analysis Attacks: Extracting Keys from Xilinx Virtex-II FPGAs," in *CCS*, 2011, pp. 111–124.
[7] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending Piracy of Integrated Circuits," in *DATE*. ACM, 2008, pp. 1069–1074.
[8] É. Rannaud, "From the Bitstream to the Netlist," in *FPGA*. ACM, 2008, pp. 264–264.
[9] P. Swierczynski *et al.*, "FPGA Trojans Through Detecting and Weakening of Cryptographic Primitives," *IEEE TCAD*, vol. 34, no. 8, pp. 1236–1249, 2015.
[10] H. Esmaeilzadeh *et al.*, "Dark Silicon and the End of Multicore Scaling," in *ISCA*. IEEE, 2011, pp. 365–376.
[11] Altera, "Cyclone V Device Handbook," Tech. Rep., Dec. 2015.
[12] R. Tessier and H. Giza, "Balancing Logic Utilization and Area Efficiency in FPGAs," in *FPL*. Springer, 2000, pp. 535–544.
[13] A. Palchaudhuri and A. S. Dhar, "Efficient Implementation of Scan Register Insertion on Integer Arithmetic Cores for FPGAs," in *VLSID*. IEEE, 2016, pp. 433–438.
[14] S. Yang, *Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0*. Microelectronics Center of North Carolina (MCNC), 1991.
[15] EPFL, "The EPFL Combinational Benchmark Suite." [Online]. Available: http://lsi.epfl.ch/benchmarks
[16] R. Maes, A. Van Herrewege, and I. Verbauwhede, "PUFKY: A Fully Functional PUF-based Cryptographic Key Generator," in *CHES*. Springer, 2012, pp. 302–319.
[17] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," in *FPL*. Springer, 1997, pp. 213–222.
[18] R. L. Rudell, "Multiple-Valued Logic Minimization for PLA Synthesis," DTIC, Tech. Rep., 1986.