Article

Bitstream Security in FPGAs

Raj Parikh *

Posted Date: 8 May 2025

doi: 10.20944/preprints202505.0607.v1

Keywords: FPGA bitstream security; Physical Unclonable Functions (PUFs); side-channel attack mitigation; hardware trojans; secure boot and configuration

 doi:10.20944/preprints202505.0607.v1

Article

Bitstream Security in FPGAs

Raj Parikh

Altera Corporation, rparikh356@gmail.com

Abstract: Field-Programmable Gate Arrays (FPGAs) rely on configuration bitstream – binary files that define the FPGA's logic and connections. Securing these bit streams is critical because anyone who intercepts or alters them can steal or sabotage the design. Adversaries are motivated by various goals: cloning hardware, stealing intellectual property (IP), bypassing feature locks, or inserting hardware. A compromised FPGA bitstream can lead to loss of competitive IP, financial losses from piracy, or even safety hazards in critical systems. This paper provides an in-depth study of FPGA bitstream security, covering common threats, protection techniques, emerging trends, and best practices for developers to maintain robust security.

Keywords: FPGA bitstream security; Physical Unclonable Functions (PUFs); side-channel attack mitigation; hardware trojans; secure boot and configuration

---

1. Introduction

A. Key Threats and Vulnerabilities in FPGA Bitstreams

Modern FPGAs (especially SRAM-based devices) load their bitstream from external memory on power-up; making the bitstream data a prime target during storage or transit. Several threats and vulnerabilities have been identified [6]

B. Reverse Engineering & IP Theft

An attacker can capture a bitstream (e.g., by eavesdropping on the FPGA configuration lines or dumping an external flash/PROM) and then attempt to reconstruct the design. Although raw bitstream bits aren't human-readable, tools can reverse-engineer the netlist from an FPGA bitstream [1]. By understanding the netlist, a pirate can duplicate proprietary algorithms or logic. Unlike ASICs that require destructive physical examination to reverse engineer, FPGAs allow easier extraction by simply reading the configuration. This makes sensitive designs vulnerable to IP theft if bitstreams are unprotected [1].

C. Cloning and Overbuilding

If an attacker can obtain the bitstream of a design, they can clone the hardware by programming it onto another identical FPGA, producing counterfeit products [1]. This is a major form of FPGA piracy – essentially "copy-paste" of hardware IP. Cloning requires minimal technical effort once the bitstream is acquired; for example, a competitor or a rogue contract manufacturer could simply program additional FPGAs with the stolen bitstream to create unauthorized copies. A related threat is overbuilding, where a contract manufacturer illicitly produces extra units beyond the authorized number using the provided bitstream [1].

D. Bitstream Tampering & Hardware Trojans

Attackers may not only copy but also modify bitstreams. Tampering with the bitstream can alter the FPGA's behavior – either to disable certain functionality, cause malfunctions, or insert malicious logic (hardware Trojan horses) [3]. Such tampering could be done in the supply chain or after deployment if an attacker gains access to the configuration memory. For example, Rahman et al. showed that if an attacker extracts the FPGA's decryption key, they can modify a decrypted bitstream to add a Trojan and re-encrypt it; the tampered bitstream will load successfully on the target device, granting the attacker a hidden foothold in the system [3]. Real-world incidents underline this threat:

 doi:10.20944/preprints202505.0607.v1

bitstream manipulation has been used to sabotage cryptographic modules and even to bypass a system's root-of-trust [3,6]. Without proper integrity checks, FPGAs have no way to detect unauthorized changes to their configuration.

E. Unauthorized Feature Activation (Service Theft)

Some FPGA-based products use bitstreams or partial reconfigurations to enable paid features or upgrades. Attackers might pirate features by exploiting the bitstream [6]. For instance, if certain premium features are unlocked via a license bit or soft-core in the bitstream, a determined adversary could flip those bits or use an extracted design to always enable the feature, bypassing payment. This threat is essentially a form of tampering targeting revenue streams, and it overlaps with cloning/piracy (unauthorized use of IP) [6].

F. Side-Channel and Key Extraction Attacks

Even when bitstreams are encrypted, the security is only as strong as the secrecy of the cryptographic keys. Attackers have developed side-channel attacks (such as power analysis or electromagnetic analysis) to extract the secret decryption keys from FPGA devices. For example, researchers demonstrated a power analysis attack that recovered the full 128-bit AES bitstream key from an FPGA with only 30,000 power traces [5]. Other physical attacks, like probing the device's memory or using advanced laser techniques, have also succeeded in reading out keys or configuration data on certain FPGA chips [5]. These vulnerabilities illustrate that an FPGA's security features can be undermined if an attacker can extract secret keys or exploit silicon-level backdoors [5]. In summary, bitstream threats range from passive attacks (eavesdropping to copy a design) to active attacks (tampering to introduce faults or trojans). The lifecycle of an FPGA (design, manufacturing, deployment, and end-of-life) presents multiple opportunities for adversaries – from intercepting bitstreams in transit, to illicit copying during manufacturing, to field attacks like reverse engineering or fault injection. Any comprehensive security strategy must therefore address confidentiality (preventing IP theft/cloning) and integrity/authenticity* (preventing tampering or unauthorized use).

In summary, bitstream threats range from passive attacks (eavesdropping to copy a design) to active attacks (tampering to introduce faults or trojans). The lifecycle of an FPGA (design, manufacturing, deployment, and end-of-life) presents multiple opportunities for adversaries – from intercepting bitstreams in transit, to illicit copying during manufacturing, to field attacks like reverse engineering or fault injection. Any comprehensive security strategy must therefore address confidentiality (preventing IP theft/cloning) and integrity/authenticity (preventing tampering or unauthorized use) [6].

2. FPGA Bitstream Protection Techniques and Technologies

To counter the above threats, FPGA vendors and researchers have developed several techniques to secure bitstreams. Key methods include encryption, authentication, obfuscation, and robust key management, often used in combination for defense-in-depth [4,6,10].

This section outlines these methods:

A. Bitstream Encryption (Confidentiality Protection)

Encryption is the primary mechanism to protect bitstream confidentiality. Modern FPGAs include on-chip decryption engines, typically implementing the Advanced Encryption Standard (AES), often at 256-bit strength [4,10]. The encrypted bitstream is stored externally (in flash or PROM), and on power-up, the FPGA decrypts the stream on-the-fly using a secret key stored in on-chip fuses, battery-backed RAM (BBRAM), or flash [4]. Some devices even support unique per-device keys, enhancing granularity in secure deployment [6]. FPGAs support 256-bit AES encryption of the bitstream in many. The encrypted bitstream is stored externally (in flash or PROM), and upon power-up the FPGA uses the internally stored key to decrypt the stream on-the-fly as it configures the device. Because decryption happens within the FPGA's secure hardware, the plaintext bitstream never

 doi:10.20944/preprints202505.0607.v1

appears on any external interface. As long as the encryption key remains secret, an attacker who reads the bitstream data from SPI flash or intercepts it during loading sees only ciphertext – preventing reverse engineering or cloning of the design.

Encryption alone does not ensure integrity. Without authentication, a modified ciphertext could still load, potentially resulting in subtle or malicious changes [6]. Hence, encryption is commonly paired with authentication. FPGA vendors typically allow the key to be stored in non-volatile one-time programmable fuses or in battery-backed RAM on the device. High-end FPGAs often use eFUSE or on-chip flash for permanent key storage, avoiding the need for a battery. Lower-cost FPGAs sometimes rely on a battery-backed SRAM key (BBRAM) – which works but requires maintaining battery power to retain the key when the device is off ither way, the key is not meant to be readable from outside the FPGA. During manufacturing, the secret key is injected into the FPGA's secure memory, and from then on, the device will automatically decrypt any incoming encrypted bitstream using that key. Some devices even support unique keys per device, so that each FPGA has its own encryption key to decrypt a bitstream tailored for it (this limits the impact of a leaked key to one device).

Limitations

Encryption alone does not guarantee that the bitstream hasn't been altered; it only protects against unauthorized readout. If an attacker somehow obtains the encryption key (via side-channel attack or chip tampering), the encryption can be completely defeated. Moreover, without additional measures, encryption by itself doesn't prevent a modified ciphertext from loading – meaning an attacker who can tweak the encrypted bitstream (even without fully decrypting it) might cause specific changes in the FPGA configuration if there's no integrity check. For these reasons, encryption is often paired with authentication. Nonetheless, enabling AES encryption is a fundamental first step to thwart casual cloning and reverse engineering – it forces the adversary to undertake a much more sophisticated attack (extracting or cracking the key) rather than simply copying bytes from a memory chip.

B. Bitstream Authentication and Integrity Checking

Authentication ensures that a bitstream originates from a trusted source and has not been modified in transit. This defends against tampering and unauthorized use [4,6].An authenticated bitstream means the FPGA will only load configuration data that has a valid cryptographic signature or hash, proving it comes from a trusted source and has not been tampered with. This protects against malicious modifications, trojans, or use of unauthorized bitstreams. Two common implementations are Hash-based Message Authentication Codes (HMAC) and digital signatures.

C. HMAC/SHA Authentication

Cryptographic hash functions (e.g., SHA-256 or HMAC-SHA) are computed for the bitstream and verified at boot. A mismatch aborts configuration. This method is widely used in devices like Xilinx UltraScale+ and Intel Stratix 10 [4]. The FPGA, during configuration, recomputes the hash/HMAC using a stored secret key and compares it to the expected value. If they do not match, the FPGA refuses to execute the bitstream. The bitstream file includes an HMAC digest (computed with a secret key known to the device), and on boot the FPGA verifies this digest before proceeding. If authentication fails (digest mismatch), the device will abort configuration, thus preventing any unauthorized or corrupted bitstream from running. It is best practice to use encryption and authentication together, performing an "encrypt-then-authenticate" scheme.

D. Digital Signatures and Secure Boot Chains

High-end FPGAs support secure boot with public-key cryptography (RSA or ECDSA) [4,6]. A root-of-trust bootloader verifies a signed first-stage bootloader, which then authenticates the bitstream. This prevents booting rogue code and enforces a trust chain. The FPGA or its boot ROM holds the corresponding public key and will verify the signature before allowing the configuration to load. The use of asymmetric cryptography means that even if an attacker intercepts the bitstream,

 doi:10.20944/preprints202505.0607.v1

they cannot forge a new valid bitstream without the private signing key. Secure boot typically works as a chain-of-trust: a small on-chip ROM bootloader verifies a signed first-stage bootloader, which then verifies the FPGA bitstream or application code, and so on. In summary, authentication complements encryption by defending against bitstream tampering and reuse of unauthorized code. Encryption stops reads of the bitstream, while authentication stops writing of rogue bitstreams. Designers concerned with security should enable these features so that any attempt to modify a bitstream – whether by flipping bits or inserting trojans – will be detected and blocked.

E. Obfuscation and Design Concealment Techniques

When encryption isn't available, or as a complementary method, obfuscation techniques complicate reverse engineering efforts [6]. Beyond cryptographic protection, various obfuscation techniques can make it harder for an attacker to reverse-engineer or misuse a bitstream. These techniques do not rely on secret keys, but rather on making the design representation intrinsically difficult to understand or modify. They serve as a secondary line of defense, especially in scenarios where full encryption might not be available (e.g., some low-cost FPGAs) or to augment security in depth. Key obfuscation approaches include:

F. Bitstream Scrambling

Older FPGAs used fixed scrambling schemes (e.g., XOR masks), but these are weak and easily defeated. Most vendors have phased this out in favor of robust encryption [6]. FPGA vendors historically have used simple scrambling on the bitstream format – a fixed XOR mask or bit permutation – to prevent casual interpretation of the bitstream. For example, older FPGAs (before AES support) often had a proprietary bitstream encoding. However, determined adversaries can reverse-engineer these schemes, and they are not cryptographically secure. Scrambling might stop a naive attacker but offers little resistance against serious reverse engineering efforts. It is now largely superseded by real encryption.

G. Logic Encryption / Logic Locking

This technique inserts key-controlled logic during design, so the FPGA only functions with the correct runtime key. It increases security against cloning and reverse engineering [6]. However, SAT-based attacks can sometimes break such protections [10]. This is a design-time obfuscation where extra "key" inputs or gates are added to the FPGA design such that the circuit only functions correctly if the proper secret key bits are applied. For instance, a designer can insert key-gated logic or "lock" certain critical functional blocks with a user-defined key. The correct key is programmed into the FPGA (for example, stored in internal registers or supplied at runtime); without it, the circuit will malfunction or output wrong results. This technique can thwart an attacker who manages to copy the bitstream – the cloned FPGA would not operate correctly without knowing the secret key. Logic locking has been researched extensively and can increase the difficulty of reverse engineering. However, sophisticated attackers can sometimes defeat logic locking via SAT attacks or bypass logic if they manage to extract the key, so it is not foolproof method. It's a useful supplemental protection, especially when combined with encryption (the key for logic locking can be another layer of security).

H. Hardware Camouflaging and Dummy Logic

This method inserts misleading or unused logic to confuse attackers. It works as a "smokescreen" against netlist reconstruction tools [6]. Designers can introduce "camouflaged" logic elements (look-alike dummy gates or routing that don't affect functionality) to confuse reverse engineering. Dummy routes and unused logic might be deliberately inserted so that an extracted netlist contains misleading or extraneous circuitry. The goal is to make it harder to discern the true design intent. Some FPGAs or design flows may allow placing decoy state machines or mix up LUT configurations in ways that only correct initialization yields a working design.

 doi:10.20944/preprints202505.0607.v1

I. Partitioning and Partial Reconfiguration Obfuscation

Researchers have proposed two-stage bitstreams and runtime assembly of designs using PUF-generated keys [8,10]. This adds complexity and layering, frustrating attacker attempts to reconstruct the complete design. An emerging idea is to split the design into parts and only combine them at runtime via partial reconfiguration or multi-boot sequences, possibly under authentication. For instance, one academic approach uses a two-stage configuration where the full design only becomes functional after a second stage bitstream is loaded that "unlocks" certain features using a PUF-generated key. By dividing the bitstream, an attacker must defeat multiple layers (and possibly multiple keys) to get the whole design. Overall, obfuscation techniques increase the effort required for an attacker to clone or understand an FPGA's bitstream. They are typically used in addition to encryption/authentication, not as a replacement (except in low-end devices where encryption isn't available, in which case clever obfuscation and secure protocols might be the only option). It's important to note that obfuscation security is often "security through complexity" – it raises the bar but does not provide mathematical guarantees. Thus, cryptographic protection remains the cornerstone of bitstream security, with obfuscation as a valuable adjunct in the security toolbox.

J. Key Management and Physical Security (Securing the "Secrets")

The strength of encryption and authentication in FPGA security ultimately hinges on protecting the keys and the configuration process from disclosure or manipulation. Robust key management and physical anti-tamper measures are therefore critical elements of bitstream security:

K. On-Chip Key Storage

Keys are stored in eFUSE, flash, or BBRAM. The best practice is to use one-time programmable memory such as eFUSE, making it tamper-resistant and non-recoverable [4,6]. Best practice is to use one-time programmable memory (eFUSE or similar) for keys so that they are non-volatile and cannot be inadvertently erased or read out. The key is typically written once at manufacturing (and sometimes can be updated by blowing new fuses to invalidate the old key, depending on device capabilities). Once programmed, the key is not directly accessible through any user interface. Designers should ensure that the key programming interface is secure (e.g., use encrypted key programming files and perform this in a trusted environment).

L. Physical Unclonable Functions (PUFs)

PUFs leverage inherent manufacturing variations in silicon to generate unique identifiers or cryptographic keys for each chip. Modern FPGAs integrate PUF circuits to either generate the bitstream key on the fly or protect it. The PUF output is used to encrypt or "wrap" the actual AES key, ensuring that even if an attacker reads out the non-volatile memory, they obtain only PUF-encrypted data, which cannot be decrypted off-chip. This approach effectively turns the FPGA into its own root of trust, making each device physically unclonable [4,8].

This mitigates the risk of stored key extraction because the key is never stored in plaintext form—it's derived from silicon each time. One caveat is that PUF responses can be noisy and require helper data or error correction, but vendors have incorporated robust circuits to make PUF-derived keys reliable for encryption use [8].

However, recent studies have shown that certain PUF architectures, like Transient Effect Ring Oscillator (TERO) PUFs, are vulnerable to side-channel attacks. Tebelmann et al. demonstrated that electromagnetic (EM) analysis could successfully extract frequency-domain information from TERO PUFs using Short-Time Fourier Transform (STFT) techniques, reducing entropy and revealing exploitable patterns [11].

To address such concerns, Aghaie and Moradi introduced a side-channel resistant architecture known as TI-PUF, which uses threshold implementation (TI) masking. Their implementation makes it possible to protect strong PUF designs against side-channel leakage during response generation, enhancing resistance against EM and power attacks in FPGA-based applications [11].

 doi:10.20944/preprints202505.0607.v1

M. Tamper Detection and Response

High-end FPGAs offer voltage glitch detection, temperature sensors, and active tamper pins. These hardware features zeroize cryptographic keys on tamper detection [4,6]. Examples include voltage/glitch sensors, temperature sensors, and active tamper pins that zeroize keys if triggered. Additionally, an enclosure might have tamper-evident seals or an active mesh that triggers key erasure if someone attempts to probe the chip. If a device detects a tamper event (cover removal, sudden clock/voltage glitches, etc.), it can lock down or wipe critical storage to prevent an attacker from gleaning the bitstream or keys. Implementing such measures adds a layer of protection, especially against skilled adversaries with physical access.

N. Side-Channel Attack Countermeasures

Side-channel attacks such as Differential Power Analysis (DPA) and Electromagnetic Analysis (EMA) can extract secret keys or PUF responses based on physical leakage. To mitigate these, FPGA vendors have implemented hardware-level defenses such as current masking, randomized clocking, dummy cycles, and noise injection in cryptographic cores [4–6]. Developers may also configure FPGAs to only decrypt once at boot or insert non-deterministic delays to prevent repeatable measurements.

Recent academic work expands on these protections. The TI-PUF architecture, proposed by Aghaie and Moradi, represents a breakthrough in side-channel resistance. By applying threshold implementation masking to any PUF design, TI-PUFs maintain full functionality while preventing intermediate variable leakage during evaluation. Their design has demonstrated high resistance to state-of-the-art SCA attacks in practice and is implementable on commercial FPGAs [11].

O. Secure Configuration Protocols

Configuration interfaces like JTAG must be secured or disabled in production. Enforcing encrypted-only bitstreams and authenticated partial reconfigurations is key [4,10]. Protocol-level lockdowns ensure attackers can't reconfigure deployed systems. The interface through which an FPGA is programmed can be a vulnerability if not secured. JTAG, for instance, is a common programming and debug interface on FPGAs. Locking down JTAG is a recommended practice so that attackers cannot use it to read back configuration memory or reprogram the device with a custom bitstream. Vendors allow JTAG access to be password-protected or permanently disabled for security. Likewise, for devices that support partial reconfiguration or remote update, the configuration ports (e.g., SPI, PCAP, etc.) should be secured — many devices have an option to only allow encrypted bitstreams or to require a valid authentication header, preventing an attacker from loading arbitrary partial bitstreams. In essence, key management and physical security are about safeguarding the root secrets (keys) and hardening the device against direct attacks. Even the best encryption algorithm fails if the key is compromised. Therefore, FPGA developers should: choose devices with proven secure key storage (eFUSE/PUF), enable tamper sensors if needed, lock debug ports, and be mindful of side-channel threats. By combining robust confidentiality, encryption, integrity, authentication, and hardware security measures, the FPGA's bitstream can be protected against a wide spectrum of attacks.

3. Emerging Trends in FPGA Bitstream Security

As technology and threats evolve, new trends are emerging to further strengthen FPGA bitstream security. Several notable directions include:

A. Lightweight Cryptography for IoT FPGAs

As FPGAs proliferate into IoT and edge computing platforms, which often have severe power and area constraints, there is a growing need for lightweight cryptographic solutions. While high-end devices implement robust cryptographic engines (e.g., AES-256, RSA), low-cost or minimal-area FPGAs can benefit from compact ciphers like PRESENT or Speck that require fewer logic resources

 doi:10.20944/preprints202505.0607.v1

[4,6]. Similarly, challenge-response authentication protocols using lightweight primitives are being explored to replace traditional RSA-based signing [9].

Although AES remains the industry standard due to its efficiency and widespread support, future variants of lightweight security stacks may become prevalent for cost-sensitive use cases. Companion secure elements may also be introduced to offload boot-time cryptographic operations in ultra-constrained deployments [10].

B. Post-Quantum Bitstream Protection

With the anticipated rise of quantum computing, many conventional public-key algorithms (RSA, ECDSA) may become vulnerable. Consequently, researchers are actively exploring Post-Quantum Cryptography (PQC) schemes—particularly lattice-based cryptographic signatures such as CRYSTALS-Dilithium and Falcon—for future-proof FPGA bitstream signing and key provisioning [4,13].

Since symmetric ciphers like AES-256 remain largely resilient under Grover's algorithm, the critical PQC integration points are digital signatures and key exchange protocols. Several proposed architectures envision FPGA bootloaders and management engines capable of verifying PQ signatures or using PQ key exchange to receive secure updates over-the-air. While no vendor has implemented PQC in commercial silicon as of 2025, trends strongly indicate it will become a mandatory feature in secure FPGAs by the end of the decade [13].

C. Physical Unclonable Functions (and Device Identity)

PUF-based key generation remains one of the most promising trends for strong device authentication and anti-cloning. New variants like Ring Oscillator PUFs, Arbiter PUFs, and Butterfly PUFs are being optimized for higher entropy, reliability, and ML-attack resistance [8,12]. Advanced concepts are trying each bitstream to a device-specific PUF fingerprint, ensuring that the configuration will not function unless deployed on its original chip [13].

PUF-enabled key provisioning also reduces the risk of key extraction, as the actual key is regenerated internally on every power cycle rather than stored statically. However, researchers continue to improve PUF robustness against side-channel and machine learning attacks through techniques like TI-PUF masking [12].

D. Enhanced FPGA Access Control and Monitoring

Recent security architectures propose integrating runtime bitstream integrity checks into the FPGA fabric itself. These include error-correcting code (ECC) verification, configuration hashing, and power anomaly detection circuits to monitor operational security in real time [13]. Such features extend the concept of secure boot into "secure runtime," where tampering during active execution can be detected and mitigated.

Runtime reconfiguration firewalls, secure regions with token-gated reprogrammability, and dynamic lockdown of configuration ports are gaining traction—especially for multi-tenant FPGA cloud environments, where shared logic needs compartmentalized protection [13].

E. Secure Remote Update Protocols

As FPGAs are often field updated (e.g., firmware upgrades), the best practice is to use secure update frameworks. This isn't a new concept, but it's becoming more standardized. Instead of ad-hoc methods, there are trends to adopt secure boot loaders that handle FPGA partial reconfigurations with the same level of authenticity checks as the initial boot. Protocols using TLS or SSH for transporting bitstreams are recommended over unencrypted channels. The industry is likely to see more integration of secure update services (possibly tied into cloud device management platforms) that ensure bitstreams in transit are encrypted and signed. Additionally, features like rollback prevention (to avoid an attacker installing an old bitstream with a known vulnerability) and version control are being integrated, analogous to secure boot in CPUs.

F. Integration of Cryptographic Co-processors

 doi:10.20944/preprints202505.0607.v1

Some next-gen FPGAs include hard processor cores and even isolated management engines. A trend is to include dedicated secure elements within the FPGA package – for example, a secure microcontroller core or a TPM-like block that can manage keys and attest the FPGA's state. This could support advanced features like measured boot (where the FPGA's configuration is hashed and reported to an external entity for verification). With FPGAs moving into data centers, such attestation (like TPM for FPGAs) may become important to ensure a server's FPGA hasn't been reconfigured to a malicious design. There are already research proposals for attestation of FPGA bitstreams using challenge-response protocols. In summary, FPGA bitstream security is actively evolving. Vendors are learning from past attacks (adding side-channel protections, separating keys, etc.) and preparing for future threats (like quantum attacks). On the horizon, we can expect FPGAs that more seamlessly blend hardware security modules, that use new cryptographic primitives suited for the post-quantum era, and that leverage each chip's unique physical identity to lock bitstreams to devices. These trends all aim to stay one step ahead of attackers in the continual cat-and-mouse game of hardware security.

4. Best Practices for FPGA Developers and Integrators

Achieving robust bitstream security is not just about the FPGA's built-in features – it also requires developers and system integrators to use those features correctly and follow security-conscious processes throughout the FPGA's lifecycle. Here are recommended best practices to maintain strong bitstream security from development through deployment and decommissioning:

4.1. Always Enable Encryption and Authentication

It may sound obvious, but the first rule is to turn on the security features provided by the FPGA. Far too many designs leave bitstreams unencrypted for convenience or lack of awareness. Enabling AES-256 encryption and bitstream authentication (HMAC or signatures) is critical to protect IP from cloning and to prevent tampering. Modern toolchains (Vivado, Quartus, etc.) have options to generate encrypted bitstreams and associated authentication data – use them, and test that the FPGA indeed refuses unencrypted/unauthenticated configurations. Encryption without authentication is risky (as shown by attacks), and authentication without encryption still leaves IP exposed – so implement both if possible.

4.2. Secure Key Handling and Storage

The secrecy of cryptographic keys must be always maintained. Use the most secure key storage available on your FPGA – e.g., program the key into eFUSE rather than only battery RAM when possible. eFUSE keys cannot be read out and do not depend on battery life. If using battery-backed keys (perhaps to allow field update of keys), ensure the battery is mounted in a tamper-resistant way and monitor its voltage (losing the key due to battery failure can brick a device, so many choose eFUSE for permanency). Never hardcode keys in FPGA logic or software; only use the vendor-provided secure key provisioning methods. Use unique keys per product or per device if your volumes and infrastructure allow – this way, one device's compromise doesn't affect all. Manage key programming in a secure environment: e.g., at manufacturing, use an HSM (Hardware Security Module) or encrypted key files so that the key is not exposed to human operators. If keys are stored in code or scripts, treat those with high confidentiality and purge any plaintext keys after programming. Also, consider using Physical Unclonable Functions if supported – they can relieve you from storing a static key at all by generating device-specific keys internally.

4.3. Implement a Chain of Trust (Secure Boot)

If your FPGA is part of a larger system (especially if it's an FPGA SoC with processors), design a secure boot flow. Even in pure FPGA systems, you might have a small MCU that feeds the bitstream – ensure that MCU checks a digital signature on the bitstream (perhaps using a secure element for

 doi:10.20944/preprints202505.0607.v1

the verification key). The idea is to create a root-of-trust: a small piece of immutable code (in ROM or eFUSE) that validates everything else. This prevents an attacker from bypassing security by simply injecting their own boot code.

4.4. Lock Down Configuration and Debug Ports

In the field, there is rarely a need for the JTAG port or other configuration interfaces to remain open for a production device. Disable or password-protect the JTAG interface once the system is deployed. If you must keep JTAG for debugging, use access control – some devices support a JTAG password or challenge-response mechanism. Similarly, disable any fallback to an unsecured boot. Many FPGAs have a mode pin or fuse to indicate "only boot from encrypted bitstream"; use this so the device will not accept a plaintext bitstream even if someone tries. If partial reconfiguration is used, treat partial bit streams with the same level of security (sign them and if possible encrypt them) and ensure the reconfiguration interface is not exposed to untrusted sources. In essence, reduce the attack surface by closing all doors except those absolutely needed.

4.5. Use Anti-Tamper Features if Available

For high-security applications, take advantage of any anti-tamper and monitoring features provided. This could mean enabling the on-chip tamper detectors that erase keys on events (if your FPGA supports it), or adding external sensors tied into the FPGA's security pins. Implement zeroization procedures – e.g., if an unexpected reset or tamper signal is received, configure the FPGA to clear its configuration memory or keys rapidly. Some FPGAs let you toggle a "KEY_CLEAR" pin; make sure it's wired and used as needed. Coatings and physical enclosures that deter probing can complement the FPGA's internal features. If side-channel attacks are a concern (for instance, your threat model includes an attacker with oscilloscopes and measurement gear), consider using power line conditioning, filtering, or concealing operations with dummy activity. Also, when you configure the FPGA, you might choose to do so in a less accessible location (for example, only allow reconfiguration in a secure bootloader, not arbitrarily during operation, to narrow the window for side-channel data collection).

4.6. Keep Bitstreams and Keys Safe in Transit and Storage

Beyond the FPGA itself, think of the whole lifecycle – when you send a bitstream to a contract manufacturer or update a device remotely, how are you protecting it? Always send encrypted bitstream files (and preferably already encrypted by the FPGA's own key, not in plaintext). If you need to send an update over the air or internet, wrap it in a secure protocol like TLS/HTTPS. Use signed update packages so that devices can verify the source. When storing bitstreams (e.g., on a server or PC), treat them as sensitive IP – limit access, and consider storing only the encrypted versions. Avoid emailing bitstreams or keys; use secure file exchange methods. In a production line, if a test requires an unencrypted bitstream (for debugging), ensure that step is done in-house and that the unencrypted file never leaves your lab. For devices that support it, you can also employ bitstream version locking or anti-rollback: include a version number in the bitstream and have the FPGA refuse to load an older version once updated, to prevent replay of insecure older firmware.

4.7. Perform Security Reviews and Testing

Integrators should include FPGA bitstream security in their overall security audits. This means doing things like threat modeling (identify how someone might attack your device's FPGA configuration) and possibly hiring experts to do penetration testing. Tools exist to attempt to reverse engineer bitstreams (e.g., for academic use); you might try them on a sacrificial design to see how easy it would be if encryption weren't enabled, reinforcing that it must be.

 doi:10.20944/preprints202505.0607.v1

4.8. Plan for End-of-Life Data Security

Finally, consider what happens at the end of your product's life or if a device is discarded. If the FPGA was storing a secret key in battery-backed RAM, ensure that battery is removed, or the key is actively cleared. If devices are decommissioned, you may want to send a "suicide" bitstream that clears all configuration memory (for flash-based FPGAs) or simply physically destroy the hardware to prevent scavenging of parts (some companies literally crush or incinerate FPGAs that contained sensitive designs). This prevents attackers from obtaining old hardware and extracting bit streams or keys from them. Also, if you ever need to RMA or loan a board with an FPGA, make sure its security settings won't allow the recipient to snoop your design (some companies use secondary bitstreams to wipe or dummy-configure an FPGA before shipping it out for returns, etc.).

By following these best practices, developers can significantly raise the bar for attackers. FPGA bitstream security is strongest when it's not reliant on any one measure but a combination of defenses: strong cryptography, careful key management, hardware tamper-resistance, and secure operational processes. The FPGA's flexibility and reconfigurability no longer must be a security weakness; with proper use of available features and prudent practices, even a reconfigurable device can be a cornerstone of a secure system.\

5. Conclusions

Bitstream security in FPGAs is a multifaceted challenge, spanning technical defenses and prudent usage practices. We have explored how threats like reverse engineering, cloning, and tampering loom over FPGA-based designs, and how techniques such as encryption and authentication form the first lines of defense. Modern FPGAs from major vendors now offer a robust arsenal of security features, and upcoming innovations (lightweight crypto, post-quantum algorithms, PUFs) promise to further harden devices against evolving adversaries. Ultimately, the responsibility lies with FPGA designers and system integrators to actively employ these security measures and remain vigilant. By understanding the attack vectors and using a layered defense – secure boot architectures, proper key management, and diligent best practices – one can greatly reduce the risk of bitstream compromise. In an era where FPGAs are deployed from consumer IoT gadgets to mission-critical infrastructure, treating bitstream security as a first-class requirement is essential to protect intellectual property, ensure device integrity, and maintain trust in reconfigurable computing platforms.

References

1. Storey, T. (2015). *Counterfeiting & theft protection measures for FPGA designs*. Electropages. https://www.electropages.com/2015/06/fpga-design-counterfeiting-theft-protection
2. Ender, M., Giechaskiel, I., Rasmussen, K. B., & Paar, C. (2020). The unpatchable silicon: A full break of the bitstream encryption of Xilinx 7-series FPGAs. *USENIX Security Symposium*. https://www.usenix.org/conference/usenixsecurity20/presentation/ender
3. Rahman, F., Jain, R., & Saha, D. (2021). An end-to-end bitstream tamper attack on flip-chip FPGAs. *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. https://doi.org/10.1109/ICCAD51958.2021.9643564
4. Xilinx & Intel. (2022). *Product security documentation*. UG570 & Stratix 10 Secure Device Manager. Retrieved from https://www.intel.com/content/www/us/en/products/details/fpga/stratix/10/overview.html
5. Swierczynski, S. (2015). *Security analysis of the bitstream encryption scheme of Altera FPGAs* (Master's thesis, Ruhr University Bochum). https://www.ei.rub.de/media/tsch/.files/thesis_sws_final.pdf
6. Duncan, M., Fong, S., & Roelke, J. (2019). FPGA bitstream security: A day in the life. *IEEE International Test Conference (ITC)*. https://ieeexplore.ieee.org/document/8988462

 doi:10.20944/preprints202505.0607.v1

7. Caroline, H. (2021). *FPGA security: Practices and pitfalls*. Dev.to. https://dev.to/hedy/fpga-security-practices-and-pitfalls-42p
8. Majzoobi, M., Koushanfar, F., & Potkonjak, M. (2014). Light-weight secure PUFs for FPGA configuration. *IEEE Transactions on Information Forensics and Security, 9*(1), 161–174. https://doi.org/10.1109/TIFS.2013.2280103
9. Vemeko. (2023). *FPGA security features: Vendor comparison (Xilinx vs Intel)*. https://vemeko.com/fpga-security-features/
10. Analog Devices. (2019). *Bitstream authentication in FPGA-based systems*. Retrieved from https://www.analog.com/en/technical-articles/bitstream-authentication-in-fpga-based-systems.html
11. Parikh, R., & Parikh, K. (2025). Survey on Hardware Security: PUFs, Trojans, and Side-Channel Attacks. Survey on Hardware Security: PUFs, Trojans, and Side-Channel Attacks[v1] | Preprints.org
12. Parikh, R., & Parikh, K. (2025). A Scalable and Power-Aware Security Health Monitor for FPGAs Using Lightweight Sensors and ML-Based Inference. A Scalable and Power-Aware Security Health Monitor for FPGAs Using Lightweight Sensors and ML-Based Inference[v1] | Preprints.org