Title: Reproducible Round-Targeted Fault Effects on SHA-256 Hardware: Implementation Failure, Not Algorithmic Break

Abstract Hardware accelerators for cryptographic hash functions are prolific in systems requiring high throughput, from TLS/SSL servers to blockchain mining hardware. We present a detailed fault injection analysis of a commercial SHA-256 hardware accelerator. Using controlled clock and voltage perturbation, we demonstrate the ability to reproducibly inject faults that target specific rounds within the SHA-256 compression function. The primary effect is not a random bit-flip but a malformed digest that is strongly biased and, critically, independent of the input message nonce. This vulnerability, a failure of the physical implementation's fault tolerance, allows an attacker to bypass authentication checks or disrupt consensus protocols without needing to solve the underlying cryptographic problem. We analyze the fault propagation and propose necessary hardware-level countermeasures.

1. Introduction The SHA-2 (Secure Hash Algorithm 2) family of functions, particularly SHA-256, remains a cornerstone of modern digital security. It is integral to protocol integrity (TLS, IPsec), software signing, and distributed consensus mechanisms (e.g., Bitcoin's Proof-of-Work). To meet performance demands, these algorithms are often implemented in dedicated hardware accelerators (ASICs or FPGAs) rather than software.

While algorithmically secure against known pre-image and collision attacks, the physical implementation of a cryptographic primitive introduces a new attack surface. Fault Injection Attacks (FIAs) attempt to subvert a system by introducing a perturbation (e.g., voltage glitch, EM pulse, or laser) at a precise moment, causing the computation to produce an incorrect result.

Our Contribution: In this work, we demonstrate a practical and reproducible fault attack against a commercial-grade SHA-256 hardware accelerator. Our findings are:

- Round-Targeting: We can reliably time the fault injection to corrupt the internal state at a specific, desired round (e.g., R18) of the 64-round compression function. - Reproducible Bias: The resulting malformed digest is not random. It collapses to a predictable, biased value. - Nonce-Independence: This biased output is independent of the input message data (or nonce), meaning an attacker can generate a desired (incorrect) hash without controlling the input.

We stress that this is an implementation failure, not an algorithmic break. The mathematical properties of SHA-256 are not compromised. The vulnerability lies in the accelerator's lack of runtime integrity checking.

2. Background and Preliminaries 2.1 The SHA-256 Algorithm SHA-256 processes an input message in 512-bit (64-byte) blocks. The core of the algorithm is the compression function, which takes the 256-bit intermediate hash value $H(i–1)$ from the previous block and the 512-bit current message block $M(i)$ to produce the next intermediate hash value $H(i)$.

This compression function is iterative, consisting of 64 rounds (or steps). In each round j (from 0 to 63), the 256-bit internal state (represented by eight 32-bit working variables a, b, c, d, e, f, g, h) is updated using a message schedule word Wj and a round constant Kj. The logic involves non-linear functions ($\Sigma 0$, $\Sigma 1$, Ch, Maj) that provide the algorithm's security.

After 64 rounds, the initial state $H(i–1)$ is added to the final working variables to produce $H(i)$. Any unhandled computational error within one of these 64 rounds will propagate, leading to an incorrect final digest.

2.2 Fault Injection Threat Model We assume an attacker has temporary physical access to the device (a "gray-box" model). The attacker can depackage the chip (if necessary) and position probes to manipulate the device's clock signal or power supply. The goal is to introduce a perturbation that is

precise enough to corrupt a calculation but not so large as to reset the entire device.

3. Experimental Setup 3.1 Device Under Test (DUT) The DUT is a commercial hardware accelerator designed for high-throughput hashing operations, implemented on a sanitized semiconductor process. The device accepts a message block and a "start" signal, returning a "done" signal upon completion with the 256-bit digest. The internal architecture appears to be a standard, fully unrolled pipeline, where each of the 64 rounds is implemented in distinct logic, allowing for high clock speeds.

3.2 Perturbation Methodology Our test bench consists of a host PC, a custom DUT target board, and a high-resolution signal generator. We employed clock glitching as our fault injection vector.

The DUT is supplied with a nominal clock signal CLK_nom. At a precise time t_fault after asserting the "start" signal, the signal generator injects a single, shorter clock pulse (a "glitch"). The timing t_fault is swept with picosecond-level resolution to map the fault effect to the physical computation stage. The host PC provides a test vector (a fixed message block M and a variable nonce N), triggers the DUT, and records the (potentially malformed) digest.

4. Fault Analysis and Results By sweeping the t_fault parameter, we observed a direct correlation between the injection timing and the resulting digest. A specific "sweet spot" in voltage and timing produced a stable, incorrect digest while the device reported "done". The pipelined nature of the DUT made it highly susceptible to round-targeting, and the bias was reproducible across runs and inputs.

5. Impact and Implications The fault enables authentication bypass, blockchain consensus manipulation, and silent integrity failures in critical systems. It does not compromise SHA-256 mathematically, only physically.

6. Countermeasures We recommend hardware lockstep redundancy, temporal redundancy, runtime integrity checks, and physical sensors for perturbation detection.

7. Conclusion Reproducible, round-targeted fault effects in SHA-256 hardware are an implementation failure, not an algorithmic weakness. Security depends equally on the algorithm and its embodiment in silicon.

References [1] Bar-El et al. (2006). The Sorcerer's Apprentice Guide to Fault Attacks. IEEE Proc. 94(2), 370–382. [2] Blömer & Seifert (2003). Fault Based Cryptanalysis of AES. Financial Cryptography. [3] NIST FIPS PUB 180-4 (2015). Secure Hash Standard (SHS).