

# Final Report

December 2025

Ian Barnaby  
Ryn Stewart  
Tome Dudanov

## I. PROBLEM STATEMENT

In-memory computing to accelerate AI requires neural networks weights to be encoded in non-volatile memory (NVM). Current heterogeneous systems require weights to be communicated from CPU to accelerator, potentially in an untrusted environment. At present, there is no way to encrypt these weights in NVM to allow for correct neural network function. Securing this operation therefore (currently) requires encrypting/decrypting weights as communicated from CPU to accelerator to prevent snooping/monitoring attacks; however, as many of these applications are low-power, this proves difficult with a constrained area/power budget. This project seeks to describe a method of weight encryption in these systems leveraging existing device properties to reduce security overhead.

## II. MOTIVATION

Non-volatile memory (NVM) has emerged as a new memory paradigm in recent years, offering non-dynamic system memory that does not require refreshing. This type of memory has also been used for in-memory computing (IMC), as the two-terminal nature of certain NVM implementations (such as ReRAM [3]) allow for direct operation within the memory cells. This has become especially relevant for ML applications, in which neural networks may be directly mapped to a crossbar array of NVM cells [?]. Security in NVM cells is already a challenge due to data longevity, but has been researched using Bonsai-Merkle Trees (BMT) to secure the memory [?]. While effective in NVM for system memory, this is not an option for IMC, as the crossbar array must hold unencrypted values for correct neural network operation.

In the SLEAC project [1], the CPU/accelerator system consists of a CPU, IMC accelerator, and a data bus connecting the two. This system is prone to snooping attacks on the bus as weight values, along with their location within the array and instruction, are communicated from CPU to accelerator. Therefore, it is desired to determine a suitable encryption/decryption method for this unique low-power application such that instructions/data may be sent securely across the bus.

### A. Threat Model

## III. EXPERIMENT

### A. Bus Encryption

We implemented three cryptographic approaches directly into the CPU-accelerator bus layer to secure any transfers

such as MNIST weights. Each scheme has advantages and disadvantages, with clear differences in hardware.

The first encryption approach that we took was the 128-bit AES-GCM authenticated encryption mode with a PUF-derived secret key. We encrypt each chunk of the message sent, producing ciphertext and an authentication tag. The bus carries this encrypted message, including the GCM tag and nonce, while during decryption, each message is verified for integrity before it is loaded. This path assumes a safely pre-established symmetric key, derived from the PUF. However, AES alone cannot establish a shared key - it has to assume that the PUF key is already synchronized between the CPU and the accelerator, even though a PUF can only reside on one device, meaning the CPU and the accelerator cannot both derive the same secret without additional bus exchange. This highlights the need for additional work to ensure secrecy, as delivering plaintext shared keys on the bus exposes them to snooping.

One way to address this issue is to use the RSA cryptosystem directly for bus encryption, eliminating the need for sharing secrets over the bus. In our implementation, we split the plaintext into OAEP-sized (214-byte) blocks, and encrypt each block with RSA-2048. Additionally, we encrypt a SHA256 digest of the plaintext and append it to the message for integrity checking. The bus, hence, carries only RSA ciphertext blocks and metadata, which upon decryption is recovered and reassembled. Then, the appended hash is compared to a recomputed digest to reveal any tampering en route. Unlike AES, this approach does not require any pre-shared secrets, since RSA enables the CPU and accelerator to encrypt and decrypt directly with their respective private and public keys. However, the reliance on repeated expensive exponentiation adds significant computational overhead, only acceptable with sparse bus traffic.

To relieve the tension between secrecy and efficiency, our third and final cryptographic approach combines the best of both worlds: using a computationally expensive operation to establish a shared key between the parties securely, followed by fast encryption with a safely pre-established symmetric key. Specifically, we use elliptic-curve Diffie-Helman (ECDH) on the P-256 curve to derive a shared secret across the bus without the risks of snooping. Encryption and decryption afterwards happen symmetrically using AES-GCM, as discussed earlier in this section. This hybrid scheme therefore achieves a practical balance: secrecy is kept during key establishment, and efficiency is maintained during more frequent bus traffic.

These cryptographic schemes differ in hardware complexity as well as security implications. The hybrid ECDH + AES approach adds elliptic curve logic on top of the AES-GCM component, so the tradeoff is more hardware complexity than the other two schemes. Yet, no scheme is perfect. Symmetric encryption alone struggles with key exchange, pure asymmetric lacks in performance, and hybrid has additional implementation costs. Each of these designs has its trade-off, balancing secrecy, speed and hardware complexity differently.

### B. Weight Integrity

In the case where pure RSA encryption is used, data integrity check for weights is added by appending a SHA256 hash (computed over the plaintext weights) ciphertext in its encrypted form, extending the encrypted block size by one. On the receiving side, the ciphertext is decrypted and the hash is stripped from the message. The SHA256 hash is then recomputed over the decrypted weights and compared with the received hash value. If the values are equal the weight update is successful, otherwise the update is ignored. This process design is shown in Figure 1

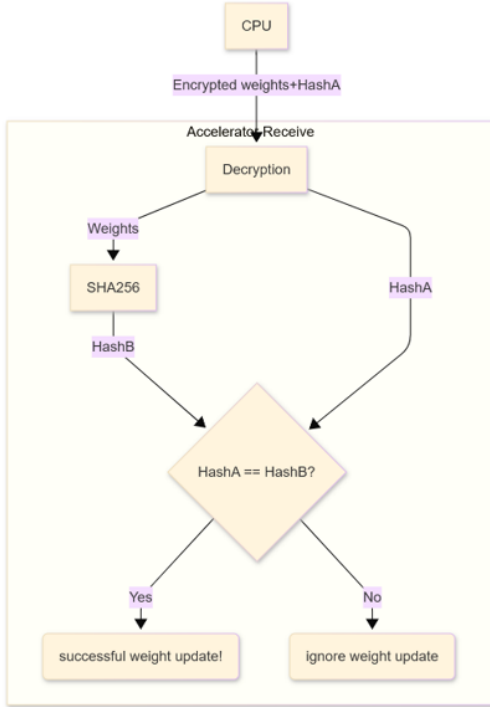


Fig. 1. Overview of decryption and integrity checking upon a weight update.

## IV. EVALUATION

### A. Encryption Performance Comparison

To evaluate the practicality of each scheme described in Subsection III-A, we measured encryption and decryption timing under identical bus traffic conditions. The results highlight clear distinctions: AES-GCM has the lowest latency, RSA-2048 suffers due to expensive operations, and the hybrid ECDH+AES-GCM approach occupies the middle ground -

essentially incurring an initial cost for key establishment, then evening out close to AES with further transmissions. Figure 1 illustrates the measured latencies for all three schemes under identical bus traffic.

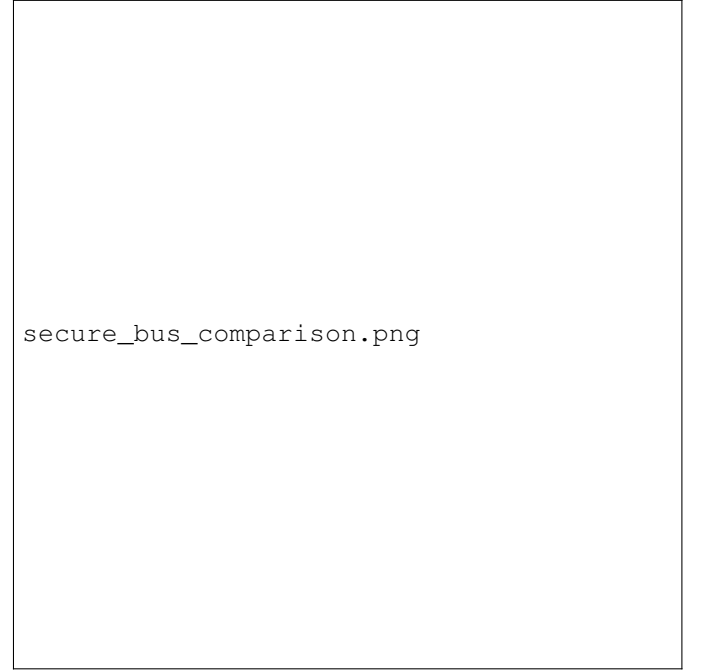


Fig. 2. Encryption method time comparison of the insecure bus, hashing only for integrity but no bus encryption, 128-bit AES-GCM encryption, ECDH key exchange with 128-bit AES-GCM encryption, and RSA-2048 encryption.

As shown in Figure 2, the insecure bus baseline transmits the message in 0.130s, while hashing only for integrity without any encryption increases the time to 0.457s. AES-GCM gives us efficient protection, finishing in 0.802s, the hybrid ECDH+AES-GCM scheme incurs a higher cost of 1.510s, and RSA is by far the slowest with 5.819 seconds to fully encrypt, send the message over the bus, and decrypt it on the other side.

### B. Weight Integrity

For an integrated functional simulation in python, the SHA256 class is used from the pycryptodome library. The integrity check was verified with a simple bit flipping function that XORs the ciphertext with a 64-bit mask where 1 bit is randomly set using the numpy random module.

To obtain a power analysis of the SHA256 module, RTL from an open source github project [8] was synthesized with Vivado for an Artix 7 FPGA development board. This Verilog implementation of SHA256 has a 512-bit block input with a 256-bit digest and executes 64 rounds of hashing, one round per cycle. Clockspeed was set to 50MHz. Using this test setup, the reported power of the RTL is detailed in Table I, notably the total is only 0.557W.

Based on this RTL project, the throughput can also be calculated by taking the number of cycles needed to complete hashing on a 512-bit block. Each block goes through 64 rounds of SHA256, and one round is executed per cycle, so the total

Dynamic Power	0.472W
Static Power	0.085W
Total Power	0.557W

TABLE I  
SHA256 POWER WITH 50MHZ CLOCK FOR FPGA DEV BOARD

time to execute is found by the following (assuming 50MHz for clockspeed):  $64 \text{ cycles} \times 20 \text{ ns} = 1.28 \mu\text{s}$ . Finally, one block contains 448 bits of data (512 - 64 bits for padding) leading to a throughput of 350Mbps.

Although these metrics will be highly dependent on the specific technology the accelerator is implemented on, they demonstrate the feasibility of adding specialized hardware to implement secure hashing in terms of power requirements and throughput. Given the need for decryption in the threat model of this system, especially using an asymmetric algorithm, SHA256 is not a bottleneck and the power is reasonable given the overall application. However, future work could further explore lower overhead data integrity checks, such as cyclic redundancy check or parity bits, using encryption to keep the verification data secret to an attacker.

## V. RELATED WORKS

### A. Cycle Count Studies

Cycle counts for encryption vary not only depending on the encryption algorithm itself, but also on the machine itself, primarily on the CPU. A cycle count, therefore, is not measured by the algorithm itself, but benchmarked on different hardware to capture performance for that specific hardware. Prior work has shown that AES can be implemented with low cycle counts on Cortex M3 and M4 microcontrollers [7], while ECC and RSA are more computationally heavy [5]. In our bus setup of 256-bytes at a time, AES-128 requires a one-time key expansion encryption and decryption, as well as 16 16-byte blocks encrypted and sent. This would take a total of 1469.4 cycles for key expansion, and 20960 cycles for encryption and decryption [7], a total of 22429.4 for a single encryption and decryption operation on the ARM Cortex M4. This figure excludes the additional cost of hashing verification. On the other hand, for RSA-2048 to encrypt and decrypt a message, on the same microcontroller, it would take 228068226 cycles to sign, and 6195481 cycles to verify [5], a total of 234263707 cycles excluding a one-time keygen cost and hashing verification. ECC-256 balances both schemes out, incurring a one-time cost of 12713277 cycles for key gen, 13102239 cycles to sign, and 24702099 cycles to verify - a total of 50517615 [5], then every message from then on will be encrypted/decrypted using AES, and therefore has its cost. Note that the ECC-256 asymmetric key exchange portion is still costly, but much less costly than RSA's operation.

### B. Integrity checking for Model Weights

Data integrity for DNN weights has become an increasingly relevant topic as the adoption of models has moved to edge compute, requiring quantization of weights. Previous works [9], [6] have shown the effectiveness of bit-flip attacks (BFAs)

on model inference, either to subtly change predictions or to completely destroy accuracy to the level of a random guess. In response to this, there have been several proposed mitigations that add data integrity checks to weights during the online operation of a model.

For example, HashTAG [4] attempts to resolve the issue by generated Pearson hashes prior to model deployment and storing them as "ground-truth signatures" per DNN layer. The hashes are stored in secure SRAM and compared against hashes recomputed during operation. This work's use of Pearson hashing allows for a low overhead (8-bit hash computed with a 256 value look-up table) data integrity check, but still leaves the model susceptible to a BFA due to collisions. Additionally, this method would not work for the system described in this paper as the expected hashes have to be precomputed and stored in secure memory.

Another low overhead mitigation is LIMA [2]. Here, the authors propose encoding the data integrity check into the weights themselves by modifying a K (defined per system) number of LSBs such that the sum of K MSBs and LSBs equals 0 per DNN layer. The key insight from this work is that the LSBs of the quantized weights minimally impact the accuracy of the model. LIMA does not increase memory overhead and only requires multiply and accumulate operations to verify integrity. However, the mitigation is still leaves a model vulnerable as an attacker could flip bits in the weights such that the LSBs and MSBs still sum to 0. Another note is that LIMA caught single bit-flips only 70.5% of the time in the worst case.

## REFERENCES

- [1] S. Davis, "SWAP Hub to Transform Satellite Imaging Performance Through Microelectronics-Enabled Artificial Intelligence - Semiconductor Digest — semiconductor-digest.com," <https://www.semiconductor-digest.com/swap-hub-to-transform-satellite-imaging-performance-through-microelectronics-enabled-artificial-intelligence/>, [Accessed 12-09-2025].
- [2] F. S. Hosseini, Q. Liu, F. Meng, C. Yang, and W. Wen, "Safeguarding the intelligence of neural networks with built-in light-weight integrity marks (lima)," in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2021, pp. 1–12.
- [3] H.-H. Hsu, T.-H. Wen, W.-S. Khwa, W.-H. Huang, Z.-E. Ke, Y.-H. Chin, H.-J. Wen, Y.-C. Chang, W.-T. Hsu, A. Lele, B. Zhang, P.-S. Wu, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, S.-H. Teng, C.-C. Chou, Y.-D. Chih, T.-Y. Chang, and M.-F. Chang, "A 22 nm floating-point rram compute-in-memory macro using residue-shared adc for ai edge device," *IEEE Journal of Solid-State Circuits*, vol. 60, no. 1, pp. 171 – 83, 2025/01. [Online]. Available: <http://dx.doi.org/10.1109/JSSC.2024.3470211>
- [4] M. Javaheripi and F. Koushanfar, "Hashtag: Hash signatures for online detection of fault-injection attacks on deep neural networks," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.
- [5] T. Oder, T. Pöppelmann, and T. Güneysu, "Beyond ecda and rsa: Lattice-based digital signatures on constrained devices," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, June 2014, pp. 1–6. [Online]. Available: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6881437>
- [6] A. S. Rakin, Z. He, J. Li, F. Yao, C. Chakrabarti, and D. Fan, "T-bfa: Targeted bit-flip adversarial weight attack," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 11, pp. 7928–7939, 2022.
- [7] P. Schwabe and K. Stoffelen, "All the aes you need on cortex-m3 and m4," in *2016 International Workshop on Selected Areas in Cryptography (SAC)*, St. John's, NL, Canada, 2016, pp. 180–201. [Online]. Available: <https://ko.stoffelen.nl/papers/sac2016-aesarm.pdf>

- [8] J. Strömbergson. *Hardware implementation of the SHA-256 cryptographic hash function*. Secworks. [Accessed: 11 November 2025]. [Online]. Available: <https://github.com/secworks/sha256>
- [9] F. Yao, A. S. Rakin, and D. Fan, “DeepHammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips,” in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1463–1480. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity20/presentation/yao>