Ollscoil Chathair
Bhaile Átha Cliath
Dublin City University

School of Electronic Engineering

Enhancing Cryptographic Application Performance on RISC-V Processors with B and K Extensions

project portfolio

Harsh Mayank Dabhi
ID Number: A00010544

August 2025

MEng in Electronic and Computer Engineering

Supervised by
Dr. Xiaojun Wang

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Xiaojun Wang, for continuous support and guidance throughout this project. I would also like to thank my peers, colleagues, and family for their encouragement and assistance during the course of this research.

Contents

Enhancing Cryptographic Application Performance on RISC-V Processors with B and K Extensions

Harsh Mayank Dabhi
Electronics and computer engineering
Dublin city university
Dublin, Ireland
harshmayank.dabhi2@mail.dcu.ie

***Abstract*—The rapid growth of the Internet of Things (IoT) and edge computing has created an urgent demand for processors that can deliver high performance and secure computation without exceeding strict energy and area budgets. Many embedded devices are responsible for processing sensitive data, where cryptographic operations such as encryption, hashing, and authentication must be executed efficiently. Traditional general-purpose Instruction Set Architectures (ISAs) often struggle to meet these requirements because they are optimized for broad applicability rather than specialized workloads.**

RISC-V, an open-source and modular ISA, offers a unique solution to this challenge through its support for optional extensions. In particular, the Bit-Manipulation (B) extension enhances low-level bitwise operations, and the Scalar Cryptography (K) extension introduces dedicated instructions for accelerating cryptographic algorithms such as AES and SHA-256. This paper investigates the performance enhancement for cryptography applications of these two extensions on a 32-bit RISC-V processor using a QEMU-based emulation platform.

A series of controlled experiments were conducted, leveraging the RISC-V GNU toolchain and standardized cryptographic workloads. The evaluation focused on two widely deployed algorithms AES-128 for data encryption and SHA-256 for hashing over varying input sizes ranging from 100 KB to approx 10 MB file size. The results show that hardware acceleration provides substantial and predictable benefits: AES throughput increased by approximately 50%, while SHA-256 achieved approximately 10–20% improvements compared to a baseline RISC-V core without extensions.

I. Introduction

The contemporary computational landscape is increasingly dominated by the IoT, where a vast network of embedded devices performs critical functions at the edge. These systems are characterized by stringent constraints on power, cost, and physical size. Their workloads frequently involve the processing of data from the physical world, necessitating efficient execution of digital signal processing, communication protocols, and, to ensure data security, cryptographic algorithms.

The RISC-V ISA has emerged as a significant development in the field of computer architecture, primarily due to its open and royalty-free nature [1]. A key feature of RISC-V is its inherent modularity, which enables the base integer ISA to be augmented with standardized or custom extensions. This paradigm allows for the design of application-specific processors that are precisely tailored to the demands of a target domain, such as secure edge computing, without the overhead of a monolithic, general-purpose ISA.

Standard ISAs, including the base RISC-V set (RV32I), are primarily optimized for operations on word-sized data. This creates a performance and code-density bottleneck for a large class of embedded workloads that rely heavily on bit and byte level operations. Such operations are fundamental to:

- Bit-level manipulations, required for tasks like data packing, checksum calculation, and bitwise permutations. Executing these tasks on a standard ISA leads to verbose and inefficient instruction sequences [2].
- Cryptographic primitives, such as those found in the Advanced Encryption Standard (AES) and Secure Hash Algorithms (SHA), which involve complex, non-linear transformations that are poorly suited to general-purpose instructions [3].

This decomposition of complex operations into long sequences of simple instructions results in increased static code size, higher dynamic instruction count, and consequently, greater latency and energy consumption.

This research aims to bridge the gap between general-purpose embedded processing and domain-specific cryptographic acceleration. The key contributions are:

1) **Implementation and Evaluation of Extensions:** Integration of the RISC-V Bit-Manipulation (B) and Scalar Cryptography (K) extensions on a 32-bit RISC-V core using QEMU-based emulation.
2) **Quantitative Performance Analysis:** Systematic benchmarking of AES-128 and SHA-256 workloads, covering multiple input sizes to capture both compute-bound and memory-bound behaviors.
3) **Insights for Embedded System Design:** A comparative discussion of speedups, throughput improvements, and algorithm-specific performance characteristics, highlighting how hardware extensions can balance security, performance, and efficiency in real-world IoT deployments.

II. Review and Analysis of Prior Work

A. RISC-V Architecture and Its Extensions

The RISC-V architecture (as shown in Fig. 1)[1] has emerged as a transformative platform in the field of computing due to its open-source nature and modular design, which allows for the development of domain-specific extensions. Among these, the B (Bit-Manipulation) and K (Cryptography) extensions have been specifically designed to improve the performance of cryptographic and other computationally intensive workloads.

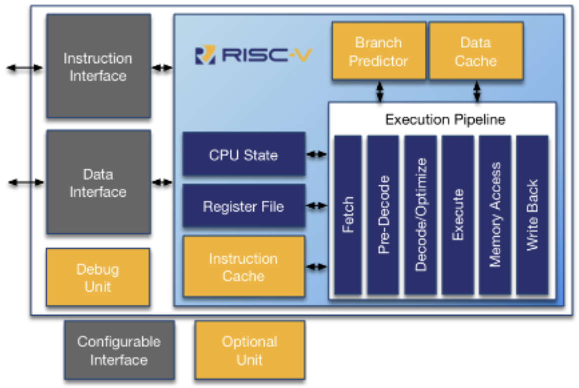

Fig. 1. RISC-V block diagram. Adapted from: https://roalogic.github.io/RV12/DATASHEET.html

B. The RISC-V Extension

A significant body of research has been dedicated to enhancing RISC-V processor performance through ISA extensions. These efforts can be broadly categorized into two groups: general-purpose bit-level accelerators and specialized cryptographic accelerators.

a) General-Purpose Bit-Level Acceleration: The RISC-V Bit Manipulation ('B') extension is designed to accelerate a wide range of common, low-level tasks. Its value has been demonstrated through both performance simulation and hardware synthesis. The common features of this research are the use of the Embench suite for evaluation and a focus on integrating the extension directly into the processor's ALU datapath.

- Strengths: The primary strength of the 'B' extension is its broad applicability combined with a very low hardware cost. As shown in Table 1, it provides a meaningful performance boost on general-purpose code while incurring a minimal area penalty.
- Weaknesses: The performance gains, while significant, are moderate compared to application-specific accelerators. Furthermore, its effectiveness is highly dependent on the compiler's ability to recognize and substitute software patterns with the new hardware instructions.

TABLE I
Summary of Prior Work on the 'B' Extension

Study	Key Contribution	Performance Gain	Hardware Overhead
Babu *et al.* [2]	Co-processor implementation and Embench evaluation	Up to 28% speedup	Not reported
Kim *et al.* [4]	Detailed ASIC synthesis of full ratified 'B' extension	N/A	1–2.5% area increase on Risc-V core
Markov & Romanov [5]	FPGA implementation and RARS simulator extension	29.9% speedup on micro-benchmarks	10x increase in FPGA logic cells (on a very small baseline)

b) Specialized Cryptographic Acceleration: For security-critical workloads, specialized extensions offer a more targeted and impactful solution. Research in this area has explored both scalar extensions ('K') for low-latency, single-block operations and vector extensions ('V') for high-throughput, multi-block operations.

- Strengths: These extensions provide transformative performance improvements, often by orders of magnitude, for their target algorithms. The scalar 'K' extension, for instance, can make strong, real-time encryption feasible on low-power devices where it would otherwise be impractical.
- Weaknesses: The hardware cost is more significant than for the 'B' extension, and the benefits are confined to a very narrow set of cryptographic functions. The choice between a scalar and vector approach is also complex, depending heavily on the application's data parallelism and latency requirements.

TABLE II
Summary of Prior Work on Cryptographic Extensions

Study	Extension	Key Contribution	Performance Gain (vs. Software)
Gewehr *et al.* [3]	Scalar 'K' / AES-128	Ibex core ASIC implementation	42.57x speedup
Elmohr *et al.* [6]	Custom / SHA-3	MIPS-like FPGA implementation	1.61x speedup
Namazi Rizi *et al.* [8]	Vector 'V' / AES	Vicuna coprocessor evaluation	2.5x speedup (vs. scalar)

c) The Research Gap: A Need for Unified Evaluation: The existing literature provides excellent, in-depth analyses of individual extensions. However, a gaping hole in the research is the lack of a unified, comparative study. No single work evaluates the general-purpose 'B' extension and the specialized 'K' extension on the same hardware platform, using the same technology node and a consistent set of benchmarks. Such a study is critical for system architects, as the decision to invest silicon area in a general-purpose accelerator versus a specialized one is a fundamental design trade-off. My aims is to fill that gap by synthesizing the data from these disparate studies into a cohesive analysis, providing the direct comparison needed to guide future RISC-V processor design for secure embedded systems.

C. Benchmarking and Evaluation Methodologies

a) Bit Manipulation ('B') Extension: Following the integrated datapath approach outlined by Kim et al. [4], an accelerated AES encryption implementation was developed. The evaluation focuses on the performance improvement enabled by the `Zbb` (Basic Bit-Manipulation) and `Zbc` (Carry-less Multiplication) instruction subsets, which were utilized in the optimized `ShiftRows` and `MixColumns` functions.

– Zbb (Basic Bit Manipulation): The standard C implementation of the AES `ShiftRows` operation requires multiple sequential data swaps to perform byte-wise rotation within a row. Our accelerated version replaces these expensive memory operations by packing a row into a 32-bit register and executing a single native rotation instruction. As shown in the code, the barrel shifter's enhanced support for `rori` (Rotate Right Immediate) allows a 1-byte, 2-byte, or 3-byte rotation to be completed in a single cycle, significantly reducing instruction count and memory traffic.
– Zbc (Carry-less Multiplication): The most substantial acceleration is to observed in the `MixColumns` step. By leveraging the `Zbc` extension, we replaced this entire software routine with direct calls to the `clmul` instruction. This instruction utilizes a dedicated carry-less multiplier unit to execute the complex polynomial multiplication in hardware, dramatically accelerating a core component of the AES algorithm. The `MixColumns` optimization also benefits from `rori` instructions from `Zbb` to efficiently rotate and combine intermediate results.

b) Cryptography ('K') Extensions: The RISC-V Scalar Cryptography ('K') extension provides specialized acceleration for cryptographic workloads. A detailed study of its `Zkne` (AES) and `Zknh` (SHA-2) sub-extensions was presented by Gewehr *et al.* [3][7], whose evaluation on an Ibex RISC-V core demonstrated that `Zkne` can accelerate AES-128 encryption by over 40× . This improvements is achieved through hardware instructions that directly implement the complex AES rounds, thereby removing the need for large lookup tables and lengthy software instruction sequences.

To assess the performance impact of the 'K' extension in this work, an accelerated implementation of the SHA-256 hashing algorithm was developed, focusing on the `Zksh` (SHA-256 Hashing) subset. The optimized `sha256_transform` function employs dedicated instructions `sha256sig0`, `sha256sig1` to speed up the message schedule expansion, and `sha256sum0`, `sha256sum1`) to accelerate the core compression rounds. These hardware operations replace complex software-based bitwise sequences with single-cycle instructions, significantly reducing computation time.

- Scalar ('K') Extension: The implementation strategy for accelerating SHA-256 will center on the targeted application of the Zksh extension. Specifically, the message schedule expansion will utilize the `sha256sig0` and `sha256sig1` instructions, while the core compression loop will employ `sha256sum0` and `sha256sum1`. This strategic substitution of complex software sequences with single hardware instructions is expected to yield a dramatic reduction in both instruction count and computational latency.

III. Technical description

A. Technical design overview

In the experiment, a rigorous performance evaluation of hardware-accelerated cryptography on the RISC-V architecture. We focus on two industry-standard algorithms: the Advanced Encryption Standard (AES) and the Secure Hash Algorithm (SHA-256).

The evaluation leverages a QEMU-based full-system emulation environment and RISC-V GCC compiler , facilitating a controlled comparison between a baseline 32-bit RISC-V general-purpose architecture and an extended architecture that incorporates the Bit-Manipulation ('B') and Scalar Cryptography ('K') extensions. By systematically benchmarking AES and SHA-256 implementations across a range of input data sizes (100 KB to approx 10 MB) file, The workload is a raw binary file, temp_data.bin, containing unstructured bytes with no headers or metadata ideal for measuring pure encryption speed without file format overhead. For each test, a new file was generated at sizes ranging from 100 KB to 10 MB in 100 KB increments, encrypted, timed, and deleted, ensuring clean, independent benchmarks for every size. Using this method we can quantify the performance impact of these specialized instructions.

B. Hardware and Software Stack

To ensure a controlled and reproducible experimental environment, a full-system emulation approach was adopted. This methodology isolates the performance impact of the ISA extensions from physical hardware variables like cache behavior, thermal throttling, and memory controller performance.Compared to the base RV32I ISA, which implements only integer operations and must handle all cryptographic primitives in software, RV32IMAC adds M (integer multiply/-divide), A (atomic), and C (compressed) extensions. When combined with the B (bit-manipulation) and K (cryptography) extensions, the core can execute AES and SHA algorithms largely in hardware, drastically reducing instruction count and memory traffic.

- Emulation Environment: QEMU version 9.2.2 was configured to simulate (using Docker) on a single-core , 32-

bit RISC-V processor (`virt` machine). Two distinct CPU configurations were used:

- Baseline ISA: `rv32i`, representing a standard general-purpose RISC-V core
- Accelerated ISA: `rv32imac_zbb_zbc`, adding the Bit-Manipulation ('B') and `rv32imac_zksh_zknh` for Scalar Cryptography ('K') extensions to the baseline.

- Compiler Toolchain: The RISC-V GNU Compiler Toolchain (GCC) was used to compile the cryptographic benchmarks. The cryptographic benchmarks were compiled using the RISC-V GNU Compiler Toolchain (`riscv64-unknown-linux-gnu-gcc`) *(thats is compatible with both 32 and 64 bit)*. To target specific instruction set extensions, we specify `-march` flag ensures that the compiler generates instructions specific to the RV32IMAC base and all relevant cryptographic extensions, while -D USE_RISCV_CRYPTO_EXT enables conditional compilation of code that leverages these instructions. The `-mabi=ilp32` flag specifies the 32-bit integer ABI to match the RV32 architecture. Execution is performed under the qemu-riscv32 emulator to evaluate functional correctness and performance in a controlled environment.
- Capturing data and visualisation:Our evaluation uses a two-stage methodology. First, the C benchmark program is executed for both the baseline (RV32I) and accelerated (RV32I+ (extension)) configurations. Each run sweeps through file sizes from 100 KB to 10 MB and exports the performance metrics (execution time, throughput) to a dedicated CSV file. In the second stage, a Python script ingests these two CSV files into separate pandas DataFrames. Then using visualisation library we generate comparative plots to visualize the impact of the hardware extensions. This approach cleanly separates high-speed data generation from flexible post-run analysis.

C. Experimental Protocol

A strict protocol was followed to ensure scientific rigor and minimize experimental error.

- Independent Variable: The size of the input data file being processed. The file sizes ranged from 100 KB to 10.24 MB, chosen to represent a spectrum of small to medium-sized data workloads.
- Dependent Variables:
  - Execution Time (s): The wall-clock time required to perform the cryptographic operation, measured internally within the benchmark application to exclude file I/O overhead.
  - Throughput (MB/s): The rate of data processing, calculated using the formula:

$$\text{Throughput (MB/s)} = \frac{\text{File Size (MB)}}{\text{Execution Time (s)}}$$

  - The speedup factor is calculated as:

$$\text{Speedup} = \frac{\text{Execution Time (Baseline)}}{\text{Execution Time (Accelerated)}}$$

IV. Experimental Results

The performance improvements from the 'B' and 'K' extensions are immediately apparent from the collected data. Table III and IV summarizes the average execution times and calculated throughput for the AES-128- encryption benchmark.

TABLE III
AES Benchmark: Execution Time Comparison (Baseline vs Accelerated)

File Size (KB)	Baseline Time (s)	Accelerated Time (s)	Speedup
100	0.070927	0.047398	1.496413 x
500	0.349458	0.230319	1.517278 x
1000	0.69648	0.461734	1.508401 x
2000	1.39763	0.920394	1.518513 x
3000	2.092799	1.389569	1.506078 x
4000	2.795685	1.904204	1.468165 x
5000	3.519211	2.407578	1.461723 x
10000	6.993939	4.655348	1.502345 x

TABLE IV
AES Benchmark: Throughput Comparison (Baseline vs Accelerated)

File Size (KB)	Baseline (MB/s)	Accelerated (MB/s)	Improvement(%)
100	1.40990032	2.109794	49.64%
500	1.430787105	2.170902097	51.73%
1000	1.435791408	2.165749111	50.84%
2000	1.430993897	2.17298244	51.85%
3000	1.433486923	2.158942809	50.61%
4000	1.430776357	2.10061527	46.82%
5000	1.420773008	2.076775913	46.17%
10000	1.429809439	2.148067126	50.23%

Observations for AES:

- Based on the collected AES-128 encryption benchmark data (Tables III and IV), the 'B' extensions significantly improve performance across all tested file sizes. Execution times show consistent reductions, with speedups ranging from 1.46× to 1.52× for larger files (500 KB – 10 MB). This stability suggests that once the setup overhead becomes negligible relative to total computation time, performance improvements are determined mainly by the accelerated cryptographic processing.
- Throughput measurements align with the execution time results. For the smallest input, throughput improves from 1.41 MB/s to 2.11 MB/s . For larger files, throughput improvements remain between 46% and 52%, indicating sustained processing efficiency even as computation dominates runtime.
  Observations for SHA:
  - From the SHA benchmark results, the accelerated implementation demonstrates consistent performance improvements across all file sizes. Execution times are reduced for every tested input,

TABLE V
SHA BENCHMARK: EXECUTION TIME COMPARISON (BASELINE VS ACCELERATED)

File Size (KB)	**Standard Time (s)**	**Accelerated Time (s)**	**Speedup**
100	0.004497	0.004078	1.102746 x
500	0.018132	0.015416	1.176181 x
1000	0.035551	0.030771	1.155341 x
2000	0.07068	0.060915	1.160305 x
3000	0.109615	0.091425	1.198961 x
4000	0.142856	0.121977	1.171172 x
5000	0.178768	0.1526	1.171481 x
10000	0.362697	0.306115	1.184839 x

TABLE VI
SHA BENCHMARK: THROUGHPUT COMPARISON (BASELINE VS ACCELERATED)

File Size (KB)	**Standard Time (s)**	**Accelerated Time (s)**	**Improvement %**
100	22.237047	24.52182442	10.27%
500	27.57555703	32.43383498	17.62 %
1000	28.12860398	32.49813136	15.53 %
2000	28.29654782	32.83263564	16.03%
3000	27.36851708	32.81378179	19.90 %
4000	28.000224	32.79306755	17.12%
5000	27.96921149	32.76539974	17.15 %
10000	27.57122336	32.66746157	18.48 %

with speedups ranging from approximately 1.10× for the smallest input (100 KB) to nearly 1.20× for mid-sized workloads (3000 KB). The improvements, though more modest than in AES, remain steady for larger file sizes (500 KB–10 MB), indicating that the acceleration primarily benefits the core hashing process rather than being dominated by setup overhead.

– Throughput improvements mirror the execution time reductions. For 100 KB, throughput increases by about 10.27%, while for larger inputs, improvements range between 15% and 20%. The peak improvements is observed at 3000 KB (19.9%), showing that the hardware acceleration sustains higher data rates for medium to large file sizes.

A. *Visualization of Performance Trends and Analysis*

Figures 2 and 3 illustrate the execution time and throughput trends for AES-128 encryption using the B extension. Figures 4 and 5 depict the SHA-256 hashing performance with the K extension. Overall, the visualizations confirm that:

– B extension delivers substantial acceleration for AES, especially in compute- and memory-intensive workloads.
– K extension provides consistent but moderate improvements for SHA-256, constrained by its sequential dependency chain.

B. *Analysis of Transient Spikes in Execution Time and Throughput*

During extensive testing of the RISC-V processor simulation using QEMU and the GCC toolchain within a Docker environment, occasional spikes were observed in the execution time and throughput graphs when comparing the baseline and extended cores. These spikes are primarily caused by the characteristics and limitations of the emulation environment rather than the performance of the ISA extensions themselves. QEMU performs dynamic instruction translation from RISC-V to host instructions, which can introduce timing variability, especially for small workloads where setup overhead constitutes a larger portion of total execution time. Resource sharing in Docker, such as CPU scheduling and memory access contention with other processes on the host system, can further contribute to intermittent delays. Minor differences in instruction sequences generated by the compiler and variable caching behavior in the emulator also play a role. Averaging results over multiple iterations and testing with larger input sizes can reduce these anomalies, providing more consistent and reliable performance data that accurately reflects the benefits of the RISC-V extensions.

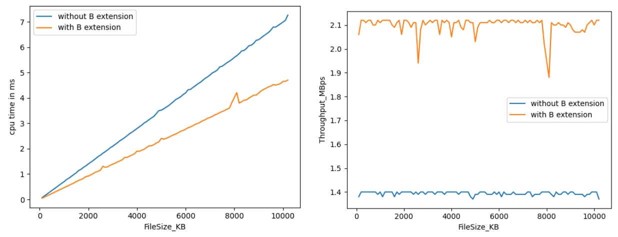

Fig. 2. AES performance: (Left) Execution time vs file size, (Right) Throughput vs file size.

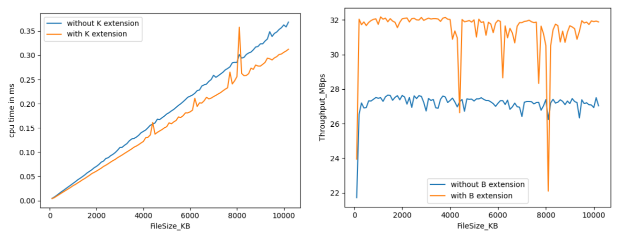

Fig. 3. SHA performance: (Left) Execution time vs file size, (Right) Throughput vs file size.

V. CONCLUSION

This work evaluated the performance benefits of RISC-V Bit-Manipulation (B) and Scalar Cryptography (K) extensions for accelerating AES-128 encryption and SHA-256 hashing in resource-constrained environments. Using a QEMU-based emulation framework with the RISC-V GCC toolchain, cryptographic workloads were benchmarked across file sizes from 100 KB to 10 MB.
For AES-128, the B-extension delivered significant improvements. Execution time reductions ranged from 1.46× to 1.52× for larger inputs, with a 1.50× improvements for smaller files. Throughput improvements increased by 46–52% for large files, confirming that AES

benefits strongly from hardware-optimized bitwise and finite-field operations.

SHA-256, accelerated using the K-extension, exhibited consistent but moderate improvements , with speedups of 1.10×–1.20× and throughput increases of 10–20%, peaking at 19.9% for medium workloads. This demonstrates the efficiency of fused cryptographic instructions for hashing.

Overall, these results confirm that selective RISC-V extensions can substantially enhance cryptographic performance with minimal hardware cost, making them well-suited for secure and efficient computation in IoT and embedded systems.

VI. References

[1] https://roalogic.github.io/RV12/DATASHEET.html

[2] P. S. Babu, S. Sivaraman, D. N. Sarma, and T. S. Warrier, ”Evaluation of Bit Manipulation Instructions in Optimization of Size and Speed in RISC-V,” in *Proc. 34th Int. Conf. on VLSI Design (VLSID)*, 2021, pp. 54-59, doi: 10.1109/VLSID51830.2021.00014.

[3] C. G. de A. Gewehr and F. G. Moraes, ”Improving the Efficiency of Cryptography Algorithms on Resource-Constrained Embedded Systems via RISC-V Instruction Set Extensions,” in *Proc. 36th Symp. on Integrated Circuits and Systems Design (SBCCI)*, 2023, doi: 10.1109/S-BCCI60457.2023.10261964.

[4] K. Kim, D. Harris, and K. Macsai-Goren, ”Design and Synthesis of RISC-V Bit Manipulation Extensions,” in *Proc. 57th Asilomar Conf. on Signals, Systems, and Computers*, 2023, doi: 10.1109/IEEECONF59524.2023.10477073.

[5] D. Markov and A. Romanov, ”Implementation of the RISC-V Architecture with the Extended Zbb Instruction Set,” in *Proc. Int. Ural Conf. on Electrical Power Engineering (UralCon)*, 2022, doi: 10.1109/Ural-Con54942.2022.9906776.

[6] M. A. Elmohr *et al.*, ”Hardware Implementation of A SHA-3 Application-Specific Instruction Set Processor,” in *Proc. Int. Conf. on Microelectronics (ICM)*, 2016, pp. 109-112, doi: 10.1109/ICM.2016.7847846.

[7] C. Gewehr *et al.*, “Hardware Acceleration of Authenticated Encryption with Associated Data via RISC-V Instruction Set Extensions in Low Power Embedded Systems,” Feb. 2024, doi: https://doi.org/10.1109/lascas60203.2024.10506132.

[8] M. Namazi Rizi, N. Zidaric, L. Batina, and N. Mentens, ”Optimised AES with RISC-V Vector Extensions,” in *Proc. 27th Int. Symp. on Design & Diagnostics of Electronic Circuits & Systems (DDECS)*, 2024, doi: 10.1109/DDECS60919.2024.10508919.

Appendices

Attached to this portfolio are the following support documentation:

A. Updated Literature Review and its presentation

B. Design plan

C. Risk assessment

D. Project development log

E. Research log

F. Source code

G. Setup Tutorial

H. Test Result and Calculation

Link to all docs : https://github.com/harshdabhi/riscv_project/tree/main/Appendix