Hardware Lab

Evaluation of cryptographic algorithms and security protocols in embedded systems

Team Members:

Pardis Sadat Zahraei - 99109777
Arian Zamani - 99109036
Seyed Reza Hosseini Dolatabadi - 99109241

The goal of this project is to design and implement a benchmark evaluation system for analyzing the performance of cryptographic algorithms and widely-used security protocols in embedded systems. This project will specifically focus on measuring system performance metrics such as Worst-Case Execution Time (WCET) to identify optimization opportunities. By developing new benchmarks that focus on modern cryptographic techniques, this project will address the problems found in older or expensive benchmark tools like Embench, Mibench, and EEMBC.

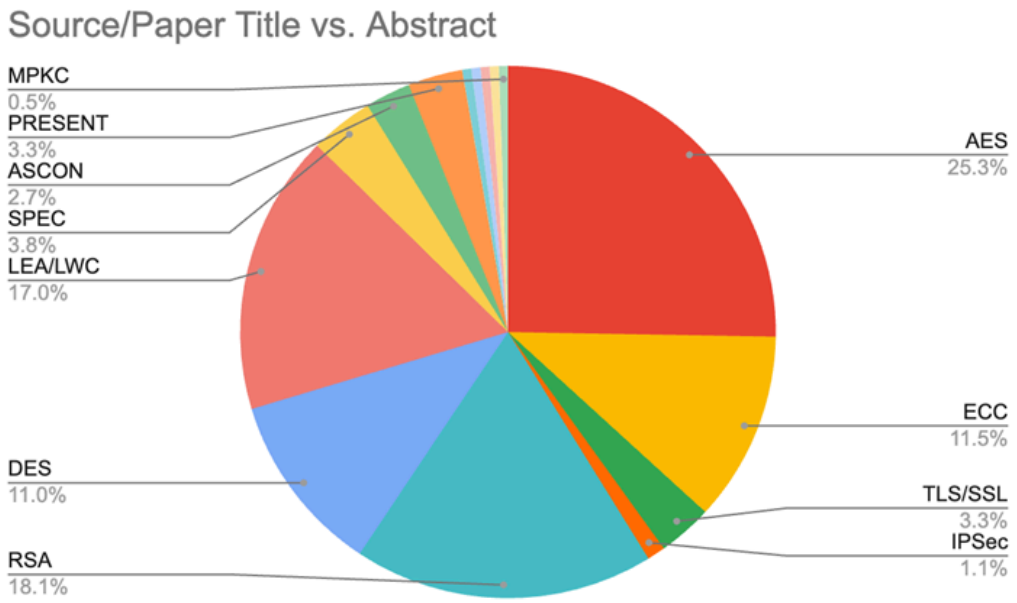
Literature Review and Data Collection

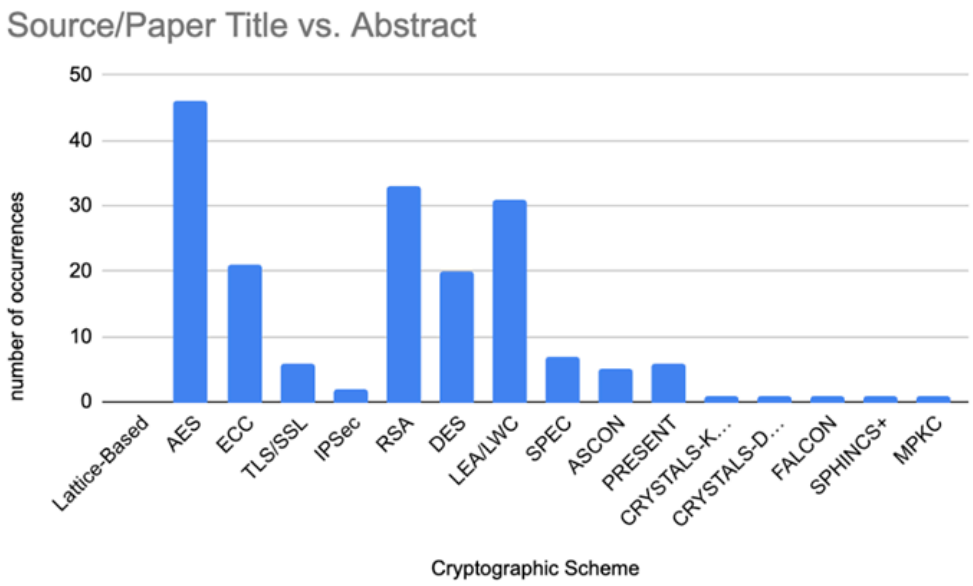
To establish a strong foundation for our research, we conducted a comprehensive review of literature on cryptographic algorithms and benchmarking in embedded systems. This involved analyzing various scholarly papers to gather relevant data and insights.

As part of our analysis, we compiled an extensive dataset, which can be accessed through the following link:

Google Sheets Dataset

Our dataset is based on information extracted from **70 research papers**. The distribution of the collected data i





Then, we wanted to select a few well-chosen algorithms that were prominent in the survey we conducted, as well as one that may not have received much attention. This way, we cover a broad spectrum of them. For this, we chose the AES, RSA, DES, and PRESENT algorithms.

Why These Four Algorithms Matter in Embedded Systems?

Embedded systems have constraints such as **low power, limited memory, and real-time processing requirements**. Cryptographic algorithms in these systems must be **efficient, secure, and optimized for hardware implementation**.

These four algorithms are key for benchmarking because they represent different cryptographic principles:

- **AES (Advanced Encryption Standard)** – The most widely used **symmetric encryption algorithm** for secure data transmission.
- **RSA (Rivest-Shamir-Adleman)** – A **public-key algorithm** for secure key exchange and authentication.
- **DES (Data Encryption Standard)** – A **historical symmetric encryption** algorithm still used in some legacy systems.
- **PRESENT** – A **lightweight block cipher** designed specifically for resource-constrained embedded applications.

---

1. AES (Advanced Encryption Standard)

- **Type**: Symmetric block cipher
- **Block Size**: 128 bits
- **Key Size**: 128, 192, or 256 bits
- **Security**: Highly secure and resistant to cryptanalysis.
- **Why It's Important**:
    - Standard encryption for embedded systems (e.g., IoT, smart cards).
    - Efficient on both software and hardware.
    - Strong resistance to attacks like differential and linear cryptanalysis.
- **Benchmark Focus**:
    - Encryption/decryption speed, power consumption, and memory usage.

---

2. RSA (Rivest-Shamir-Adleman)

- **Type**: Asymmetric (Public-Key) encryption
- **Key Size**: 1024, 2048, or 4096 bits
- **Security**: Security increases with key size but is computationally expensive.
- **Why It's Important**:
    - Used for secure **key exchange, authentication, and digital signatures**.
    - Essential for embedded systems requiring secure communication.
    - Computationally expensive, making it slower than symmetric encryption.

- **Benchmark Focus**:
    - Key generation time, encryption/decryption speed, and energy consumption.

---

3. DES (Data Encryption Standard)

- **Type**: Symmetric block cipher
- **Block Size**: 64 bits
- **Key Size**: 56 bits
- **Security**: Weak by modern standards (susceptible to brute-force attacks).
- **Why It's Important**:
    - Historical significance: DES was the first widely used encryption standard.
    - Still used in some **legacy systems and hardware-constrained environments**.
    - Fast encryption, but vulnerable to modern attacks.
- **Benchmark Focus**:
    - Speed, power consumption, and security level compared to AES.

---

4. PRESENT (Lightweight Block Cipher)

- **Type**: Symmetric block cipher
- **Block Size**: 64 bits
- **Key Size**: 80 or 128 bits
- **Security**: Strong for lightweight encryption but with a smaller block size than AES.
- **Why It's Important**:
    - Designed specifically for **low-power embedded systems and IoT**.
    - More energy-efficient than AES and DES.

    - Suitable for applications where memory and power are limited.
- **Benchmark Focus**:
    - Hardware efficiency, power consumption, and security trade-offs.

Comparison Table

Algorithm	Type	Block Size	Key Size (bits)	Security Level	Speed	Power Consumption	Use Case in Embedded Systems
AES	Symmetric	128 bits	128, 192, 256	Very strong	Fast	Moderate	Secure communication, data encryption
RSA	Asymmetric	N/A	1024, 2048, 4096	High (with large keys)	Slow	High	Key exchange, authentication
DES	Symmetric	64 bits	56	Weak (brute-force attacks possible)	Fast	Low	Legacy systems, financial applications

PRESENT	Symmetric	64 bits	80, 128	Moderate (lightweight security)	Fast	Very Low	IoT, RFID, smart cards

---

Why These Four Are Key for Benchmarking?

Benchmarking cryptographic performance in embedded systems focuses on:

1. **Computational Efficiency** – AES and PRESENT are fast, while RSA is computationally expensive.
2. **Memory and Storage Requirements** – PRESENT is lightweight, while RSA requires significant memory.
3. **Power Consumption** – PRESENT is optimized for minimal energy use, while RSA is power-hungry.
4. **Security** – AES is highly secure, whereas DES is weak but still relevant in legacy systems.

These algorithms cover **a wide spectrum of use cases** in embedded systems, making them essential for evaluating cryptographic performance.

Next, we analyze the benchmarks, the algorithms they each cover, their usages, and a comparison between them.

Report on Cryptographic Benchmarks for Embedded Systems

Cryptographic benchmarks are essential for assessing the performance, resource usage, and security implications of algorithms deployed in embedded systems.

Cryptographic benchmarks are tools and frameworks used to evaluate the efficiency, robustness, and performance of cryptographic algorithms. They typically measure factors like:

- **Speed**: Time taken for encryption, decryption, key generation, etc.
- **Memory Usage**: Amount of memory consumed during operation.
- **Energy Efficiency**: Especially critical for battery-operated devices.
- **Algorithm Suitability**: Tailoring algorithms for specific embedded or constrained environments.

- **Open Source/Closed Source**
- **Hardware Platforms**

These benchmarks are vital for IoT, secure communication systems, and embedded devices operating in hostile or resource-limited settings.

Prominent Cryptographic Benchmarks

1. SPEC CPU2017

- **Purpose**:
  Primarily used for evaluating general CPU performance with a focus on cryptographic workloads. It is well-suited for devices that require robust computation for tasks like secure data encryption and decryption.
- **Key Features**:
  - Provides insights into system performance using standard cryptographic libraries.
  - Emphasizes both symmetric (AES) and asymmetric (RSA) cryptographic operations.
- **Use Cases**:
  - High-performance computing systems.
  - Embedded systems requiring significant cryptographic processing.
- **Challenges**:
  - May be computationally intensive for highly resource-constrained devices.
- **Open Source/Closed Source**: Closed source.
- **Hardware Platforms**: General-purpose CPUs; commonly tested on Intel, AMD, and ARM-based systems.
- **Notes**: Requires substantial computational resources, making it less ideal for very constrained embedded systems.

---

2. EEMBC IoTConnect

- **Purpose**:
  Developed for IoT environments, this benchmark is optimized for lightweight devices with constrained resources. It evaluates performance in real-world scenarios, such as secure communication protocols in IoT networks.
- **Key Features**:
  - Focuses on speed and energy efficiency.
  - Includes algorithms like AES (encryption) and ECC (key exchange).
  - Addresses IoT-specific challenges like intermittent connectivity and low power.
- **Use Cases**:
  - IoT devices in healthcare, smart cities, and industrial automation.
  - Evaluating secure communication protocols like MQTT and CoAP.

- **Challenges**:
    - Limited suitability for high-throughput applications.
- **Open Source/Closed Source:** Closed source; requires EEMBC membership for access.
- **Hardware Platforms:** Designed for IoT-focused microcontrollers and SoCs like ARM Cortex-M series**.**
- **Notes:** Provides energy profiling through the EEMBC EnergyMonitor hardware tool.

---

3. NIST Lightweight Cryptography (LWC)

- **Purpose**:
  A NIST initiative to identify and standardize lightweight cryptographic algorithms for constrained environments. It evaluates performance on devices with limited computational power and memory.
- **Key Features**:
    - Focus on lightweight algorithms suitable for IoT and embedded systems.
    - Includes a comprehensive evaluation of encryption, hashing, and authenticated encryption algorithms.
    - Algorithms such as ASCON, GIFT-COFB, and Xoodyak are assessed.
- **Use Cases**:
    - Devices requiring ultra-light cryptographic solutions, such as RFID tags, smartcards, and low-power IoT nodes.
- **Challenges**:
    - Still in the standardization phase; may require further validation for specific use cases.
- **Open Source/Closed Source**: Open source; algorithms and tools provided by contributors to the NIST initiative.
- **Hardware Platforms**: Simulated and real hardware, including microcontrollers like STM32 and ESP32.
- **Notes**: Covers a variety of cryptographic schemes focusing on IoT and constrained environments.

---

4. BEEBS (Benchmarks for Embedded Execution and Benchmarking Suite)

- **Purpose**:
  This benchmark suite is designed for embedded processors and provides a realistic measure of cryptographic performance in constrained environments.
- **Key Features**:
    - Benchmarks commonly used cryptographic primitives like AES and RSA.
    - Suitable for both research and development in the embedded domain.
    - Includes a variety of workloads to test diverse processor architectures.

- **Use Cases**:
    - Academic research on embedded cryptography.
    - Testing performance on low-power microcontrollers.
- **Challenges**:
    - Limited focus on energy profiling compared to newer benchmarks.
- **Open Source/Closed Source:** Open source.
- **Hardware Platforms:** Targets embedded processors; compatible with ARM Cortex-M, AVR, and RISC-V boards.
- **Notes:** A versatile benchmark used in academic research and testing.

---

5. RIOT Cryptographic Benchmark

- **Purpose**:
  Specifically tailored for the RIOT operating system, this benchmark evaluates cryptographic operations in highly constrained IoT environments.
- **Key Features**:
    - Targets the cryptographic needs of RIOT-based devices.
    - Includes AES, ECC (Curve25519), and RSA for testing cryptographic strength and efficiency.
    - Focuses on interoperability with lightweight network stacks.
- **Use Cases**:
    - IoT sensor networks.
    - Battery-operated devices running RIOT OS.
- **Challenges**:
    - Limited to RIOT-specific use cases; lacks generalization to other operating systems.
- **Open Source/Closed Source**: Open source.
- **Hardware Platforms**: Tailored for devices running RIOT OS; tested on boards like STM32 Nucleo and Arduino.
- **Notes**: Focuses on cryptographic efficiency for RIOT OS, but less generalized for other platforms.

---

6. WolfSSL Benchmarking Suite

- **Purpose**:
  A specialized toolset for evaluating the performance of cryptographic algorithms implemented in the WolfSSL library. It is highly optimized for embedded systems and IoT devices.
- **Key Features**:

    - Measures the speed of encryption, decryption, signing, and verification operations.
    - Includes a wide range of algorithms like AES, RSA, ECC, ChaCha20, and Poly1305.
    - Lightweight and designed for resource-constrained environments.
- **Use Cases**:
    - IoT devices requiring optimized cryptographic operations.
    - Embedded systems in need of secure communication stacks.
- **Challenges**:
    - Limited to WolfSSL implementations; might not generalize to other libraries.
- **Open Source/Closed Source**: Dual-licensed; open source under GPL and available under commercial licensing.
- **Hardware Platforms**: Optimized for ARM Cortex-M, ESP32, and other microcontrollers.
- **Notes**: Strong emphasis on TLS performance for embedded systems.

---

7. mBench Crypto

- **Purpose**:
  A benchmarking tool tailored for embedded platforms, focusing on evaluating the performance of cryptographic algorithms in realistic scenarios.
- **Key Features**:
    - Includes algorithms like AES, SHA-256, and ECC.
    - Provides insights into latency and throughput under constrained environments.
    - Optimized for microcontroller platforms.
- **Use Cases**:
    - Microcontrollers and embedded platforms.
    - Applications where latency is critical, such as real-time systems.
- **Challenges**:
    - Limited focus on asymmetric cryptography compared to other tools.
- **Open Source/Closed Source:** Open source**.**
- **Hardware Platforms:** Embedded platforms like ARM Cortex-M and ESP32.
- **Notes:** Focuses on evaluating latency and throughput on constrained devices.

---

8. Ascon Benchmark

- **Purpose**:
  A benchmark designed around the Ascon algorithm, a finalist in the NIST Lightweight Cryptography competition, aimed at secure and efficient encryption for IoT devices.
- **Key Features**:

    - Focuses on Ascon's authenticated encryption and hashing capabilities.
    - Demonstrates low memory usage and high speed in constrained environments.
    - Suitable for both software and hardware implementations.
- **Use Cases**:
    - IoT devices with severe resource constraints.
    - Lightweight applications requiring authenticated encryption.
- **Challenges**:
    - Limited to the Ascon algorithm; does not benchmark a variety of algorithms.
- **Open Source/Closed Source**: Open source.
- **Hardware Platforms**: Embedded devices like ARM Cortex-M series and custom ASICs.
- **Notes**: Concentrates exclusively on the Ascon algorithm, particularly relevant for NIST LWC.

---

9. Post-Quantum Cryptography Benchmark

- **Purpose**:
  Evaluates the performance of cryptographic algorithms designed to be secure against quantum computing attacks, focusing on future-proof security for embedded systems.
- **Key Features**:
    - Includes algorithms shortlisted by NIST for post-quantum cryptography (e.g., Kyber, Dilithium, and Falcon).
    - Measures key generation, signing, and verification times.
    - Analyzes memory and bandwidth requirements.
- **Use Cases**:
    - Critical infrastructure requiring long-term security.
    - IoT and embedded systems preparing for a post-quantum future.
- **Challenges**:
    - Algorithms are still evolving, and standards are not finalized.
- **Open Source/Closed Source:** Open source; most implementations are contributed to the NIST PQC project.
- **Hardware Platforms:** Various; includes general CPUs and embedded devices like ARM Cortex-M.
- **Notes:** Still evolving due to the ongoing standardization of post-quantum algorithms.

---

10. Performance of Symmetric and Lightweight Cryptographic Algorithms

- **Purpose**:
  A comprehensive evaluation of various symmetric and lightweight cryptographic algorithms to assess their suitability for constrained environments.
- **Key Features**:
  - Includes symmetric encryption algorithms like AES and ChaCha20.
  - Benchmarks lightweight alternatives such as GIFT and Present.
  - Provides insights into energy consumption and computational overhead.
- **Use Cases**:
  - Battery-powered devices such as wearables.
  - Applications requiring energy-efficient encryption.
- **Challenges**:
  - Limited to symmetric cryptographic algorithms; lacks asymmetric algorithm evaluation.
- **Open Source/Closed Source:** Varies by study; many benchmarks are open source in academic publications.
- **Hardware Platforms:** Microcontrollers like STM32, AVR, and low-power IoT devices.
- **Notes:** Focuses on energy efficiency and lightweight algorithm performance in real-world scenarios.

General Hardware Platforms

These hardware categories are broadly used across benchmarks:

1. **Microcontrollers (MCUs)**:
   - **Examples**: ARM Cortex-M series, AVR, PIC, ESP32, STM32.
   - **Features**: Low-power consumption, limited memory, designed for IoT and embedded systems.
   - **Use Cases**: Lightweight cryptography, energy-efficient applications.
2. **System-on-Chip (SoCs)**:
   - **Examples**: Raspberry Pi (Broadcom), Qualcomm Snapdragon (IoT-specific versions), NXP i.MX RT series.
   - **Features**: Combines CPU, GPU, memory, and I/O on one chip; mid-range performance.
   - **Use Cases**: Applications with moderate processing and cryptographic needs.
3. **Application Processors**:
   - **Examples**: ARM Cortex-A series, Intel Atom, AMD Ryzen Embedded.
   - **Features**: Higher performance, more memory; used in applications requiring intensive computations.
   - **Use Cases**: High-performance cryptography, secure gateways, or servers in constrained environments.
4. **Custom Hardware/ASICs**:
   - **Examples**: FPGA-based platforms (e.g., Xilinx, Altera), ASICs designed for cryptographic processing.

    - **Features**: Optimized for speed and energy efficiency for specific cryptographic algorithms.
    - **Use Cases**: High-speed encryption, post-quantum cryptography, and real-time systems.

Specific Boards

These boards are frequently mentioned in the context of cryptographic benchmarks:

1. ARM Cortex-M Boards

- **Common Models**:
    - STM32 Nucleo Boards (e.g., Nucleo-F446RE)
    - TI Tiva C LaunchPad
    - Nordic Semiconductor nRF52840
- **Features**: Widely used in IoT and embedded systems. Features support for AES, ECC, and lightweight cryptography.
- **Benchmarks Used**:
    - NIST LWC
    - WolfSSL Benchmark Suite
    - mBench Crypto

---

2. AVR Boards

- **Common Models**:
    - Arduino Uno (ATmega328P)
    - Arduino Mega (ATmega2560)
- **Features**: Low-power, basic capabilities for lightweight cryptography; good for academic research.
- **Benchmarks Used**:
    - BEEBS
    - Performance of Symmetric Cryptographic Algorithms

---

3. ESP32 (by Espressif Systems)

- **Features**: Dual-core MCU with Wi-Fi and Bluetooth. Integrated cryptographic accelerators for AES, RSA, and SHA.
- **Applications**: IoT devices, smart home systems.
- **Benchmarks Used**:
    - NIST LWC

    - WolfSSL
    - mBench Crypto

---

4. Raspberry Pi Series

- **Common Models**: Raspberry Pi 4, Raspberry Pi Zero W.
- **Features**: Versatile and cost-effective platform for testing cryptographic algorithms on general-purpose CPUs.
- **Applications**: High-level prototyping, post-quantum cryptography.
- **Benchmarks Used**:
    - SPEC CPU2017
    - Post-Quantum Cryptography Benchmark

---

5. FPGA Boards

- **Common Models**:
    - Xilinx Zynq-7000 SoC
    - Intel/Altera Cyclone V
- **Features**: Configurable for optimized cryptographic tasks, high performance in constrained energy envelopes.
- **Applications**: Custom cryptographic implementations, high-speed encryption.
- **Benchmarks Used**:
    - Ascon Benchmark
    - Post-Quantum Cryptography Benchmark

---

6. NXP i.MX RT Series

- **Common Models**: i.MX RT1060, RT1050.
- **Features**: Cross between an MCU and an SoC, with secure boot and cryptographic accelerators.
- **Applications**: IoT gateways, secure payment systems.
- **Benchmarks Used**:
    - WolfSSL
    - NIST LWC

---

7. Intel Atom and AMD Ryzen Embedded

- **Common Models**:

    - Intel Atom x6000E series
    - AMD Ryzen Embedded V1000
- **Features**: Supports higher-level cryptographic operations and suitable for resource-rich embedded systems.
- **Applications**: Secure industrial systems, IoT hubs.
- **Benchmarks Used**:
    - SPEC CPU2017
    - Post-Quantum Cryptography

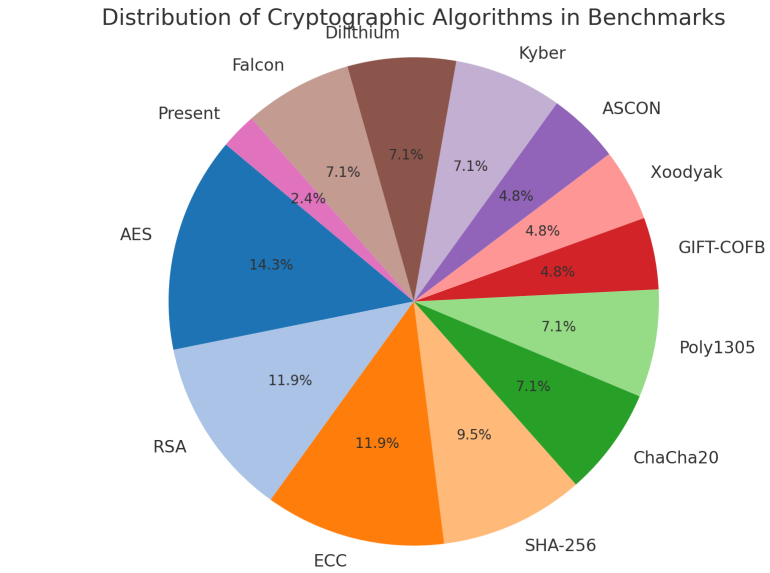
Energy Monitoring Hardware

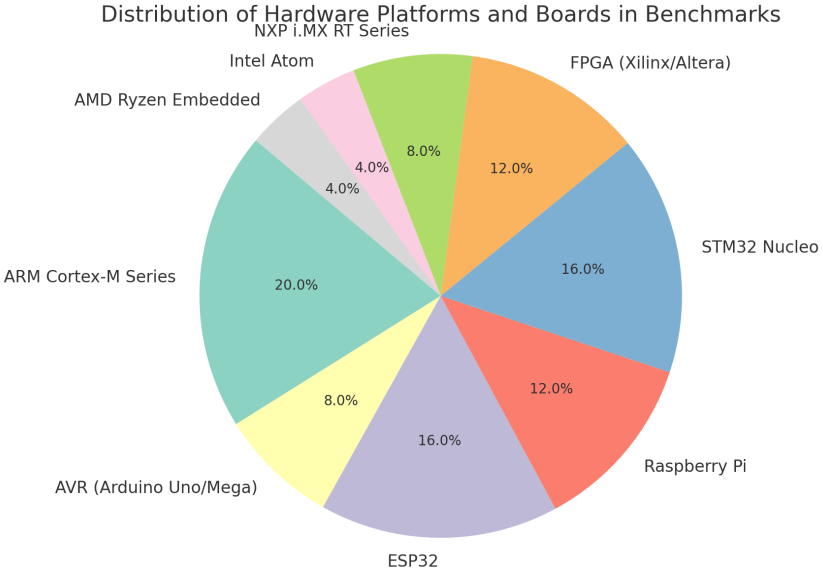
For energy efficiency profiling, these tools are often paired with hardware platforms:

- **EEMBC EnergyMonitor**: Compatible with IoTConnect benchmarks; measures power consumption of MCUs.
- **Joulescope**: General-purpose energy analyzer for embedded systems.
- **Monsoon Power Monitor**: Used to track current and voltage in IoT devices.

Hardware and Platform Links

- **STM32 Boards**: STM32 Official
- **ESP32**: ESP32 Official
- **Raspberry Pi**: Raspberry Pi Official
- **Xilinx FPGA Boards**: Xilinx Official





Introducing Our Cryptographic Benchmark

After analyzing numerous existing cryptographic benchmarks, we identified a key limitation: none adequately account for the worst-case scenario with a 99.7% confidence level. Since confidence and reliability are paramount in cryptographic performance evaluation, our benchmark introduces two new factors:

1. **Standard Deviation (σ):** Providing insights into performance variability.
2. **Average + 3σ:** Offering a statistically backed worst-case performance metric.

By incorporating these elements, our benchmark delivers a more detailed and insightful analysis compared to conventional worst-case assessments.

Key Features

- **Enhanced Statistical Analysis:** Evaluates cryptographic performance with added reliability metrics.
- **Open-Source Accessibility:** Freely available to the community for modification and improvement.
- **Hardware Independence:** Designed to work on any microcontroller or CPU without requiring additional libraries.

Hardware Platform: STM32

Our implementation is optimized for STM32 microcontrollers, leveraging their advantages:

- **High Performance & Low Power:** STM32 boards offer an excellent balance between computational power and energy efficiency.
- **Robust Security Features:** Many STM32 chips come with hardware-accelerated cryptographic support.
- **Wide Adoption & Support:** Well-documented and supported by a large developer community.

Despite this optimization, our benchmark is **not limited to STM32**. It can be seamlessly deployed on other platforms, including ARM Cortex-M, ESP32, RISC-V, and general-purpose CPUs, without requiring additional libraries or dependencies.

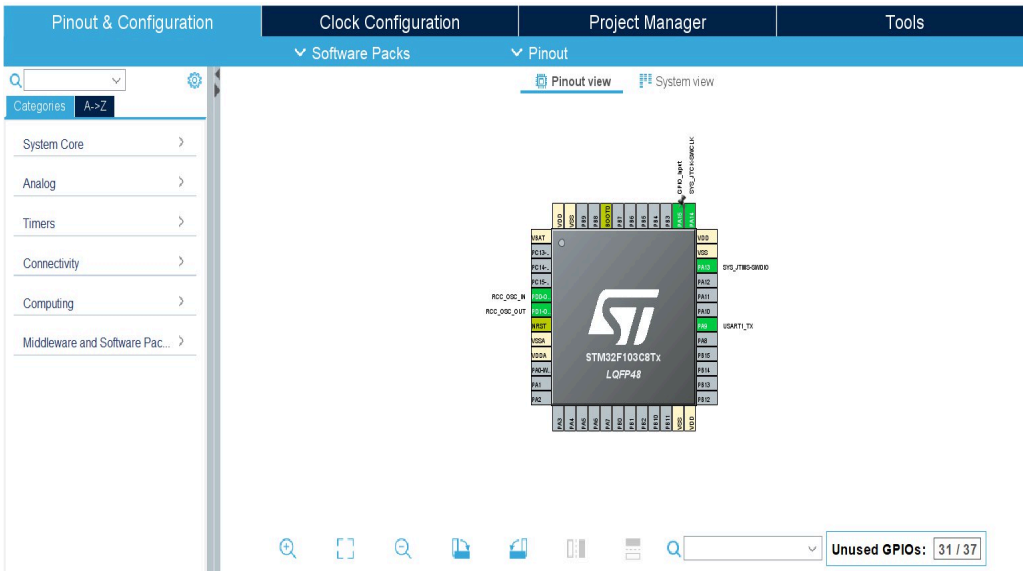
Use Cases

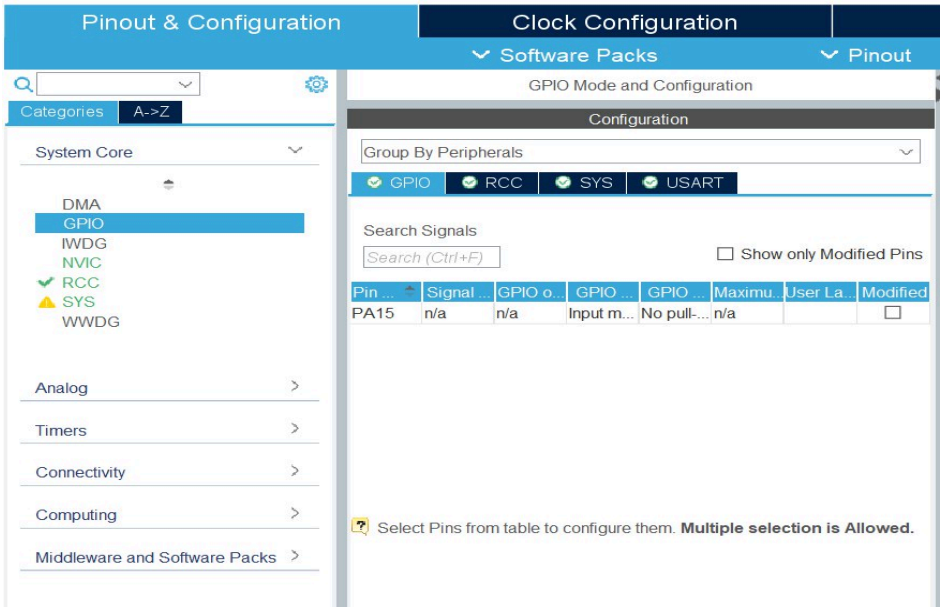
- Evaluating cryptographic performance for embedded and IoT systems.
- Comparing algorithm efficiency with a statistically backed worst-case analysis.
- Enhancing reliability assessments for security-critical applications.

By addressing the limitations of existing benchmarks, our framework provides a **more comprehensive** and **accessible** tool for cryptographic performance evaluation.

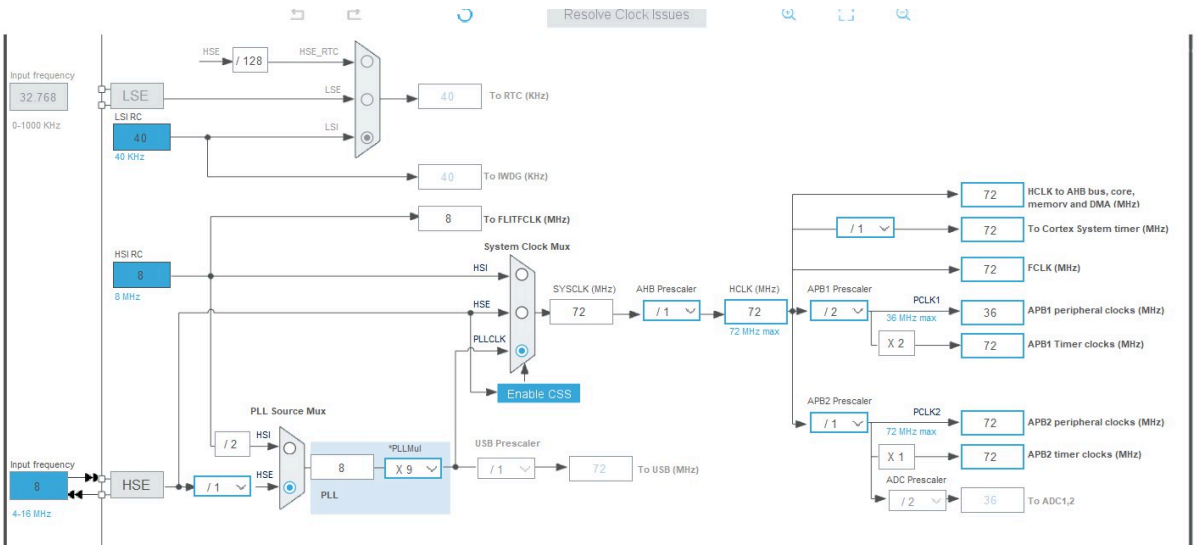
Implementation

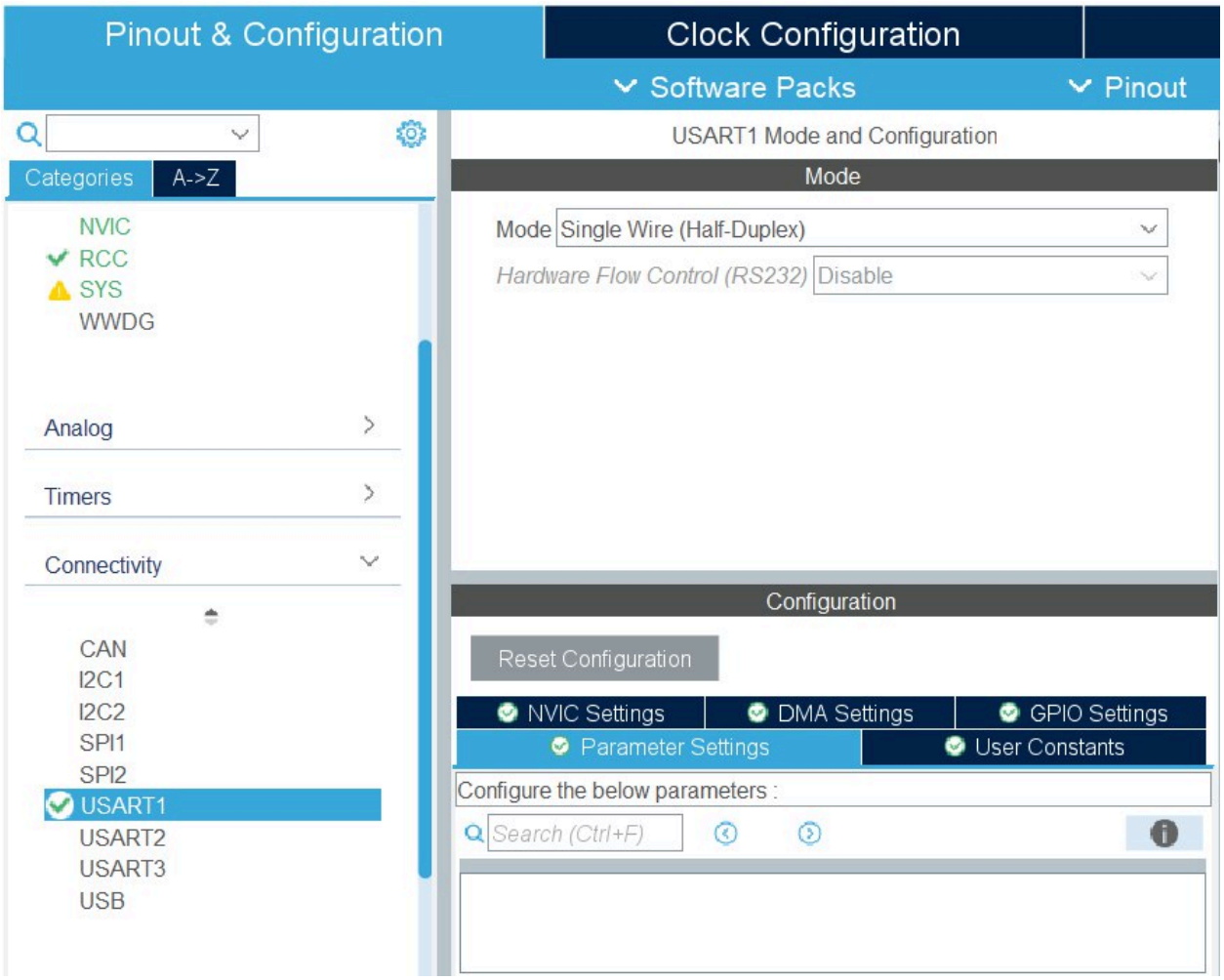
This is the scheme of the initial page of creating an MCU project. By clicking on each pin, one can easily rename it or set its type (input, output, etc.).





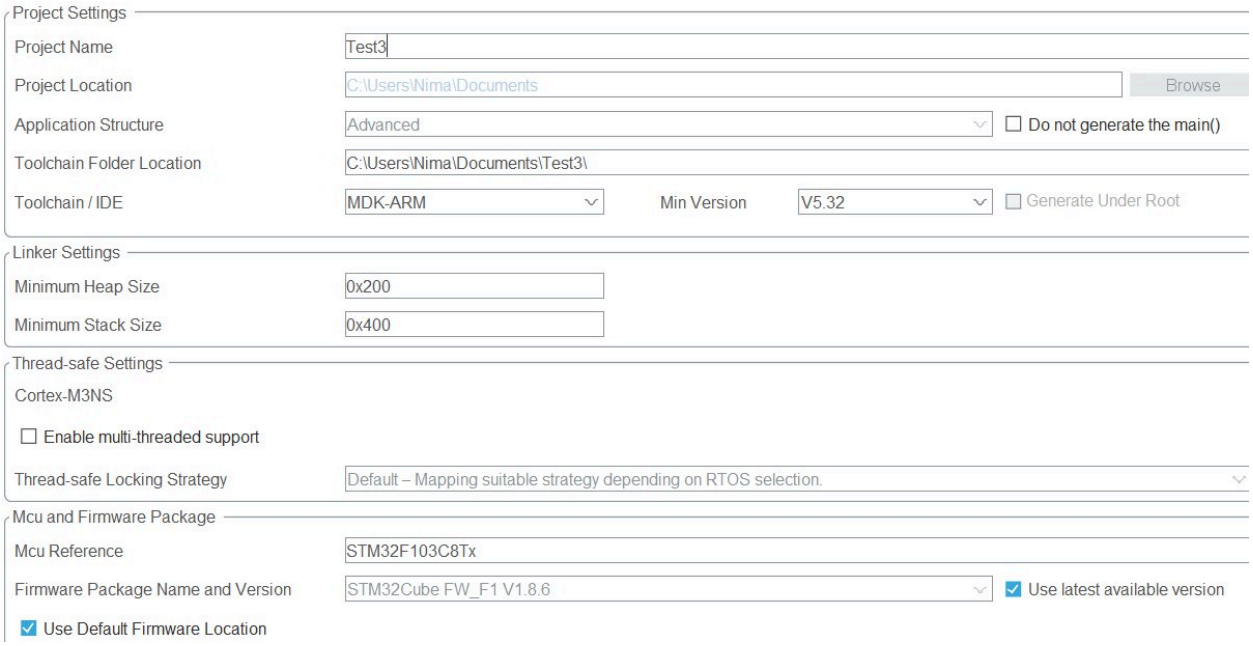
Under the Clock Configuration tab, we can set the frequency of the board to its maximum.



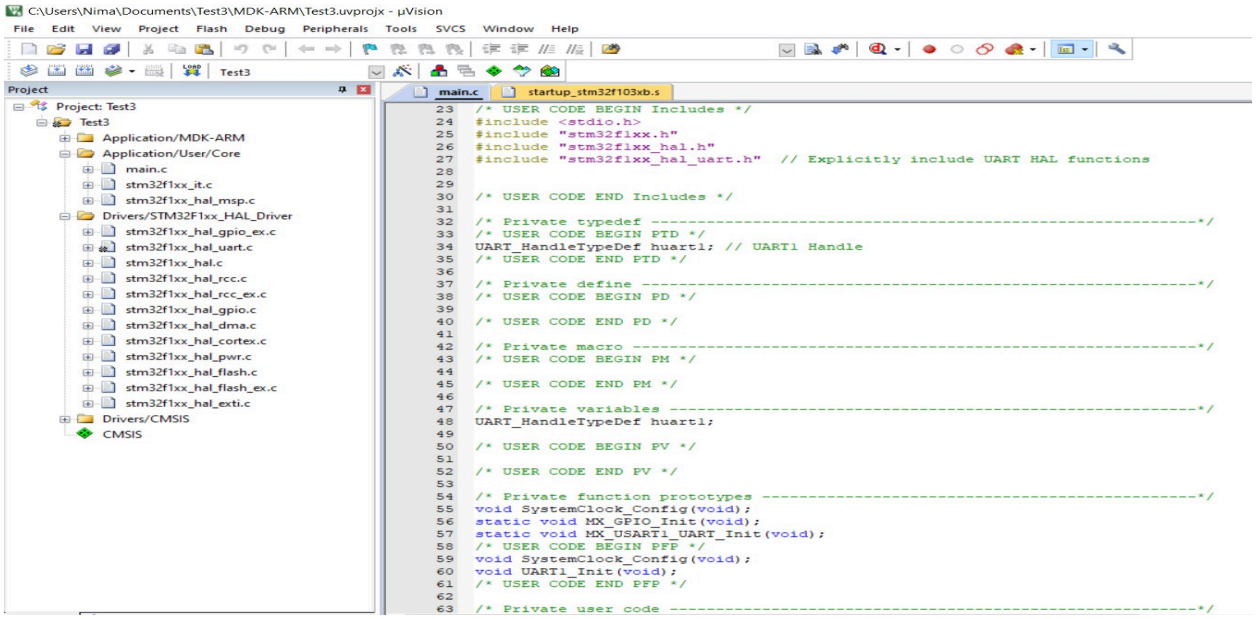


Under the Project Manager tab, we set the following settings for the project. Make sure to select the proper IDE and Compiler “MDK-ARM” in order to be able to edit the project

via “Keil”. After doing this step, click “Generate Code” and the codes related to the settings are automatically generated and the project will be initiated.



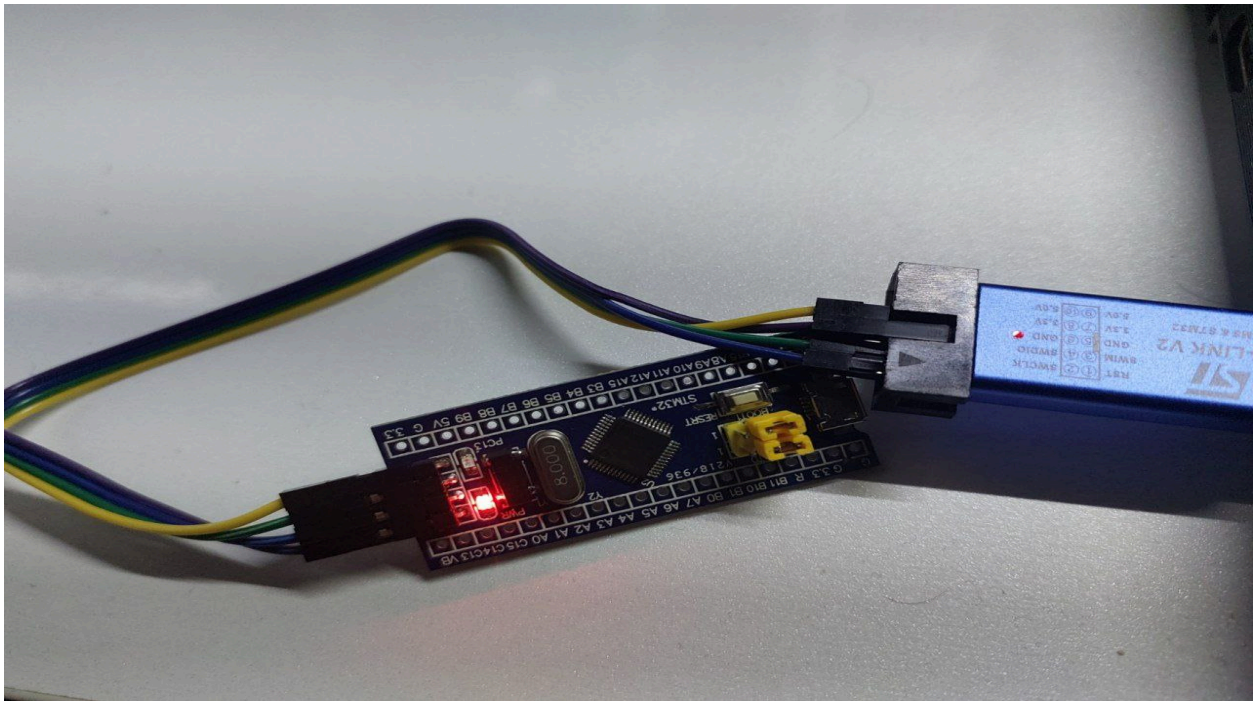
This is the generated code. To make sure every connection is all right, we can check the device’s serial number through the “Target” settings under the “debug” tab. Then we can build the code and see if it’s error-free.



This is the output of the second successful build. Then we can download the project into the STM32 board and run it.

```
Build Output
Build started: Project: Test3
*** Using Compiler 'V6.21', folder: 'C:\Keil_v5\ARM\ARMCLANG\Bin'
Build target 'Test3'
"Test3\Test3.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:00
```

This is the board after loading the codes into it.



```
Load "Test3\\Test3.axf"
Erase Done.
Programming Done.
Verify OK.
Application running ...
Flash Load finished at 23:17:00
```

Challenges

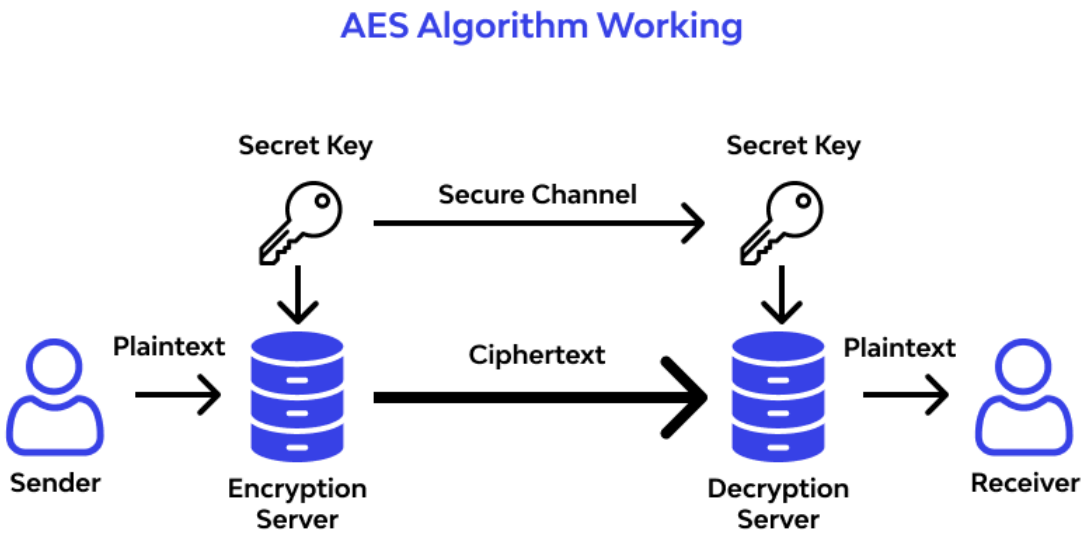
One of the main challenges in this project was the limited availability of resources and tutorials for working with the STM32 board, making it difficult to implement and debug cryptographic benchmarks efficiently. Additionally, the development process was hindered by frequent bugs, requiring extensive troubleshooting and debugging efforts. These challenges not only slowed progress but also necessitated a deeper understanding of the hardware and low-level optimizations to ensure accurate performance measurements.

Codes

AES:

This code implements AES algorithm in C, providing both encryption and decryption capabilities with support for 128, 192, and 256-bit key sizes. The implementation follows AES's core operations including SubBytes (using S-box substitution), ShiftRows (cyclic shifting of rows), MixColumns (matrix multiplication in Galois Field), and AddRoundKey (XOR with round key). The code includes a key expansion routine that generates round keys, and the main encryption process performs 10, 12, or 14 rounds depending on the key size, with each round applying these operations in sequence except for the final round which skips MixColumns.
The implementation also features a comprehensive performance measurement system that analyzes the algorithm's execution characteristics over multiple iterations. It calculates key performance metrics including average execution time, worst-case execution time, standard deviation, and the three-sigma boundary (average + 3*standard deviation), which provides a statistical upper bound for execution time with 99.7% confidence. The code uses proper memory management with dynamic allocation for expanded keys, includes basic error handling, and implements lookup tables for

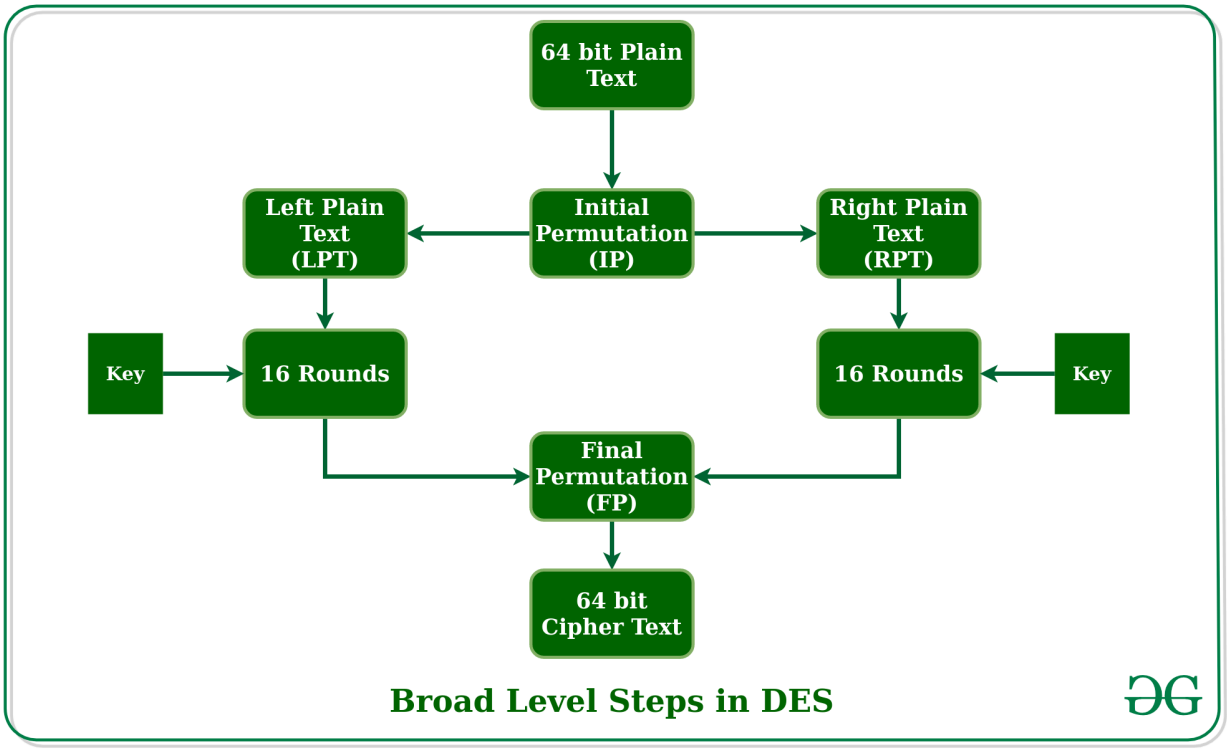
efficient S-box operations, though it's worth noting that this basic implementation might need additional security hardening for production use.



DES:

This code implements the Data Encryption Standard (DES) algorithm, a symmetric-key block cipher that operates on 64-bit blocks of data using a 56-bit key (though a 64-bit key is input, with 8 bits used for parity). The code includes all the standard DES components: initial permutation (IP), 16 rounds of Feistel network operations, final inverse permutation (IP^{-1}), and key scheduling. The implementation includes lookup tables for various DES operations including S-boxes, permutation tables, and key schedule shifts.

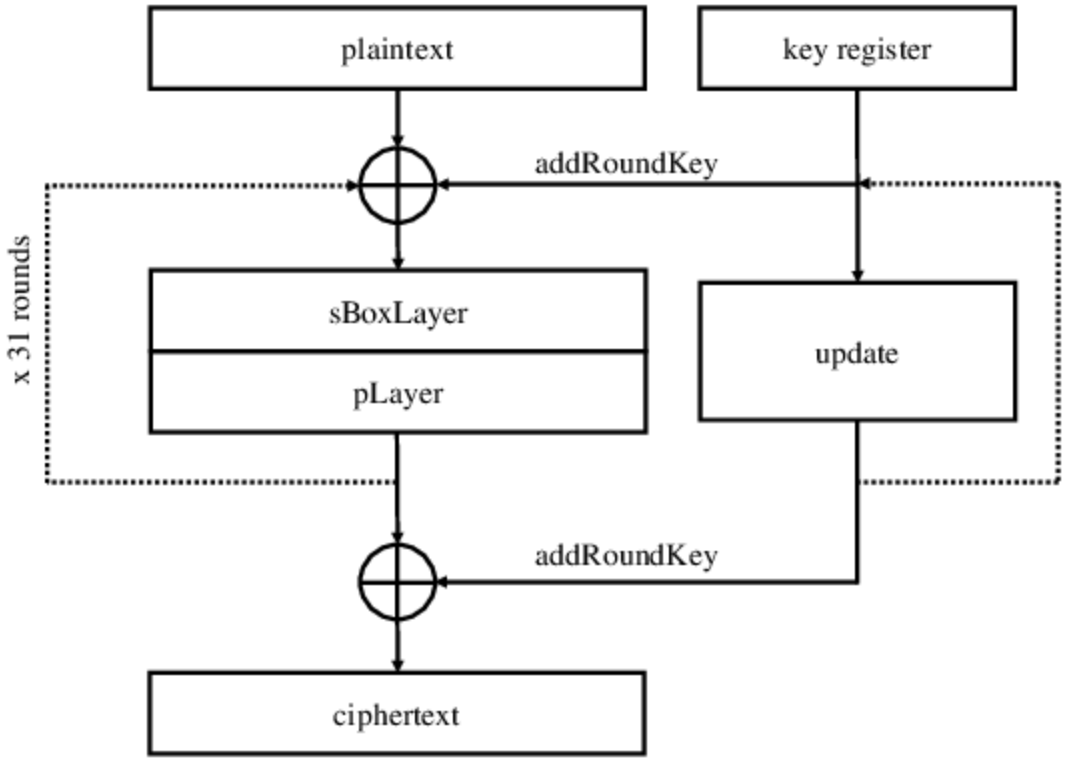
The main function demonstrates the implementation by performing encryption and decryption operations in a loop to measure performance metrics. It takes a test input (0x9474B8E8C73BCA7D) and runs 100 iterations of encryption followed by decryption, measuring execution time for each iteration. The code calculates and outputs performance statistics including average execution time, worst-case execution time, standard deviation, and a "three sigma" boundary (average + 3 * standard deviation). The specific implementation follows the FIPS PUB 46-3 standard and uses bitwise operations extensively to perform the required permutations and transformations of the data.



PRESENT:

This code implements the PRESENT block cipher, a lightweight cryptographic algorithm designed for resource-constrained devices. The implementation handles 64-bit blocks using an 80-bit key and operates over 32 rounds. The core components include an SBox for nibble substitution (4-bit segments), a permutation layer that rearranges bits according to a fixed pattern, and a key scheduling algorithm that generates round keys. The code provides functions to convert between different data representations (hex strings, bytes, and 64-bit integers) and implements both encryption and decryption operations.

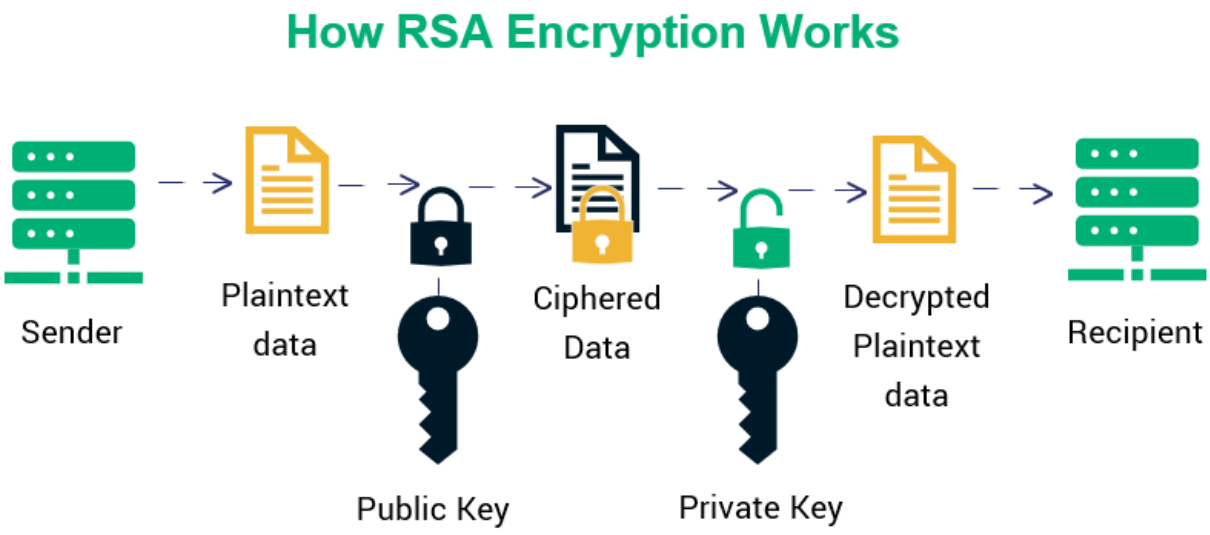
The implementation is structured around two main operations: encryption and decryption, with the decryption process being the inverse of encryption. Each encryption round consists of three steps: XORing the state with a round key, applying the SBox substitution to each nibble, and performing the bit permutation. The key schedule expands the 80-bit master key into 32 round keys through a process of rotation, SBox application to the most significant nibble, and XORing with a round counter. The main function includes benchmarking code that measures execution time statistics (average, worst-case, and standard deviation) over 100 iterations of encrypting and decrypting a test message.



RSA:

This code implements the RSA (Rivest-Shamir-Adleman) encryption algorithm, a widely used public-key cryptosystem. The core functions include `isPrime` for primality testing, `gcd` for finding greatest common divisors, `totient` for calculating Euler's totient function, and functions to generate public (`randome`) and private (`private_key`) keys. The encryption and decryption processes are handled by the `encrypt` and `decrypt` functions respectively, which use modular exponentiation (`pomod`) to perform the cryptographic operations on each character of the input message.

The main function performs a performance analysis of the RSA implementation by running 100 iterations with fixed prime numbers (p=101, q=2029) and a randomly generated 50-character message. For each iteration, it measures the execution time using clock() functions and calculates statistics including average execution time, worst-case time, standard deviation, and the "three-sigma" boundary (average + 3 * standard deviation). This benchmarking approach helps evaluate the algorithm's performance characteristics and reliability under repeated use.



Results

This table shows the evaluation of algorithms on our benchmark:

Algorithm	Average Execution Time (seconds)	Worst Execution Time (seconds)	Standard Deviation	Average + 3σ (seconds)
PRESENT	0.000012	0.000036	0.000025	0.000087
DES	0.000038	0.000066	0.000058	0.000212
RSA	0.000084	0.000213	0.000169	0.000590
AES	0.000168	0.000275	0.000345	0.001202

The benchmark evaluation results reveal a clear hierarchy in performance among the tested cryptographic algorithms. PRESENT emerged as the most efficient algorithm, demonstrating superior performance with the lowest average execution time of 0.012 milliseconds and worst-case execution time of 0.036 milliseconds. DES and RSA followed with moderately higher execution times, while AES showed the highest computational overhead. The standard deviation measurements indicate that PRESENT also offers the most consistent performance, with the least variation in execution times. This makes it particularly suitable for embedded systems where predictable performance is crucial. The Average + 3σ values, representing a 99.7% confidence interval of execution times, further confirm PRESENT's reliability and efficiency

advantage over the other algorithms. These findings suggest that for resource-constrained embedded systems where timing predictability is essential, PRESENT could be the preferred choice, offering an optimal balance between security and performance.

For a better experience, we have also written a GUI so users can just click and have the scoreboard ready! This is really important as others can contribute to our project and modify our code or add new algorithms, and it all works.

```
// Main function to create the GUI window
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {
    WNDCLASS wc = {0};
    wc.lpfnWndProc = WindowProc;
    wc.hInstance = hInstance;
    wc.lpszClassName = "LeaderboardWindow";
    RegisterClass( lpWndClass: &wc);

    HWND hWnd = CreateWindow("LeaderboardWindow", "Execution Time Leaderboard", WS_OVERLAPPEDWINDOW | WS_VISIBLE,
                             100, 100, 800, 700, NULL, NULL, hInstance, NULL);

    MSG msg = {0};
    while (GetMessage( lpMsg: &msg, hWnd: NULL, wMsgFilterMin: 0, wMsgFilterMax: 0)) {
        TranslateMessage( lpMsg: &msg);
        DispatchMessage( lpMsg: &msg);
    }
    return 0;
}
```

Which has a run leaderboard and everyone can use it!