Edge-Connected Microcontroller Security

Divya Syal, Gavin Ryder, Neena Ekanathan

Advisors: Behnam Dezfouli, Yuhong Liu

Project Supported by STMicroelectronics

Overview

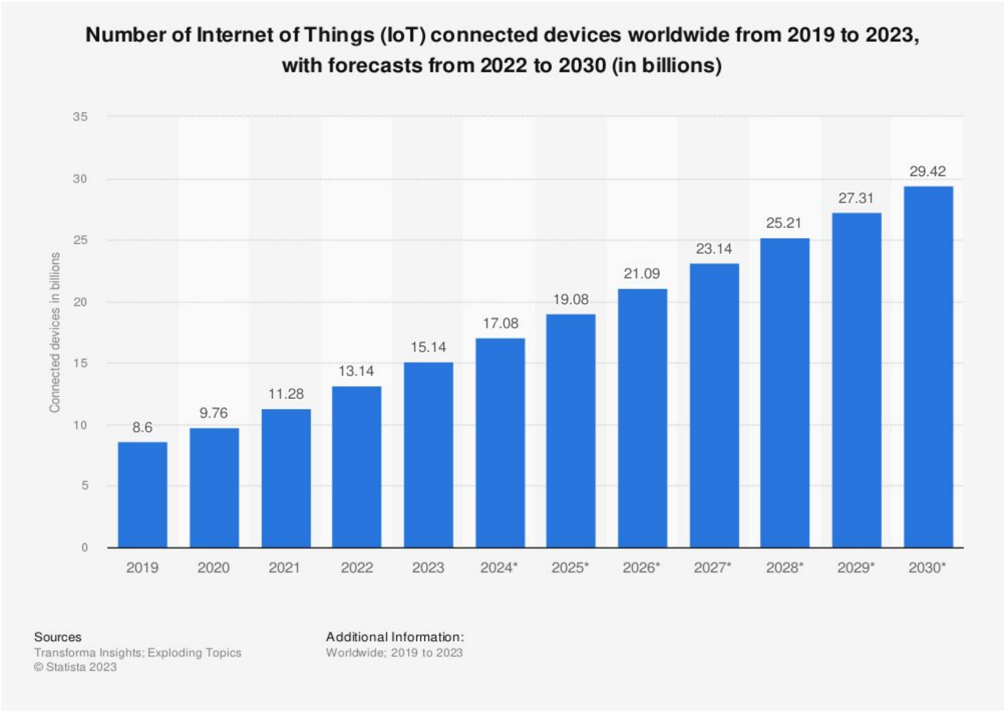
- **Section 1: Project Background**
- Section 2: Design Process and Testing
- Section 3: Results and Analysis
- Section 4: Future Work

Background

MCU Technical Background

What are MCUs?

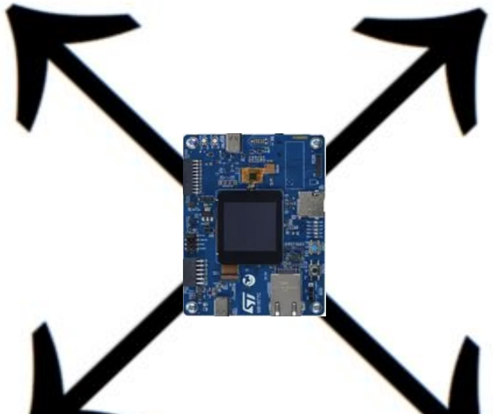
- **Versatile, compact embedded processors**
  - Specializes in one operation
- **Microcontroller devices and applications are more prevalent than ever**
  - Integrated into a range of applications
- **Software must be performant and secure**
  - Preventing malicious code execution and maintaining performance is crucial
  - e.g., 90% of cyberattacks were performed via vulnerable IoT devices in 2021 (Liebermann, 2022)
  - Cryptography is foundational to security
  - Power consumption, energy efficiency, latency



MCU Example Applications

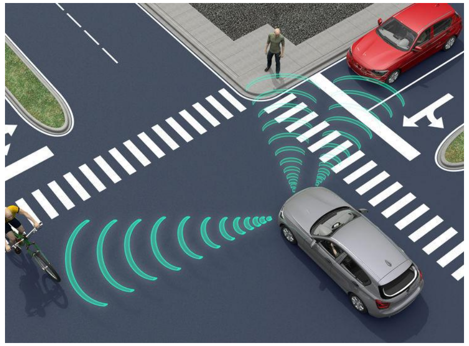
Security Camera

Thermostat



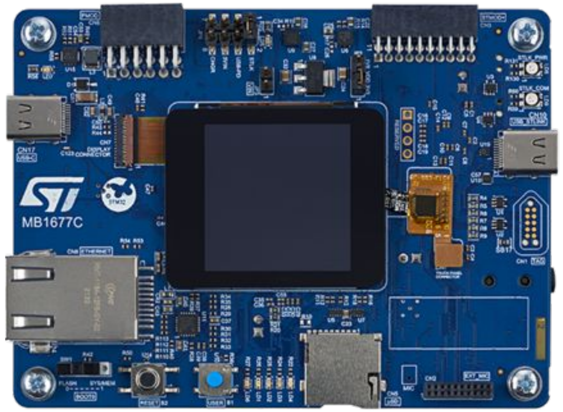
MCU

Medical Device



Autonomous Vehicle

MCU Criteria



1 256-bit encryption

- Highly secure and common encryption technique
- Support for sensitive data

2 CPU Performance

- High computational speed
- High power efficiency

3 Hardware-based Root of Trust (RoT)

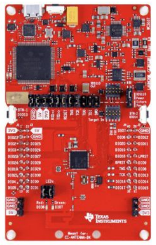
- Lays foundation for secure ops, defines secure chain of trust
- Inherently trusted, stores keys, immune from malware

4 Certificate-based software authentication

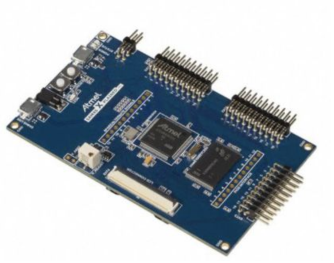
- Leverages cryptography to generate digital certificates
- Prevents the execution of malicious code

STM32H573 vs. other Platforms

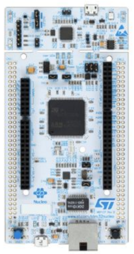
MCU	256-bit Hardware Support	High Performance CPU	Hardware RoT	Certificate-Based Authentication
TI MSP-430FR59xx	Yes	No	No	No
Atmel SAM4S	No	Yes	No	No
STM32F767II (old)	No	Yes	No	No
STM32H573	*Yes*	*Yes*	*Yes*	*Yes*



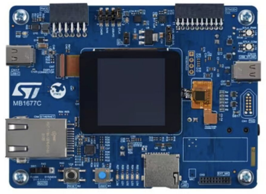
TI



Atmel



Old STM32F



New STM32H573

STM32 Toolchain

- **STM32CubeMX**
  - To configure and initialize new projects (pre-development)
- **STM32CubeProgrammer**
  - Facilitates flashing of STM32 MCUs
- **STM32CubeIDE**
  - For application development in C

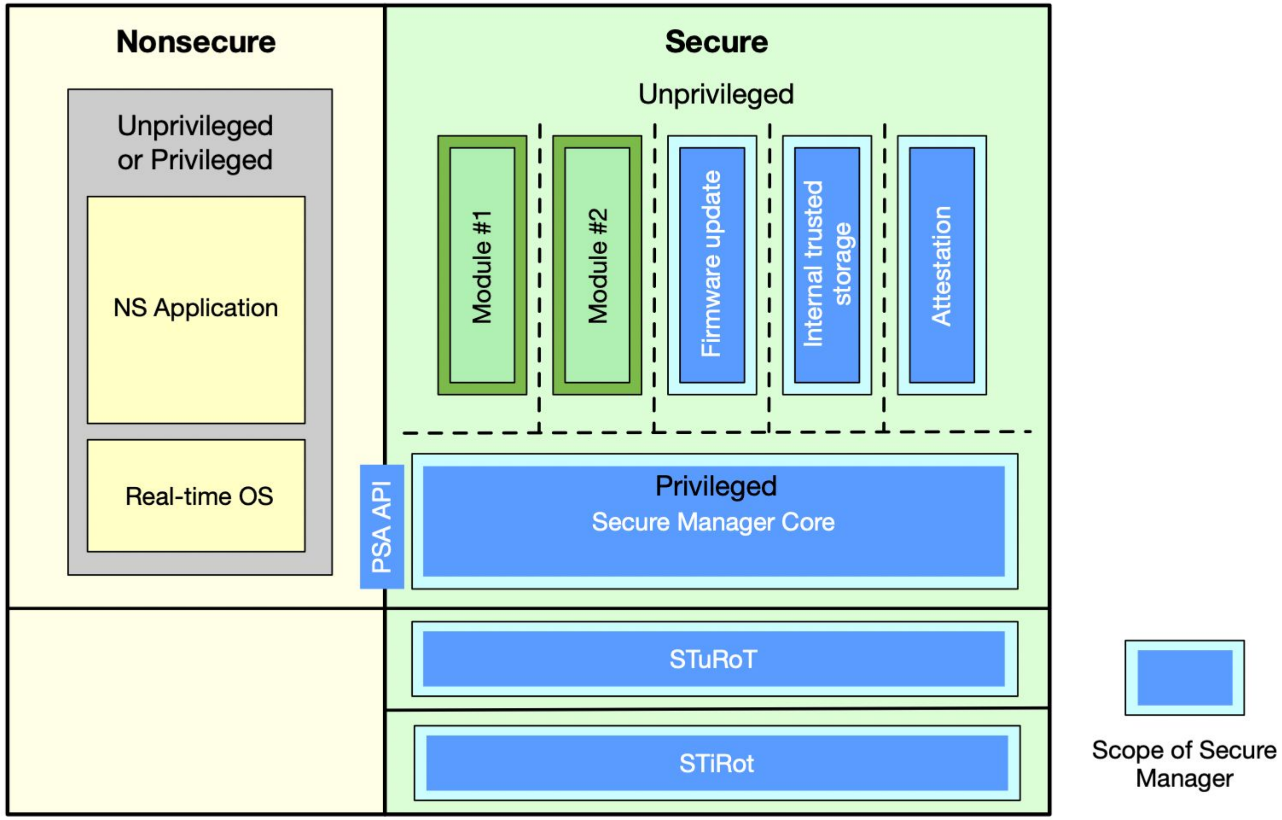
STM32H573 Features

STM32 Trust TEE

- **Industry leading solution**
    - Offers enhanced security, tamper detection, high performance CPU, cryptography support
- **Provides the Secure Manager**
    - Installable secure firmware
        - Part of first series to offer a “system-on-chip” security system
    - Provides ready-to-use secure services
        - e.g., attestation, encryption, trusted storage, isolation, etc.
    - ST’s custom implementation of the Trusted Execution Environment (TEE)
        - ARM Trusted Firmware-M (TF-M): open source implementation

Secure Manager

- **Leverages the ARM TrustZone**
  - Enter secure mode when enabled
- **3 main components**
  - Secure apps (top)
  - Secure Manager Core
  - 2-level RoT (bottom)
- **Use Platform Security Architecture (PSA) API calls to access Secure Services**



Evaluation Metrics

Performance

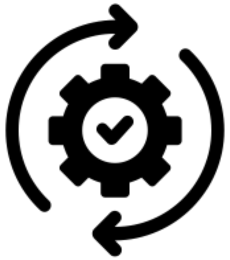
- Ensure Secure Manager does not *significantly* degrade overall performance

Security

- Only runs validated firmware
- Meets standards for security

Efficiency

- Cost efficiency
- Overhead, execution time, power consumption



Research Objectives

Evaluate feasibility of solution

- Determine if the power variation between nonsecure and secure modes is within an appropriate range

Estimate physical resources needed

- Gauge the current, energy, power, charge, etc. consumed by the STM32 as part of resources needed for a larger overall system

Enhance Secure Manager Performance

- Identify areas of improvement: conditions where the Secure Manager has a large impact on other aspects or uses excessive resources

Study Concerns

Ethical

- As attacks on IoT devices increase, MCU security is critical to ensuring safe and proper execution of a wide range of systems we interact with on a daily basis

Health & Safety

- These MCUs are typically always active, therefore they must adhere to the required and safe power/energy consumption levels

Economic + Env.

- These results can provide a benchmark for the resources required by the STM32, which can be allocated in advance, and leveraged to reduce energy consumption

Standards & Constraints

Testing App #1: GPIO

- Commonly used I/O to test basic functionality
- **Implementation:**
  - Blink all LEDs on the board 30 times
  - Delay of 25ms between on and off modes

Testing App #2: I^2C

- A standard protocol for communicating with peripherals (e.g., sensors)
- **Implementation:**
  - 4 bytes of sensor readings sent 1000 times over I^2C channel

Testing App #3: Encryption Algorithm

- ECDSA algorithm (256-bit key)
- Utilizes PSA (ARM Platform Security Architecture)
- **Implementation:**
  - Retrieves the key
  - Calculates the signature of fixed data

Overview

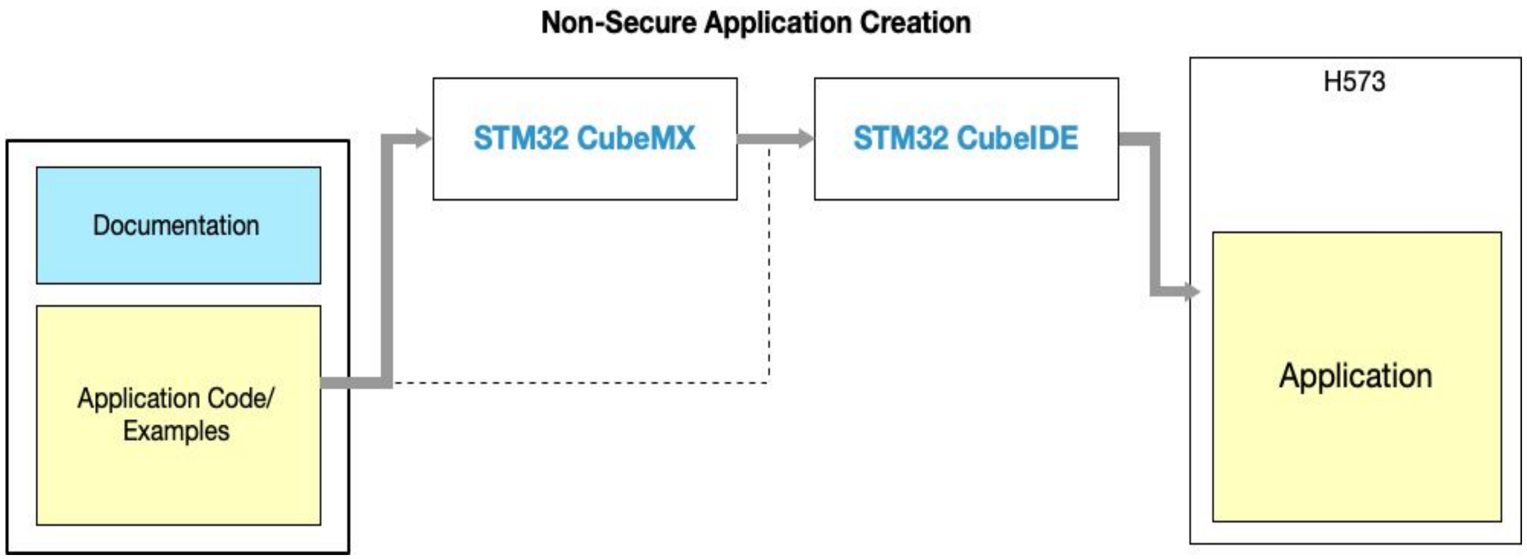
- Section 1: Project Background
- **Section 2: Design Process and Testing**
- Section 3: Results and Analysis
- Section 4: Future Work

Design Process

Configuration of code and testbench

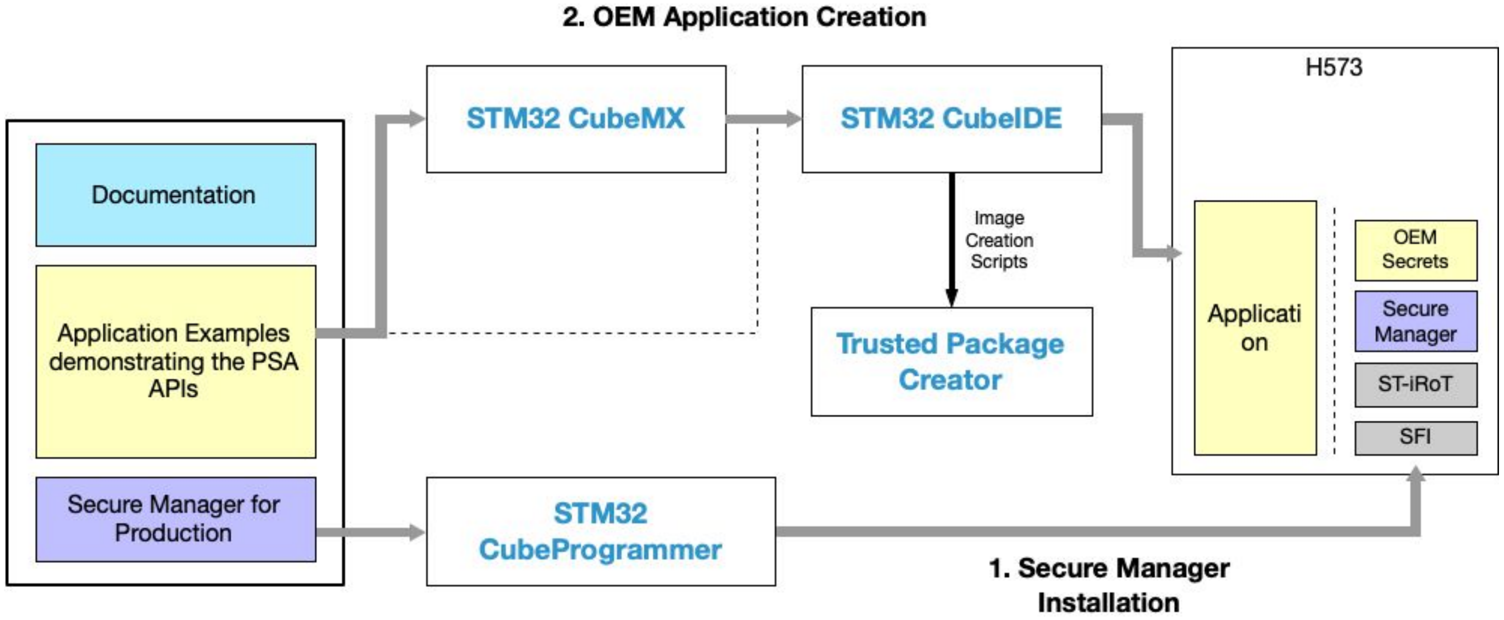
Non-Secure Development Process

Developing applications that *do not* use the Secure Manager or Services



Secure Development Process

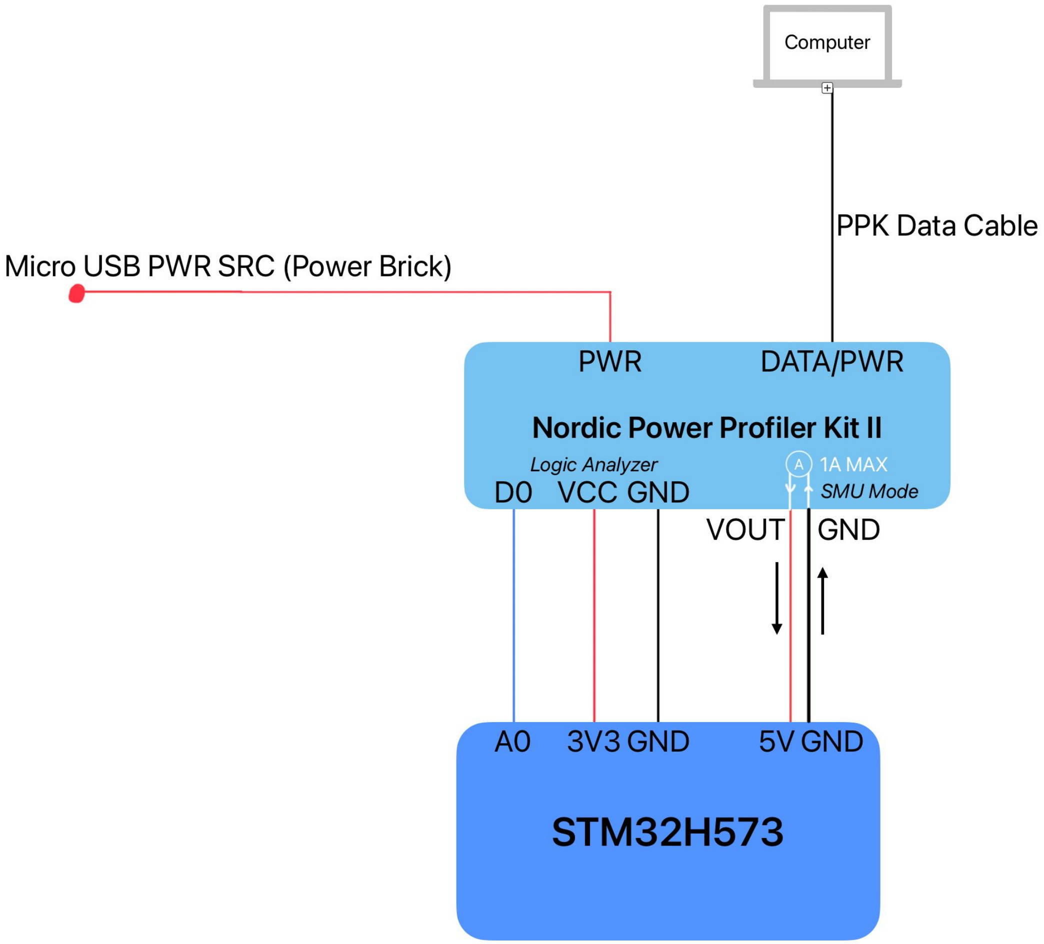
Developing applications that *use* the Secure Manager and/or Services



Testbed Architecture

STM32H573 powered by PPK II

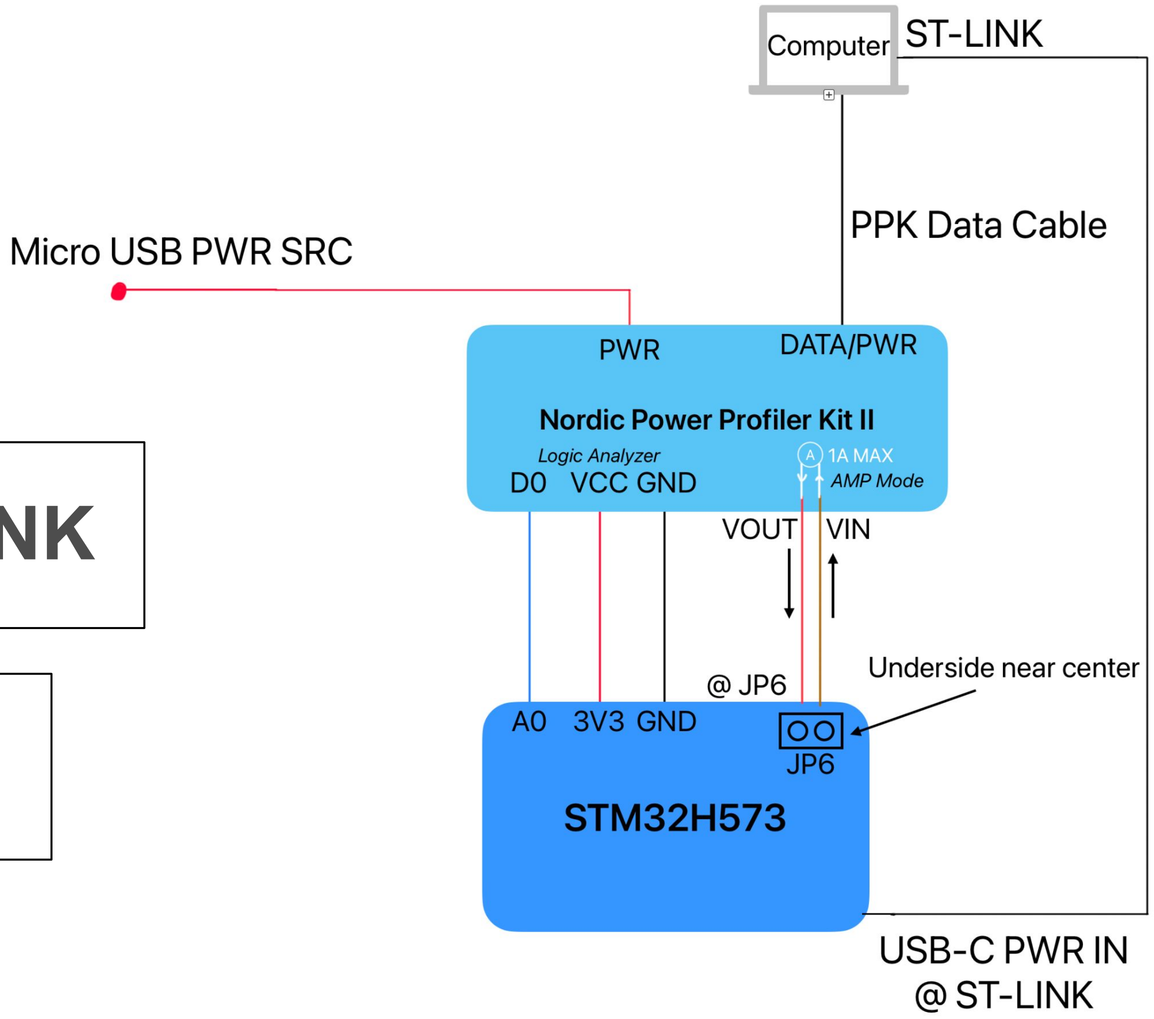
PPK II acting as voltmeter



Testbed Architecture (continued)

STM32H573 powered by ST-LINK

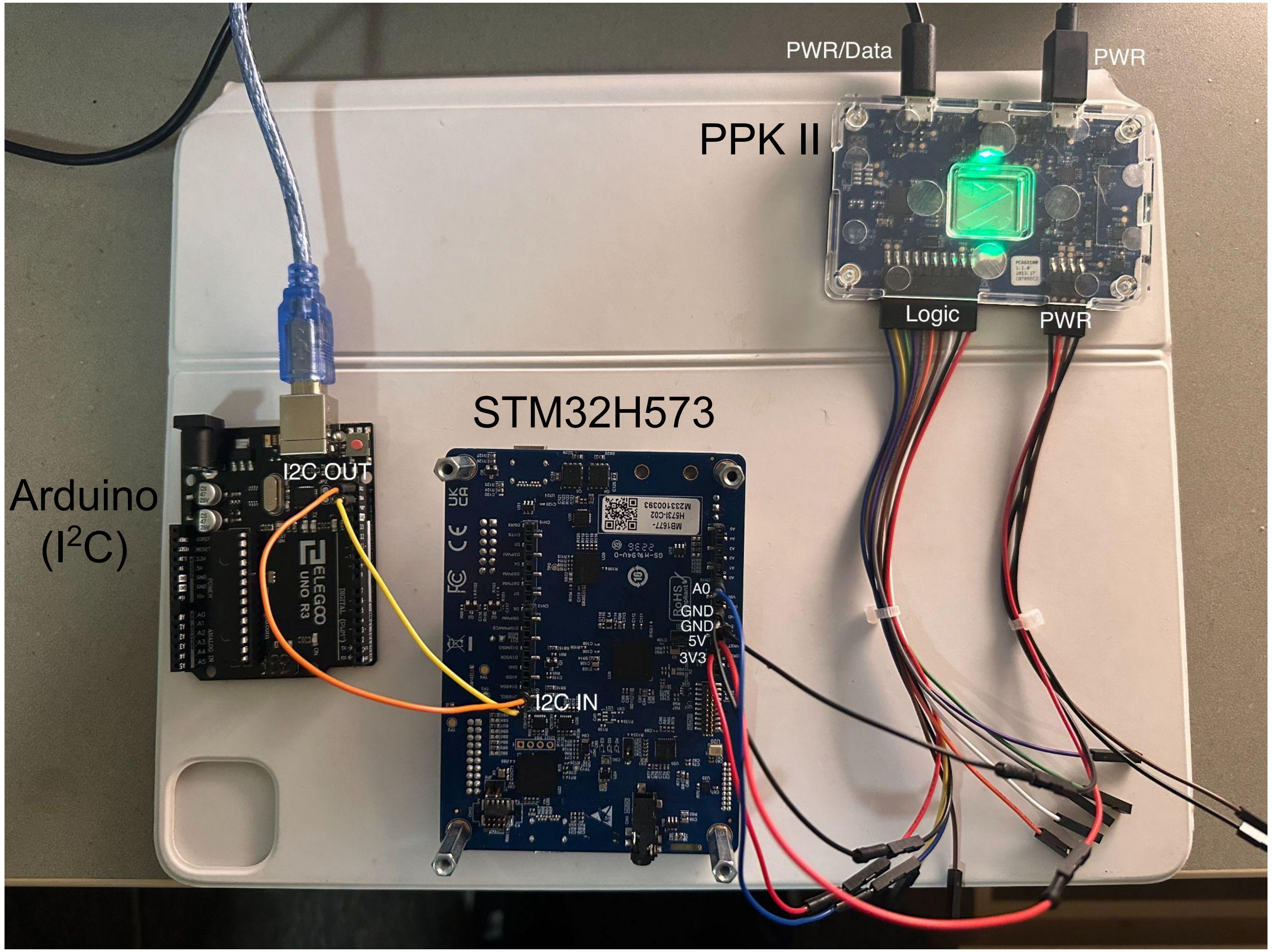
PPK II acting as ammeter



In Practice

Example Setup for I²C testing

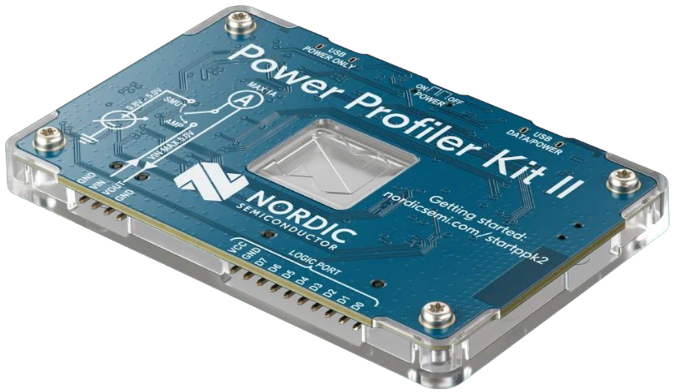
- Arduino as slave device
- I²C communication through SCL and SDA pins on each board



Testing

Data Collection Methodology

Power Profiler Kit (PPK)



- **Performance, logic, and power analysis**
- **Acts as a power source for device under test**
- **Integrates with software**
- **Can take up to 100,000 samples/sec**
- **Operates in source and current modes**
    - Provides up to 5V at 1A

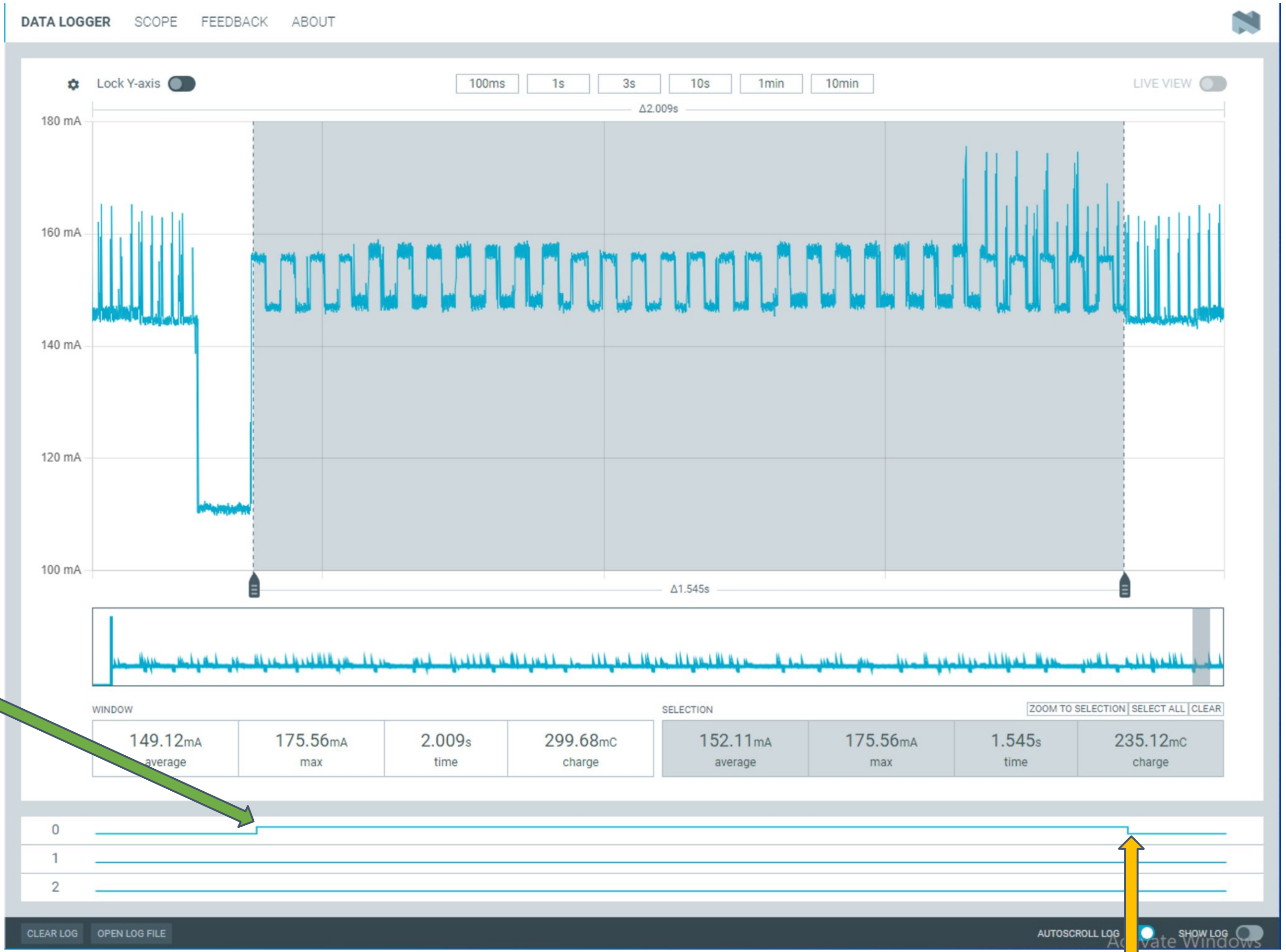
Measuring w/ PPK

```
long sampleCount = 0;
long sampleThreshold = 1000; //1k samples
startTiming();
while(1); {
      float distance; //written via memory
      uint8_t distanceBytes[4]; //32 bits total
      // sent 1000 times → 4000 bytes

      // Request 4 bytes of data from slave device (Arduino)
      if (HAL_I2C_Master_Receive(&hi2c1,
                (uint16_t) (ARDUINO_I2C_ADDRESS << 1, distanceBytes,
                sizeof(distanceBytes), HAL_MAX_DELAY) == HAL_OK) {
          // Convert received bytes back to float
          memcpy(&distance, distanceBytes, sizeof(distance));
          sampleCount++;
      } else {
          Error_Handler();
      }

      if (sampleCount > sampleThreshold) {
          stopTiming();
          break;
      }

      HAL_Delay(1); // Poll every 1 ms
}
```



Testing Methodology

Clock Frequency

- Fix CPU clock frequency to 250 MHz
- Eliminate impact of clock frequency on data

Metrics

- Monitor application start to completion time
- Time, average current drawn, charge

Calculation

- Use metrics from PPK to calculate energy consumption
- Focus on energy instead of power

Repetition

- Run 30 trials for each application
- Calculate averages to compare against other applications

Overview

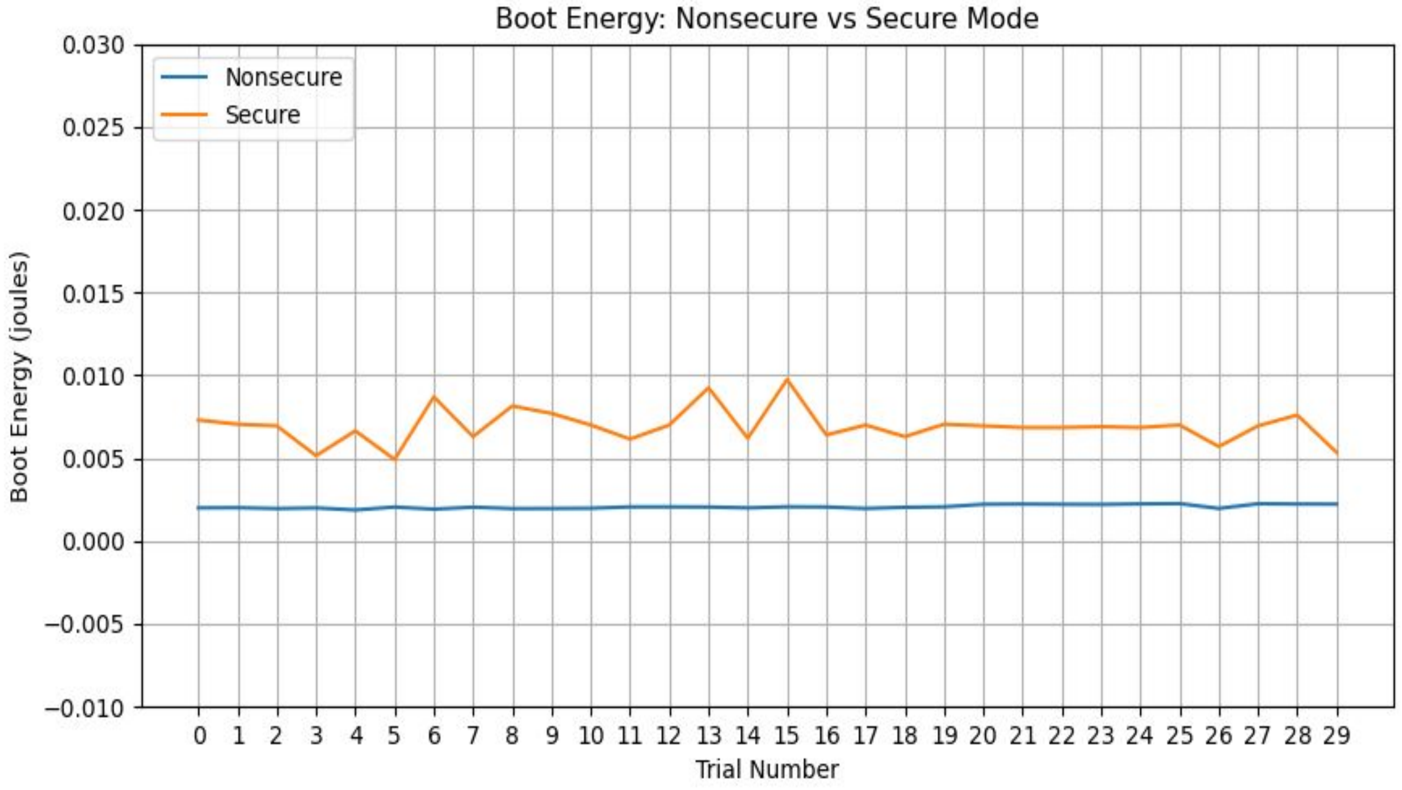
- Section 1: Project Background
- Section 2: Design Process and Testing
- **Section 3: Results and Analysis**
- Section 4: Future Work

Results and Analysis

Use-case recommendations based on data

Boot Results

- **~500% longer runtime for Secure boot**
- **~44% more current drawn for Secure boot**
- **Energy consumption 200% more for Secure boot versus Non-Secure boot**



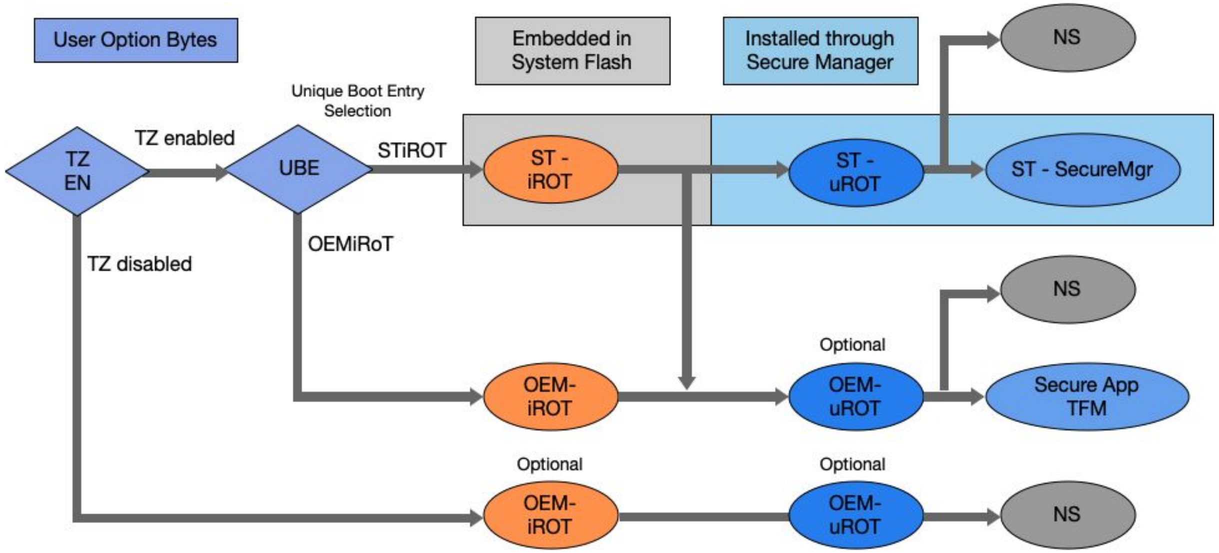
Boot Analysis

Secure

- Boot path with TrustZone **enabled**
- STiRoT, STuRoT and Secure Manager installation

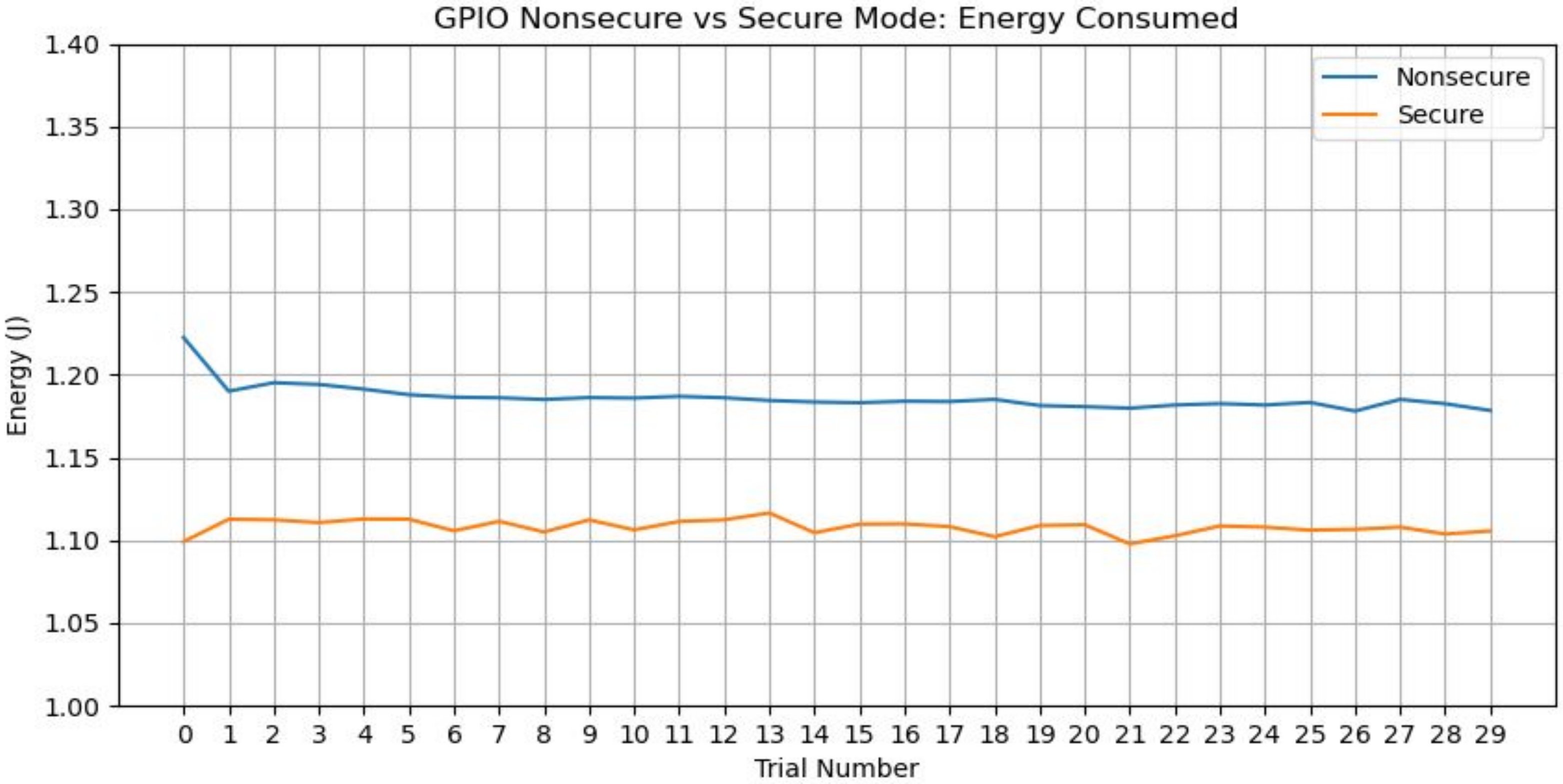
Non-Secure

- Boot path with TrustZone **disabled**
- No extra steps



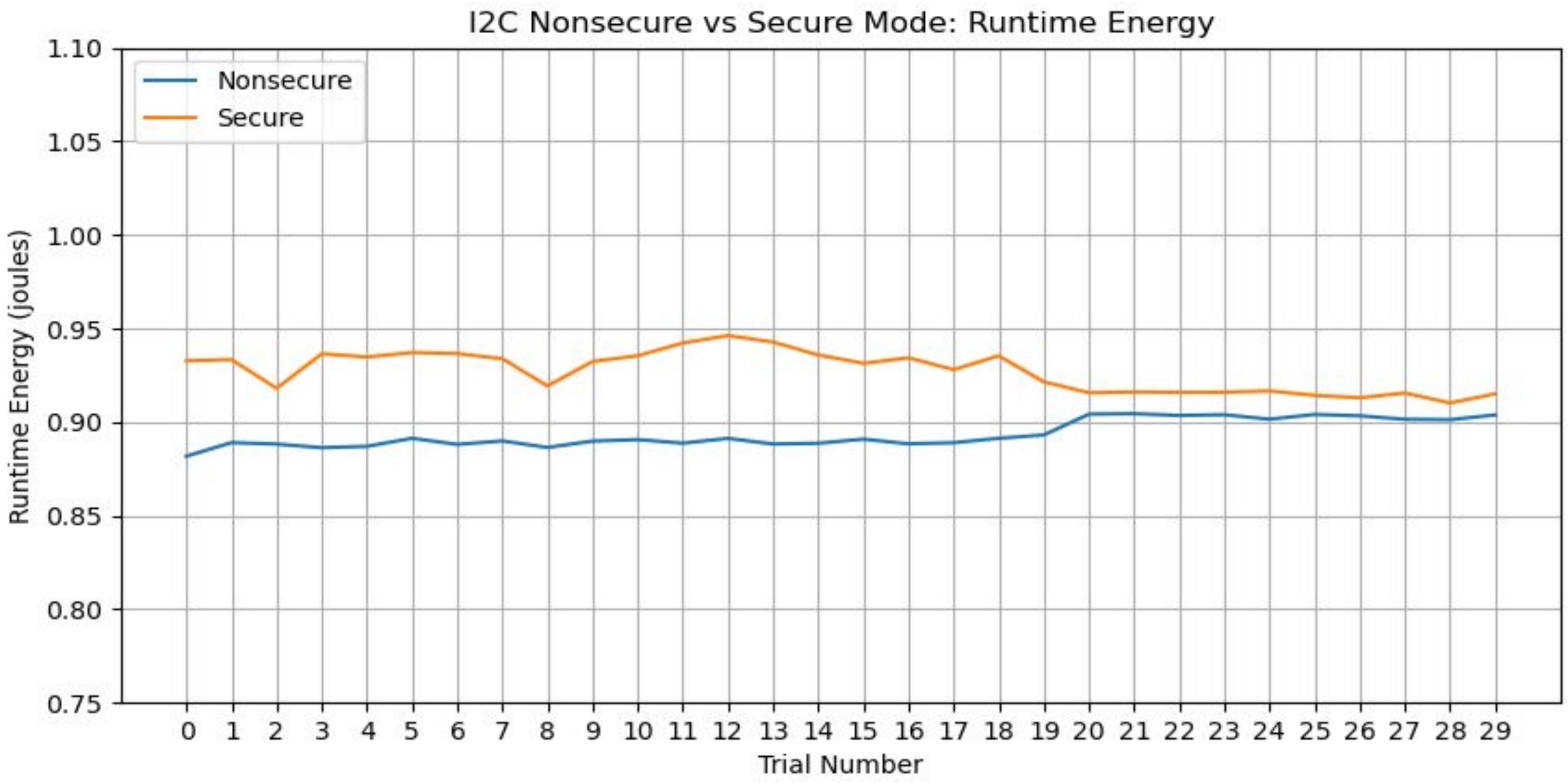
GPIO Results

- ~0.1% longer runtime for Non-Secure application
- ~6% more current drawn for Non-Secure application
- Energy consumption 6% more for Non-Secure versus Secure application



I²C Results

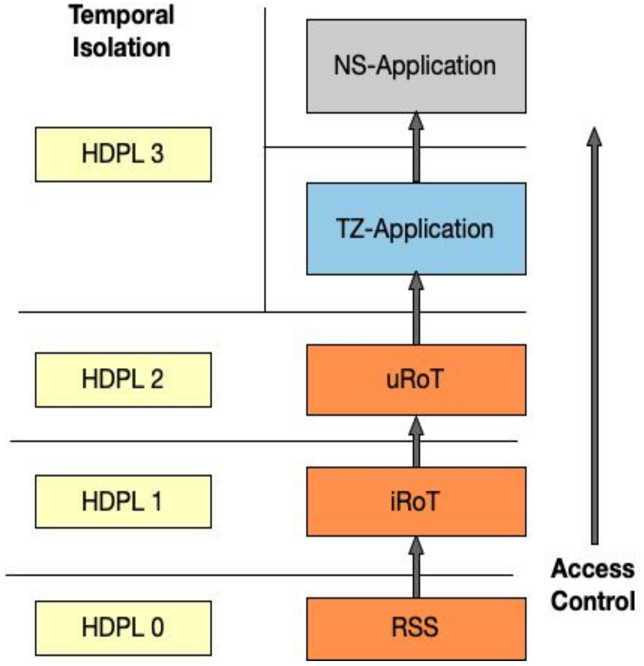
- **~1%** longer runtime for Secure application
- **~5%** more current drawn for Secure application
- Energy consumption **4%** more for Secure versus Non-Secure application



Purely Secure vs. Purely Non-Secure Analysis

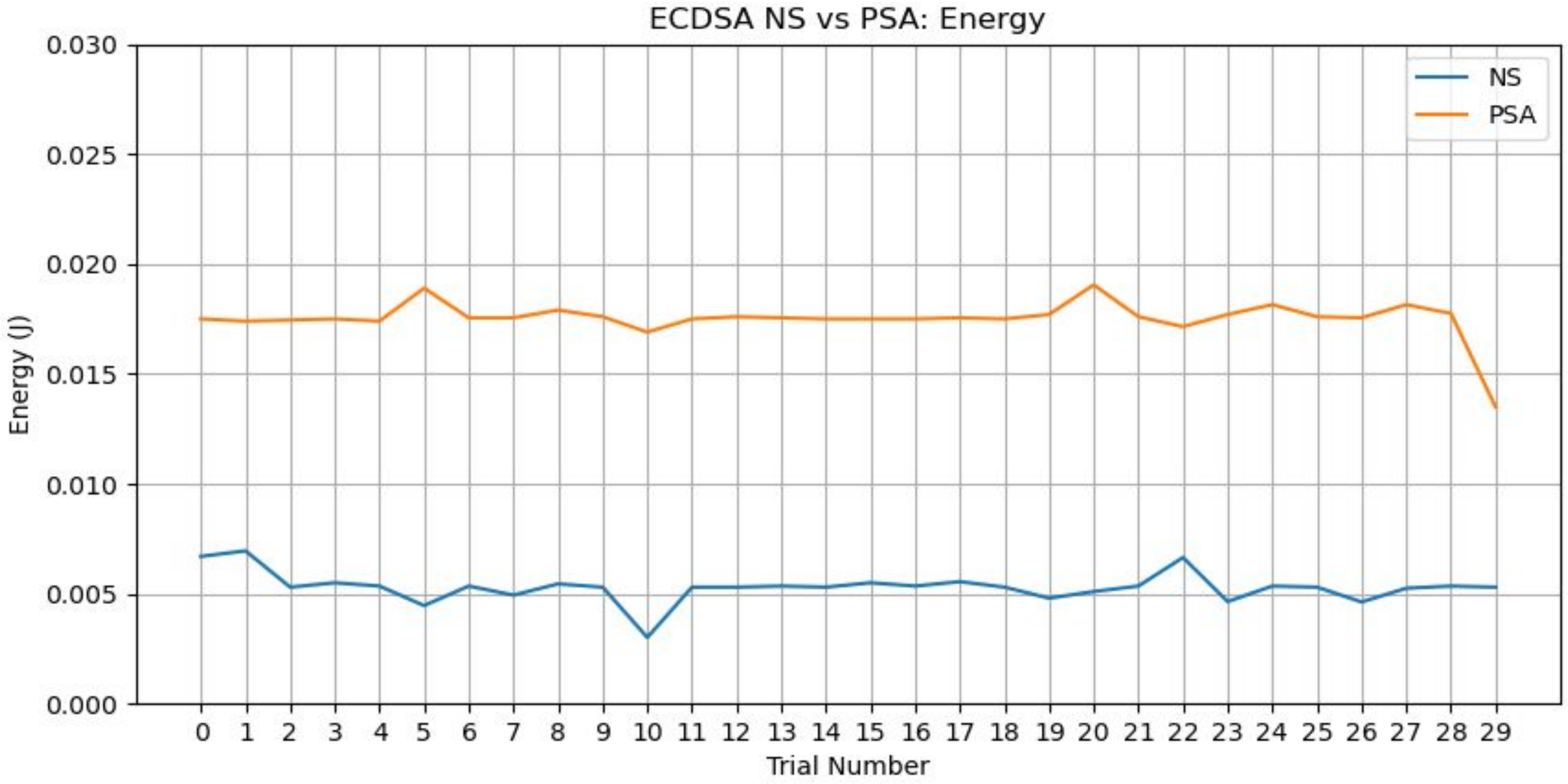
GPIO and I²C

- Secure application → Secure Manager running in the background, always checking for security violations, fully sandboxed
  - Minimal increase in energy consumption
- Non-secure application → no Secure Manager, access to all resources, no checks



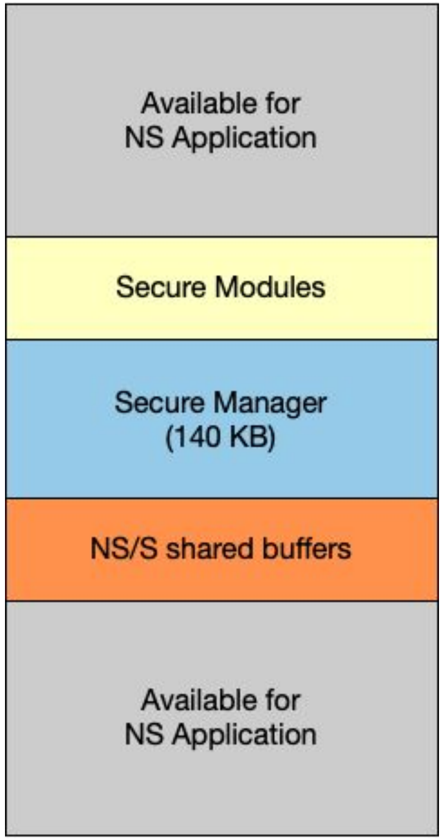
ECDSA Results

- **10x** longer runtime for PSA application
- **~75%** less current drawn by PSA application
- Energy consumption **~230%** more for PSA versus purely NS application



Secure Service Analysis

- Significant energy consumption increase
- Interrupt to switch modes for PSA call
  - NS → S → NS
- Shared buffer for NS/S data, change control of SRAM area
- Tamper-resistant hardware unique keys for cryptographic services with Secure boot



SRAM memory mapping

Summary of Results

	Best Performance	Lowest Current	Least Energy
Boot	Non-Secure	Non-Secure	Non-Secure
GPIO	Equivalent*	Secure	Secure
I^2C	Equivalent*	Non-Secure	Non-Secure
ECDSA	Non-Secure	Secure	Non-Secure

*Minimal but not negligible difference

Use Case Recommendations

Most Important Metric	Recommendation
Execution Time	Either*
Energy	Either*
Boot Time	Purely Non-Secure
Security	Purely Secure/PSA Services

*Purely non-secure and purely secure,
Minimal but not negligible difference

Overview

- Section 1: Project Background
- Section 2: Design Process and Testing
- Section 3: Results and Analysis
- **Section 4: Future Work**

Future Work

Where we would go from here…

Extensions & Optimizations

- **Testing more secure services beyond cryptography**
  - e.g., attestation, internal trusted storage
- **Implementing and testing multi-threaded application**
  - This project only tested single-threaded applications
  - Will examine the energy and power consumption variation with more context switching due to threading
- **Optimizations for the Secure Manager**
  - Will explore the internal mechanisms and where performance may be improved accordingly

Special thanks to STMicroelectronics

life.augmented

Thank You

Questions?

References

1. N. Liebermann, “2021 IoT Security Landscape - SAM Seamless Network,” SAM, Apr. 07, 2022.
2. ST Microelectronics. Getting started with STiROT (ST immutable Root Of Trust) for STM32H5 MCUs. ST Microelectronics, Sept. 2023.
3. ST Microelectronics. Secure Manager for STM32H5 - STMicroelectronics. STMicroelectronics, 2023.
4. ST Microelectronics. STM32H563/573. ST Microelectronics, 2023.
5. Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. “Trusted Execution Environment: What It is, and What It is Not”. In: 2015 IEEE Trustcom/BigDataSE/ISPA (Aug. 2015).
6. SeongHan Shin et al. “An Investigation of PSA Certified”. In: Proceedings of the 17th International Conference on Availability, Reliability and Security (Aug. 2022).

Appendix

Additional Content and Data

Testing Applications

Boot

- Time before application start
- Power on → beginning of executing our code

GPIO

- Blink all LEDs on the board 30 times
- Delay of 25 ms between on and off

I^2C

- Send 4000 bytes of sensor readings
- 4 bytes sent 1000 times over I^2C channel

ECDSA

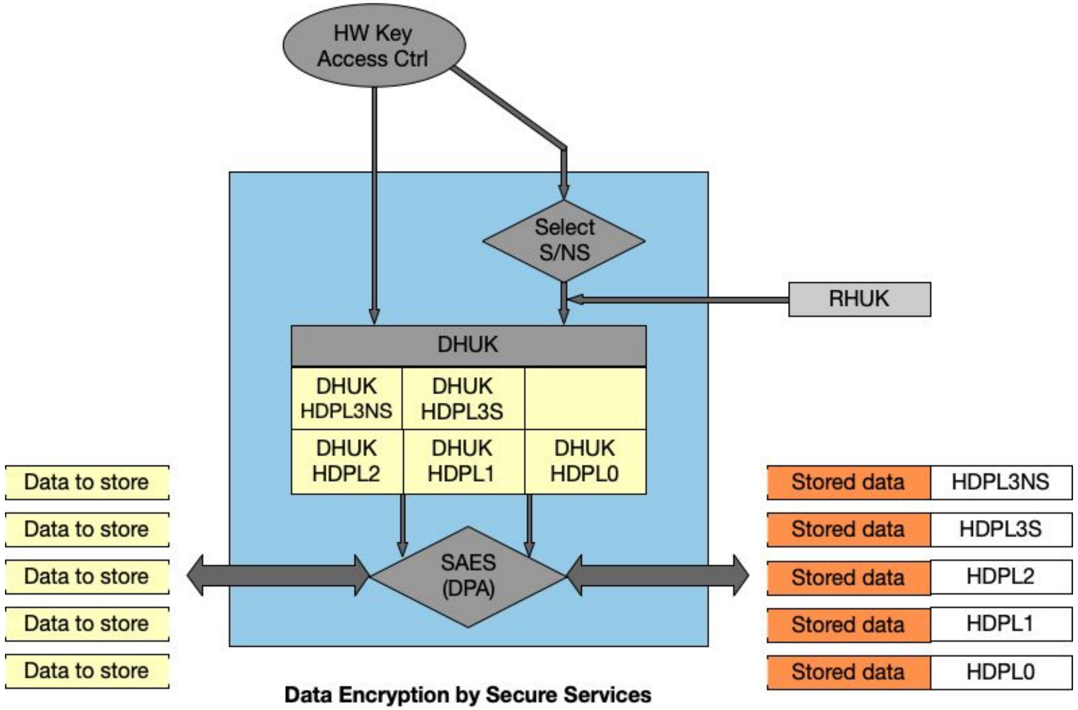
- Retrieves the key
- Calculate the signature of fixed data

Secure Boot - Product States

Open - Completely open for debugging and flashing
TZ-Closed - NS application open for debugging; Secure closed
Closed - NS and Secure domains are closed
Locked - NS and Secure domains closed (No reopening)

- **Non-secure boot only allows the product in an OPEN state**
- **Secure boot requires specification of other product states**

Cryptography Analysis



Data Encryption by Secure Services

- Tamper-resistant hardware for cryptographic services with Secure Manager
- Derived Hardware Unique Key (DHUK) per level
- Board-specific Root Hardware Unique Key (RHUK)

Energy Consumption Calculation

E: Energy
V: Voltage
Q: Charge
I: Current
t: Time

$E = V * I * t$

$I = dQ/dt$ (change over time)

$I = Q/t \rightarrow Q = I*t$

$\boxed{\text{Energy} = V*Q}$

Results and Analysis: Boot Results

Statistical distribution of boot energy consumed (in joules):

	Nonsecure: Boot Energy	Secure: Boot Energy
Mean	0.002073	0.006932
Median	0.002052	0.006950
St. Dev	0.000115	0.001062

Results and Analysis: GPIO Results

Statistical distribution of boot energy consumed (in joules):

	Nonsecure: Energy	Secure: Energy
Mean	1.186150	1.108098
Median	1.184800	1.108500
St. Dev	0.007953	0.004312

Results and Analysis: I2C Results

Statistical distribution of runtime energy consumed (in joules):

	Nonsecure: Runtime Energy	Secure: Runtime Energy
Mean	0.893637	0.927172
Median	0.890725	0.931850
St. Dev	0.007148	0.010702

Results and Analysis: ECDSA Results

Statistical distribution of energy consumed (in joules):

	NS: Energy	PSA: Energy
Mean	0.005299	0.017525
Median	0.005300	0.017550
St. Dev	0.000693	0.000872

Standards & Constraints

- **Testing Application #1: GPIO**
  - Commonly used I/O to test basic functionality
  - Implementation: Blink all LEDs on the board 30 times & delay of 25 ms between on and off
- **Testing Application #2: I^2C**
  - A standard protocol for communicating with peripherals (e.g., sensors)
  - Implementation: Send 4000 bytes of sensor readings & 4 bytes sent 1000 times over I^2C channel
- **Testing Application #3: Encryption Algorithm**
  - ECDSA algorithm (256-bit key)
  - Utilizing PSA (ARM Platform Security Architecture)
    - A hardware security standard from ARM for establishing uniform core security infrastructure
    - PSA Developer APIs empower developers to leverage hardware security features, e.g., cryptography, secure storage, and attestation
  - Implementation: Retrieves the key & calculates the signature of fixed data

Testing Applications

Boot

- Time before application start
- Power on → beginning of executing our code

GPIO

- Blink all LEDs on the board 30 times
- Delay of 25 ms between on and off

I^2C

- Send 4000 bytes of sensor readings
- 4 bytes sent 1000 times over I^2C channel

ECDSA

- Retrieves the key
- Calculate the signature of fixed data