ORSHIN

D5.3

Intra-device and inter-device secure communication prototypes

Project number	101070008
Project acronym	ORSHIN
Project title	Open-source resilient Hardware and software for Internet of things
Start date of the project	1st October, 2022
Duration	36 months
Call	HORIZON-CL3-2021-CS-01

Deliverable type	DEM – Demonstrator, pilot, prototype
Deliverable reference number	CL3-2021-CS-01/ D5.3/ 1.0
Work package contributing to the deliverable	WP5
Due date	Jun 2025 – M33
Actual submission date	30th June 2025

Responsible organisation	SEC
Editor	SEC
Dissemination level	PU - public
Revision	1.0

Abstract	This document represents the user guide needed to setup and test the intra- and inter-devices communication prototypes. The intra-devices communication prototype is aimed at showcasing the NSCP protocol, an improved version of the SCP03 protocol, used to allow a secure communication between a client and a Secure Element. The inter-devices communication prototype is aimed at showcasing the BlueBrothers, two novel protocols replacing the standard Bluetooth pairing and session establishment, used to allow a secure communication between two devices via Bluetooth (Classic and Low Energy).
Keywords	Demonstrator, intra-devices communication, inter-devices communication, user guide

Funded by the European Union under grant agreement no. 101070008. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

Editor
SEC

Contributors (ordered according to beneficiary numbers)
Sacchetti, Tommaso (ECM)
Gorla, Federico (SEC)
Battistello, Alberto (SEC)
Molteni, Maria Chiara (SEC)

Reviewers
Volodymyr Bezsmertnyi (NXP)
Olivier Thomas (TXP)

Executive Summary

The ORSHIN project aims to develop secure, open-source hardware with secure connectivity.

In Task 5.4 we designed an original and practical protocol that ensures security and privacy (S&P) for **intra-device communication**. To achieve this, we leveraged open-source hardware, implementing the prototype on both an ARM-based STM32 microcontroller and a RISC-V microcontroller.

The prototype uses the New Secure Channel Protocol (NSCP) to enable secure communication between a client and a Secure Element. NSCP is a novel protocol specifically designed for industrial IoT environments, aiming to strengthen the security of connections between microcontrollers and Secure Elements while offering improved efficiency over SCP03, the current industry standard. By utilizing the Xoodyak cryptographic primitive, NSCP provides strong security with significantly reduced computational overhead, making it well-suited for resource-constrained devices.

In Task 5.2 we designed an original and practical protocol that ensures security and privacy (S&P) for **inter-device communication**. To achieve this, we implemented the prototype on both an ARM-based nRF52 microcontroller and a Raspberry Pi.

The prototype uses the BlueBrothers protocols to enable secure communication between two devices via Bluetooth (Classic and Low Energy). BlueBrothers are two novel protocols replacing the standard Bluetooth pairing and session establishment. They rely on strong and performant cryptographic primitives such as ASCON and C25519, enabling stronger security guarantees with reduced or minimal computational overhead, making them suitable for resource-constrained devices.

This document serves as a user guide for setting up and testing these prototypes, developed by exploiting the capabilities of open-source hardware.

Table of Content

List of Figures

Chapter 1 Introduction

The ORSHIN project studies resilient, open-source hardware and software solutions for the Internet of Things (IoT).

Within embedded devices, components typically communicate via low-level buses such as I2C and SPI. These intra-device communications are often left unprotected or rely on ad-hoc, frequently proprietary, security mechanisms. The rise of open hardware offers increased transparency into these internal interactions, presenting a valuable opportunity to enhance security and privacy.

To enhance the security and privacy of intra-device communication in embedded systems, we developed a robust and practical protocol called New Secure Communication Protocol (NSCP), by leveraging the capabilities of open-source technologies. it is an alternative to SCP03 consisting of a lightweight secure channel protocol that utilizes Xoodyak [1], a cryptographic primitive known for its efficiency and minimal resource requirements. The new protocol aims to simplify the operational framework to provide adequate security while maximizing throughput. For detailed formal definitions, please refer to the ORSHIN deliverable D5.2.

In *Chapter 2* we report a user guide needed to setup and test the intra-devices communication prototype. The prototype is aimed at showcasing the NSCP protocol. This work was done by exploiting the potential of open-source hardware.

Device-to-device communication in constrained environments, such as those found in IoT and IIoT systems, often lacks adequate security and privacy guarantees for various reasons. However, the rise of open-source hardware is changing this landscape by giving access to more advanced and transparent hardware components, enabling the integration of essential security features directly into embedded device designs.

To improve the security and privacy of inter-device communication in embedded systems, we developed two novel protocols, the BlueBrothers, as alternatives to standard Bluetooth pairing and session establishment. These protocols leverage efficient and robust cryptographic primitives, including ASCON and C25519, providing enhanced security guarantees while maintaining low computational overhead, making them well-suited for resource-constrained devices.

In Chapter 3 we report a user guide needed to setup and test the inter-devices communication prototype. The prototype is aimed at showcasing the BlueBrothers protocols. This work was done by exploiting the potential of open-source hardware.

In Chapter 4 we report our conclusion on the work; Chapter 5 and 6 are respectively the List of abbreviations and the Bibliography.

Chapter 2 User guide for secure intra-device communication prototype

2.1 Setup

The setup is split between two components that will communicate over the I2C channel: the Nexys A7 FPGA, the secure element used as a *slave*, and a Raspberry Pi 4B+ functioning as a *master.*

Both of these boards will run the code responsible for the implementation of the NSCP protocol over I2C for the slave and master part, respectively.

The setup consists of the following main Hardware components:

- Nexys A7 FPGA Board [2]
- JTAG-HS2 Debugger [3]
- Raspberry Pi 4B+ [4] [5]

Furthermore, the complete sources needed to test the system are given in the following repos:

- **orshin-rpi_scp03**
  NSCP and SCP03 Master (running on the RPI4) – [6]
  - The repo contains two branches
    - ***"main"*** has the SCP03 implementation
    - ***"nscp"*** has the NSCP implementation

- **orshin-corev-secure-element**
  NSCP and SCP03 Slave (running on the Nexys FPGA) – [7]
  - The repo contains two branches
    - "***main***" has the SCP03 implementation
    - "***nscp***" has the NSCP implementation

2.2 Prerequisites

The Nexys A7 needs to be configured following the official instructions [8].

The process involves loading a bitstream implementing the CoreV architecture through the board USB port (Step 1, Option 1 in [8]). The bitstream used in this project can be found in [9]. The document also explains how to set up and use the HS2 debugger.

After programming the FPGA, it is useful to check if everything is working correctly by running the example program described at Step 2 in [8].

Finally, to compile and run the sources found in the NSCP repos, it is necessary to configure an installation of Eclipse as described in [10]. The guide explains how to set up the IDE with the right compiler for the CoreV architecture with which it is possible to compile and run the test program from sources. With Eclipse configured this way, it will be possible to import the orshin-corev-secure-element which contains the implementation of the Secure Element and SCP03/NSCP. How to use the repos is described in sections 2 and 3.

The bistream implements the CoreV-MCU architecture. A user manual describing its inner workings can be found in [11].

2.3 Connections

The Raspberry Pi 4 exposes a I2C master over GPIO 3 (SCL), GPIO 2 (SDA) and PIN 9 (Ground).

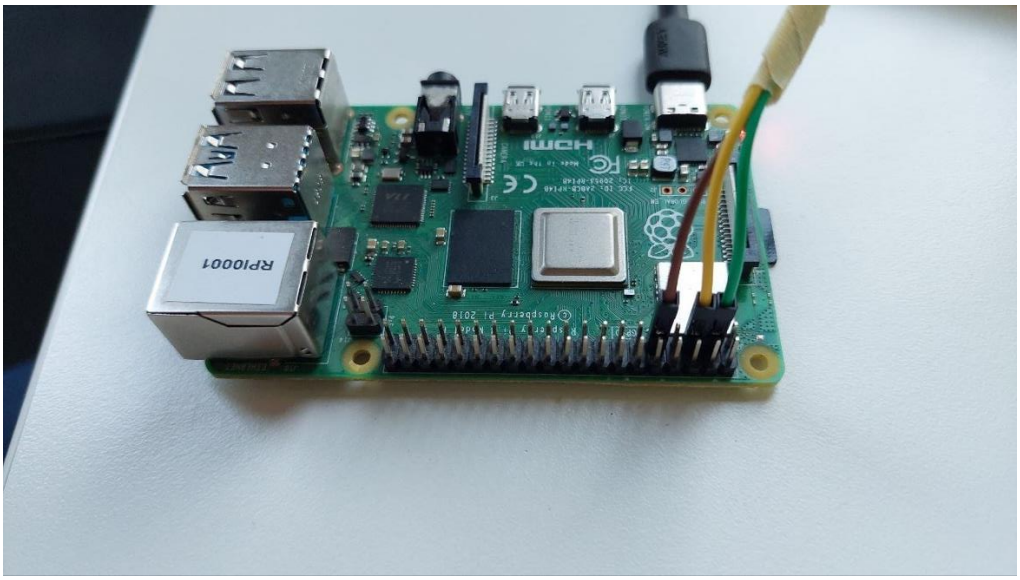

Figure 1 Raspberry Pi 4 connections

In Figure 1, the cable colors represent the various lines as described below:

- Brown: Ground
- Yellow: Clock
- Green: Data

The relevant pins on the Nexys A7 are shown in Figure 2.

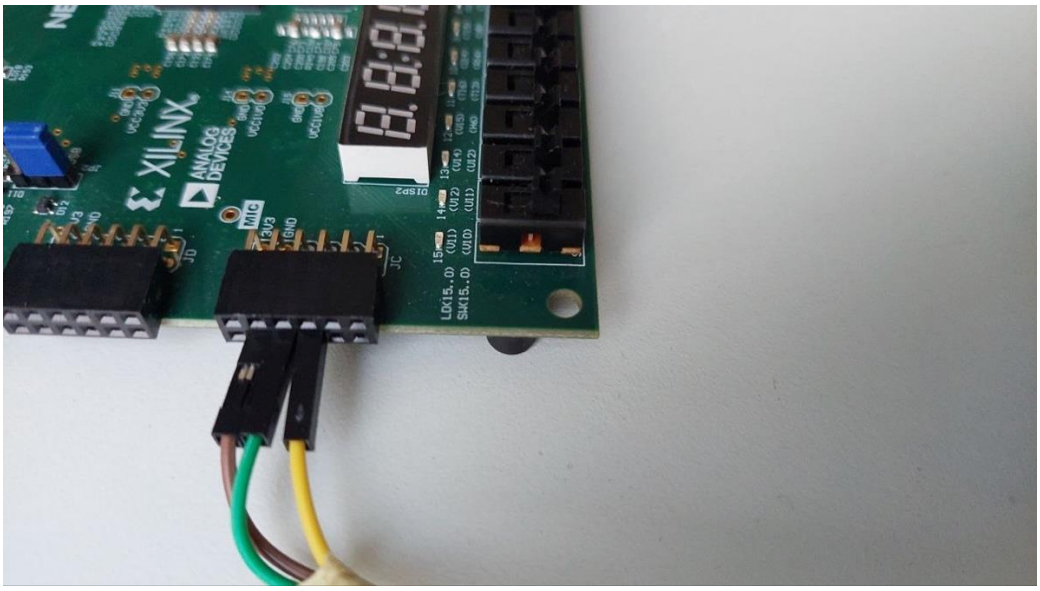

Figure 2 Pins on Nexys A7

The pins are connected to the JC Pmod expansion that can be found in the lower left corner of the board.

Finally, the debugger is connected to the lower row of the JB Pmod expansion as in shown inFigure 3.

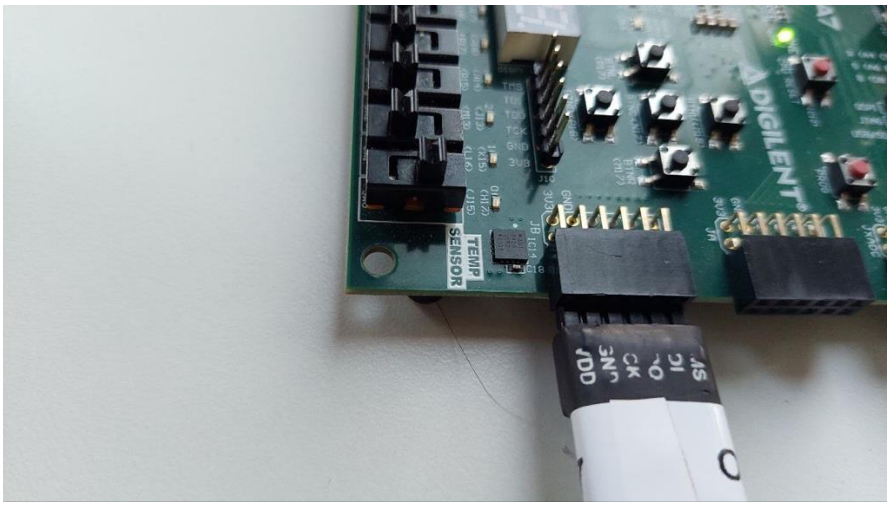

Figure 3 Connection of the debugger.

It is important to note that the last switch (J15) needs to be in the upper position for the debugger to correctly work.

In Figure 4is shown a scheme of the complete setup.

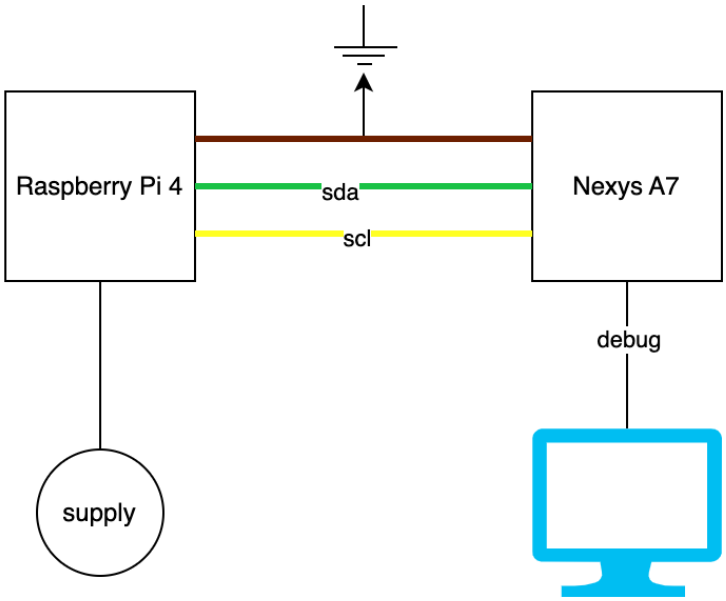

Figure 4 Complete setup.

2.4 Secure Element implementation

The code for the Secure Element can be found in report in the Github repository [7].

The repository consists of two branches: “main” contains the implementation of the SCP03 protocol whereas “nscp” contains the implementation for NSCP.

To test the project, it is necessary to import it into a correctly configured Eclipse IDE and subsequently build the “cli_test” subproject. With a debugger connected, it will be possible to stop and examine the execution as one normally would in an Eclipse based IDE (eg. ST Cube IDE).

The main.c file contains the code that creates the only FreeRTOS task present, namely "secpat_i2c_receive", which leverages the I2C channel to send and receive data to and from a master. In *i2c_secpat/secpat.c* there's the code that loops over the received bytes, one by one, and populates a RX buffer named "rx_packet". When the parsing is complete, the code populates a "tx_packet" buffer with the computed answer to be sent to the master.

By checking out the *main* or *nscp* branch, it will be possible to choose which implementation of the protocol to run. The master repo will have the same organization and the same branch needs to be checked out on both ends in order to have a working system.

The program prints its output on the serial port exposed by the USB connected to the Nexys board. When correctly loaded, the program will print the strings in Figure 5before waiting for data to be sent over I2C.

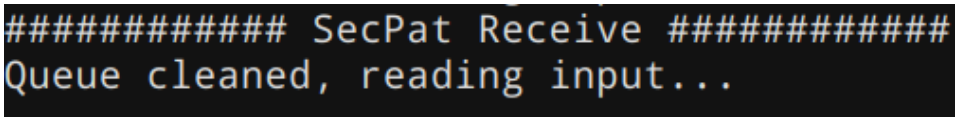

Figure 5 Program print

2.5 Master implementation

The code for the master will run onto a Raspberry Pi4 and can be found in the Github repository in [6].

The repository consists of two branches: "main" contains the implementation of the SCP03 protocol whereas "nscp" contains the implementation for NSCP.

After choosing the appropriate branch to checkout, to compile and run the program, the following steps need to be followed:

1. Enter the test program directory

   ```
   cd nano-package-xoodyak/examples/se05x_crypto/linux/
   ```

2. Create the build directory:

   ```
   mkdir build && cd build
   ```

3. Configure the project

   ```
   cmake ../ -DPLUGANDTRUST_SCP03=ON -DPLUGANDTRUST_DEBUG_LOGS=ON
   ```

4. Compile

   ```
   make
   ```

5. Run the executable

   ```
   ./ex_se05x_crypto
   ```

If everything worked as expected, the output should be similar to the one in Figure 6.

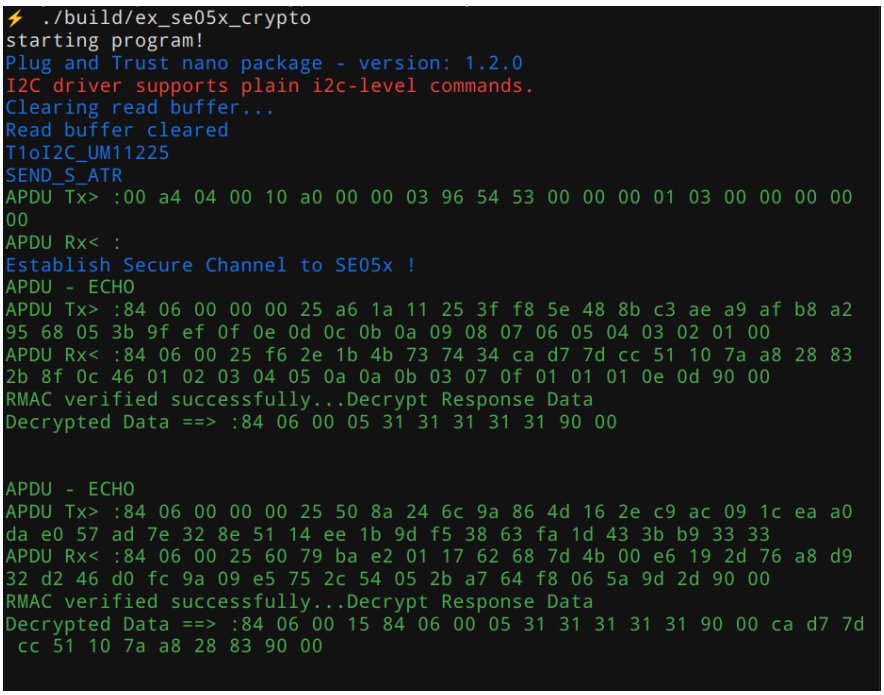

Figure 6 Output of the program

From Figure 6it is possible to see the output of the test program as it establishes a secure channel and then proceeds to send encrypted messages of increasing sizes. The protocol implements the ECHO command which sends and expects to receive the same unencrypted data.

Chapter 3 User guide for secure inter-devices communication prototype

3.1 Setup

The resources needed to test the communication are in the **bb-protocols** repository, whose README files extensively cover installation and usage.

3.1.1 Bluetooth Classic

The Bluetooth Classic (BC) [12] setup involves two Raspberry Pi 4B+ that will communicate over BC. One of the two devices is the Central, which initiates the connection, and the other is the Peripheral.

The BC implementation comes in the form of a C library used by two applications, one for the Central and one for the Peripheral. It is located in the **bb-portable** folder of the repository.

3.1.2 Bluetooth Low Energy

The Bluetooth Low Energy (BLE) [12] setup involves two nRF52 [14] devices that will communicate over BLE; the roles are the same, i.e., Central and Peripheral. The BLE implementation comes in the form of a patch file for NimBLE [13], an open-source BLE stack developed by Apache. Patch is contained in the **bb-nimble** folder and is specific to NimBLE 1.8.0. Our BLE implementation replaces the existing Bluetooth security and stack. The two devices can interact using BLE as usual.

3.2 Installation

3.2.1 Bluetooth Classic

To use the BB protocols over BC, one needs to compile the central.c and peripheral.c sources and run them on two separate devices. This can be achieved by running the following commands:

```
git clone https://github.com/sacca97/bb-protocols.git
cd bb-protocols/bb-portable
cmake .
make central && make peripheral
```

The compiled binaries can then be executed, one in the Central and one in the Peripheral, by running, in the following order:

```
./peripheral
./central --address AA:BB:CC:DD:EE:FF
```

Where `AA:BB:CC:DD:EE:FF` must be replaced by the actual Bluetooth address of the Peripheral device.

This demo relies on hardcoded pre-shared keys to run the BB-Session protocol, hence assuming that the devices have previously performed BB-pairing to share such keys.

3.2.2 Bluetooth Low Energy

To use the BB protocols over BLE, one needs to patch an existing NimBLE stack code and install it in two separate nRF52 devices. To set up and install NimBLE, we point to the official repository of mynewt OS (https://github.com/apache/mynewt-core). The only requirement is to apply the "bb-protocols.patch" using the "git apply" command before flashing the devices.

One should flash the btshell (Central) and bleprph (Peripheral) applications on the two devices, respectively. The BLE implementation is a fully functional implementation of BlueBrothers. Devices can run BB-pairing to pair and BB-session to establish a secure session if they have already paired. The device with btshell offers a CLI to control the devices and perform various actions such as *connect*, *disconnect*, *pair*, and *unpair*. These commands are further explained in the application itself via the *help* command.

Figure 7 shows our BLE testbed with two nRF52. One of them is connected to the Nordic Power Profiler Kit, which enables monitoring the power consumption of the device.

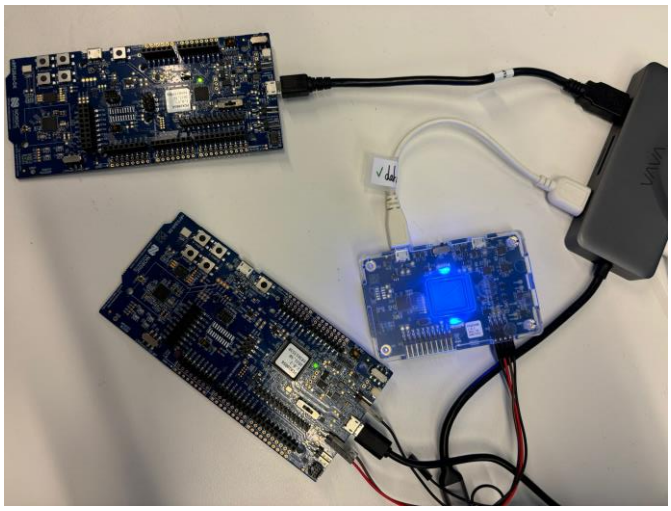

Figure 7 BLE tested setup

Chapter 4 Conclusion

This deliverable provides a brief overview of the prototypes implementing secure intra- and inter-devices communication.

Specifically, considering the intra-device communication, it presents the implementation of two secure communication protocols: the standard SCP03 and the newly developed NSCP, which is better suited for resource-constrained environments due to its use of the Xoodyak cryptographic algorithm. The communication setup involves a Nexys A7 FPGA, serving as the Secure Element (slave), and a Raspberry Pi 4B+ acting as the master. The Nexys A7 FPGA runs the open-source CoreV architecture.

Regarding inter-device communication, this deliverable presents two implementations of the BlueBrothers protocols. The Bluetooth Classic (BC) setup involves two Raspberry Pi 4B+ devices communicating over BC, with one acting as the Central (initiator) and the other as the Peripheral. In the Bluetooth Low Energy (BLE) setup, two nRF52 devices are used to communicate over BLE, following the same role distribution: one as Central and the other as Peripheral.

All implementation code is publicly available under permissive open-source licenses. For more information, refer to Deliverables D5.1 and D5.2.

Chapter 5 List of abbreviations

Abbreviation	Translation
BB	BlueBrothers
BC	Bluetooth Classic
BLE	Bluetooth Low Energy
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input/Output
I2C	Inter-Integrated Circuit
IDE	integrated development environment
IoT	Internet of Things
MCU	Microcontroller unit
NSCP	New Secure Communication Protocol
RPI4	Raspberry Pi 4
S&P	Security and Privacy
SCP03	Secure Communication protocol 03
SPI	Serial Peripheral Interface
USB	Universal Serial Bus

Chapter 6 Bibliography

[1] Xoodyak

J. Daemen, Joan, Seth Hoffert, Michael Peeters, Gilles Assche, and Ronny Keer. 2020. “Xoodyak, a Lightweight Cryptographic Scheme.” IACR Transactions on Symmetric Cryptology, June, 60–87.

Available: https://doi.org/10.46586/tosc.v2020.iS1.60-87

[2] Nexys A7: FPGA Trainer Board recommended for ECE curriculum, https://digilent.com/shop/nexys-a7-fpga-trainer-board-recommended-for-ece-curriculum/?srsltid=AfmBOop9dZRsS3GlkK_ul8c1C-sR-FNdCMQqfXUdcOByjJcHqEracL6K

[3] JTAG-HS2 Programming cable, https://digilent.com/shop/jtag-hs2-programming-cable/?srsltid=AfmBOoqO6kvqoStJckTw-mflzllgdr39xf9c_x-XdBGNgQvhMY8z70-Z

[4] Raspberry Pi 4, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[5] Raspberry Pi 4: pinout I2C, https://pinout.xyz/pinout/i2c

[6] NSCP and SCP03 Master, Github repository: https://github.com/securitypattern/orshin-rpi_scp03

[7] NSCP and SCP03 Slave, Github repository: https://github.com/securitypattern/orshin-corev-secure-element/tree/main

[8] CORE-V-MCU Quick Start Guide: https://github.com/openhwgroup/core-v-mcu/blob/master/emulation/quickstart/README.md

[9] CORE-V-MCU Nexys, Github repository: https://github.com/openhwgroup/core-v-mcu/blob/master/emulation/quickstart/core_v_mcu_nexys.bit

[10] Command-Line-Interface test routines for the CORE-V MCU, Github repository: https://github.com/openhwgroup/core-v-mcu-cli-test

[11] CORE-V-MCU user manual: https://docs.openhwgroup.org/projects/core-v-mcu/

[12] Bluetooth Core Specification 6.1: https://www.bluetooth.com/specifications/specs/core-specification-6-1/

[13] Apache NimBLE: https://github.com/apache/mynewt-nimble

[14] Nordic nRF52: https://www.nordicsemi.com/Products/Development-hardware/nRF52-DK