D4.3: Test performance and results

Project number:	238811
Project acronym:	**UNIQUE**
Project title:	Foundations for Forgery-Resistant Security Hardware
Start date of the project:	01.09.2009
Duration:	33 months

Deliverable type:	Report
Deliverable reference number:	238811/ D4.3 / 1.0
Deliverable title:	Test performance and results
WP contributing to the deliverable:	WP4
Due date:	2012-05-31 (M33)
Actual submission date:	2012-05-31 (M33)

Responsible organisation:	SIRRIX
Authors:	Timm Korte (Sirrix), Vincent van der Leest (IID), Peter Simons (IID), Erik van der Sluis (IID), Roel Maes (K.U.Leuven), Anthony Van Herrewege (K.U.Leuven), Remy Chau (TCS)
Abstract:	This document contains the description of the tests performed on the demonstrators and the results of those tests.
Keywords:	Test framework, demonstrators

Dissemination level:	Public
Revision:	V1.0

Instrument:	STREP
Thematic Priority:	ICT

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Scope of document

This document contains the description of the tests performed on the prototype demonstrators and the results of those tests.

1.2 List of abbreviations

ASIC	Application Specific Integrated Circuit
CRP	Challenge-Response Pair
DB	Database
DFF	Data Flip-Flop
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GPIO	General Purpose I/O
ID	Identity
I/O	Input / Output
I2C	Inter Integrated Circuit
LED	Light-Emitting Diode
LFSR	Linear Feedback Shift Register
LR-PUF	Logically Reconfigurable PUF
MPW	Multi Project Wafer
NVM	Non-Volatile Memory
PC	Personal Computer
PUF	Physically Unclonable Function
RFID	Radio Frequency Identification
RO	Ring Oscillator
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
TSMC	Taiwan Semiconductor Manufacturing Company
UART	Universal Asynchronous Receiver/Transmitter

1.3 Document overview

In section 2, we describe the components that make up the common test environment for all tests performed on the prototype demonstrators. Section 3 then delivers a short recap of the implemented use-cases on which testing was performed as described in section 4 and a summary of the test results in section 5. Further test results concerning the low-level functional tests as well as the PUF characterization can be found in the full evaluation report in D3.3.

2 Common Test environment

The test environment for the demonstrators consists of the same main components also used in the PUF evaluation:

- A board with sockets for the UNIQUE ASIC (physical part of the PUF)
- An FPGA board and firmware (logical part of the PUF & implementation of test cases)
- A host PC
- Various lab devices (climate chamber, test equipment…)

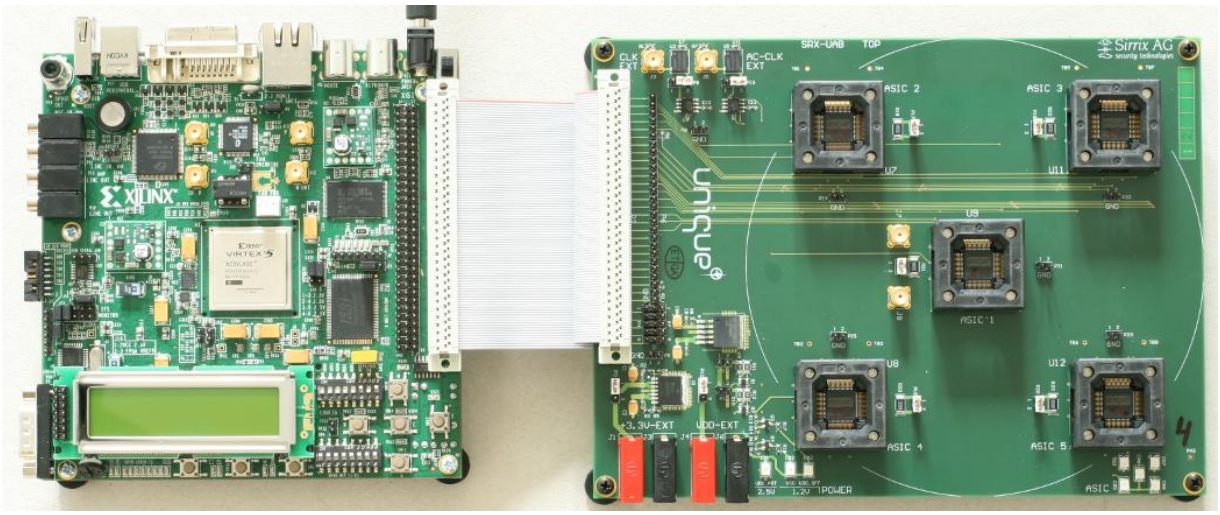

Figure 1: FPGA and custom ASIC board

2.1 ASIC board

Several requirements have been taken into account during the development of the ASIC board in order to facilitate penetration testing, consisting of special form-factor requirements dictated by the security evaluation equipment, connectors and sockets in order to interface with the ASIC on the one side and the FPGA-board on the other as well as matching power supply and clock planning as listed below.

Form factor requirements:

- Open-top ASIC socket to support front-side analysis
- Several (preferably 5) sockets on board for testing multiple ASICs at once, board should be functional if only a subset of sockets is populated.
- In order to make back side analysis possible, the board is opened behind the central ASIC position so that it is possible to observe the component during operation.
- The central position supports both socket insertion or direct ASIC soldering in order to limit the working distance for optical analysis
- Direct access to ASIC power lines for side channel analysis
- Large clearance (15 cm of diameter) on the top of the board for extreme temperature testing (using heater head)

Connectors and Sockets:

- Connection between ASIC and FPGA-board via Ribbon-Cable
- SMA differential sockets for external clocking options
- Test pads and ground connectors at convenient locations

Power Budget:

- Estimated at 52mW @ 33MHz per ASIC
- Factors for temp/clock variations up to x6

Power supply:

- ASIC: VDD=1.2V, VDD_OFF=1.2V, VDD_PST=2.5V should be derived from FPGA supplied voltages (3.3V and 5V)
- Design should support side channel analysis
- Both VDD and VDD_OFF should be controllable by incoming connector signals
- Core supplies should lag IO-supplies by >= 1us

Clock generation:

- ASIC core clock @ 33MHz, ASIC active core clock @ 33 MHz – 65 MHz
- Clocking selectable by jumper (either onboard or supplied externally via differential SMA sockets)

Other:

- ASIC scan chain support, all test signals from at least one socket routed to the board connector
- Board should support operating temperatures from -40°C to +125°C

2.2 FPGA board

The Xilinx Virtex 5 development kit HW-V5-ML501-UNI-G has been selected as FPGA board for the UNIQUE project. This board has been used to implement VHDL building blocks for both the demonstrators as well as the PUF Test Framework. The most important properties which are met by this selected board are:

- Simple connectors and interfaces
  - Expansion headers for connection with ASIC board
  - RS-232 for connection with host PC
  - JTAG for programming the FPGA
  - Differential clock connectors
- The connection to the ASIC must use interface levels of 2.5V.
- Sufficient FPGA size.
- Many LEDs, a display and other peripherals (like switches, buttons, etc.).
- Multiple on board non-volatile memories, with one of at least 1kByte in at least two pages and a second one of at least 64 bytes.

3 Use Case description

3.1 Use Case 1

The following short description has been included from D4.2 for reference.

Use case 1 considers mutual authentication of a resource-constrained PUF-enabled RFID token (simply referred to as *token* from here on) and a more powerful RFID reader (referred to as *verifier* from here on). The demonstrator emulates a token using the UNIQUE ASIC and the FPGA, while the verifier is emulated on a host PC connected to the FPGA board. The token-verifier communication is emulated by a serial UART interface in the prototype.

3.1.1 Protocol

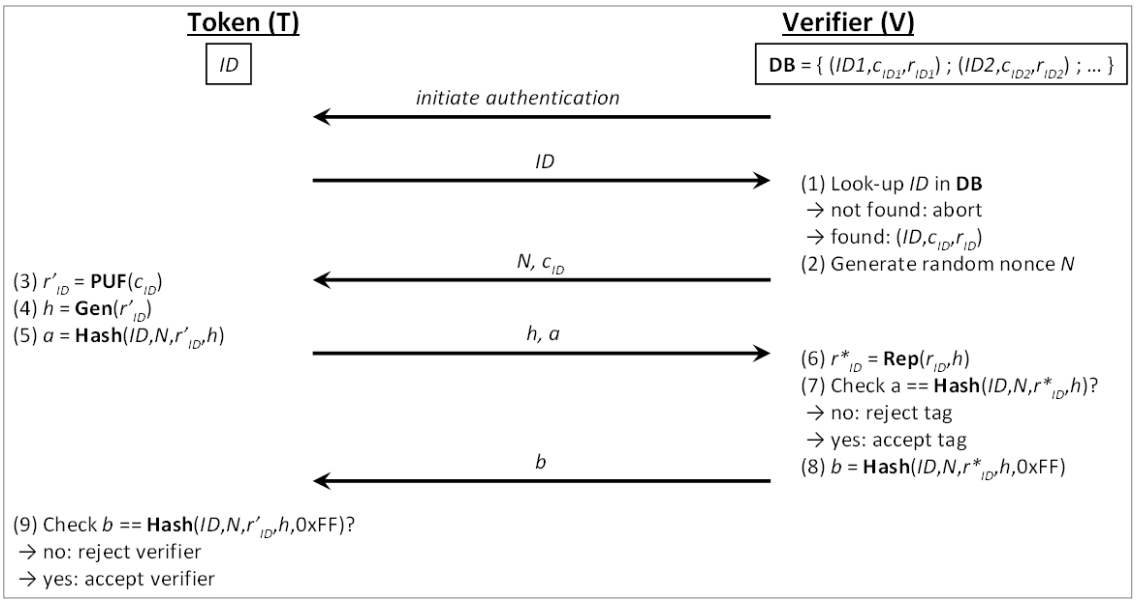

Figure 2: Use Case 1 Mutual Authentication Protocol

The actions and communications required to achieve mutual authentication between token and verifier are shown in the protocol description in Figure 2. The prerequisite for this protocol is that the verifier holds a database of PUF challenge-response pairs (CRPs) and corresponding identity vectors (ID) for every token it needs to authenticate. This database is constructed during an enrolment phase of the tokens in a secured environment.

For more details on the protocol and its design and security details we refer to [1].

For more detailed description of the Use Case 1 setup and its components, please refer to D4.2 section 2.1.

3.2 Use Case 2

The demonstrator of Use Case 2 will be used to show how PUFs can be used to securely bind software to specific hardware (Hardware/Software-binding). Furthermore, using the LR-PUFs that have been defined by the UNIQUE project, it will be demonstrated how this hardware can be protected from illegitimate software version changes (prevention of software downgrading).

3.2.1 FPGA Architecture

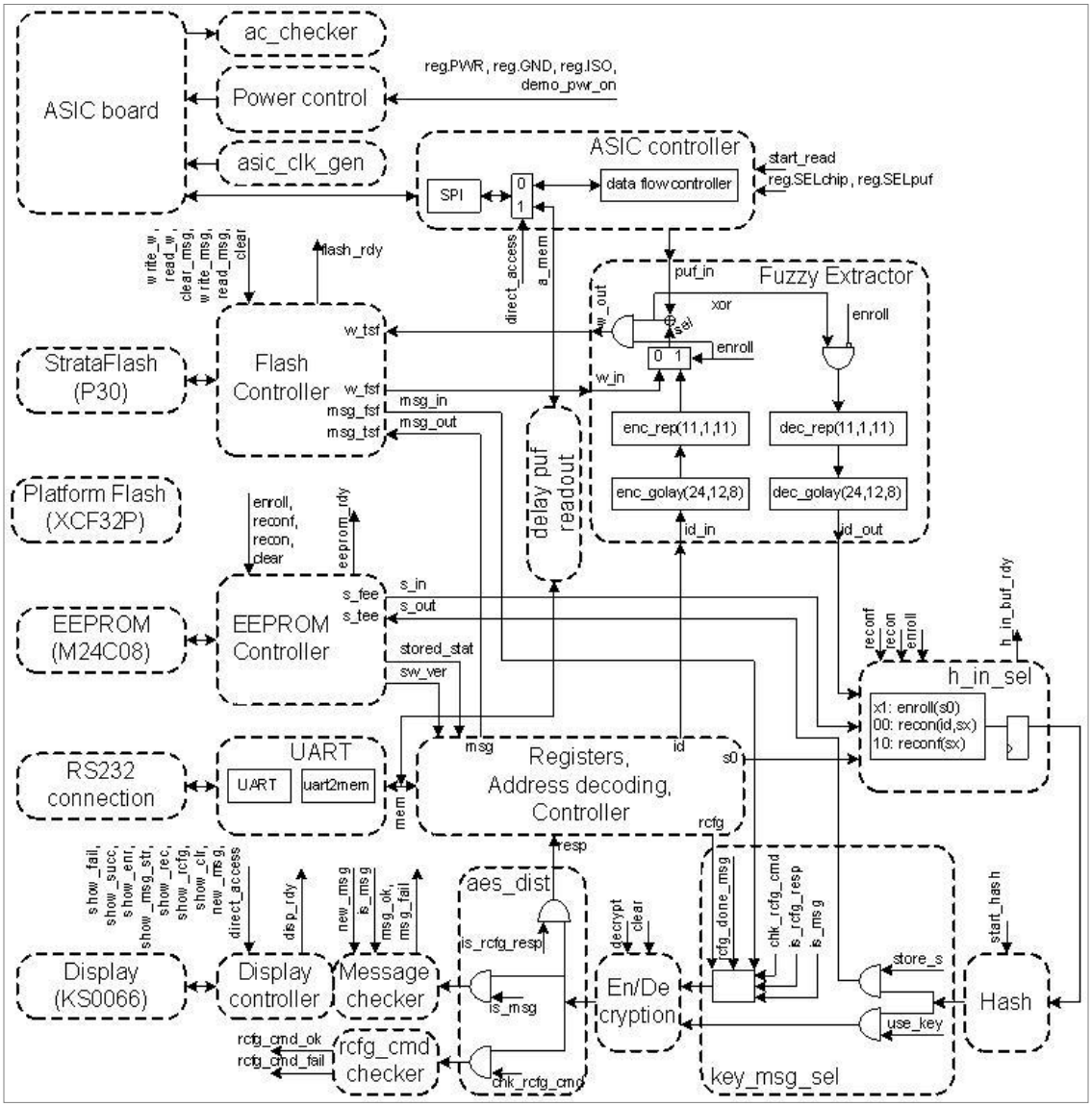

Figure 3: FPGA architecture overview of Use Case 2

For more detailed description of the Use Case 1 setup and its components, please refer to D4.2 section 2.2.

4 Behavioural testing

This section describes the behavioural testing performed on the prototype use-case implementations.

4.1 Use Case 1

Use case 1 considers mutual authentication of a resource-constrained PUF-enabled RFID token and a more powerful RFID reader. The demonstrator emulates the token using the UNIQUE ASIC and the FPGA, while the reader is emulated using the PC. The underlying security protocol has been formally proved to achieve mutual authentication of a genuine token and an honest reader in the presence of any computationally constrained adversary with the following properties:

1. The adversary controls the communication channel between token and reader, i.e., the adversary can eavesdrop, insert, delete and modify any protocol message.
2. The adversary learns whether the reader accepted the token.
3. The adversary can read the non-volatile memory of the token.
4. The adversary cannot access the responses of the PUF of the token.

Further, the protocol should have the availability property, which ensures that a genuine token is always accepted by an honest reader. The resulting functional and security test cases are listed in Table 1 and Table 2, respectively.

Table 1: Use case 1: functional tests (availability)

Test	Description	Expected result
FT-AV	Execute protocol between token and reader.	Reader accepts token and token accepts reader.

Table 2: Use case 1: security tests (mutual authentication)

Test	Description	Expected Result
ST-TA (token authentication)	Execute the protocol between token and reader. While the protocol is running, change one or more bits of the value of *ID* or *a* sent from token to reader.	Reader should reject token and abort the protocol.
ST-RA (reader authentication)	Execute the protocol between token and reader. While the protocol is running, change one or more bits of the value of *N*, c_{ID} or *b* sent from reader to token.	Token should reject reader and abort the protocol.

Test	Description	Expected Result
ST-SCA (side channel analysis)	Execute the protocol between token and reader and record the values (ID, N, c_{ID}, b) sent from reader to token. While the protocol is running, perform standard side-channel attacks against the token to extract the PUF response r_{ID}=PUF(c_{ID}).	It should be infeasible to extract the PUF response r_{ID} from the token.
ST-IA (invasive attack)	Perform invasive attacks against the token to extract the PUF response r_{ID}=PUF(c_{ID}) while the token is running the protocol with the reader.	It should be infeasible to extract the PUF response r_{ID} from the token.

Note that the demonstrator resembles a proof-of-concept implementation of use case 1 that shows the viability and efficiency of our approach. It does not yet include protection against side channel and invasive attacks, which means that the ST-SCA and ST-IA test will most probably fail. In fact, the PUF response r_{ID} can be obtained by a basic invasive attack from the current implementation by just tapping the connection between the ASIC and the FPGA. Protection against side channel attacks can be easily added to the demonstrator by using side channel aware designs for the underlying algorithms of the token. Securing the implementation against invasive attacks, however, requires implementing the token part of the demonstrator in ASIC. Given the strict schedule of the UNIQUE ASIC manufacturing process, this was not possible since the ASIC design had to be finished before the use cases were fully developed.

4.1.1 Basic demo flow

Information on how to execute the steps laid out in the basic demo flow can be found in the manual delivered as part of the Use Case 1 WP4 files.

Preparation

- Setup the FPGA board and demo PC with the firm- and software.
- Push the reset button to put FPGA in initial state.

Enrollment

- Select wanted ASIC, PUF and ID.
- Run enrollment executable.
- Enrollment data should now have been added to enrollment database.

FT-AV

- Select ASIC, PUF and ID for which enrollment has been executed

- Run verification executable.
- Successful verification status should be shown both on the PC side and LED on the FPGA board should turn on.

ST-TA

- Option 1 - Changing ID
    - Select ASIC & PUF for which enrollment has been executed. Select ID for which enrollment has not been executed.
    - Run verification executable.
    - Failed verification should now be shown on the PC side, status LED on FPGA board should stay off.

- Option 2 - Changing *a*
    - Select ID for which enrollment has been executed. Select ASIC and/or PUF for which enrollment has not been executed.
    - Run verification executable.
    - Failed verification should now be shown on the PC side, status LED on FPGA board should stay off.

ST-RA

Modification of either *N* or *b* requires modification of the executables delivered with Use Case 1. Since this would require recompilation of the executables, we omit the required steps on how to do this. Below we present an option to verify ST-RA which does not require any drastic changes to files.

- Option 1 - Changing c_{ID}
    - Select ASIC, PUF & ID for which enrollment has been executed.
    - Open database file and modify the cID value for the selected ID.
    - Run verification executable.
    - Failed verification should now be shown on the PC side, status LED on FPGA board should stay off.

ST-SCA & ST-IA

These tests were implemented due to lack of side-channel attack protection and invasive attack protection on the token side.

4.2 Use Case 2

This section starts with a basic demo flow and then provides some variants to show what can and cannot be done.

The result of every command to the LR PUF is shown in the GUI (see Figure 4: Use Case 2 GUI), which is used to perform all the actions required to test the use case, and on the FPGA board's display (except for the ASIC selection, this is shown with the LEDs on the ASIC board).

The actual tests performed and their expected results are listed in the instructions on how to operate the GUI in order to test the specified functionality for use case 2.

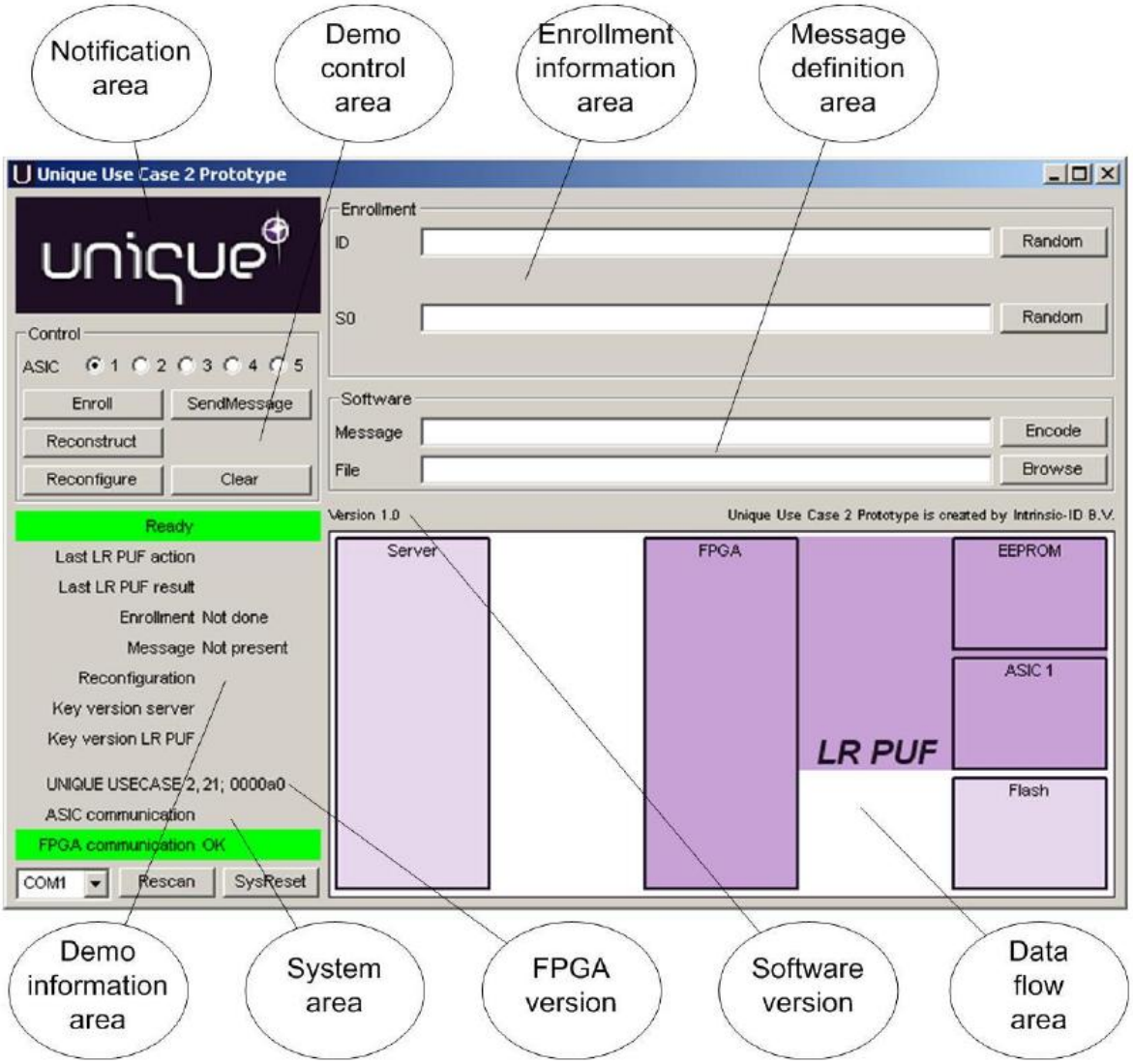

Figure 4: Use Case 2 GUI

Note for testing: It is strongly recommended that enrollment always takes place on ASIC 1. Because the FPGA always tries to perform a reconstruction after reset (when enrolled and message is available). As ASIC 1 is selected by default after reset or power up, this ASIC will be used to do the reconstruction. When another ASIC than ASIC 1 has been used for enrollment this reconstruction will fail.

4.2.1 Information about the system setup

The LR PUF system in the demo is built with separate components (FPGA, EEPROM, ASIC and Flash memory).

In the real system the FPGA, EEPROM and the PUF-structure integrated in the ASIC will be integrated into one chip, or be included together as part of a bigger chip and the Flash memory will be external.

Therefore the data in the flash memory is considered to be freely accessible and can be changed or copied. The data in the EEPROM is considered safe for non-invasive attacks.

4.2.2 Basic demo flow

Preparation

- Setup the demo
- Push the Clear button to start from a clean LR PUF
- Push the SysReset button to put the FPGA in initial state

Enrollment

- Fill in ID and S0 fields or use the Random buttons to generate random values
- Select ASIC 1 (is already selected by default)
- Push the Enroll button
- Enrollment should now be successfully done

Create and send message

- Fill in the message in the Message field
- Push the Encode button to generate the encoded message file
- Push the SendMessage button
- The message should now be stored in the FPGA board's flash memory

Reconstruct

- Push the Reconstruct button
- After reconstruction is finished the message should be shown on the FPGA board's display

Reconfigure

- Push the Reconfigure button
- A correct response should have been received (key version for the Server and LR PUF is now 2)

Check that reconfiguration has been done

- Push the Reconstruct button
- Because the message that is present in the FPGA board's flash is coded with the previous key, the reconstruction should now fail

Create and send new message

- Fill in the new message in the Message field
- Push the Encode button to generate the encoded message file
- Push the SendMessage button
- The message should now be stored in the FPGA board's flash memory

Reconstruct

- Push the Reconstruct button
- After reconstruction is finished the new message should be shown on the FPGA board's display

4.2.3 Enrollment for the second time

Preparation

- Run the entire basic demo flow

Second enrollment

- Fill in ID and S0 fields or use the Random buttons to generate random values
- Push the Enroll button
- The enrollment should now have failed

Check that nothing has changed in the LR PUF due to second enrollment

- Push the Reconstruct button
- After reconstruction is finished the last message from the basic demo flow should be shown on the FPGA board's display

4.2.4 Using the helper data on a different chip

To mimic that the helper data is copied to another chip, another ASIC than the one used for enrollment is selected, while the helper data remains present in the FPGA board.

Preparation

- Run the entire basic demo flow (assumed to have been done with ASIC 1)

Run reconstruction with another ASIC

- Select one of ASIC 2 to 5
- Push the Reconstruct button
- After reconstruction is finished no message should be shown on the FPGA board's display and reconstruction must have failed

Run reconstruction with original ASIC

- Select ASIC 1

- Push the Reconstruct button
- After reconstruction is finished the new message should be shown on the FPGA board's display

4.2.5 Trying to go back

This flow mimics that a rogue Server is used to try to roll back the key version.

This may be not so easy to show for a large group of people and it is a bit harder to explain because it also involves changing files.

Preparation

- Run the entire basic demo flow

Put the software version of the Server back

- In the installation directory: Open Unique_UC2.ini
- Change the value in the field swVerServer to 1
- Close Unique_UC2.ini
- Push the SysReset button
- Now the key versions of the server and LR PUF are different

Check reconstruction

- Push the Reconstruct button
- Because the message in the FPGA board's flash is still encoded with the key of the LR PUF, this will pass

Check new message, created with the rolled back key version of the Server

- Fill in message, push Encode, push SendMessage
- Push Reconstruct
- Because the message is generate with the wrong key, it cannot be decoded, so reconstruction failed.

Try to reconfigure so that the Server's software version will be used from now

- Push the Reconfigure button
- Because the reconfigure command is encoded with the previous key the LR PUF cannot decode the command, and therefore will not reconfigure

Go back to the original Server and check that the LR PUF state has not changed

- Open Unique_UC2.ini, change the value in the field swVerServer to 2, close Unique_UC2.ini
- Push SysReset
- Now the key versions of Server and LR PUF are the same again
- Push the Reconstruct button

- Reconstruction fails, because the message sent by the rogue server is still loaded in the FPGA board's flash memory. This failure shows that messages sent by another server cannot be opened.
- Push the Browse button and select a message starting with msg_sw002
- Push SendMessage
- Push Reconstruct
- Now the original message is decoded and shown. This shows that even when another server attacks, messages from the original server can still be decoded.

4.2.6 Use case 2 with Buskeeper PUF

Use case 2 was first designed to demonstrate a hardware/software binding application using a SRAM PUF. The test described in this section introduces the usage of Buskeeper PUF instead of the SRAM PUF in the use case 2.

Both SRAM and Buskeeper are memory-based PUF. However, unlike the SRAM PUF, the Buskeeper PUF "memory" cells can be spread over the component, which makes probing and reverse engineering attacks much more difficult.

Moreover, deliverable D3.3 mentions that, like SRAM PUFs, Buskeeper PUFs are robust and remain unpredictable over temperature and voltage variations. Therefore, the fuzzy extractor specifically designed to correct SRAM PUFs data errors in use case 2 can also be used for Buskeeper PUFs.

To test the buskeeper PUFs in use case 2 environment, the following steps are performed:

Preparation

- Setup the demo to run the use case 2
- Push the clear button to start from a clean LR PUF
- Push the SysReset button to put the FPGA in initial state
- Through UART interface write in the FPGA SELpuf register at address x"F00001" the value x"51" to select the buskeeper PUF.

Enrollment

- Fill in ID and S0 fields or use the Random buttons to generate random values
- Select ASIC 1 (is already selected by default)
- Push the Enroll button
- Enrollment should now be successfully done

Create and send message

- Fill in the new message in the Message field
- Push the Encode button to generate the encoded message file
- Push the SendMessage button
- The message should now be stored in the FPGA board's flash memory

Reconstruct

- Push the Reconstruct button
- After reconstruction is finished the new message should be shown on the FPGA board's display

Verify that SRAM Puf is not used

As SRAM PUF is selected by default in use case 2, this step ensures that SRAM PUF is not selected when running this use case with the buskeeper PUF.

- Restart the board by powering down and powering up
- Push the Reconstruct button
- The reconstruction fails because SRAM PUF is selected
- Select Buskeeper PUF by writing in the FPGA SELpuf register at address x"F00001" the value x"51"
- Push the Reconstruct button
- The reconstruction succeeds

Those steps have been performed with five different UNIQUE ASICs in order to verify operation with this different PUF type.

5 Test results

All specified tests have been performed on both use case implementations in the demonstrator prototypes and delivered the expected results.

As for the variant of use case 2 using the Bus-Keeper PUFs, all tests also succeeded. Although those tests have only been run under normal operating conditions, the results show that use case 2 can also be performed using the buskeeper PUF instead of SRAM PUF. This is in line with the buskeeper PUF assessment results described in deliverable D3.3 at least under normal operating conditions.

6 References

[1] R. Maes, R. Peeters, A. Van Herrewege, C. Wachsmann, S. Katzenbeisser, A. Sadeghi, and I. Verbauwhede, "*Reverse Fuzzy Extractors: Enabling Lightweight Mutual Authentication for PUF-enabled RFIDs*," In Financial Cryptography and Data Security - 16th International Conference, FC 2012, Lecture Notes in Computer Science, Springer-Verlag, 16 pages, 2012