

FutureTPM

D5.4

Report on implementation

Project number:	779391
Project acronym:	FutureTPM
Project title:	Future Proofing the Connected World: A Quantum- Resistant Trusted Platform Module
Start date of the project:	1 st January, 2018
Duration:	36 months
Programme:	H2020-DS-LEIT-2017

Deliverable type:	DEM
Deliverable reference number:	DS-06-779391 / D5.4/ 1.0
Work package contributing to the deliverable:	WP 5
Due date:	June 2020 – M30
Actual submission date:	14th of July, 2020

Responsible organisation:	IFAG
Editor:	Michael Albrecht
Dissemination level:	PU
Revision:	1.0

Abstract:	This deliverable w final version of VI TPM and the SW-	ill report the publ M based QR TPI based QR TPM.	ic overview on the M, HW based QR
Keywords:	Software-TPM, Implementation	Virtual-TPM,	Hardware-TPM,



The project FutureTPM has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 779391.

Editor

Michael Albrecht (IFAG)

Contributors (ordered according to beneficiary numbers)

Andreas Wallner (IFAG), Thomas Pöppelmann (IFAG)

Christine Wright (RHUL), Daniele Sgandurra (RHUL), Fernando Sanchez Ortega (RHUL), Harry Lockyer (RHUL)

Paulo Sérgio Alves Martins (INESC-ID), Luís Fiolhais (INESC-ID), Rogério Paludo (INESC-ID)

Internal Reviewers

Sofianna Menesidou (UBITECH), Dimitris Papamartz (UBITECH) Li Chen (SURREY), Athanasios Giannetsos (DTU)

Disclaimer

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The content of this document reflects only the author's view – the European Commission is not responsible for any use that may be made of the information it contains. The users use the information at their sole risk and liability.

Executive Summary

The overall goal of FutureTPM is to design a Quantum-Resistant (QR) Trusted Platform Module (TPM) by designing and developing QR algorithms suitable for inclusion in a TPM; based on the selection of various families of QR crypto primitives as described in D2.1 [17]. For each primitive, detailed attention is being paid to their implementation details and performance evaluation towards their validation in the full range of TPM environments, namely hardware (hTPM), software (sTPM) and virtualization (vTPM) environments. Accompanied with the QR algorithm design and implementation, the FutureTPM project will demonstrate three use cases in secure mobile wallets and payments, activity tracking and device management, which will provide environments and applications to validate the feasibility and performance of the newly developed QR TPM in these three selected real-world systems that may be affected by the advent of quantum computing as a threat to security.

In this context, this deliverable reports the design and implementation of the quantum-resistant (QR) HW-TPM (Task 5.3), gives a consolidated overview on the final implementation of the QR SW-TPM (Task 5.1) and describes the current status of the final QR virtual-TPM implementation (Task 5.2), scheduled for M30.

Open Issues, documented in D5.1 and D5.2 are resolved, NTTRU is implemented as well as L-DAA on the promised level of optimization. The QR HW-TPM Demonstrator (D5.3) is finally available, including updated security schemes (BLISS replacing qtesla) and ready for integration- and use case testing (WP6).

Contents

Executive Summary	II
List of Figures	IV
List of Tables	v
Chapter 1 Introduction	1
1.1 Scope and Purpose	1
1.2 Relation to Other WPs and Deliverables	2
1.3 Deliverable Structure	3
Chapter 2 Update on the SW-TPM	4
2.1 New Endpoints	4
2.2 Miscellaneous SW-TPM Updates	5
2.3 Resolved and Open Issues	6
Chapter 3 Virtual TPM Overview	7
3.1 QR Virtual-TPM Architecture	7
3.1.1 Choice of QEMU	7
3.1.2 Choice of QR-Libtpms	8
3.1.3 Implementation of Virtual TPM	8
3.1.4 Using SW-TPM within V-TPM	9
3.1.5 QR V-TPM	
3.2 QR Cryptographic Algorithms in V-TPM	13
3.2.1 SPHINCS+	
3.2.2 Rainbow	15
3.2.3 BIKE	
3.2.4 Integration of SPHINCS+, Rainbow and BIKE into QR-Libtpms	
3.2.5 Open Issues	
Chapter 4 HW-TPM Demonstrator Overview	
4.1 QR HW-TPM Demonstrator	
4.2 FPGA Evaluation Platform	
4.3 Architecture of the HW-TPM Demonstrator	
4.4 Basic Operation of the QR HW-TPM Demonstrator	21
4.5 New Endpoints	21
4.6 How to add new algorithms	
4.7 Using the HW-TPM demonstrator	
4.8 Example	23
4.9 TSS Adaptations	24

4.10	Hardware Acceleration of PQC	25
Chapter	5 Conclusions and Next Steps	28
List of A	bbreviations	29
Bibliogra	aphy	30
Appendi	ces	32

List of Figures

Figure 1: Types of TPM according to TCG 1
Figure 2: Overview of the QR TPM environments
Figure 3: Architecture of V-TPM
Figure 4: Output of Kyber Encrypting a File in a VM12
Figure 5: Output of Signing Operation on a File in a VM12
Figure 6: "swtpm socket" command17
Figure 7: Simplified system overview on the DesignStart SoC
Figure 8: General overview of the QR HW-TPM and comparison to the IBM SW-TPM 20
Figure 9: Block diagram of the QR HW-TPM demonstrator
Figure 10: Architecture of IBM's TPM2.0 SW implementation (from http://ibmswtpm.sourceforge.net/)23
Figure 11: Communication Diagram between Alice and Bob for setup and repeated usage steps 24
Figure 12 TSS architecture / parts25
Figure 13 Option for hardware acceleration of PQC
Figure 14: SPHINCS+ Compilation
Figure 15: SPHINCS+ Unit Test
Figure 16: SPHINCS+ Unit Test Results (Record 0 to 4)
Figure 17: SPHINCS+ Unit Test Results (Record 95 to 99)
Figure 18: Rainbow Unit Test Results (Record 0 to 3)
Figure 19: Rainbow Unit Test Results (Record 96 to 99)
Figure 20: Rainbow Unit Test - Key Generation, Signature Generation and Verification
Figure 21: BIKE Unit Test
Figure 22: BIKE Unit Test Results (Record 0 to 4)
Figure 23: BIKE Unit Test Results (Record 95 to 99)
Figure 24: SPHINCS Code Copied Into QR-Libtpms42
Figure 25: Rainbow Code Copied Into QR-Libtpms42
Figure 26: BIKE Code Copied Into QR-Libtpms

Figure 27: CryptSphincsPlus.c, CryptRainbow.c, CryptBIKE.c and CryptUtil.c Copie Libtpms	ed Into QR- 43
Figure 28: CryptSphincsPlus_fp.h, CryptRainbow_fp.h and CryptBIKE_fp.h Copied Into	QR-Libtpms 44
Figure 29: New BIKE files Copied Into QR-Libtpms	44
Figure 30: SPHINCS+, Rainbow and BIKE Compiled Into QR-Libtpms	45
Figure 31: SPHINCS+, Rainbow and BIKE Compiled Into QR-Libtpms	46

List of Tables

Table 1: V-TPM Performance Results	. 11
Table 2: Testing and Timings of the V-TPM within the Docker Container	. 12
Table 3: SPHINCS+ Parameter Sets	. 13
Table 4: NIST Categories	. 14
Table 5: Rainbow Parameter Sets	. 15

Chapter 1 Introduction

1.1 Scope and Purpose

Research on quantum computers has drawn attention from governments and industry. If, as predicted, a large-scale quantum computer becomes a reality within the next 15 years, existing public-key algorithms will be open to attack. FutureTPM is aimed at designing and developing a Quantum-Resistant (QR) Trusted Platform Module (TPM). FutureTPM's main goal is to enable a smooth transition from current TPM environments, based on existing widely used and standardised cryptographic techniques, to systems providing enhanced security through QR cryptographic functions, including secure authentication, encryption and signing functions.

According to TPM 2.0 specifications [19], there are five different types of TPM 2.0 implementations:

- **Discrete TPMs**: they are dedicated chips that implement TPM functionality in their own tamper-resistant semiconductor package. Theoretically, they are the most secure type of TPM as, for instance, their packages are required to implement tamper resistance.
- **Integrated TPMs**: they are included as part of another chip. While they use hardware that resists software bugs, they are not required to implement tamper resistance.
- **Firmware TPMs**: these are software-only solutions that run in a CPU's trusted execution environment. Since these TPMs are entirely software solutions that run in trusted execution environments, these TPMs are more likely to be vulnerable to software bugs.
- **Software TPMs**: they are software emulators of TPMs that depend on the environment that they run in. Typically, they offer the same level of security of their execution environment, and so are vulnerable to software bugs. Therefore, they are typically used for development purposes.
- Virtual TPMs: they are meant to be provided by a hypervisor to allow virtual machines to share a single instance of a TPM. These TPMs rely on the hypervisor to provide them with an isolated execution environment that is hidden from the software running inside virtual machines.

TRUST ELEMENT	SECURITY LEVEL	SECURITY FEATURES	RELATIVE COST	TYPICAL APPLICATION
DISCRETE TPM	HIGHEST	TAMPER RESISTANT HARDWARE	\$\$\$	CRITICAL SYSTEMS
INTEGRATED TPM	HIGHER	HARDWARE	\$\$	GATEWAYS
FIRMWARE TPM	HIGH	TEE	\$	ENTERTAINMENT SYSTEMS
SOFTWARE TPM	NA	NA	cc	TESTING & PROTOTYPING
VIRTUAL TPM	HIGH	HYPERVISOR	ç	CLOUD ENVIRONMENT

Figure 1: Types of TPM according to TCG

FutureTPM is investigating technologies for a new generation of TPM-based solutions, including hardware (HW), software (SW) and virtualization environments, by incorporating QR cryptographic primitives. In addition, FutureTPM aims to prove and validate the applicability, usability, effectiveness and value of the QR TPM concepts, models and algorithms in real-world settings, including industry and e-commerce, which may be affected by the advent of quantum computing.

In this deliverable, we describe the current implementation status of the QR HW-TPM demonstrator; overview the concept and progress on the QR Virtual TPM; and give an update on the QR SW-TPM.

The subsequent figure gives an overview of the QR TPM environments developed within WP5. It shows that all QR TPM environments are based on the IBM SW TPM, where the QR VM-TPM and the QR SW-TPM share the same code-basis and the QR VM-TPM implements additional crypto algorithms (BIKE, SPHINS+, Rainbow), as has been described in D2.2 [10]. Furthermore, it gives an update on the algorithms that have now been implemented in D5.4 – latter are marked bold.



Figure 2: Overview of the QR TPM environments.

1.2 Relation to Other WPs and Deliverables

WP2 deals with the selection of which QR cryptographic primitives should be implemented in each TPM environment, both providing advice and receiving feedback from WP5. WP5 serves as the basis for the developments of the TPM environments to be demonstrated in the envisioned use cases (WP6). In particular, the QR SW-TPM will be the basis for the "Personal Activity and Health Kit Data Tracking" use case, the QR Virtual TPM will serve as basis for the "Device Management" use-case and the QR HW-TPM will be used by the "Secure Mobile Wallet and Payments" use-case.

Nevertheless it has to be stated within WP6, that the 1st round of experimentation and evaluation, was based on SW-TPM only, for all three use cases. In the current project phase, as HW TPM and vTPM become more and more available, a 2nd round of experiments will show the adaption of the use cases to their dedicated QR target platforms.

Within WP5, Task 5.1 (Implementation and Evaluation of Software QR TPM) dealt with the development of the QR SW-TPM. Task 5.2 (Implementation and Evaluation of Virtual QR TPM) and Task 5.3 (Implementation and Evaluation of Hardware QR TPM) are devoted to the design, implementation and evaluation of the QR algorithms in a VM-based TPM and a HW-TPM demonstrator, respectively. Furthermore, Task 5.4 deals with the development of a TPM Software Stack (TSS) API that covers the newly introduced QR cryptographic algorithms and, finally, in Task 5.5 a HW coprocessor to accelerate lattice operations is going to be developed.

1.3 Deliverable Structure

This deliverable is structured as follows:

- **Chapter 2** provides an update of progress on the software TPM that has been made since the submission of D5.3 in M27.
- **Chapter 3** details the concept of the second version of the quantum-resistant VM-based TPM and gives an overview of the current state of its development.
- **Chapter 4** details the design of the final version of the quantum-resistant HW-based TPM demonstrator.
- **Chapter 5** concludes the deliverable and provides a roadmap for the next steps.

The following D5.4 updates are reflected in this document:

Consolidated, public summary of TSS and TPM environments, based on D5.1-D5.3, including detailed summaries for SW- & virtual-TPM, an overview of HW-TPM and a brief description of PQC HW Acceleration

SW-TPM

• Adapted TSS to reflect PQC algorithms of QR TPM

Virtual-TPM

- Derived VM-TPM-specifics from SW-TPM implementation
- Integrated BIKE and Rainbow into QR-Libtpms
- Integrated SW-TPM/QR-Libtpms with QEMU/KVM to provide V-TPM functionalities

HW Demonstrator

- IFX to provide INDEV/UBITECH with SW simulator & HW TPM
- Use of TSS over TPC/IP as main interface for the integration
- Use HW-TPM FW compiled for PC platform for use case integration (PAY)
- Description of PQC Hardware Acceleration

Chapter 2 Update on the SW-TPM

This chapter gives an incremental update on the changes performed on the Software TPM and TPM Software Stack since the previous deliverable D5.3.

This release includes appropriate fixes for all the bugs identified during the integration and experimentation activities of WP6 [12]. It also extends the NTTRU capabilities with general data encryption and decryption. Furthermore, the previous four SW-TPM endpoints for key encapsulation and decapsulation under Kyber and NTTRU were merged into two endpoints given their functionality.

The TSS was updated accordingly to support the new endpoints for key encapsulation and decapsulation. Currently, the TSS allows the generation of encrypted salted sessions under Kyber or NTTRU. Additionally, all regression tests have been extended to support the new endpoints.

2.1 New Endpoints

In the deliverable D5.3 the NTTRU lacked functionality for general data encryption and decryption. This has been implemented and thoroughly tested in this release. Thus, following the guidelines provided in D5.1 [13] the new commands for NTTRU are:

The **TPM2_NTTRU_Encrypt** encrypts a user-provided plaintext, with the following parameters:

- **Input**: key_handle for a loaded key handle to a NTTRU public key and the user's plaintext message (the message has a maximum size of MAX_DIGEST_BUFFER).
- **Output**: a ciphertext generated by the TPM.

The **TPM2_NTTRU_Decrypt** command decrypts a ciphertext generated by TPM2_NTTRU_Encrypt, with the following parameters:

- **Input**: key_handle must reference a loaded private NTTRU key and a ciphertext generated by TPM2_NTTRU_Encrypt.
- **Output**: the TPM will decrypt the cipher object to obtain the original plaintext message.

The TSS API already supports the new NTTRU endpoint for general data encryption (nttruencrypt) and decryption (nttrudecrypt).

- nttrudecrypt => a NTTRU-only command where a decryption is performed using the selected loaded key and an input is received to be encrypted. Its usage is:
 - ./nttrudecrypt -hk LOADED_KEY_HANDLE -ie IN_CIPHER -od OUT_PLAIN

Additionally, new endpoints for key **encapsulation** and **decapsulation** were added to the SW-TPM and TSS replacing the previous commands **TPM2_NTTRU_Enc**, **TPM2_NTTRU_Dec**, **TPM2_Kyber_Enc**, and **TPM2_Kyber_Dec**, with the commands bellow.

The **TPM2** _**Enc** command encapsulates a shared secret, with the following parameters:

- **Input**: key_handle for a loaded public key, either under Kyber or NTTRU.
- **Output**: a shared secret and a ciphertext generated by the TPM.

Respectively, **TPM2_Dec** decapsulates a shared secret, with the following parameters:

- **Input**: key_handle must reference a private loaded key and a ciphertext generated by TPM2_Enc.
- **Output**: the TPM will decapsulate the cipher object to obtain the shared_secret.

To reflect the additions of these commands to the SW-TPM the TSS API was updated by calling the appropriate methods where required and with the commands bellow for direct encapsulation and decapsulation.

- encapsulate => a command where an encapsulation is performed using the selected loaded key. An example of its usage is:
 - ./encapsulate –hk LOADED_KEY_HANDLE -c OUT_CIPHER -ss OUT_SHARED_SEC
- decapsulate => a NTTRU-only command where a decapsulation is performed using the selected loaded key and a ciphertext previously generated by the nttru_enc command. An example of its usage is:
 - ./decapsulate -hk LOADED_KEY_HANDLE -c IN_CIPHER -ss OUT_SHARED_SEC

The extension of NTTRU to support general data encryption and decryption and the new endpoint for key encapsulation and decapsulation completes all agreed upon required algorithms by the QR-TPM and advances one of the still open issued of the deliverable D5.3.

2.2 Miscellaneous SW-TPM Updates

This session presents a summary of the actions taken to fix the open issues since the last public deliverable D5.1 and also a minor fix from D5.3.

Regarding the updates since D5.1:

- Dilithium and Kyber implementation instances were both updated to their latest versions from the NIST submission (round 2) [14][15] and the changes required by the current versions are backwards-compatible with the previously described TPM stack versions in D5.1 [13].
- Two notable bugs were found during the implementation and integration in the V-TPM environment. The first related to generation of a key from a predetermined seed which it stemmed from the fact that the seed provided by the user was ignored during the key generation process. The fix entailed calling the PRNG state using the seed instead of ignoring it. The second bug was related to authenticated sessions using Kyber keys. This bug caused inconsistencies across libtpms and the SW-TPM. Fixing this bug required updating the incorrect functions. After merging theses fixes the QR-TPM stack was successfully deployed across all use cases in WP6, where its results were published in D6.3 [12].
- Support for QR-TLS was added using the TPM as a cryptographic engine. The following example demonstrates the usage of the OpenSLL engine. The command "tpm2tss-genkey" creates a key-pair which is used by the command "openssl" to generate a self-signed certificate.

\$ tpm2tss-genkey -a dilithium dilithium.tss \$ openssl req -new -x509 -engine tpm2tss -key dilithium.tss keyform engine -out dilithium.crt

To accept TLS HTTP connections on port 8443, supported on Kyber and Dilithium, a server is instantiated with

```
$ openssl s_server -cert dilithium.crt -key dilithium.tss -keyform
engine -engine tpm2tss -accept 8443 -www
```

Then a client can connect to server by executing

```
$ echo "GET index.html" | openssl s_client -engine tpm2tss -connect
localhost:8443
```

Thus, a key-exchange supported on Kyber takes places, and the server authenticates itself with a Dilithium signature.

Finally, the SW-TPM was updated from D5.3 to verify the secret size before calling the general decryption methods of Kyber and NTTRU. This prevents the SW-TPM of trying to decrypt an oversized secret. A secret that does not fit into the predefined space causes the SW-TPM to crash with a stack overflow error.

2.3 Resolved and Open Issues

The following issues have been resolved since the last public release D5.1 of the SW-based TPM implementation:

- The regression test suite has been updated and there is no integrity error in the Salt tests for Kyber.
- Kyber and Dilithium were updated to the latest version submitted in NIST competition.
- There won't be a new API endpoint for SHAKE hashes. The consortium decided that it would be best to invest more time in the remaining L-DAA issues and supporting the consortium members in WP6 (this issue was originally described in D5.1).
- The modified OpenSSL broke backwards-compatibility with other TLS ciphers when running certain regression tests. Therefore, similarly to the SHAKE endpoint, it was decided by the consortium that the failing regressions tests in the OpenSSL test suite will not be considered.

This deliverable fixed the following open issues of D5.3:

• The functionality of the NTTRU algorithm has been fully extended on the SW-TPM and TSS to support general data encryption, decryption, and salted sessions.

However, the following challenges still remain "active":

 The L-DAA issues have not been updated. Nonetheless, the consortium is working on a new L-DAA proposal planned to overcome the issues opened in D5.1 [2]. The new implementation of L-DAA will be detailed in the release of WP2.

Chapter 3 Virtual TPM Overview

This chapter reports on the design of the first version of the Quantum-resistant VM-based Trusted Platform Module (TPM), QR Virtual-TPM or QR VM-TPM. The differences between this chapter and the corresponding chapter in D5.3 are as follows:

- Section 3.1.5 has been updated to report that:
 - We have resolved the problem with the TIS buffer size and have tested some TPM commands on the V-TPM with performance measurements taken.
 - We have successfully integrated SW-TPM/QR-Libtpms with QEMU/KVM and tested Kyber and Dilithium using the "Alice and Bob" protocol as described in the D5.1 document.
- Section 3.2.4 has been updated to report that we have successfully compiled BIKE into QR-Libtpms
- Section 3.2.5 has been updated to report that we have commenced the testing of SPHINCS+, Rainbow and BIKE but we have encountered a technical issue.
- Section 3.3 (Future Work) has been removed because the content has been integrated into Section 3.2.5.
- Appendix D has been updated to add details of integration of BIKE into QR-Libtpms.
- Appendix E has been added to report an issue with TSS library when invoking QR functionalities.

3.1 QR Virtual-TPM Architecture

3.1.1 Choice of QEMU

Several methods of VM-TPM have been proposed and developed, however, none of them entirely fits with the requirements of the FutureTPM framework. A comparison of VM-TPM feasibility has already been done in the literature [18], and an overview will be given in the remaining sections of this chapter. Three implementation activities on two systems have been proposed, with emulations on **Xen** and **QEMU** (short for Quick EMUlator) and a pass-through for Xen. [10] QEMU is the chosen emulator, and KVM is the chosen virtualization layer. In particular, QEMU pass-through makes use of the HW-TPM of the host, and strongly links a single guest to the host. Therefore, when used as a pass-through, this approach is infeasible in a cloud-based implementation where multiple guests are offered their own TPM functionality. Equally, it makes migration of guests infeasible, as the HW-TPM state cannot be migrated. Instead, both of these functionalities are possible when, instead of using the pass-through functionality, QEMU uses a TPM emulator (SW-TPM) as backend. QEMU with TPM emulation currently exists, in particular by exploiting Libtpms¹ (see next section for more information) to emulate a TPM device, along with a modified TPM Interface Specification (TIS) on the guest BIOS. Migration is possible in this situation as the TPM state is saved along with the VM image. Each guest has its own emulated TPM, and therefore states cannot be shared between VMs.

¹ <u>https://github.com/stefanberger/libtpms</u>



Figure 3: Architecture of V-TPM

3.1.2 Choice of QR-Libtpms

The libtpms is a library to support emulation of TPM 1.2 and TPM 2.0. It is implemented independently of a backend for storage and an interface for communication. It is thus a flexible library for the integration of TPM functionality into hypervisors, primarily into QEMU. A Libtpms with QR cryptographic functionality included (QR-Libtpms) is used in this chosen architecture. This QR-Libtpms includes implementations of Dilithium, Kyber and L-DAA. We have identified the modifications that have been made to the Libtpms for the purpose of adding new algorithms, i.e. SPHINCS+, Rainbow and BIKE.

3.1.3 Implementation of Virtual TPM

In this architecture (see Figure 3), a Virtual Machine (VM) is connected using a driver to a TPM emulator (based on QR-Libtpms) by using TPM-TIS buffer. Each VM contains a TSS QR, which is the modified version of the TSS of IBM with the new functions which have QR-algorithms added in the virtual TPM code such as Kyber and Dilithium.

The functions (*createprimarykey, signed, etc*) executed by the TSS, are captured by the Virtual-TPM driver. The Virtual-TPM driver is created by the hypervisor QEMU when we power on the VM with this command:

```
qemu-system-x86_64 -display sdl -accel kvm  -m 1024 -boot d -bios
bios-256k.bin -boot menu=on  -chardev
socket,id=chrtpm,path=/tmp/mytpm1/swtpm-sock  -tpmdev
emulator,id=tpm0,chardev=chrtpm  -device tpm-tis,tpmdev=tpm0
virtualmachine.img
```

This command creates in the VM these two components:

- /dev/tpm0 → TPM Device Driver
- /dev/tpmrm0 → multi-process entry point.

Please note that the TPM emulator must be started before trying to access it through QEMU. This TPM emulator implements two components:

- 1. Command *channel* for transferring TPM commands and responses.
- 2. Control *channel* over which we can use to reset, modify or initialize the TPM state, among other things.

As previously described, it is mandatory to create an instance of the TPM emulator before QEMU tries to create the Virtual-TPM driver. The next command is an example of an instance of the TPM emulator which is using *UNIX socket* for the communication:

```
mkdir /tmp/mytpm1
./swtpmsocket --tpmstate dir=/tmp/mytpm1 --tpm2 -ctrl
type=unixio,path=/tmp/mytpm1/swtpm-sock --log level=20
```

This emulator is connected to the driver virtual TPM by using TIS.

The consortium was testing the Virtual TPM architecture with QR-Libtpms code (that includes only Dilithium, Kyber and L-DAA). Initially when using the QR algorithms in the QR-Libtpms, the communication between the VM and the Libtpms did not work properly. To resolve this issue, we have modified the TIS buffer size in Kernel of the VM and QEMU. See Section 3.1.5.

3.1.4 Using SW-TPM within V-TPM

In the following, we describe how to use the SW-TPM and V-TPM.

1. In the same host

To use the IBM TSS for TPM 2.0 directly with SW-TPM in the same host, we need to use the following commands:

Start SW-TPM in one terminal:

```
mkdir /tmp/myvtpm
swtpm socket --tpmstate dir=/tmp/myvtpm --tpm2 --ctrl type=tcp,port=2322
\
    --server type=tcp,port=2321 --flags not-need-init
```

Execute indicative operations with the QR-TSS stack in another terminal:

```
export TPM_COMMAND_PORT=2321 TPM_PLATFORM_PORT=2322 \
   TPM_SERVER_NAME=localhost TPM_INTERFACE_TYPE=socsim \
   TPM_SERVER_TYPE=raw
tssstartup
tsspcrread -ha 10
count 1 pcrUpdateCounter 21
```

To reset SW-TPM run the following command:

swtpm ioctl -i --tcp :2322

2. Using a TSS in a VM and V-TPM

Start SW-TPM in one terminal in the host machine:

```
mkdir /tmp/mytpm1
./swtpmsocket --tpmstate dir=/tmp/mytpm1 --tpm2 -ctrl
type=unixio,path=/tmp/mytpm1/swtpm-sock --log level=20
```

Each VM has installed a TSS QR which is a modified version of the TSS of IBM with new functions with the aim of checking the QR algorithms added in the virtual TPM such as Kyber and Dilithium.

The functions are executed by the QR-TSS are captured by the Virtual-TPM driver, which is created by the hypervisor QEMU when we power on the VM with this command:

qemu-system-x86_64 -display sdl -accel kvm -m 1024 -boot d -bios bios-256.bin -boot menu=on -chardev socket, id=chrtpm, path=/tmp/mytpm1/swtpm-sock -tpmdev emulator, id=tpm0, chardev=chrtpm -device tpm-tis, tpmdev=tpm0 virtualmachine.img

The consortium has created a Dockerfile to be used to test in local the SW-TPM and QR-Libtpms and QR-TSS. The technical steps to create a virtual environment using a Dockerfile are available in a separate whitepaper.

3.1.5 QR V-TPM

There were issues with the TIS buffer size which had to be edited to allow larger cryptographic keys to be allocated correctly. This was amended by editing the following file:

./qemu/hw/tpm/ tmp tis.h

where TPM TIS BUFFER MAX is the variable which needs to be changed.

The buffer size had to be changed to 200 KB in order to accommodate the largest cryptographic keys being used by the consortium. In making the file, the flag --ignore-errors needs to be included

to allow the file to be compiled, otherwise an unsigned int errors can occur. It should be noted that the latest version of GCC needs to be installed on the device before QEMU is made².

After successful compilation, the modified QEMU was created and the other steps were made to boot the QEMU image with the V-TPM running, commands can be issued to the V-TPM, some issues with performance were noted as tss startup command took slightly longer as previous with 0.003817601 seconds time elapsed to initiate the command. The increased buffer size should address the previous issue with regards to the keys being larger than the allocated buffer size. Key generation commands worked with increased key size with no major impact on the performance.

Some TPM commands where tested on the V-TPM to see the functionality and the how well they worked the tests where benchmarked with the following *perf.* The results are shown in Table 1...

Command	Purpose	Time Elapsed
tssstartup	Initiates startup to the V-TPM	0.005564852 seconds
tssstartauthsession	Starts session with the V-TPM	0.004963542 seconds
tsshierarchychangeauth	Prevents rouge users changing policy of the V-TPM	0.004920123 seconds
tssdictionaryattackparameters	Simulates a dictionary-based attack against the V-TPM	0.004927550 seconds
tssgetcapability	Used top display what the TMP can do such as display what algorithms it can handle and use.	0.003924411 seconds

Table 1: V-TPM Performance Results

The V-TPM has been integrated with the QR-SWTPM with the use of a Docker container by allowing KVM to run inside it for the V-TPM to correctly work. Running the V-TPM within Docker requires some extra libraries to enable KVM to run within the container so the V-TPM can run in the hypervisor³. Table 2: Testing and Timings of the V-TPM within the Docker Container shows the timing results of running the commands in the container. These commands are based on the Alice and Bob protocol described in the D5.1 document.

Command	Timing
/.kyberencrypt -hk 80000001 -id	real 0m0.257s
test.txt -oe enc.bin	user 0m0.142s

² It can be noted during compilation, some (unrelated) errors may still occur, and you may be asked to submit a bug report, but this does not affect the V-TPM.

³ See: <u>https://blog.scottlowe.org/2017/11/24/using-docker-machine-kvm-libvirt/</u>

	sys 0m0.107s
./sign -hk 80000001 -dilithium -	real 0m0.265s
if enc.bin -os sig.bin -pwdk	user 0m0.136s
dilithium	sys 0m0.120s
./load -hp 80000000 -ipr	real 0m0.257s
kyber_priv.bin -ipu kyber_pub.bin	user 0m0.139s
-pwdp sto	sys 0m0.111s

Table 2: Testing and Timings of the V-TPM within the Docker Container

Figure 4 and Figure 5 show the output and the timing of two QR commands issued within a VM.

[root@f18059b	e96a3 utils]#	time ./k	yberencrypt	-hk 8000000	1 -id	test.txt	-oe	enc.bin
real OmO.2 user OmO.1 sys OmO.1	58s 35s 15s	-dilithiu	m -if enc.b:	Ln -os sig.b	in -p\	wdk dilit	hium	

Figure 4: Output of Kyber Encrypting a File in a VM



Figure 5: Output of Signing Operation on a File in a VM

We have found an issue with TSS library when invoking QR functionalities. See Appendix E for more details. This issue is of non-deterministic nature, so it affects the functionalities only during some of the runs, while overall the V-TPM functionalities are fine.

3.2 QR Cryptographic Algorithms in V-TPM

The consortium has followed the steps described in the D5.1 document on adding new algorithms.

The integration plan is aimed at adding the following QR cryptographic algorithms into QR-Libtpms:

- SPHINCS+
- Rainbow
- BIKE

3.2.1 SPHINCS+

SPHINCS+ is a stateless variation of the stateful XMSS hash-based signature scheme. The distinguishing characteristic of this signature scheme is that its security is based only on the hardness of symmetric primitives. The signature size is around 30KB, which is acceptable in many scenarios. The main downside of the scheme is that it is slow and resource-intensive. In applications where speed and, to a lesser extent, signature size are not very important and there is a general consensus within the cryptographic community that it will be standardized by NIST [21].

3.2.1.1 SPHINCS+ Chosen Parameter Set

The consortium has examined six parameter sets put forward by the SPHINCS+ working group. For a security level, there are tradeoffs between signature size (s) and speed (f). These are shown in Table 3.¹

SPHINCS+	n	h	d	log(t)	k	w	bitsec	Sec level	Sig bytes
-128s	16	64	8	15	10	16	133	1	8 080
-128f	16	60	20	9	30	16	128	1	16 976
-192s	24	64	8	16	14	16	196	3	17 064
-192f	24	66	22	8	33	16	194	3	35 664
-256s	32	64	8	14	22	16	255	5	29 792
-256f	32	68	17	10	30	16	254	5	49 216

Table 3: SPHINCS+ Parameter Sets

For each sec level, one size-optimised (ending on 's' for "small") and one speed-optimised (ending on 'f' for "fast") parameter set.

The consortium has chosen to use the "-128f" parameter set which has 128 bitsec and security level 1 (second row from the top).

The SPHINCS+ "-128f" has the following parameter set:

n : the security parameter = 16
h : the height of the hypertree = 60
d : the number of layers in the hypertree = 20
t : the number of leaves of a FORS tree
log(t) = 9
k : the number of trees in FORS = 30
w : the Winternitz parameter = 16
Bits per second = 128
Signature in bytes = 16976

It makes sense to choose security level 1 because NIST has 5 security categories for security evaluation of PQC algorithms, and NIST recommends focusing on categories 1, 2 and 3 (see Table 4).[17]

NIST Category	Description
1	Attack with similar complexity of those required for breaking AES-128
2	Attack with similar complexity of those required for collision search SHA256/SHA3-256
3	Attack with similar complexity of those required for breaking AES-192
4	Attack with similar complexity of those required for collision search SHA384/SHA3-384
5	Attack with similar complexity of those required for breaking AES-256

Table 4: NIST Categories

In other words, SPHINCS+ implementation has:

- Public Key Size = 32 Bytes
- Secret Key Size = 64 Bytes
- Signature Size = 16976 Bytes
- Signed Message Size = Signature Size + Message Size

To implement SPHINCS+, the consortium has used the standard codebase from the SPHINCS+ site: <u>https://sphincs.org/</u> The SPHINCS+ code was unit tested. The unit test results demonstrated that SPHINCS+ key generation, signature generation and verification are working correctly which provided the assurance that the SPHINCS+ code is fit for purpose. To implement SPHINCS+ in SW-TPM and TSS, the consortium has made the relevant code changes. For more details, see Appendix A.

3.2.2 Rainbow

Rainbow is a multivariate digital signature scheme. It is a generalization of the Unbalanced Oil and Vinegar (UOV) structure. This design allows parameterizations that are more efficient at the cost of additional algebraic structure. The Rainbow signature scheme was analyzed to be EUF-CMA secure utilizing a hash construction with a random salt.

Since the original Rainbow signature scheme was published in 2005, the scheme has been studied with various parameters. NIST commented that the spectrum of Rainbow parameters allows for optimization in a diverse array of use cases. In the NIST second round submission, the key generation algorithm for the original Rainbow scheme was improved, the nine parameter sets in the first round submission was narrowed down to three sets. Two variants of Rainbow signatures were proposed in order to make a trade-off in key size and performance.

It is also commented by NIST that a further benefit of Rainbow is that it has also been studied in other contexts, including in lightweight applications. Overall, Rainbow has the advantages of being simple and easy to implement and very fast, but it has the limitation of having large key sizes [22].

3.2.2.1 Rainbow Chosen Parameter Set

In the NIST first round submission, there were nine parameter sets. In the NIST second round submission, the key generation algorithm for the original Rainbow scheme was improved, the nine parameter sets in the first round submission was narrowed down to three sets. These are shown in Table 5.[22]

NIST	Standard Rainbow		Cyclic Rainbow		Compressed Rainbow	
Category	pk KB	sk KB	pk KB	sk KB	pk KB	sk
1/11	149.0	93.0	58.1	93.0	58.1	64B
III/IV	710.6	511.4	206.7	511.4	206.7	64B
V/VI	1,705.5	1,227.1	491.9	1,227.1	491.9	64B

Table 5: Rainbow Parameter Sets

Rainbow Signature sizes are: 48B, 140B, 184B [22]

The consortium has chosen to implement the Standard (or Classic) Rainbow parameter set for NIST Security Category I/II in order to be consistent with SPHINCS+ "-128f" parameter set which has security level 1.

In other words, Rainbow implementation has:

- Public Key Size = 149000 Bytes
- Secret Key Size = 93000 Bytes
- Signature Size = 184 Bytes (Maximum)
- Signed Message Size = Signature Size + Message Size

To implement Rainbow, the consortium has used the Rainbow codebase from the NIST site: <u>https://csrc.nist.gov/Projects/post-quantum-cryptography/round-2-submissions</u>. The Rainbow code was unit tested. The unit test results demonstrated that Rainbow key generation, signature generation and verification are working correctly which provided the assurance that the Rainbow code is fit for purpose. To implement Rainbow in SW-TPM and TSS, the consortium has made the relevant code changes. For more details, see Appendix B.

3.2.3 BIKE

3.2.3.1 BIKE Chosen Parameter Set

BIKE (Bit Flipping Key Encapsulation) is a suite of algorithms for key encapsulation based on quasicyclic moderate density parity-check (QC-MDPC) codes that can be decoded using bit flipping decoding techniques.⁹ Key Encapsulation mechanism (KEM) is composed of 3 algorithms: [23]

- GEN outputs a public encapsulation key pk and a private decapsulation key sk,
- ENCAPS takes as input an encapsulation key pk and outputs a ciphertext c and a symmetric key K, and
- DECAPS takes as input a decapsulation key sk and a cryptogram c and outputs a symmetric key K or a decapsulation symbol (unless using implicit rejection).

The BIKE suite consists of 3 variants: BIKE-1, BIKE-2 and BIKE-3. Each variant offers different performance trade-offs [23].

- BIKE-1 has fast, inversion-less key generation and larger public keys (2 blocks).
- BIKE-2's key generation is more expensive but public keys are smaller. BIKE-3's decapsulation invokes the decoding algorithm on a "noisy" syndrome.
- BIKE-3 is fundamentally distinct from BIKE-1 and BIKE-2, mainly in terms of security and security-related aspects like choice of parameters.

To be consistent with SPHINCS+ and Rainbow, the consortium has chosen to implement BIKE-1 at NIST Security Level 1. The BIKE-1 implementation has: [23]

- Public Key Size = 2541 Bytes
- Secret Key Size = 249 Bytes
- Cipher Text Size = 2541 Bytes (Maximum)

To implement BIKE, the consortium has used the standard codebase from the BIKE site: <u>https://bikesuite.org/</u> The BIKE code was unit tested. The unit test results demonstrated that BIKE key generation, encapsulation and decapsulation are working correctly which provided the assurance that the BIKE code is fit for purpose. For more details, see Appendix C.

3.2.4 Integration of SPHINCS+, Rainbow and BIKE into QR-Libtpms

To integrate SPHINCS+, Rainbow and BIKE into QR-Libtpms, the SPHINCS+, Rainbow and BIKE source files, the modified SW-TPM and TSS files were copied into the relevant QR-Libtpms folders and successfully compiled. For more details, see Appendix D.

The newly added algorithms (SPHINCS+, Rainbow and BIKE) have been successfully integrated into the SW-TPM with a Docker container with reference to D5.1 section 3.2.2.

3.2.5 Open Issues

We have started testing the integrated functionalities offered by SPHINCS+, Rainbow and BIKE, successfully integrated into SW-TPM/QR-Libtpms (see Section 3.2) in the context of the V-TPM similarly as above (using the "Alice and Bob" protocol). Because TPM is designed to be "algorithm agile", the SW-TPM and TSS code have been changed to point to run the new algorithms and no changes were made to network commands such as "swtpm".

When configuring the SW-TPM, the "swtpm" command is needed for the user to send commands to the SW-TPM (see Figure 6).

Without this command, no commands can be sent via tss2 and the newly added algorithms cannot be tested. We found an issue when setting the flags "not-need-init". The issue found is a "SetBit" memory issue which has become present when running the SW-TPM, once the command has a terminal response with an error message:

Entering failure mode, mode 4 location SetBit line 98 (see Figure 6)

This was investigated and there were no changes to this from the Kyber integration to the newly added algorithms. A post was made on the Github repository with regards to this issue and its ticket is still waiting for a response.

[root@3c3d4a8a85cd ~]# swtpm socket --tpmstate dir=/tmp/myvtpm --tpm2 --ctrl type=tcp,port=2322,bindaddr=172.17.0.2 --server type=tcp,port=2321,bindaddr=172.17.0.2 --flags not-need-init libtpms/tpm2: Entering failure mode; code: 4, location: SetBit line 98 Segmentation fault (core dumped) [root@3c3d4a8a85cd ~]#

Figure 6: "swtpm socket" command

Chapter 4 HW-TPM Demonstrator Overview

This chapter reports on the design of the quantum-resistant HW TPM (QR HW-TPM).

4.1 QR HW-TPM Demonstrator

The herein described HW-TPM demonstrator is based on a fork of the open-source implementation of the SW-TPM 2.0 produced by IBM and Microsoft running bare-metal on a Cortex-M3 CPU synthesized for an FPGA evaluation platform. It comes with added support for new QR algorithms, namely the key encapsulation mechanism (KEM) NewHope [1] and the digital signature scheme (DSS) BLISS [4].

The following discusses the architecture of the HW-TPM demonstrator, the implementation of a minimal OS environment necessary to run the modified QR SW-TPM on the FPGA, the implementation of the QR algorithms BLISS and NewHope on the HW-TPM demonstrator. Moreover, an outline for concepts on hardware acceleration of cryptographic primitives is provided.

The main reasons for the chosen architecture are:

- the ability to integrate the accelerator that will be built as part of D5.5
- the advantage to not be bound by the limitations posed by existing platforms, especially with regards to available memory
- the analysis and testing possibilities for all details of the platform. This would allow for more detailed lessons learnt and guidelines on the considerations for possible integrated HWbased TPM implementations as will be generated during WP6.

4.2 FPGA Evaluation Platform

The ARM DesignStart Eval platform is used for the evaluation of the QR HW-TPM. It is optimized for the DesignStart MPS2+ board and well-suited for rapid prototyping. The MPS2+ board is supplied with fixed encrypted FPGA implementations of all the Cortex-M processors integrated into a prototype System-on-Chip (SoC). In addition, DesignStarts includes the encrypted netlist of an ARM Cortex-M3 processor with modifiable source code for a SoC with useful peripherals like PSRAM, Ethernet, Audio, VGA, SPI and GPIO. A simplified block diagram of the system is provided in Figure 7. The Cortex-M3 is closely related to the ARM Secure Core (SC) SC300, which also implements the ARMv7 instruction set and additional security functions. The SC300 is used in commercially available security microcontrollers. Usage of the prototyping SoC thus allows deriving general statements about the resource requirements of the TPM software and implementations of cryptographic software (e.g. memory footprint, execution time, etc.) for SC300/Cortex-M3. Due to the similarities between SC300 and Cortex-M3 and with the accessibility of the platform, without a non-disclosure agreement (NDA), it is assessed as a suitable prototyping platform. When an unmodified bitstream containing the prototype SoC with the ARM Cortex-M3 processor is loaded to the FPGA together with program code (i.e., a hex file), it directly allows evaluation of the resource requirements and performance of the software implementation. Moreover, it is possible to modify the source code of the prototype SoC subsystem. This way coprocessor implementations can be integrated into the SoC and then get loaded onto the FPGA. For this the Advanced High-performance Bus (AHB) already contains a spare save that can be easily replaced by an accelerator (see Figure 4). Of course, a DesignStart environment cannot be shipped as a HW-based TPM solution due to the cost of the FPGA and hardware requirements of the relatively powerful SoC. An actual QR HW-TPM does most likely not require Ethernet, Audio, or VGA. Thus, DesignStart's main purpose is to provide an evaluation platform that is accessible and easy to program and use but that still allows to derive information and data relevant for actual hardware implementations.



Figure 7: Simplified system overview on the DesignStart SoC

4.3 Architecture of the HW-TPM Demonstrator

As already pointed out, an FPGA platform running a typical embedded Processor is used to build the QR HW-TPM. To build its firmware, a fork of the IBM SW-TPM extended with QR algorithms (the "QR SW-TPM") is being used. Several adaptions were necessary to make this fork run in a bare metal environment – we will point them out subsequently. A general overview on this is given in Figure 8. As with the other QR TPM variants developed within FutureTPM, communication takes place via TCP/IP. Thus, the IBM SW-TPM running on the FPGA development board has been supplemented by a minimal operating system (OS) environment comprised of a TCP/IP stack and a scheduler that switches between the TCP/IP stack and the QR SW-TPM, which is depicted in Figure 9. For evaluation purposes we make use of the Ethernet Stack as it is easier to integrate than SPI. The Ethernet peripheral is part of the DesignStart FPGA design and low-level drivers are available.



Figure 8: General overview of the QR HW-TPM and comparison to the IBM SW-TPM

TPM



Figure 9: Block diagram of the QR HW-TPM demonstrator

To get this running, several adaptations were necessary:

The IBM SW-TPM had to be ported to the FPGA platform. Most importantly, as the IBM SW-TPM depends on the OpenSSL library for cryptographic operations and OpenSSL cannot be built for the chosen FPGA platform, the OpenSSL library had to be replaced by the WolfCrypt⁴ open-source cryptographic library.

⁴ https://www.wolfssl.com/products/wolfcrypt-2/

- As TCP/IP stack the so-called BSD-socket-based A Light-weight TCP/IP stack (IwIP)⁵ was chosen. Again, this implementation had to be ported to the FPGA platform. Moreover, it was necessary to adapt the IBM SW-TPM implementation to this TCP/IP stack and to intertwine both through callbacks.
- Because the lwIP stack needs to execute housekeeping functionality regularly, a minimalistic scheduler was implemented that allows for the IBM SW-TPM and the lwIP stack to run in parallel as two threads.
- Moreover, Ethernet drivers were ported to the chosen FPGA platform and compilation errors in the IBM SW-TPM implementation were corrected.
- A method to support non-volatile memory (NVM) emulation was implemented: As the used FPGA board is not equipped with NVM, encapsulation keys have been defined statically within the QR SW-TPM source code, which is sufficient for the purpose of demonstration.

Apart of these fundamental developments, the HW-TPM demonstrator has – similar to the other FutureTPM TPM platforms – been extended with the QR algorithms NewHope and BLISS. Details on that will be given subsequently.

4.4 Basic Operation of the QR HW-TPM Demonstrator

As already outlined, the communication with the QR HW TPM demonstrator is similar to the one described in D5.1 for the SW-TPM. It requires the usage of sockets to establish a TCP/IP connection between the QR SW-TPM running on the FPGA platform and a client. Through this connection, the QR HW TPM mimics the physical TPM command transmission interface (TCTI) layer found in the physical TPM.

After receiving data, the QR HW-TPM demonstrator executes the requested command by validating the session, its internal state, the command code, and its key handles. Then, the remaining data is forwarded to the Command Dispatcher where the unmarshalling functions are selected from the decoded command code and its data is unmarshalled. Finally, the command is executed. If there is any data to be returned to the client, the same procedure is applied in reverse order. The marshalling functions are obtained from the command code; the data is marshalled and then sent back to the client through the same TCP/IP connection.

4.5 New Endpoints

To support new QR algorithms, the following endpoints have been implemented in the QR HW-TPM demonstrator.

- NewHope512-CCA:
 - Key Generation: to generate a NewHope key one must use the standardized TPM2_Create, TPM2_CreateLoaded and TPM2_CreatePrimary functions, already provided by the TPM2 specification, with the TPM_ALG_NEWHOPE value and the corresponding security mode *k*.
 - TPM2_NEWHOPE_Enc encapsulates a shared secret, with the following parameters:
 - Input: *key_handle* for a loaded key handle to a NewHope public key.
 - Output: a *shared secret* and a *ciphertext* generated by the TPM.

⁵ <u>https://savannah.nongnu.org/projects/lwip/</u>

- TPM2_NEWHOPE_Dec decapsulates a shared secret, with the following parameters:
 - Input: key_handle must reference a private loaded NewHope key and a ciphertext generated by TPM2_NEWHOPE_Enc.
 - Output: the TPM will decapsulate the cipher object to obtain the shared_secret.
- BLISS-I⁶:
 - Key Generation: to generate a BLISS key one must use the standardized TPM2_Create, TPM2_CreateLoaded and TPM2_CreatePrimary functions, already provided by the TPM2 specification, with the TPM_ALG_BLISS value and the corresponding security mode *mode*.
 - Signing: to sign a digest using the QR HW-TPM, the user must use the following functions: TPM2_Sign and TPM2_Quote.
 - Signature Verification: to verify a BLISS signature, the user must use the TPM2_VerifySignature function.

4.6 How to add new algorithms

As the HW-TPM demonstrator is based on a fork of the IBM SW-TPM, the strategy discussed in D5.1 for the addition of new algorithms applies here.

4.7 Using the HW-TPM demonstrator

The QR HW-TPM demonstrator FPGA board offers an RJ45 Ethernet interface. However, note that its TCP/IP implementation does not support the network management protocol DHCP. Thus, its address is fixed to 192.168.0.16.

⁶ The BLISS variant GALACTICS was used because of its improvements over the original BLISS paper in that sampling is constant time, improving security. <u>https://eprint.iacr.org/2019/511.pdf</u>

After connecting the FPGA board to the power supply, the board automatically boots the QR SW-TPM and listens on the same TCP/IP ports as the IBM SW-TPM, that is, port 2321 for the command server and port 2322 for the platform server. Like the IBM SW-TPM, it is addressable by the TSS implementations. The basic structure is depicted in Figure 10.



Figure 10: Architecture of IBM's TPM2.0 SW implementation (from <u>http://ibmswtpm.sourceforge.net/</u>).

4.8 Example

This subsection will give an example, where Alice (the QR HW-TPM) communicates with Bob (a local QR SW-TPM) to transmit an encrypted and authenticated message using the QR HW-TPM demonstrator. The idea behind this example is motived by the following: Unlike key agreements, that are well-known from the world of classical cryptography (e.g. Diffie-Hellman key agreement [3]), where both participating parties contribute to a same extent to the exchanged secret, KEMs like NewHope are more like a one-way road. There, the sender fully determines the exchanged secret. However, using several runs of an IND-CCA-secure KEM like NewHope, we can mimic an authenticated (perfect-forward secret) key agreement:

First, Alice and Bob obtain long-term KEM keys and exchange their public keys. Second, Alice uses NewHope to send a secret s1 to Bob using Bob's long-term public key. Likewise, Bob sends a secret s2 to Alice encrypted under her long-term public key. The third step ensures the connection enjoys perfect forward secrecy. Alice generates a NewHope ephemeral key-pair and sends the resulting public-key to Bob. Bob then sends another secret s3 to Alice using Alice's ephemeral key. At the end, Alice and Bob can create a session key by hashing the concatenation of s1, s2 and s3. Using this session key, Alice encrypts a message, additionally signs it using her BLISS private key and sends the ciphertext together with the signature to Bob. Bob checks the signature and on success decrypts the ciphertext using the previously agreed session key.

Figure 11 depicts the communication diagram.



usage steps

The use-case is run using the TSS command line binaries, extended to support the PQ algorithms.

4.9 TSS Adaptations

As described above, the TSS is the piece of software on a client that will communicate with the TPM. As such a modified TSS that supports NewHope and BLISS was needed for the QR HW-TPM Demonstrator.

Similar to the approach chosen for the QR HW-TPM Demonstrator that is based on an open-source implementation published by IBM, the TSS is split into a library part that handles communication, and individual binaries that each implement calling of a single TPM command.



Figure 12 TSS architecture / parts

The communication part of the TSS could remain the same it already implements the TPM command transmission interface used by the QR HW-TPM Demonstrator.

The changes that were needed comprise mostly of support for the new algorithm IDs that are used to select the NewHope or BLISS algorithm. In detail this means:

- The create (calling TPM2_Create), createprimary (TPM2_CreatePrimary) and loadexternal (TPM2_CreateLoaded) executables needed to be extended by command line switches to use the algorithm IDs for NewHope and BLISS, and accept the according response packets. Data structures sent by the TPM are stored 1:1 in files for any output data.
- Similarly, *sign* (TPM2_Sign) and *verify* (TPM2_Verify) needed to be extended to allow usage of the BLISS algorithm ID.
- The NewHope integration is built in two new commands, TPM2_NEWHOPE_Enc and TPM2_NEWHOPE_Dec. For these two commands two new executables *NewHope_Enc* and *NewHope_Dec* where added. Both commands take parameters in accordance with other executables to write their output parameters to files or the standard output.

With these changes, all commands and command parameters that were added to the QR HW-TPM demonstrator can be used. To do so one has to define the TPM_SERVER_NAME environment variable to be set to the IP address of the demonstrator:

\$ export TPM_SERVER_NAME="192.168.0.16"

With this setting, any of the executables can be called to send the according command to the QR HW-TPM demonstrator, as seen above in the description of the example use-case.

4.10 Hardware Acceleration of PQC

PQC routines can be accelerated in hardware with various degrees of flexibility. We classify hardware-based PQC accelerators in the dimensions: coupling, algorithmic flexibility, and extent of acceleration. In Figure 13 an abstract microcontroller architecture is provided. The CPU subsystem comprises a registerfile and is connected to a high performance main bus. The main bus is used to read and write the non-volatile memory (NVM) for code and permanently stored data and the SRAM for temporary data. A peripheral bus connects peripherals with lower performance requirements, like

a timer or Universal Asynchronous Receiver Transmitter (UART). The grey components describe various possible options for the placement of hardware-based PQC accelerators. From the locality depicted in Figure 8 the level of coupling an accelerator has to the main CPU can be derived.

- High coupling Instruction Set Extension (ISE): With ISE new instructions are added to the microarchitecture of the CPU that can be used to speedup certain operations of PQC schemes. Usually, the added instructions are stateless to ease integration and have a low overhead as a large part of the CPU subsystem is reused. However, when the CPU executes the instructions from the ISE it cannot perform other tasks. Exemplarily, an ISE may comprise of additional arithmetic instructions for the computation of the NTT butterfly or instructions that facilitate Gaussian sampling.
- Medium coupling Co-Pro: The CPU can be extended by tightly coupled cryptographic coprocessors (Co-Pro) that are optimized for PQC. Such co-processors typically have direct access to the registerfile and may also contain an internal registerfile and state. This direct access leads to decreased memory transfer times. They can thus operate on larger datasets and may reuse hardware of the Core. A Co-Pro could for example implement a full NTT accelerator or SHA-3 core.
- Loose-coupling Accelerator on Bus: An accelerator for PQC may be added to the main bus of the microcontroller or to a peripheral bus if data transfer time is of lower importance. Similar to other peripherals the CPU transfers data from the registerfile/SRAM into the memory address space of the peripheral, starts an operations and then periodically controls the operation, feeds new data or just waits until the operation is finished. After an operation is finished the CPU obtains the result and copies it into the registerfile/SRAM. A peripheral on the system bus can usually execute complex operations independently of the CPU on internal memory. This frees the CPU for other tasks, e.g., I/O and allows integration into a SoC system with minimal dependencies on other subsystems. A loosely coupled accellerator may implement a full NTT accelerator, a SHA-3 core, or even a monolithic implementation of a full PQC scheme (e.g., NewHope).

Another dimension is the level of algorithmic flexibility the accelerator offers. A hardware-based accelerator may support acceleration for

- a single parameter set of a particular scheme (e.g., NewHope-512),
- all parameter sets of a particular scheme (e.g., NewHope),
- several parameter sets of a particular class of schemes (e.g., a binomial sampler for Kyber, NewHope, Saber),
- or may implement functions that can accelerate a wide variety of schemes efficiently.

Moreover, the extent of acceleration is an important additional dimension. An accelerator may

- implement a subfunction of a building block (e.g., a butterfly or a part of the SHA-2 round function),
- one complete building block (e.g., the SHA-3, NTT, or binomial sampling) or multiple building blocks, or
- a full scheme (e.g., NewHope).

With respect to related work, the highest performance is usually achieved by accelerators that are loosely coupled, accelerate a single parameter set and implement a full scheme. In such cases the accelerators can have their own high performance internal memory and are not constrained by the data path of the CPU (e.g., a 32-bit memory bus).

For a QR HW TPM, the coupling of the accelerator is assessed to be of minor importance and depends mostly on the required performance and achievable cost position. The most important property is the flexibility to accelerate at least several parameter sets of a particular class of schemes or wide variety of schemes. This is because of the uncertainties of the NIST competition and additional need for TCG to approve algorithms.



Figure 13 Option for hardware acceleration of PQC

Chapter 5 Conclusions and Next Steps

In this deliverable, we have discussed the concept of the Quantum-Resistant Virtual Trusted Platform Module, which is now ready in a first version by M30. Moreover, we have given an update to the final state of the Quantum-Resistant Software Trusted Platform Module and an overview of the architecture of the final version of the Quantum-Resistant Hardware Trusted Platform Module demonstrator.

The next steps will be to finalize the implementation of the QR Virtual TPM (Task 5.2) and further optimization of L-DAA (Task 5.5). The TPM SW Stack (Task 5.4) is updated to support all PQC algorithms implemented in the three QR TPM environments. In Task 5.5, the HW QR crypto-coprocessor is implemented to accelerate the Ring-LWE schemes NewHope and BLISS in HW. Finally, performance evaluations and testing are done.

Outlook to D5.5 Deliverables

The final deliverable document in M33 will provide a HW-TPM Coprocessor implementation, final optimizations for the SW-TPM environment and last updates on vTPM integration and testing results.

SW-TPM

• Possibly Bug fixes and optimization (esp. L-DAA), after T5.4

vTPM

• Possibly late results on Sphinx+, Bike, Rainbow integration and testing, after T5.4

HW-Coprocessor Demonstrator

- Implementation of HW-Coprocessor demonstrator
- Evaluation of cryptographic coprocessor and crypto performance testing

In case of very late technical implementation results after M33, updates will be documented in the final deliverable document of WP2 at M36.

List of Abbreviations

Abbreviation	Translation
AHB	Advanced High Performance Bus
BIOS	Basic Input Output System
DHCP	Dynamic Host Configuration Protocol
DSS	Digital Signature Scheme
FPGA	Field Programmable Gate Array
нพ	Hardware
NDA	Non-disclosure Agreement
KEM	Key Encapsulation Mechanism
NVM	Non-Volatile Memory
NVRAM	Non-Volatile Random Access Memory
PCR	Platform Configuration Registers
QEMU	Quick EMUlator
QR	Quantum Resistant
SC	Secure Core
SH	Shell
SoC	System on Chip
SW	Software
тсті	TPM command transmission interface
TIS	TPM Interface Specification
ТРМ	Trusted Platform Module
TSS	TPM Software Stack
UC	Use-case
VM	Virtual Machine
VNC	Virtual Network Computing
WP	Work Package

Bibliography

- [1] E. Alkim, L. Ducas, T. Poppelmann and P. Schwabe, "Post-quantum Key Exchange A New Hope," in USENIX Security Symposium, Vancouver, Canada, 2016.
- [2] L. Chen, N. E. Kassem, A. Lehmann and V. Lyubashevsky, "A Framework for Efficient Lattice-Based DAA," in 1st Workshop on Cyber-Security Arms Race (CYSARM'19), London, United Kingdom, 2019.
- [3] W. Diffie and M. E. Hellman, "New Directions in Cryptography," IEEE Transactions on Information Theory. 22 (6), p. 644–654, 1976.
- [4] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler and D. Stehlé, "Dilithium - Submission to the NIST post-quantum project," 30 03 2019. [Online]. Available: https://pq-crystals.org/dilithium/data/dilithium-specification-round2.pdf
- [5] Bindel, N., Akleylek, Alkim, E., Barreto, P., Buchmann, J., Eaton, E., ... Zanon, G. (2019, 26
 4). qTesla Submission to the NIST post-quantum project. Retrieved from NIST: https://qtesla.org/wp-content/uploads/2019/04/qTESLA_round2_04.26.2019.pdf
- [6] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler and D. Stehlé, "CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM," in 2018 IEEE European Symposium on Security and Privacy, EuroS&P, London, UK, 2018.
- [7] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler and D. Stehlé, "Dilithium - Submission to the NIST post-quantum project," 30 03 2019. [Online]. Available: <u>https://pq-crystals.org/dilithium/data/dilithium-specification-round2.pdf</u>.
- [8] N. El Kassem, L. Chen, R. El Bansarkhani, A. El Kaafarani, J. Camenisch, P. Hough, P. Martins and L. Sousa, "More efficient, provably-secure direct anonymous attestation from lattices," Future Generation Computer Systems, Volume 99, pp. 425-258, 2019
- [9] TCG, "TCG Algorithm Registry," 7 2 2018. [Online]. Available: <u>https://trustedcomputinggroup.org/wp-content/uploads/TCG-</u> _Algorithm_Registry_Rev_1.27_FinalPublication.pdf
- [10] T.F. Consortium, "D2.2 Second Report on New QR Cryptographic Primitives," 2019
- [11] T.F. Consortium, "D5.2 Second version of implementation", Christian Hanser, 2019
- [12] T.F. Consortium, "D6.3 Demonstrators Implementation Report First Release", Sotiris Koussouris, 2018
- [13] T.F. Consortium, "D5.1 First version of implementation", Daniele Sgandurra, 2017
- [14] NTTRU: "Truly Fast NTRU Using NTT", Vadim Lyubashevsky and Gregor Seiler, Cryptology ePrint Archive, Report 2019/040, 2019, <u>https://eprint.iacr.org/2019/040</u>
- [15] NIST Round2 submission: CRYSTALS-KYBER, Peter Schwabe and Roberto Avanzi and Joppe Bos and Leo Ducas and Eike Kiltz and Tancrede Lepoint and Vadim Lyubashevsky and John Μ. Schanck and Gregor Seiler and Damien Stehle, 2019. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-2/submissions/CRYSTALS-Dilithium-Round2.zip :
- [16] NIST Round2 submission: CRYSTALS-DILITHIUM, Vadim Lyubashevsky and Leo Ducas and Eike Kiltz and Tancrede Lepoint and Peter Schwabe and Gregor Seiler and Damien Stehle, 2019, <u>https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-2/submissions/CRYSTALS-Dilithium-Round2.zip</u>
- [17] T.F. Consortium, "D2.1 First Report on New QR Cryptographic Primitives", Gagliatdoni, T., 2018
- [18] Cucurull, J and Guasch, S.: "Virtual TPM for a secure cloud: fallacy or reality?" Scytl Secure Electronic Voting, 2014.

- [19] TCG, "Trusted Platform Module 2.0: A Brief Introduction", 2015 [Online]. Available at: <u>https://www.trustedcomputinggroup.org/wp-content/uploads/TPM-2.0-A-Brief-</u> <u>Introduction.pdf</u>
- [20] Aragon, N., Barreto P. S. L. M., Bettaieb S., Bidoux L., Blazy, O., Deneuville J., Gaborit, P., Gueron, S., Güneysu, T., Melchor, C. A., Misoczki, R., Persichetti, E., Sendrier, N., Tillich. J., Vasseur, V. and Zémor, G., "BIKE: Bit Flipping Key Encapsulation", Round 2 Submission 2019. [Online]. Available at: <u>https://bikesuite.org/files/round2/spec/BIKE-Spec-Round2.2019.03.30.pdf</u>
- [21] Aumasson J., Bernstein D. J., Dobraunig C., Eichlseder M., Fluhrer S., Gazdag S., Hülsing A., Kampanakis P., Kölbl S., Lange T., Lauridsen M. M., Mendel F., Niederhagen R., Rechberger C., Rijneveld J. and Schwabe P., SPHINCS+ Submission to the NIST postquantum project, 2019
- [22] Ding J., Chen M., Petzoldt A., Schmidt D. and Yang B. Y., "*Rainbow*", *The 2nd NIST Standardization Conference for Post-Quantum Cryptosystems*, 2019.
- [23] Aragon, N., Barreto P. S. L. M., Bettaieb S., Bidoux L., Blazy, O., Deneuville J., Gaborit, P., Gueron, S., Güneysu, T., Melchor, C. A., Misoczki, R., "BIKE: Bit Flipping Key Encapsulation", Round 2 Submission.
- [24] T.F. Consortium, "D5.3 Final version of implementation", Michael Albrecht, 2020

Appendices

Appendix A: SPHINCS+ Implementation

To implement SPHINCS+, the consortium has used the standard codebase from the SPHINCS+ site: <u>https://sphincs.org/</u> The SPHINCS+ source files were then renamed with a prefix of "sphincsplus" in order to make them identifiably belonging to SPHINCS+ when ported into QR-Libtpms. The "#Include" statements within the source files were also appropriately renamed.

The sphincsplus-Makefile was modified (due to name changes) and executed. It produces an executable, sphincsplus-PQCgenKAT_sign. Before integrating into QR-Libtpms, the SPHINCS+ code was unit tested by executing sphincsplus-PQCgenKAT_sign.

Activiti	es □ Terminal * Mon 11:16 ₹ 4. 🔒
-	uhal005@adminhp-HP-EliteBook-840-G1: -/Documents/SPHINCS+/SPHINCS+ Test Environment A 🛛 💮 🗄
	File Edit View Search Terminal Help
	<pre>uhat053gadnthp=HD=ElteBook=640-01:-/Documents/SPUINCS=Test Environment AS 1s sphincsplus-ht. sphincsplus-hash_address.sphincsplus-hakefile sphincsplus-tors.c sphincsplus-hash_address.sh sphincsplus-hakefile sphincsplus-tors.c sphincsplus-tors.c sphincsplus-tors.c sphincsplus-hash_address.shortsplus-tors.c sphincsplus-tors.c sphincs</pre>
8	sphicesplus-rmg.c: In function 'AE5256_ECG': sphicesplus-reg.c:123:9: warning: variable 'ciphertext_len' set but not used [-Humused-but-set-variable] for tophertext_len; //
▶_	sphincsplus-rng.c: In function 'randomyter_init': sphincsplus-rng.c:142:22: wirning: unused parameter 'security_strength' [-Hunused-parameter] Int security_trength)
	sphincsplus-POggenXI_sign.c: In function 'nain': sphincsplus-POgenXI_sign.c:Bills warning: ignoring return value of 'fscanf', declared with attribute warn_unused_result [-Hunused-result] fscanf(fp_ret, "we', "scunt);
	<pre>sphincsplus-PQCgenKAT_sign.c:104:13: warning: ignoring return value of 'fscanf', declared with attribute warn_unused_result [-Wunused-result] fscanf(fp_req. "%llu", Anlen); ////////////////////////////////////</pre>
	uhl805gednthh=W=EltEBook=840.61:/Documents/SMPINES*Test Environment AS li sphicsplus-aph.h. sphicsplus-hash_address.c.sphicsplus-hakefile sphicsplus-PQGgenkAT_sign sphinsplus-rng.h. sphicsplus-utils.h test sphicsplus-fors.c.sphinsplus-hash_address.h.sphinsplus-hakefile sphinsplus-PQGgenkAT_sign.c.sphinsplus-sign.c.sphinsplus-wots.c. sphinsplus-fors.h.plinsplus-hakefile.sphinsplus-hakefile sphinsplus-PQGgenkAT_sign.c.sphinsplus-sign.c.sphinsplus-wots.c. sphinsplus-fors.h.plinsplus-hakefile.set.splus-splinsplus-hakefile.sphinsplus-rng.c.sphinsplus-wots.h. uhl8058gednthh=w=ElteBook=840-61:-/Documents/SMPINES*TestEartForments/A

Figure 14: SPHINCS+ Compilation

Activit	Terminal - Mon 17:10
6	uhal005@adminhp-HP-EliteBook-840-G1: -/Documents/SPHINCS+/SPHINCS+-Test-Environment-A Edit: View: Search: Terminal: Help
	<pre>10959adrihp=P=1LtBook=840-G1=_/Documents/SPHINCS+/SPHINCS+Test=Environment-A5 1s 10059adrihp=hp=FiltEBook=840-G1=_/Documents/SPHINCS+Test=Environment-A5 1s 10059adrihp=hp=Shincsplus=hash_address.c sphincsplus=hash_ash256.c sphincsplus=PGCpenkAT_sign sphincsplus=rign.c sphincsplus=works.c 10059adrihp=hp=EltEBook=840-G1=_/Documents/SPHINCS+Test=Environment-A5 /splincsplus=PGCgenkAT_sign 10059adrihp=hp=1EltEBook=840-G1=_/Documents/SPHINCS+Test=Environment-A5 /splincsplus=PGCgenkAT_sign 10059adrihp=hp=1EltEBook=840-G1=_/Documents/SPHINCS+Test=Environment-A5 /splincsplus=PGCgenkAT_sign 10059adrihp=hp=1EltEBook=840-G1=_DocumentS/SPHINCS+Test=Environment-A5 /splincSPHINCS+Test=1Environment-A5 /splincSPHINCS</pre>
2	inceplus-api.h sphincsplus-hash.address.c sphincsplus-hash.sha256.c sphincsplus-PCGenATA_sign sphincsplus-PCGEnATA_sign sphincsplus-rg.c sphi
8	
▶_	

Figure 15: SPHINCS+ Unit Test

The unit test produced a response file, sphincsplus-PQCsignKAT_64.rsp which contains 100 test result records, each with a different message length. The following 2 figures show the SPHINCS+ unit test results for record 0 to 4 and for record 95 to 99.

Activitie	s 🥫 Text Editor 🕶		¥41 @ ₹
6	Open 👻 📇	sphincsplus-PQCsignKAT_64.rsp -/Documents/SPIENCS+SPIENCS+Fe2=Environment-A	save 😑 👄 👁 🚳
 	3 SPHINCS- Count = 0 SpHINCS- exect = 0615502140158C5E05595FE0- message = 081C509574051869749031 exessage = 081C1409574801869749031 secret key = 7C993548087948018 secret key = 7C99354807948018 secret key = 7C99354807948018 secret key = 7C99354807948018 secret key = 769354807948018 secret key = 760556628420417664 message = 08158729550628420417664 secret key = 4863280135919645494 secret key = 486328013914674391 secret key = 486328013914674391 secret key = 486328013914674391 secret key = 486328013914674391 secret key = 486328013911678360+ secret key = 486328013911678360+ secret key = 58698 secret key = 58698	IEF7A2576772E24CC28C479009D800C9A8CFDE7856A8C286F98F97ED9854108D2E1FFA1 IFFA2576772E24CC28C479009D800C9A8CFDE7856A8C286F98F97ED9854108D2E1FFA1 IFFA3576772E24CC28C479009D800C9A8CFDE7855A8C2 IFFA3576772E24CC28C479009D800C9A8CFDE7855A8C2 IFFA3576772E24CC28C479009D800C9A8CFDE7855A8C2 IFFA357761254CC2828080F51741111121144 I0864052674C28086F5C51322880457824280FEE50100E48F2195A2801E818794E998E05292025C9A8E10960596EF0811820889F79850740E89AC498394679983003295CAC8320F3054 IA1298064385E7121CA26FC199EC70C3543830557F005C03CF123A450648EF6A43C808 IFF59F7C2954A81E0754610ACA299A80F65C099232M073F78985226225EF98C42227E8A38C1A82568299E4408A961869C6F839838170C049 IFF59F7C2954A81E0754610ACA299A80F65C09923M073F789852258222FF98C2227E8A38C1A82568299E4408A961869C6F839838170C049 IFF59F7C2954A81E0754610ACA299A80F65C09923M073F78985228679852357783713C838EA005542368299E4408A961869C6F839838170C049 IFF59F7C2954A81E07546105557005767775918005720671375780922867897132689860555778713C838EA005542348098AF011450F338005592C8A IFFA33A7F3E597C90685799067576972E0881185E049672522028C48709818E43462851C23277C9388846C12EC4790E3AC0A000085FFF91928F8E886165E09A66C754529C0889856C481 IFFA33A7F3E597C9068579706776972E0881185E0496729E41282020AF09818E43462851C23277C9388846C12EC4790E3AC0A000085FFF91928F8E886165E09A66C754529C08898C644 IFFA33A7F3E597C9068579706776972E0881185E0496729E41282026AF09818E43462851C23277C9388846C12EC4790E3AC0A000085FFF91928F8E886165E09A66C754529C088898F644	4AAC@3212678C@1D@2B7E3E89AK 70F65D95FFE4395657014E6C492 DCF34AA348F82CE88CF
	public key = 4FD6BCF6161023F9DCDE secret key = 1D836E889E462598CD10 signed message length = 17075 signed message = 5AE51C0808FF8D5084D9EF62065C33848	H41F#FD0042573898503168ACA88691C655799D80007 :C028369583C5947CF88919EC2872C280247CB15A55694FD68CF6161023F9DC08461F0FD0042573898909168ACA88691C655799008007 358C8FE08A10095C70582302385C329902A6962C2C6EF8F49EBA97ZF312ED4F66889A41F3E5E9C88C28E573A8230A971C0936C0888D1885A80EAD38804400138608EAA8044898E548481	0FB56E9361662E54D48B23CB291
	count = 3 seed = 38C09402178C13EDFDBEAS7EDE message length = 132 ressage = 27AFSB52A046471EFCD728C93B49198B sublic key = FF009305043EE0F302F slgned message length = 17108 slgned message length = 17108	IF3A336F8F69FED1D54648CE30BCC64847A5C9917C2E28C4D5F626E937F80329FCF8A16 IS5A61CDE8E8891251C5A8B85E828FD36ED9FF9F0F45783ECB1A86728CB8748426ADFF96123C88FAC28C6C58A9C60071761292262C65F200F47751F8831778A68B783768B87F5EFFF86 IY246989698125172180278B808268FC78723591 IB53F23350607321857DB328BA645F8DF3A8900426552FF69385D543EE8F38258746689698E373F5172168028ED60589C26F7C7B2591 S8BE6489115011B566D81122F4E5CAF9668984466CFAE168A21427FEAFF6C6F68FAF22E1DF9F8E347EF84F9C80A0921F99C867E88546FC03725F9848517924E66189CC6404878F218558	E11AC35F353A6F24400B80B287€ SCEA7A017312EA69B9D54DF14AC
	count = 4 escal = F3902A7815F37BC7F5802D8C86 message = COF0A61124780A8FF00318F779A3B866 UC0F0A61124780A8FF00318F779A3B866 uL012c key = 4970299CF6509A8053 secret key = 2CA59C6CF33C53803747 signed message = St26A657782C87F2756809F45DC182A8B	E5848082E8856917189628F88408C06AA4106E903980A1872450AFA4EC189A57332914 1359640659CA7A83FE406EAE9T046428010A88794C6735A8FE8788F409741817872105A394E54F0C5789A788748C246487398AC9865115644C569AE25305285751342574C03346000C191 17692F5A8FA9482FCEE7EF087F8F17135ECC3FF3F07F749FD2F99CF65698A865578697769A80A861821CAAAC650834804C00007434E87 F6292F5A8FA9482FCEE7EF087F8F17135ECC3FF3F07F749FD2F99CF65698A865578697769A80A861821CAAAC650834804C0007434E87	58A6273546616896D0C5ECE6A84 1C93887B8888FD457D099892D8E1
::::		Plain Text 🔻 Tab Width: B 🔻	Ln 1, Col 1 🔹 INS

Figure 16: SPHINCS+ Unit Test Results (Record 0 to 4)

Activit	ies 📑 Text Editor 👻	Thu 23:05		1 41 日 -
(2)	open▼ ≞	sphincsplus-PQCsignKAT_64.rsp -/Documents/DHENCS-FORS-FORS-FORS-FORS-A	Save	=
	Count = 95 seed = 2489C04BA57D149A60F446670C13C29998B52F3BAD548A751D7134B694DB25ABFA834FB4BA45 message length = 3168 message =	SE105AE27D575CBD02899		
	1F7AB96E8C14D1A5094672D7034FA8F81703A2CC18983C972CC66736CD988031AC8A479CED21A1F6349 public key = 1AF7BEAC03F707B1C8708B7E02B8F757E603954120560AC64457273121A03E597 secret key = 09931E321722B082C5C0A10F12BA845498FC703E76A4404871892F149877FFBB1AF7BE	938DF85F3E83161646D881B9AC3EA22F8098088E2E8A4E9975714E5A98985817F426C41F39683496868 EAC03F70781C870887E028BF757E0C95F312050AC64457273121AD3E597	69AF917564A264840188FA127FC	200DC16A9E663
	signed message length = 20144 signed message = 23366A1CC30EE902759F9EE09F8778B609F612660E2CDD8BE9EA916A8DE53780F179C3A053F6B5C33B4	4605F003B7D8AD08513B436D056F581E9A875455B06A4BCED529B3DA5687F8716721ECFE08F068056DE	C0D08145BEC3E8EDF8BA8C3EAA7	4FBA1CB1397155
	count = 96 seed = 26CF86072604DFA38AE07399838B8336F18EE59E9F23AE4C81E73D49964997EF21C85F5412F9	9A78A1EC39FC6228C36CA		
3	Message Length = 3201 message = DE897F02AE7292AAAFA6A0CAD52929113410F28A972B4184E894C4D31081420751560956F49CE287726	635625AFC3CA6698FBFDE4D0A05EF243DF190BA1CE780EB572590E01E6E283E1963F2B072280CEB3655	52F65BD405F1A284DDBED07BA61	24453D30CC28CE
<u>}-</u>	public key = 14(EA/AB/DE/DI4F07-384A0533802462/A95312/29(F84802/E1/E01859) secret key = F834651264A592888BA69884840B1DC0EDF876A93548F27F981968C68880BA3A1114CEA7 signed message length = 20177	7AB20EAED14F07F3B4A65038D24622A9E3122F299CFB486E7E17E0CB591		
- 📈	51g1med message = E7B3459C3AF2B3A68D2446B1B374A1083448B4043D4BB25A0C081978EBCD1A1A191CC4E1DE0B880E12E	E5882BF0E5D866346C22A14CF90F460804726AC3DA7FFB9348751EB8034B070D7062C59E9899B945168	B61C452F9A85A88357088457C22F	37FF9F736F996
	<pre>count = 97 seed = 13F1F44609AA5AC8532788F74C9E6447A6CE4294C037867F43DF554370EE261D05C7260EEBF4 nessage length = 3234</pre>	46D6694D0850B8343F8E5		
	message = 32588990C53664849FFC71EBC953F7A8ECA6298F6AA15A83BF6923805921B1C860BBFC544A39C364EF6 public key = EF086C290018298AE073902C845F6180A1E8486C62328E95672CC690BF733FA2E secret key = Bef334B259800cAECF6F09BAS64EA26A412394543A8A9FE120A47E67F080C7	609281481E946C994F96829D6639727A534556808641E9A516F913F7FE5592C2A40C8278F5AF08D4504 C29D018298AE673902CB45F6180A1E8486C023E0E96F2CC800B7633FAZE	B5387C20945654F08168247A98F	i6A43A5020955F
	signed message iength = 20210 signed message = 735173823224A1AC9D911E68B5C4CE3E83AC9A65710899FAB653561773A99CE7EE65F64D7FEFF384A7B	BC2A870B7BCAA3BDF41605F6B10687D2BBC46BA245F42F21E2F1256930BFAB6FB7ECA15900A86CSAFD1	7860FD89D70DA5055EA803A0880!	V31D4F5B46634C
	count = 98 seed = 6F6E47E8336ADEE99B2C52CF2DC8D461E8A54C3DF2F08199A9F0816AF8455381054CE47A7766 message Length = 3267	5726D3AFC2E2F2BEAF8E8		
	message = 06709683FE7BFD74B3ACD21AF3B98B74CA73D0126CB315538937CAC4EF0AD458B765A26DCCE1C90C559 public key = 1CE4F13E8E576198C331A1E8FCCB1005198637DF0FA5CD6B42DDFD2FF7E789FE	9CE691E7EB3E0A497D357E1AB583C761439C0A66D1164518F01B6894067925753CC2866A91552FCD0EF	029C2284C620CAF364DE6C56EB4	LEE8E4431D9BE2
	Secret Key = 3/519402E8021F2257255C8U2E4994F3533C8E08U058F7/51CE9200798518FALE4F1 signed message length = 20243 signed message	135853/814863314158FCC8100214803/0FUFASC0084200FU2F4/F/844F		010101110004
	Conut = 08 CO222291/12822378949444575388780145782233495654823575376926125778917376415331	232.01709415192010010011000013701020000114109200021013442001644023141900430116104	D410642117E30689AA6630671A6.	.82032A311880C
	5000 = C62C02200153939C.3004A03A412AAA030AAA44028048EE085002C.159108897913088F2022F722 Ressage Length = 3300 Ressage =	213/784804CF2AE9ELC4E		
	DLL ANDED SAKES SUBBOLICA DU SKATSFEDSTSTS SKOUESTAVSKAT 993 SAKES ADV / TARSKATED 19906A SUF EALED FLUI public key = 4819584Cof4230DBBEBSTATES SKATSFEDSTSTSTSTSTSTSTSTSTSTSTSTSTSTS secret key = 690482BFF6C1DBBAC6071DD395ADF69ESSE1BFC4E0992AB650FFB5E60A02B1724B19E0 stipned message length = 20276	1407E402C0744034720735C035C5007C54977C03E52579FEC63UCC8442EACA8488E574249E908501 84C64F239DD8E88781E5272FD9CF23712E79405F79C8488A889D39FE64A	40/30AD3/C0A443483EC1C382081	:00704040010FC
	signed message = 040885E5CD96F086407E47C5A9919876978636E81C662901732C68458CEA392338C706FCBDD6ADF554D	D2B9FE89BE5A6DBD6B82957C65049EB3346805C81B679455F8BE612BDCDD9C1B5F9D2463CEC3887BBA2	56B6C5FE13B3FB8DF6C431AE0F8	4BB2855E11C7DE
		Plain Text 🔹	r Tab Width: 8 ▼ Ln 1, Col 1	▼ INS

Figure 17: SPHINCS+ Unit Test Results (Record 95 to 99)

The unit test results demonstrated that SPHINCS+ key generation, signature generation and verification are working correctly which provided the assurance that the SPHINCS+ code is fit for purpose.

Following the instructions given in the D5.1 [13] document on how to add new algorithms to TPM, the consortium has made the relevant code changes to SW-TPM for the following: AlgorithmCap.c, CryptUtil.c, Implementation.h, InternalRoutines.h, Makefile-common, Marshal.c, Marshal_fp.h, TpmTypes.h, Unmarshal.c and Unmarshal_fp.h.

In addition, two new files were created: CryptSphincsPlus.c and CryptSphincsPlus_fp.h.

CryptSphincsPlus.c has three important functions:

- 1. Key Generation
- 2. Signature Generation

3. Verification of Signature

The code was taken from sphincsplus-sign.c. The interfaces were then modified to adhere to the TPM specification:

Key Generation						
CryptSphincsPlusGenerateKey(// IN/OUT: The object structure in which the key is created. OBJECT *sphincsplusKey, // IN: if not NULL, the deterministic RNG state RAND_STATE *rand);						
Signature Generation						
CryptSphincsPlusSign(TPMT_SIGNATURE *sigO OBJECT *key, TPM2B_DIGEST *hIn);	Put, // IN: key to use // IN: the digest to sign					
Verification of Signature						
CryptSphincsPlusValidateSignature TPMT_SIGNATURE *sig, // IN: signature OBJECT *key, // IN: public modulus TPM2B_DIGEST *digest // IN: The digest being validated);						

The consortium has made similar code changes to TSS. In the Utils folder: certify.c, create.c, createloaded.c, createprimary.c, loadexternal.c, objecttemplates.c, quote.c, sign.c, tssmarshal.c, tssprint.c, Unmarshal.c and verifysignature.c. In the Utils/ibmtss folder: Implementation.h, TPM_Types.h, tssmarshal.h, tssprint.h and Unmarshal_fp.h.

Appendix B: Rainbow Implementation

To implement Rainbow, the consortium has used the Rainbow codebase from the NIST site: <u>https://csrc.nist.gov/Projects/post-quantum-cryptography/round-2-submissions</u>. The Rainbow source files were then renamed with a prefix of "rainbow" in order to make them identifiably belonging to Rainbow when ported into QR-Libtpms. The "#Include" statements within the source files were also appropriately renamed.

The rainbow-Makefile was modified (due to name changes) and executed. It produced four executables: rainbow-PQCgenKAT_sign, rainbow-genkey, rainbow-gensignature and rainbow-verify. Before integrating into QR-Libtpms, the Rainbow code was unit tested by executing rainbow-PQCgenKAT_sign. The test produced a response file, rainbow-PQCsignKAT_92960.rsp, which contains 100 test result records, each with a different message length. The following two figures show the Rainbow unit test results for record 0 to 3 and for record 96 to 99 respectively.

Activit	Activities 📲 Text Editor 🗝	Fri 11204		¥ 41. @ •
-	Open+ B	rainbow-PQCsignKAT_92960.rsp		=
-	# RAINBOW(16,32,32,32) - classic			
.=	count = 0 seed = 0015502300158C5FC95595FE04FE7425767E2E24C28C479009D860C948CE0F705648C206F9FF97E2	MR5410802F1FE01		
-	<pre>nessage length = 33 nessage = DB1(dD0D7)46(BEDEADE3D3E8A030EAA3A2(DD07E83EADE5E22E7EBE57BB556AC8))</pre>			
A	public key = 2098EFFA86ACA91173C8848C92DA239F04420EC64853F6938E1515C391D3776D9C399330691	1771638868F89F46D42173D933D595EC1369E34C5C23853B488E589876ED432F6D89A7BA8C58DA8CF3F8487680E7EF	0087B2358A84/	A78893FAA8081
-	7C9935A0807694AA0C6D10E4DB681ADD2FD81A25CCB1480320CD739936737F2DEB584CE650D512373651DEBE	B44498CC743BADB4C1FB79896E2C6AC7592CCD95911A7E419896D814A236D96458E7202EDABA35DB3D7A312E38866	88FBA8208881	E875AD5758843
?	signed message * signed message * signed message * signed message *	04844 EBA 1976 2467 886 464 EARE 78180 C77 71 27 EBEB 2748 EAGE COD 16 24 78 78 188 77 0 EAANOEB 201415 C23 18170 0 C	S610EEACDCC2	
-			JUIDENACOCCE	
0	<pre>count = 1 seed = 643358F29E5DE62842C941766BA129B0643B5E7121CA26CFC190EC7DC3543B30557FDD5C03CF123A4 seed = 643358F29E5DE62842C941766BA129B0643B5E7121CA26CFC190EC7DC3543B30557FDD5C03CF123A4 seed = 64358F29E5DE62842C941766BA129B0643B5E7121CA26CFC190EC7DC3543B30557FDD5C03CF123A4 seed = 64558F29E5DE62842E94 seed = 64558F29E5DE62842E94 seed = 64558F29E5DE628F2 seed = 64558F29E5DE628F2 seed = 64558F29E5DE628F2 seed = 64558F2 seed = 64558F seed = 64558F seed = 64558F seed = 64558F seed = 6455 seed = 64558F seed = 6455F seed =</pre>	156048EFEA43CB68		
	Pessage tength = 00 pessage = 225D522CEAC61930A07503FB59F7C2F936A3E0754B1DA3CA299A80F8C5DF9223A073E7890E02E	BF98CA2227EBA38C1A82568209E460BA961869C6F83983B170CD49	100010122160	
	public key = P0020C303341203/300842813C8A084A623300A6818184A6060901403A33008C430/3200F428 secret key =	103/124/040FEF00747712/04/040403340404041/3/2014040404032577344052773404040111120/AFCD44130212/44/14	400039423100	3547807850354
	signed message length = 130	P3DR91FE063/0/EC4E000/139036632400000E00632C0800EBEEF06CUE///423020E0R36C0C220EF3CCCB042CEF6	190493303181	2047097070203
	<pre>stgned nessage = 22505CE2CEAC61930A07503FB59F7C2F936A3E075481DA3CA299A80F8C5DF9223A073E7B90E02EBF98CA2223</pre>	7EBA38C1A82568209E46D8A961869C6F83983817DCD49586D52F22F867D6569CB6DA1EDC888888D29258FA78D889893	140E722D3D88	DC5A3C70B48A1
- 1	count = 2			
<u>e</u> 16	<pre>seed = BrrsarpAsuB4C2D8Bub2E4047888804K2FA12S00AbSCA4C9F918B505707Fk77S95101889149C970443 nessage length = 99</pre>	1241080700084358		
	Pessage = 288C488F29363EAEE469A7E33524538AA8666AE98988EAA1901F18593283DA214389E9E1973F7FF8E6C6AAA3C	:88908E580883412EFE96DEECE3846D8C468C7789228789775ABDF56AED6416C98833788CB7A4984815DA1B14668DC	F34AA34BF82C	EBBCF
	public key = 6EDE64057CF0C84082EF7E065ED0E482CE617DF0308E171A3D3DC8E736C561C4223D9C83A3E7 secret key =	8F558A85973CEF80820178AUC811F4084007AF4CU04A808U08UC81DF88BD0727E0053225241EFUC0458E95C3124F2	9465430C707E	519C4C2AD2908
	10830E8892402598CD1CCD28309583C5847CF88919EC2872C288247CB15A5509A04080E4487D08DBEDF5A388 signed message length = 163	AAB92D8519DA5DFC4DEB78370574FED40514889758490EA1AEE87A875F1CC1DF89088CDe0D484A8000450C2969D00A	A985883E88DF	/BB548C52D505
	signed message = 288C488F29363EAEE469A7E33524538AA866AE98986EAA19D1F10593203DA214389E9E1973F7FF8E6C6AAA3C	08900E50D003412EFE96DEECE3046D8C468C7709228789775A8DF56AED6416C90033780CB7A4984815DA1B14660DC	F34AA34BF82CI	EBBCFDD668941
	count = 3			
	<pre>seed = 58C094D2178C13EDFDBEAS7EDBF3A536F8F69FED1D54648CE3D0CCB4847A5C9917C2E28C4D5F620E5 nessage length = 132</pre>	937F0D329FCF8A16		
	<pre>PRESSage = 2F7AF5B52A046471EFCD720C9384919BE05A61CDE8E8B01251C5AB885E820FD36ED9FF6FDF45783EC81A8672</pre>	28C8B74B426ADFF96123C08FAC2BC6C5BA9C0DD71761292262C65F20DF47751F0831770A68B7B37608B7F5EFFF86E1	1AC35F3S3A6F	2440088082878
	public key = D307E4F4B31BC07FCFE8B790FD08042BF534B2537B1C76D1CC6311F9681A6463709F96F1FA8 secret key =	885468693118080A663EA74CB72671DC39149222792C7EC9DCFD1E485A6943E037F1D98195002016FAA3A833A2FD5A	224F9EDA6E33	932C78B9088A1
	539577C87F2088FBEDFF1853F235D607321857D8328BA645F8DF3A89DD426552B23A4665ABAC674C2E60D709 signed message length = 196	26E089DE7F62F4C81C188EB12C6DB3F2CF866D079E57AAD7E62477338A6A28CE43CEC28C074786A0309CE8AFCCD1F8	380ZAEAC647A	21900C1A788A7
	slgned message = 2F7AF5B52A046471EFCD720C93B4919BE05A61CDE8E8B01251C5AB885E820FD36ED9FF6FDF45783EC81A8672	8C8B748426ADFF96123C88FAC28C6C58A9C0DD71761292262C65F280F47751F0831770A68B783760887F5EFFF86E1	1AC35F353A6F	2440088082878
	count = 4			
	<pre>seed = F1902A7815F378C7F5802D8C8CE5848D82E8856917180628F884D8C06AA41D6E903980A107245DAFA message length = 165</pre>	44EC109A57332914		
	Pessage ≈ 1CDF0AE1124780A8FF00318F779A3886B3504D059CA7AB3FE4D6EAE9FD46428D1DABB704C0735A8FE8708F48	197410178723D9A304E54FDC5789A7B0748C246487308AC9665115644C569AE253D5205751342574C03346DDDC1950	A62735466168	96D0CSECE0A04
:::	<pre>public key = 73DCFB3E68A8BC2231DC19F1B1A3CD767DB799C928BC4FF33SED2134E28F26F58E8E8E8E8BB2 secret key =</pre>	PR008ASEECA618E9923F47D0AD1254244C87673C398DA3E65D023A72E4384AFC399598F570A8606ACCF264BE7D9695	87C3AF6CDDB9	EF9CE87AB5BC4
		Plain Text 👻 Tab Width: 8 👻	Ln 1, Col 1	 INS

Figure 18: Rainbow Unit Test Results (Record 0 to 3)

Activities 📑 Text Editor +	Pri 11:05	ý 41 B +
Open* B	rainbow-PQCsignKAT_92960.rsp	Save 🗉 🔿 🙆
public key = 370419E8006EESD33AC423EAC40C91167850F64DC65F3	394859C70850CC999E48296E6CAFD8AA2ADEADE5DAEBA5AC9FBEA78FF27A7912DCF1F4678766190D8418506C1E2171CC24F91A6F	C18DA9FE7E61E4B66154A823E7B74533EEB138F912D19
09931E321732B082EC9CA1DF12BA485498FC7D3E76A404871892F41987 signed message length = 3232 cloned message =	777FFBB73C2CAF93D6C9F618C1C9AA079C9E07C5A3249E97AEB71DDDF441476BD6AB6E2C815F8CD8E140EBAB0D998267723C61A4	F6ED794818E4E835F01A3A442E64D87F1F85C6FAC1723
1F7AB96E8C1401A5094672D7034FA8F81703A2CC18983C972CC66736C0	D988031AC8A479CED21A1F6349380F85F3E83161646088189AC3EA22F8098888E2E8A4E9975714E5A98985817F426C41F3968349	686869AF917564A264840188FA127FC3200DC16A9E663
seed = 26CF866726D4DFA38AE8739983888336F18EE59E9F23AE4C81E message length = 3281 message =	E73049964997EF21C85F5412F9A78A1EC39FC6228C36CA	
DE897F02AE7292ABAFA6A0CAD52929113410F28A97284184E894C40316 public key = 21AE401E02FF69800D51651B224E2382E48281D9F0089 secret key =	081420751560956F495E28772635625AFC3CA6698F8F0E4D0A05EF2430F1908A1CE780E8572590E01E66283E1963F28072280CE8 94373106C8C7A96A057E9A38062E9788F9E81F73A6E6BF106F04B7ADE6ED808873CAEC58F72CDE494750D6110003305429A6C639	365552F658D405F1A284DD8ED07BA61C4453D30CC28C8 58DA1DD7D000091374728EE65CA75FF0A8B0C692474A5
F838451E465929888AE968484081DC0EDF876A93548F27F981966C8886 signed message length = 3265 signed message =	08A3A110F69B2B5A210EDF89FC0401D013007CF82D424185BF412DA1C0A1C56B92CF6A869E4554882F61D1967FBA71BC5980886A	F3334A40825E1FDE73FDECCE3D7CB4121D6E5336B8888
DE897F02AE7292ABAFA6A8CAD52929113410F2BA972B4184E894C4D316	081420751560956F49CE28772635625AFC3CA6698F8FDE4D0A05EF243DF1908A1CE780E8572590E01E6E283E1963F28072286CE8	365552F658D405F1A284DD8ED078A61C4453D30CC28C8
count = 97 seed = 13F1F44609AA5AC8532788F74C9E6447A6CE4294C037867F430 message length = 3234	DF5543708E261005C7260EE8F46D6694D0085088343F8E5	
<pre>sbgg = sbgg = public key = DA95286E71299FF23C19743AC2437874EFB6E0428A92 public key = DA95286E7129FF23C1974 public key = DA95286E7129FF23C1974 public key = DA95286FF27E8 public key = DA95286F7129FF23C1974 public key = DA95286F7129F771 public key = DA95286F771 public key = DA95286F771 public key =</pre>	92181C86088FC544A39C364EF6D9281481E946C994F96829D6639727A33455660D8641E9A518F913F7FE5592C2A48C8278F5AFD8D 25132F90260CAC88A88595871E32FF368C62A8C4F4367F4C895438876A1C4F6768E1FC2ECFD8AFC3CE89908A2177863B42649102	450485387C20945654F08168247A98F56A43A5020955F 374DC7428DD8BAD145D5168C34275880AD02AB3D0DF0C
8E43348258900CAECF0FD9BA584EA26A4123D4543A8A0FE126D4A7E07F signed message length = 3298 signed message =	F6867AFB7A11E83942EADE67E0B91E434A7A8BF8D9B9588F720FEC7A6A7CAD6DCD9D384A3B80E9C1820ABB1A8AE5883754181DFE	50F11CCB1FAD0EECAAEE1858F3D385D177FCA14426479
525E8B98C55864849FFC71EBC953F7A0ECA6298F6AA15A83BF69238D59	92181C86008BFC544A39C364EF6D9281481E946C994F96829D6639727A5345566D8641E9A510F913F7FE5592C2A40C8278F5AFD8D	458485387C20945654F88168247A98F56A43A5828955F
<pre>count = 98 seed = 6F6E47E8336ADEE99B2C52CF2DC8D461E0A54C3DF2F08199A9F message Length = 3267 processage 1 = 3267</pre>	F0816AF8455381054CE47A776672603AFC2E2F28EAF8E8	
00709083FE7BFD7483ACD21AF3898B74CA73DD126C8315538937CAC4EF public key = CE6A2DE3F33CA88C09A8626EE8A48CA9E7673B88800689 secret key =	F0AD4508705A260CCE1C96C559CE691E7EB3E0A4970357E1AB503C761439C8A66D1164518F01B6894067925753CC2866A91552FC 960826C90980807F480A66C089135A85EF018E53A7DE75657AE8E603B3ECC2C640E5A5A668F48B60C8FEBB300664786597A4F2	D0EF029C2284C620CAF364DE6C56EB41EE0E4431D9BE2 F52CE053AC89FB70D75032F48DF1AF5590C75959FD08C
37519A02E001F2257259C0D2E499AF3533C8ED0DD5BF7751CCE920D79 signed message length = 3331 signed message =	98518FA540355C3588A7742E44D6E301A8E8F8883FA65295E66898021DE1156D8858E0A95E186AFC8F80944088BC7438AD282756	08725F2D4879FE6C3846D59ACFEC17A093C8A02F79874
00769683FE78FD7483ACD21AF3898874CA73DD126C8315538937CAC4EF	F8AD4588765A26DCCE1C90C559CE691E7E83E0A497D357E1AB583C761439C8A66D1164518F01B6894067925753CC2866A91552FC	D0EF029C2284C620CAF364DE6C56EB41EE0E4431D98E2
<pre>count = 99 seed = CB2E6226615393FC3BD4AB3A412AAA030AAD40E8648EE6856D2 message length = 3300</pre>	2C1591D8897915D88FZ022F722137784884CF2AE9ECC4E	
<pre>Message = D21A0883A2356805E678673C45F8055FC526663F692AF9935AEA307F14 public key = 602FD60E188DF3EB26C8D35DA2319AE351850F4C758DC secret key =</pre>	4A5C418979906A5DFE42EBFED1487E4822874A85AF28995E085EC8007ECA4977C63EE5299FEC63DCC8C42EEACAB488E574249E90 CC17ED094C68366F5985BC5FCE5AA1230178C0DF807F2604A8365866A219835E6C6F0882EFFA68870E4FB18791258C4CE6252698	856146758AD97CBA443485EC1C5820BEB0964640010Fe 684EB0C703342F4C8313FFE349F464077A7535FC379E8
<pre>d904228FF6C1D08AsC071D0395ADF69E5SE1BFC4E0992A8650FF85E60A signed message length = 3364 signed message =</pre>	A02817248994E6097E87F878BA4A4AA7C4AA98A04332E2CAABA02E51A2F5E99D979C705B3FE76195BEC1FA21259C878F38BA47AD	D037E8DFF9EE38340459C8FEF937CCF622A36DF6ACD54
D21A68B3A2356805E678673C45F8055FC5266E3F692AF9935AEA307F14	4ASC41B979966ASDFE42EBFED14B7E4822874ABSAF2899SE085EC8007ECA4977C63EE5299FEC63DCC8C42EEACAB488E574249E9D	856146750AD97CBA443485EC1C5820BEB0964640010F
	Plain T	lext * Tab Width: 8 * Ln 1, Col 1 * INS

Figure 19: Rainbow Unit Test Results (Record 96 to 99)

An additional test was done by performing the following steps:

- 1. Create a message file for signing.
- 2. Create a signature file.
- 3. Execute rainbow-genkey with parameters [pk_file_name] [sk_file_name].
- 4. Execute rainbow-gensignature with parameters [sk_file_name] [file_to_be_signed].
- 5. Copy the signature generated into the signature_file.
- 6. Execute rainbow-verify with parameters [pk-file-name] [signature_file_name] [message_file_name].

See the below figure for steps 3, 4 and 6.

Activit	ies 🖸 Terminal + Tue 13:16	÷ 4. A
6	uhal005@adminhp-HP-EliteBook-840-G1: -/Documents/Rainbow File Edit View Search Terminal Help	• •
• • • • • • • • • •	File Edit View Search Terminal Heip Oublo05gadrinhp=HP-filteBook-840-61:-focuments/Rathbows 1s Ra_Classic rathbow-genkey.or rathbow-bis_come.or rathbow-genkey.or rathbow-genkey.or rathbow-genkey.or rathbow-genkey.or <td>rainbow-verify .o rainbow-verify.c rainbow-verify.o ,o</td>	rainbow-verify .o rainbow-verify.c rainbow-verify.o ,o
	unatoosgautunp+nP+EttteBook-a4a-u1:~/votuments/Rathooms	

Figure 20: Rainbow Unit Test - Key Generation, Signature Generation and Verification

The unit test results demonstrated that Rainbow key generation, signature generation and verification are working correctly which provided the assurance that the Rainbow code is fit for purpose.

Using the SW-TPM code that has been modified to include SPHINCS+, the consortium has made further changes to include Rainbow: AlgorithmCap.c, CryptUtil.c, Implementation.h, InternalRoutines.h, Makefile-common, Marshal.c, Marshal_fp.h, TpmTypes.h, Unmarshal.c and Unmarshal_fp.h.

In addition, two new files were created: CryptRainbow.c and CryptRainbow_fp.h.

CryptRainbow.c has three important functions:

- Key Generation
- Signature Generation
- Verification of Signature

The code was taken from rainbow-sign.c. The interfaces were then modified to adhere to the TPM specification:

Key Generation						
CryptRainbowGenerateKey(// IN/OUT: The object structure in which the key is created. OBJECT *rainbowKey, // IN: if not NULL, the deterministic RNG state RAND_STATE *rand);						
Signature Generation						
CryptRainbowSign(TPMT_SIGNATURE OBJECT TPM2B_DIGEST);	*sigOut, *key, *hIn	// IN: key to use // IN: the digest to sign				
Verification of Signature						
CryptRainbowValidateSignature(TPMT_SIGNATURE OBJECT TPM2B_DIGEST);	*sig, *key, *digest	// IN: signature // IN: public modulus // IN: The digest being validated				

Using the TSS code that has been modified to include SPHINCS+, the consortium has made further changes to include Rainbow. In the Utils folder: certify.c, create.c, createloaded.c, createprimary.c, loadexternal.c, objecttemplates.c, quote.c, sign.c, tssmarshal.c, tssprint.c, Unmarshal.c and verifysignature.c. In the Utils/ibmtss folder: Implementation.h, TPM_Types.h, tssmarshal.h, tssprint.h and Unmarshal_fp.h.

Appendix C: BIKE Implementation

To implement BIKE, the consortium has used the standard codebase from the BIKE site: <u>https://bikesuite.org/</u>. The BIKE source files were then renamed with a prefix of 'BIKE-' in order to make them identifiably belonging to BIKE when ported into QR-Libtpms. The "#Include" statements within the source files were also appropriately renamed.

The BIKE-Makefile was modified (due to name changes) and executed (see Figure 10). It produced four executables: bike-nist-kat. Before integrating into QR-Libtpms, the BIKE code was unit tested by executing bike-nist-kat.



Figure 21: BIKE Unit Test

The test produced a response file, PQCkemKAT_BIKE_8838.rsp, which contains 100 test result records. The following two figures show the BIKE unit test results for record 0 to 4 and for record 96 to 99 respectively.

Activi	ities 🔋 Text Editor 🕶	Fri 14:37	± 41 G ≁
	Open → 🖻	PQCkemKAT_BIKE_8838.rsp -/Deciments/8KL/Doctorbod_Implementation (Foture 17%)	Save 🗏 🔘 🗑 🙆
-	# BIKE		
	<pre>count = 0 seed = 061550234D158C5EC95595FE04EF7A25767F2E24CC2BC479D09D86Df</pre>	C9A8CFDE7056A8C266F9EF97ED08541DBD2E1FFA1	
	pk = 8702EF0394A6AEC44AD80995DCC9D755A2177E1FC682A15D1EA5FA4A46D13D8	B7EE74EED917FFF183443371E06E51C58426CD826B955E4E6E6002BAAD5866B71E2922C082C8C6E2189415BEE0BC62229974B78D9A	D7B71C076041C86B27C129C4FB42D914DC4D63
A	sk = seconoccoccoccoccoccoccoccoccoccoccoccocco	500500040005000600000000000000000000000	810000000000400000000000000000000000000
?	CC = A826449EB131567EB7A85288C2B53985F4286B6D6898D04D0750E6C63DA8B5B SS = C1C96E2B8B1D23E52F02AD3A766A75ADBEDF78A1558B9441284AB534EE	86263316590C441794AFD8CA54F92455D28F2D65B5B785746AE4887295E869E3DE441A078980EF7D2D51F171E6609E5EA8F94A8903 D0BDE36	764243CD135D7E387701F95D4BB710B90E6558
8	count = 1 seed = D81C4D8D734FCBFBEADE3D3F8A839FAA2A2C9957E835AD55822E758F ok =	F\$7BB556ACB1ADDE6AEEB4A5A875C3BFCADFA958F	
_	6DC528C86568426C85D5036537F885889F7C0150ECA11FC3880729C15517262 sk =	2371648ED547FB88718AD951372876FEFFF249E4E762AABE7366E8CE48DD6FE86914FBED499E9A5C58E0EBA540E6690AE115AE96A7	B10F8F4E54E4D690C55727327251E87D1F3D08
P-	00010000000000000000000000000000000000	000000000000000000000000000000000000000	000000000000000000000000000000000000000
2	99FC845126815C4AAE174998C62F434830FD00C7FC32432AAB805F94C846688 ss = 4F96266984A162C992759475D484196C6CC6F38FA7A743293597B74865	B7CBB80CD2676603DB3BF78EE9A778249714755A5E688B12807AE28270184A4E9FEFCE03460F61A1C6D46981AA2F909CBF74613546 50F3850	7CC96771657615358201ADB124383AC8073655
	<pre>count = 2 seed = 64335BF29E5DE62842C941766BA129B0643B5E7121CA26CFC190EC77</pre>	DC3543830557FDD5C03CF123A456D48EFEA43C868	
	pk = 39C1F569B2E3981286D42FEA5527B89F8EF0CDFBA5B4B3690F4623A603472B2	2355158C8798579F09F30208AE6D7454B4B83F1506EAC88F696900287E2AF5465E898ED11921E6865CA8B38376FF6628C9DC427305	DB647B7379AD5C78A3F355204EC72D33F21AFE
	sk = 000000000000000000000000000000000000	000000000000000000000000000000000000000	000000800000000000000000000000000000000
	CC = 19C9A512B4D4229CA0A48B963A356F2B39B9D0079BCDEFCA0733B31B67E1440 ss = 9E96D6CADEDD2E2724AB9C3AB45F436481F7C50F6A000CFCDE9090FA3E	81218F29A16E67C9CF398EDE45SA9860EE0ECA895F90931A8C87437D82E38E986F4B31F0C6375E130C68E143021626703FD3101C28 ESAA4F8	225D917C9134B5BF03ADF4E611D45B83195EDE
	count = 3 seed = 225D5CE2CEAC61930A07503FB59F7C2F936A3E075481DA3CA299A80F	F8C5DF9223A073E7B90E02EBF98CA2227EBA38C1A	
	pk = 3014028253A7514E6100FB6D477404F2400C54EBF3970EF0E1305D6C2B17A0E	E9355AAD73D88750A65762B7E62E71D648DC466017980AEE569D47E280380CC3E68D788AE234BE83AE2EF6ED3B4326A412498BBEAD	22218BECF09D678DAFCB7B98012456AA879B2E
	5K = 00000000000000000000000000000000000	000000000000000000000000000000000000000	000000000000000000000000000000000000000
	29C913ABB7F3E9B5A96B1FB2EF4EAFB8ACD43E73D1AEB231576937FF2B3DAFA 55 = 4A6CB0A3C5F8C16C46F92ABA022F3FD03A3212210C8ADCFD33300CE1F3	x79F0692598321272EEE57748AEDDF77056536D908117105A2C082505628D075838C1AD5ED0C21FD879C30D04EFD98FE1039C7E984 3838099	FE942A8427BEBA1595885A92A71884662E7C8C
	<pre>count = 4 seed = EDC76E7C1523E3862552133FEA4D2AB85C69F854A9354F0846456A2# ok =</pre>	A407E071DF4650EC0E0A5666A52CD09462D0C51F9	
	B28D0328B9C34573330EB39416D2D2B785F165483936B0AC5A7ECEF503B4EC2 sk =	29EB8F22F2A1222758E59D6B5322796229CB05557DBC8972F1D3AFA5CBDDC1FEEAF43C8E65AC7453A288601F04052C49A3FA81A351	598D6D30813653D685358B8AB47720F3E1C020
	ct =	000000000000000000000000000000000000000	000000000000000000000000000000000000000
	3908A71FD014D434E9FBF57EE3FDBEA0F12324B2C51A137884A688FCD94D27C ss = 6358096EF8C31F80564125FCC3EE6242139945853C9A48E6CA48D24884	C66E480BE6A6252A40FF9193FCC94F35F433A5E6F202C88DD497D5844B66784SA992A8F51D8A411A6B5989F21D3EFA2028A73FFEF3 4SDBC80	5D7FB29311281FC061ED78F95D21D6425C9843
:::	<pre>count = 5 seed = AA93649193C2C5985ACF8F9E6AC50C36AE16A2526D7C684F7A38B4AE ok =</pre>	BCD7B6FF790EB2BADCEB9BC73B8D66251F97AAAAA	
		Plain Text = T	ab Width: 8 Ln 1, Col 1 INS

Figure 22: BIKE Unit Test Results (Record 0 to 4)

Activit	ties 📲 Text Editor 🕶	Fri 14:37	± 41 G +
		PQCkemKAT_BIKE_8838.rsp -/Decument/J0787/Continued_Implementation/FinantTPM	Save 🗏 🔘 🕲 🤒
9	00000000000000000000000000000000000000	000000000000000000000000000000000000000	866888888888888888888888888888888888888
	A9EE72948C586D8755E34101718BE0D63A1EE7F2A2E25ADE38818D067DCDF237888 55 = AF42D7D86204145DE5F88CC335E8247A5FDF88583E3FEFFE9C12C08F8020C8	'386881596C55793C28FC13C24D6D7E8F8615AA8346D50A0685D4A8EFC3D9925F793CA31A3CF18A13A6202FEF05400758 77	FC9F6CB217BE34B7B8B5C5B0402003390DA0AACA12
	count = 95 seed = 121D90E70AF6204445D0DEB28AC0C108262719E9FD3476ACA748BFDE89FA ok =	04D8D5F89A6Z4E8A75D880431F8D10AD28F	
-	75DF9854EF16A881D45F82288309C895577F8DFA3158FD97E892C970D5C0C5A5C8F	ASF5AA130A1BF7A6877B9SDF74B9563453A18805ADDF501BF08E3D5C6DBC5B7D5D723B4680BE43563E5AB6174B719AA7	7FF8504313486206663E1787750507608C0C7A4D01
?		000000000000000000000000000000000000000	800880818888888888888888888888888888888
•	80064CB494F72604E510F8CB41F77171BA6C09444F2FB8C95CF02370B085BCF3E2A 55 = 40D43148EF9A288ECAA798BCCD8FD1569C9BF0192783CD84777A703E982596	18883384E62F5EE116A8F83448807912175E79664798488CDAC33F164195308E3B3E26F92A2EEED0EE82500887DA5D2E8 14	5783318EC1FB9B4E7F0F0B055F098E14668842556C
_	count = 96 seed = B3AC6503206ACCC2A92CBC210D020A2654726911D11CE676AA04FEAA08AF	ID20C654E4105883AE470EC3A8299075D420	
^ _	pk = 0F8F458877960F5D48D38B4DC2994F8D58FE255CC594E925296D3565071CA86CE212	378A5DF8F5282F5A988288F15845D85FC1C6821D94356A5861224CFF94F8C8AC641F8D4FDA58D4434618F12488A56FF1	CF19648748E8482E8C1EA1063D7EE42C16852B6237
-	sk =	000000000000000000000000000000000000000	000000000000000000000000000000000000000
-	ct = sogreenpeserzhiedenpisonenzenendaagzeiendenzenzennaarnenzenaar	LE 9 7 4 5 5 7 6 5 9 9 9 7 7 7 4 3 8 9 5 9 7 4 4 1 8 7 7 7 8 9 8 4 8 4 4 5 5 6 7 1 1 4 9 8 8 7 5 9 6 8 7 7 1 9 5 8 7 7 7 1 7 5 8 8 5 7 1 8 5 7 1 7 5 8 8 5 7 1 8 5 7 1 7 5 8 8 5 7 1	DD00EEEEE17001EE7E3E3A3DED4106A1E00C30E50E
	ss = F2E4C8DE2610698BA6157F6774F25481AA39776774F6859A573D67831FF510	9	00011111110111111111111111111111111111
	count = 97		
	<pre>seed = S9EFF60B1EF618SDB34EE1E3B1D02F1S9106CECEAA79BEB74923B4F5623D pk =</pre>	8C5208F5D2594A1F7C6C64D12CF144E9ED4	
	DA6707C5417A86203BD9B8C44F1C78E9B44434BD2FE5E98BFA401EAE95FE50879B2 sk =	AEC93D749746F6589C850E81981D70485F8E05C4D2677C5FF8D2D5886EFC58817D8095029099A2835383E7951DC9EA18	AC5D89D61A85CFED3A671328A60877D82B7DF9584E
	00000000000000000000000000000000000000	000000000000000000000000000000000000000	000000000000000000000000000000000000000
	FAB1083825A1CC15C40E6291D45E29F9EE13E44A4CE72F8620BF5F8ACEC5A72D200 ss = 0BFFC566BD167CE761B83B253E40734C3A95A4F85EA903EE727BCC49F48A5C	EB5FEBCC18D5BCB48781433A3D0548F61E0DE7AB51121B1C3D606F231AF936F626F2447D267F7B37F7EE404F6AF02282 9	C76D06EAD8CC798213DD3467D89A383B545A6115C8
	count = 98 seed = DDDC40DC318E4737D3E474E7568B37E4CB2E53C8B3E768EEA4DE7669EE048	1104FR017868CFF050F00244404044FF6812	
	pk =	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	4300000010001407300704000010427080070001
	SBEEC0504C/E/4/005A2/14E1F3E82E00802953/E0F9/1132020E128180E/F988940	arr1xb1E/0EDC20/F0EF/30xC0C0F3/03B30000F212D3B/1/03DEC39/B2E/22X09FC43000DFF04ADL/CC00B03/D31913	439090923FCF14C72007C0400C8138073B3273B737
	ct =	000000000000000000000000000000000000000	990999999999199999999999999999999999999
	A7C6A89882088812EDF88850A3952DD860248401C01A58788810073C3C02D778211 ss = 423C18DD6AC74D300A163F8EE86648FF4E654DA90C11123168FC28ABE1020E	38A55E790184F05494E89FFEECB603AFDAA515FFA0FA8B4E89D99664894E9D00948651C209C2D7175DC2E5844193F406 j9	8EDB4E6597C7AB4A479A37EC498183B65610426298
	count = 99 seed = 2A6F73868815366F572AEB6C79E272CC2187095FE09575F18072C9D677DA	38C9C8A48C393B7524684D299BEDD268C88	
	pk = 498DEDEAF7760EAEE44A31A5021D94410DB2826EDEEDB8773F023AB255EF31871A5/	135C9203A75B97603C17F3166ACDEBC98A174FE789B3EC58F809DCE31158019BFB3FA02BA330DA088CBB347407A86617	66811B92EC46E5687D48F321BC9883A57A52AE79BE
	sk =	000000000000000000000000000000000000000	000000000000000000000000000000000000000
	ct = EAS1825862889980083838782EA9EDER574581487864585304E642E53269840DE1E1		2468427E948095488C1E8CE49C8E49E87788EE59E
	ss = 9E4C29ECFAA8316E7C46EA8DD74F67634CDE36E5F75A17797F3264F2A82D2F0	anniaraeonnovieriareoaoroi eenriaeennainoinoeroeroeroeroeroeroeroeroeroeroeroeroero	2400421520003340051F05EA950F49E07780FF39F
	-	PlainText	Tab Width: 8 ▼ Ln 1, Col 1 ▼ INS

Figure 23: BIKE Unit Test Results (Record 95 to 99)

The unit test results demonstrated that BIKE key generation, encapsulation and decapsulation are working correctly which provided the assurance that the BIKE code is fit for purpose.

Using the SW-TPM code that has been modified to include SPHINCS+ and Rainbow, the consortium has made further changes to include BIKE: AlgorithmCap.c, AsymmetricCommands.c, CommandAttributeData.h, CommandDispatchData,h, Commands.h, CryptUtil.c, Implementation.h, InternalRoutines.h, Makefile-common.txt, Marshal.c, Marshal_fp.h, Object.c, TpmTypes.h, Unmarshal.c and Unmarshal_fp.h.

In addition, six new files were created: CryptBIKE.c, CryptBIKE_fp.h, BIKE_Dec_fp.h, BIKE_Dec_fp.h, BIKE_Enc_fp.h and BIKE_Encrypt_fp.h

CryptBIKE.c has five important functions:

- Key Generation
- Encapsulation
- Decapsulation
- Encryption
- Decryption

The code was taken from BIKE-kem.c. The interfaces were then modified to adhere to the TPM specification:

Key Generation	Key Generation						
CryptBIKEGenerateKey(
// IN/OUT: The object structure in which the key is created.							
OBJECT *BIKEKey,							
// IN: if not l	// IN: if not NULL, the deterministic RNG state						
RAND_STA	ATE *rand						
);							
Encapsulation							
CryptBIKEEncapsulate	e(
// IN: The o	bject structure which co	ontains the public key used in					
// the encap	osulation.						
TPMT_PUE	BLIC	*publicArea,					
// OUT: the	shared key						
TPM2B_BI	KE_SHARED_KEY	*SS,					
// OUT: the	cipher text						
TPM2B BI	KE CIPHER TEXT	*ct					
):							
/,							
Decapsulation							
CryptBIKEDecapsulate	e(
// IN: The o	bject structure which co	ontains the secret key used in					
// the decar	sulation.	·					
TPMT SEN	NSITIVE	*sensitive.					
// IN: BIKE	security mode	,					
TPMT BIK	E SECURITY	k.					
// IN: the cit	oher text.						
TPM2B BI	KE CIPHER TEXT	*ct					
// OUT: the	shared key.						
TPM2B BI	KE SHARED KEY	*\$\$					
);							
Encryption							
CryptBIKEEncrypt(
// OUT: The	e encrypted data						
TPM2B BI	KE ENCRYPT	*cOut,					
// IN: The o	bject structure in which	the key is created.					
OBJECT	-	*BIKEKey,					
// IN: the da	ata to encrypt	•·					
TPM2B	V 1	*dln					
);							
Decryption							
CryptBIKEDecrypt(
// OUT: The	e decrypted data						
TPM2B		*cOut,					
// IN: The o	bject structure in which	the key is created.					
OBJECT	-	*BIKEKey,					
// IN: the da	ata to decrypt	-					
TPM2B BI	KE ENCRYPT	*dln					
);	-						

Using the TSS code that has been modified to include SPHINCS+ and Rainbow, the consortium has made further changes to include BIKE. In the Utils folder: CommandAttributeData.c, Commands.c, Commands_fp.h, create.c, createloaded.c, createprimary.c, loadexternal.c, objecttemplates.c, objecttemplates.h, tssauth20.c, tssmarshal.c, and Unmarshal.c. In the Utils/ibmtss folder: Implementation.h, Parameters.h, TPM_Types.h, tssmarshal.h and Unmarshal_fp.h.

Appendix D: SPHINCS+, Rainbow and BIKE Integration Into QR-Libtpms

1. The SPHINCS+, Rainbow and BIKE source files were copied to ./Docker_libtpmsQR/libtpms_inescID/src/tmp2/crypto/openssI folder:



Figure 25: Rainbow Code Copied Into QR-Libtpms

arainbow-parallel_matrix_op.h
arainbow-rng.c
arainbow-rng.h
arainbow-tils.c
arainbow-tils.hash.c
arainbow-tils_hash.h
arainbow-tils_prng.c
arainbow-tils_prng.h
brainbow-tils_prng.h
brainbow-tils_pr



Figure 26: BIKE Code Copied Into QR-Libtpms

2. CryptSphincsPlus.c, CryptRainbow.c, CryptBlKE.c and CryptUtil.c were copied to ./Docker_libtpmsQR/libtpms_inescID/src/tmp2/crypto/openssl folder:

Activit	ies 🛛 🗟 Files 🔻								
		🔂 Home	Documents	Docker_libtpmsQR	libtpms_inesID		crypto	openssl	▶
	Name			_	_				
	BnConvert	fp.h							
	BnMath fp	.h							
	BnMemory	fp.h							
-8-	BnValues.h								
A	CryptBIKE.	:							
_	CryptBIKE.	ı							
2	CryptCmac	.c							
	📓 CryptDataE	cc.c							
	CryptDes.c								
(≷)	CryptDilith	ium.c							
	CryptEccKe	yExchange.	c						
	CryptEccMa	ain.c							
<u>-</u>	CryptEccSig	gnature.c							
	CryptHash.	c							
	CryptHash	Data.c							
	CryptKyber	.с							
	CryptLDaa.	c							
	CryptPrime	e.c							
	CryptPrime	Sieve.c							
		ow.c							
	CryptRand.	C							
		at a							
	CryptSohio	csPlus c							
	CryptSym c	c51-105.C							
	CryptUtil.c								
	🖹 dilithium-n	tt.c							

Figure 27: CryptSphincsPlus.c, CryptRainbow.c, CryptBIKE.c and CryptUtil.c Copied Into QR-Libtpms

3. CryptSphincsPlus_fp.h, CryptRainbow_fp.h and CryptBIKE_fp.h were copied to ./Docker_libtpmsQR/libtpms_inescID/src/tmp2/crypto folder:



Figure 28: CryptSphincsPlus_fp.h, CryptRainbow_fp.h and CryptBIKE_fp.h Copied Into QR-Libtpms

4. The following new BIKE files were copied to ./Docker_libtpmsQR/libtpms_inescID/src/tmp2 folder: BIKE_Dec_fp.h, BIKE_Decrypt_fp.h, BIKE_Enc_fp.h and BIKE_Encrypt_fp.h.



Figure 29: New BIKE files Copied Into QR-Libtpms

- 5. The SW-TPM code (modified to include SPHINCS+, Rainbow and BIKE) were copied to ./Docker_libtpmsQR/libtpms_inescID/src/tmp2 folder: AlgorithmCap.c, AsymmetricCommands.c, CommandAttributeData.h, CommandDispatchData.h, Cryptutil.c Implementation.h, InternalRoutines.h, Marshal.c, Marshal_fp.h, Object.c, TpmTypes.h, Unmarshal.c and Unmarshal_fp.h.
- The TSS code in the utils folder (modified to include SPHINCS+, Rainbow and BIKE) were copied to ./Docker_libtpmsQR/tpm_sim/tss/utils folder: CommandAttributeData.c, Commands.c, Commands_fp.h, certify.c, create.c, createloaded.c, createprimary.c, loadexternal.c, objecttemplates.c, objecttemplates.h, quote.c, sign.c, tssauth20.c, tssmarshal.c, tssprint.c, Unmarshal.c and verifysignature.c.
- 7. The TSS code in the utils/ibmtss folder (modified to include SPHINCS+, Rainbow and BIKE) were copied to ./Docker_libtpmsQR/tpm_sim/tss/utils/ibmtss folder: Implementation.h, Parameters.h, TPM_Types.h, tssmarshal.h, tssprint.h and Unmarshal_fp.h.
- 8. The following new BIKE files were copied to ./Docker_libtpmsQR/tpm_sim/tss/utils/ibmtss folder: BIKE_Dec_fp.h, BIKE_Decrypt_fp.h, BIKE_Enc_fp.h and BIKE_Encrypt_fp.h.
- 9. Three Makefiles in the ./Docker_libtpmsQR/libtpms_inesID/src folder have been modified to include SPHINCS+, Rainbow and BIKE. They are:
 - Makefile
 - Makefile.am
 - Makefile.in

10. Build the docker file by running the command: sudo docker build . -t swtpm_qr -f ./Dockerfile

The following two figures show that SPHINCS+, Rainbow and BIKE were successfully compiled into QR-Libtpms:



Figure 30: SPHINCS+, Rainbow and BIKE Compiled Into QR-Libtpms



Figure 31: SPHINCS+, Rainbow and BIKE Compiled Into QR-Libtpms

Appendix E: TSS2 Issues Found In V-TPM

We have found an issue with TSS2 in that sometimes it is not possible to send commands to the V-TPM. In particular, some commands do work on the V-TPM as found, such as get_capability, but anything which requires some key generation to be uses won't return anything. This issue might happen when making TSS2 from source, and this is required for the QR algorithms to be implemented. A ticket regarding the issue found has been raised GitHub repo of the project. With the Docker/V-TPM integration, this is sometime problematic to use as V-TPM may crash during its install and some network/transportation issues. Another example of this issue would be that sometimes the V-TPM docker image won't verify the keys as the received length is drastically shorter than the sent length.

When running tssverifysignature this issue was found, as shown in the following:

./verifysignature -hk 80000002 -dilithium -if enc.bin -is sig.bin -v verifysignature: Hashing message file enc.bin with halg 000b verifysignature: Copying hash verifysignature: hash length 32 b4 23 58 e6 7a 84 95 10 9a 5a d4 e1 7e d0 c7 53 51 9f 99 ee 3e 06 7e 88 21 fb 12 29 47 dd 60 a9 TSS Command PreProcessor: Input parameters keyHandle TPM HANDLE 80000002 digest length 32 b4 23 58 e6 7a 84 95 10 9a 5a d4 e1 7e d0 c7 53 51 9f 99 ee 3e 06 7e 88 21 fb 12 29 47 dd 60 a9 signature sigAlg TPM ALG DILITHIUM TPMU SIGNATURE selection 002d not implemented TSS Execute20: Command 00000177 marshal TSS Execute valist: Step 1: initialization TSS Execute valist: Step 5: command encrypt TSS Sessions GetDecryptSession: Found 0 decrypt sessions at 0 TSS Execute valist: Step 6 calculate HMACs TSS Execute valist: Step 7 set command authorizations TSS Execute valist: Step 8: process the command TSS AuthExecute: Executing TPM2 VerifySignature TSS Socket Open: Opening localhost:2321-raw TSS Socket SendCommand: TPM2 VerifySignature TSS Socket SendCommand length 2792

[...]
5a d4 e1 7e d0 c7 53 51 9f 99 ee 3e 06 7e 88 21
fb 12 29 47 dd 60 a9 03
TSS_Socket_ReceiveCommand length 10
80 01 00 00 00 0a 00 00 02 db
TSS_Socket_Close: Closing localhost-raw
verifysignature: failed, rc 000002db

As said before, this issue is of non-deterministic nature, so it affects the functionalities only during some of the runs, while overall the V-TPM functionalities are fine. We hope the developers of TSS will fix this issue soon.