A Preliminary Study of PQC Implementations for Satellite Communication Networks

Shengbo Xu
Video Entertainment
Irdeto BV
Hoofddorp, The Netherlands
sxu@irdeto.com

Ana Vazquez Alonso
Video Entertainment
Irdeto BV
Hoofddorp, The Netherlands
ana.alonso@irdeto.com

Phil Eisen
Core Technology
Irdeto
Ottawa, Canada
phil.eisen@irdeto.com

Jacques Légaré
Core Technology
Irdeto
Ottawa, Canada
jacques.legare@irdeto.com

***Abstract* — Satellite communication plays an important role in both civil and military applications and thus its security is very critical. This paper studies secure boot and update mechanisms for satellite systems and implements them based on the NIST selected post-quantum crypto (PQC) algorithms on RISC-V platforms. Also, the paper presents a preliminary impact analysis of PQC algorithms on satellite communication systems.**

Keywords — satellite communication, cryptography, post-quantum crypto (PQC), RISC-V

I. INTRODUCTION

It has long been known that satellite communication has played a vital role in many specific applications since its birth in 1957 [1], such as weather forecasting, earth observation, satellite telephony and television, navigation and position, and military communications, etc.

Nowadays satellite communication has also entered the Internet era, helping Internet Service Providers (ISP) serve people living in remote rural areas and passengers in airplanes. SpaceX's Starlink has built a large satellite constellation using a low earth orbit to deliver broadband internet services globally [2].

Furthermore, satellite's larger coverage has also attracted the mobile communication industry. The Third Generation Partnership Project (3GPP) has initiated standardization efforts to integrate Non-Terrestrial Networks (NTN) to Terrestrial Networks (TN) such as 5G [3]. The latest models of smart phones from some manufacturers have already demonstrated their support of satellite connectivity. According to 3GPP, there will be more promising features to be expected with 5.5G / 6G networks.

Since satellite communication has so many applications in our daily life, it is important to safeguard the security of satellite communication networks. The Consultative Committee for Space Data Systems (CCSDS) has published a series of reports on security threats against space missions since 2006. The 2022 revision of the CCSDS green book summarizes the following types of threats [4]:

- Data modification
- Data interception
- Ground system loss
- Jamming
- Denial-of-service
- Replay
- Software threats
- Unauthorized access
- Tainted hardware components
- Supply chain threats

Recent research has confirmed the feasibility of some threats mentioned above. In 2023, J. Willbold *et al* [5] systematically analyzed the security of three real-world satellite firmware images and found several security critical vulnerabilities in them. Their research found modern in-orbit satellites lack proper implementation of protection mechanisms. They also conducted a survey with 19 professional satellite developers and most of the feedback acknowledges their findings. J. Willbold presented their research results at Black Hat USA, 2023, drawing a lot of attention from the audience.

In 2023, D. Maurice-Michel posted a blog describing how to hack an ESA experimental satellite. This attack could lead to control of the satellite [6].

In 2022, L. Wouters demonstrated an attack on Starlink's user terminals at the Black Hat security conference in Las Vegas, bypassing the security firmware authentication check with a fault injection attack [7].

The above mentioned publications are very likely just the tip of the iceberg as the large portion of satellite systems are still closed, not open to security researchers as stated in [5].

A satellite is typically composed of multiple modules or subsystems, such as attitude determination and control system, communication module, command and data handling system, payload data handling system, and power system. As such, conventional thinking has been that a satellite system is too complex to be hacked. However, a complex system very likely contains software and/or hardware bugs. According to information security research, there are 15 ~ 50 errors per 1000 lines of delivered code [8]. Although only a fraction of errors have security implications, it is highly desirable to have a secure update mechanism in satellite systems, allowing both firmware and software update over-the-air (OTA). Secure OTA not only helps patch security bugs but can upgrade functionality in launched satellites. All of these would save in-orbit satellites from potential attacks and extend their effective lifetime.

This paper focuses on an implementation of a secure update mechanism for satellite systems based on PQC

algorithms. The rest of this paper is outline as follows: Section 2 will give a brief introduction of secure boot design. Section 3 will introduce PQC algorithms. Section 4 will describe a prototype implementation of secure boot based on PQC algorithms on RISC-V platforms. Section 5 will analyze the impact of PQC algorithms in satellite communication protocols. Section 6 will conclude the paper.

II. Secure Boot

Secure boot is a security feature specified by various standardization organizations, such as the Trusted Computing Group (TCG) [9] and the Unified Extensible Firmware Interface (UEFI) [10].

Figure 1 below shows a functional block diagram of a generic computer system, with the focus on introducing secure boot process. For clarity, this diagram might not represent the current satellite systems.

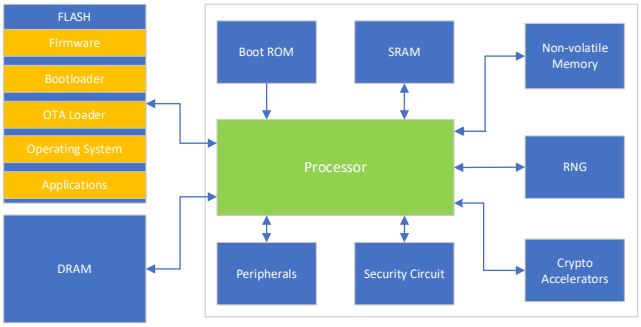

Figure 1 – A Generic Computer System

It is worth noting that most of these functional blocks shown in Figure 1 can be integrated in one system-on-chip (SOC), with the exception of the large size of DRAM and/or FLASH memory chips.

A. Secure Boot Process

After Power-On Reset (POR), the processor on the SOC first executes code from Boot ROM, which initializes on-chip hardware modules and peripherals and verifies the firmware code stored in the external FLASH. If the verification is successful, the firmware code is loaded into on-chip SRAM for execution.

The firmware code normally implements low-level drivers for the on-chip hardware modules and implements a crypto library. It also calibrates the DDR bus interface in case there is a DRAM chip connected to the SOC.

After the firmware is finished, the Boot ROM code loads the bootloader code into SRAM or DRAM (depending on the memory requirement) and verifies its signature. If the verification is successful, the bootloader will be executed to further load the operating system and applications into SRAM or DRAM. The bootloader verifies the signature of the operating system and the operating system verifies the signatures of applications.

The above describes a normal secure boot process, in which all signature verification is successful. If any code module's signature fails in verification, the processor will enter an exception handling state.

The secure boot process starts from a trusted Boot ROM code (which is masked in chip hardware by manufacturers and thus cannot be changed) and checks the code stored in external FLASH step by step to ensure all the executed code is from the designated developers and has not been modified. This is the so-called trust chain and the Boot ROM is the root of trust.

B. Secure Update Process

In case a partial or a full update of FLASH code is needed, a notification message or control command will be sent to the receiving device (in this case, the satellite) to get system ready for a firmware and/or software update (for simplicity we call this a code update hereafter).

For code update, the OTA loader needs to be loaded from FLASH into working memory. Before execution, the Boot ROM verifies the OTA loader signature in the same way as described in the secure boot process. Only if the verification is successful does the processor start the OTA loader, downloading the new code via OTA download protocol (depending on the employed network technology).

To protect its authenticity and secrecy, the new code is normally signed and encrypted. After the download is completed, the OTA loader verifies the new code's signature and decrypts the downloaded code. Once the decryption and verification of new code is successful, the OTA loader writes it into FLASH (to replace the old software or store it in a new separate section and mark the previous code partition as backup) and updates the code version in non-volatile memory.

After update, the OTA loader may trigger a hardware reset to restart the secure boot process (as described in the previous section) to run the new code.

C. Cryptographic Algorithms

As described above, secure boot and update processes both employ digital signature schemes (and hash functions) to verify the authenticity of external keys and codes loaded from external memory into internal memory for execution or verify the authenticity of the new downloaded code via OTA.

Currently, the conventional digital signature schemes (such as RSA-PSS [11] and EC-DSA [12]) are utilized in code authentication. Because satellites are typically in operation for many years, it is important that the algorithms used stay unbroken for a long time. In recent years, a new risk has developed with the advent of quantum computers. If a large enough quantum computer could be built, Shor's algorithm [13] would render approaches based on factoring or discrete log (including RSA and ECC) insecure. To mitigate the potential risks from quantum computers, quantum-safe algorithms (also called post-quantum crypto – PQC) shall be used in conjunction with the above mentioned conventional algorithms (to create hybrid signatures, for example).

The recently introduced PQC algorithms could introduce new challenges to satellite systems due to the following reasons: 1) the size of keys and/or ciphertexts/signatures is much larger than conventional asymmetric ciphers (see the Section 3); 2) the operation of PQC algorithms is also more complex than conventional ones; 3) there is no clear best choice of PQC algorithm due to the concern of future attacks on those new ciphers. The rest of this paper will discuss the implementation of stateless PQC signature schemes and their impact on satellite systems. The stateful LMS and XMSS

schemes [25] are not discussed in this paper as they require careful state management and thus could cause operational complexity [26].

III. PQC Algorithms

The NIST PQC initiative aims to select good candidates in two different categories: signature algorithms, and key encapsulation mechanisms (KEMs). These two categories encompass most typical uses for asymmetric cryptographic algorithms. In this paper, we will focus only on signature algorithms as used in the secure boot and update processes.

As part of the PQC initiative, 23 submissions were evaluated by the cryptographic community, and eventually 3 signature schemes were chosen by NIST [14] – Dilithium, FALCON and SPHINCS+. Each of these algorithms uses a unique approach that is quite different from classical asymmetric ciphers.

A. Brief description

1) Dilithium

Dilithium [15] is an algorithm in the CRYSTALS (CRYptographic SuiTe of Algebraic LatticeS) family. As the name implies, this algorithm leverages lattice-based cryptography, in particular the Module Shortest Integer Solution (M-SIS) problem. The math is done over a finite field polynomial ring of degree 256. The design goals were conservative security, simplicity, and a small key and signature size.

2) FALCON

FALCON [16] is also a lattice-based cryptographic algorithm, this time based on the NTRU problem. The design goals were compactness and efficiency.

3) SPHINCS+

SPHINCS+ [17] is a hash-based algorithm, based on fairly old schemes such as Merkle trees and Winternitz signature schemes. Its security is based on the security of the underlying hash function. The design goals were to create a stateless algorithm based on a well-understood problem.

B. Algorithm Variations and NIST Security Levels

To evaluate the security of all PQC candidate algorithms, NIST has established a set of security levels [18]. The table below summarizes the PQC signature algorithms and their security levels.

Table 1 – PQC Algorithms Security Levels

NIST Security Level	Reference	PQC Algorithm
1	Brute-force search for AES-128 key	FALCON LogN9, SPHINCS+SHA256 128f/s
2	Random search for collision in SHA3-256	Dilithium2
3	Brute-force search for AES-192 key	Dilithium3, SPHINCS+SHA256 192f/s
5	Brute-force search for AES-256 key	Dilithium5, FALCONLogN10, SPHINCS+SHA256 256f/s

C. Key and Signature Size

To present an overall implementation impact, the public key size, private key size, and signature size of Dilithium, FALCON, and SPHINCS+ algorithms are summarized in the Table 2 below.

Table 2 – Key and Signature sizes of FALCON, Dilithium, and SPHINCS+

Algorithm	Public key size (bytes)	Private key size (bytes)	Signature size (bytes)
Dilithium 2	1312	2528	2420
Dilithium 3	1952	4000	3293
Dilithium 5	2592	4864	4595
FALCON LogN9	897	1281	657
FALCON LogN10	1793	2561	1271
SPHINCS +128s	32	64	7856
SPHINCS +192s	48	96	16224
SPHINCS +256s	64	128	29792

IV. PQC Implementations

In terms of performance, the NIST submissions have benchmarks but they are inconsistent. This paper attempts to benchmark the three signature algorithms in the exact same environment.

A. Hardware Platforms

As a proof of concept, we selected RISC-V processor platforms since its instruction set is open source and we are participating in the EU TRISTAN (Together for RISc-v Technology and ApplicatioNs) project [19].

1) ESP32C3 platform [20]

This platform has the following features:

- 32-bit RISC-V single-core processor, up to 160 MHz
- 384 KB ROM
- 400 KB SRAM (16 KB for cache)

2) Arty7 100T platform [21]

This platform integrates Altera FPGA with an implementation of the FreNox RISC-V processor provided by Technolution [22].

- 32-bit RISC-V single-core processor, up to 450 MHz
- 10KB SRAM
- 16MB FLASH
- 256MB DRAM

B. Implementation of PQC Algorithms

On the above two platforms, we used the reference code from FALCON, Dilithium and SPHINCS+ submitters to

implement the secure boot and secure update functions as described in Section 2.

The first version of the implementation uses the minimal subset of the ISA (rv32im), which is easy for porting onto any RISC-V processors. Furthermore, our implementation only focuses on the signature verification function of those PQC signature schemes introduced in Section 3 as the secure boot and secure update only need signature verification functions. The public key used for signature verification is hard-coded.

For this first reference implementation, we do not include any hardware acceleration. We will further investigate the possibility of optimizing PQC implementations by using some hardware accelerators such as floating-point instructions, big number hardware acceleration modules and/or certain crypto hardware accelerators like hash hardware. We are in discussions with Technolution regarding which hardware accelerators will be supported in new FPGA.

This initial implementation does not include any hardware interrupts, making the execution linear and predictable.

C. *Measurements*

Table 3 shows the code size and data size of our implementation of FALCON, Dilithium and SPHINCS+ signature verification functions.

Table 3 – Code and Data Size of FALCON, Dilithium and SPHINCS+s Implementations

Algorithm	Text (Bytes)	Data (Bytes)	Read-only Data (Bytes)	Total (Bytes)
Dilithium 2	27260	3744	1608	32612
Dilithium 3	27040	5264	1608	33912
Dilithium 5	26964	7200	1608	35772
FALCON LogN9	16040	1568	4768	22376
FALCON LogN10	16040	3088	4768	23896
SPHINCS+128s	9236	7904	764	17904
SPHINCS+192s	11540	16288	1512	29340
SPHINCS+256s	11540	29872	1512	42924

Figure 2 and 3 below show the execution time of a single signature verification over a payload of 1024-Byte code using FALCON, Dilithium and SPHINCS+s algorithms with different NIST security levels on Arty A7 100T and ESP32C3 platforms, respectively.

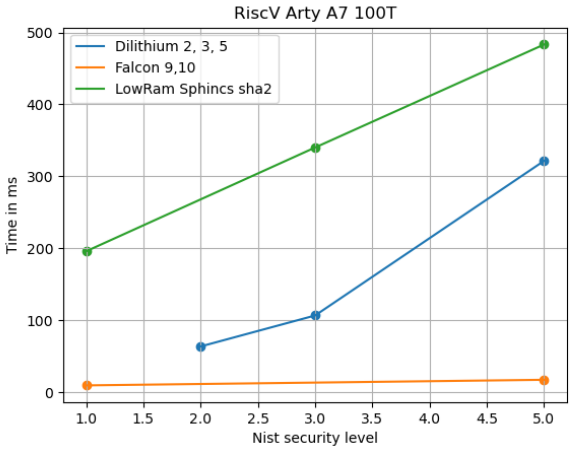

Figure 2 – FALCON, Dilithium and SPHINCS+s signature verification time on Arty 100T platform

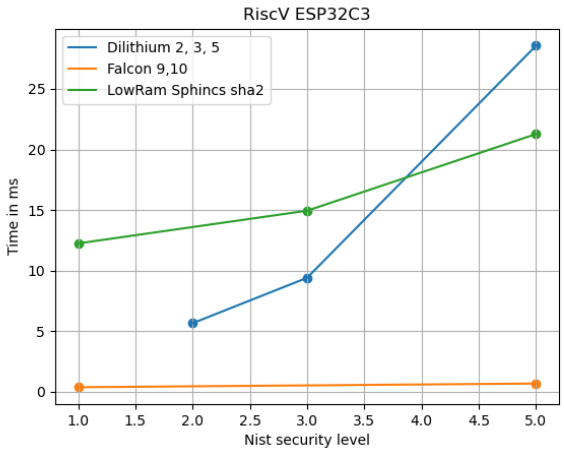

Figure 3 – FALCON, Dilithium and SPHINCS+s signature verification time on ESP32C3 platform

D. *Comparisons*

Table 3 manifests the range of memory footprints for the algorithms considered in this study: SPHINCS+128s and SPHINCS+256s have the smallest and largest footprints, respectively. Of the remaining algorithms, FALCON variants have the smallest footprints. Meanwhile, Figures 2 and 3 show that the time performance of all FALCON variants is much better than Dilithium and SPHINCS+s variants (though some variants do not have the same security level).

To the best of our knowledge, PQC benchmarking has been done mostly on Intel x86 and ARM Cortex-M3 and/or M4 processors [27]. This hinders a comparison with our RISC-V implementation.

Additionally, most (if not all) implementation of PQC algorithms on RISC-V are based on extended instruction set and/or take advantage of hardware acceleration [28].

Note that in our initial implementations, we have disabled all hardware interrupts, ensuring the execution is linear and predictable.

V. Impact on Satellite Systems

European Space Research and Technology Center has published SAVOIR (Space Avionics Open Interface Architecture) Flight Computer Initialization Sequence Generic Specification [23], which describes ESA requirements for the initialization of a typical spacecraft's on-board computer. We have analyzed the *Nominal Sequence* (described in Section 4.1.1 of [23]). The *Nominal Sequence* only contains two software modules: Boot SW and Application SW. The secure boot mechanism described in Section 2 of this paper has additional software modules such as boot loader and OTA loader. Implementing the secure boot and secure update functions (described in Section 2 of this paper) requires adding steps and/or modifying certain steps in the *Nominal Sequence* (described in Section 4.1.1 of [24]).

As described in Section 4, the implementation of secure boot based on PQC signature schemes requires roughly 30K bytes ROM memory for code (normally does not take much hardware resource) and roughly 10K bytes of data memory in SRAM (which might be significant in satellite systems). As for the signature verification time, it varies depending on the PQC algorithm chosen. FALCON (level 5) takes about 30 milliseconds, whilst Dilithium (level 5) takes 360 milliseconds on the Arty A7 100T platform, a platform which is intended for satellite systems. Evidently, adding a PQC based secure boot mechanism will increase the boot time of satellite systems, although the impact is acceptable as secure boot runs at the very beginning of boot sequence after power on and it is once-off till the next power on cycle.

Although the use of PQC algorithms leads to increased signature sizes, the impact on the transmission of signed code from a ground station to an in-orbit satellite is minimal: the packet format definition of the Packet Utilization Standard (PUS) [24] is designed to be flexible enough to transport variable length data. For very large transmissions (larger than 64K bytes), multiple packets are used. Using the PUS service in this way, the larger sizes of PQC-signed code are not an impediment, although of course the larger size of a typical PQC signature would take more time to transmit.

Last but not least, we should also consider the support for crypto agility in deployment. Our current implementation only supports one algorithm at a time, since the PQC algorithm is fixed at compilation. This might be acceptable in cases where the computer in the satellite system can be fully updated in-orbit. This is possible with an FPGA solution, in which the Boot ROM code is included in the FPGA bit file. If the FPGA option is not available, having multiple PQC algorithms pre-integrated in the Boot ROM is an option.

VI. Conclusion and Future Work

This paper presents a prototype of PQC algorithms based secure boot and secure update for satellite communication systems. For the purpose of secure boot and secure update, we do not see any blocking issues with implementing PQC-based secure boot and secure update mechanisms for satellite systems with the assumption that certain extensions or modifications of the existing boot sequence [23] and additional hardware resources are possible.

The results presented in this paper are our first-year research results for the TRISTAN project. During the remaining time of this project, we will continue to work with our partners on 1) optimizing the implementation of PQC algorithms and 2) continuing to follow up NIST's fourth round of PQC selection (which may add new PQC signature algorithms).

Acknowledgment

This work is sponsored partly by EU TRISTAN project (ID number 101095947). We would like to thank Tingting Lin, Xiaoxi Dong, Umut Demirci, Raghav Iyer, Md. Abu Faisal for their helpful discussions, as well as the Technolution team for their support. We would also thank anonymous reviewers for their constructive comments.

References

[1] O. Kodheli, E. Lagunas, N. Maturo, S. K. Sharma, B. Shankar, J. F. M. Montoya, J. C. M. Duncan, D. Spano, S. Chatzinotas, S. Kisseleff et al., "Satellite Communications in the New Space Era: A Survey and Future Challenges," IEEE Communications Surveys & Tutorials, 2020.

[2] Starlink, https://www.starlink.com/technology

[3] Technical Specification Group Radio Access Network; Study on New Radio (NR) to Suppoer Non-Terrestrial Networks (Release 15), Standard 3GPP TR 38.811, Technical Report, 2020.

[4] The Consultative Committee for Space Data Systems (CCSDS), Security threats against space missions; CCSDS 350.1-G-3, Feburary 2022.

[5] J. Willbold, M. Schloegel, M. Vögele, M. Gerhardt, T. Holz, and A. Abbasi, "Space Odyseey: An experimental Software Security Analysis of Satellites", IEEE Symposium on Security and Privacy (SP), 2023.

[6] Didelot Maurice-Michel, How to hack an ESA's experimental satellite. https://www.deadf00d.com/post/how-to-hack-an-esa-experimental-satellite.html

[7] L. Wouters, "Glitched on Earth by Humans: A Black-Box Security Evaluation of the SpaceX Starlink User Terminal", Black Hat USA 2022, Las Vegas, US, 2022 (https://www.youtube.com/watch?v=NXqLMmGwJm0)

[8] [online] https://infosectests.com/cissp-study-references/domain-8-app-dev/code-defects/

[9] Trusted Computing Group, "Trusted Platform Module Library Specification, Famliy 2.0", Revision 1.59, November 2019.

[10] Unified Extensible Firmware Interface (UEFI) Specification, Version 2.9, March 2021.

[11] IETF, RFC 8017, PKCS#1: RSA Cryptography Specifications, Version 2.2, November 2016.

[12] IETF, RFC 6979, Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA), August 2013.

[13] P.W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer", SIAM Rev., vol. 41, No. 2, pp.303-332, 1999.

[14] NIST IR 8413, Status report on the third round of the NIST Post-Quantum Cryptography Standardization Process, July 2022.

[15] Dilithium website, https://www.pq-crystals.org/dilithium/

[16] FALCON website, https://FALCON-sign.info/

[17] SPHINCS+ website, https://sphincs.org/

[18] PQC Security evaluation criteria, https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/evaluation-criteria/security-(evaluation-criteria)

[19] TRISTAN project, https://tristan-project.eu/.

[20] ESP32C3 platform, https://www.espressif.com/en/products/socs/esp32-c3

[21] Arty7 100T platform, https://digilent.com/shop/arty-a7-100t-artix-7-fpga-development-board/

[22] Technolotion FreNox RISC-V Softcore, https://www.technolution.com/advance/frenox/?noredirect=en-GB.

[23] ESA-ESTEC, SAVOIR Flight Computer Initialisation Sequence Generic Specification, Revision 2, Issed on November 18, 2021.

[24] ESA-ESTEC, Space engineering – Telemetry and telecommand packet utilization, ECSS-E-ST-70-41C, April 15, 2016.

[25] NIST SP 800-208, Recommendation for stateful hash-based signature schemes. https://doi.org/10.6028/NIST.SP.800-208

[26] T. Wiggers, K. Bashiri, S. Kolbl, J. Goodman and S. Kousidis, Hash-based signatures: state and backup management. https://www.ietf.org/archive/id/draft-wiggers-hbs-state-00.html

[27] M.J. Kannwischer, J. Rijneveld, P. Schwabe, K. Sto_elen, pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4. Workshop Record of the Second NIST PQC Standardization Conference (2019), https://eprint.iacr.org/2019/844

[28] P. Nannipieri, S. Di Matteo, L. Zulberti, F. Albicocchi, S. Saponara, and L. Fanucci, A RISC-V Post Quantum Cryptography Instruction Set Extension for Number Theoretic Transform to Speed-Up CRYSTALS Algorithms, IEEE Access, Volume 9, 2021. PP: 150798 – 150808, DOI: 10.1109/ACCESS.2021.3126208