[SSC22-XII-04]

First demonstration of a post-quantum key-exchange with a nanosatellite

Simon M. Burkhardt, Willi Meier, Christoph F. Wildfeuer
Institute for Sensors and Electronics, University of Applied Sciences and Arts Northwestern Switzerland
Klosterzelgstr. 2, 5210 Windisch; +41-562028792
simon.burkhardt@fhnw.ch

Ayesha Reezwana, Tanvirul Islam, Alexander Ling
Centre for Quantum Technologies, National University of Singapore
3 Science Drive 2, 120435; +65-86545795
cqtayes@nus.edu.sg

ABSTRACT

We demonstrate a post-quantum key-exchange with the nanosatellite SpooQy-1 in low Earth orbit using Kyber-512, a lattice-based key-encapsulation mechanism and a round three finalist in the NIST PQC standardization process. Our firmware solution runs on an on-board computer that is based on the Atmel AVR32 RISC microcontroller, a widely used platform for nanosatellites. We uploaded the new firmware with a 436.2 MHz UHF link using the CubeSat Space Protocol (CSP) and performed the steps of the key exchange in several passes over Switzerland. The shared secret key generated in this experiment could potentially be used to encrypt RF links with AES-256. This implementation demonstrates the feasibility of a quantum-safe authenticated key-exchange and encryption system on SWaP constrained nanosatellites.

Introduction

Security techniques for commercial satellites are poorly developed despite the rapid increase in the number of satellite missions. New constellations will increase the number of satellites dramatically over the next decade. It can be expected that with an increase in the number of operational satellites, the number of cyber-attacks on spacecraft communication will also increase.[1]

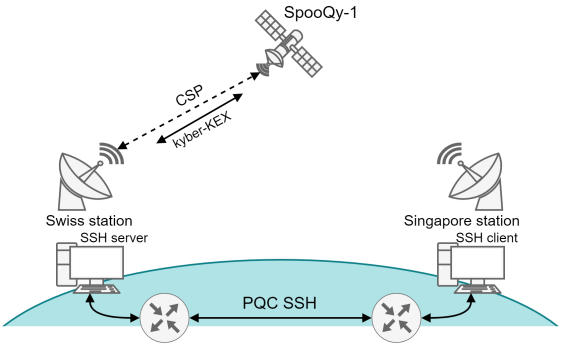

Figure 1: Satellite infrastructure used in this experiment

In SatCom systems, the symmetric key encryption algorithms - where the same key is used for both encoding and decoding messages - are frequently used. This approach to encryption in satellite communication requires that every party establishes a unique secret key with every other party with whom they would like to communicate. Furthermore, adding to this problem, each party must obtain all its secret keys in advance because possession of an appropriate key is a necessary prerequisite to establish a secure communication channel with another party. The number of shared key pairs that every party needs to store increases according to $n(n-1)/2$, where n may be the number of satellites in a constellation. Considering 100 satellites, this totals 4950 key pairs. Key-management can therefore become an important challenge for larger satellite fleets. To solve these issues, an asymmetric key encryption algorithm could be adopted where the key used to encrypt the message is different from the key used to decrypt the message. Such an approach requires each of the communication parties to maintain two keys only - one that is kept private and a second one that is made publicly available.

Using public-key exchange protocols for space applications has recently been proposed.[2] In most public-key systems the public keys are generated with RSA and Elliptic Curve Cryptography (ECC). However, it is now anticipated that Quantum Computers (QC) will be able to break both RSA and ECC when the technology to manufacture enough quantum nodes becomes available.

In order to solve this problem, the National Institute of Standards and Technology (NIST) has

initiated a process to solicit, evaluate, and standardize one or more quantum-resistant public-key cryptographic algorithms. The goal of Post-Quantum Cryptography (PQC), (also called quantum-resistant cryptography) is to develop cryptographic systems that are secure against both quantum and classical computers and can interoperate with existing communication protocols and networks. Kyber is one of the finalists in the NIST PQC project and it was chosen because it is a secure and efficient Key Encapsulation Mechanism (KEM), whose security is based on the hardness to solve the Learning-with-Errors (LWE) problem over module lattices.[3,4]

Our SpooQy-1 satellite consisted of two GomSpace components which handle communication, commands and data. The AX100 COM module had a SHA-1 Hash-based Message Authentication scheme (HMAC) but no encryption for the data. SHA-1 is well known for being insecure.[5]

The second module, the A3200 On-Board Computer (OBC) was used for flight controls and mission software. The OBC ran the open source libcsp[6] implementation of the CSP which offers optional encryption using a 128-bit symmetric XTEA algorithm. With the goal of moving towards a quantum secure satellite infrastructure (including CubeSats), it is crucial to embed PQC into current hardware and software projects. Many of the CubeSats run on Size, Weight and Power (SWaP) constrained onboard computers. We present a demonstration of a successful key exchange with this experiment using the older AVR32 microcontroller architecture as well as some performance measurements on the more recent ARM Cortex-M4 architecture. Figure 1 shows our setup of the ground stations and the SpooQy-1 satellite.

SpooQy-1 CubeSat

Development and objectives

The main objective of SpooQySat, the SpooQy-1 CubeSat, was to demonstrate an in-orbit space-compatible quantum light source SPEQS (the Small Photon Entangling Quantum System) to increase the Technology Readiness Level (TRL) of future global Quantum Key Distribution (QKD) networks. QKD is a family of secure communication techniques used to generate private and shareable random secret keys that can be exchanged between two parties only. Essentially, QKD requires the exchange of individual photons and therefore very low-loss optical links need to be established. Optical fibers are limited to about 100 km before losses become overwhelming. Free-space optical losses are much lower. However, the main drawback for free-space QKD is key exhaustion due to failed key generation because of bad weather. Here PQC can provide a fallback solution if keys cannot be exchanged by QKD. Also, PQC may become the standard for encrypting the RF wireless satellite data links since it does not require special hardware and optical communication. That was the motivation to implement a PQC algorithm on SpooQy-1 after the main objective for the mission had been accomplished.

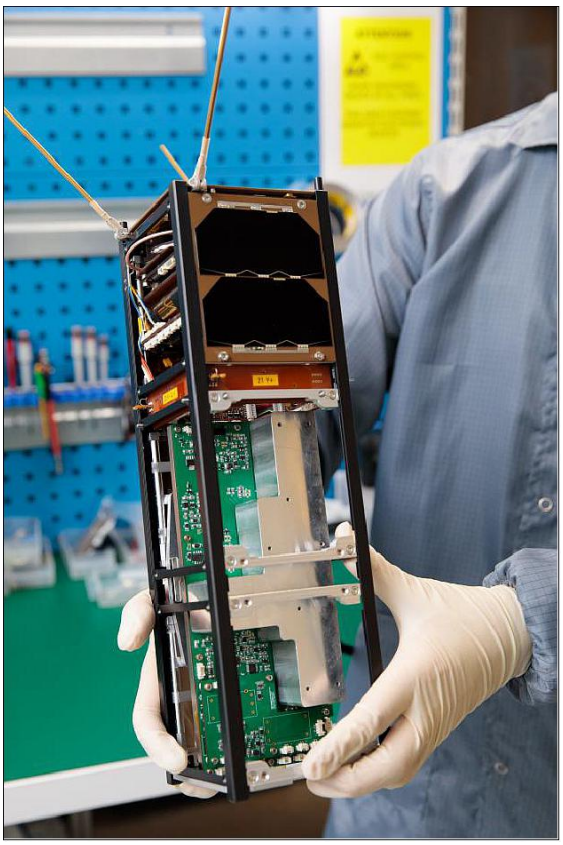

Figure 2: Partially integrated engineering model of SpooQySat, a 3U CubeSat. Removed solar panels reveal structural model of the SPEQS payload.

Experiments of a basic SPEQS source started in 2012 with high-altitude balloon tests followed by a correlated SPEQS source in 2013.[7,8] In 2016, a Space-qualified, correlated SPEQS source was tested in low Earth orbit on the NUS Galassia CubeSat.[9]

Figure 3: Singapore UHF ground station on the roof top at NUS campus.

Figure 4: Switzerland UHF ground station at the campus of FHNW in Windisch.

SpooQy-1 was then designed and built at the Centre for Quantum Technologies, National University of Singapore to demonstrate an entangled photon pair-source in space. SpooQy-1 was deployed to Low Earth Orbit (LEO) from the international space station on 17th June 2019 and provided the first demonstration of entanglement in space on a nanosatellite.[10] In Figure 2 the partially integrated engineering model is shown. Fully assembled, the CubeSat mass is 2.6 kg, and its peak system power consumption is 3.9W.

The Singapore ground station is located on top of an eighteen storied building at the NUS campus shown in Figure 3. A secondary UHF ground station, shown in Figure 4, is established in Switzerland to provide additional data download opportunities. The ground stations are built using the GomSpace UHF hardware and have identical setups. Both ground stations are equipped with a twinned Yagi antenna with a tracking mount. The rotor is controlled by a Linux based server computer (NanoCom MS100). The ground station radio (NanoCom GS100) is the ground counterpart (with a 25 W power amplifier) for the NanoCom AX100 radio on-board SpooQy-1, designed specifically as an integrated component to request/respond via the CSP protocol during operation.

AVR32 platform

The SpooQy-1 nanosatellite uses the NanoMind A3200 on-board computer from GomSpace which utilizes a Microchip AT32UC3C0512C micro controller running a real time operating system (FreeRTOS) along with proprietary mission specific software. On-board are 128MB of external flash storage which can be accessed through the C stdlib file IO functions. On the flash memory there is a FAT file system present which can be accessed through an FTP implementation for CSP. An additional 32MB of SDRAM can be used to load a binary RAM image file `nanomind.elf` from the file system and boot from there. This enables the satellite with the capability to run new code once it is in orbit.

Firmware framework

GomSpace delivers the NanoMind with a Software Development Kit (SDK) and documentation to build and expand mission firmware for their AVR platform. The SDK consists of a fully featured mission control software with the software parts. The Figure 5 shows a visual representation of the software components that are involved both in the ground station and the satellite.

Our goal was to implement the Kyber algorithm alongside this mission control software to demonstrate that the Kyber source code can run on the AVR32 platform. The following chapter describes the challenges, solutions and recommendations when using PQC algorithms on satellite hard- and firmware.

Implementing the key exchange

Kyber key encapsulation mechanism

The PQ CRYSTALS Kyber algorithm is a quantum secure Key Encapsulation Mechanism (KEM). A KEM can be used in combination with a Key Derivation Function (KDF) to generate a common symmetric key. In the case of Kyber, SHA-256 is used as the KDF. The reference implementation from PQ CRYSTALS contains the API source code for such a KEM inside `kex.c` as well as a principal protocol definition in section 5 of.[3] The API offers two types of key exchange: the "Unilaterally Authenticated Key Exchange" (UAKE) and the "Mutually Authenticated Key Exchange" (AKE), which is the preferred and most secure method. Whereby the authentication does not authenticate the participants but guarantees that each party has derived the same symmetric key. For user authentication a separate algorithm like Dilithium[11] would be required. In our experiment we had the advantage of using pre-shared secrets to authenticate both parties using HMAC that had already been implemented in the AX100 COM module. However, since the implemented HMAC is based on SHA-1 it is not quantum-safe.

AVR32 toolchain

Compilation of the GomSpace firmware is done using the AVR32 tool chain (version 3.4.2). It contains `avr32-gcc` (gcc version 4.4.7) for Debian-based Linux systems. A drawback of this outdated C compiler is that it only supports C language up to the C99 standard. The Kyber implementation is included in the liboqs project from the open quantum safe organization.[12] Liboqs is a sandbox to experiment with many different PQC algorithms which are participating in the NIST standardization process. However, this project requires C11 standard and utilizes functions like `aligned_alloc` which are unavailable from the `avr32-gcc` compiler. We therefore focused on the standalone Kyber algorithm rather than including multiple NIST candidate algorithms. In a first step, the original Kyber source code from the PQ CRYSTALS organization[13] was integrated into the AVR32 auto build system for the NanoMind. This step includes platform specific adjustments to the Kyber source code like the random number source.

Random number generators

To guarantee quantum safety, the Kyber algorithm requires a Random Number Generator (RNG) that can produce 256 bits of entropy. The original Kyber implementation uses the `/dev/urandom` pseudo-random number generator on Linux based systems. Although being pseudo-random, it is considered to be safe for cryptographic applications.[14,15]

SpooQy-1's OBC is only running an RTOS and not a full operating system which would offer such a secure RNG. This is why the Kyber source code was modified to replace reading from `/dev/urandom` with the pseudo-random function `rand()` from the `avr32-gcc` stdlib. This affects the `randombytes()` function used during the asymmetric key pair generation. Another issue is that on AVR32 a True Random Number Generator (TRNG) is missing. For time reasons we decided not to implement SpooQy-1's on-board Quantum Random Number Generator (QRNG) into this experiment. In a practical application the `rand()`-function is the weakest point of failure because the private keys from pseudo-randomly generated key pairs can sometimes be recovered as demonstrated in.[16] We strongly advise, using a TRNG or even a QRNG to get the required 256 bits of entropy for Kyber. As a less secure alternative one can seed the PRNG using a true random number or a pre-shared secret seed using `srand(seed)`. In this experiment we used the default seed. It should also be pointed out that the ground station is using the secure RNG, as depicted in Figure 6. If the ground station initiates the key exchange, the random ingredients for the common secret are therefore cryptographically secure. This would not be the case, if the satellite initiates the key exchange.

Practical key exchange application

The original implementation in `test_kex.c` from the Kyber source code served as a reference implementation for our AVR32 application.[13] Both the SDK for the satellite and the ground station allow the developer to implement callback handlers for custom features on both ends (in our source code: `kex_pub.c` and `kex_kyber.c`, respectively[17]). We implemented the Kyber API into callback functions for the satellite's command parser. Limited by the

available time to develop such an implementation, we decided to implement only the satellite's back end and perform the message exchange manually using the File Transfer Protocol (FTP). All keys and temporary arrays are not just stored in RAM but also in hex-format as ASCII characters on the NanoMind's and ground station's file systems. This has another practical reason: in case of a reboot the keys can be recovered from the flash memory. During the key exchange the encapsulated message files are not exchanged automatically but manually using FTP upload and download commands. For this purpose, we implemented a way to read and write these message files on both stations. The two code snippets in Listing 1 and Listing 2 show the principle behind the code that was executed as part of our key exchange experiment. The full code is available in our Github repository.[17]

```
FILE *fp;
fp = fopen(filename, "w+");
for(int i=0;i<(int)bytes;i++)
  fprintf(fp,"%02x", in[i]);
fclose(fp);
```

Listing 1: Source code of how the messages are written to files inside writeHexFile(filename,in,bytes).

```
readHexFile("/flash/ake_senda.txt",
  ake_senda, KEX_AKE_SENDABYTES);
kex_ake_sharedB(ake_sendb, ka, ake_senda,
  ska, pkb); // Run by Bob
writeHexFile("/flash/COMMON.key", ka,
  KEX_SSBYTES); // final key
writeHexFile("/flash/ake_sendb.txt",
  ake_sendb, KEX_AKE_SENDBBYTES);
```

Listing 2: Source code of how kex_ake_sharedB() is performed with ake_senda.txt as input.

The file "COMMON.key" then contains the HEX representation of the exchanged key. This key is 32 bytes long, or 64 characters if stored in ASCII HEX format using `printf(%02x)`.

```
f02473a6ab18617b3e0dbcc565b4b64e23
  f12a284a6dbfbf5cd3cde4ac5e2e21
```

Listing 3: The contents of COMMON.key after the successful key exchange.

Communication channel

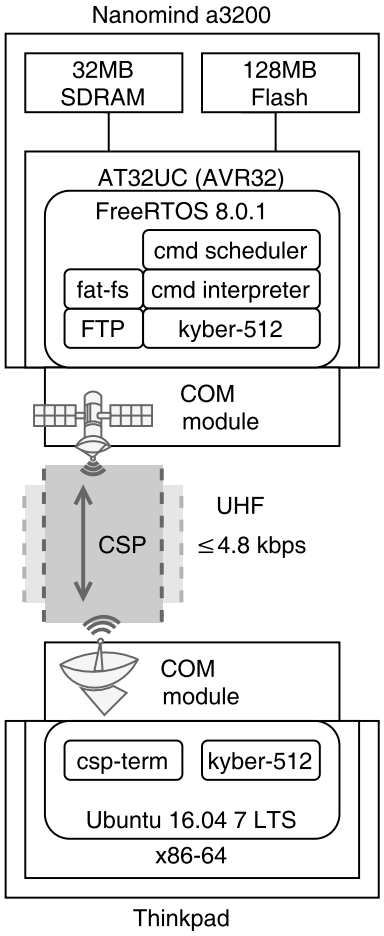

Figure 5: Hardware and software components involved in the setup.

Resource utilization

As shown in Table 1 our application of Kyber-512 uses approximately 13 kB more flash (`text`) and 8 kB more RAM (`bss`) than the default mission firmware without Kyber. The additional 8032 bytes in the RAM are a direct consequence of the several global arrays required to store the keys and messages. This is an increase of 40x (flash) respectively 500x (RAM) compared to the insecure XTEA implementation which uses a 128-bit key. If compiled as a RAM image to boot from the SDRAM, less RAM is available for the program which limits the RAM usage of the mission firmware. This is especially relevant for the large arrays used to hold the keys. In our case there is only enough RAM to have temporary arrays for one key exchange. The on-board

simulation of a key exchange using two parties would not fit into the memory.

Table 1: Memory usage (in bytes) on AVR32 compared to the default project without any cryptographic function, when compiling for a RAM image.

Algorithm	flash	RAM
default (none)	467'608	31'024
XTEA	+344	+16
SHA1-HMAC	+1'856	+16
XTEA & SHA1-HMAC	+2'192	+32
Kyber-512	+12'976	+8'032
Kyber-718	+13'016	+11'680
Kyber-1024	+13'248	+15'808

Exchanging keys

Setup

For the demonstration we assume that the ground station is "Alice" and SpooQy-1 is "Bob". SpooQy-1 has the firmware with the Kyber-512 implementation uploaded and booted. The ground station has two pieces of software running: the csp-term and our executable implementation of the Kyber API. We use the command scheduler on the satellite to schedule commands that are unknown to the ground station terminal.

Two ground stations had been used. One at the NUS Campus in Singapore and one at FHNW in Switzerland. The two ground segments were connected through a quantum-safe version of the Secure Shell Protocol (SSH), which simplified collaborative work, since all ground stations could be remote controlled.

Experiment

Figure 6 shows the performed key exchange where Alice is the initiator. In a first step, one secret/public key pair is generated each for Alice (sk_A, pk_A) and for Bob (sk_B, pk_A). Next, the public keys are exchanged between the two stations by downloading/uploading the text files. Alice then starts the key encapsulation mechanism by generating a third key pair (esk_A, epk_A) which is used as the basis for the common secret key. Note that this key pair is generated using a cryptographically secure RNG. The output of both the key generation (epk_A) and start of the authentication (c_2) is then uploaded to Bob. Bob then performs several encapsulation and decapsulation operations, which enables him to use several outputs (K, K_1, tk') to derive the final key. The second output from Bob's encapsulation is then downloaded to Alice again, where a final decapsulation generates the same components for the key derivation as Bob already has. Alice and Bob are now in possession of the same common key. To verify this, we downloaded Bob's key to the ground station to compare it with Alice's key. The full command sequence is shown in the Listing 4.

```
0  GND > kex-init
1  SAT > kex_kyb -i
2  GND > ftp_upload ./PKA.key /flash/PKB.key
3  GND > ftp_download /flash/PKA.key ./PKB.key
4  GND > kex-pub -A
5  GND > ftp_upload ./ake_senda.txt
6  SAT > kex_kyb -B
7  GND > ftp_download /flash/COMMON.key
8  GND > mv ./COMMON.key ./SATELLITE.key
9  GND > ftp_download /flash/ake_sendb.txt
10 GND > kex-pub -C
11 GND > diff ./COMMON.key ./SATELLITE.key
```

Listing 4: command sequence for the KEX experiment

Benchmarking SSH with PQC algorithms

Since SpooQy-1 reentered Earth right after we performed the key exchange, we could not do benchmarking studies for the RF link. However, we performed tests for a quantum-safe version of the SSH protocol that we used in the Singapore-Switzerland internet link. Here we benchmarked the new NIST round 3 candidates against currently used Elliptic-Curve Diffie-Hellman (ECDH). In Figure 7 we show the results for the handshake times in the quantum-safe version of SSH that show the average over 1000 handshakes as a function of different key-exchange algorithms. For all algorithms the authentication was performed with Dilithium 2. It appears that the lattice-based algorithms perform similarly and on par with ECDH. However, the code-based algorithm classic McEliece is taking more time since its public key is substantially larger.

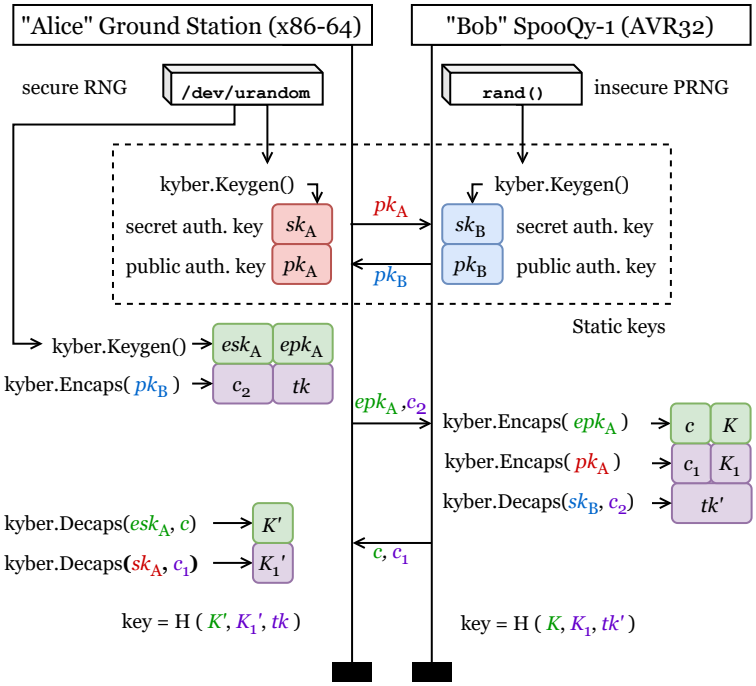

Figure 6: Mutually authenticated key exchange where the ground station is the initiator.

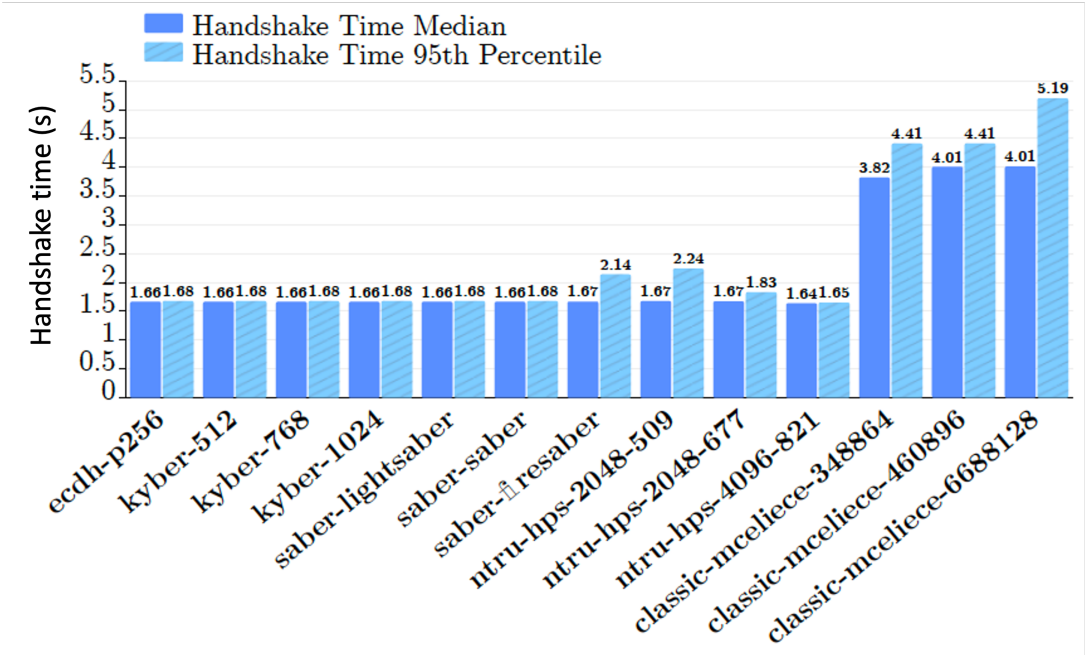

Figure 7: Total handshake time in seconds as a function of the round-3 NIST key-exchange algorithms. Median (dark-blue) and 95th percentile (light-blue). The 200 ms round trip time between Singapore and Switzerland is included. The classical key-exchange (ecdh-p256) is shown as a baseline. All key-exchange algorithms are authenticated with Dilithium 2 and the numbers shown are from averaging over 1000 handshake times.

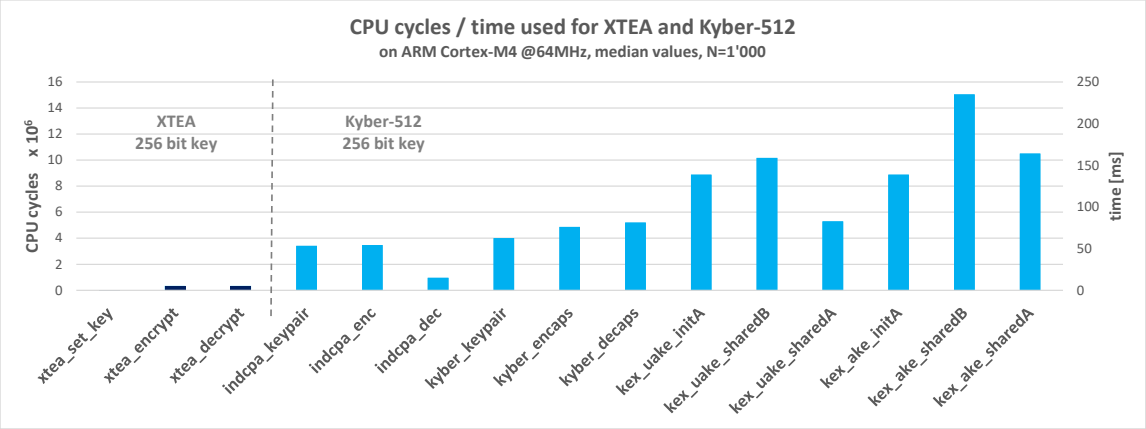

Figure 8: Performance measurements on an ARM Cortex-M4 microcontroller.

We also conducted benchmarking measurements to compare Kyber-512 to XTEA. Keep in mind that XTEA is a symmetric block cipher while Kyber is an asymmetric key encapsulation mechanism. The comparison between the two algorithms should therefore only serve as a point of reference for embedded developers. Since XTEA is the only cryptographic function implemented in libcsp it does make sense to compare its computational effort to the proposed Kyber-512 algorithm. For the performance measurements we use a hardware that is comparable to SpooQy-1's OBC. Here, we used the STM32F407VG microcontroller based on the ARM Cortex-M4 RISC architecture. Figure 8 shows the median values for 1'000 iterations of various cryptographic functions executed on this microcontroller.

We see that Kyber-512 needs substantially more resources than XTEA. However, as OBCs on CubeSats become more powerful and the trend is towards using higher performing architectures like the ARM Cortex-A9 (NanoMind Z7000), PQC algorithms should present a feasible alternative to their classical counterparts.

Conclusion

We were able to demonstrate a quantum secure key exchange with a nanosatellite in low Earth orbit using the Kyber-512 KEM API. Implementing a PQC algorithm on an embedded micro controller brings new weaknesses that need to be addressed by the developer, such as the usage of a cryptographically secure RNG. There is also a potential risk that the implementation of the new algorithms do not support the old MCU architecture currently used for developing nanosatellites. The integration of Kyber into the libcsp project is planned as a follow-up project at FHNW. Since SpooQy-1 has also demonstrated recently a working QRNG,[18] we may use those random numbers in the next satellite mission SpooQy-2 as a cryptographically secure RNG instead of the PRNGs described previously. For a future project one could take on the task of improving the security features of libcsp by implementing the Kyber algorithm for example. There would however be the challenge of providing a secure RNG which cannot be programmed into the library generically.

Acknowledgement

We thank Raffael Anklin for his help with the AVR32 microcontroller, and Frank Imhof for benchmarking the SG-CH connection.

References

[1] Manulis, M., Bridges, C.P., Harrison, R. et al. Cyber security in New Space. Int. J. Inf. Secur. 20, 287–311 (2021), https://doi.org/10.1007/s10207-020-00503-w

[2] A. Godhwani, M. Murfield, T. Delaney, Kok-Song Fong, P. Browne and S. Hryckiewicz, "The use of PKI in next generation UHF SATCOM," 2011 - MILCOM 2011 Military Communications Conference, 2011, pp. 1733-1738, https://doi.org/10.1109/MILCOM.2011.6127561.

[3] J. Bos et al., "CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM," 2018 IEEE European Symposium on Security and Privacy (EuroS&P), 2018, pp. 353–367, https://doi.org/10.1109/EuroSP.2018.00032

[4] NIST standardization process for quantum-resistant public-key cryptographic algorithms,

Online, https://csrc.nist.gov/Projects/Post-Quantum-Cryptography

[5] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, Y. Markov, "The First Collision for Full SHA-1," in: Katz J., Shacham H. (eds) Advances in Cryptology – CRYPTO 2017. CRYPTO 2017. Lecture Notes in Computer Science, vol 10401. Springer, Cham, https://doi.org/10.1007/978-3-319-63688-7_19

[6] J. De C. Christiansen, J. Ledet-Pedersen, P. Wullf, Y. Shoji and D. E. Holmstroem, "Cubesat Space Protocol - A small network-layer delivery protocol designed for Cubesats," Online on Github, 2011, https://github.com/libcsp/libcsp

[7] W. Morong, A. Ling, D. Oi, "Quantum Optics for Space Platforms," Optics & Photonics News 23(10), 2012, pp. 42–49 https://doi.org/10.1364/OPN.23.10.000042

[8] Z. Tang, R. Chandrasekara, Y.Y. Sean, C. Cheng, C. Wildfeuer, A. Ling "Near-space flight of a correlated photon system," Sci. Rep., 4 (6366), 2014, https://doi.org/10.1038/srep06366

[9] Z. Tang, et al. "Generation and analysis of correlated pairs of photons onboard a nanosatellite," Phys. Rev.Appl, 5 (054022), 2016, https://doi.org/10.1103/PhysRevApplied.5.054022

[10] A. Villar, A. Lohrmann, X. Bai, T. Vergoossen, R. Bedington, C. Perumangatt, H. Y. Lim, T. Islam, A. Reezwana, Z. Tang, R. Chandrasekara, S. Sachidananda, K. Durak, C. F. Wildfeuer, D. Griffin, D. K. L. Oi, and A. Ling, "Entanglement demonstration on board a nanosatellite," Optica 7, 2020, pp. 734–737, https://doi.org/10.1364/OPTICA.387306

[11] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler and D. Stehlé, "CRYSTALS-Dilithium Algorithm Specifications and Supporting Documentation", IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES), 2018, Issue 1, https://doi.org/10.13154/tches.v2018.i1.238-268

[12] Open Quantum Safe organization, "C library for prototyping and experimenting with quantum-resistant cryptography," Online on Github, 2018, https://github.com/open-quantum-safe/liboqs

[13] J. Bos et al., "Kyber," Online on Github, 2018, https://github.com/pq-crystals/kyber

[14] T. Fischer, "Testing Cryptographically Secure Pseudo Random Number Generators with Artificial Neural Networks," 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), 2018, pp. 1214–1223, https://doi.org/10.1109/TrustCom/BigDataSE.2018.00168

[15] Federal Office for Information Security, atsec information security GmbH, "Documentation and analysis of the linux random number generator", Version 4.11, 2022, https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/LinuxRNG/LinuxRNG_EN_V4_5.pdf

[16] N. Heninger, Z. Durumeric, E. Wustrow and J. A. Halderman, "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices," *USENIX Security Symposium.*, vol. 12, 2012, pp. 205–220

[17] S. M. Burkhardt, " fhnw-ise-qcrypt/PQKEX-nanosat-src", FHNW ISE (Post-)Quantum Cryptography Group on Github, 2021, https://github.com/fhnw-ise-qcrypt/PQKEX-nanosat-src

[18] A. Reezwana, T. Islam, J. A. Grieve, C. F. Wildfeuer and A. Ling, "Generating Quantum Random Numbers on a CubeSat (SpooQy-1)," 2020 Conference on Lasers and Electro-Optics (CLEO), 2020, pp. 1–3, https://10.1364/CLEO_AT.2020.ATu3S.3