Post-Quantum Stateful Hash-Based Signature Scheme for Improved Bluetooth Security

Peter Anthony
Department of Electrical and Computer Engineering
California State Polytechnic University, Pomona
paanthony@cpp.edu

Andres Colon
Department of Electrical and Computer Engineering
California State Polytechnic University, Pomona
colon@cpp.edu

Ian Lieu
Department of Electrical and Computer Engineering
California State Polytechnic University, Pomona
ielieu@cpp.edu

Arron Lu
Department of Electrical and Computer Engineering
California State Polytechnic University, Pomona
arronlu@cpp.edu

Dr. Mohamed El-Hadedy (Aly)
Assistant Professor
Department of Electrical and Computer Engineering
California State Polytechnic University, Pomona
mealy@cpp.edu

Dr. Valerio Formicola
Cybersecurity Scientist at Siemens, NJ
valerio.formicola@siemens.com

California State Polytechnic University, Pomona, CA 91768
March 28, 2021

Abstract – The existence of quantum computers poses a threat towards current security primitives, which rely on forms of prime number factorization, through Shor's Algorithm. This brings great concern in securing the data communications with nuclear power plants, bank transactions, satellite communications, and many more security sensitive operations. The National Institute of Technology (NIST) considers the advent of quantum computing as a matter of the highest concern and estimates the potential damages at a minimum of 1 billion dollars. To combat this pressing issue, NIST has been inviting scientists to contribute to post-quantum cryptographic primitives that can be used to resist quantum attacks. Postquantum cryptography is one of the best methods available to secure current applications, which are heavily involved in trading and moving data through unsecured channels. The use of internet is so broad that it is used in both high-performance computing facilities as well as in Internet of Things applications. These cloud-based services support numerous fields including the medical, industrial, and military sectors. Due to the necessity of the Internet in the modern era as well as the threat quantum computers pose, these communications must be secured with new encryption, digital signatures, and security standards if they hope to be useful in the post-quantum era. The many different types of post-quantum algorithms, lattice-based cryptography, multivariate cryptography, hash-based cryptography, code-based cryptography, super singular elliptic curve isogeny cryptography, symmetric-key quantum resistance, are outcomes of multiple worldwide competitions driven by NIST. The adaption of these algorithms on both hardware and software platforms are a big question since not that many attempts exist to study these primitives and how they can efficiently be implemented to fit different applications from high performance ones to the little IoT applications. Our team aims to tackle a part of this problem through the implementation of the LMS scheme, a hash-bashed cryptography algorithm, on two FPGAs communicating with each other through Bluetooth.

I. Introduction

Our team is implementing a hash-based signature scheme on an IoT low powered device to simultaneously improve the security of Bluetooth and to accelerate the key generation time of the scheme. The major problem with implementing a quantum robust hash-based signature scheme is that the key generation time can be significantly long when implemented with its native parameters. Our solution is to test both a Light Weight Crypto (LWC) and hardware/software partition for accelerated hashing. In addition, Bluetooth is known to be insecure regarding current security standards; however, this is mainly to comply with resource constraints such as power. By passing hash-based signatures through Bluetooth, the security will be enhanced without sacrificing performance. This research paper will discuss the following topics: Section II. Post-Quantum Algorithms: Leighton-Micali Signatures (LMS) vs eXtended Merkle Signature Scheme (XMSS), Section III: LMS Performance Comparison on Desktop PC vs PYNQ-Z1, Section IV: LMS Problem Solutions, & Section V: AES and Bluetooth Overview with Project Implementation, Section VI: Conclusion.

II. Leighton-Micali Signatures (LMS) vs eXtended Merkle Signature Scheme (XMSS)

To combat the threat that post-quantum computing poses to current cryptography efforts, two signature schemes have been recognized as quantum secure: LMS and XMSS. LMS and XMSS operate, generally, the same way in that they both use a Hash Based Signature (HBS) scheme. Both algorithms use a Merkle Tree and a One Time Signature (OTS) scheme to generate public and private key pairs to use for signing purposes [5]. The OTS scheme refers to a signature scheme where every private key can only be used to sign a message and its corresponding public key can only be used to verify the signed message once, which ensures that collision attacks are rendered useless. The Merkle tree is basically a binary tree where the leaves of the tree are the OTS public keys. The nodes are then hashed together until the root is found [6]. For a signature, the OTS signature, the corresponding public key or root, and the path from the leaf to the root of the Merkle tree are used. A verifier would then take that information and would compute the root of the Merkle tree to see if the resulting values were the same. If the resulting root was the same, then the signature would be verified and if the resulting root was different, then the signature would not be verified. Where the two HBS schemes differ is that LMS uses WOTS instead of WOTS+ [6]. In essence, XMSS uses a system that introduces an extra layer of randomness, and thus security, into its signature scheme; however, the introduction of the randomness makes key generation, signing, and verifying much longer than LMS for tangible security benefit. As a result, it was concluded that while XMSS might be theoretically more secure, the increased time it required for key generation, verification, and public key generation would not be worth its application. Instead, a focus on LMS would allow for better power usage while still applying the core functions of HBS schemes.

III. Test Results

The runtime is the most significant performance metric for the implementation of LMS because it determines how it can be used and what it can be used for. In terms of runtime, the signing and verifying times are negligible because they are both executed within milliseconds no matter what device they are run on. The key generation time is the most significant parameter because both private and public keys must be generated; this means the entire Merkle tree and its WOTS chains must be generated in this time.

The default LMS signature system was tested on two environments: a desktop running Ubuntu 18.04 and on the PYNQ-Z1's ARM processor also with Ubuntu 18.04. The desktop had an Intel Core i7-6700K CPU clocked at 4.0 GHz with 8 threads while the PYNQ-Z1 had an ARM Cortex-A9 dual-core processor clocked at 650MHz. Because the key generation function utilizes multithreading, the performance on the desktop is expected to be significantly better than the performance on the ARM processor.The key generation time on the desktop was about 5 minutes, while on the ARM processor, it was just under 3 hours. Though a large performance difference was expected between the two environments, a difference in run time this large was quite unexpected. The most resource intensive part of the key generation is the hashing utilizing SHA-256, which takes up 92% of the entire key generation time [6]. This leads to two potential solutions for accelerating the algorithm: the adaptation of a Lightweight Crypto (LWC) Hash in software or the adaptation of a hardware/software partition using a dedicated hardware SHA-256 hashing core.

IV. Adapting a Lightweight Hash Function into LMS

The security of LMS is based on the strength of the hash function that it uses. In terms of security, any given LWC hash function may be less secure than LMS's default SHA-256; therefore, an LWC hash may not provide the same level of security that SHA-256 does. It would, however, allow the implementation of a LWC solution of LMS on a low-power IoT device such as the PYNQ-Z1. A LWC hash used with LMS may not provide enough security for a quantum computer; however, considering the power and time constraints on an IoT device, it would provide the best key generation time for a purely software implementation of LMS on an IoT device.

The second solution involves a hardware/software partition between the LMS scheme and a hardware hash core. Currently, the LMS code that we are implementing can use two different forms of the SHA-256 function. One form is written in C and is used in cases where the device does not have access to the open-ssl library [10]. If the device has access to open-ssl, then the assembly version of SHA is utilized instead. Due to the inherent differences between C and assembly, we have concluded that further optimization of the SHA-256 in software would be nearly impossible. Instead of pursuing further optimization of the hash in software, the utilization of a hardware implementation of SHA-256 in conjunction with LMS would significantly reduce the key generation time. Theoretically, a hardware/software partition could reduce the time needed by 5 to 6 times, allowing the PYNQ-Z1 board to generate keys in 36 minutes compared to the previous 3 hours. There are two other significant benefits to this approach. The first is that the power consumed on the PYNQ-Z1 will be reduced because a significant portion of the computations will be performed on the fabric of the board. The second benefit is that the integrity of LMS will be preserved by using the same hash function, but in hardware. As previously mentioned, the use of a LWC algorithm could result in a decrease in security depending on the security of the LWC algorithm. With a hardware/software partition, the full security of LMS can be expected while providing a substantial speed up in performance.

V. Bluetooth Implementation

Bluetooth, more specifically the Low Energy, sees use on thousands of devices such as smart watches, IoT devices, car systems, and many other applications. The reasoning behind this is the low power draw as well as providing a wireless communication service. The downside is that sacrifices to security have been made in order to run on resource constrained devices. Bluetooth currently uses AES encryption as the standard for securing the data exchange between devices. In order to encrypt and decrypt data, AES relies on a key exchange to create a single key that communicating devices share. How it works is each device creates their own short-term key, then they agree on a temporary key combined with a nonce (random number), that public temporary key is then combined with the short-term key to create a private key [12]. Once this is completed the devices now share a secret key that only the two devices know. This exchange takes place during the pairing process during an initial Bluetooth connection. This encryption scheme has proven secure with current technology but with the emergence of quantum machines, it becomes easily breakable. To secure the data between devices against these computers Bluetooth security must move to a signing model, like LMS which proves authenticity, as opposed to the key exchange model AES currently uses.

Since LMS is focused on the enhancement of data integrity and security, and this is something Bluetooth Low Energy falls short on, we decided to implement it alongside this communication protocol. Due to the complexity of the protocol, a Bluetooth stack is required which handles the exchanging of data and authentication between devices. The physical hardware used to communicate are two PYNQ-Z1 boards, each using the Espresssif ESP-WROOM-32 PMOD (Peripheral Module). To allow communication between the board and the PMOD, UART communication is established on both PMOD ports of the PYNQ-Z1. The software Bluetooth stack uploaded to the esp32 uses Bluekitchen's Btstack [11]. This stack is both open source and compatible with the PMOD which makes it a desirable candidate. When a message on one PYNQ board is created and signed using LMS it then sends the data via a UART (universal asynchronous receiver-transmitter), which is a wired communication protocol, to the esp32 which then transmits the data over Bluetooth. The second board can then receive the message and verify that the data is correct and has not been tampered with. Any updates to the status of this project as well as any programs made for use with this project will be uploaded to the GitHub Repository for this project linked in the appendix [13].

VI. Conclusion

At the time of this report's submission, we have successfully adopted a new hash function – SHA-3 – into LMS as a proof of concept for replacing the hash function with both LWC and hardware implementations. Adding an LWC hash to LMS should follow the same process as adding SHA-3; however, not all LWC hashes are compatible with the program due to input/output constraints. Similarly, a SHA-256 hardware core would only require the inclusion of a driver and integration of the HDL code onto the fabric of the PYNQ-Z1. For the purposes of this project the ideal scenario is to replace the embedded AES encryption within the Bluetooth stack with a hash-based signature scheme. Due to time constraints, this may not be fully implemented; however, it would be an ideal solution to Bluetooth's security issues. Although it may not be the most efficient method of implementation, the passing of hash-based signatures through Bluetooth will provide a significant increase in security. The completion of this project aims to provide a substantial step towards a cryptographically secure the post-quantum era for IoT devices and any application that would require an accelerated key generation time.

Appendix

The following are steps to get Cisco's hash-sigs program running on the PYNQ:

1. Create a bootable SD card for PYNQ Z1 using Petalinux. [9]
2. Download and install the Ubuntu root file system on the root of the SD card. [9]
3. Copy Cisco's 'hash-sigs' folder to the root of the SD card under the directory: home/ubuntu/ [10]
4. Eject SD card and place into PYNQ Z1. Ensure PYNQ jumpers are set to boot from SD.
5. Connect PYNQ through UART.
6. Power the PYNQ.
7. Login using the following format: username:password [ubuntu:temppwd]
8. Run the following commands:
    a. sudo apt update
    b. sudo apt install build-essential
    c. sudo apt-get install libssl-dev
    d. cd hash-sigs
    e. make all
    f. ./demo

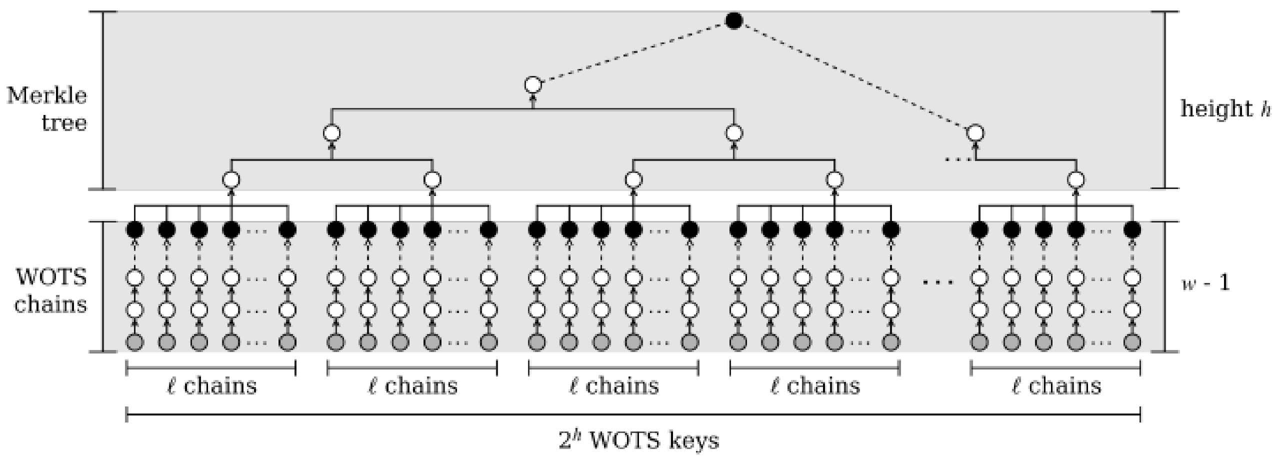
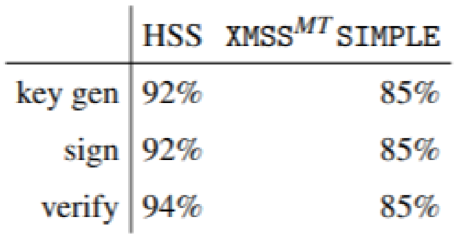


Figure 1. LMS Structure and OTS [6]

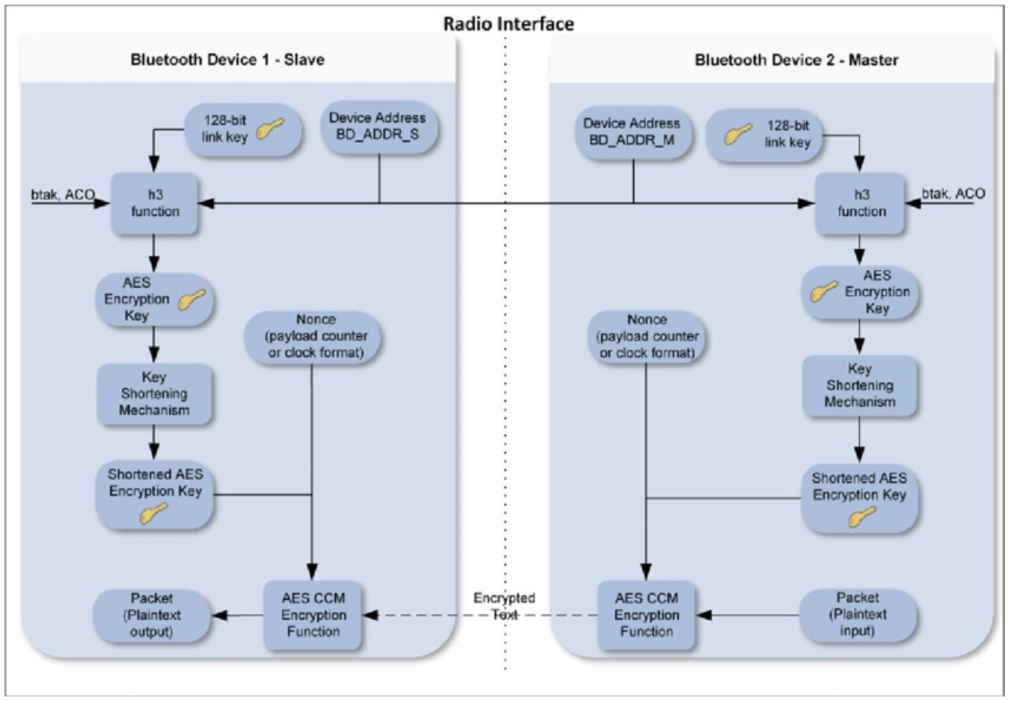


	HSS	XMSS^{MT} SIMPLE
key gen	92%	85%
sign	92%	85%
verify	94%	85%

Figure 2. HSS (LMS vs XMSS Percentage of Time Spent on Hashing [6]

Implementation	Function	Run Time
Desktop (SHA-256)	Key Generation	5 minutes
	Signing	< 1 second
	Verifying	< 1 second
PYNQ-Z1 (SHA-256)	Key Generation	3 hours
	Signing	1 minute
	Verifying	< 1 second
Desktop (SHA-3)	Key Generation	7 minutes
	Signing	3 seconds
	Verifying	< 1 second

Figure 3. Run Time for LMS Implementations



Radio Interface
Bluetooth Device 1 - Slave
Bluetooth Device 2 - Master
128-bit link key
Device Address BD_ADDR_S
Device Address BD_ADDR_M
128-bit link key
btak, ACO
h3 function
h3 function
btak, ACO
AES Encryption Key
AES Encryption Key
Nonce (payload counter or clock format)
Nonce (payload counter or clock format)
Key Shortening Mechanism
Key Shortening Mechanism
Shortened AES Encryption Key
Shortened AES Encryption Key
Packet (Plaintext output)
AES CCM Encryption Function
Encrypted Text
AES CCM Encryption Function
Packet (Plaintext input)

Figure 4. Bluetooth AES-CCM Encryption Procedure [12]

References

[1] Michele Mosca, “Post-Quantum Cryptography”, 6th international Workshop, PQCrypto 2014, Waterloo, ON, Canada, October 1-3.

[2] Tristan Moore, “Quantum Computing and Shor’s Algorithm”, June 7th, 2016: https://sites.math.washington.edu/~morrow/336_16/2016papers/tristan.pdf

[3] Michele Mosca, **“**Cybersecurity in an era with quantum computers: will we be ready?”, https://eprint.iacr.org/2015/1075

[5] Kampanakis, Fluhrer, “LMS vs XMSS: Comparison of two Hash-Based Signature Standards” https://eprint.iacr.org/2017/349.pdf

[6] Campos, Kohlstadt, Reith, Stottinger, “LMS vs XMSS: Comparison of Stateful Hash-Based Signature Schemes on ARM Cortex-M4” https://eprint.iacr.org/2020/470.pdf

[7] McGrew, Curcio, Fluhrer, “Leighton-Micali Hash-Based Signatures” https://tools.ietf.org/html/rfc8554

[8] NIST. Nist issues first call for lightweight cryptography to protect small electronics. [Online]. Available: https://www.nist.gov/news-events/news/2018/04/ nist-issues-first-call-lightweight-cryptography-protect-small-electronics

[9] DSPsandbox, “Ubuntu on ZYNQ and ZYNQMP Devices” https://www.dspsandbox.org/ubuntu-on-zynq-and-zynqmp-devices/

[10] Cisco, “hash-sigs” https://github.com/cisco/hash-sigs

[11] Btstack, https://github.com/bluekitchen/btstack

[12] “NIST Special Publication 800-121 Revision 2, Guide to Bluetooth Security” https://www.researchgate.net/publication/329973291_NIST_Special_Publication_800-121_Revision_2_Guide_to_Bluetooth_Security

[13] Postquantum Crypto Signature, https://github.com/Reconfigurable-Computing-CalPoly-Pomona/postquantum_crypto_signature-