A Long Goodbye to RSA and ECDSA, and Quick Hello to SLH-DSA

Small Keys, But Larger Signatures, and no Haraka hashes

Prof Bill Buchanan OBE FRSE · Follow
Published in ASecuritySite: When Bob Met Alice · 6 min read · 2 days ago

There are two NIST-approved PQC (Post Quantum Cryptography) alternatives for digital signatures. One uses the “darling” lattice method (Dilithium), and the other uses good old hash-based signatures (SPHINCS+). Some, though, worry that the Learning With Error (LWE) approach of lattice methods might be cracked at some time in the future, and so NIST wants alternatives, and one of the most robust from a security point-of-view is **hash-based signatures**. So let’s meet the might SLH-DSA (aka SPHINCS+) method.

Introduction

Well, as if cybersecurity doesn’t have enough acronyms. There’s RIP, OSPF, TCP, IP, SSH, AES, and so many others. Now, there are three really important ones to remember: **ML-KEM** (Module Lattice-Based Key Encapsulation Mechanism), **ML-DSA** (Module Lattice-Based Signature Standard) and **SLH-DSA** (Stateless Hash-based Digital Signature Standard). ML-KEM is defined in the **FIPS 203** standard, ML-DSA is **FIPS 204**, and for SLH-DSA, we have **FIPS 205**.

Many, though, would recognise ML-KEM as **CRYSTALS-Kyber**, ML-DSA as **CRYSTALS Dilithium** and **SLH-DSA** as the **SPHINCS+** method. And, so, on the 13th of August 2024, FIPS 204 was born [here]:

FIPS 204

Federal Information Processing Standards Publication

Module-Lattice-Based Digital Signature Standard

Category: Computer Security **Subcategory: Cryptography**

Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8900

This publication is available free of charge from:
https://doi.org/10.6028/NIST.FIPS.204

Published August 13, 2024

and, on the same day, NIST published FIPS 205 [here]:

FIPS 205

Federal Information Processing Standards Publication

Stateless Hash-Based Digital Signature Standard

Category: Computer Security **Subcategory: Cryptography**

Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8900

This publication is available free of charge from:
https://doi.org/10.6028/NIST.FIPS.205

Published: August 13, 2024

And also FIPS 203 [here]:

FIPS 203

Federal Information Processing Standards Publication

Module-Lattice-Based Key-Encapsulation Mechanism Standard

Category: Computer Security **Subcategory: Cryptography**

Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8900

This publication is available free of charge from:
https://doi.org/10.6028/NIST.FIPS.203

Published August 13, 2024

At present, there is only one replacement for our key exchange methods: ML-KEM, and two replacements for digital signatures (eg RSA, ECDSA and EdDSA): ML-DSA and SLH-DSA. Both ML-KEM and ML-DSA are lattice-based, while SLH-DSA uses a hash-based signature approach, which is stateless.

SLH-DSA

With this, we have a number of private keys, and which can then be hashed within a Merkle Tree to produce a root public key signature. The problem with this method is that we cannot reuse one of our private keys, as we have shown the path it takes to get to the root public key. For this, SPHINCS+ converts the method into a stateless method, and uses trees of hashes.

SPHINCS+ was one of the winners in the NIST standard for PQC (Post Quantum Cryptography), was proposed by Bernstein et al. in 2015 and updated in [2]. SPHINCS+ 256 128-bit has a public key size of 32 bytes, a private key size of 64 bytes, and a signature of 17KB. It has now been standardized by NIST as FIPS 205, and can be used with SHA-256 or SHAKE-256. These include SLH-DSA-SHAKE-128f and SLH-DSA-SHA2–128f.

The following provides an analysis of the PCQ methods for digital signing:

Method	Public key size	Private key size	Signature size	
RSA-2048	256	256	256	
ECC 256-bit	64	32	256	
Crystals Dilithium 2	1,312	2,528	2,420	1
Crystals Dilithium 3	1,952	4,000	3,293	3
Crystals Dilithium 5	2,592	4,864	4,595	5
SLH-DSA-SHA2-128f	32	64	17,088	1
SLH-DSA-SHA2-192f	48	96	35,664	3
SLH-DSA-SHA2-256f	64	128	49,856	5

We can see that the public and private key are small, with only 32 bytes for the public key and 64 bytes for the private key for SLH-DSA-SHA2–128f. This

is much smaller than Dilithium (ML-DSA-512). But the digital signature is larger, with 17,088 bytes against 2,420 for the equivalent Dilithium signature.

For stack memory size on an ARM Cortex-M4 device [1] and measured in bytes:

Method	Key generation	Sign	Verify
Crystals Dilithium 2 (Lattice)	36,424	61,312	40,664
Crystals Dilithium 3	50,752	81,792	55,000
Crystals Dilithium 5	67,136	104,408	71,472
Falcon 512 (Lattice)	1,680	2,484	512
Falcon 1024	1,680	2,452	512
SLH-DSA-SHA2-128f	2,192	2,248	2,544
SLH-DSA-SHA2-192f	3,512	3,640	3,872
SLH-DSA-SHA2-256f	5,600	5,560	5,184

For code size on an ARM Cortex-M4 device [1] and measured in bytes. Note, no Rainbow assessment has been performed in [1], so LUOV (an Oil-and-Vinegar method) has been used to give an indication of performance levels:

Method	Memory (Bytes)
Crystals Dilithium 2 (Lattice)	13,948
Crystals Dilithium 3	13,756
Crystals Dilithium 5	13,852
Falcon 512 (Lattice)	117,271
Falcon 1024	157,207
Sphincs SHA256-128f Simple	4,668

```
Sphincs SHA256-192f Simple              4,676
Sphincs SHA256-256f Simple              5,084
```

For performance, it is much slower than ML-DSA (Dilithium) for key generation, signing and verification [here]:

	Type	Method	Key gen	Factor	Sign	Factor	Verify	Score (Dec)	Overall	Score (Enc)	Score (Dec)	Overall
1	picnic3_L1	Hash/ZKP	21187	1	23348413	100.7	19186836	261.3	10	3	3	16
2	picnic_L3_full	Hash/ZKP	22571	1.1	10877666	46.9	12483100	170	10	5	5	20
3	picnic3_L3	Hash/ZKP	24184	1.1	47229332	203.7	36701854	499.9	10	3	5	18
4	picnic3_L5	Hash/ZKP	25035	1.2	75838669	327.1	58815102	801	10	3	5	18
5	picnic_L1_full	Hash/ZKP	25052	1.2	4854620	20.9	5917140	80.6	10	5	8	23
6	picnic_L5_full	Hash/ZKP	25414	1.2	15670149	67.6	13711927	186.7	10	5	5	20
7	picnic_L1_UR	Hash/ZKP	26617	1.3	9290846	40.1	7755920	105.6	10	5	5	20
8	picnic_L1_FS	Hash/ZKP	26839	1.3	8025732	34.6	6777198	92.3	10	5	8	23
9	picnic_L3_UR	Hash/ZKP	36216	1.7	23156076	99.9	21778246	296.6	10	5	5	20
10	picnic_L5_UR	Hash/ZKP	41418	2	39313637	169.5	34446115	469.1	8	3	5	16
11	picnic_L5_FS	Hash/ZKP	44942	2.1	37066573	159.9	28836125	392.7	8	3	5	16
12	Dilithium2-AES	Lattice	67697	3.2	231878	1	73424	1	8	10	10	28
13	picnic_L3_FS	Hash/ZKP	72919	3.4	20939121	90.3	18323702	249.6	8	5	3	16
14	Dilithium3-AES	Lattice	104515	4.9	374297	1.6	113536	1.5	8	10	10	28
15	Dilithium2	Lattice	116511	5.5	342726	1.5	112506	1.5	8	10	10	28
16	Dilithium5-AES	Lattice	164148	7.7	416647	1.8	166482	2.3	8	10	8	26
17	Dilithium3	Lattice	191331	9	534254	2.3	180350	2.5	8	8	8	24
18	Dilithium5	Lattice	307765	14.5	610807	2.6	417971	5.7	5	8	8	21
19	SPHINCS+-Haraka-128f-s	Hash	1114859	52.6	27587176	119	1521957	20.7	5	3	5	13
20	SPHINCS+-Haraka-128f-r	Hash	1296477	61.2	31847101	137.3	2308807	31.4	5	3	5	13
21	SPHINCS+-Haraka-192f-s	Hash	1733557	81.8	47717755	205.8	2509526	34.2	5	3	5	13
22	SPHINCS+-Haraka-192f-r	Hash	2034468	96	61820381	266.6	3691384	50.3	5	3	5	13
23	SPHINCS+-SHA256-128f-s	Hash	3088404	145.8	72191077	311.3	8962488	122.1	3	3	3	9
24	SPHINCS+-SHA256-192f-s	Hash	3920103	185	119085653	513.6	12269960	167.1	3	3	3	9
25	SPHINCS+-SHAKE256-128f-s	Hash	3993469	188.5	118778065	512.2	11837565	161.2	3	3	3	9
26	SPHINCS+-SHA256-128f-r	Hash	4576725	216	115180200	496.7	16236483	221.1	3	3	3	9
27	SPHINCS+-Haraka-256f-s	Hash	4656137	219.8	89990034	388.1	2534462	34.5	3	3	5	11
28	SPHINCS+-SHAKE256-192f-s	Hash	5766316	272.2	166899175	719.8	16662894	226.9	3	3	3	9
29	SPHINCS+-SHA256-192f-r	Hash	6343609	299.4	187556226	808.9	22966293	312.8	3	3	3	9
30	SPHINCS+-Haraka-256f-r	Hash	6398014	302	119210092	514.1	3793534	51.7	3	3	5	11
31	SPHINCS+-SHAKE256-128f-r	Hash	6831602	322.4	176444474	760.9	21769720	296.5	3	3	3	9
32	SPHINCS+-SHAKE256-192f-r	Hash	9735939	459.5	284106213	1225.2	30835025	420	3	0	3	6
33	SPHINCS+-SHA256-256f-s	Hash	10771651	508.4	220637782	951.5	12168947	165.7	3	3	3	9
34	SPHINCS+-SHAKE256-256f-s	Hash	15684219	740.3	318508169	1373.6	16422940	223.7	3	0	3	6
35	Falcon-512	Lattice	24656358	1163.7	1085984	4.7	183949	2.5	0	8	8	16
36	SPHINCS+-SHAKE256-256f-r	Hash	27044805	1276.5	564867404	2436.1	32709637	445.5	0	0	3	3
37	SPHINCS+-SHA256-256f-r	Hash	28940978	1366	571268028	2463.7	29935214	407.7	0	0	3	3
38	SPHINCS+-Haraka-256s-s	Hash	69028778	3258.1	998765490	4307.3	1330020	18.1	0	0	5	5
39	Falcon-1024	Lattice	74741342	3527.7	2204927	9.5	359553	4.9	0	8	8	16

Overall, the Haraka hashing method is the fastest but has not been approved as a FIPS standard. The two main standards use the NIST-approved hashes of

SHA2 and SHAKE. Overall, there are two main methods for signing: a “pure” version (slh_sign) and a “pre-hash” version (hash_slh_sign). With the pre-hash version, we hash the message before it is sent for signature. This means that there is less data to deal with for the signature and can thus reduce the computational load on the signing application.

Code

We can install SHLDSA with [here]:

```
pip install slh-dsa
```

The code to implement this is [here]:

```
from slhdsa import KeyPair, sha2_128s,sha2_128f, sha2_192s, sha2_192f, sha2_256s
import binascii
import sys

para=shake_256f
message='Hello'
method='sha2_128s';

if (len(sys.argv)>1):
 message=str(sys.argv[1])
if (len(sys.argv)>2):
 method=str(sys.argv[2])

if (method=='sha2_128s'): para=sha2_128s
elif (method=='sha2_128f'): para=sha2_128f
elif (method=='sha2_192s'): para=sha2_192s
elif (method=='sha2_192f'): para=sha2_192f
elif (method=='sha2_256s'): para=sha2_256s
elif (method=='sha2_256f'): para=sha2_256f
elif (method=='shake_128s'): para=shake_128s
elif (method=='shake_128f'): para=shake_128f
elif (method=='shake_192s'): para=shake_192s
```

```
elif (method=='shake_192f'): para=shake_192f
elif (method=='shake_256s'): para=shake_256s
elif (method=='shake_256f'): para=shake_256f

kp = KeyPair.gen(para)
sig = kp.sign(message.encode())
rtn=kp.verify(message.encode(), sig)
print(f"Message: {message}")
print(f"Method: {method}\n")

rtn=kp.verify(b"In correct message",sig)
print("\nBad signature valid: ",rtn)
print("Signature (first 32 bytes): ",binascii.hexlify(sig[:32]))
print(f"Signature length: {len(sig)} bytes")
print("Signature valid: ",rtn)
print("Privte key size: ",len(kp.sec.key[0]*2))
print("Public key size: ",len(kp.pub.key[0]))
```

A sample run with SHA2–128f gives [here]:

```
Message: Post Quantum Crypto
Method: sha2_128f

Private key size:  32
Public key size:  16
Signature (first 32 bytes):  b'b1ecc5b25c8ee52ba0d42de7684e56c544e2ec64ae42f93be
Signature length: 17088 bytes
Signature valid:  True
Bad signature valid:  False
```

A sample run with SHA2–256f gives [here]:

```
Message: Post Quantum Crypto
Method: sha2_256f
```

```
Private key size:  64
Public key size:  32

Signature (first 32 bytes):  b'1d18751a62751584e4f93e52f31f5f2c7033ae77936a5f405
Signature length: 49856 bytes
Signature valid:  True

Bad signature valid:  False
```

Conclusions

SLH-DSA is a solid signature and has small keys, but the signature is larger than RSA, ECDSA and ML-DSA. It also has rock-solid security proofs, which is not quite the case for RSA, ECDSA and ML-DSA. So, as an alternative to ML-DSA, it's a great method. A little slow in places but highly secure.

References

[1] Kannwischer, M. J., Rijneveld, J., Schwabe, P., & Stoffelen, K. (2019). pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4 [here].

[2] Bernstein, D. J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., & Schwabe, P. (2019, November). The SPHINCS+ signature framework. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (pp. 2129–2146)

Cryptography Cybersecurity

Written by Prof Bill Buchanan OBE FRSE

Follow

32K Followers · Editor for ASecuritySite: When Bob Met Alice

Professor of Cryptography. Serial innovator. Believer in fairness, justice & freedom. Based in Edinburgh. Old World Breaker. New World Creator. Building trust.

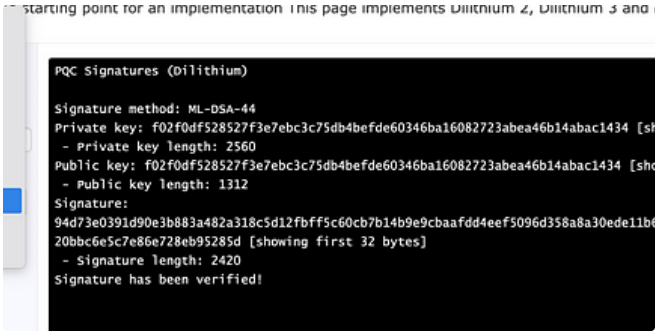
More from Prof Bill Buchanan OBE FRSE and ASecuritySite: When Bob Met Alice

Prof Bill Buchanan OBE FRSE

The Real Satoshi?

I found an interesting Twitter thread [here]:

Oct 5 257 6

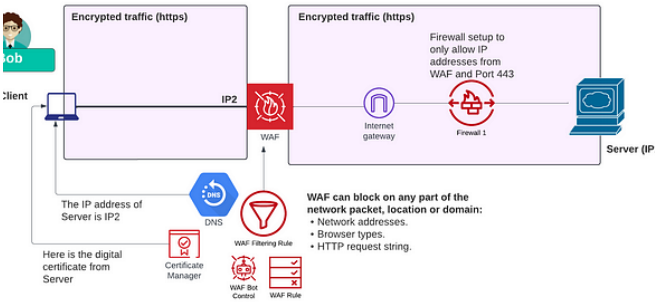


Prof Bill Buchanan O… in ASecuritySite: When Bob…

A Long Goodbye to RSA, ECDSA and EdDSA, and Big Hello to ML-…

Just this week, Cloudflare released the ML-DSA methods within their CIRCL library, and…

Oct 15 104 1

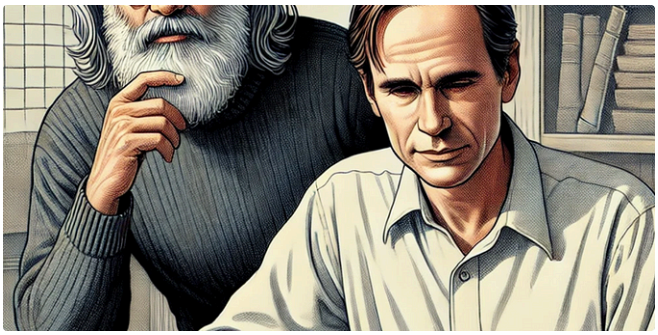


Prof Bill Buchanan O... in ASecuritySite: When Bob...

WAFs Show That End-to-End TLS Encryption Tunnels Can Be Broke...

But it needs the trust of a proxy in-between

Oct 6 51 2



Prof Bill Buchanan OBE FRSE

After 48 Years, It's A Long Goodbye to the Diffie-Hellman Method

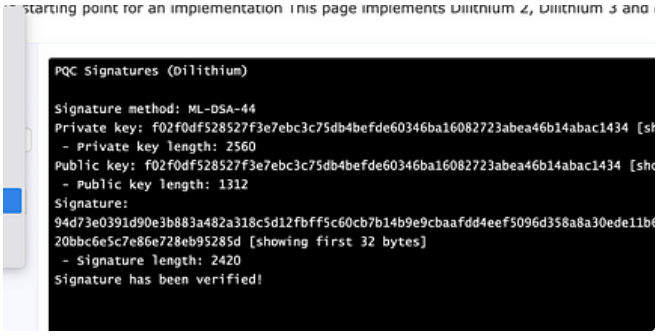
This week, in my lecture, I will outline one of the most amazing methods ever created in...

Oct 16 249 4

See all from Prof Bill Buchanan OBE FRSE

See all from ASecuritySite: When Bob Met Alice

Recommended from Medium



Prof Bill Buchanan O... in ASecuritySite: When Bob...

A Long Goodbye to RSA, ECDSA and EdDSA, and Big Hello to ML-...

Just this week, Cloudflare released the ML-DSA methods within their CIRCL library, and...

Oct 15 104 1

Armando Rodrigues in Radio Hackers

LoRaWAN—Everything You Need to Know About The Global IoT...

What you should know before starting any LoRaWAN project.

5d ago 178 2

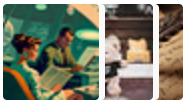
Lists

Tech & Tools

21 stories · 336 saves

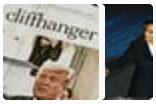
General Coding Knowledge

20 stories · 1699 saves



Medium's Huge List of Publications Accepting...

378 stories · 3847 saves



Staff Picks

756 stories · 1420 saves

Harendra

How I Am Using a Lifetime 100% Free Server

Get a server with 24 GB RAM + 4 CPU + 200 GB Storage + Always Free

Oct 26 1.8K 23

Henrique Centieiro & Bee ... in Limitless Invest...

$35 Trillion Economic Collapse Begins—The Simplest Way to...

99% Ignore This Strategy Behind My 8-Year Path to Retirement

6d ago 2.2K 55

Abdur Rahman in Stackademic

Python is No More The King of Data Science

5 Reasons Why Python is Losing Its Crown

Oct 23 2.7K 19

Salvatore Raieli in Towards Data Science

The Savant Syndrome: Is Pattern Recognition Equivalent to…

Exploring the limits of artificial intelligence: why mastering patterns may not equal…

5d ago 1.7K 31

See more recommendations