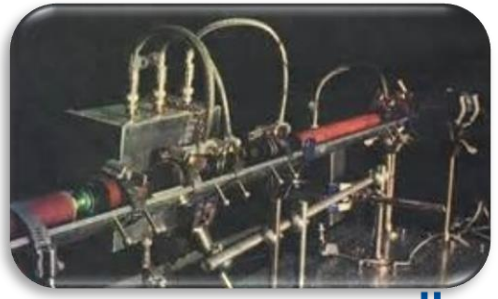
Stay quantum safe
future-proofing encrypted secrets

Christian Paquin
@chpaquin
Principal Program Manager
Microsoft Research

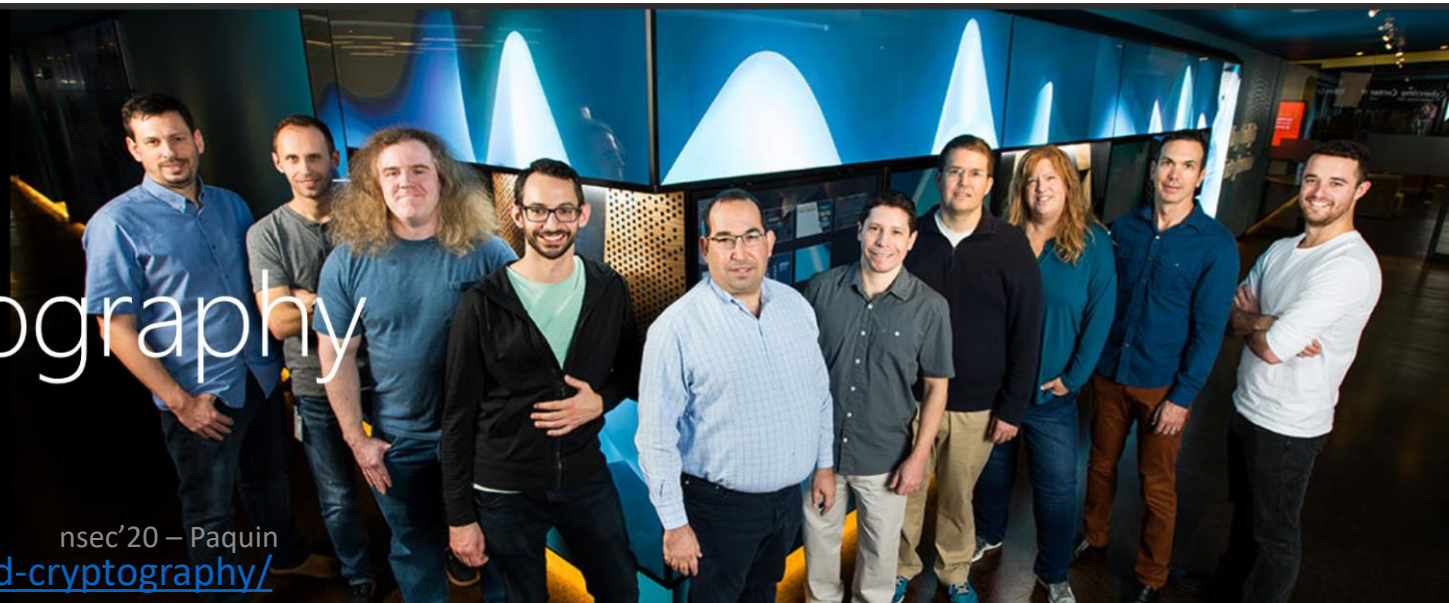
nsec 2020

About



Université de Montréal

- Studied quantum cryptography 25 years ago at University of Montreal
- Worked in the industry as a cryptographic engineer
- Joined Microsoft more than a decade ago
  - Now with *MSR Security & Crypto* team, working with cutting-edge crypto tech

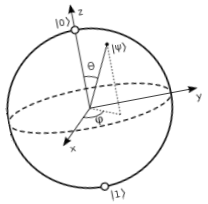


Post-quantum cryptography

https://www.microsoft.com/en-us/research/group/security-and-cryptography/

The Quantum Revolution

- Quantum computers use the properties of *quantum mechanics* to implement algorithms not possible on classical computers



$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

- A lot of R&D around the globe
  - My colleagues are building the full stack: from the chip to the SDK https://www.microsoft.com/quantum/

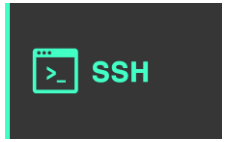
QDK

Q#

The Quantum Menace

- Quantum computers are bad news for cryptography!
  - Shor (1994) solves the factoring (breaks RSA) and discreet log (breaks DSA, Diffie Hellman, and elliptic curve variants) problems in polynomial time

- Breaks ~~most~~ all the asymmetric crypto in use today



- Could be built within 10-15 years

- We need new *quantum-safe* cryptography

Need to migrate to quantum-safe crypto soon

- Capture now, decrypt later
- Updating standards takes a long time
  - TLS, SSH, IKE, PKI, S/MIME, …
- Unknown impact on code base
  - Longer key/message/sig sizes
  - Slower running times
  - Code agility

Do your apps protect data that needs to be kept secret for more than 10 years?

NIST competition

Nov 2017
- Round 1
- 69 submitted

Apr 2018
- 1st NIST PQC workshop

Jan 2019
- Round 2

Aug 2019
- 2nd NIST PQC workshop

2020/2021
- Round 3

2022-2024
- Draft standards

Encryption / Key Encapsulation

BIG QUAKE	Guess Again	LOTUS	RLCE-KEM
BIKE	HILA5	McNie	Round2
CFPKM	HQC	Mersenne756839	RQC
Classic McEliece	KCL	NewHope	SABER
Compact LWE	KINDI	NTRUEncrypt	SIKE
CRYSTALS-KYBER	LAC	NTRU-HRSS-KEM	Three Bears
DAGS	LAKE	NTRU Prime	Titanium
Ding KEX	LEDAkem	NTS-KEM	
DME	LEDApkc	Odd Manhattan	
EMBLEM	Lepton	Ouroboros-R	
R.EMBLEM	LIMA	PQ RSA-Enc	
FrodoKEM	Lizard	QC-MDPC KEM	
Giophantus	LOCKER	Ramstake	

Signature

CRYSTALS-DILITHIUM	pqNTRUSign
DRS	Picnic
DualModeMS	PQ RSA-Sig
FALCON	pqsigRM
GeMSS	qTESLA
Gravity-SPHINCS	RaCoSS
Gui	Rainbow
HiMQ-3	SPHINCS+
LUOV	WalnutDSA
MQDSS	

NIST competition

Nov 2017
- Round 1
- 69 submitted

Apr 2018
- 1st NIST PQC workshop

Jan 2019
- Round 2
- 26 remaining

Aug 2019
- 2nd NIST PQC workshop

2020/2021
- Round 3

2022-2024
- Draft standards

Encryption / Key Encapsulation

BIKE
HQC
Classic McEliece
CRYSTALS-KYBER
LAC
LEDAcrypt
FrodoKEM
NewHope
NTRU
NTRU Prime
NTS-KEM
ROLLO
Round5
RQC
SABER
SIKE
Three Bears

Signature

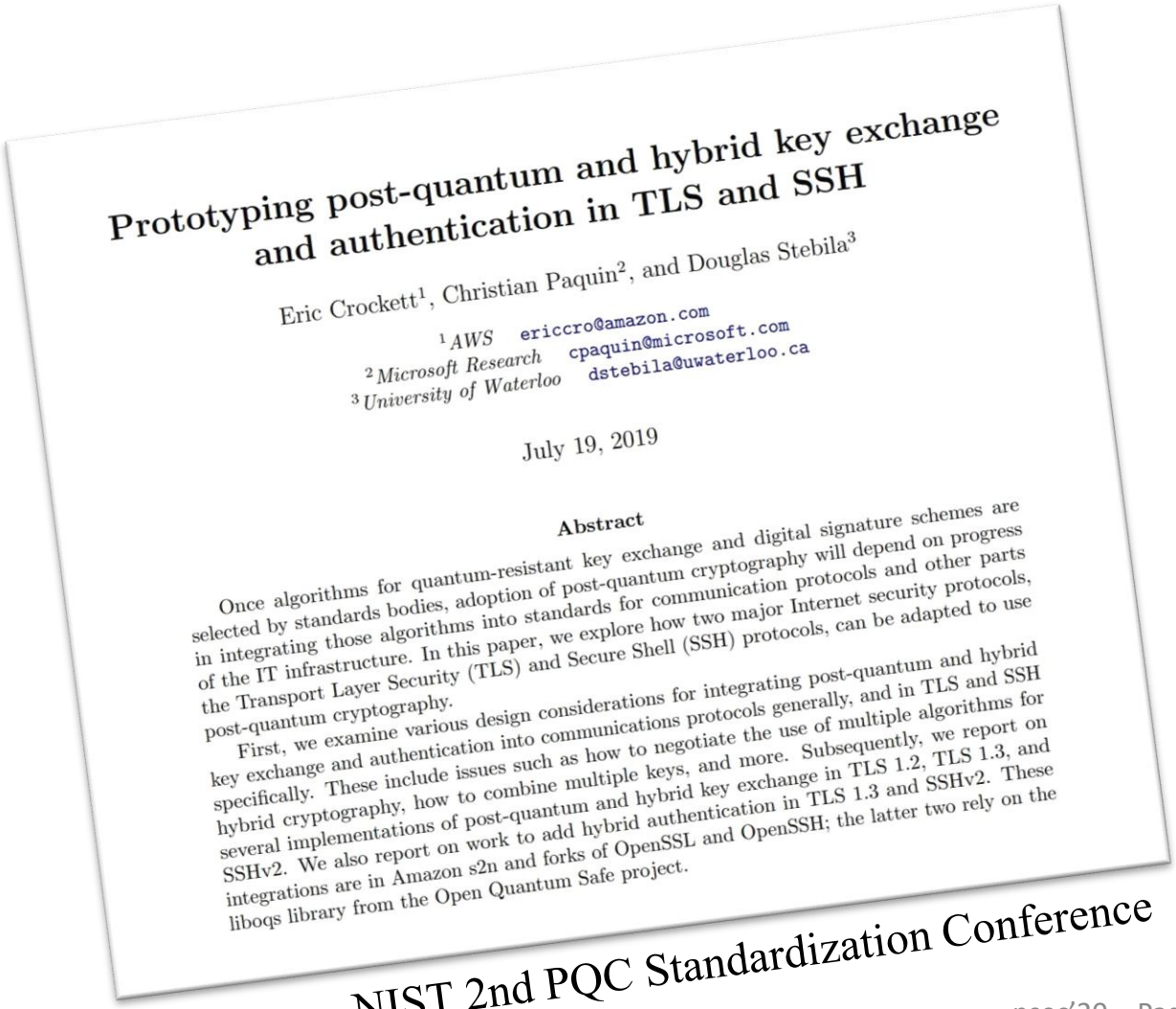
CRYSTALS-DILITHIUM
Picnic
FALCON
GeMSS
qTESLA
Rainbow
SPHINCS+
LUOV
MQDSS

OPEN QUANTUM SAFE

- C library created to simplify integration of PQC into applications
- Contributions from

- Supports many NIST round 2 KEM and signature schemes
- Integrations into boringssl, OpenSSL, OpenSSH, OpenVPN
- C++, C#, Go, Java, and Python wrappers
- https://openquantumsafe.org/

Prototyping PQC



Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH

Eric Crockett[1], Christian Paquin[2], and Douglas Stebila[3]

[1] *AWS* ericcro@amazon.com
[2] *Microsoft Research* cpaquin@microsoft.com
[3] *University of Waterloo* dstebila@uwaterloo.ca

July 19, 2019

Abstract

Once algorithms for quantum-resistant key exchange and digital signature schemes are selected by standards bodies, adoption of post-quantum cryptography will depend on progress in integrating those algorithms into standards for communication protocols and other parts of the IT infrastructure. In this paper, we explore how two major Internet security protocols, the Transport Layer Security (TLS) and Secure Shell (SSH) protocols, can be adapted to use post-quantum cryptography.

First, we examine various design considerations for integrating post-quantum and hybrid key exchange and authentication into communications protocols generally, and in TLS and SSH specifically. These include issues such as how to negotiate the use of multiple algorithms for hybrid cryptography, how to combine multiple keys, and more. Subsequently, we report on several implementations of post-quantum and hybrid key exchange in TLS 1.2, TLS 1.3, and SSHv2. We also report on work to add hybrid authentication in TLS 1.3 and SSHv2. These integrations are in Amazon s2n and forks of OpenSSL and OpenSSH; the latter two rely on the liboqs library from the Open Quantum Safe project.

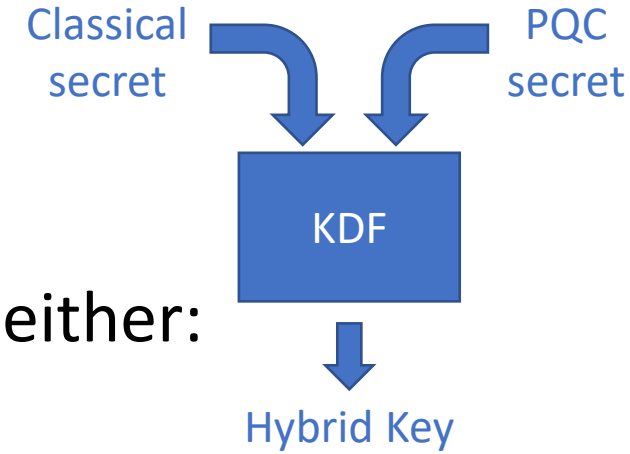
NIST 2nd PQC Standardization Conference

- Analyze various options to integrate PQC into TLS and SSH
- Focus on hybrid scenarios
- Lessons learned from OpenSSL, OpenSSH, and s2n integrations

https://eprint.iacr.org/2019/858

Hybrid scenarios

- Early migration should use a hybrid of classical/PQ schemes
  - Security of today + safety net against quantum computer
  - Secure if one of the two is secure



- TLS and SSH negotiate one algorithm; need to define either:
  - new combo schemes
  - a new hybrid approach

- Consider backward compatibility, performance, latency, data flow

TLS case study

```
       Client                                                  Server

Key  ^ ClientHello
Exch | + key_share*
     | + signature_algorithms*
     | + psk_key_exchange_modes*
     v + pre_shared_key*         -------->
                                                       ServerHello  ^ Key
                                                      + key_share*  | Exch
                                                 + pre_shared_key*  v
                                             {EncryptedExtensions}  ^  Server
                                             {CertificateRequest*}  v  Params
                                                    {Certificate*}  ^
                                              {CertificateVerify*}  | Auth
                                                        {Finished}  v
                                 <--------     [Application Data*]
     ^ {Certificate*}
Auth | {CertificateVerify*}
     v {Finished}                -------->
       [Application Data]        <------->     [Application Data]
```

- Added PQ/hybrid KEX & auth
- TLS 1.2 (OpenSSL 1.0.2)
- TLS 1.3 (OpenSSL 1.1.1)
  - PQ algs masquerade as EC curves
  - Concatenation strategy more secure than 1.2 (KDF hashes transcripts)
  - Spec pub key and sig limit: 2^{16}-1 bytes, cert limit: 2^{24}-1 bytes
    - OpenSSL limit is smaller
  - Tested with OpenSSL/boringssl tools, apache, nginx

- https://github.com/open-quantum-safe/openssl

cpaquin@CPAQUINSB: /mnt/c/c

```
cpaquin@CPAQUINSB:/mnt/c/demo/openssl$ apps/openssl s_server -cert p256_qteslapi.crt -key p256_qteslapi.key -www -tls1_3
Using default temp DH parameters
ACCEPT
```

cpaquin@CPAQUINSB: /mnt/c/c

```
cpaquin@CPAQUINSB:/mnt/c/demo/openssl$ apps/openssl s_client -curves p256_frodo640aes -CAfile p256_qteslapi.crt -connect localhost:4433
---
No client certificate CA names sent
Peer signature type: ECDSA p256 - qTesla-I-p
Server Temp Key: p256_frodo640aes hybrid
---
SSL handshake has read 30661 bytes and written 10058 bytes
Verification: OK
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 119592 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 0 (ok)
---
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol  : TLSv1.3
    Cipher    : TLS_AES_256_GCM_SHA384
    Session-ID: 98798307A8B4100073801F2FEEE748C004E4E5E70D5A721F4BB71F364AD67478
    Session-ID-ctx:
    Resumption PSK: 7986E9BBF15CB1F3CD50B138DD2A6D1B6E1BCCA1D63A80D6BFED24BCF40D64B0FB72222231EC4E7AB4D2DF1A07AA62E2
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 7200 (seconds)
    TLS session ticket:
    0000 - 3b 89 b5 e2 fc 29 9d 50-53 2c 7c 74 87 17 38 f1   ;....).PS,|t..8.
    0010 - 46 21 f7 32 9a 5b 43 26-72 3b 6f 0f af 80 eb ff   F!.2.[C&r;o.....
    0020 - 77 98 7c 93 73 85 c8 b1-8a 5f de 6e 1e 23 6a 8f   w.|.s...._.n.#j.
    0030 - a5 c9 d5 ef 71 df 79 29-cb 2c a1 a3 f4 7c 18 f2   ....q.y).,...|..
    0040 - cc 6e cd a8 ae 23 34 d9-0f 8e 92 2b df 80 4a f9   .n...#4....+..J.
    0050 - 85 33 ae 17 04 9a 19 3a-7c 45 12 16 5e 54 06 b0   .3.....:|E..^T..
    0060 - f5 72 07 fa ea 19 a6 8d-c4 d3 b0 60 37 1f c2 eb   .r.........`7...
    0070 - 93 2b 06 ad 6f 14 44 ae-52 db dd 43 88 41 14 ae   .+..o.D.R..C.A..
    0080 - e3 5d e0 26 2a 4a fd 69-8a b0 8e 6e 96 59 f2 6a   .].&*J.i...n.Y.j
    0090 - 14 ce 7f ac bf 90 c3 1d-c1 d6 1e b2 8d 27 99 f1   .............'..
```

https://github.com/open-quantum-safe/openssl

SSH case study

- Added PQ/hybrid KEX & auth to OpenSSH
- Define new algorithms
  - e.g.: ecdh-nistp384-sike-503-sha384@openquantumsafe.org
- Supports both client and server public key authentication
- Spec message size limit: 2^{32} bytes
  - large enough for all round 2 candidates, but OpenSSH limit is smaller (2^{18})
- https://github.com/open-quantum-safe/openssh-portable

cpaquin@CPAQUINSB: ~

```
cpaquin@CPAQUINSB:/mnt/c/demo/openssh-portable$ /mnt/c/demo/openssh-portable/ssh -l cpaquin -o 'KexAlgorithms=ecdh-nistp384-sike-p434-sha384@openquantumsafe.org' -o HostKeyAlgorithms=ssh-p256-picni
cl1fs -o PubkeyAcceptedKeyTypes=ssh-p256-picnicl1fs -o StrictHostKeyChecking=no -i ~/ssh_client/id_p256-picnicl1fs -p 2222 localhost -v
OpenSSH_7.9p1, OpenSSL 1.1.1  11 Sep 2018
debug1: Connecting to localhost [127.0.0.1] port 2222.
debug1: Connection established.
debug1: identity file /home/cpaquin/ssh_client/id_p256-picnicl1fs type 33
debug1: identity file /home/cpaquin/ssh_client/id_p256-picnicl1fs-cert type -1
debug1: Local version string SSH-2.0-OpenSSH_7.9
debug1: Remote protocol version 2.0, remote software version OpenSSH_7.9
debug1: match: OpenSSH_7.9 pat OpenSSH* compat 0x04000000
debug1: Authenticating to localhost:2222 as 'cpaquin'
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: algorithm: ecdh-nistp384-sike-p434-sha384@openquantumsafe.org
debug1: kex: host key algorithm: ssh-p256-picnicl1fs
debug1: kex: server->client cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
debug1: kex: client->server cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
debug1: expecting 31 msg
debug1: Server host key: ssh-p256-picnicl1fs SHA256:75K2JpVdjU4jwhloSX5lkItakueN2hmLoV3bmtW7+tw
debug1: Host '[localhost]:2222' is known and matches the P256_PICNICL1FS host key.
debug1: Found key in /home/cpaquin/.ssh/known_hosts:78
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: rekey after 134217728 blocks
debug1: Will attempt key: /home/cpaquin/ssh_client/id_p256-picnicl1fs P256_PICNICL1FS SHA256:dcWtg4cO+vDJhZ4H4apPTO/rJeE1wA9ceq/pnf1/hG0 explicit
debug1: SSH2_MSG_EXT_INFO received
debug1: kex_input_ext_info: server-sig-algs=<ssh-ed25519,ssh-oqsdefault,ssh-dilithium2,ssh-dilithium4,ssh-mqdss3148,ssh-picnicl1fs,ssh-picnicl1ur,ssh-picnicl3fs,ssh-picnicl3ur,ssh-picnicl5fs,ssh-pi
cnicl5ur,ssh-picnic2l1fs,ssh-picnic2l3fs,ssh-qteslapi,ssh-qteslapiii,ssh-sphincsharaka128frobust,ssh-rsa3072-oqsdefault,ssh-p256-oqsdefault,ssh-rsa3072-dilithium2,ssh-p256-dilithium2,ssh-p384-dilit
hium4,ssh-rsa3072-mqdss3148,ssh-p256-mqdss3148,ssh-rsa3072-picnicl1fs,ssh-p256-picnicl1fs,ssh-rsa3072-picnicl1ur,ssh-p256-picnicl1ur,ssh-p384-picnicl3fs,ssh-p384-picnicl3ur,ssh-p521-picnicl5fs,ssh-
p521-picnicl5ur,ssh-rsa3072-picnic2l1fs,ssh-p256-picnic2l1fs,ssh-p384-picnic2l3fs,ssh-rsa3072-qteslapi,ssh-p256-qteslapi,ssh-p384-qteslapiii,ssh-rsa3072-sphincsharaka128frobust,ssh-p256-sphincshara
ka128frobust,ssh-rsa,rsa-sha2-256,rsa-sha2-512,ssh-dss,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521>
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password,keyboard-interactive
debug1: Next authentication method: publickey
debug1: Offering public key: /home/cpaquin/ssh_client/id_p256-picnicl1fs P256_PICNICL1FS SHA256:dcWtg4cO+vDJhZ4H4apPTO/rJeE1wA9ceq/pnf1/hG0 explicit
debug1: Server accepts key: /home/cpaquin/ssh_client/id_p256-picnicl1fs P256_PICNICL1FS SHA256:dcWtg4cO+vDJhZ4H4apPTO/rJeE1wA9ceq/pnf1/hG0 explicit
debug1: Authentication succeeded (publickey).
Authenticated to localhost ([127.0.0.1]:2222).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: pledge: network
debug1: client_input_global_request: rtype hostkeys-00@openssh.com want_reply 0
debug1: Remote: /home/cpaquin/ssh_server/authorized_keys:1: key options: agent-forwarding port-forwarding pty user-rc x11-forwarding
debug1: Remote: /home/cpaquin/ssh_server/authorized_keys:1: key options: agent-forwarding port-forwarding pty user-rc x11-forwarding
Last login: Mon May 11 14:39:08 2020 from 127.0.0.1
cpaquin@CPAQUINSB:~$
```

https://github.com/open-quantum-safe/openssh-portable

Key Encapsulation Mechanisms

KEM scheme	OpenSSL 1.0.2 TLS 1.2	OpenSSL 1.1.1 TLS 1.3	OpenSSH 7.9 SSH2
BIKE 1/2/3 L1/3/5 (round 1)	✓	✓	✓
Frodo KEM 640/976 AES/SHAKE	✓	✓	✓
Frodo KEM 1344 AES/SHAKE	☑	☑	✓
Kyber 512/768/1024	✓	✓	✓
LEDAcrypt KEM LT 12/32/52	✓	✓	✓
NewHope 512/1024 CCA	✓	✓	✓
NTRU HPS (2048-509/677)/(4096-821)	✓	✓	✓
NTRU HRSS 701	✓	✓	✓
NTS KEM (12,64)	✗	✗	✗
LightSaber/Saber/FireSaber KEM	✓	✓	✓
SIKE p434/p503/p610/p751	✓	✓	✓

KEM integrations for both PQ and hybrid (with ECDHE)

Legend:

✓ Success

☑ Works with code mods

✗ Did not work

Signatures

KEM scheme	OpenSSL 1.1.1 TLS 1.3	OpenSSH 7.9 SSH2
Dilithium 2/3/4	✓	✓
MQDSS 31 48/64	☑	✓
Picnic L1 FS/UR	☑	✓
Picnic L3/L5 FS/UR	✗	✓
Picnic2 L1 FS	✓	✓
Picnic2 L3/L5 FS	☑	✓
qTesla I/III-size/III-speed (round 1)	✓	✓
Rainbow Ia Classic	☑	☑
Rainbow Ia Cyclic/Compressed	✓	✓
Rainbow IIIc/Vc Classic/Cyclic/Compressed	☑	✗
SPHINCS+ * 128s *	✓	✓
SPHINCS+ * 128f/192f/192s/256f/256s *	☑	✓

Signature integrations for both PQ and hybrid (with ECDSA)

Legend:

✓ Success

☑ Works with code mods

✗ Did not work

PQC TLS benchmarking

Two experiments to measure PQC impact on TLS performance

1) Simulated connections with various latency and packet-loss

2) Real-world retrievals of various page sizes from various geo-located VMs

https://eprint.iacr.org/2019/1447

PQCrypto 2020

Benchmarking Post-Quantum Cryptography in TLS

Christian Paquin
Microsoft Research
cpaquin@microsoft.com

Douglas Stebila and Goutam Tamvada
University of Waterloo
dstebila@uwaterloo.ca, gtamvada@edu.uwaterloo.ca

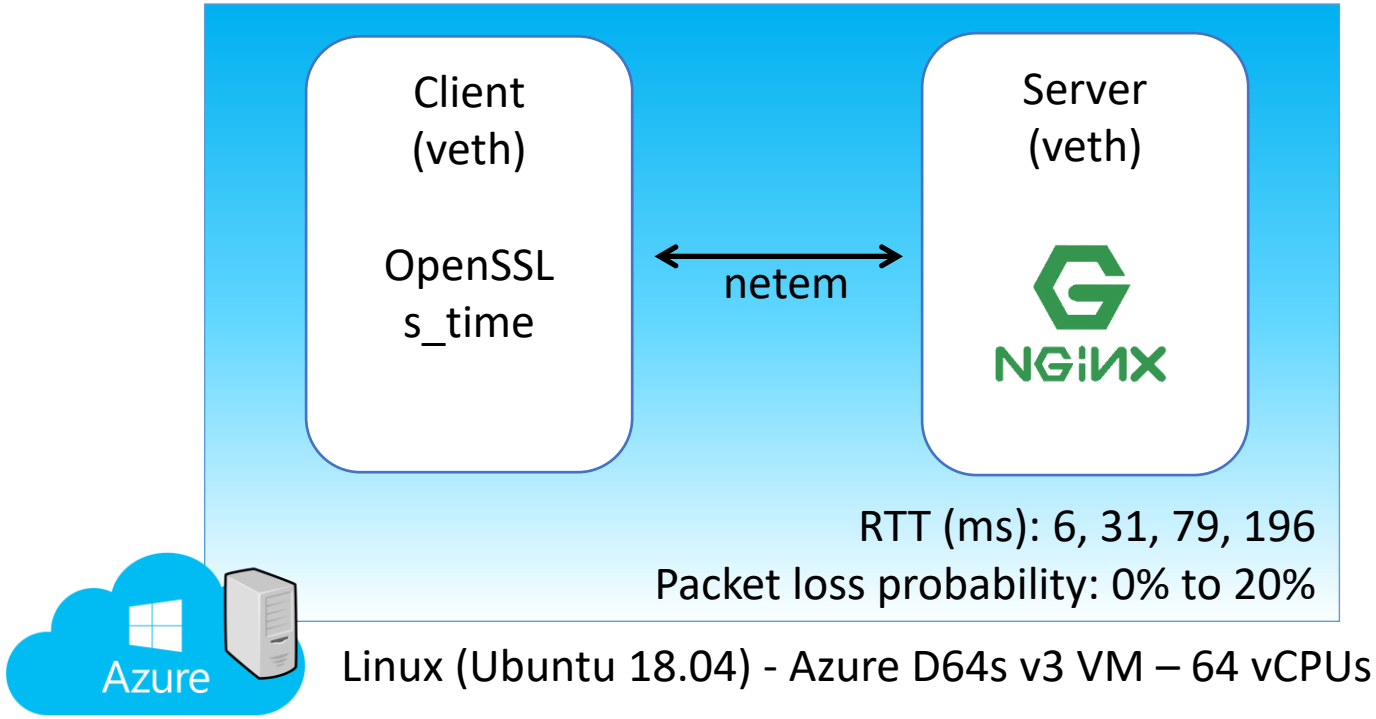
February 6, 2020

Abstract

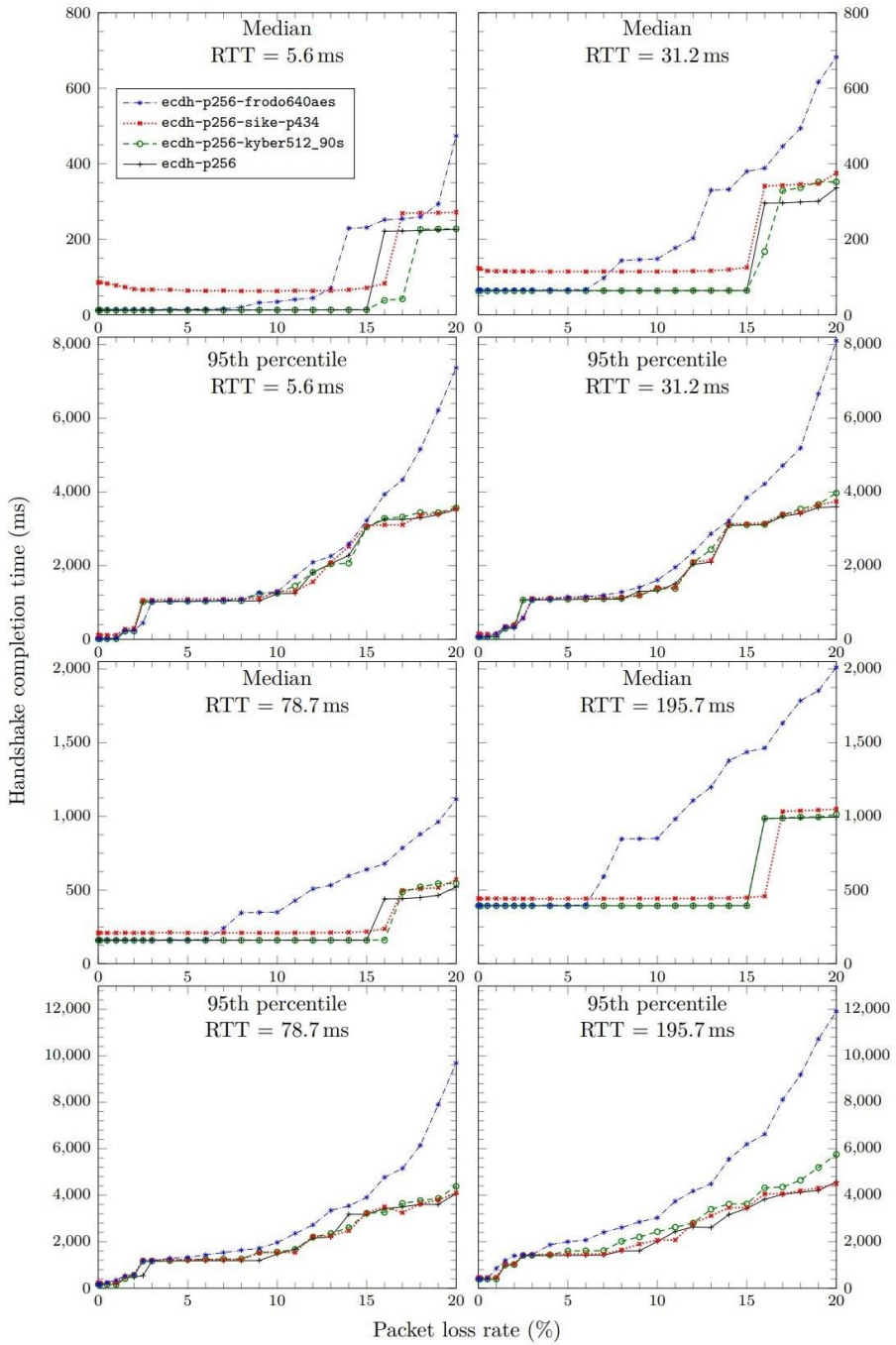
Post-quantum cryptographic primitives have a range of trade-offs compared to traditional public key algorithms, either having slower computation or larger public keys and ciphertexts/signatures, or both. While the performance of these algorithms in isolation is easy to measure and has been a focus of optimization techniques, performance in realistic network conditions has been less studied. Google and Cloudflare have reported results from running experiments with post-quantum key exchange algorithms in the Transport Layer Security (TLS) protocol with real users' network traffic. Such experiments are highly realistic, but cannot be replicated without access to Internet-scale infrastructure, and do not allow for isolating the effect of individual network characteristics.

In this work, we develop and make use of a framework for running such experiments in TLS cheaply by *emulating* network conditions using the networking features of the Linux kernel. Our testbed allows us to independently control variables such as link latency and packet loss rate, and then examine the performance impact of various post-quantum-primitives on TLS connection establishment, specifically hybrid elliptic curve/post-quantum key exchange and post-quantum digital signatures, based on implementations from the Open Quantum Safe project. Among our key results, we observe that packet loss rates above 3–5% start to have a significant impact on post-quantum algorithms that fragment across many packets, such as those based on unstructured lattices. The results from this emulation framework are also complemented by results on the latency of loading entire web pages over TLS in real network conditions, which show that network latency hides most of the impact from algorithms with slower computations (such as supersingular isogenies).

Benchmarking results #1

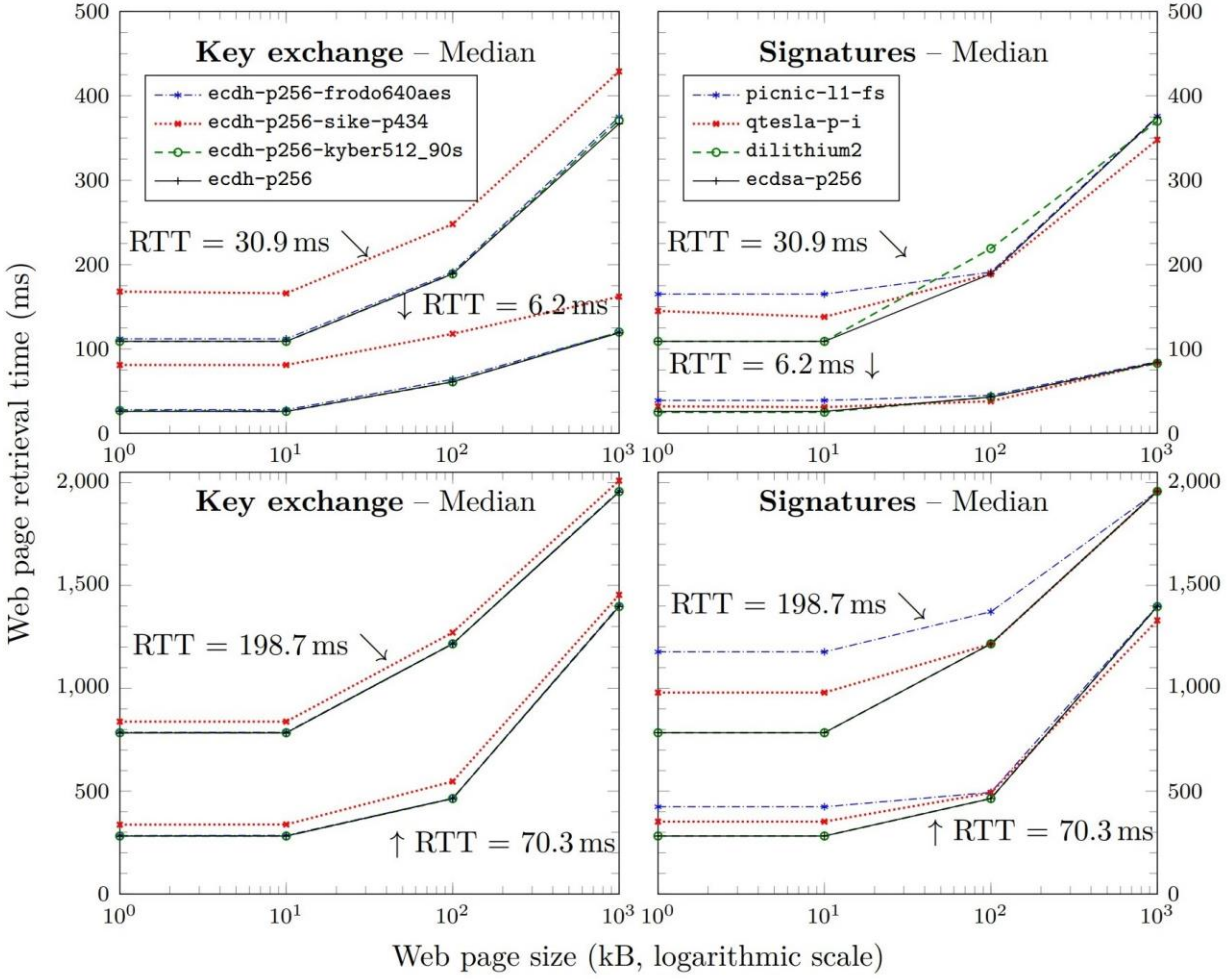


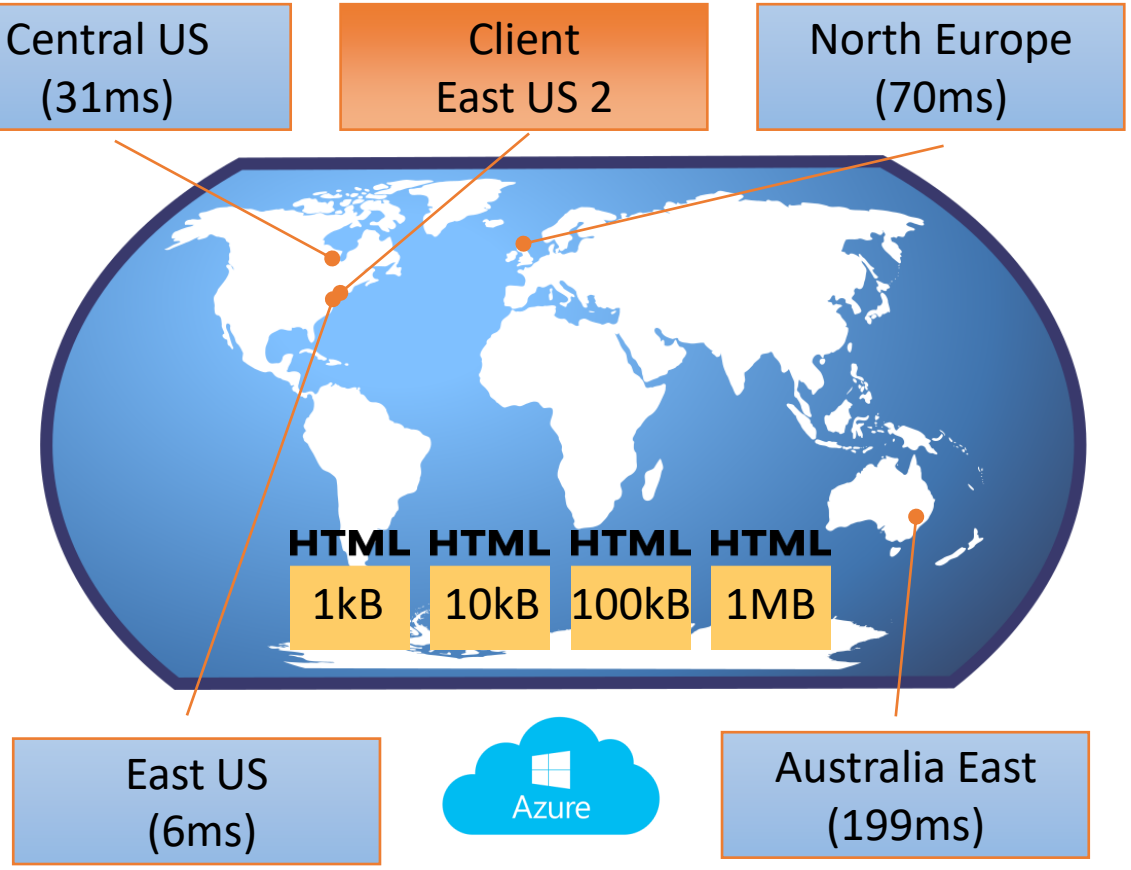
Linux (Ubuntu 18.04) - Azure D64s v3 VM – 64 vCPUs



“ Packet loss rate > 3-5%: larger fragmented artefacts need to be retransmitted ”

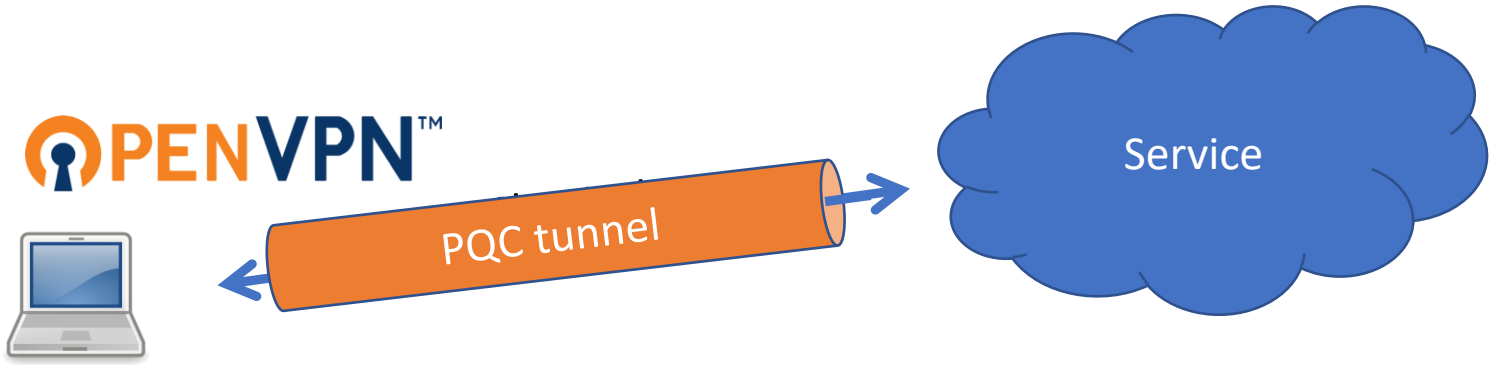
Benchmarking results #2





"The overhead of slower TLS connection establishment diminishes as a proportion of the overall page load time "

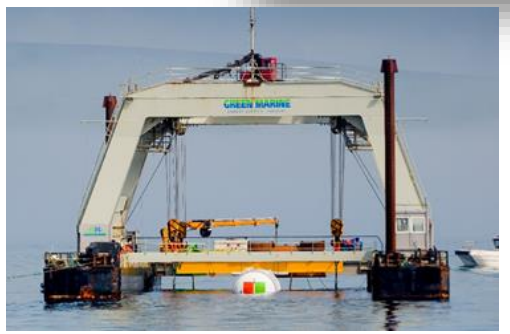
PQ VPN tunnels



- OpenVPN 2.4.8 integration
  - Uses OQS's OpenSSL fork
  - Easy legacy app tunneling
  - https://www.microsoft.com/en-us/research/project/post-quantum-crypto-vpn/

- Project Natick PQC VPN experiment
  - Natick is an underwater datacenter module off the coast of Scotland
  - We run a PQ VPN from Redmond
    - Uses ECDHE-P256 + SIKEp434 hybrid
    - Rekeying every hour

  - https://www.microsoft.com/en-us/research/project/post-quantum-crypto-tunnel-to-the-underwater-datacenter/



Quantum computers are coming…

Let's make sure
our crypto is

quantum safe!

cpaquin@microsoft.com
@chpaquin

https://www.microsoft.com/en-us/research/project/post-quantum-cryptography/