Post-quantum cryptography

Daniel J. Bernstein & Tanja Lange

University of Illinois at Chicago; Ruhr University Bochum & Technische Universiteit Eindhoven

12 September 2020

Cryptography

Sender
“Alice”

Receiver
“Bob”

Tsai Ing-Wen picture credit: By 總統府, Attribution, Wikimedia. Donald Trump picture credit: By Shealah Craighead - White House, Public Domain, Wikimedia.

Cryptography

Sender
"Alice"

Untrustworthy network
"Eve"

Receiver
"Bob"

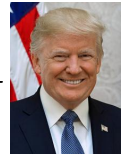
- Motivation #1: Communication channels are spying on our data.
- Motivation #2: Communication channels are modifying our data.

Tsai Ing-Wen picture credit: By 總統府, Attribution, Wikimedia. Donald Trump picture credit: By Shealah Craighead - White House, Public Domain, Wikimedia.

Cryptography

Sender
"Alice"

Untrustworthy network
"Eve"



Receiver
"Bob"

- Motivation #1: Communication channels are spying on our data.
- Motivation #2: Communication channels are modifying our data.
- Literal meaning of cryptography: "secret writing".
- Achieves various security goals by secretly transforming messages.
  - Confidentiality: Eve cannot infer information about the content
  - Integrity: Eve cannot modify the message without this being noticed
  - Authenticity: Bob is convinced that the message originated from Alice

Commonly used systems

Sender
"Alice"

Untrustworthy network
"Eve"

Receiver
"Bob"

Cryptography with symmetric keys
AES-128. **AES-192**. **AES-256**. **AES-GCM**. **ChaCha20**. **HMAC-SHA-256**. **Poly1305**. **SHA-2**. **SHA-3**. **Salsa20**.

Cryptography with public keys
BN-254. **Curve25519**. **DH**. **DSA**. **ECDH**. **ECDSA**. **EdDSA**. **NIST P-256**. **NIST P-384**. **NIST P-521**. **RSA encrypt**. **RSA sign**. **secp256k1**.

Tsai Ing-Wen picture credit: By 總統府, Attribution, Wikimedia. Donald Trump picture credit: By Shealah Craighead - White House, Public Domain, Wikimedia.


神威
太湖之光

Algorithms for Quantum Computation: Discrete Logarithms and Factoring

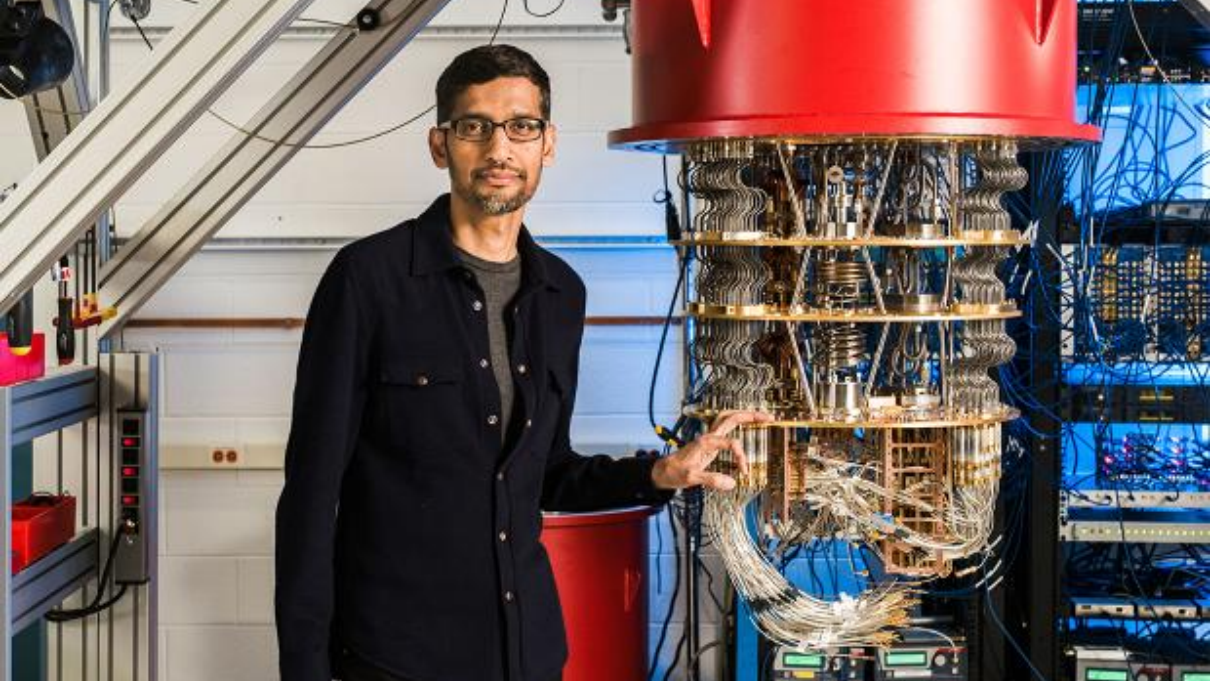
Peter W. Shor
AT&T Bell Labs
Room 2D-149
600 Mountain Ave.
Murray Hill, NJ 07974, USA

Abstract

A computer is generally considered to be a universal computational device; i.e., it is believed able to simulate any physical computational device with a cost in computation time of at most a polynomial factor. It is not clear whether this is still true when quantum mechanics is taken into consideration. Several researchers, starting with David Deutsch, have developed models for quantum mechanical computers and have investigated their compu-

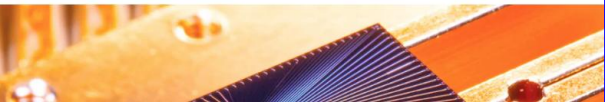
[1, 2]. Although he did not ask whether quantum mechanics conferred extra power to computation, he did show that a Turing machine could be simulated by the reversible unitary evolution of a quantum process, which is a necessary prerequisite for quantum computation. Deutsch [9, 10] was the first to give an explicit model of quantum computation. He defined both quantum Turing machines and quantum circuits and investigated some of their properties.

The next part of this paper discusses how quantum com-



Premium

Quantum computing could end encryption within five years, says Google boss



Mr Pichai said a combination of artificial intelligence and quantum would "help us tackle some of the biggest problems we see", but said it was important encryption evolved to match this.

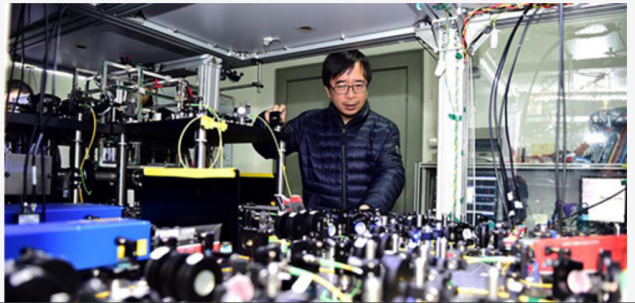
"In a five to ten year time frame, quantum computing will break encryption as we know it today."

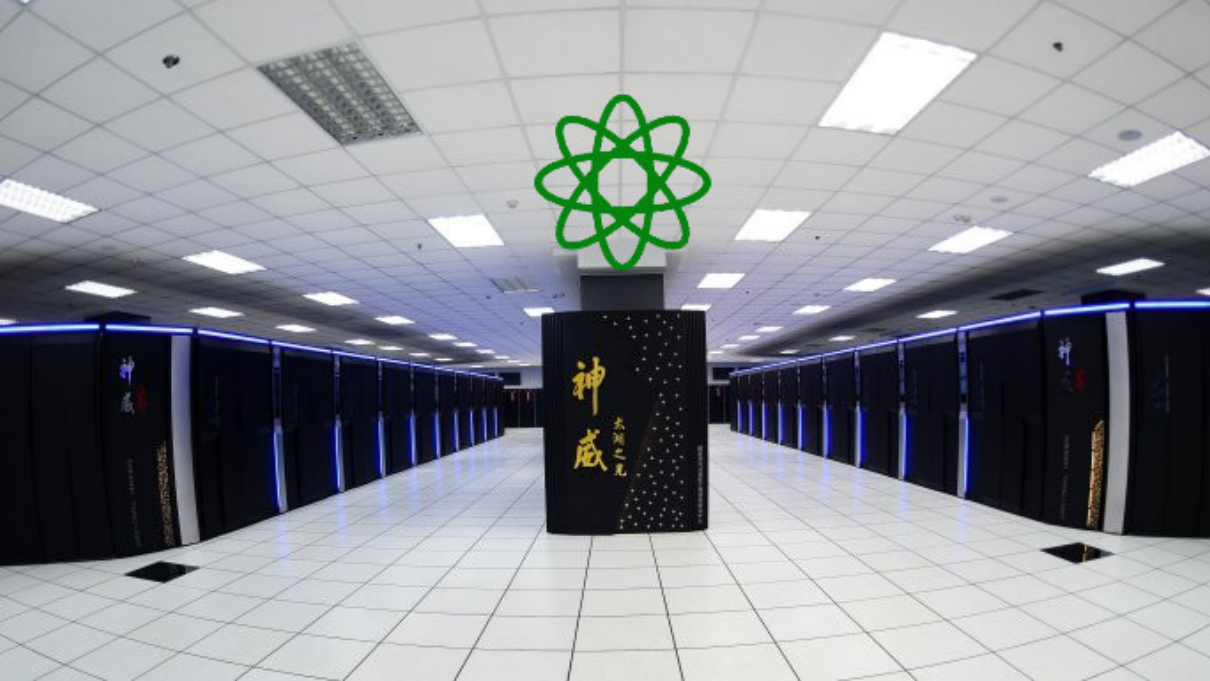
This is because current encryption methods, by which information such as texts or passwords is turned into code to make it unreadable, rely upon the fact that classic computers would take billions of years to decipher that code.

Quantum computers, with their ability to be

Chinese researchers expect quantum leap in computing, challenging Google's supremacy

Source: Global Times Published: 2020/8/26 14:58:42

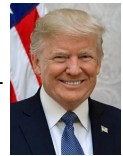



神威
太湖之光

Commonly used systems

Sender
"Alice"

Untrustworthy network
"Eve"



Receiver
"Bob"

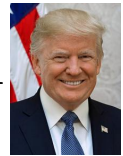
Cryptography with symmetric keys
AES-128. **AES-192**. **AES-256**. **AES-GCM**. **ChaCha20**. **HMAC-SHA-256**. **Poly1305**. **SHA-2**. **SHA-3**. **Salsa20**.

Cryptography with public keys
BN-254. **Curve25519**. **DH**. **DSA**. **ECDH**. **ECDSA**. **EdDSA**. **NIST P-256**. **NIST P-384**. **NIST P-521**. **RSA encrypt**. **RSA sign**. **secp256k1**.

Commonly used systems

Sender
"Alice"

Untrustworthy network
"Eve" with quantum computer



Receiver
"Bob"

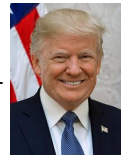
Cryptography with symmetric keys
AES-128. **AES-192**. **AES-256**. **AES-GCM**. **ChaCha20**. **HMAC-SHA-256**. **Poly1305**. **SHA-2**. **SHA-3**. **Salsa20**.

Cryptography with public keys
BN-254. **Curve25519**. **DH**. **DSA**. **ECDH**. **ECDSA**. **EdDSA**. **NIST P-256**. **NIST P-384**. **NIST P-521**. **RSA encrypt**. **RSA sign**. **secp256k1**.

Symmetric-key authenticated encryption

Sender
"Alice"

Untrustworthy network
"Eve" with quantum computer



Receiver
"Bob"

- Very easy solutions **if Alice and Bob already share long secret key k**:
  - "One-time pad" for confidentiality.
  - "Wegman–Carter MAC" for integrity and authenticity.
- AES-256: Standardized method to expand **short secret key** (256-bit k) into string indistinguishable from long secret key.
- AES introduced in 1998 by Daemen and Rijmen. Security analyzed in papers by dozens of cryptanalysts.
- No credible threat from quantum algorithms. Grover costs 2^{128}.
- Some results assume attacker has quantum access to computation, then some systems are weaker . . . but I'd know if my laptop had turned into a quantum computer.

Post-quantum cryptography

Cryptography under the assumption that the attacker has a quantum computer.

- 1994: Shor's quantum algorithm. 1996: Grover's quantum algorithm.
  Many subsequent papers on quantum algorithms: see `quantumalgorithmzoo.org`.
- 2003: Daniel J. Bernstein introduces term Post-quantum cryptography.
- 2006: First International Workshop on Post-Quantum Cryptography.
  PQCrypto 2006, 2008, 2010, 2011, 2013, 2014, 2016, 2017, 2018, 2019, (soon) 2020.
- 2015: NIST hosts its first workshop on post-quantum cryptography.
- 2016: NIST announces a standardization project for post-quantum systems.
- 2017: Deadline for submissions to the NIST competition.
- 2019: Second round of NIST competition begins.
- 2020: Third round of NIST competition begins.

21 December 2017: NIST posts 69 submissions from 260 people.

BIG QUAKE. **BIKE**. **CFPKM**. **Classic McEliece**. **Compact LWE**. **CRYSTALS-DILITHIUM**. **CRYSTALS-KYBER**. **DAGS**. **Ding Key Exchange**. **DME**. **DRS**. **DualModeMS**. **Edon-K**. **EMBLEM and R.EMBLEM**. **FALCON**. **FrodoKEM**. **GeMSS**. **Giophantus**. **Gravity-SPHINCS**. **Guess Again**. **Gui**. **HILA5**. **HiMQ-3**. **HK17**. **HQC**. **KINDI**. **LAC**. **LAKE**. **LEDAkem**. **LEDApkc**. **Lepton**. **LIMA**. **Lizard**. **LOCKER**. **LOTUS**. **LUOV**. **McNie**. **Mersenne-756839**. **MQDSS**. **NewHope**. **NTRU Prime**. **NTRU-HRSS-KEM**. **NTREncrypt**. **NTS-KEM**. **Odd Manhattan**. **OKCN/AKCN/CNKE**. **Ouroboros-R**. **Picnic**. **pqNTRUSign**. **pqRSA encryption**. **pqRSA signature**. **pqsigRM**. **QC-MDPC KEM**. **qTESLA**. **RaCoSS**. **Rainbow**. **Ramstake**. **RankSign**. **RLCE-KEM**. **Round2**. **RQC**. **RVB**. **SABER**. **SIKE**. **SPHINCS+**. **SRTPI**. **Three Bears**. **Titanium**. **WalnutDSA**.

By end of 2017: 8 out of 69 submissions attacked.

BIG QUAKE. **BIKE**. **CFPKM**. **Classic McEliece**. **Compact LWE**. **CRYSTALS-DILITHIUM**. **CRYSTALS-KYBER**. **DAGS**. **Ding Key Exchange**. **DME**. **DRS**. **DualModeMS**. **Edon-K**. **EMBLEM and R.EMBLEM**. **FALCON**. **FrodoKEM**. **GeMSS**. **Giophantus**. **Gravity-SPHINCS**. **Guess Again**. **Gui**. **HILA5**. **HiMQ-3**. **HK17**. **HQC**. **KINDI**. **LAC**. **LAKE**. **LEDAkem**. **LEDApkc**. **Lepton**. **LIMA**. **Lizard**. **LOCKER**. **LOTUS**. **LUOV**. **McNie**. **Mersenne-756839**. **MQDSS**. **NewHope**. **NTRU Prime**. **NTRU-HRSS-KEM**. **NTRUEncrypt**. **NTS-KEM**. **Odd Manhattan**. **OKCN/AKCN/CNKE**. **Ouroboros-R**. **Picnic**. **pqNTRUSign**. **pqRSA encryption**. **pqRSA signature**. **pqsigRM**. **QC-MDPC KEM**. **qTESLA**. **RaCoSS**. **Rainbow**. **Ramstake**. **RankSign**. **RLCE-KEM**. **Round2**. **RQC**. **RVB**. **SABER**. **SIKE**. **SPHINCS+**. **SRTPI**. **Three Bears**. **Titanium**. **WalnutDSA**.

Some less security than claimed; some really broken; some attack scripts.

By end of 2018: 22 out of 69 submissions attacked.

BIG QUAKE. **BIKE**. **CFPKM**. **Classic McEliece**. **Compact LWE**. **CRYSTALS-DILITHIUM**. **CRYSTALS-KYBER**. **DAGS**. **Ding Key Exchange**. **DME**. **DRS**. **DualModeMS**. **Edon-K**. **EMBLEM and R.EMBLEM**. **FALCON**. **FrodoKEM**. **GeMSS**. **Giophantus**. **Gravity-SPHINCS**. **Guess Again**. **Gui**. **HILA5**. **HiMQ-3**. **HK17**. **HQC**. **KINDI**. **LAC**. **LAKE**. **LEDAkem**. **LEDApkc**. **Lepton**. **LIMA**. **Lizard**. **LOCKER**. **LOTUS**. **LUOV**. **McNie**. **Mersenne-756839**. **MQDSS**. **NewHope**. **NTRU Prime**. **NTRU-HRSS-KEM**. **NTRUEncrypt**. **NTS-KEM**. **Odd Manhattan**. **OKCN/AKCN/CNKE**. **Ouroboros-R**. **Picnic**. **pqNTRUSign**. **pqRSA encryption**. **pqRSA signature**. **pqsigRM**. **QC-MDPC KEM**. **qTESLA**. **RaCoSS**. **Rainbow**. **Ramstake**. **RankSign**. **RLCE-KEM**. **Round2**. **RQC**. **RVB**. **SABER**. **SIKE**. **SPHINCS+**. **SRTPI**. **Three Bears**. **Titanium**. **WalnutDSA**.

Some less security than claimed; some really broken; some attack scripts.

30 January 2019: 26 candidates retained for second round.

. **BIKE**. . **Classic McEliece**. .
CRYSTALS-DILITHIUM. **CRYSTALS-KYBER**. . . . .
. . . . **FALCON**. **FrodoKEM**.
GeMSS. . . . . **HILA5**. . .
HQC. . **LAC**. **LAKE**. **LEDAkem**. **LEDApkc**. . . . **LOCKER**.
. **LUOV**. . . **MQDSS**. **NewHope**. **NTRU Prime**.
NTRU-HRSS-KEM. **NTRUEncrypt**. **NTS-KEM**. .
. **Ouroboros-R**. **Picnic**. . .
. . . . **qTESLA**. . **Rainbow**.
. . . **Round2**. **RQC**. . **SABER**. **SIKE**. **SPHINCS+**.
. **Three Bears**. . .

Some less security than claimed; some really broken; some attack scripts.
Merges for second round: HILA5 & Round2; LAKE, LOCKER, & Ouroboros-R;
LEDAkem & LEDApkc; NTRUEncrypt & NTRU-HRSS-KEM.

By end of 2019: 30 out of 69 submissions attacked.

. **BIKE**. . **Classic McEliece**. .
CRYSTALS-DILITHIUM. **CRYSTALS-KYBER**. . . .
. . . . **FALCON**. **FrodoKEM**.
GeMSS. . . . . **HILA5**. . .
HQC. . **LAC**. **LAKE**. **LEDAkem**. **LEDApkc**. . . . **LOCKER**.
. **LUOV**. . . **MQDSS**. **NewHope**. **NTRU Prime**.
NTRU-HRSS-KEM. **NTRUEncrypt**. **NTS-KEM**. .
. **Ouroboros-R**. **Picnic**. . .
. . . **qTESLA**. . **Rainbow**.
. . . **Round2**. **RQC**. . **SABER**. **SIKE**. **SPHINCS+**.
. **Three Bears**. . .

Some less security than claimed; some really broken; some attack scripts.
Merges for second round: HILA5 & Round2; LAKE, LOCKER, & Ouroboros-R;
LEDAkem & LEDApkc; NTRUEncrypt & NTRU-HRSS-KEM.

22 July 2020: 15 candidates retained for third round.

BIKE. **Classic McEliece**.
CRYSTALS-DILITHIUM. **CRYSTALS-KYBER**.
FALCON. **FrodoKEM**.
GeMSS.
HQC.
NTRU Prime.
NTRU-HRSS-KEM. **NTRUEncrypt**. **NTS-KEM**.
Picnic.
Rainbow.
SABER. **SIKE**. **SPHINCS+**.

Some less security than claimed; some really broken; some attack scripts.
Merges for second round: HILA5 & Round2; LAKE, LOCKER, & Ouroboros-R;
LEDAkem & LEDApkc; NTRUEncrypt & NTRU-HRSS-KEM.
Merges for third round: Classic McEliece & NTS-KEM.

National Academy of Sciences (US)

4 December 2018: Report on quantum computing

Don't panic. "Key Finding 1: Given the current state of quantum computing and recent rates of progress, it is highly unexpected that a quantum computer that can compromise RSA 2048 or comparable discrete logarithm-based public key cryptosystems will be built within the next decade."

National Academy of Sciences (US)

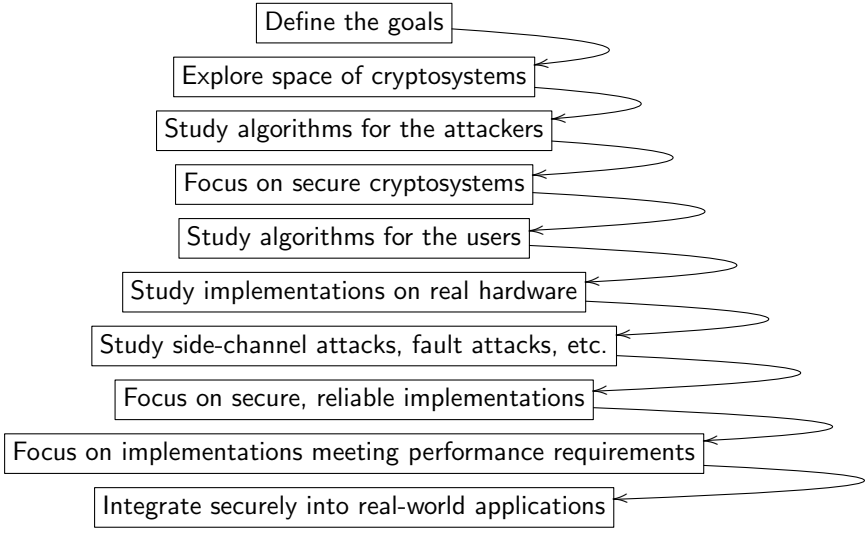
4 December 2018: Report on quantum computing

Don't panic. "Key Finding 1: Given the current state of quantum computing and recent rates of progress, it is highly unexpected that a quantum computer that can compromise RSA 2048 or comparable discrete logarithm-based public key cryptosystems will be built within the next decade."

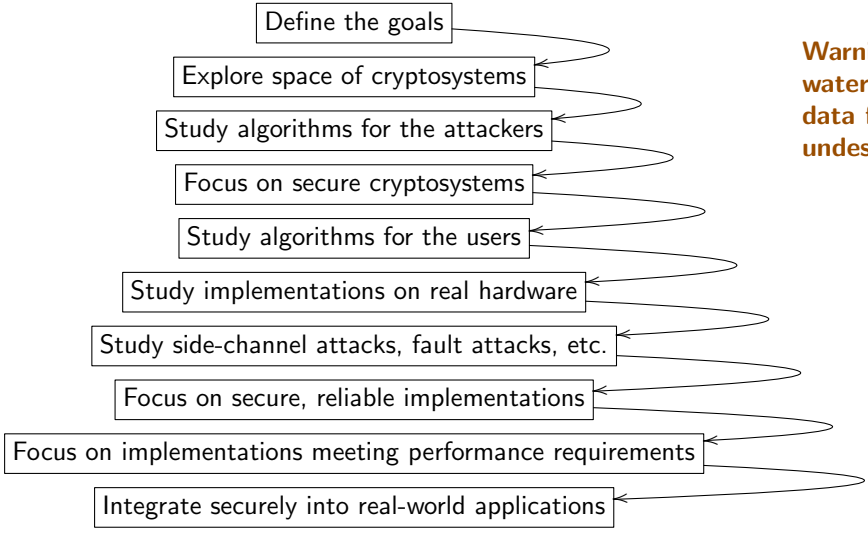
Panic. "Key Finding 10: Even if a quantum computer that can decrypt current cryptographic ciphers is more than a decade off, the hazard of such a machine is high enough—and the time frame for transitioning to a new security protocol is sufficiently long and uncertain—that prioritization of the development, standardization, and deployment of post-quantum cryptography is critical for minimizing the chance of a potential security and privacy disaster."

"[Section 4.4:] In particular, all encrypted data that is recorded today and stored for future use, will be cracked once a large-scale quantum computer is developed."

Many stages of research from design to deployment



Many stages of research from design to deployment

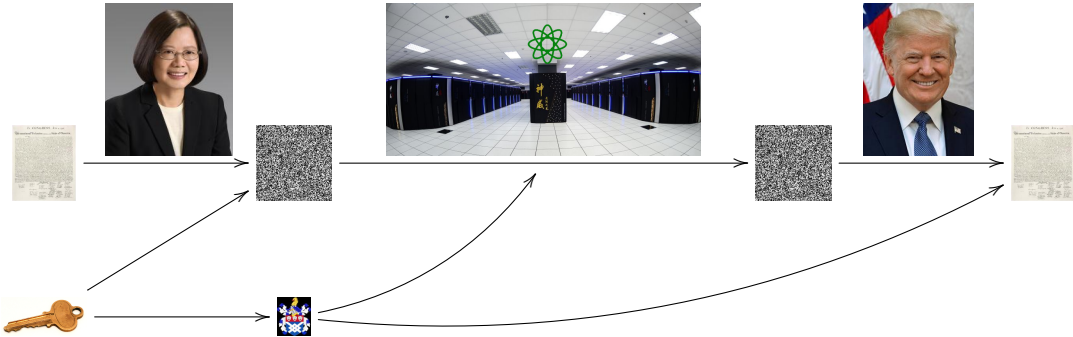


Warning: waterfall data flow, undesirable.

Major categories of public-key post-quantum systems

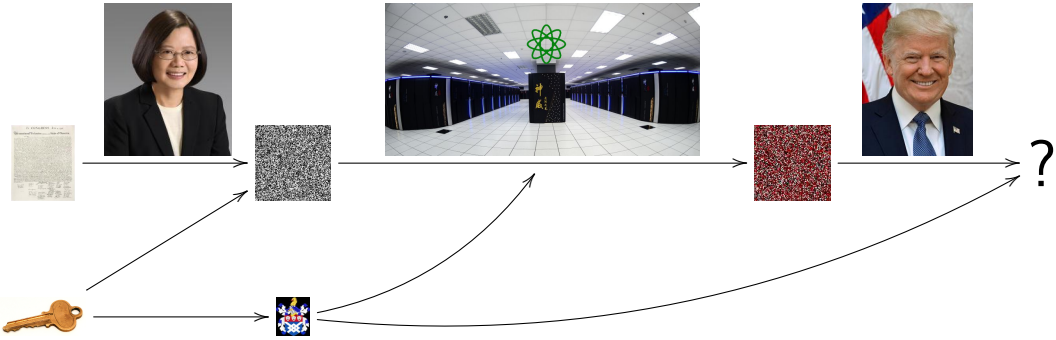
- **Code-based** encryption: McEliece cryptosystem has survived since 1978. Short ciphertexts and large public keys. Security relies on hardness of decoding error-correcting codes.
- **Hash-based** signatures: very solid security and small public keys. Require only a secure hash function (hard to find second preimages).
- **Isogeny-based** encryption: new kid on the block, promising short keys and ciphertexts and non-interactive key exchange. Security relies on hardness of finding isogenies between elliptic curves over finite fields.
- **Lattice-based** encryption and signatures: possibility for balanced sizes. Security relies on hardness of finding short vectors in some (typically special) lattice.
- **Multivariate-quadratic** signatures: short signatures and large public keys. Security relies on hardness of solving systems of multivariate equations over finite fields.

Post-quantum public-key signatures



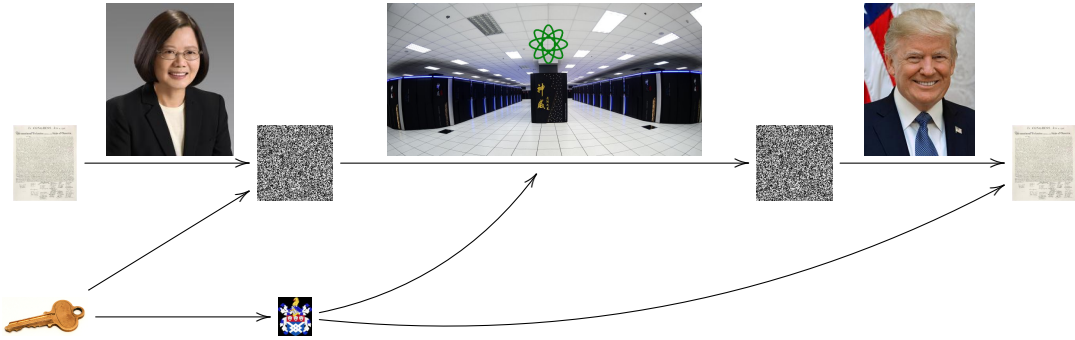
- Secret key  , public key  .

Post-quantum public-key signatures



- Secret key  , public key  .

Post-quantum public-key signatures: hash-based

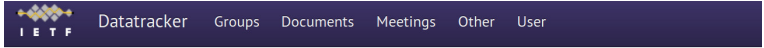


- Secret key  , public key  .
- Only one prerequisite: a good hash function, e.g. SHA3-512, ...
  Hash functions map long strings to fixed-length strings. $H : \{0,1\}^* \to \{0,1\}^n$.

  Signature schemes use hash functions in handling  .
- Quantum computers affect the hardness only marginally (Grover, not Shor).
- Old idea: 1979 Lamport one-time signatures; 1979 Merkle extends to more signatures.

On the fast track: stateful hash-based signatures

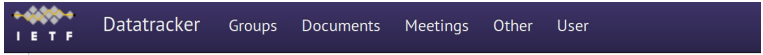
- CFRG has published 2 RFCs: RFC 8391 and RFC 8554



Internet Research Task Force (IRTF)
Request for Comments: 8391
Category: Informational
ISSN: 2070-1721

A. Huelsing
TU Eindhoven
D. Butin
TU Darmstadt
S. Gazdag
genua GmbH
J. Rijneveld
Radboud University
A. Mohaisen
University of Central Florida
May 2018

XMSS: eXtended Merkle Signature Scheme



Internet Research Task Force (IRTF)
Request for Comments: 8554
Category: Informational
ISSN: 2070-1721

D. McGrew
M. Curcio
S. Fluhrer
Cisco Systems
April 2019

Leighton-Micali Hash-Based Signatures

On the fast track: stateful hash-based signatures

- CFRG has published 2 RFCs: RFC 8391 and RFC 8554
- NIST has gone through two rounds of requests for public input, most are positive and recommend standardizing XMSS and LMS. Only concern is about statefulness in general.

On the fast track: stateful hash-based signatures

- CFRG has published 2 RFCs: RFC 8391 and RFC 8554
- NIST has gone through two rounds of requests for public input, most are positive and recommend standardizing XMSS and LMS. Only concern is about statefulness in general.

- ISO SC27 JTC1 WG2 has started a study period on stateful hash-based signatures.

A signature scheme for empty messages: key generation

A signature scheme for empty messages: key generation

First part of signempty.py

```
import os
import hashlib

def keypair():
  secret = sha3_256(os.urandom(32))
  public = sha3_256(secret)
  return public,secret
```

A signature scheme for empty messages: key generation

First part of signempty.py

```
import os
import hashlib

def keypair():
  secret = sha3_256(os.urandom(32))
  public = sha3_256(secret)
  return public,secret
```

```
>>> import signempty
>>> import binascii
>>> pk,sk = signempty.keypair()
>>> binascii.hexlify(pk)
b'a447bc8d7c661f85defcf1bbf8bad77bfc6191068a8b658c99c7ef4cbe37cf9f'
>>> binascii.hexlify(sk)
b'a4a1334a6926d04c4aa7cd98231f4b644be90303e4090c358f2946f1c257687a'
```

A signature scheme for empty messages: signing, verification

Rest of signempty.py

```
def sign(message,secret):
  if message != '': raise Exception('nonempty message')
  signedmessage = secret
  return signedmessage

def open(signedmessage,public):
  if sha3_256(signedmessage) != public:
    raise Exception('bad signature')
  message = ''
  return message
```

A signature scheme for empty messages: signing, verification

Rest of signempty.py

```
def sign(message,secret):
  if message != '': raise Exception('nonempty message')
  signedmessage = secret
  return signedmessage

def open(signedmessage,public):
  if sha3_256(signedmessage) != public:
    raise Exception('bad signature')
  message = ''
  return message
```

```
>>> sm = signempty.sign('',sk)
>>> signempty.open(sm,pk)
''
```

For more see Tanja's talks page for lecture on hash-based signatures and code snippets (some included here as bonus slides) at PQC Mini-School @Academia Sinica.

The best post-quantum systems in round 3 of the NIST competition

(= systems from us and from our colleagues at Academia Sinica)

- https://classic.mceliece.org: Classic McEliece.
  Code-based encryption.
- https://www.pqcrainbow.org: Rainbow.
  Multivariate-quadratic signatures.
- https://ntruprime.cr.yp.to: NTRU Prime.
  Lattice-based encryption.
- https://sphincs.org: SPHINCS+.
  Hash-based signatures.

Further information

- https://pqcrypto.org our overview page.
- PQCrypto 2016, PQCrypto 2017, PQCrypto 2018 all with slides from the talks; PQCrypto 2020 (21–23 September) online, free registration.
- https://pqcrypto.eu.org: PQCRYPTO EU Project.
  - PQCRYPTO recommendations.
  - Free software libraries (libpqcrypto, pqm4, pqhw).
  - Many reports, scientific articles, (overview) talks.
- https://2017.pqcrypto.org/school: PQCRYPTO summer school with 21 lectures on video, slides, and exercises.
- https://2017.pqcrypto.org/exec and https://pqcschool.org/index.html: Executive school (less math, more perspective).
- Quantum Threat Timeline from Global Risk Institute, 2019.
- https://csrc.nist.gov/Projects/post-quantum-cryptography/Post-Quantum-Cryptography-Standardization: NIST PQC competition.

Bonus slides

A signature scheme for 1-bit messages: key generation, signing

A signature scheme for 1-bit messages: key generation, signing

First part of signbit.py

```
import signempty

def keypair():
  p0,s0 = signempty.keypair()
  p1,s1 = signempty.keypair()
  return p0+p1,s0+s1

def sign(message,secret):
  if message == 0:
    return ('0' , signempty.sign('',secret[0:32]))
  if message == 1:
    return ('1' , signempty.sign('',secret[32:64]))
  raise Exception('message must be 0 or 1')
```

A signature scheme for 1-bit messages: verification

Rest of signbit.py

```
def open(signedmessage,public):
  if signedmessage[0] == '0':
    signempty.open(signedmessage[1],public[0:32])
    return 0
  if signedmessage[0] == '1':
    signempty.open(signedmessage[1],public[32:64])
    return 1
  raise Exception('message must be 0 or 1')
```

A signature scheme for 1-bit messages: verification

Rest of signbit.py

```
def open(signedmessage,public):
  if signedmessage[0] == '0':
    signempty.open(signedmessage[1],public[0:32])
    return 0
  if signedmessage[0] == '1':
    signempty.open(signedmessage[1],public[32:64])
    return 1
  raise Exception('message must be 0 or 1')
```

```
>>> import signbit
>>> pk,sk = signbit.keypair()
>>> sm = signbit.sign(1,sk)
>>> signbit.open(sm,pk)
1
```

A signature scheme for 4-bit messages: key generation

First part of sign4bits.py

```
import signbit

def keypair():
  p0,s0 = signbit.keypair()
  p1,s1 = signbit.keypair()
  p2,s2 = signbit.keypair()
  p3,s3 = signbit.keypair()
  return p0+p1+p2+p3,s0+s1+s2+s3
```

A signature scheme for 4-bit messages: sign & verify

Rest of sign4bits.py

```
def sign(m,secret):
  if type(m) != int: raise Exception('message must be int')
  if m < 0 or m > 15:
    raise Exception('message must be between 0 and 15')
  sm0 = signbit.sign(1 & (m >> 0),secret[0:64])
  sm1 = signbit.sign(1 & (m >> 1),secret[64:128])
  sm2 = signbit.sign(1 & (m >> 2),secret[128:192])
  sm3 = signbit.sign(1 & (m >> 3),secret[192:256])
  return sm0+sm1+sm2+sm3

def open(sm,public):
  m0 = signbit.open(sm[0:2],public[0:64])
  m1 = signbit.open(sm[2:4],public[64:128])
  m2 = signbit.open(sm[4:6],public[128:192])
  m3 = signbit.open(sm[6:],public[192:256])
  return m0 + 2*m1 + 4*m2 + 8*m3
```

Do not use one secret key to sign two messages!

```
>>> import sign4bits
>>> pk,sk = sign4bits.keypair()
>>> sm11 = sign4bits.sign(11,sk)
>>> sign4bits.open(sm11,pk)
11
>>> sm7 = sign4bits.sign(7,sk)
>>> sign4bits.open(sm7,pk)
7
>>> forgery = sm7[:6] + sm11[6:]
>>> sign4bits.open(forgery,pk)
15
```

Lamport's 1-time signature system

Sign arbitrary-length message by signing its 256-bit hash:

```
def keypair():
  keys = [signbit.keypair() for n in range(256)]
  public,secret = zip(*keys)
  return  public,secret

def sign(message,secret):
  msg = message.to_bytes(200, byteorder="little")
  h = sha3_256(msg)
  hbits = [1 & (h[i//8])>>(i%8) for i in range(256)]
  sigs = [signbit.sign(hbits[i],secret[i]) for i in range(256)]
  return sigs, message

def open(sm,public):
  message = sm[1]
  msg = message.to_bytes(200, byteorder="little")
  h = sha3_256(msg)
  hbits = [1 & (h[i//8])>>(i%8) for i in range(256)]
  for i in range(256):
    if hbits[i] != signbit.open(sm[0][i],public[i]):
      raise Exception('bit %d of hash does not match' % i)
  return message
```