Long-term security for cars

Daniel J. Bernstein[1,2] Tanja Lange[1]

[1]Technische Universiteit Eindhoven

[2]University of Illinois at Chicago

16 November 2016

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 computational properties. This paper gives Las Vegas algorithms for finding discrete logarithms and factoring integers on a quantum computer that take a number of steps which is polynomial in the input size, e.g., the number of digits of the integer to be factored. These two problems are generally considered hard on a classical computer and have been used as the basis of several proposed cryptosystems. (We thus give the first examples of quantum cryptanalysis.)

[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 computation relates to classical complexity classes. We will thus first give a brief intuitive discussion of complexity classes for those readers who do not have this background. There are generally two resources which limit the ability of computers to solve large problems: time and space (i.e., memory). The field of analysis of algorithms considers the asymptotic demands that algorithms make for these resources as a function of the problem size. Theoretical computer scientists generally classify algorithms as efficient when the number of steps of the algorithms grows as a polynomial in the size of the input. The class of prob-


D-Wave
D-Wave
D-Wave

D-Wave quantum computer isn't universal . . .

- Can't store stable qubits.
- Can't perform basic qubit operations.
- Can't run Shor's algorithm.
- Can't run other quantum algorithms we care about.

D-Wave quantum computer isn't universal . . .

- Can't store stable qubits.
- Can't perform basic qubit operations.
- Can't run Shor's algorithm.
- Can't run other quantum algorithms we care about.
- Hasn't managed to find any computation justifying its price.
- Hasn't managed to find any computation justifying 1% of its price.

But universal quantum computers are coming & are scary

- Massive research effort. Tons of progress summarized in, e.g., https://en.wikipedia.org/wiki/Timeline_of_quantum_computing.

But universal quantum computers are coming & are scary

- Massive research effort. Tons of progress summarized in, e.g., https://en.wikipedia.org/wiki/Timeline_of_quantum_computing.
- Mark Ketchen, IBM Research, 2012, on quantum computing: "Were actually doing things that are making us think like, 'hey this isn't 50 years off, this is maybe just 10 years off, or 15 years off.' It's within reach."
- Fast-forward to 2022, or 2027. Universal quantum computers exist.

But universal quantum computers are coming & are scary

- Massive research effort. Tons of progress summarized in, e.g., https://en.wikipedia.org/wiki/Timeline_of_quantum_computing.
- Mark Ketchen, IBM Research, 2012, on quantum computing: "Were actually doing things that are making us think like, 'hey this isn't 50 years off, this is maybe just 10 years off, or 15 years off.' It's within reach."
- Fast-forward to 2022, or 2027. Universal quantum computers exist.
- Shor's algorithm computes in polynomial time:
  - Integer factorization. RSA is dead.
  - Discrete-logarithms in finite fields. DSA is dead.
  - Discrete-logarithms on elliptic curves. ECDSA is dead.
- This breaks all current public-key cryptography on the Internet!

But universal quantum computers are coming & are scary

- Massive research effort. Tons of progress summarized in, e.g., https://en.wikipedia.org/wiki/Timeline_of_quantum_computing.
- Mark Ketchen, IBM Research, 2012, on quantum computing: "Were actually doing things that are making us think like, 'hey this isn't 50 years off, this is maybe just 10 years off, or 15 years off.' It's within reach."
- Fast-forward to 2022, or 2027. Universal quantum computers exist.
- Shor's algorithm computes in polynomial time:
  - Integer factorization. RSA is dead.
  - Discrete-logarithms in finite fields. DSA is dead.
  - Discrete-logarithms on elliptic curves. ECDSA is dead.
- This breaks all current public-key cryptography on the Internet!
- Also, Grover's algorithm speeds up brute-force searches.
- Example: Only 2^{64} quantum operations to break AES-128; 2^{128} quantum operations to break AES-256.

Is there any hope? Yes!

Post-quantum crypto is crypto that resists attacks by quantum computers.

- PQCrypto 2006: International Workshop on Post-Quantum Cryptography.

Is there any hope? Yes!

Post-quantum crypto is crypto that resists attacks by quantum computers.

- PQCrypto 2006: International Workshop on Post-Quantum Cryptography.
- PQCrypto 2008.

Is there any hope? Yes!

Post-quantum crypto is crypto that resists attacks by quantum computers.

- PQCrypto 2006: International Workshop on Post-Quantum Cryptography.
- PQCrypto 2008.
- PQCrypto 2010.

Is there any hope? Yes!

Post-quantum crypto is crypto that resists attacks by quantum computers.

- PQCrypto 2006: International Workshop on Post-Quantum Cryptography.
- PQCrypto 2008.
- PQCrypto 2010.
- PQCrypto 2011.
- PQCrypto 2013.
- PQCrypto 2014.

Is there any hope? Yes!

Post-quantum crypto is crypto that resists attacks by quantum computers.

- PQCrypto 2006: International Workshop on Post-Quantum Cryptography.
- PQCrypto 2008.
- PQCrypto 2010.
- PQCrypto 2011.
- PQCrypto 2013.
- PQCrypto 2014.
- New EU project, 2015–2018:
  PQCRYPTO, Post-Quantum Cryptography for Long-term Security.

PQCRYPTO
ICT-645622

NSA announcements

August 11, 2015

> *IAD recognizes that there will be a move, in the not distant future, to a quantum resistant algorithm suite.*

NSA announcements

August 11, 2015

IAD recognizes that there will be a move, in the not distant future, to a quantum resistant algorithm suite.

August 19, 2015

IAD will initiate a transition to quantum resistant algorithms in the not too distant future.

NSA announcements

August 11, 2015

> *IAD recognizes that there will be a move, in the not distant future, to a quantum resistant algorithm suite.*

August 19, 2015

> *IAD will initiate a transition to quantum resistant algorithms in the not too distant future.*

NSA comes late to the party and botches its grand entrance.

NSA announcements

August 11, 2015

> *IAD recognizes that there will be a move, in the not distant future, to a quantum resistant algorithm suite.*

August 19, 2015

> *IAD will initiate a transition to quantum resistant algorithms in the not too distant future.*

NSA comes late to the party and botches its grand entrance.

Worse, now we get people saying "Don't use post-quantum crypto, the NSA wants you to use it!".

Post-quantum becoming mainstream

- PQCrypto 2016: 22–26 Feb in Fukuoka, Japan, with more than 200 participants

- NIST is calling for post-quantum proposals; expect a small competition.
- PQCrypto 2017, Netherlands:
  - Jun 19 – 23 PQC school; Jun 22 & 23 Executive school
  - Jun 26 – 28 PQCrypto

Confidence-inspiring crypto takes time to build

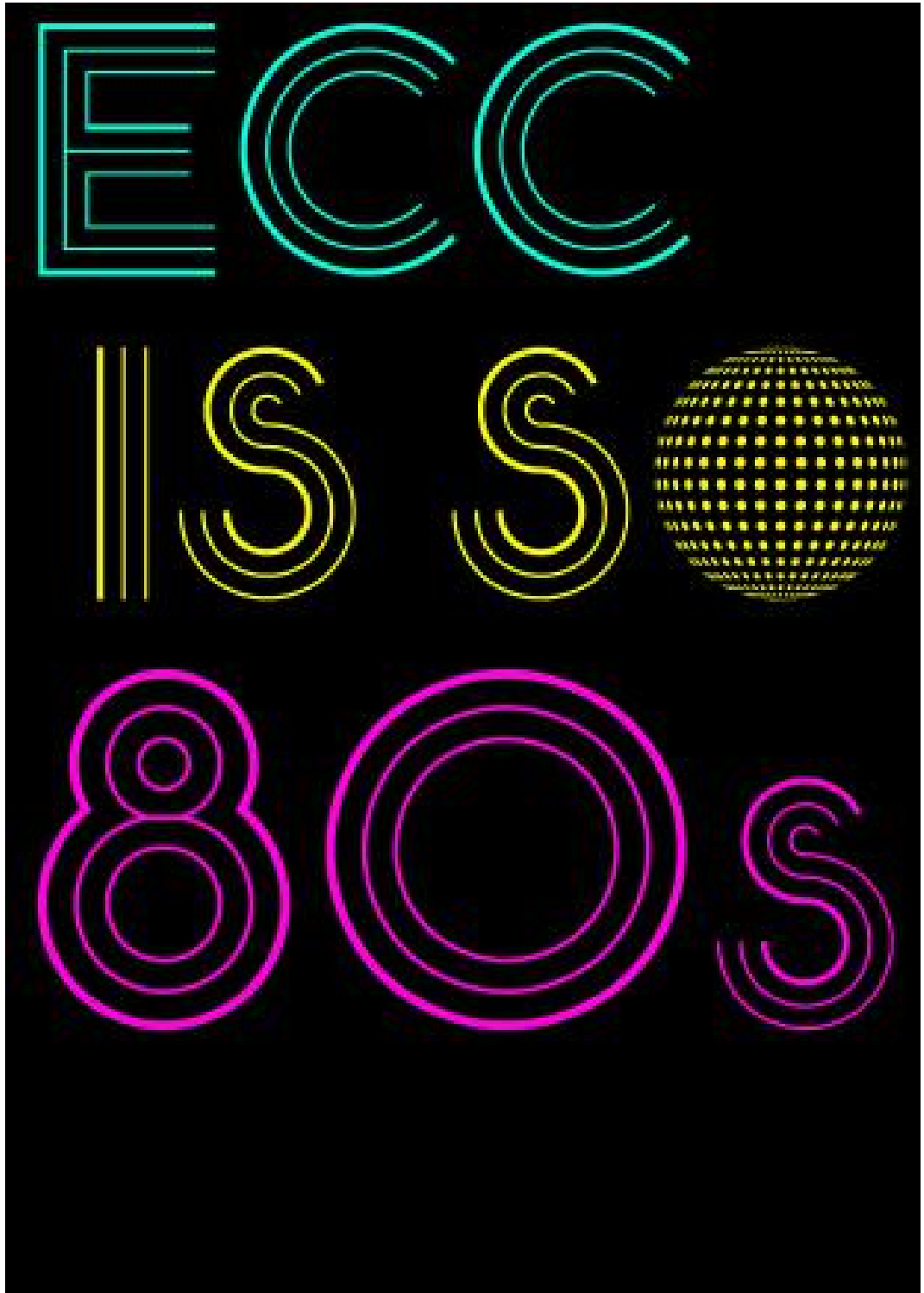
- Many stages of research from cryptographic design to deployment:
  - Explore space of cryptosystems.
  - Study algorithms for the attackers.
  - Focus on secure cryptosystems.

Confidence-inspiring crypto takes time to build

- Many stages of research from cryptographic design to deployment:
  - Explore space of cryptosystems.
  - Study algorithms for the attackers.
  - Focus on secure cryptosystems.
  - Study algorithms for the users.
  - Study implementations on real hardware.
  - Study side-channel attacks, fault attacks, etc.
  - Focus on secure, reliable implementations.
  - Focus on implementations meeting performance requirements.
  - Integrate securely into real-world applications.

Confidence-inspiring crypto takes time to build

- Many stages of research from cryptographic design to deployment:
  - Explore space of cryptosystems.
  - Study algorithms for the attackers.
  - Focus on secure cryptosystems.
  - Study algorithms for the users.
  - Study implementations on real hardware.
  - Study side-channel attacks, fault attacks, etc.
  - Focus on secure, reliable implementations.
  - Focus on implementations meeting performance requirements.
  - Integrate securely into real-world applications.
- Example: ECC introduced **1985**; big advantages over RSA. Robust ECC is starting to take over the Internet in **2015**.
- Post-quantum research can't wait for quantum computers!


ECC
IS
80s

Even higher urgency for long-term confidentiality

- Today's encrypted communication is being stored by attackers and will be decrypted years later with quantum computers. Danger for human-rights workers, medical records, journalists, security research, legal proceedings, state secrets, ...

- Signature schemes can be replaced once a quantum computer is built – but there will not be a public announcement

Even higher urgency for long-term confidentiality

- Today's encrypted communication is being stored by attackers and will be decrypted years later with quantum computers. Danger for human-rights workers, medical records, journalists, security research, legal proceedings, state secrets, ...

- Signature schemes can be replaced once a quantum computer is built – but there will not be a public announcement ... and an important function of signatures is to protect operating system upgrades.
- Protect your upgrades *now* with post-quantum signatures.

Next slide:
Initial recommendations
of long-term secure post-quantum systems

Daniel Augot, Lejla Batina, Daniel J. Bernstein, Joppe Bos,
Johannes Buchmann, Wouter Castryck, Orr Dunkelman,
Tim Güneysu, Shay Gueron, Andreas Hülsing,
Tanja Lange, Mohamed Saied Emam Mohamed,
Christian Rechberger, Peter Schwabe, Nicolas Sendrier,
Frederik Vercauteren, Bo-Yin Yang

Initial recommendations

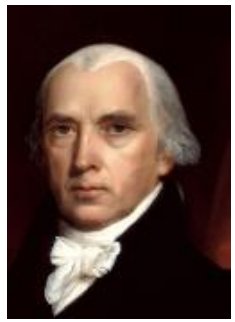
- **Symmetric encryption** Thoroughly analyzed, 256-bit keys:
  - AES-256
  - Salsa20 with a 256-bit key

  Evaluating: Serpent-256, ...
- **Symmetric authentication** Information-theoretic MACs:
  - GCM using a 96-bit nonce and a 128-bit authenticator
  - Poly1305
- **Public-key encryption** McEliece with binary Goppa codes:
  - length $n = 6960$, dimension $k = 5413$, $t = 119$ errors

  Evaluating: QC-MDPC, Stehlé-Steinfeld NTRU, ...
- **Public-key signatures** Hash-based (minimal assumptions):
  - XMSS with any of the parameters specified in CFRG draft
  - SPHINCS-256

  Evaluating: HFEv-, ...

Post-quantum public-key signatures: hash-based

$m \longrightarrow m, s \longrightarrow m, s \longrightarrow m$

$k \longrightarrow K$

- Secret key k, public key K.
- Only one prerequisite: a good hash function, e.g. SHA3-512. Hash functions map long strings to fixed-length strings. Signature schemes use hash functions in handling m.
- Old idea: 1979 Lamport one-time signatures.
- 1979 Merkle extends to more signatures.
- Many further improvements.
- Security thoroughly analyzed.

A signature scheme for empty messages: key generation

A signature scheme for empty messages: key generation

```
from simplesha3 import sha3256

def keypair():
  secret = sha3256(os.urandom(32))
  public = sha3256(secret)
  return public,secret
```

A signature scheme for empty messages: key generation

```
from simplesha3 import sha3256

def keypair():
  secret = sha3256(os.urandom(32))
  public = sha3256(secret)
  return public,secret
```

```
>>> import signempty
>>> pk,sk = signempty.keypair()
>>> binascii.hexlify(pk)
'a447bc8d7c661f85defcf1bbf8bad77bfc6191068a8b658c99c7...'
>>> binascii.hexlify(sk)
'a4a1334a6926d04c4aa7cd98231f4b644be90303e4090c358f29...'
```

A signature scheme for empty messages: signing, verification

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

def open(signedmessage,public):
  if sha3256(signedmessage) != public:
    raise Exception('bad signature')
  message = ''
  return message
```

A signature scheme for empty messages: signing, verification

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

def open(signedmessage,public):
  if sha3256(signedmessage) != public:
    raise Exception('bad signature')
  message = ''
  return message
```

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

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

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

```
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

```
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

```
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

```
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: signing

```
def sign(m,secret):
  if type(m) != int: raise Exception('m must be int')
  if m < 0 or m > 15:
    raise Exception('m 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
```

A signature scheme for 4-bit messages: verification

```
def open(sm,public):
  m0 = signbit.open(sm[0:33],public[0:64])
  m1 = signbit.open(sm[33:66],public[64:128])
  m2 = signbit.open(sm[66:99],public[128:192])
  m3 = signbit.open(sm[99:132],public[192:256])
  return m0 + 2*m1 + 4*m2 + 8*m3
```

Achtung: 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[:99] + sm11[99:]
>>> sign4bits.open(forgery,pk)
15
```

Lamport's 1-time signature system

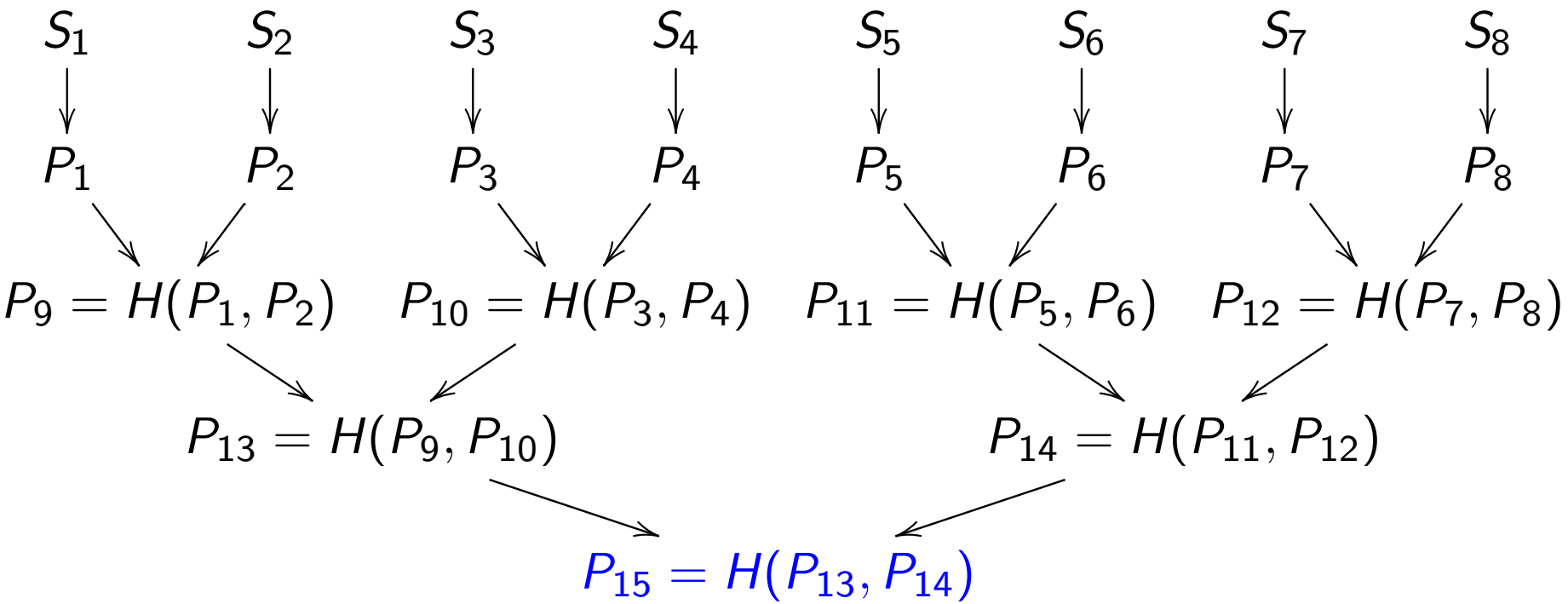
- Scale up to 256-bit messages.
- Sign arbitrary-length message by signing its 256-bit hash:

```
def sign(message,secret):
  h = sha3256(message)
  hbits = [1 & (ord(h[i/8])>>(i%8)) for i in range(256)]
  sigs = [signbit.sign(hbits[i],secret[64*i:64*i+64])
          for i in range(256)]
  return ''.join(sigs) + message
```

- Space improvement: "Winternitz signatures".

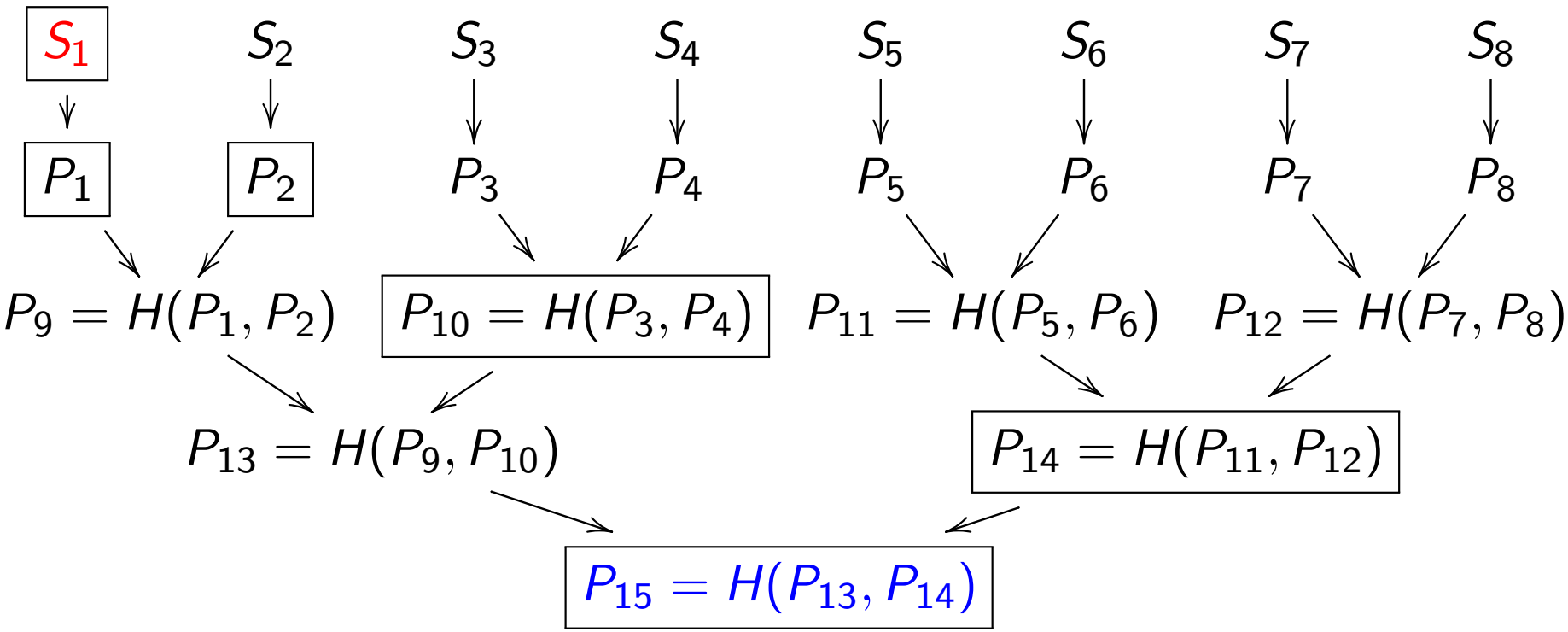
Merkle's (e.g.) 8-time signature system

Hash 8 Lamport one-time public keys into a single Merkle public key P_{15}.



Signature in 8-time Merkle hash tree

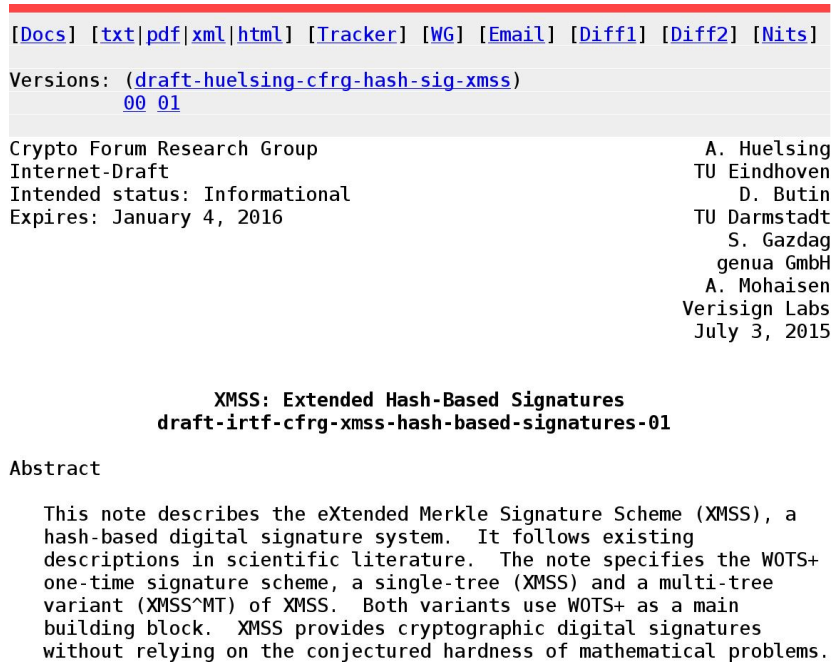
Signature of first message: $(\texttt{sign}(m, S_1), P_1, P_2, P_{10}, P_{14})$.



Pros and cons

Pros:

- Post quantum
- Only need secure hash function
- Small public key
- Security well understood
- Fast
- Proposed for standards: https://tools.ietf.org/html/draft-irtf-cfrg-xmss-hash-based-signatures-01



[Docs] [txt|pdf|xml|html] [Tracker] [WG] [Email] [Diff1] [Diff2] [Nits]

Versions: (draft-huelsing-cfrg-hash-sig-xmss)
00 01

Crypto Forum Research Group
Internet-Draft
Intended status: Informational
Expires: January 4, 2016

A. Huelsing
TU Eindhoven
D. Butin
TU Darmstadt
S. Gazdag
genua GmbH
A. Mohaisen
Verisign Labs
July 3, 2015

XMSS: Extended Hash-Based Signatures
draft-irtf-cfrg-xmss-hash-based-signatures-01

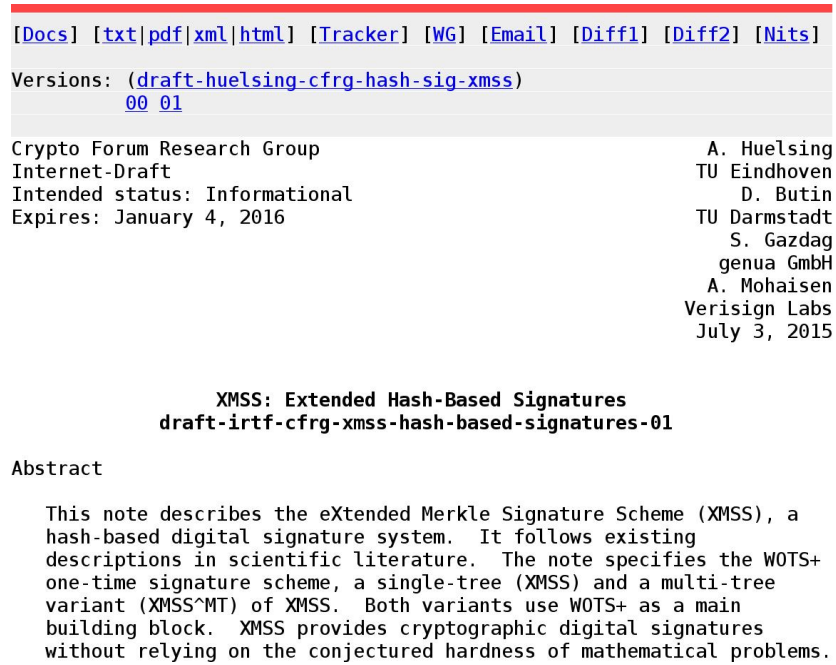
Abstract

This note describes the eXtended Merkle Signature Scheme (XMSS), a hash-based digital signature system. It follows existing descriptions in scientific literature. The note specifies the WOTS+ one-time signature scheme, a single-tree (XMSS) and a multi-tree variant (XMSS^MT) of XMSS. Both variants use WOTS+ as a main building block. XMSS provides cryptographic digital signatures without relying on the conjectured hardness of mathematical problems.

Pros and cons

Pros:

- Post quantum
- Only need secure hash function
- Small public key
- Security well understood
- Fast
- Proposed for standards: https://tools.ietf.org/html/draft-irtf-cfrg-xmss-hash-based-signatures-01



[Docs] [txt|pdf|xml|html] [Tracker] [WG] [Email] [Diff1] [Diff2] [Nits]

Versions: (draft-huelsing-cfrg-hash-sig-xmss)
00 01

Crypto Forum Research Group
Internet-Draft
Intended status: Informational
Expires: January 4, 2016

A. Huelsing
TU Eindhoven
D. Butin
TU Darmstadt
S. Gazdag
genua GmbH
A. Mohaisen
Verisign Labs
July 3, 2015

XMSS: Extended Hash-Based Signatures
draft-irtf-cfrg-xmss-hash-based-signatures-01

Abstract

This note describes the eXtended Merkle Signature Scheme (XMSS), a hash-based digital signature system. It follows existing descriptions in scientific literature. The note specifies the WOTS+ one-time signature scheme, a single-tree (XMSS) and a multi-tree variant (XMSS^MT) of XMSS. Both variants use WOTS+ as a main building block. XMSS provides cryptographic digital signatures without relying on the conjectured hardness of mathematical problems.

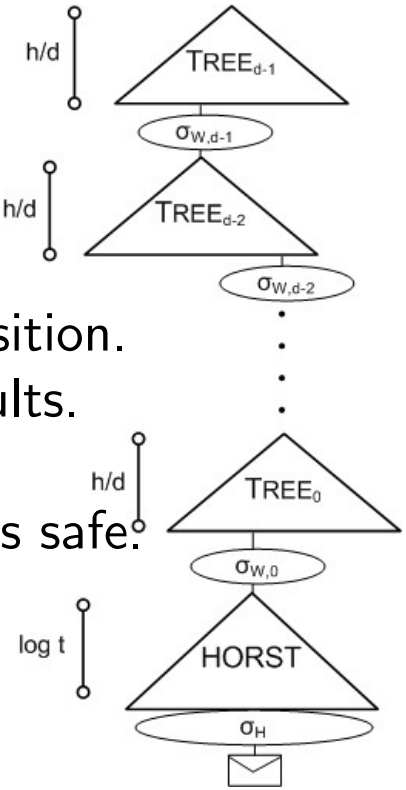
Cons:

- Biggish signature.
- Stateful. Adam Langley "for most environments it's a huge foot-cannon."

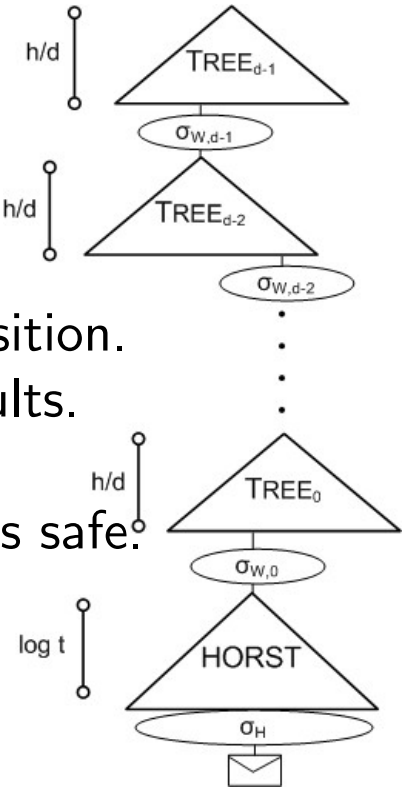
Useful for firmware upgrades (big server keeps state) or smart cards (HW counter).

Stateless hash-based signatures

- Idea from 1987 Goldreich:
  - Signer builds huge tree of certificate authorities.
  - Signature includes certificate chain.
  - Each CA is a hash of master secret and tree position.
    This is deterministic, so don't need to store results.
  - **Random** bottom-level CA signs message.
    Many bottom-level CAs, so one-time signature is safe.

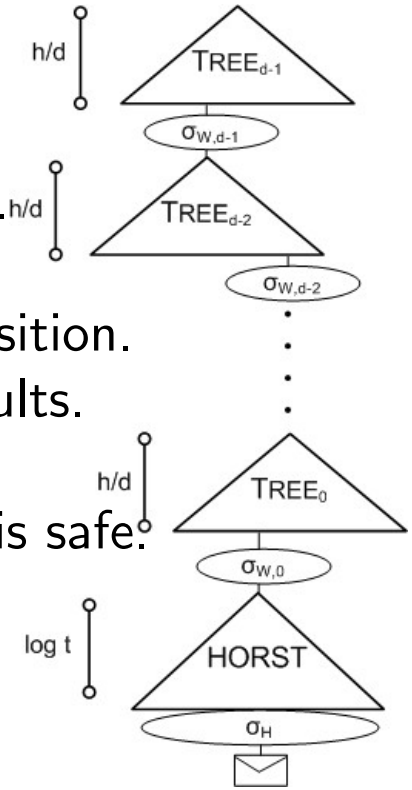


Stateless hash-based signatures



- Idea from 1987 Goldreich:
  - Signer builds huge tree of certificate authorities.
  - Signature includes certificate chain.
  - Each CA is a hash of master secret and tree position.
    This is deterministic, so don't need to store results.
  - **Random** bottom-level CA signs message.
    Many bottom-level CAs, so one-time signature is safe.
- 0.6 MB: Goldreich's signature with good 1-time signature scheme.
- 1.2 MB: average Debian package size.
- 1.8 MB: average web page in Alexa Top 1000000.

Stateless hash-based signatures



- Idea from 1987 Goldreich:
  - Signer builds huge tree of certificate authorities.
  - Signature includes certificate chain.
  - Each CA is a hash of master secret and tree position.
    This is deterministic, so don't need to store results.
  - **Random** bottom-level CA signs message.
    Many bottom-level CAs, so one-time signature is safe.
- 0.6 MB: Goldreich's signature with good 1-time signature scheme.
- 1.2 MB: average Debian package size.
- 1.8 MB: average web page in Alexa Top 1000000.
- 0.041 MB: SPHINCS signature, new optimization of Goldreich.
  Modular, guaranteed as strong as its components (hash, PRNG).
  Well-known components chosen for 2^{128} post-quantum security. `sphincs.cr.yp.to`

Examples of other post-quantum systems

- For symmetric crypto: use 256-bit keys
- Code-based encryption is well studied but has big keys; research into more compact systems.
- NTRU: lattice-based encryption system from late 1990's. Fast; relatively small ciphertext. Patent will expire 2017.
- BLISS signature scheme. Very recent lattice-based signature scheme. More modern system (has security proof) but hard to implement securely.
  CHES 2016 (Groot Bruinderink, Hülsing, Lange, Yarom) showed vulnerability under side-channel attacks.
- Many multivariate-quadratic systems. Some broken, some not. Highlight: very small signatures.
- More exotic possibility that needs analysis: isogeny-based crypto. Highlight: supports DH.

Further resources

- https://pqcrypto.org: Our survey site.
  - Many pointers: e.g., PQCrypto conference series.
  - Bibliography for 4 major PQC systemss.
- PQCrypto 2016 with slides and videos from lectures (incl. winter school)
- https://pqcrypto.eu.org: PQCRYPTO EU project.
  - Expert recommendations.
  - Free software libraries. (Coming soon)
  - More benchmarking to compare cryptosystems. (Coming soon)
  - 2017: workshop and spring/summer school.
- https://twitter.com/pqc_eu: PQCRYPTO Twitter feed.
  - Get used to post-quantum cryptosystems.
  - Improve; implement; integrate into real-world systems.