Post-quantum cryptography

Tanja Lange
(with Daniel J. Bernstein)

Technische Universiteit Eindhoven

PQCRYPTO
ICT-645622

17 January 2016

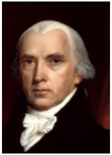
8th Winter School on Quantum Cybersecurity

Cryptography

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

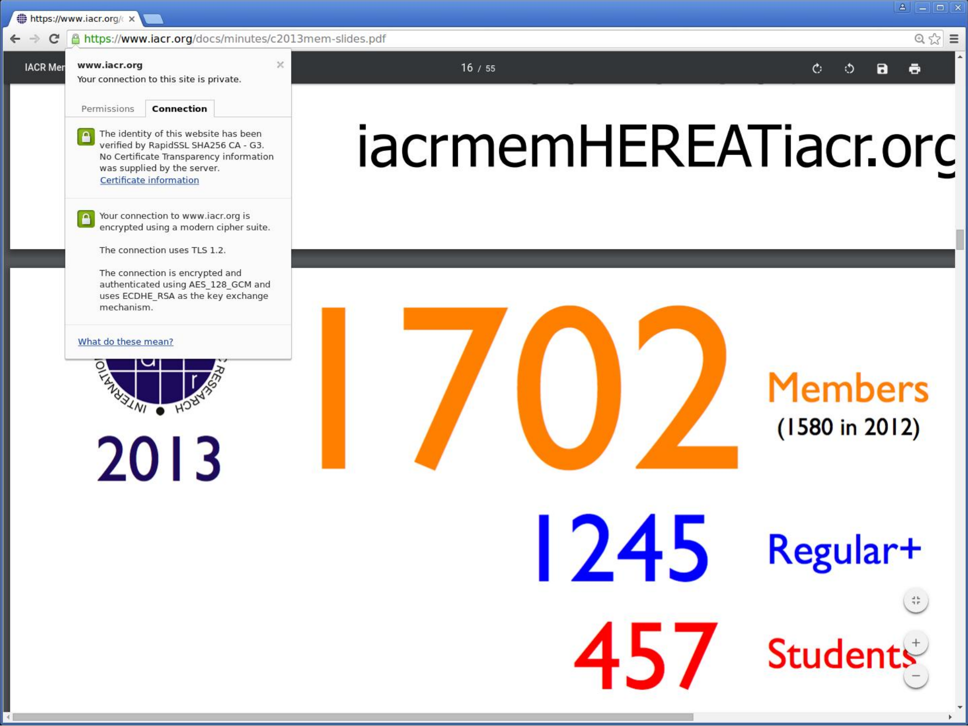
Sender
"Alice"

Untrustworthy network
"Eve"



Receiver
"Bob"

- Literal meaning of cryptography: "secret writing".
- Achieves various security goals by secretly transforming messages.



https://www.iacr.org/docs/minutes/c2013mem-slides.pdf
IACR Mem
16 / 55
www.iacr.org
Your connection to this site is private.
Permissions
Connection
The identity of this website has been verified by RapidSSL SHA256 CA - G3. No Certificate Transparency information was supplied by the server.
Certificate information
Your connection to www.iacr.org is encrypted using a modern cipher suite.
The connection uses TLS 1.2.
The connection is encrypted and authenticated using AES_128_GCM and uses ECDHE_RSA as the key exchange mechanism.
What do these mean?
iacrmemHEREATiacr.org
2013
1702 Members
(1580 in 2012)
1245 Regular+
457 Students


https://www.iacr.org/
https://www.iacr.org/docs/minutes/c2013mem-slides.pdf
IACR Mer
16 /
www.iacr.org
Your connection to this site is private.
Permissions
Connection
The identity of this website has been verified by RapidSSL SHA256 CA - G3. No Certificate Transparency information was supplied by the server.
Certificate information
Your connection to www.iacr.org is encrypted using a modern cipher suite.
The connection uses TLS 1.2.
The connection is encrypted and authenticated using AES_128_GCM and uses ECDHE_RSA as the key exchange mechanism.
What do these mean?
iacrm

Secret-key encryption

- Prerequisite: Alice and Bob share a secret key .
- Prerequisite: Eve doesn't know .
- Alice and Bob exchange any number of messages.
- Security goal #1: **Confidentiality** despite Eve's espionage.

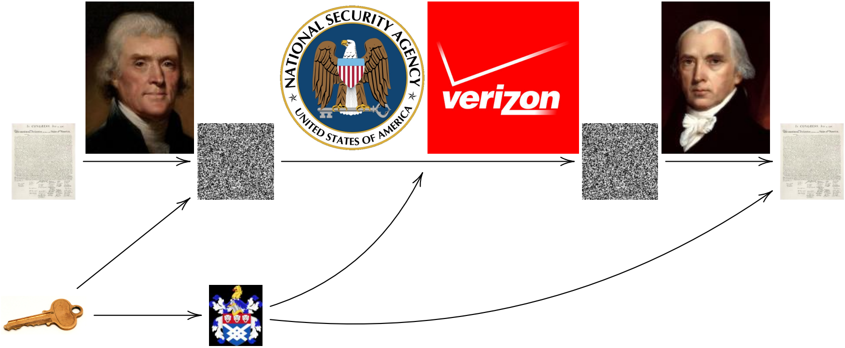
Secret-key authenticated encryption

- Prerequisite: Alice and Bob share a secret key .
- Prerequisite: Eve doesn't know .
- Alice and Bob exchange any number of messages.
- Security goal #1: **Confidentiality** despite Eve's espionage.
- Security goal #2: **Integrity**, i.e., recognizing Eve's sabotage.

Secret-key authenticated encryption

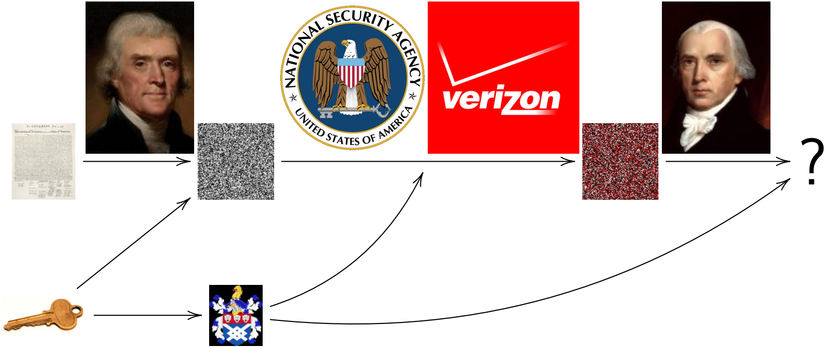
- Prerequisite: Alice and Bob share a secret key .
- Prerequisite: Eve doesn't know .
- Alice and Bob exchange any number of messages.
- Security goal #1: **Confidentiality** despite Eve's espionage.
- Security goal #2: **Integrity**, i.e., recognizing Eve's sabotage.

Public-key signatures



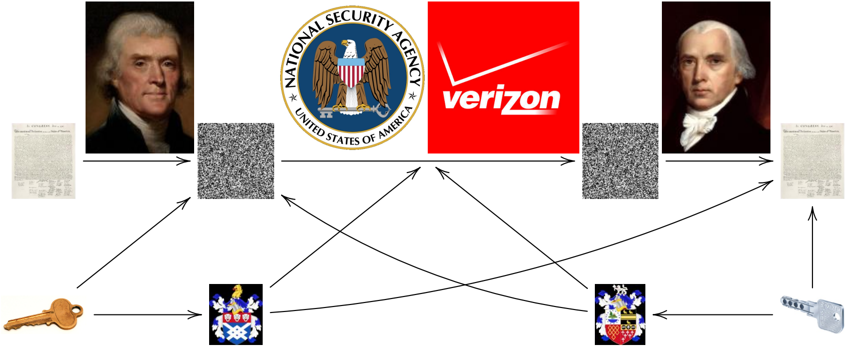
- Prerequisite: Alice has a secret key  and public key .
- Prerequisite: Eve doesn't know . Everyone knows .
- Alice publishes any number of messages.
- Security goal: Integrity.

Public-key signatures



- Prerequisite: Alice has a secret key  and public key .
- Prerequisite: Eve doesn't know . Everyone knows .
- Alice publishes any number of messages.
- Security goal: Integrity.

Public-key authenticated encryption ("DH" data flow)



- Prerequisite: Alice has a secret key  and public key .
- Prerequisite: Bob has a secret key  and public key .
- Alice and Bob exchange any number of messages.
- Security goal #1: Confidentiality.
- Security goal #2: Integrity.

Many more security goals studied in cryptography

- Protecting against denial of service.
- Stopping traffic analysis.
- Securely tallying votes.
- Searching encrypted data.
- Much more.

Attackers exploit physical reality

- 1996 Kocher: Typical crypto is broken by **side channels**.
- Response: Hundreds of papers on side-channel defenses.

Attackers exploit physical reality

- 1996 Kocher: Typical crypto is broken by **side channels**.
- Response: Hundreds of papers on side-channel defenses.
- Focus of this lecture: Large universal **quantum computers**.
- Mark Ketchen, IBM Research, 2012, on quantum computing: "We're 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 solves in polynomial time:
  - Integer factorization. RSA is dead.
  - The discrete-logarithm problem in finite fields. DSA is dead.
  - The discrete-logarithm problem on elliptic curves. ECDHE is dead.
- This breaks all current public-key cryptography on the Internet!

Attackers exploit physical reality

- 1996 Kocher: Typical crypto is broken by **side channels**.
- Response: Hundreds of papers on side-channel defenses.
- Focus of this lecture: Large universal **quantum computers**.
- Mark Ketchen, IBM Research, 2012, on quantum computing: "We're 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 solves in polynomial time:
  - Integer factorization. RSA is dead.
  - The discrete-logarithm problem in finite fields. DSA is dead.
  - The discrete-logarithm problem on elliptic curves. ECDHE 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.

Physical cryptography: a return to the dark ages

- Example: Locked briefcases.
- One-time pad is information-theoretically secure, i.e. no computational assumptions.
- Horrendously expensive.
- Can call it "locked-briefcase cryptography" but it's much more expensive than normal crypto.

Physical cryptography: a return to the dark ages

- Example: Locked briefcases.
- One-time pad is information-theoretically secure, i.e. no computational assumptions.
- Horrendously expensive.
- Can call it "locked-briefcase cryptography" but it's much more expensive than normal crypto.
- Broken again and again. Much worse track record than normal crypto.
- Easy to screw up. Easy to backdoor. Hard to audit.

Physical cryptography: a return to the dark ages

- Example: Locked briefcases.
- One-time pad is information-theoretically secure, i.e. no computational assumptions.
- Horrendously expensive.
- Can call it “locked-briefcase cryptography” but it’s much more expensive than normal crypto.
- Broken again and again. Much worse track record than normal crypto.
- Easy to screw up. Easy to backdoor. Hard to audit.
- Very limited functionality: e.g., no public-key signatures.

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 started to take over the Internet in **2015**.
- Can't wait for quantum computers before finding a solution!

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, ...

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!

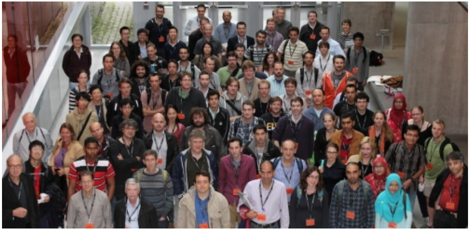
Post-quantum crypto is crypto that resists attacks by quantum computers.

- 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.

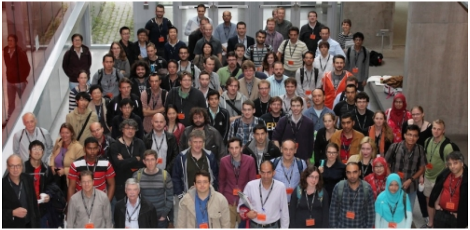
- PQCrypto 2006: International Workshop on Post-Quantum Cryptography.
- PQCrypto 2008.
- PQCrypto 2010.
- PQCrypto 2011.
- PQCrypto 2013.
- PQCrypto 2014.



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.

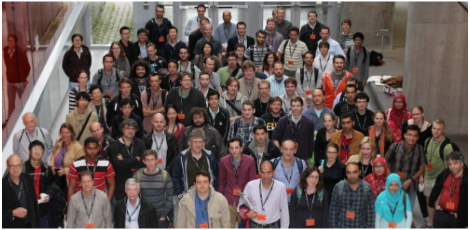


- PQCrypto 2016: 24–26 Feb in Fukuoka Japan. Winter school 22&23. https://pqcrypto2016.jp
- PQCrypto 2017 planned (in Europe).

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.



- PQCrypto 2016: 24–26 Feb in Fukuoka Japan.
  Winter school 22&23. https://pqcrypto2016.jp
- PQCrypto 2017 planned (in Europe).
- New EU project, 2015–2018:
  PQCRYPTO, Post-Quantum Cryptography for Long-term Security.

Post-Quantum Cryptography for Long-term Security

- Project funded by EU in Horizon 2020.
- Starting date 1 March 2015, runs for 3 years.
- 11 partners from academia and industry, TU/e is coordinator

Work packages

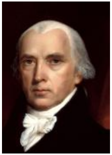
Technical work packages

- WP1: Post-quantum cryptography for small devices
  Leader: Tim Güneysu, co-leader: Peter Schwabe
- WP2: Post-quantum cryptography for the Internet
  Leader: Daniel J. Bernstein, co-leader: Bart Preneel
- WP3: Post-quantum cryptography for the cloud
  Leader: Nicolas Sendrier, co-leader: Lars Knudsen

Non-technical work packages

- WP4: Management and dissemination
  Leader: Tanja Lange
- WP5: Standardization
  Leader: Walter Fumy

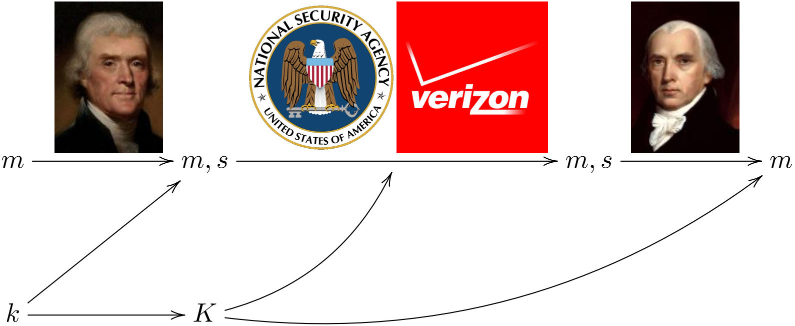
Post-quantum secret-key authenticated encryption



$$m \xrightarrow[k]{} c \longrightarrow c \xrightarrow[k]{} m$$

- Very easy solutions if secret key k is long uniform random string:
  - "One-time pad" for encryption.
  - "Wegman–Carter MAC" for authentication.
- AES-256: Standardized method to expand 256-bit k into string indistinguishable from long k.
- 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}.

Post-quantum public-key signatures: hash-based



- 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 <u>empty</u> messages: key generation

A signature scheme for <u>empty</u> 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)
'a447bc8d7c661f85defcf1bbf8bad77bfc6191068a8b658c99c7ef4cbe37cf9
>>> binascii.hexlify(sk)
'a4a1334a6926d04c4aa7cd98231f4b644be90303e4090c358f2946f1c257687
```

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('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
```

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}.

$$S_1 \to P_1,\quad S_2 \to P_2,\quad S_3 \to P_3,\quad S_4 \to P_4,\quad S_5 \to P_5,\quad S_6 \to P_6,\quad S_7 \to P_7,\quad S_8 \to P_8$$

$$P_9 = H(P_1, P_2) \qquad P_{10} = H(P_3, P_4) \qquad P_{11} = H(P_5, P_6) \qquad P_{12} = H(P_7, P_8)$$

$$P_{13} = H(P_9, P_{10}) \qquad P_{14} = H(P_{11}, P_{12})$$

$$P_{15} = H(P_{13}, P_{14})$$

Signature in 8-time Merkle hash tree

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

$$
\begin{tikzcd}[column sep=small]
\boxed{S_1} \arrow[d] & S_2 \arrow[d] & S_3 \arrow[d] & S_4 \arrow[d] & S_5 \arrow[d] & S_6 \arrow[d] & S_7 \arrow[d] & S_8 \arrow[d] \\
\boxed{P_1} \arrow[d] & \boxed{P_2} \arrow[dl] & P_3 \arrow[d] & P_4 \arrow[dl] & P_5 \arrow[d] & P_6 \arrow[dl] & P_7 \arrow[d] & P_8 \arrow[dl] \\
P_9 = H(P_1, P_2) \arrow[dr] & & \boxed{P_{10} = H(P_3, P_4)} \arrow[dl] & & P_{11} = H(P_5, P_6) \arrow[dr] & & P_{12} = H(P_7, P_8) \arrow[dl] & \\
& P_{13} = H(P_9, P_{10}) \arrow[drr] & & & & \boxed{P_{14} = H(P_{11}, P_{12})} \arrow[dll] & & \\
& & & \boxed{P_{15} = H(P_{13}, P_{14})} & & & &
\end{tikzcd}
$$

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	A. Huelsing
Internet-Draft	TU Eindhoven
Intended status: Informational	D. Butin
Expires: January 4, 2016	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

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

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

Crypto Forum Research Group — A. Huelsing
Internet-Draft — TU Eindhoven
Intended status: Informational — D. Butin
Expires: January 4, 2016 — 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:

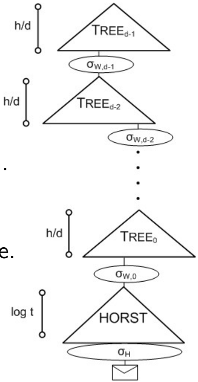
- Post quantum
- Only need secure hash function
- Small public key
- Security well understood
- Fast
- Proposed for standards: https://tools.ietf.org/html/draft-irtf-cfrg-xmss-hash-based-signatures-01

Cons:

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

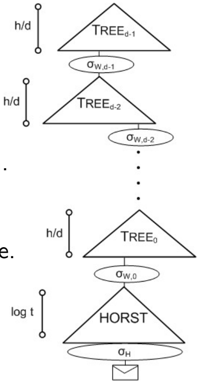
Stateless hash-based signatures

- 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.



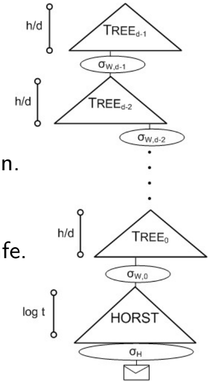
Stateless hash-based signatures

- 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.



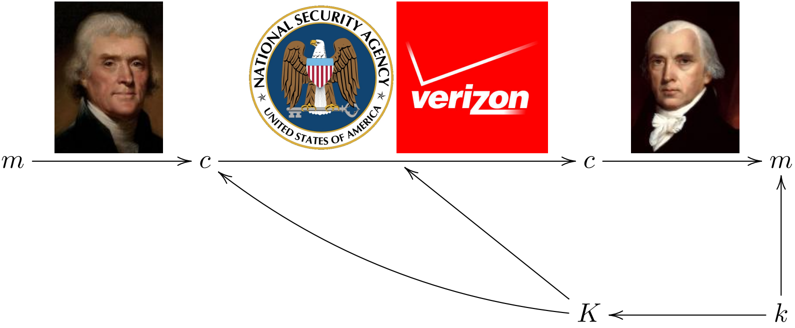
PQCRYPTO
ICT-645622

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

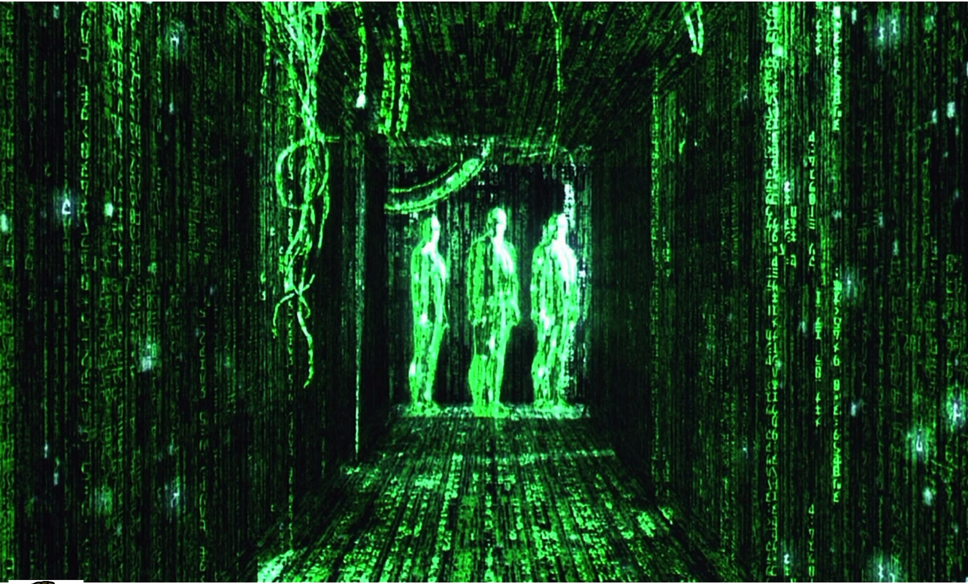
Post-quantum public-key encryption: code-based



- Alice uses Bob's public key K to encrypt.
- Bob uses his secret key k to decrypt.
- Code-based crypto proposed by McEliece in 1978.
- Almost as old as RSA, but much stronger security history.
- Many further improvements.

Error correction

- Digital media is exposed to memory corruption.
- Many systems check whether data was corrupted in transit:
  - ISBN numbers have check digit to detect corruption.
  - ECC RAM detects up to two errors and can correct one error.
    64 bits are stored as 72 bits: extra 8 bits for checks and recovery.
- In general, k bits of data get stored in n bits, adding some redundancy.
- If no error occurred, these n bits satisfy $n-k$ parity check equations; else can correct errors from the error pattern.
- Good codes can correct many errors without blowing up storage too much;
  offer guarantee to correct t errors (often can correct or at least detect more).
- To represent these check equations we need a matrix.



Hamming code

Parity check matrix $(n = 7, k = 4)$:

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

An error-free string of 7 bits $\mathbf{b} = (b_0, b_1, b_2, b_3, b_4, b_5, b_6)$ satisfies these three equations:

$$\begin{array}{ccccccccc} b_0 & & & +b_3 & +b_4 & & +b_6 & = & 0 \\ & b_1 & & +b_3 & & +b_5 & +b_6 & = & 0 \\ & & b_2 & & +b_4 & +b_5 & +b_6 & = & 0 \end{array}$$

If one error occurred at least one of these equations will not hold.
Failure pattern uniquely identifies the error location,
e.g., $1, 0, 1$ means

Hamming code

Parity check matrix $(n = 7, k = 4)$:

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

An error-free string of 7 bits $\mathbf{b} = (b_0, b_1, b_2, b_3, b_4, b_5, b_6)$ satisfies these three equations:

$$\begin{array}{ccccccccc} b_0 & & & +b_3 & +b_4 & & +b_6 & = & 0 \\ & b_1 & & +b_3 & & +b_5 & +b_6 & = & 0 \\ & & b_2 & & +b_4 & +b_5 & +b_6 & = & 0 \end{array}$$

If one error occurred at least one of these equations will not hold.
Failure pattern uniquely identifies the error location,
e.g., $1, 0, 1$ means b_4 flipped.

Hamming code

Parity check matrix $(n = 7, k = 4)$:

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

An error-free string of 7 bits $\mathbf{b} = (b_0, b_1, b_2, b_3, b_4, b_5, b_6)$ satisfies these three equations:

$$\begin{array}{ccccccccc} b_0 & & & +b_3 & +b_4 & & +b_6 & = & 0 \\ & b_1 & & +b_3 & & +b_5 & +b_6 & = & 0 \\ & & b_2 & & +b_4 & +b_5 & +b_6 & = & 0 \end{array}$$

If one error occurred at least one of these equations will not hold.
Failure pattern uniquely identifies the error location,
e.g., $1, 0, 1$ means b_4 flipped.
In math notation, the failure pattern is $H \cdot \mathbf{b}$.

Coding theory

- Names: code word $\mathbf{c}$, error vector $\mathbf{e}$, received word $\mathbf{b} = \mathbf{c} + \mathbf{e}$.
- Very common to transform the matrix so that the left part has just 1 on the diagonal (no need to store that).

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

- Many special constructions discovered in 65 years of coding theory:
  - Large matrix H.
  - Fast decoding algorithm to find $\mathbf{e}$ given $\mathbf{s} = H \cdot (\mathbf{c} + \mathbf{e})$, whenever $\mathbf{e}$ doesn't have too many bits set.
- Given large H, usually very hard to find fast decoding algorithm.
- Use this difference in complexities for encryption.

Code-based encryption

- 1971 Goppa: Fast decoders for many matrices H.
- 1978 McEliece: Use Goppa codes for public-key cryptography.
  - Original parameters designed for 2^{64} security.
  - 2008 Bernstein–Lange–Peters: broken in $\approx 2^{60}$ cycles.
  - Easily scale up for higher security.
- 1986 Niederreiter: Simplified and smaller version of McEliece.
  - Public key: H with 1's on the diagonal.
  - Secret key: the fast Goppa decoder.
  - Encryption: Randomly generate e with t bits set.
    Send $H \cdot e$.
  - Use hash of e to encrypt message with symmetric crypto (with 256 bits key).
- Very fast constant-time decryption:
  https://binary.cr.yp.to/mcbits.html.

Security analysis

- Some papers studying algorithms for attackers:
  1962 Prange; 1981 Omura; 1988 Lee–Brickell; 1988 Leon; 1989 Krouk;
  1989 Stern; 1989 Dumer; 1990 Coffey–Goodman; 1990 van Tilburg;
  1991 Dumer; 1991 Coffey–Goodman–Farrell; 1993 Chabanne–Courteau;
  1993 Chabaud; 1994 van Tilburg; 1994 Canteaut–Chabanne;
  1998 Canteaut–Chabaud; 1998 Canteaut–Sendrier;
  2008 Bernstein–Lange–Peters; 2009 Bernstein–Lange–Peters–van Tilborg;
  2009 Bernstein (post-quantum); 2009 Finiasz–Sendrier;
  2010 Bernstein–Lange–Peters; 2011 May–Meurer–Thomae;
  2011 Becker–Coron–Joux; 2012 Becker–Joux–May–Meurer;
  2013 Bernstein–Jeffery–Lange–Meurer (post-quantum);
  2015 May–Ozerov.

Security analysis

- Some papers studying algorithms for attackers:
  1962 Prange; 1981 Omura; 1988 Lee–Brickell; 1988 Leon; 1989 Krouk;
  1989 Stern; 1989 Dumer; 1990 Coffey–Goodman; 1990 van Tilburg;
  1991 Dumer; 1991 Coffey–Goodman–Farrell; 1993 Chabanne–Courteau;
  1993 Chabaud; 1994 van Tilburg; 1994 Canteaut–Chabanne;
  1998 Canteaut–Chabaud; 1998 Canteaut–Sendrier;
  2008 Bernstein–Lange–Peters; 2009 Bernstein–Lange–Peters–van Tilborg;
  2009 Bernstein (post-quantum); 2009 Finiasz–Sendrier;
  2010 Bernstein–Lange–Peters; 2011 May–Meurer–Thomae;
  2011 Becker–Coron–Joux; 2012 Becker–Joux–May–Meurer;
  2013 Bernstein–Jeffery–Lange–Meurer (post-quantum);
  2015 May–Ozerov.
- 256 KB public key for 2^{146} pre-quantum security.
- 512 KB public key for 2^{187} pre-quantum security.
- 1024 KB public key for 2^{263} pre-quantum security.

Security analysis

- Some papers studying algorithms for attackers:
  1962 Prange; 1981 Omura; 1988 Lee–Brickell; 1988 Leon; 1989 Krouk;
  1989 Stern; 1989 Dumer; 1990 Coffey–Goodman; 1990 van Tilburg;
  1991 Dumer; 1991 Coffey–Goodman–Farrell; 1993 Chabanne–Courteau;
  1993 Chabaud; 1994 van Tilburg; 1994 Canteaut–Chabanne;
  1998 Canteaut–Chabaud; 1998 Canteaut–Sendrier;
  2008 Bernstein–Lange–Peters; 2009 Bernstein–Lange–Peters–van Tilborg;
  2009 Bernstein (post-quantum); 2009 Finiasz–Sendrier;
  2010 Bernstein–Lange–Peters; 2011 May–Meurer–Thomae;
  2011 Becker–Coron–Joux; 2012 Becker–Joux–May–Meurer;
  2013 Bernstein–Jeffery–Lange–Meurer (post-quantum);
  2015 May–Ozerov.
- 256 KB public key for 2^{146} pre-quantum security.
- 512 KB public key for 2^{187} pre-quantum security.
- 1024 KB public key for 2^{263} pre-quantum security.
- Post-quantum (Grover): below 2^{263}, above 2^{131}.

Many more post-quantum suggestions

- QC-MDPC: variant with much smaller keys, but is it secure?
- Many more code-based systems. Some broken, some not.
- NTRU: 1990s "lattice-based" system, similar to QC-MDPC. Security story less stable than code-based cryptography.
- Many more lattice-based systems. Some broken, some not. e.g., 2014 quantum break of 2009 Smart–Vercauteren system.
- Many multivariate-quadratic systems. Some broken, some not. Highlight: very small signatures.
- More exotic possibility that needs analysis: isogeny-based crypto. Highlight: supports DH.

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-, ...

Further resources

- https://pqcrypto.org: Our survey site.
  - Many pointers: e.g., PQCrypto 2016.
  - Bibliography for 4 major PQC systems.
- https://pqcrypto.eu.org: PQCRYPTO EU project.
  Coming soon:
  - Expert recommendations.
  - Free software libraries.
  - More benchmarking to compare cryptosystems.
  - 2017: workshop and spring/summer school.
- https://twitter.com/pqc_eu: PQCRYPTO Twitter feed.