Post-Quantum Authentication in OpenSSL with Hash-Based Signatures

Denis Butin, Julian Wälde, and Johannes Buchmann

TU Darmstadt, Germany

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Overall Motivation

- Networking requires authentication; authentication is realized by cryptographic signature schemes
- Shor's algorithm (1994): most public-key cryptography (RSA, DSA, ECDSA) breaks once large quantum computers exist
- *Post-quantum cryptography*: public-key algorithms thought to be secure against quantum computer attacks

---

Overall Motivation

- Networking requires authentication; authentication is realized by cryptographic signature schemes
- Shor's algorithm (1994): most public-key cryptography (RSA, DSA, ECDSA) breaks once large quantum computers exist
- *Post-quantum cryptography*: public-key algorithms thought to be secure against quantum computer attacks

---

- Quantum computers are not available yet, but deployment of new crypto takes time, so transition must start now
- Well established post-quantum signature schemes: hash-based cryptography (XMSS and variants)
- Our goal: make post-quantum signatures available in a popular security software library: OpenSSL

Building Block: One-Time Signature Schemes (I)

- No complexity assumption
- Post-quantum
- Minimal security requirements: secure cryptographic hash function
- *Secure* meaning collision-resistant or merely second-preimage-resistant, depending on scheme
- Existing One-Time Signature (OTS) schemes: Lamport-Diffie (1979), Winternitz, W-OTS$^+$ (2013)...

Building Block: One-Time Signature Schemes (II)

- Private key randomly generated, public key is function of private key involving hash function
- Advanced OTS schemes feature a parameter for time/memory trade-off
- OTS schemes are inadequate on their own in practice: each OTS private key can only be used to sign a *single* message

Many-Time Signatures: Merkle Signature Scheme (I)

- Make OTS practical by combining many of them in a single structure: a complete binary tree
- Merkle signature scheme: combine 2^h OTS key pairs in a hash tree of height h
- Public key of the Merkle Signature Scheme is root of the binary tree
- Private key is the collection of OTS private keys

Many-Time Signatures: Merkle Signature Scheme (II)

- Central idea for signing and verifying with Merkle: authentication path (sequence of tree nodes to reconstruct path to root from leaf)
- Signatures also contain leaf index, to keep track of which OTS keys were already used (and therefore cannot be reused securely) — scheme said to be "stateful"

XMSS and XMSS^{MT}

Main differences with plain Merkle signature scheme:

- Specifically use the Winternitz OTS (or one of its variants)
- XOR-ing with random bitmasks for node construction
- In XMSS^{MT}, multiple layers of XMSS trees are used. Lowest layer signs actual messages, other layers sign root nodes of lower XMSS trees
- Optional in Internet-Draft (see next slide), mandatory in paper: OTS secret keys are generated from a seed value by a PRNG, so only seed must be stored
- Mandatory in Internet-Draft, not part of paper: domain separation to prevent multi-target attacks

Current Standardisation Process at CFRG / IRTF

Now in IRSG poll, expected to become first RFC on post-quantum cryptography:

Versions: (draft-huelsing-cfrg-hash-sig-xmss)
00 01 02 03 04 05 06 07 08 09 10

Crypto Forum Research Group
Internet-Draft
Intended status: Informational
Expires: January 25, 2018

A. Huelsing
TU Eindhoven
D. Butin
TU Darmstadt
S. Gazdag
genua GmbH
J. Rijneveld
Radboud University
A. Mohaisen
SUNY Buffalo

July 24, 2017

XMSS: Extended Hash-Based Signatures
draft-irtf-cfrg-xmss-hash-based-signatures-10

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. Instead, it is proven that it only relies on the properties of cryptographic hash functions. XMSS provides strong security guarantees and is even secure when the collision resistance of the underlying hash function is broken. It is suitable for compact implementations, relatively simple to implement, and naturally resists side-channel attacks. Unlike most other signature systems, hash-based signatures can withstand so far known attacks using quantum computers.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering

Motivations for Cryptographic Library Integration

- Bridge the gap towards actual use — system designers generally don't use stand-alone crypto scheme implementations
- Crypto library use often needed anyway in new crypto schemes. For hash-based signatures, we rely on existing implementations of hash functions, e.g. SHA256
- When local fork mature, pull request expected to speed up adoption in mainstream releases since library maintainers only need to check idiomatic leverage of library layers and make strategic decision, not implement scheme
- Prevent unofficial integration attempts by ill-informed third parties

Some Popular Cryptographic Libraries

- OpenSSL (C only) — focus on TLS/SSL, some other protocols supported, wide range of cryptographic algorithms. Security issues and extensive bug report processing backlog due to unsufficient manpower. Minimal documentation, but still the single most widely used crypto library.
- Projects forked from OpenSSL after Heartbleed: LibreSSL (OpenBSD, since 2014), BoringSSL (Google, since 2014)
- Bouncy Castle (Java, C#) — crypto API, ASN.1, TLS, X.509, S/MIME (email signing and encryption), CMS, OpenPGP... Founded 2000, MIT License
- Many others!

OpenSSL in General

- Open source toolkit implementing TLS, SSLv3 and a general purpose crypto library
- Successor of SSLeay (1995–1998)
- Historically, two main components:
  - `libcrypto`: implementation of cryptographic schemes and corresponding X.509 support
  - `libssl`: TLS/SSL security protocols, relying on `libcrypto`
- Nowadays: offers command line tool for key parameter creation, X.509 certificate generation, encryption / decryption, message digest calculation, S/MIME handling . . .

OpenSSL Abstraction Layers and Formats

- EVP — high-level cryptographic functions
- ASN.1 — standardised formal notation for networking data description interoperability
- NIDs — numbered values for ASN.1 object identifiers (OIDs) and other symbols. Used to identify crypto primitives
- Cipher suites — named combinations of algorithms (including authentication and encryption) used for client/server negotiation (Not OpenSSL-specific, but TLS/SSL-specific concept)

open-quantum-safe

Open Quantum Safe (OQS) project by Mosca and Stebila

Fork of OpenSSL 1.0.2 that adds quantum-safe cryptographic algorithms (`libcrypto` extension) and cipher suites (`libssl` extension)

Currently, focus on RLWE key exchange, but support of other primitives planned. Includes hybrid cipher suites which also use ECDHE key exchange.

Currently supports NewHope, Frodo and an older RLWE-KEX from IEEE S&P 2015.

openquantumsafe.org

Used as Basis: XMSS Reference Implementation

- Follows draft-irtf-cfrg-xmss-hash-based-signatures (Internet-Draft within the IRTF Crypto Forum Research Group)
- Plain C
- Uses structures for XMSS / XMSS^{MT} keys and signatures for clarity
- Closely follows Internet-Draft
- Prioritises readability, not performance, especially for algorithms not part of the I-D (since irrelevant for interoperability)

Available: pqsignatures.org » Publications

EVP Integration — Overview

- EVP functions: high level interface to OpenSSL cryptographic functions
- *Single consistent interface regardless of the underlying algorithm*
- Steps:
  - Wrote EVP functions mirroring keygen, initialisation, signing, verifying, etc
  - Those functions call corresponding reference implementation functions and bridge them with EVP
  - Inserted new functionality by defining a static `EVP_PKEY_METHOD` (see next slide)

EVP Integration — Method Mapping

```
void load_pmeth()
       {static EVP_PKEY_METHOD x = {
        NID_xmss, EVP_PKEY_FLAG_AUTOARGLEN,
        pkey_xmss_init, pkey_xmss_copy, pkey_xmss_cleanup,
        0, // paramgen_init
        0, //paramgen
        pkey_xmss_keygen_init, pkey_xmss_keygen, pkey_xmss_sign_init, pkey_xmss_sign,
        0, //verify_init
        pkey_xmss_verify,
        0, //verify_recover_init
        0, //verify_recover
        0, //signctx_init
        0, //signctx
        0, //verifyctx_init
        0, //verifyctx
        0, //encrypt_init
        0, //encrypt
        0, //decrypt_init
        0, //decrypt
        0, //derive_init
        0, //derive
        pkey_xmss_ctrl, pkey_xmss_ctrl_str};
        EVP_PKEY_meth_add0(&x);
       }
```

ASN.1 Integration (1/2)

- ASN.1 (Abstract Syntax Notation One): standardised formal notation to describe networking data for ("serialisation")
- Provides consistency across systems, hiding system-specific specificities
- Associated with standardised encoding rules, i.e. Distinguished Encoding Rules (DER)
- Commonly used e.g. in X.509 certificates
- Supported by OpenSSL and required to work with X.509 certificates

ASN.1 Integration (2/2)

Serialised structures, e.g. for XMSS secret key:

```
ASN1_SEQUENCE(xmssasn1sk) = {
 ASN1_SIMPLE(xmssasn1sk,idx,LONG),
 ASN1_SIMPLE(xmssasn1sk,n,LONG),
 ASN1_SIMPLE(xmssasn1sk,param,wots_param),
 ASN1_SIMPLE(xmssasn1sk,h,LONG),
 ASN1_SIMPLE(xmssasn1sk,PRF,ASN1_OCTET_STRING),
 ASN1_SIMPLE(xmssasn1sk,root,ASN1_OCTET_STRING),
 ASN1_SIMPLE(xmssasn1sk,seed,ASN1_OCTET_STRING),
 ASN1_SIMPLE(xmssasn1sk,nodes,ASN1_OCTET_STRING),
 ASN1_SIMPLE(xmssasn1sk,wots_keys,ASN1_OCTET_STRING)}
ASN1_SEQUENCE_END(xmssasn1sk);
// Macros:
DECLARE_ASN1_FUNCTIONS(xmssasn1sk)
IMPLEMENT_ASN1_FUNCTIONS(xmssasn1sk)
```

NID Integration

Automatic NID generation using OID as an input

Official NIDs for XMSS will be defined by IANA when Internet-Draft becomes RFC, so used preliminary value

OpenSSL has internal NID table, can expand statically, e.g. `NID_xmss = OBJ_create(<XMSS OID>, "xmssWithSHA256", ...)`

Cipher Suite Integration — Overview

We first integrated XMSS in a cipher suite using (pre-quantum) DH key exchange, in an OpenSSL 1.0.1 release

Then migrated to open-quantum-safe (OpenSSL 1.0.2 fork), which required some additional changes to non-cipher-suite aspects (OpenSSL seems to be constantly refactoring)

Finally, integrated XMSS in cipher suite using post-quantum key exchange, i.e. BCNS15

Cipher Suite Integration

Cipher suites defined in libssl, i.e. `s3_lib.c`, as one giant array `OPENSSL_GLOBAL SSL_CIPHER ssl3_ciphers[]`

```
{1,
 "RLWE-XMSS-AES256-GCM-SHA384", // Cipher suite name
 TLS1_CK_OQSKEX_GENERIC_XMSS_WITH_AES_256_GCM_SHA384,
 SSL_kOQSKEX_GENERIC, // KEX (currently BCNS15)
 NID_xmss, // NID, generated elsewhere, from OID
 SSL_AES256GCM,
 SSL_AEAD,
 SSL_TLSV1_2,
 SSL_NOT_EXP | SSL_HIGH | SSL_FIPS,
 SSL_HANDSHAKE_MAC_SHA384 | TLS1_PRF_SHA384,
 256,
 256,},
```

Implemented Test Framework

- PEM (container file format) file creation utilities for secret key, public key (using secret key as input), certificate (using keys as input)
- Certificate verification utility
- Format: base64 translation of X.509 ASN.1 keys
- `ssl_client` and `ssl_server` utilities that communicate and perform full TLS/SSL session using a chosen cipher suite
- Works with cipher suites using XMSS for authentication (in TLS/SSL, server typically authenticates itself to client ; converse is optional)

Conclusions

- Post-quantum cryptography must be deployed now to prepare in advance against quantum computer-aided attacks
- Many post-quantum cryptography schemes exist, but very little availability in popular security software libraries
- We experimentally integrated hash-based signature schemes such as XMSS in the popular OpenSSL security library
- Not straightforward, but we hope our account will help others to integrate post-quantum schemes into OpenSSL
- Not discussed here: we also integrated XMSS (including optimisations) in the official release of the (Java) *Bouncy Castle* library: `https://www.bouncycastle.org/releasenotes.html`

Thank You!

pqsignatures.org