




Production ready XMSS

Frans van Dorsselaer
Maximilian Fillinger, Pepijn Westen, Thomas Schaap
Niels Musters

Contents

- Origin of this library
- XMSS in a nutshell
- Use cases
- Requirements
- Fault Resilience
- Verification & Signing
- Guidance, Assurance, Evaluation

XMSS (features)

- Public key signatures, PQ-safe 
- Extremely secure, only requires preimage resistance of hash function (widely accepted; NIST, NLNCSA, ...)
- Predetermined maximum number of signatures (ranging from 1,000 to 1,000,000) 
- Stateful (private key needs to be updated when signing) 



XMSS (numbers)

- Public key: < 100 bytes
- Private key:
 - Static: < 200 bytes
 - Dynamic: < 100 bytes
 - Cache (optional): up to a few megabytes
- Signature: about 2 kiB
- Key generation is slow: > 1 billion hashes

Use cases → Firmware, Authenticity, ...

- Signing at manufacturer
(trusted environment, trusted equipment)
- Verification in device
(untrusted environment)
- Examples:
 - Automotive
 - Smart meter



Asymmetric requirements

- Signing:
 - State storage
 - Backup / load-balancing redundancy
 - Bit error resilience
- Verification:
 - Fast
 - Small
 - Fault injection resilience

Bit error resilience

- Simple booleans are too naive
 - Use Hamming(8,4), don't use all 0s or all 1s
 - 8-bit helps platforms such as ARM (immediate addressing mode)
- Redundancy
 - Perform critical operations multiple times and compare results
 - Store values in multiple fields and compare pre- and post-operation
- Integrity
 - Digest all critical data pre- and post-operation

Fault injection resilience (verification only)

On top of bit error resilience: protect against power/clock glitching

Example attacks:

- (timed) instruction skipping at boot
- (timed) data manipulation at boot

Hardware platform dependency (verification)

A generic software implementation can never provide fault injection resilience!

- The API was designed to enable (facilitate) resilience, at no cost for those that do not require it.
- API supports all scenarios:
 - Minimum size
 - Maximum performance
 - Maximum resilience

Verification library

Design goals:

- Small
 - Unused algorithm (SHA-2 or SHAKE) can be disabled at compile time
- Fast
 - Even when verifying multiple times for resilience
- Pluggable
 - Allow use of hardware accelerated hash implementation

Result:

```
XmssError xmss_calculate_expected_public_key(  
    XmssValue256 *restrict expected_public_key,  
    const XmssBuffer *restrict msg,  
    const XmssPublicKeyBlob *restrict pub_key,  
    const XmssSignatureBlob *restrict signature);
```

Verification example

- Raspberry Pi Pico microcontroller
- SHA-256 (SHAKE256/256 disabled)

→ 1,800 bytes code

→ < 2k RAM



Signing (storage)

API separates:

- Static part (small, stored once)
- Dynamic part (small, stored often)
- Public cache part (large, stored once, not required)

This allows **smartcard** implementations, PCs, etc.

Storing is done before signing. Multiple signatures can be reserved.

Signing (backup)

Contradiction:

- Long-term availability requires backups
- State must never be reused; implies no backups

API solution: **partitioning** (the RFC way, not the NIST way)

- Each partition can be sub-divided
- Consecutive partitions can be glued

Solves both backups and redundancy (active-active)

Signing (key generation)

Random must be provided by user; the library is deterministic (requirement).

API supports generating the key in parts for:

- Easy progress monitoring
- Multithreading

Example: largest key on Intel i9: **< 2 minutes**.

(on a smartcard probably > 1 day).

Fox Crypto additional security feature

Standard XMSS leaks:

- Number of signatures generated
- Partition used

→ **Index obfuscation**

Pseudo-random permutation (Fisher-Yates shuffle) of indices.

Fully compatible with standard XMSS, transparent for verification.

Guidance

Library is available at <https://github.com/FoxCryptoNL/xmss>

Free (MIT license)

Includes guidance for:

- Resilience
- Storage
- Backups
- Hardware acceleration
- Public key pinning

Guidance example

CRYPTO
part of FOX-IT

XMSS Library

Q Search

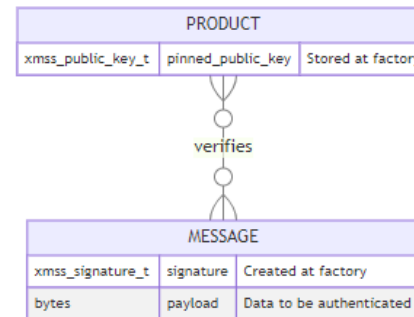
XMSS Library

- Public key authentication**
- Signature verification
- Public key obfuscation
- Signature count hiding
- Hash optimization
- Secure boolean functions
- Secure conditional branches
- Classes
- Files

Public key authentication

The public key needs to be authenticated before it can be trusted to verify signatures.

As a common scenario, we consider a manufacturer creating products that require firmware updates at the site of the customer after production. Usually, the public key is 'authenticated' by storing it, preferably irreversibly, in the product at manufacturing time; for example:



Depending on the backup strategy, one could pin multiple public keys; for example:

Assurance / Evaluation

Fox Crypto on premise:

- Evaluation evidence: **targets Common Criteria EAL5**
- To be published later:
 - Test suite (already developed)
 - Examples
 - High level language wrappers

