Batch me if you PQ-Sign
-- 7th NIST PQ Seminar

Carlos Aguilar-Melchor
Martin R. Albrecht
Thomas Bailleux
James Howe
Andreas Hülsing
David Joseph
Marc Manzano

Nina Binzel
@NinaBindel
nbindel
TLS – Quick Explainer

Client → Encrypted Application Data → Server

Server’s certificate including the server’s verification key

Generate signature over the transcript using the server’s signing key
Signature generation might become the computational bottleneck of TLS connections.
Higher computational cost of PQ signatures severely impacts the ability of systems to scale and might inhibit their migration to PQC, especially in higher-throughput settings.
Batch Signing to the Rescue


- Uses Merkle trees to decrease number of signature generations needed to ease scalability for classical signature standards

- Particularly useful when used with **PQ signatures**
- We show how to apply it to applications **beyond the intended TLS use case** and to also **decrease communication cost** in addition to computation costs
Outline

Exemplified using TLS

Batch Signing Scheme Using Merkle Trees
- Construction Idea
- Security and Privacy Guarantees
- Experimental Results

Use-Cases to Decrease Computation and Communication Cost

Exemplified using TLS
Using Batch Signing for TLS
Main Idea: Using Batch Signing for TLS


- Request 1
- Request 2
- Request 3
- Request 4
- Request 5

\[
H_1 = \text{Hash}(R_1, R_2) \\
H_2 = \text{Hash}(R_3, R_4) \\
H_3 = \text{Hash}(R_5, \text{Dummy}) \\
H_4 = \text{Hash}(H_1, H_2) \\
H_5 = \text{Hash}(H_3, \text{Dummy}) \\
\text{Root} = \text{Hash}(H_4, H_5)
\]

Generate signature over the root.

- Request 1
- Request 2
- Request 3
- Request 4
- Request 5
- Dummy
Main Idea: Using Batch Signing for TLS


- **Path 1**: Generate unique paths to reconstruct Merkle tree
- **Path 2**: Generate signature over the root

- \(H1 = \text{Hash}(R1, R2)\)
- \(H2 = \text{Hash}(R3, R4)\)
- \(H3 = \text{Hash}(R5, \text{Dummy})\)
- \(H4 = \text{Hash}(H1, H2)\)
- \(H5 = \text{Hash}(H3, \text{Dummy})\)
- Root = \(\text{Hash}(H4, H5)\)

- **Path 1**
- **Path 2**
- **Path 3**
- **Path 4**
- **Path 5**

- **Request 1**
- **Request 2**
- **Request 3**
- **Request 4**
- **Request 5**

- **Dummy**

- Only 1 instead of five signatures needs to be generated
- At the cost of small increase of the signature size and latency
KEMTLS replaces static server authentication with a static KEM, so that only the involved KEM public keys need to be signed rather than the transcript.

- KEMTLS leads to a higher throughput with no latency increase.
- KEMTLS needs a number of significant infrastructure changes.
Security and Privacy Guarantees
Security Guarantees

Batch signatures are unforgable if the signatures computed over the Merkle tree root are unforgable (EUF-CMA) and the (tweakable) hash function used to build the Merkle tree is target collision resistant (SM-TCR).

Essentially the same as plain signature unforgability

• Improvement over IETF Internet Draft that required a collision-resistant hash function.
• Leads to decreased authentication paths (half the size).
• SM-TCR is a fundamentally weaker assumption.
Privacy Guarantees

Batch Privacy
Given two signatures, an adversary cannot decide whether they were signed in the same batch.

• We can’t achieve that because of the shared root signature throughout the batch.
• Adversary learns two connections were in a certain time window to the same client

Weak Batch Privacy
Signatures from the same batch do not leak anything about a message for which no signature is made available

Batch signatures offer ‘weak batch privacy’ if the hash function used to build the Merkle tree is a one-time pseudo-random function.
Experimental Results for TLS
Experimental Setup

The high-level architecture and dependencies of our batch signing TLS experiments

- OpenSSL fork making use of liboqs (PQ algorithms) and ring (ECDSA)
- RUST implementation of batch signing, including building the Merkle tree and batch signing functionalities
- Between 1-4 clients and 1 server
- Batch size of 16 or 32 (to optimize latency)
- Averaged over 20 sec
- Google Cloud C2 instance with Intel 3.9 GHz Cascade Lake processor
Signature Sizes (in Bytes)

For PQ signature schemes: very small relative increase

ECDSA-256
Falcon-512
SPHINCS+-256f

Dilithium-II
Falcon-1024

Dilithium-V

SPHINCS+-128f

<200 byte
<100 byte
# TLS Handshakes/sec

- Speed-up for faster PQ algs ~1.5x-2.0x
- Slow PQ algs ~4x
- Dilithium-II more handshakes than ECDSA
- No infrastructure changes, just client update
- Optimized for max #handshakes and acceptable latency delay

### Chart

- ECDSA-256
- Dilithium-II
- Dilithium-V
- Falcon-512
- Falcon-1024
- Falcon-512 (fpemu)
- Falcon-1024 (fpemu)
- SPHINCS+-128f
- SPHINCS+-256f
- Batch signing for 16 or 32 messages

- Plain signatures

### Performance

- 1.25x
- 1.72x
- 1.72x
- 1.54x
- 1.79x
- 3.2x
- 3.15x
- 4.67x
- 3.73x

**Notes:**
- Optimized for max #handshakes and acceptable latency delay
### TLS Handshakes/sec Together with Latency Increase

<table>
<thead>
<tr>
<th>Signature Scheme</th>
<th>Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECDSA-256</td>
<td>18</td>
</tr>
<tr>
<td>Dilithium-II</td>
<td>15</td>
</tr>
<tr>
<td>Dilithium-V</td>
<td>12</td>
</tr>
<tr>
<td>Falcon-512</td>
<td>9</td>
</tr>
<tr>
<td>Falcon-1024</td>
<td>6</td>
</tr>
<tr>
<td>Falcon-512 (fpemu)</td>
<td>3</td>
</tr>
<tr>
<td>Falcon-1024 (fpemu)</td>
<td>0</td>
</tr>
<tr>
<td>SPHINCS+-128f</td>
<td></td>
</tr>
<tr>
<td>SPHINCS+-256f</td>
<td></td>
</tr>
<tr>
<td>Plain signatures</td>
<td></td>
</tr>
<tr>
<td>Batch signing for 16 or 32 messages</td>
<td>0-18</td>
</tr>
</tbody>
</table>

Latency:
- **Median**
- **90th percentile**
- **99th percentile**
Latency comparison

Max number of requests that can be answered immediately

- Flexibility to react to number of incoming request by adjusting tree size
- Slower increase of latency with increased number of requests after capacity of server’s signature generations is reached.

Latency comparison diagram:

- Plain signatures
- Batch signatures

# Requests per second
Use-cases Beyond TLS to Decrease Communication and Computational Cost
Reducing Computational Cost

Hardware Security Modules (HSMs)
- Generate large sets of (short-lived) certificates
- HSMs are significantly slower than traditional CPUs:

  \[
  \sim 10,000 \text{ sig/sec} \quad \text{vs} \quad \sim 100 \text{ sig/sec}
  \]
  
  modern commodity CPUs \quad \text{enterprise-grade HSM}

- Refrain from giving performance comparison, since PQ HSMs are not available yet. Therefore, a performance comparison would not reflect reality.
- Under assumption that certificate requester is able to verify batch signatures
Existing Proposals to Use Batch Signing to Decrease Communication Cost


- Using Merkle trees transform any signature scheme into a stateful signature scheme with compressed signature size


- Introducing a new certificate format to decrease the signature/certificate/communication size in TLS

In comparison, the presented batch signing approach does not need infrastructure changes and can be offered by the signer as one signature scheme that can be negotiated by the requester.
Reducing Communication Cost

Generally, use-cases where verifiers are aware that batch signing is used and in which batch their request is.

During TLS communications in which clients communicate with servers from the same batch, the certificate could consist of the unique path as signature as long as the full certificate with path and root signature is communicated once.

The HSM can broadcast root signature and drop it from the individual batch signatures.
Use-case: Certificate Renewal in Mutual TLS (mTLS) Mesh

• Saving **communication cost** within the mesh network
• Reduced **storage** requirements for certificates

Saving **computational cost**
Summary

- Provision of security and privacy foundation using the reductionist approach
- Study performance trade-offs for TLS:
  - Increased throughput for PQ algorithm of $1.5x - 4.6x$
  - Under increased signature size of less than 200 byte
  - Under acceptable latency increase of at most 25% or 0.5 ms
- Suggest and explain use-cases beyond TLS

IACR eprint 2023/492

Thank you!

We are looking for interns/residents. Check out sandboxaq.com/careers

All icons from flaticon with premium.

@NinaBindel

nbindel