TLS Batch Signing

Fun with Merkle trees

David Benjamin
TLS handshake costs

- Key derivations and other symmetric crypto
  - Comparatively cheap
- (EC)DH operation
  - Ephemeral key with fast EC curve
- Signature with long-lived key
  - May be expensive
  - Faster algorithms may be unavailable (customer-provided RSA key)
  - Long-lived secrets may have extra protections (RPC to remote key, HSM, etc)

Can we lower the signature costs?
Batch signatures

- Combine signing inputs into Merkle tree
- Sign the root once
- Ship Merkle tree paths to each client

\[
\text{HashLeaf}(msg) = \text{Hash}(0x00 \ || \ msg) \\
\text{HashNode}(left, right) = \text{Hash}(0x01 \ || \ left \ || \ right)
\]

opaque Node[Hash.length];

struct {
    uint32 index;
    Node path<Hash.length..2^16-1>;
    opaque root_signature<0..2^16-1>;
} BatchSignature;
Example

```
<table>
<thead>
<tr>
<th>Input 1</th>
<th>Hash</th>
<th>Blinder</th>
<th>Hash</th>
<th>Input 2</th>
<th>Hash</th>
<th>Blinder</th>
<th>Hash</th>
<th>Input 3</th>
<th>Blinder</th>
<th>Signature</th>
</tr>
</thead>
</table>
```
Verifying

- **Hash input**
- **Recompute root hash by hashing path nodes**
  - Index determines whether to hash on left or right
- **Verify signature against recomputed root**

\[
\text{HashLeaf}(\text{msg}) = \text{Hash}(0x00 \ || \ \text{msg}) \\
\text{HashNode}(\text{left}, \text{right}) = \text{Hash}(0x01 \ || \ \text{left} \ || \ \text{right})
\]

opaque Node[Hash.length];

struct {
    uint32 index;
    Node path<Hash.length..2^{16}-1>;
    opaque root_signature<0..2^{16}-1>;
} BatchSignature;
Advertising support

- New SignatureScheme code points specify hash and base algorithm
  - ecdsa_secp256r1_sha256_batch
  - ecdsa_secp384r1_sha384_batch
  - ecdsa_secp521r1_sha512_batch
  - ed25519_batch
  - ed448_batch
  - rsa_pss_pss_sha256_batch
  - rsa_pss_rsaes_sha256_batch
Amortize signing costs

- While signer is busy, batch up inputs for the next signature
- $N$ hashes multiplies signing capacity by $2^{N-1}$
  - 264 extra bytes in signature (using SHA-256) gives 128×
  - 360 extra bytes gives 1,024×
  - 680 extra bytes gives 1,048,576×...
- Signer and verifier TLS stacks must be modified
- Works with unmodified certificate and signing infrastructure
  - Only signing input changes
- Requires modified peers
  - Average load of existing deployments decreases if many peers support it
  - Under load, preferentially serve batchable peers as DoS mitigation
Details

- Domain separation
  - Signing inputs preserve input context string
  - Root is signed with distinct context string
- Blinding nodes avoid leaking information about tree siblings
  - Signing payloads are potentially confidential with ESNI
  - Costs one hash output in batch signature size
- Padding nodes come from other nodes in tree level
- Reveals some information about signer load
Questions?

Bonus slides

Gratuitous slide-based animation
Verifying signatures
Verifying signatures
Verifying signatures

Hash path node

Input 1
Hash
Blinder

Input 2
Hash
Blinder

Input 3
Hash
Blinder

Signature
Hash
Pad

Index = 2
Verifying signatures

Hash path node

Signature

Index = 2
Verifying signatures