The Picnic Digital Signature Algorithm

NIST First PQC Standardization Conference
April 2018

Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig and Greg Zaverucha
Overview

Security depends only on problems related to symmetric key primitives
  Unique design, conservative assumptions
  Secure hash function (ROM/QROM analysis implies all the usual properties: CR, PR, etc.)
  Secure block cipher (key recovery given a single plaintext/ciphertext pair)

The core of Picnic is an efficient zero knowledge proof for binary circuits
  Create a signature scheme using a non-interactive proof
  Use the Fiat-Shamir transform or Unruh transform

Performance characteristics
  Keys are small, signing and verification times are fast
  Signatures are relatively large
Picnic Signatures

Key Generation:
- Generate a random plaintext block \( p \)
- Generate a random secret key \( sk \)
- Compute \( C = \text{LowMC}(sk, p) \)
- Picnic public key is \( pk = (C, p) \), secret key is \( sk \)

Sign\((sk, pk, m)\):
- Prove knowledge of \( sk \) such that \( C = \text{LowMC}(sk, p) \)
- Message \( m \) and public key \( pk \) are bound to the proof when computing the challenge
- Picnic signature is the proof
Picnic Signatures

Key Generation:
- Generate a random plaintext block \( p \)
- Generate a random secret key \( sk \)
- Compute \( C = \text{LowMC}(sk, p) \) \(^\text{Must be hard to recover } sk\)
- Picnic public key is \( pk = (C, p) \), secret key is \( sk \)

Sign(\( sk, pk, m \)):
- Prove knowledge of \( sk \) such that \( C = \text{LowMC}(sk, p) \)
- Message \( m \) and public key \( pk \) are bound to the proof when computing the challenge
- Picnic signature is the proof \(^\text{Must be zero-knowledge}\)
Proof System

ZKBoo: zero knowledge proofs for statements about circuits.

Public circuit with AND and XOR gates

Prover knows $x_1 \ldots x_n$ such that the circuit evaluates to $y_1 \ldots y_m$

Proof system

Picnic uses ZKB++, a variant optimized for short proofs
Built with a hash function and KDF (both SHAKE)

Non-interactive proofs
  Fiat-Shamir transform gives ROM security
  Unruh’s transform gives QROM security, 1.6x larger signatures

Proof size depends on
  The security level; we use parallel repetition to achieve soundness
  The number of AND gates in the circuit
The LowMC Block Cipher

LowMC is a block cipher introduced by Albrecht et al. at Eurocrypt. Designed for nontraditional block cipher applications, like MPC and FHE.

Compared to more common primitives:
- About 7x fewer than AES, and 30x fewer than SHA-256
- Newer design, but we only need key recovery to be difficult

Highly parameterizable, some of our choices
- Tradeoff between AND and XOR gates: balance signature size and signing time
- Only one plaintext-ciphertext pair is revealed per key
- Keysize = blocksize (128, 192 and 256 bits)

Parameter Sets

Parameter sets for the three AES security levels, L1, L3, L5
  Each with different LowMC, SHAKE and # of parallel repetitions
  Fiat-Shamir (FS) and Unruh (UR) variants
## Performance: Key and Signature Size

### Signature and key sizes (bytes)

<table>
<thead>
<tr>
<th>Parameter Set</th>
<th>Public Key</th>
<th>Private Key</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Picnic-L1-FS</td>
<td>32</td>
<td>16</td>
<td>34,000</td>
</tr>
<tr>
<td>Picnic-L1-UR</td>
<td>32</td>
<td>16</td>
<td>53,929</td>
</tr>
<tr>
<td>Picnic-L3-FS</td>
<td>48</td>
<td>24</td>
<td>76,740</td>
</tr>
<tr>
<td>Picnic-L3-UR</td>
<td>48</td>
<td>24</td>
<td>121,813</td>
</tr>
<tr>
<td>Picnic-L5-FS</td>
<td>64</td>
<td>32</td>
<td>132,824</td>
</tr>
<tr>
<td>Picnic-L5-UR</td>
<td>64</td>
<td>32</td>
<td>209,474</td>
</tr>
</tbody>
</table>
Performance: Timings

Optimized Implementation (ms), Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz

<table>
<thead>
<tr>
<th>Parameter Set</th>
<th>Keygen</th>
<th>Sign</th>
<th>Verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>Picnic-L1-FS</td>
<td>0.00</td>
<td>1.95</td>
<td>1.36</td>
</tr>
<tr>
<td>Picnic-L1-UR</td>
<td>0.00</td>
<td>2.64</td>
<td>1.91</td>
</tr>
<tr>
<td>Picnic-L3-FS</td>
<td>0.01</td>
<td>6.61</td>
<td>4.63</td>
</tr>
<tr>
<td>Picnic-L3-UR</td>
<td>0.01</td>
<td>8.84</td>
<td>6.29</td>
</tr>
<tr>
<td>Picnic-L5-FS</td>
<td>0.02</td>
<td>14.71</td>
<td>10.64</td>
</tr>
<tr>
<td>Picnic-L5-UR</td>
<td>0.02</td>
<td>18.67</td>
<td>13.60</td>
</tr>
</tbody>
</table>
TLS Experiments

Are there challenges to using Picnic in TLS?
   We added Picnic to the Open Quantum Safe library (OQS), the OQS fork of OpenSSL and Apache web server

Experiment:
   Use Picnic-signed X.509 certificates certifying Picnic keys
   L1-FS parameter set
   Use Picnic certificates to authenticate TLS 1.2 connections
   Fetch HTML files

Performance, client-side latency:
   For 45B files: increase of 1.4x to 1.7x
   For 100KB files: increase of 1.1x to 1.3x

Challenges:
   TLS 1.2 has limit of $2^{16} - 1$ bytes/signature: too short for our higher security parameter sets

<table>
<thead>
<tr>
<th>Ciphersuite</th>
<th>Page Size</th>
<th>Mean fetch time (seconds)</th>
<th>Mean fetch time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEX, SIG</td>
<td></td>
<td>Slow network</td>
<td>Fast network</td>
</tr>
<tr>
<td>ECDHE-ECDSA</td>
<td>45B</td>
<td>0.470</td>
<td>0.299</td>
</tr>
<tr>
<td></td>
<td>(not PQ)</td>
<td>100K</td>
<td>1.226</td>
</tr>
<tr>
<td></td>
<td>LWEFRODO-RSA</td>
<td>45B</td>
<td>0.578</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100K</td>
<td>1.335</td>
</tr>
<tr>
<td></td>
<td>LWEFRODO-PICNIC</td>
<td>45B</td>
<td>0.984</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100K</td>
<td>1.733</td>
</tr>
<tr>
<td></td>
<td>SIDH-RSA</td>
<td>45B</td>
<td>0.655</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100K</td>
<td>1.370</td>
</tr>
<tr>
<td></td>
<td>SIDH-PICNIC</td>
<td>45B</td>
<td>1.084</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100K</td>
<td>1.738</td>
</tr>
</tbody>
</table>
HSM Experiments

What if a CA wants to protect Picnic signing keys in a hardware security module?

We experimented with the Utimaco SecurityServer Se50 LAN V4 Experiment:

- Implement Picnic key generation and signing in an HSM. Ported our reference implementation.
- Generate self-signed root cert using new Picnic key pair
- Receive certificate signing request for RSA key pair and issue X.509 certificate

Performance was acceptable and porting reference implementation was straightforward
Highlights of Picnic

Unique design
Conservative assumptions
Efficient and tested in real world protocols

More information: microsoft.github.io/Picnic/

Spec and design documents, research paper from CCS 2017
Talks and related work (RWC 2018 talk)
Link to OQS/OpenSSL, code from our HSM demo
Implementations