Picnic Post-Quantum Signatures from Zero Knowledge Proofs

MELISSA CHASE, MSR

THE PICNIC TEAM

DAVID DERLER
STEVEN GOLDFEDER
JONATHAN KATZ
VLAD KOLESNIKOV
CLAUDIO ORLANDI

SEBASTIAN RAMACHER
CHRISTIAN RECHBERGER
DANIEL SLAMANIG
XIAO WANG
GREG ZAVERUCHA
Post-quantum cryptography

A sufficiently powerful quantum computer could factor numbers and compute discrete logarithms
  ◦ Breaks essentially all standardized public key crypto
  ◦ E.g. RSA, DSA, ECDSA are insecure

Post-quantum cryptography: Design new schemes that
  ◦ can be run on classical machines
  ◦ Remain secure even if adversary has a quantum computer

Why now? Existing quantum computers only handle a few bits!
  ◦ Designing and deploying cryptography is slow!
    ◦ Propose assumptions and schemes
    ◦ Determine candidate parameters
    ◦ Analyze and attack schemes/assumptions
    ◦ Optimize surviving candidates
    ◦ Implement and deploy new schemes
    ◦ Deprecate old algorithms
Post-quantum cryptography

If quantum computers can break factoring and discrete log based crypto, is anything still hard?

Some proposed quantum hard problems:
- Lattice-based problems
- Supersingular isogeny Diffie–Hellman (SIDH)
- Code-based problems
- Multi-variate polynomial problems
- Symmetric key primitives (hash functions, block ciphers)
Post-quantum cryptography

ECDSA gives us small keys, small signatures and fast signing and verification
- But it is insecure against a quantum adversary

Are there any comparable post-quantum proposals?

<table>
<thead>
<tr>
<th></th>
<th>Public key size</th>
<th>Signature size</th>
<th>Signing time</th>
<th>Verification time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lattice (LWE)</td>
<td>Very large</td>
<td>Small</td>
<td>Fast</td>
<td>Fast</td>
</tr>
<tr>
<td>Lattice (Ring-LWE)</td>
<td>Large</td>
<td>Small</td>
<td>Fast</td>
<td>Fast</td>
</tr>
<tr>
<td>SIDH</td>
<td>Moderate</td>
<td>Large</td>
<td>Very slow</td>
<td>Very slow</td>
</tr>
<tr>
<td>Multivariate</td>
<td>Small</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Moderate</td>
</tr>
<tr>
<td>Hash (stateful)</td>
<td>Small</td>
<td>Small</td>
<td>Fast</td>
<td>Fast</td>
</tr>
<tr>
<td>Hash (stateless)</td>
<td>Small</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Fast</td>
</tr>
</tbody>
</table>
Picnic: Our post-quantum signature scheme

Based on symmetric primitives: a hash function + a block cipher
- Concretely we suggest: SHAKE and LowMC

Efficiency
- Small keys, moderate signature size, moderate signing and verification time

New approach
- Significant opportunity for further optimization
- Diversity of approaches for non-number-theoretic assumptions
Roadmap

Picnic: Basic approach
Picnic: Building blocks
Performance
Picnic 2.0
Conclusion
Picnic: basic approach

Signature from identification scheme (similar to DSA/ECDSA):

Public key = $F(sk)$

Signature = proof of knowledge of $sk$ (using message as nonce)
- *Proof must not leak $sk$, so we need a zero knowledge proof*

Challenge: we need a **hard to invert function $F$**, and a **zero knowledge proof system**
- Both need to be secure against quantum adversary
Picnic building blocks: ZKBoo

ZKBoo [GMO16]: zero knowledge proofs for statements about circuits.

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

Built on hash functions and PRNG

Cost depends on the number of AND gates in the circuit and security level
A toy example: Prover wants to prove knowledge of $a, b$ such that $a \oplus b = c$

Prover:

- **Step 1:** XOR secret share inputs
  - Pick random bits $a_1, a_2$ that XOR to $a$ and $b_1, b_2$ for $b$
  - $a_1 \oplus a_2 = a, b_1 \oplus b_2 = b$
- **Step 2:** compute output shares for $\oplus$ gate
  - $c_1 = a_1 \oplus b_1, c_2 = a_2 \oplus b_2$
- **Step 3:** commit to shares
  - Pick random strings $r_1, r_2$
  - Compute $h_1 = H(a_1, b_1, r_1), h_2 = H(a_2, b_2, r_2)$

Verifier:

- **Step 4:** Pick 1 or 2 at random
- **Step 5:**
  - Check that $c_1 \oplus c_2 = c$ and $a_1 \oplus b_1 = c_1$
  - Check that $h_1 = H(a_1, b_1, r_1)$

**Why is this convincing?**

- If Prover computes $h_1, h_2$ using $a_1, a_2, b_1, b_2$ such that $a_1 \oplus b_1 = c_1, a_2 \oplus b_2 = c_2,$ and $c_1 \oplus c_2 = c$ we’re done:
  - $(a_1 \oplus a_2) \oplus (b_1 \oplus b_2) = (a_1 \oplus b_1) \oplus (a_2 \oplus b_2) = c_1 \oplus c_2 = c$
- If not, Prover gets caught with probability at least 1/2
Picnic building blocks: ZKBoo (intuition)

A toy example: Prover wants to prove knowledge of \(a, b\) such that \(a \oplus b = c\)

**Prover:**
- Step 1: XOR secret share inputs
  - Pick random bits \(a_1, a_2\) that XOR to \(a\) and \(b_1, b_2\) for \(b\)
  - \(a_1 \oplus a_2 = a\), \(b_1 \oplus b_2 = b\)
- Step 2: compute output shares for \(\oplus\) gate
  - \(c_1 = a_1 \oplus b_1, c_2 = a_2 \oplus b_2\)
- Step 3: commit to shares
  - Pick random strings \(r_1, r_2\)
  - Compute \(h_1 = H(a_1, b_1, r_1), h_2 = H(a_2, b_2, r_2)\)

**Verifier:**
- Step 4: Pick 1 or 2 at random
- Step 5:
  - Check that \(c_1 \oplus c_2 = c\) and \(a_1 \oplus b_1 = c_1\)
  - Check that \(h_1 = H(a_1, b_1, r_1)\)

*Why does this hide \(a, b\)?*
- Verifier gets to see:
  - \(a_1, b_1\) reveals no information about \(a, b\)
  - \(c_1 = a_1 \oplus b_1, c_2 = c \oplus c_1\)
  - \(h_2\): hash of randomized inputs

Obviously trivial: just a toy example!
Picnic building blocks: ZKBoo (intuition)

Decrease cheating probability
- Run $t$ copies of proof with fresh randomness, verifier picks a challenge for each
- Probability of cheating decreases exponentially. $(1/3^t)$

Eliminate interaction
- Fiat-Shamir: Choose challenge by hashing $(c_1, c_2, h_1, h_2)$ from all copies.
- If $1/3^t$ is small enough, cheating prover can try hashing many sets of messages, will never find one he can correctly respond to
- Also include signature message in the hash.

What if we want a circuit with
- ANDs
- More gates?
Picnic building blocks: ZKBoo

Foundation for ZKBoo: MPC-in-the-head [IKOS07]

- Approach for constructing ZK proofs from Multi Party Computation
- Multi Party Computation
  - N parties with private input $x_i$
  - Want to compute $f(x_1, ..., x_n)$
  - Even if $n - 1$ parties combine their information, they learn nothing else
- To prove “I know $x$ such that $F(x) = 1$”
  - Choose random values such that $x_1 \oplus \cdots \oplus x_n = x$
  - Imagine N parties each with input $x_i$.
  - Internally run MPC between them to compute $F(x_1 \oplus \cdots \oplus x_n)$.
  - Record all messages sent and received.
  - For each party commit to “view”:
    - input $x_i$, randomness, messages sent, messages received
    - Verifier chooses $i$
    - Prover reveals views for all parties except $i$
Foundation for ZKBoo: MPC-in-the-head [IKOS07]

- Approach for constructing ZK proofs from Multi Party Computation
- Multi Party Computation
  - N parties with private input \( x_i \)
  - Want to compute \( f(x_1, \ldots, x_n) \)
  - Even if \( n - 1 \) parties combine their information, they learn nothing else
- To prove “I know \( x \) such that \( F(x) = 1 \)”
  - Choose random values such that \( x_1 \oplus \cdots \oplus x_n = x \)
  - Imagine N parties each with input \( x_i \).
  - Internally run MPC between them to compute \( F(x_1 \oplus \cdots \oplus x_n) \).
  - Record all messages sent and received.
  - For each party commit to “view”:
    - input \( x_i \), randomness, messages sent, messages received
    - Verifier chooses \( i \)
    - Prover reveals views for all parties except \( i \)

Zero Knowledge
Verifier gets to see views of all parties except \( i \)
MPC guarantees it learns nothing besides \( F(x) \)
Picnic building blocks: ZKBoo

Foundation for ZKBoo: MPC-in-the-head [IKOS07]

- Approach for constructing ZK proofs from Multi Party Computation
- Multi Party Computation
  - N parties with private input $x_i$
  - Want to compute $f(x_1, ..., x_n)$
  - Even if $n - 1$ parties combine their information, they learn nothing else
- To prove “I know $x$ such that $F(x) = y$”
  - Choose random values such that $x_1 \oplus \cdots \oplus x_n = x$
  - Imagine N parties each with input $x_i$.
  - Internally run MPC between them to compute $F(x_1 \oplus \cdots \oplus x_n)$.
  - Record all messages sent and received.
  - For each party commit to “view”:
    - input $x_i$, randomness, messages sent, messages received
    - Verifier chooses $i$
    - Prover reveals views for all parties except $i$

Soundness

If all parties behave correctly, output will be $F(x_1 \oplus \cdots \oplus x_n)$
If $F(x) \neq y$ either
  - A party misbehaved
  - Views are inconsistent
  - Catch this with probability $p$
  - Repeat many times
Picnic building blocks: ZKBoo (intuition)

A toy example: Prover wants to prove knowledge of \(a, b\) such that \(a \oplus b = c\)

Prover:
- Step 1: XOR secret share inputs
  - Pick random bits \(a_1, a_2\) that XOR to \(a\) and \(b_1, b_2\) for \(b\)
  - \(a_1 \oplus a_2 = a\), \(b_1 \oplus b_2 = b\)
- Step 2: compute output shares for \(\oplus\) gate
  - \(c_1 = a_1 \oplus b_1\), \(c_2 = a_2 \oplus b_2\)
- Step 3: commit to shares
  - Pick random strings \(r_1, r_2\)
  - Compute \(h_1 = H(a_1, b_1, r_1)\), \(h_2 = H(a_2, b_2, r_2)\)

Verifier:
- Step 4: Pick 1 or 2 at random
- Step 5:
  - Check that \(c_1 \oplus c_2 = c\) and \(a_1 \oplus b_1 = c_1\)
  - Check that \(h_1 = H(a_1, b_1, r_1)\)
Picnic building blocks: ZKBoo (intuition)

A toy example: Prover wants to prove knowledge of $a, b$ such that $a \oplus b = c$

Prover:
- Step 1: XOR secret share inputs
  - Pick random bits $a_1, a_2$ that XOR to $a$ and $b_1, b_2$ for $b$
  - $a_1 \oplus a_2 = a$, $b_1 \oplus b_2 = b$
- Step 2: compute output shares for $\oplus$ gate
  - $c_1 = a_1 \oplus b_1$, $c_2 = a_2 \oplus b_2$
- Step 3: commit to shares
  - Pick random strings $r_1, r_2$
  - Compute $h_1 = H(a_1, b_1, r_1)$, $h_2 = H(a_2, b_2, r_2)$

Verifier:
- Step 4: Pick 1 or 2 at random
- Step 5:
  - Check that $c_1 \oplus c_2 = c$
  - Check that $h_1 = H(a_1, b_1, r_1)$
  - $a_1 \oplus b_1 = c_1$

Inputs $P_1: a_1, b_1$
$P_2: a_2, b_2$

MPC

Commit to views

Obviously trivial: just a toy example!
Picnic building blocks: ZKBoo

ZKBoo makes MPC-in-the-head practical

Minimize communication
- Fix 3 parties (in general communication is $n^2$)
- $P_i$ only receives messages from $P_{i+1}$

Observation:
- we said V checks that messages sent = messages received
- Instead, could check any function on views of $P_i$ and $P_{i+1}$ up to that point
- Message received can be function of current state of $P_{i+1}$ and previous state of $P_i$
- Optimize MPC in this model
Picnic building blocks: ZKB++

ZKB++: Optimized ZKBoo [CDGORRSZ17]
- Identify places where e.g. values can safely be recomputed by the verifier, or represented by a short seed
- Reduces signature size by more than factor of 2
- Security analysis in random oracle model

Variant based on Unruh’s transform [Unruh 15]
- Security analysis in quantum random oracle model
- Our optimized implementation increases signature size by 1.6x over basic ZKBoo++
  - Still shorter than original ZKBoo
Picnic: basic approach

Signature from identification scheme (similar to DSA/ECDSA):

Public key = F(sk)

Signature = proof of knowledge of sk (using message as nonce)
  *Proof must not leak sk, so we need a zero knowledge proof

Challenge: we need a hard to invert function F, and a zero knowledge proof system
  *Both need to be secure against quantum adversary
Picnic building blocks: choosing F

ZKBoo++: Prover/signer can prove he knows sk such that the circuit F evaluates to pk

What F should we choose?
- F must be hard to invert
- Proof/signature size depends on number of AND gates in circuit for F

We can use a block cipher as well:
- PK: $R, Enc_{sk}(R)$

<table>
<thead>
<tr>
<th></th>
<th>Sec level</th>
<th>AND gates</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>128</td>
<td>5440</td>
</tr>
<tr>
<td>SHA-2</td>
<td>256</td>
<td>&gt; 25000</td>
</tr>
<tr>
<td>SHA-3</td>
<td>256</td>
<td>38400</td>
</tr>
<tr>
<td>Noekeon</td>
<td>128</td>
<td>2048</td>
</tr>
<tr>
<td>Trivium</td>
<td>80</td>
<td>1536</td>
</tr>
<tr>
<td>PRINCE</td>
<td></td>
<td>1920</td>
</tr>
<tr>
<td>Fantomas</td>
<td>128</td>
<td>2112</td>
</tr>
<tr>
<td>Kreyvium</td>
<td>128</td>
<td>1536</td>
</tr>
<tr>
<td>FLIP</td>
<td>128</td>
<td>&gt; 100000</td>
</tr>
<tr>
<td>MIMC</td>
<td>128</td>
<td>10337</td>
</tr>
<tr>
<td>MIMC</td>
<td>256</td>
<td>41349</td>
</tr>
<tr>
<td>LowMC</td>
<td>128</td>
<td>&lt; 800</td>
</tr>
<tr>
<td>LowMC</td>
<td>256</td>
<td>&lt; 1400</td>
</tr>
</tbody>
</table>
Picnic building blocks: LowMC

New block cipher introduced by [ARSTZ15]
Substitution-permutation-network design

Parameterizable:
- allows for minimizing AND gates or AND depth
- Tradeoffs between #s of AND gates and XOR gates
- Variable key and block sizes
- Allows for different security levels and # of plaintext ciphertext pairs the attacker will be given

For our application
- Few (but not minimal) AND gates: balance signature size and signing time
Picnic building blocks: LowMC

New block cipher introduced by [ARSTZ15]

Substitution-permutation-network design

Security for our application

- Several different security levels based on desired security for signature
- Only 1 plaintext-ciphertext pair is revealed
- Keysize = blocksize
- Attackers goal is key recovery*
- Weaker than traditional indistinguishable security with many plaintext-ciphertext pairs
- Our parameters may be conservative
Roadmap

Picnic: Basic approach
Picnic: Building blocks
Performance
Picnic 2.0
Conclusion
Picnic 1.0 Performance

3 parameter levels
- L1: 128 bits classical, 64 bits quantum
- L3: 192 bits classical, 96 bits quantum
- L5: 256 bits classical, 128 bits quantum

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>34000</td>
</tr>
<tr>
<td>Picnic-L1-UR</td>
<td>32</td>
<td>16</td>
<td>53929</td>
</tr>
<tr>
<td>Picnic-L3-FS</td>
<td>48</td>
<td>24</td>
<td>76740</td>
</tr>
<tr>
<td>Picnic-L3-UR</td>
<td>48</td>
<td>24</td>
<td>121813</td>
</tr>
<tr>
<td>Picnic-L5-FS</td>
<td>64</td>
<td>32</td>
<td>132824</td>
</tr>
<tr>
<td>Picnic-L5-UR</td>
<td>64</td>
<td>32</td>
<td>209474</td>
</tr>
</tbody>
</table>
Picnic 1.0 Performance

3 parameter levels
- L1: 128 bits classical, 64 bits quantum
- L3: 192 bits classical, 96 bits quantum
- L5: 256 bits classical, 128 bits quantum

Optimized constant- time 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>5.41</td>
<td>3.70</td>
</tr>
<tr>
<td>Picnic-L1-UR</td>
<td>0.00</td>
<td>6.12</td>
<td>4.24</td>
</tr>
<tr>
<td>Picnic-L3-FS</td>
<td>0.01</td>
<td>17.07</td>
<td>11.61</td>
</tr>
<tr>
<td>Picnic-L3-UR</td>
<td>0.01</td>
<td>19.01</td>
<td>13.08</td>
</tr>
<tr>
<td>Picnic-L5-FS</td>
<td>0.02</td>
<td>36.47</td>
<td>24.70</td>
</tr>
<tr>
<td>Picnic-L5-UR</td>
<td>0.02</td>
<td>39.21</td>
<td>26.90</td>
</tr>
</tbody>
</table>
Experiments

TLS integration:
◦ What if we want to use Picnic for TLS authentication?
◦ Added Picnic to the *Open Quantum Safe library* (OQS), the OQS fork of OpenSSL and Apache web server
◦ Use Picnic to create X509 certificates certifying Picnic public keys
◦ Use resulting certificates to establish TLS 1.2 connections

HSM implementation:
◦ What if a CA wants to store Picnic signing keys in an HSM?
◦ Experimented with the Utimaco SecurityServer Se50 LAN V4
◦ Implemented Picnic key generation and signing in an HSM.

See Picnic design document For details
Roadmap

Picnic: Basic approach
Picnic: Building blocks
Performance
Picnic 2.0
Conclusion
Picnic 2.0 building blocks: [KKW18 proofs]

[KKW18] introduced an improved proof system

- ZKBoo soundness for 1-round: 1/3 because we fully check 1 party of 3.
- What if we could fully check n-1 out of n?
- We could run fewer parallel repetitions!

Need to guarantee:

- We can check each opened parties
- We can increase the number of parties without increasing communication
- We can regenerate n-1 views from little information

Use MPC in the preprocessing model

- Commit to preprocessing, and use cut-and-choose to check
- Protocol just has 1 broadcast bit/AND gate from each party
- Just need to send broadcast bits from unopened party

- Picnic 2.0 uses 64 parties, checks 63.
- Improves signature size by almost a factor of 3

Need to make sure this communication is small: clever tree data structure
Picnic 2.0 building blocks: [KKW18] proofs

Signatures sizes for Picnic with [KKW18] proofs

<table>
<thead>
<tr>
<th>Security Level</th>
<th>Previous Size (bytes)</th>
<th>New Size (bytes)</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1-FS</td>
<td>32,838</td>
<td>12,359</td>
<td>2.7x</td>
</tr>
<tr>
<td>L3-FS</td>
<td>74,134</td>
<td>27,172</td>
<td>2.7x</td>
</tr>
<tr>
<td>L5-FS</td>
<td>128,176</td>
<td>46,282</td>
<td>2.8x</td>
</tr>
</tbody>
</table>

- Sizes given are the average case sizes
- The implementation from ePrint 2018/475 suggests it’s possible to have the same performance
- The parameters using the Unruh transform are unchanged
Picnic 2.0 building blocks: Optimized LowMC

[KPPRR17, D18]

LowMC was designed to support arbitrary parameter sets (key size, block size, # rounds, # s-boxes)

This work optimizes for the Picnic parameters:

- LowMC is an SPN cipher
- rounds have a s-box (nonlinear) part and a linear part
- Picnic: small nonlinear part and a large linear part
- Reorder operations to combine some linear steps

### Picnic 2.0 building blocks: Optimized LowMC

#### Running times with optimized LowMC circuit

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Sign (ms, old)</th>
<th>Sign (ms, new)</th>
<th>Verify (ms, old)</th>
<th>Verify (new)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1-FS</td>
<td>5.41</td>
<td>2.37</td>
<td>2.28x</td>
<td>1.89</td>
</tr>
<tr>
<td>L1-UR</td>
<td>6.12</td>
<td>3.08</td>
<td>1.99x</td>
<td>2.47</td>
</tr>
<tr>
<td>L3-FS</td>
<td>17.07</td>
<td>5.50</td>
<td>3.10x</td>
<td>4.49</td>
</tr>
<tr>
<td>L3-UR</td>
<td>19.01</td>
<td>7.43</td>
<td>2.56x</td>
<td>5.98</td>
</tr>
<tr>
<td>L5-FS</td>
<td>36.47</td>
<td>9.74</td>
<td>3.74x</td>
<td>8.05</td>
</tr>
<tr>
<td>L5-UR</td>
<td>39.21</td>
<td>12.58</td>
<td>3.12x</td>
<td>10.25</td>
</tr>
</tbody>
</table>

- This compares versions of the constant time implementations
- Times are milliseconds on an Intel Core i7-4790 CPU @ 3.60GHz
- Does not include [KKW18] proofs
Picnic 2.0 building blocks: Optimized LowMC

Running times with optimized LowMC circuit

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Sign (ms, old)</th>
<th>Sign (ms, new)</th>
<th>Verify (ms, old)</th>
<th>Verify (new)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1-FS</td>
<td>5.41</td>
<td>2.37</td>
<td>2.28x</td>
<td>1.89</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.28x</td>
<td>1.96x</td>
</tr>
<tr>
<td>L1-UR</td>
<td>6.12</td>
<td>3.08</td>
<td>1.99x</td>
<td>2.47</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.99x</td>
<td>1.72x</td>
</tr>
<tr>
<td>L3-FS</td>
<td>17.07</td>
<td>5.50</td>
<td>3.10x</td>
<td>4.49</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.10x</td>
<td>2.59x</td>
</tr>
<tr>
<td>L3-UR</td>
<td>19.01</td>
<td>7.43</td>
<td>2.56x</td>
<td>5.98</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.56x</td>
<td>2.19x</td>
</tr>
<tr>
<td>L5-FS</td>
<td>36.47</td>
<td>9.74</td>
<td>3.74x</td>
<td>8.05</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.74x</td>
<td>3.07x</td>
</tr>
<tr>
<td>L5-UR</td>
<td>39.21</td>
<td>12.58</td>
<td>3.12x</td>
<td>10.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.12x</td>
<td>2.62x</td>
</tr>
</tbody>
</table>

- This compares versions of the constant time implementations
- Times are milliseconds on an Intel Core i7-4790 CPU @ 3.60GHz
- Does not include [KKW18] proofs
Conclusions

New postquantum signature proposal
- Based on symmetric primitives: a hash function + hard-to-invert function (concretely SHAKE and LowMC)
- Small keys, moderate signature size, moderate signing and verification time
- Modular construction from ZK proofs

Lots of opportunity for further optimization
- Further optimize current proof system?
- Further design of MPC protocols for this setting?
- Propose new proof system (sublinear proofs?)
  - Ligero [AHIV17] is work in this direction
- Further optimizations for LowMC?
- Security analysis of LowMC for our parameters
- Or alternative functions \( F \)?

More info, see https://microsoft.github.io/Picnic/. Picnic 2.0 parameters and code available later this week.