Equihash: Asymmetric Proof-of-Work based on the Generalized Birthday Problem

Alex Biryukov    Dmitry Khovratovich

University of Luxembourg

February 22nd, 2016
Proof of Work in cryptocurrencies

PoW – certificate of certain amount of work. In cryptocurrencies:

- Verifier – cryptocurrency users;
- Prover – cryptocurrency miner.
Proof of Work as a client puzzle

In TLS client puzzles:

- **Verifier** – server that establishes a secure connection;
- **Prover** – client that may want to DoS the server with signature computation.
Asymmetric verification

Clearly, the proof search
Clearly, the proof search must be more expensive than verification.
HashCash/Bitcoin Proof-of-Work with hash function $H$:

$S$ – proof, if $H(S) = \underbrace{00\ldots0}_q$.

$2^q$ calls to $H$ for prover, 1 call for verifier.
But here come ASICs..

Regular cryptographic hash $H$ is 30,000 less expensive on ASIC due to small custom chip.
Since 2003, *memory-intensive* computations have been proposed.

Computing with a lot of memory would require a very large and expensive chip.

With large memory on-chip, the ASIC advantage vanishes.
Hash function with two iterations over memory of size $N$.

- $V_i = F(V_{i-1})$;
- $V_N' = V_N$;
- $V'_i = F(V'_{i+1} || V_i)$.
Compute the hash using $\frac{N}{m} + m$ memory units and $3N$ calls to $F$ (instead of $2N$):

- Store every $m$-th block;
- When entering a new interval, precompute its $m$ inputs.

Optimal point is $m = \sqrt{N}$.
Approach 2. Argon2

Memory-hard hashing function, that won Password Hashing Competition in 2015:

- Simple randomized-graph design with high-penalty tradeoffs.
Approach 2. Argon2

Memory-hard hashing function, that won Password Hashing Competition in 2015:

- Simple randomized-graph design with high-penalty tradeoffs.
- However, no easy verification.
Approach 3. Collision search

1. Verifier sends seed $S$;
2. Prover generates $2^k$ 2k-bit hashes $H(S\|1), H(S\|2), \ldots, H(S\|2^k)$.
3. Prover shows a collision $H(S\|i) = H(S\|j)$. Short and efficient.
Approach 3. Collision search

1. Verifier sends seed $S$;
2. Prover generates $2^k$ 2k-bit hashes $H(S||1)$, $H(S||2)$, ..., $H(S||2^k)$.
3. Prover shows a collision $H(S||i) = H(S||j)$. Short and efficient.

Problem: the $\rho$-based collision search finds collisions in the same $2^k$ time but no memory.
Original: given $2^k$ lists $L_j$ of $n$-bit strings $\{X_i\}$, find distinct $\{X_{i_j} \in L_j\}$ such that

$$X_{i_1} \oplus X_{i_2} \oplus \cdots \oplus X_{i_{2^k}} = 0.$$
Solution is found by iterative sorting
\(O(2^{\frac{n}{k+1}})\) time and memory

- Sort by first \(\frac{n}{k+1}\) bits;
- Store XOR of collisions;
- Repeat for next \(\frac{n}{k+1}\) bits, etc.
\(O(2^{\frac{n}{k+1}})\) time and memory

- Sort by first \(\frac{n}{k+1}\) bits;
- Store XOR of collisions;
- Repeat for next \(\frac{n}{k+1}\) bits, etc.

Problem: not amortization-free: it is easy to modify the algorithm to get many solutions quickly:
  - Collide on other bits;
  - Not collision but XOR to some constant.

After all, \(qM\) memory yields \(q^{k+1}\) solutions in time \(qT\).
Interestingly, the solution reveals how it was found:

\[ H(x_1) \oplus H(x_2) \oplus H(x_3) \oplus H(x_4) \cdots \oplus H(x_{2^k}) = 0. \]

\[ \text{equal in } \frac{n}{k+1} \text{ bits} \]

\[ \text{equal in } \frac{n}{k+1} \text{ bits} \]

\[ \text{equal in } \frac{2n}{k+1} \text{ bits} \]

We then strongly require such pattern and disallow other solutions.

Amortization is impossible then.
To avoid centralization, there must be always a chance to find a solution (Poisson process).

\[
P : \mathcal{R} \times \mathcal{I} \times \mathcal{S} \rightarrow \{\text{true, false}\}.
\]

How to increase expected solving time and make the probability non-zero at the beginning?
Difficulty filter: $S$ is valid if $P(R, I, S) = true$ and $H(S)$ has $q$ leading zeros.

Problem composition takes the best properties from each component.
**Equihash**: given seed $I$, find $V$ and $\{x_j\}$ such that

$$H(I \| V \| x_1) \oplus H(I \| V \| x_2) \oplus \cdots \oplus H(I \| V \| x_{2^k}) = 0. \quad (1)$$

$$H(I \| V \| x_1 \| x_2 \| \cdots \| x_{2^k}) = 00 \ldots 0^{q \text{ zeroes}}. \quad (2)$$

$$H(x_1) \oplus H(x_2) \oplus H(x_3) \oplus H(x_4) \cdots \oplus H(x_{2^k}) = 0. \quad (3)$$
Tradeoff for Equihash

Time penalty for reducing memory by the factor of $q$:

$$C_2(q) \approx 2^k q^{k/2} k^{k/2-1} = O(q^{k/2}).$$

Tunable steepness.

Memoryless computation: run recursive memoryless collision search for expanding functions (a bit worse):

$$2^{n/2} + 2^k + \frac{n}{k+1}.$$
Using $p$ processors, we can get $p$-factor speed-up in time.

On GPU and FPGA this leads to increased memory bandwidth (factor $p$), which becomes bottleneck.
The only possible solution is mesh-based sorting with one core per memory block on custom ASIC, but this is expensive (10x larger chip).
<table>
<thead>
<tr>
<th>$n$</th>
<th>$k$</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Memory-full</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Peak memory</td>
</tr>
<tr>
<td>96</td>
<td>5</td>
<td>2.5 MB</td>
</tr>
<tr>
<td>128</td>
<td>7</td>
<td>8.5 MB</td>
</tr>
<tr>
<td>160</td>
<td>9</td>
<td>32.5 MB</td>
</tr>
<tr>
<td>176</td>
<td>10</td>
<td>64.5 MB</td>
</tr>
<tr>
<td>192</td>
<td>11</td>
<td>128.5 MB</td>
</tr>
<tr>
<td>96</td>
<td>3</td>
<td>320 MB</td>
</tr>
<tr>
<td>144</td>
<td>5</td>
<td>704 MB</td>
</tr>
<tr>
<td>192</td>
<td>7</td>
<td>2.2 GB</td>
</tr>
<tr>
<td>240</td>
<td>9</td>
<td>8.3 GB</td>
</tr>
<tr>
<td>96</td>
<td>2</td>
<td>82 GB</td>
</tr>
<tr>
<td>288</td>
<td>8</td>
<td>11 TB</td>
</tr>
</tbody>
</table>
Questions?