Client puzzles for TLS

Alex Biryukov, Dmitry Khovratovich

University of Luxembourg

Also Erik Nygren, Samuel Erb (Akamai)

July 19th, 2016
- TLS server performs expensive cryptographic operations at the first stage of handshake;
- A malicious client can initiate a connection and drop it after the initial message;
- Existing countermeasure: rate limit.

Problem of rate limiting: does not distinguish good from bad clients behind NAT or Proxy; well-behaved clients are more penalized than bad ones.
Complementary idea:
- Insert a puzzle into HelloRetryRequest;
- Client solves a puzzle and returns a solution;
- Verification should be fast.

Minor problem: clients with limited resources (e.g., IoT) will suffer. Clients not supporting puzzles can be rate limited.
The only required extension is multi-purpose:

- Client indicates of supported puzzle types;
- Server sends puzzle type and input;
- Client sends a solution (shorter than 64 KB).

\[\begin{array}{l}
\text{Client} \\
\text{ClientHello} \\
\quad + \text{ClientPuzzleExtension} \\
\quad + \text{ClientKeyShare} \\
\text{ClientHello} \\
\quad + \text{ClientPuzzleExtension} \\
\quad + \text{ClientKeyShare} \\
\text{Server} \\
\text{HelloRetryRequest} \\
\quad + \text{ClientPuzzleExtension} \\
\text{ServerHello} \\
\quad \text{ServerKeyShare} \\
\quad \{\text{EncryptedExtensions}\} \\
\end{array}\]
Types of puzzles

1. Cookie: client just echoes back the token;
2. SHA-256: server sends $S$, client finds $N$ such that $\text{SHA-256}(N, S)$ has certain number of initial zero bits;
3. SHA-512: similar.

SHA puzzles are well known from the Bitcoin cryptocurrency. Equihash was developed by Biryukov and Khovratovich [NDSS’2016] and is employed as a Proof-of-Work in the anonymity-enhanced currency Zcash.
Brute-force attacks are most efficient on custom hardware: multiple computing cores on large ASICs:
- $2^{32}$ hashes/joule on ASIC;
- $2^{17}$ hashes/joule on laptop.

ASICs have high entry costs, but FPGA and GPU are employed too. *Memory-intensive* computations have been as remedy, as computing with a lot of memory would require a very large and expensive chip, the ASIC advantage vanishes.
Given seed $I$, find $V$ and $\{x_j\}$ such that

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

where $H$ is Blake2b, and

$$H(I \parallel V \parallel x_1 \parallel x_2 \parallel \cdots \parallel x_{2^k}) = 00\ldots0*\cdots* \text{q zeroes}.$$

Solved by Wagner’s algorithm. Verification: $2^k$ hashes.
$O(2^{\frac{n}{k+1}})$ time and memory

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

Memory reduction by $q$ gives $O(q^{k/2})$ time penalty.

Public implementation available: https://github.com/khovratovich/equihash
## Suggested parameters

<table>
<thead>
<tr>
<th>$n$</th>
<th>$k$</th>
<th>Complexity</th>
<th>Time</th>
<th>Solution size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Peak memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>5</td>
<td>2.5 MB</td>
<td>0.25 sec</td>
<td>88 B</td>
</tr>
<tr>
<td>102</td>
<td>5</td>
<td>5 MB</td>
<td>&lt; 0.5 sec</td>
<td>92 B</td>
</tr>
<tr>
<td>114</td>
<td>5</td>
<td>20 MB</td>
<td>&lt; 2 sec</td>
<td>100 B</td>
</tr>
<tr>
<td>70</td>
<td>4</td>
<td>2 MB</td>
<td>0.1 sec</td>
<td>54 B</td>
</tr>
<tr>
<td>80</td>
<td>4</td>
<td>8 MB</td>
<td>0.2 sec</td>
<td>58 B</td>
</tr>
<tr>
<td>90</td>
<td>4</td>
<td>32 MB</td>
<td>1 sec</td>
<td>62 B</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>128 MB</td>
<td>4 sec</td>
<td>66 B</td>
</tr>
<tr>
<td>96</td>
<td>3</td>
<td>320 MB</td>
<td>10 sec</td>
<td>45 B</td>
</tr>
<tr>
<td>144</td>
<td>5</td>
<td>704 MB</td>
<td>15 sec</td>
<td>120 B</td>
</tr>
</tbody>
</table>

Time on single-thread 2.1 GHz CPU.