Faster Implementation of Modular Exponentiation in JavaScript

Francisco Corella
fcorella@pomcor.com

Karen Lewison
kplewison@pomcor.com
Context

• This work is part of an effort to develop a cryptographic authentication toolkit for developers of web applications

• Outline
  – Cryptographic authentication
  – Modular Exponentiation
  – JavaScript
Cryptographic Authentication

• A prover can authenticate to a verifier by proving knowledge of a private key
• The private key may pertain to any kind of public key cryptosystem:
  – Encryption
  – Key exchange
  – Digital signature
• Digital signature is not objected to by any governments, encryption and key exchange may be subject to export controls
Cryptographic Authentication by Digital Signature

• Verifier generates random nonce
• A challenge is constructed from material including the verifier’s nonce
• The prover signs the challenge with its private key
Digital Signature Cryptosystems

• RSA
  – 1977
  – Dual-purpose: encryption, signature

• DSA
  – Designed by the NSA to provide signature but no encryption

• ECDSA
  – Elliptic curve version of DSA
Elliptic curve vs. classical cryptography

- Classical crypto (RSA, DSA, DH) relies on the difficulty of factoring (RSA) or computing discrete modular logarithms (DSA, DH)
- ECC (ECDSA, ECDH, EC version of El Gamal) relies on the difficulty of computing discrete “logarithms” in the group of points of an elliptic curve
- ECC requires shorter keys (because it is not vulnerable to index-calculus attacks) and therefore is faster
- But ECC has a trust problem after the Snowden revelations
- We plan to provide both classical and elliptic curve crypto in our toolkit
DSA Is Important!

• DSA is the only cryptosystem that nobody objects to:
  – It is not under suspicion of hiding a government back door, even though it was designed by the NSA
  – It is not objected to by governments because it does not provide encryption or key exchange
Modular Exponentiation

- Modular exponentiation is what determines the performance of classical crypto algorithms
- RSA requires one modular exponentiation for signing, and one with short exponent for verifying
- RSA with CRT requires two modular exponentiations for signing, with half-size moduli
- DSA requires one modular exponentiation for signing, two for verifying
- DH requires one modular exponentiation by each key-exchange participant
- ECC uses scalar multiplication of curve points instead of modular exponentiation
Techniques for Implementing Modular Exponentiation

\[ y = g^x \mod m \]

- \( g^x \) would not fit in any storage
- Repeated multiply-and-reduce would take forever
- Square-and-multiply with reduce-as-you-go takes too long if reduction uses division
- Montgomery reduction avoids division
- Sliding-window exponentiation improves on square-and-multiply
- Karatsuba multiplication is asymptotically faster than long multiplication, should help for large moduli
Karatsuba Multiplication

• Recursive multiplication with 3 recursive calls instead of 4:
  \[x = x_1 b + x_0\]
  \[y = y_1 b + y_0\]
  \[xy = (b^2 + b)x_1 y_1 - b(x_1 - x_0)(y_1 - y_0) + (b + 1)x_0 y_0\]
• Karatsuba runs in time \(\Theta(n^{\log_2 3})\) instead of \(\Theta(n^2)\)
• “As a rule of thumb, Karatsuba is usually faster when the multiplicands are longer than 320-640 bits” (Wikipedia)
JavaScript

• The language of web applications
• Runs in the browser
• Originally intended for simple tasks in web pages
• Now a feature-rich language used on clients and servers
• Arguably the most important programming language today
JavaScript Not Designed for Cryptography

• Interpreted => slower than a compiled language like C
• Floating point but no integer arithmetic!!
• Options for implementing cryptographic authentication in a web application
  – Web Cryptography API spec of W3C?
  – Stanford JavaScript Crypto Library?
Web Cryptography API

• Appealing:
  – Crypto available to JavaScript apps
  – But implemented in C and/or assembly language and/or hardware

• But:
  – No DSA!
  – Asynchronous interface!!
  – Unfinished and in a state of flux
Stanford JavaScript Crypto Library (SJCL)

• Started as a fast implementation of AES in Javascript
• GitHub project has added public key cryptography
SJCL (continued)

- But
  - SJCL focuses on ECC
  - No RSA or classical DSA

- SJCL does provide classical DH, and implements modular exponentiation to that purpose

- SJCL features Montgomery reduction and sliding-window exponentiation

- But no Karatsuba multiplication
We Chose Another Option:

• Build our own big integer library and our own crypto algorithms
  – We were hoping to improve modular exponentiation performance by a factor of 2 using Karatsuba
  – Karatsuba did not help for < 4000 bit moduli
  – But we increased performance by a factor of 6 to 8 without Karatsuba!
Performance Results for DSA-DH bit lengths in Firefox on Mac with 1.7 GHz 64-bit Processor

<table>
<thead>
<tr>
<th>Security strength</th>
<th>112</th>
<th>128</th>
<th>192</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponent</td>
<td>224</td>
<td>256</td>
<td>384</td>
<td>512</td>
</tr>
<tr>
<td>Modulus</td>
<td>2048</td>
<td>3072</td>
<td>7680</td>
<td>15360</td>
</tr>
<tr>
<td>Stanford library</td>
<td>74 ms</td>
<td>180 ms</td>
<td>1549 ms</td>
<td>7908 ms</td>
</tr>
<tr>
<td>Pomcor library</td>
<td>10 ms</td>
<td>23 ms</td>
<td>199 ms</td>
<td>1050 ms</td>
</tr>
<tr>
<td>Performance improvement</td>
<td>7x faster</td>
<td>8x faster</td>
<td>8x faster</td>
<td>8x faster</td>
</tr>
</tbody>
</table>
Performance Results for RSA-with-CRT bit lengths in Firefox on Mac with 1.7 GHz 64-bit Processor

<table>
<thead>
<tr>
<th>Security strength</th>
<th>112</th>
<th>128</th>
<th>192</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponent</td>
<td>2048</td>
<td>3072</td>
<td>7680</td>
<td>15360</td>
</tr>
<tr>
<td>Modulus</td>
<td>1024</td>
<td>1536</td>
<td>3840</td>
<td>7680</td>
</tr>
<tr>
<td>Stanford library</td>
<td>148 ms</td>
<td>460 ms</td>
<td>6636 ms</td>
<td>50818 ms</td>
</tr>
<tr>
<td>Pomcor library</td>
<td>25 ms</td>
<td>75 ms</td>
<td>882 ms</td>
<td>7424 ms</td>
</tr>
<tr>
<td>Performance improvement</td>
<td>6x faster</td>
<td>6x faster</td>
<td>8x faster</td>
<td>7x faster</td>
</tr>
</tbody>
</table>
## Performance Results for DSA-DH bit lengths in Chrome on Phone with 2.3 GHz 32-bit Processor

<table>
<thead>
<tr>
<th>Security strength</th>
<th>Exponent</th>
<th>Modulus</th>
<th>Stanford library</th>
<th>Pomcor library</th>
<th>Performance improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>112</td>
<td>2048</td>
<td>315 ms</td>
<td>46 ms</td>
<td>7x faster</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>3072</td>
<td>742 ms</td>
<td>103 ms</td>
<td>7x faster</td>
</tr>
<tr>
<td></td>
<td>192</td>
<td>7680</td>
<td>6264 ms</td>
<td>644 ms using Karatsuba</td>
<td>10x faster</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>15360</td>
<td>34460 ms</td>
<td>3379 ms using Karatsuba</td>
<td>10x faster</td>
</tr>
</tbody>
</table>

- Exponent: 224, 256, 384, 512
- Modulus: 2048, 3072, 7680, 15360
- Stanford library: 315 ms, 742 ms, 6264 ms, 34460 ms
- Pomcor library: 46 ms, 103 ms, 644 ms using Karatsuba, 3379 ms using Karatsuba
- Performance improvement: 7x faster, 7x faster, 10x faster, 10x faster
Performance Results for RSA-with-CRT bit lengths in Chrome on Phone with 2.3 GHz 32-bit Processor

<table>
<thead>
<tr>
<th>Security strength</th>
<th>112</th>
<th>128</th>
<th>192</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponent</td>
<td>2048</td>
<td>3072</td>
<td>7680</td>
<td>15360</td>
</tr>
<tr>
<td>Modulus</td>
<td>1024</td>
<td>1536</td>
<td>3840</td>
<td>7680</td>
</tr>
<tr>
<td>Stanford library</td>
<td>710 ms</td>
<td>2108 ms</td>
<td>29300 ms</td>
<td>Not tested</td>
</tr>
<tr>
<td>Pomcor library</td>
<td>115 ms</td>
<td>263 ms</td>
<td>3263 ms</td>
<td>Not tested</td>
</tr>
<tr>
<td>Performance improvement</td>
<td>6x faster</td>
<td>7x faster</td>
<td>7x faster</td>
<td></td>
</tr>
</tbody>
</table>
Practical Benefits

• With our fast implementation of modular exponentiation on a laptop...
  – Crypto authentication using DSA becomes practical on a 64-bit laptop at all security strengths
  – And on a 32-bit phone at security levels 112, 128 and 256
Thank you for your attention

• These slides are available online at
  – http://pomcor.com/documents/ModExpInJS.pptx

• See also the blog post at

• Or contact us at
  – fcorella@pomcor.com  +1.619.770.6765
  – kplewison@pomcor.com  +1.669.300.4510