Fast Cryptography in Genus 2

Joppe W. Bos
Joint work with
Craig Costello, Huseyin Hisil, Kristin Lauter

Workshop on Elliptic Curve Cryptography 2013
Fast Cryptography in Genus 2

From a practical perspective!

Joppe W. Bos

Joint work with
Craig Costello, Huseyin Hisil, Kristin Lauter

Workshop on Elliptic Curve Cryptography 2013
Motivation - I

This is the ECC Workshop: we all like (elliptic) curves!

<table>
<thead>
<tr>
<th>Group</th>
<th>DH</th>
<th>ECDH</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{F}_{p_1}^*$, $\times$</td>
<td>$(\mathbb{F}_{p_1}, \times)$</td>
<td>$(\mathbb{E} \left( \mathbb{F}_{p_2} \right), +)$</td>
</tr>
<tr>
<td>Security level (bits)</td>
<td>$\log_2 p_1$</td>
<td>$\log_2 p_2$</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>128</td>
<td>3072</td>
<td>256</td>
</tr>
<tr>
<td>192</td>
<td>7680</td>
<td>384</td>
</tr>
<tr>
<td>256</td>
<td>15360</td>
<td>521</td>
</tr>
</tbody>
</table>

Source: NSA – The case for Elliptic Curve Cryptography
This is the ECC Workshop: we all like (elliptic) curves!

<table>
<thead>
<tr>
<th>Security level (bits)</th>
<th>log₂ ( p_1 )</th>
<th>log₂ ( p_2 )</th>
<th>Ratio DH cost : ECDH cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>3072</td>
<td>256</td>
<td>10:1</td>
</tr>
<tr>
<td>192</td>
<td>7680</td>
<td>384</td>
<td>32:1</td>
</tr>
<tr>
<td>256</td>
<td>15360</td>
<td>521</td>
<td>64:1</td>
</tr>
</tbody>
</table>

Why? Performance!

Source: NSA – The case for Elliptic Curve Cryptography
This is the ECC Workshop: we all like (elliptic) curves!

<table>
<thead>
<tr>
<th>Security level (bits)</th>
<th>$\log_2 p_1$</th>
<th>$\log_2 p_2$</th>
<th>Ratio DH cost : ECDH cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>3072</td>
<td>256</td>
<td>10:1</td>
</tr>
<tr>
<td>192</td>
<td>7680</td>
<td>384</td>
<td>32:1</td>
</tr>
<tr>
<td>256</td>
<td>15360</td>
<td>521</td>
<td>64:1</td>
</tr>
</tbody>
</table>

Why?: Performance!

Source: NSA – The case for Elliptic Curve Cryptography
Can we do better?

Reduce the **cost** of the group operation

- Use a different curve representation
- Use a different coordinate system
- E.g. **twisted Edwards curves** with **extended twisted Edwards coordinates**
- See the Explicit-Formulas Database
Can we do better?

Reduce the **cost** of the group operation

- Use a different curve representation
- Use a different coordinate system
- E.g. **twisted Edwards curves** with **extended twisted Edwards coordinates**
- See the Explicit-Formulas Database

Reduce the **number** of group operations

- Reduce the number of **point additions**
  e.g. use large window sizes
- Reduce the number of **point doublings**
  e.g. scalar decomposition
Can we do better?

Reduce the cost of the group operation

• Use a different curve representation
• Use a different coordinate system
• E.g. twisted Edwards curves with extended twisted Edwards coordinates
• See the Explicit-Formulas Database

Reduce the number of group operations

• Reduce the number of point additions
e.g. use large window sizes
• Reduce the number of point doublings
e.g. scalar decomposition

Other optimizations

• Montgomery ladder
• Fast finite field arithmetic:
  Curves over “special” primes
• Implementations using all the features of the architecture: e.g. special instructions, SIMD instructions
Can we do better?

Reduce the **cost** of the group operation

- Use a different curve representation
- Use a different coordinate system
- E.g. **twisted Edwards curves** with **extended twisted Edwards coordinates**
- See the Explicit-Formulas Database

Reduce the **number** of group operations

- Reduce the number of **point additions**
  e.g. use large window sizes
- Reduce the number of **point doublings**
  e.g. scalar decomposition

Other optimizations

- Montgomery ladder
- Fast finite field arithmetic:
  Curves over “special” primes
- Implementations using all the features
  of the architecture: e.g. special
  instructions, SIMD instructions

Change the setting!

- Consider genus 2
  - Different cost of the group operation
  - Different number of group operations
- Genus 2 equivalent of Montgomery ladder
  - Kummer surface
- GLV on genus 2 curves?
Both curves have around $p$ points over $\mathbb{F}_p$

Hasse-Weil: 

$$p + 1 - 2g\sqrt{p} \leq \#C(\mathbb{F}_p) \leq p + 1 + 2g\sqrt{p}$$
Can’t do “chord-and-tangent” in genus 2
Roughly speaking: group elements are pairs of points

\[ y^2 = x^3 + a_2 x^2 + a_1 x + a_0 \]

\[ y^2 = x^5 + a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0 \]

Why genus 2?

Roughly speaking: group elements are pairs of points

\[ \#E(F_p) \approx p \quad \text{versus} \quad \#\text{Jac}_C(F_p) \approx p^2 \]
Wasn’t this considered before?

2006: D. J. Bernstein: *Elliptic vs. hyperelliptic*, ECC Workshop

“Can we obtain higher speeds at comparable security levels using genus-2 hyperelliptic curves?”

Unfortunately:

“genus-2 point counting is too slow to reach 256 bits”

No point counting → no cryptographic genus 2 curves
Wasn’t this considered before?

2006: D. J. Bernstein: Elliptic vs. hyperelliptic, ECC Workshop
“Can we obtain higher speeds at comparable security levels using genus-2 hyperelliptic curves?”
Unfortunately:
“genus-2 point counting is too slow to reach 256 bits”
No point counting → no cryptographic genus 2 curves

Fortunately, there has been significant progress
2011: Gaudry-Kohel-Smith: Counting points on genus 2 curves with real multiplication, Asiacrypt
After seven years Genus 2 is ready to rumble!

Practical performance comparison
Genus 1 versus Genus 2

- 128-bit security level
- High-end 64-bit platforms (although we considered embedded devices as well)
- Use all the available tricks!
Practical performance comparison
Genus 1 versus Genus 2

- 128-bit security level
- High-end 64-bit platforms (although we considered embedded devices as well)
- Use all the available tricks!
- Let’s start with an arithmetic interlude: Why do we care about “special” primes?
In genus 1 “special” primes are used to speed-up modular reduction

- NIST $p_{224} = 2^{224} - 2^{96} + 1$
- NIST $p_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
- Bernstein $p_{25519} = 2^{255} - 19$
In genus 1 “special” primes are used to speed-up modular reduction

- NIST $p_{224} = 2^{224} - 2^{96} + 1$
- NIST $p_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
- Bernstein $p_{25519} = 2^{255} - 19$

### Mersenne primes

- Prime of the form $2^q - 1$, with $q$ prime
- Allows very efficient modular arithmetic
- Gaudry-Schost found a cryptographic Kummer surface over $\mathbb{F}_p$ with $p = 2^{127} - 1$

<table>
<thead>
<tr>
<th>#</th>
<th>$q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>31</td>
</tr>
<tr>
<td>9</td>
<td>61</td>
</tr>
<tr>
<td>10</td>
<td>89</td>
</tr>
<tr>
<td>11</td>
<td>107</td>
</tr>
<tr>
<td>12</td>
<td>127</td>
</tr>
<tr>
<td>13</td>
<td>521</td>
</tr>
<tr>
<td>14</td>
<td>607</td>
</tr>
</tbody>
</table>
In genus 1 “special” primes are used to speed-up modular reduction

- NIST $p_{224} = 2^{224} - 2^{96} + 1$
- NIST $p_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
- Bernstein $p_{25519} = 2^{255} - 19$

Mersenne primes

- Prime of the form $2^q - 1$, with $q$ prime
- Allows very efficient modular arithmetic
- Gaudry-Schost found a cryptographic Kummer surface over $\mathbb{F}_p$ with $p = 2^{127} - 1$

≈ 128-bit security for genus 2

<table>
<thead>
<tr>
<th>#</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>31</td>
</tr>
<tr>
<td>9</td>
<td>61</td>
</tr>
<tr>
<td>10</td>
<td>89</td>
</tr>
<tr>
<td>11</td>
<td>107</td>
</tr>
<tr>
<td>12</td>
<td>127</td>
</tr>
<tr>
<td>13</td>
<td>521</td>
</tr>
<tr>
<td>14</td>
<td>607</td>
</tr>
</tbody>
</table>
\[ c = a + b \mod (2^{127} - 1) = \begin{cases} 
  a + b & \text{if } (a + b) \leq 2^{127} - 1 \\
  a + b - (2^{127} - 1) & \text{if } (a + b) > 2^{127} - 1 
\end{cases} \]

Constant-time: addition + conditional subtraction

\[ = \text{addition} + \text{subtraction} + \text{masking (uses registers)} \]

Zero is represented by 0 or 2^{127} − 1

Mersenne to the rescue! – Modular addition

\[ a + b < 2^{128} \]
Mersenne to the rescue! – Modular addition

\[ c = a + b \mod (2^{127} - 1) = \begin{cases} 
  a + b & \text{if } (a + b) \leq 2^{127} - 1 \\
  a + b - (2^{127} - 1) & \text{if } (a + b) > 2^{127} - 1
\end{cases} \]

Constant-time: addition + conditional subtraction

= addition + subtraction + masking (uses registers)

\[ R(x) = x - \left\lfloor \frac{x}{2^{127}} \right\rfloor (2^{127} - 1) = x - \left\lfloor \frac{x}{2^{127}} \right\rfloor 2^{127} + \left\lfloor \frac{x}{2^{127}} \right\rfloor \]

Zero is represented by 0 or 2^{127} – 1
Mersenne to the rescue! – Modular addition

\[ c = a + b \mod (2^{127}-1) = \begin{cases} a + b & \text{if } (a + b) \leq 2^{127} - 1 \\ a + b - (2^{127}-1) & \text{if } (a + b) > 2^{127} - 1 \end{cases} \]

Constant-time: addition + conditional subtraction

= addition + subtraction + masking (uses registers)

\[ R(x) = x - \left\lfloor \frac{x}{2^{127}} \right\rfloor (2^{127} - 1) = x - \left\lfloor \frac{x}{2^{127}} \right\rfloor 2^{127} + \left\lfloor \frac{x}{2^{127}} \right\rfloor \]

If the msb is zero then leave it at zero
If the msb is one then set it to zero
Idea: use the bit-reset instruction!

\[ \in \{0, 1\} \]
Mersenne to the rescue! – Modular addition

\[ a + b < 2^{128} \]

\[ c = a + b \mod (2^{127} - 1) = \begin{cases} 
  a + b & \text{if } (a + b) \leq 2^{127} - 1 \\
  a + b - (2^{127} - 1) & \text{if } (a + b) > 2^{127} - 1 
\end{cases} \]

Constant-time: addition + conditional subtraction

= addition + subtraction + masking (uses registers)

\[ R(x) = x - \left\lfloor \frac{x}{2^{127}} \right\rfloor (2^{127} - 1) = x - \left\lfloor \frac{x}{2^{127}} \right\rfloor 2^{127} + \left\lfloor \frac{x}{2^{127}} \right\rfloor 
\]

Compute: \( c = R(a + b) \) when \( 0 \leq a, b < 2^{127} \) then \( 0 \leq c < 2^{127} \)

Avoid masking and extra register usage

Cost modular addition: \( 2x \text{ add} + 1x \text{ bit-reset instruction} \)
\[ c = a \times b = c_H 2^{128} + c_L, \text{ with} \]
\[ 0 \leq a, b < 2^{127}, 0 \leq c_L < 2^{128} \quad \text{and} \quad 0 < c_H \leq \left\lfloor \frac{(2^{127} - 1)^2}{2^{128}} \right\rfloor = 2^{126} - 1 \]

\[ c \equiv c_H 2^{128} + c_L - 2c_H (2^{127} - 1) \equiv c_L + 2c_H \left( \text{mod} \ (2^{127} - 1) \right) \]
Mersenne to the rescue! – Modular multiplication

\[ c = a \times b = c_H 2^{128} + c_L, \text{ with} \]

\[ 0 \leq a, b < 2^{127}, 0 \leq c_L < 2^{128} \quad \text{and} \quad 0 < c_H \leq \left\lfloor \frac{(2^{127} - 1)^2}{2^{128}} \right\rfloor = 2^{126} - 1 \]

\[ c \equiv c_H 2^{128} + c_L - 2c_H (2^{127} - 1) \equiv c_L + 2c_H \pmod{(2^{127} - 1)} \]

\[ c \equiv R(R(c_L) + 2c_H) \pmod{(2^{127} - 1)} \]

Can be > \(2^{128}\)
Mersenne to the rescue! – Modular multiplication

\[ c = a \times b = c_H 2^{128} + c_L, \text{ with} \]
\[ 0 \leq a, b < 2^{127}, 0 \leq c_L < 2^{128} \quad \text{and} \quad 0 < c_H \leq \left\lfloor \frac{(2^{127} - 1)^2}{2^{128}} \right\rfloor = 2^{126} - 1 \]

\[ c \equiv c_H 2^{128} + c_L - 2c_H (2^{127} - 1) \equiv c_L + 2c_H \pmod{(2^{127} - 1)} \]

Can be > \( 2^{128} \)

\[ c \equiv R(R(c_L) + 2c_H) \pmod{(2^{127} - 1)} \]

Reduction cost: 6x add, 2x bit-reset, 1x shift

\[ \leq 2^{127} + 2(2^{126} - 1) = 2(2^{127} - 1) \]
Mersenne to the rescue! – Modular multiplication

\[ c = a \times b = c_H2^{128} + c_L, \text{ with} \]

\[ 0 \leq a, b < 2^{127}, 0 \leq c_L < 2^{128} \quad \text{and} \quad 0 < c_H \leq \left\lfloor \frac{(2^{127} - 1)^2}{2^{128}} \right\rfloor = 2^{126} - 1 \]

\[ c \equiv c_H2^{128} + c_L - 2c_H(2^{127} - 1) \equiv c_L + 2c_H \left( \text{mod} \ (2^{127} - 1) \right) \]

\[ c \equiv R(R(c_L) + 2c_H) \left( \text{mod} \ (2^{127} - 1) \right) \]

\[ \leq 2^{127} + 2(2^{126} - 1) = 2(2^{127} - 1) \]

Reduction cost: 6x add, 2x bit-reset, 1x shift
Multiplication: 4x mul and 5x add instruction
Montgomery friendly primes

Interleaved radix-$2^b$ Montgomery multiplication

$$C \equiv A \cdot B \cdot 2^{-bn} \mod p, \mu = -p^{-1} \mod 2^b, A = \sum_{i=0}^{n-1} a_i 2^{bi}$$

C=0

for $i = 0$ to $n - 1$ do

$$C = C + a_i \cdot B$$

$$q = \mu \cdot C \mod 2^b$$

$$C = \frac{C + q \cdot p}{2^b}$$

Montgomery: Modular Multiplication Without Trial Division. Math. of Comp. 1985
Montgomery friendly primes

Interleaved radix-$2^b$ Montgomery multiplication

\[ C \equiv A \cdot B \cdot 2^{-bn} \mod p, \mu = -p^{-1} \mod 2^b, A = \sum_{i=0}^{n-1} a_i 2^{bi} \]

\[ C = 0 \]

\textbf{for} \ i = 0 \ \textbf{to} \ n - 1 \ \textbf{do} \]

\[ C = C + a_i \cdot B \]

\[ q = \mu \cdot C \mod 2^b \]

\[ C = \frac{C + q \cdot p}{2^b} \]

Montgomery: \textit{Modular Multiplication Without Trial Division}. Math. of Comp. 1985
Montgomery friendly primes

Interleaved radix-\(2^b\) Montgomery multiplication

\[ C \equiv A \cdot B \cdot 2^{-bn} \mod p, \mu = -p^{-1} \mod 2^b, A = \sum_{i=0}^{n-1} a_i 2^{bi} \]

C=0
for \( i = 0 \) to \( n - 1 \) do

\[ C = C + a_i \cdot B \]

\[ q = \mu \cdot C \mod 2^b \]

\[ C = \frac{C + q \cdot p}{2^b} \]

If \( p = \pm 1 \mod 2^b \) then \( \mu = \mp 1 \mod 2^b \)

Montgomery: *Modular Multiplication Without Trial Division*. Math. of Comp. 1985
Montgomery friendly primes

Interleaved radix-$2^b$ Montgomery multiplication

\[ C \equiv A \cdot B \cdot 2^{-bn} \mod p, \mu = -p^{-1} \mod 2^b, A = \sum_{i=0}^{n-1} a_i 2^{bi} \]

\[ C = 0 \\
\text{for } i = 0 \text{ to } n - 1 \text{ do} \\
\quad C = C + a_i \cdot B \\
\quad q = \mu \cdot C \mod 2^b \\
\quad C = \frac{C + q \cdot p}{2^b} \]

Not much we can do: this is the multiplication

If \( p = \pm 1 \mod 2^b \) then \( \mu = \mp 1 \mod 2^b \)

Additionally, if \( p \) has a “special” form: avoid muls

Montgomery: *Modular Multiplication Without Trial Division*. Math. of Comp. 1985
Montgomery friendly primes

Interleaved radix-$2^b$ Montgomery multiplication

$$C \equiv A \cdot B \cdot 2^{-bn} \mod p, \mu = -p^{-1} \mod 2^b, A = \sum_{i=0}^{n-1} a_i 2^{bi}$$

C=0

for $i = 0$ to $n - 1$ do

$C = C + a_i \cdot B$
$q = \mu \cdot C \mod 2^b$
$C = \frac{C + q \cdot p}{2^b}$

Not much we can do: this is the multiplication

If $p = \pm 1 \mod 2^b$ then $\mu = \mp 1 \mod 2^b$

Additionally, if $p$ has a “special” form: avoid muls

Example: $2^b (2^b - c) - 1$

$2^{127} - 1 = 2^{64} (2^{63} - 0) - 1$

Montgomery: Modular Multiplication Without Trial Division. Math. of Comp. 1985
Benchmark Platform

- Intel Core i7-3520M (Ivy Bridge) processor at 2893.484 MHz
- hyperthreading turned off and overclocking (“turbo boost”) disabled
Security & Benchmark Platform

Benchmark Platform

- Intel Core i7-3520M (Ivy Bridge) processor at 2893.484 MHz
- hyperthreading turned off and overclocking ("turbo boost") disabled

Generic Attack: Pollard rho

- [Pollard-MoC78]
- $\sqrt{(\pi r)/(2 \#\text{Aut})}$, where $\#\text{Aut} \geq 2$ for curves with group order $h \cdot r$

Battle #1

NISTp-256 versus Generic1271
Battle #1

NISTp-256 versus Generic1271

Generic genus 1 versus Generic genus 2

Generic?
- No special requirements on the curve
- Techniques can be applied to all genus 1 or genus 2 curves
- Use “special” primes for efficiency
- Use prime order curves for optimal security
# NISTp-256 versus Generic1271

<table>
<thead>
<tr>
<th></th>
<th>NISTp-256</th>
<th>Generic1271</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>p</strong></td>
<td>(2^{256} - 2^{224} + 2^{192} + 2^{96} - 1)</td>
<td>(\begin{cases} 2^{127} - 1 \ 2^{64}(2^{63} - 0) - 1 \end{cases})</td>
</tr>
<tr>
<td><strong>Order</strong></td>
<td>Prime order</td>
<td>Prime order</td>
</tr>
<tr>
<td><strong>Scalar multiplication</strong></td>
<td>windowing</td>
<td>windowing</td>
</tr>
<tr>
<td><strong>Coordinate / curve</strong></td>
<td>Jacobian coordinates with (a = -3) for short Weierstrass curves</td>
<td>[CL]</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>(\sqrt{(\pi r)/(2 \cdot 2)} \approx 2^{127.8})</td>
<td>(\sqrt{(\pi r)/(2 \cdot 2)} \approx 2^{126.8})</td>
</tr>
</tbody>
</table>

We use arithmetic on imaginary quadratic curves using homogeneous projective coordinates. We optimized the formulas from:

## NISTp-256 versus Generic1271

<table>
<thead>
<tr>
<th></th>
<th>NISTp-256</th>
<th>Generic1271</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>$p$</strong></td>
<td>$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$</td>
<td>$\begin{cases} 2^{127} - 1 \quad (a) \ 2^{64}(2^{63} - 0) - 1 \quad (b) \end{cases}$</td>
</tr>
<tr>
<td><strong>Order</strong></td>
<td>Prime order</td>
<td>Prime order</td>
</tr>
<tr>
<td><strong>Scalar multiplication</strong></td>
<td>windowing</td>
<td>windowing</td>
</tr>
<tr>
<td><strong>Coordinate / curve</strong></td>
<td>Jacobian coordinates with $a = -3$ for short Weierstrass curves</td>
<td>[CL]</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>$\sqrt{\frac{(\pi r)}{(2 \cdot 2)}} \approx 2^{127.8}$</td>
<td>$\sqrt{\frac{(\pi r)}{(2 \cdot 2)}} \approx 2^{126.8}$</td>
</tr>
<tr>
<td><strong>Double</strong></td>
<td>$3M+5S$</td>
<td>$34M+6S$</td>
</tr>
<tr>
<td><strong>Addition</strong></td>
<td>$11M+5S$</td>
<td>$44M+4S$</td>
</tr>
<tr>
<td><strong>Mixed addition</strong></td>
<td>$7M+4S$</td>
<td>$37M+5S$</td>
</tr>
</tbody>
</table>
## NISTp-256 versus Generic1271

<table>
<thead>
<tr>
<th></th>
<th>NISTp-256</th>
<th>Generic1271</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Order</strong></td>
<td>Prime order</td>
<td>Prime order</td>
</tr>
<tr>
<td><strong>Scalar multiplication</strong></td>
<td>windowing</td>
<td>windowing</td>
</tr>
<tr>
<td><strong>Coordinate / curve</strong></td>
<td>Jacobian coordinates with $a = -3$ for short Weierstrass curves</td>
<td>[CL]</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>$\sqrt{\left(\pi r\right) / (2 \cdot 2)} \approx 2^{127.8}$</td>
<td>$\sqrt{\left(\pi r\right) / (2 \cdot 2)} \approx 2^{126.8}$</td>
</tr>
</tbody>
</table>

### Numbers

<table>
<thead>
<tr>
<th>Genus 1: NISTp-256</th>
<th>658,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genus 2: generic1271 (a)</td>
<td>248,000</td>
</tr>
<tr>
<td>Genus 2: generic1271 (b)</td>
<td>295,000</td>
</tr>
</tbody>
</table>
Battle #2

GLV-\(j=0\) versus Buhler/Koblitz/\(GLV\)
Scalar Decomposition over Prime Fields

- Gallant, Lambert, Vanstone [GLV-C01]
- Use non-trivial endomorphism
- Larger endomorphism ring means larger dimensional scalar decomposition
Scalar Decomposition over Prime Fields

- Gallant, Lambert, Vanstone [GLV-C01]
- Use non-trivial endomorphism
- Larger endomorphism ring means larger dimensional scalar decomposition

Scalar Decomposition over Prime Fields

- Genus 1 over $\mathbb{F}_p$
  - 256-bit primes
  - Allows: 2-GLV

- Genus 2 over $\mathbb{F}_p$
  - 128-bit primes
  - Allows: 4-GLV
Reducing the Number of Point Doublings

- $d$-dimensional scalar decomposition
- Decompose a scalar $k$ into $d$ “mini-scalars” $k_i \approx \frac{d}{\sqrt{k}}$
- Perform a multi-scalar multiplication with these $d$ smaller scalars
Reducing the Number of Point Doublings

- $d$-dimensional scalar decomposition
- Decompose a scalar $k$ into $d$ “mini-scalars” $k_i \approx \frac{d}{\sqrt{k}}$
- Perform a multi-scalar multiplication with these $d$ smaller scalars

Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \frac{d}{\sqrt{k}}$

\[
[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i] P = [k_0]P + [k_1][\lambda]P + \cdots + [k_{d-1}][\lambda^{d-1}]P
\]
Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \sqrt[d]{k}$

$$[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i]P = [k_0]P + [k_1]([\lambda]P) + \cdots + [k_{d-1}](\lambda^{d-1}P)$$

Reducing the Number of Point Doublings

- $d$-dimensional scalar decomposition
- Decompose a scalar $k$ into $d$ “mini-scalars” $k_i \approx \sqrt[d]{k}$
- Perform a multi-scalar multiplication with these $d$ smaller scalars

Example: $d = 2$

$\begin{array}{cccc}
k_0 &=& k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\
k_1 &=& k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \end{array}$

Precompute: $\{\emptyset, P, [\lambda]P, P + [\lambda]P\}$
Reducing the Number of Point Doublings

- \(d\)-dimensional scalar decomposition
- Decompose a scalar \(k\) into \(d\) “mini-scalars” \(k_i \approx d\sqrt{k}\)
- Perform a multi-scalar multiplication with these \(d\) smaller scalars

Assume we can multiply efficiently by (powers) of some integer \(\lambda \approx d\sqrt{k}\)

\[
[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i] P = [k_0]P + [k_1]([\lambda]P) + \cdots + [k_{d-1}]( [\lambda^{d-1}] P )
\]

Approach #1

Precompute: \(\{ \emptyset, P, [\lambda]P, P + [\lambda]P \}\)

Example: \(d = 2\)
Reducing the Number of Point Doublings

- \( d \)-dimensional scalar decomposition
- Decompose a scalar \( k \) into \( d \) “mini-scalars” \( k_i \approx \sqrt[d]{k} \)
- Perform a multi-scalar multiplication with these \( d \) smaller scalars

Assume we can multiply efficiently by (powers) of some integer \( \lambda \approx \sqrt[d]{k} \)

\[
[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i] P = [k_0]P + [k_1](\lambda P) + \cdots + [k_{d-1}](\lambda^{d-1} P)
\]

Example:

\( d = 2 \)

Precompute: \( \{ \emptyset, P, [\lambda]P, P + [\lambda]P \} \)
Reducing the Number of Point Doublings

- *d*-dimensional scalar decomposition
- Decompose a scalar $k$ into $d$ “mini-scalars” $k_i \approx \frac{d}{\sqrt{k}}$
- Perform a multi-scalar multiplication with these $d$ smaller scalars

Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \frac{d}{\sqrt{k}}$

$$[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i] P = [k_0]P + [k_1](\lambda P) + \cdots + [k_{d-1}](\lambda^{d-1}P)$$

**Approach #1**

Precompute: $\{\emptyset, P, [\lambda]P, P + [\lambda]P\}$

Example: $d = 2$

$k_0 = \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \end{bmatrix}$

$k_1 = \begin{bmatrix} k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \end{bmatrix}$
Reducing the Number of Point Doublings

- **d-dimensional scalar decomposition**
- Decompose a scalar $k$ into $d$ “mini-scalars” $k_i \approx \frac{d}{\sqrt{k}}$
- Perform a multi-scalar multiplication with these $d$ smaller scalars

Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \frac{d}{\sqrt{k}}$

\[
[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i] P = [k_0]P + [k_1](\lambda P) + \cdots + [k_{d-1}](\lambda^{d-1}P)
\]

**Approach #1**

Precompute: $\emptyset, P, [\lambda]P, P + [\lambda]P$

Example: $d = 2$
Reducing the Number of Point Doublings

- $d$-dimensional scalar decomposition
- Decompose a scalar $k$ into $d$ "mini-scalars" $k_i \approx \frac{d}{\sqrt{k}}$
- Perform a multi-scalar multiplication with these $d$ smaller scalars

Assume we can multiply efficiently by (powers) of some integer $\lambda \approx \frac{d}{\sqrt{k}}$

$$[k]P = \sum_{i=0}^{d-1} [k_i \lambda^i]P = [k_0]P + [k_1]([\lambda]P) + \cdots + [k_{d-1}]([\lambda^{d-1}]P)$$

**Approach #2**

Precompute: $\begin{cases} \{\emptyset, P, 2P, 3P\} \\ \{\emptyset, [\lambda]P, 2[\lambda]P, 3[\lambda]P\} \end{cases}$

Example: $d = 2$
Reducing the Number of Point Doublings

- $d$-dimensional scalar decomposition
- Decompose a scalar $k$ into $d$ “mini-scalars” $k_i \approx d\sqrt{k}$
- Perform a multi-scalar multiplication with these $d$ smaller scalars

Assume we can multiply efficiently by (powers) of some integer $\lambda \approx d\sqrt{k}$

$$[k]P = \sum_{i=0}^{d-1} [k_i\lambda^i]P = [k_0]P + [k_1](\lambda P) + \cdots + [k_{d-1}](\lambda^{d-1}P)$$

**Approach #2**

Precompute: $\{\emptyset, P, 2P, 3P\}$

Example: $d = 2$
Buhler-Koblitz curves

- \( C / \mathbb{F}_p : y^2 = x^5 + a \)
- \( \psi : \text{Jac}(C) \rightarrow \text{Jac}(C), \psi(D) = [\lambda]D, \) for \( 0 < \lambda < r \)
- Decompose the scalar using [PJL]
  Cost: 20 long integer muls

Curve Choice

\[
\begin{align*}
\begin{cases}
p_{127m} = (2^{63} - 27433)2^{64} + 1 \\
a = 17 \\
\mu = -p_{127m}^{-1} \mod 2^{64} = -1
\end{cases}
\end{align*}
\]
254-bit prime order

\[
\begin{align*}
\begin{cases}
p_{128n} = 2^{128} - 24935 \\
a = 3^7
\end{cases}
\end{align*}
\]
256-bit prime order

[PJL] Park, Jeong, Lim: *Speeding up point multiplication on hyperelliptic curves with efficiently-computable endomorphisms.* Eurocrypt 2002
Offline
Pre-compute $2^4$ points
$11A+3\psi$

Online
$64D+64A$

Curve Choice

\[
\begin{aligned}
\begin{cases}
p_{127m} = (2^{63} - 27433)2^{64} + 1 \\
a = 17 \\
\mu = -p_{127m}^{-1} \text{ mod } 2^{64} = -1
\end{cases}
\end{aligned}
\]

254-bit prime order

\[
\begin{aligned}
\begin{cases}
p_{128n} = 2^{128} - 24935 \\
a = 3^7
\end{cases}
\end{aligned}
\]

256-bit prime order
BuhlerKoblitzGLV – 4-dimensional GLV

**Curve Choice**

\[
\begin{cases}
p_{127m} = (2^{63} - 27433)2^{64} + 1 \\
a = 17
\end{cases}
\]

\[
\mu = -p_{127m}^{-1} \mod 2^{64} = -1
\]

254-bit prime order

\[
\begin{cases}
p_{128n} = 2^{128} - 24935 \\
a = 3^7
\end{cases}
\]

256-bit prime order

**Offline**

Pre-compute \(2^4\) points

\[11A + 3\psi + 1I + (3+4) \cdot 15M\]

**Online**

\[64D + 64A \rightarrow 64D + 64MA\]

Recall: \(A = 44M + 4S\), \(MA = 37M + 5S\)

**Additional cost:** \(1I + 105M\)

**Savings:** \(64(A - MA) = 448M - 64S\)

Speedup when: \(I < 279M\)

Montgomery: *Speeding the Pollard and elliptic curve methods of factorization. Math. of Comp. 1987*
<table>
<thead>
<tr>
<th></th>
<th>GLV-j=0</th>
<th>BuhlerKoblitzGLV</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>$2^{256} - 11733$</td>
<td>$\left{ \begin{align*} 2^{128} - 24935 \ (2^{63} - 27433)2^{64} + 1 \end{align*} \right.$ (a)</td>
</tr>
<tr>
<td>Order</td>
<td>Prime order</td>
<td>Prime order</td>
</tr>
<tr>
<td>Scalar multiplication</td>
<td>2-dimensional GLV</td>
<td>4-dimensional GLV (approach #1)</td>
</tr>
<tr>
<td>Coordinate / curve</td>
<td>j-invariant 0 in Weierstrass form $y^2 = x^3 + 2$</td>
<td>Buhler-Koblitiz curve $y^2 = x^5 + a$</td>
</tr>
</tbody>
</table>

GLV-j=0 versus BuhlerKoblitzGLV

<table>
<thead>
<tr>
<th></th>
<th>GLV-j=0</th>
<th>BuhlerKoblitzGLV</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )</td>
<td>( 2^{256} - 11733 )</td>
<td>( \begin{cases} 2^{128} - 24935 \ (2^{63} - 27433)2^{64} + 1 \end{cases} ) (( a )) (( b ))</td>
</tr>
<tr>
<td>Order</td>
<td>Prime order</td>
<td>Prime order</td>
</tr>
<tr>
<td>Scalar multiplication</td>
<td>2-dimensional GLV</td>
<td>4-dimensional GLV (approach #1)</td>
</tr>
<tr>
<td>Cost scalar multiplication</td>
<td>( 1\mathbf{I} + 904\mathbf{M} + 690\mathbf{S} )</td>
<td>20 integer muls + ( 3\psi + 2\mathbf{I} + 5005\mathbf{M} + 748\mathbf{S} )</td>
</tr>
<tr>
<td>Security</td>
<td>( \sqrt{\frac{(\pi r)}{(2 \cdot 6)}} \approx 2^{127.0} )</td>
<td>( \sqrt{\frac{(\pi r)}{(2 \cdot 10)}} \approx 2^{125.7} )</td>
</tr>
</tbody>
</table>

### GLV-\(j=0\) versus BuhlerKoblitzGLV

<table>
<thead>
<tr>
<th></th>
<th>GLV-(j=0)</th>
<th>BuhlerKoblitzGLV</th>
</tr>
</thead>
<tbody>
<tr>
<td>(p)</td>
<td>(2^{256} – 11733)</td>
<td>(\begin{cases} 2^{128} – 24935 \ (2^{63} – 27433)2^{64} + 1 \end{cases})</td>
</tr>
<tr>
<td>Order</td>
<td>Prime order</td>
<td>Prime order</td>
</tr>
<tr>
<td>Scalar multiplication</td>
<td>2-dimensional GLV</td>
<td>4-dimensional GLV (approach #1)</td>
</tr>
<tr>
<td>Cost scalar multiplication</td>
<td>1I + 904M + 690S</td>
<td>20 integer muls + (3\psi+2I+5005M+748S)</td>
</tr>
<tr>
<td>Security</td>
<td>(\sqrt{\frac{\pi r}{(2 \cdot 6)}} \approx 2^{127.0})</td>
<td>(\sqrt{\frac{\pi r}{(2 \cdot 10)}} \approx 2^{125.7})</td>
</tr>
</tbody>
</table>

| Genus 1: GLV-\(j=0\) | 145,000 |
| Genus 2: BuhlerKoblitzGLV (a) | 164,000 |
| Genus 2: BuhlerKoblitzGLV (b) | 156,000 |

Battle #3

curve25519 versus Kummer1271
Battle #3

curve25519 versus Kummer1271

Use the Kummer surface from
Laddering algorithms

Elliptic curves

- [M] differential addition: compute $P + Q$ from \{\(P, Q, P - Q\)\} without $y$-coord
- to compute $kP$ keep \{\(mP, (m + 1)P\)\} such that $(m + 1)P - mP = P$
- Identify $P = (P_x, P_y)$ and $-P = (P_x, -P_y)$
- Cost for double+differential add: $5\text{M} + 4\text{S}$

Laddering algorithms

Elliptic curves

- [M] differential addition: compute $P + Q$ from $\{P, Q, P - Q\}$ without $y$-coord
- to compute $kP$ keep $\{mP, (m + 1)P\}$ such that $(m + 1)P - mP = P$
- Identify $P = (P_x, P_y)$ and $-P = (P_x, -P_y)$
- Cost for double+differential add: $5M + 4S$

Genus 2 curves

Work on the Kummer surface associated to a Jacobian, rather than on the Jacobian itself

- [SS] genus 2 analogue $\text{Jac}(C) \rightarrow K$ is 2-to-1
- [G] faster Kummer surface
- [C] even faster “squares only” setting on the Kummer surface
- Cost for double+differential add: $16M + 9S$

[C] Cosset: *Factorization with genus 2 curves*. Math. of Comp. 2010
### Laddering algorithms

#### Elliptic curves
- [M] differential addition: compute \( P + Q \) from \( \{P, Q, P - Q\} \) without \( y \)-coord
- to compute \( kP \) keep \( \{mP, (m + 1)P\} \) such that \((m + 1)P - mP = P\)
- Identify \( P = (P_x, P_y) \) and \(-P = (P_x, -P_y)\)
- Cost for double+differential add: \( 5M + 4S \)

#### Genus 2 curves
Work on the Kummer surface associated to a Jacobian, rather than on the Jacobian itself
- [SS] genus 2 analogue \( \text{Jac}(C) \rightarrow K \) is 2-to-1
- [G] faster Kummer surface
- [C] even faster “squares only” setting on the Kummer surface
- Cost for double+differential add: \( 16M + 9S \)

- no additions: does allow scalar multiplication
- attractive setting for Diffie-Hellman like protocols
- Inherently runs in constant time

---

**References**

- [C] Cosset: *Factorization with genus 2 curves*. Math. of Comp. 2010
### curve25519 versus Kummer1271

<table>
<thead>
<tr>
<th></th>
<th>curve25519</th>
<th>Kummer1271</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>$2^{255} - 19$</td>
<td>$\begin{cases} 2^{127} - 1 \quad (a) \ 2^{64}(2^{63} - 0) - 1 \quad (b) \end{cases}$</td>
</tr>
<tr>
<td>Order</td>
<td>8 · 253-bit prime / 4 · 253-bit prime</td>
<td>16 · 250-bit prime / 16 · 251-bit prime</td>
</tr>
<tr>
<td>Scalar multiplication</td>
<td>Montgomery ladder</td>
<td>Kummer ladder</td>
</tr>
<tr>
<td>Coordinate / curve</td>
<td>Montgomery curve</td>
<td>“Squares only” setting on a Kummer surface</td>
</tr>
<tr>
<td>Double + dif. add</td>
<td>$5M + 4S$</td>
<td>$16M + 9S$</td>
</tr>
<tr>
<td>Security</td>
<td>$\sqrt{\left(\frac{\pi r}{\sqrt{2 \cdot 2}}\right)} \approx 2^{125.8}$</td>
<td>$\sqrt{\left(\frac{\pi r}{\sqrt{2 \cdot 2}}\right)} \approx 2^{124.8}$</td>
</tr>
</tbody>
</table>

Bernstein, Duif, Lange, Schwabe: *High-speed high-security signatures.* CHES 2011
### curve25519 versus Kummer1271

<table>
<thead>
<tr>
<th></th>
<th>curve25519</th>
<th>Kummer1271</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>$2^{255} - 19$</td>
<td>$\begin{cases} 2^{127} - 1 \quad (a) \ 2^{64}(2^{63} - 0) - 1 \quad (b) \end{cases}$</td>
</tr>
<tr>
<td>Order</td>
<td>8 · 253-bit prime / 4 · 253-bit prime</td>
<td>16 · 250-bit prime / 16 · 251-bit prime</td>
</tr>
<tr>
<td>Scalar multiplication</td>
<td>Montgomery ladder</td>
<td>Kummer ladder</td>
</tr>
<tr>
<td>Double + dif. add</td>
<td>$5M + 4S$</td>
<td>$16M + 9S$</td>
</tr>
<tr>
<td>Security</td>
<td>$\sqrt{\frac{(\pi r)}{(2 \cdot 2)}} \approx 2^{125.8}$</td>
<td>$\sqrt{\frac{(\pi r)}{(2 \cdot 2)}} \approx 2^{124.8}$</td>
</tr>
</tbody>
</table>

| Genus 1: curve25519 | 182,000 |
| Genus 2: Kummer1271 (a) | 117,000 |
| Genus 2: Kummer1271 (b) | 139,000 |

Bernstein, Duif, Lange, Schwabe: *High-speed high-security signatures.* CHES 2011
## Summary: genus 1 versus genus 2 over prime fields

<table>
<thead>
<tr>
<th>Curve</th>
<th>cycles</th>
<th>CT</th>
<th>protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genus 1: NISTp-256</td>
<td>658,000</td>
<td>?</td>
<td>all</td>
</tr>
<tr>
<td>Genus 2: generic1271 (a)</td>
<td>248,000</td>
<td>X</td>
<td>all</td>
</tr>
<tr>
<td>Genus 1: GLV-j=0</td>
<td>145,000</td>
<td>X</td>
<td>all</td>
</tr>
<tr>
<td>Genus 2: BuhlerKoblitzGLV (b)</td>
<td>156,000</td>
<td>X</td>
<td>all</td>
</tr>
<tr>
<td>Genus 1: curve25519</td>
<td>182,000</td>
<td>✓</td>
<td>some</td>
</tr>
<tr>
<td>Genus 2: Kummer1271 (a)</td>
<td>117,000</td>
<td>✓</td>
<td>some</td>
</tr>
</tbody>
</table>

**Generic**

- Genus 2 > 2.5 faster than genus 1
- Mersenne prime $2^{127}-1$ very efficient in practice
- NISTp-256 arithmetic ($2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$) is relatively slow
## Summary: genus 1 versus genus 2 over prime fields

<table>
<thead>
<tr>
<th>Curve</th>
<th>cycles</th>
<th>CT</th>
<th>protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genus 1: NISTp-256</td>
<td>658,000</td>
<td>?</td>
<td>all</td>
</tr>
<tr>
<td>Genus 2: generic1271 (a)</td>
<td>248,000</td>
<td>❌</td>
<td>all</td>
</tr>
<tr>
<td>Genus 1: GLV-j=0</td>
<td>145,000</td>
<td>❌</td>
<td>all</td>
</tr>
<tr>
<td>Genus 2: BuhlerKoblitzGLV (b)</td>
<td>156,000</td>
<td>❌</td>
<td>all</td>
</tr>
<tr>
<td>Genus 1: curve25519</td>
<td>182,000</td>
<td>✔</td>
<td>some</td>
</tr>
<tr>
<td>Genus 2: Kummer1271 (a)</td>
<td>117,000</td>
<td>✔</td>
<td>some</td>
</tr>
</tbody>
</table>

### Endomorphism

- Genus 1 slightly faster than genus 2
  (better genus 1 assembly implementation?)
- Montgomery friendly primes **faster** than primes of the form $2^{128} - c$
Summary: genus 1 versus genus 2 over prime fields

<table>
<thead>
<tr>
<th>Curve</th>
<th>cycles</th>
<th>CT</th>
<th>protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genus 1: NISTp-256</td>
<td>658,000</td>
<td>?</td>
<td>all</td>
</tr>
<tr>
<td>Genus 2: generic1271 (a)</td>
<td>248,000</td>
<td>✗</td>
<td>all</td>
</tr>
<tr>
<td>Genus 1: GLV-j=0</td>
<td>145,000</td>
<td>✗</td>
<td>all</td>
</tr>
<tr>
<td>Genus 2: BuhlerKoblitzGLV (b)</td>
<td>156,000</td>
<td>✗</td>
<td>all</td>
</tr>
<tr>
<td>Genus 1: curve25519</td>
<td>182,000</td>
<td>✓</td>
<td>some</td>
</tr>
<tr>
<td>Genus 2: Kummer1271 (a)</td>
<td>117,000</td>
<td>✓</td>
<td>some</td>
</tr>
</tbody>
</table>

Ladder

- Genus 2 faster than genus 1
- Thanks to the Kummer surface by Gaudry & Schost
  the Mersenne prime $2^{127} - 1$ comes to the rescue again
Genus 2 has many advantages over elliptic curves

- Larger endomorphism ring
  4-GLV possible in genus 2 versus 2-GLV in genus 1
- Can use the Mersenne prime $2^{127} - 1$
- Laddering using the Kummer surface is very efficient
- This results are on a 64-bit platform, smaller primes have more potential on embedded devices

**Final score**

genus 1 *versus* genus 2

1 : 2
Related / ongoing work

- Genus 2 curves over $\mathbb{F}_{p^2} \rightarrow 8$-dimensional scalar decomposition
  - Allows for 64-bit primes $p$
  - Faster attacks, reduced security from 128-bit to $\approx 112$-bit
- Practical analysis of security genus 1 versus genus 2 over $\mathbb{F}_p$
  - What is the effect of using the automorphism group in practice?

Future work

- Unlikely to attract attention from industry if less than order of magnitude faster: More work is needed!
- Using endomorphisms on the Kummer surface?
Related / ongoing work

- Genus 2 curves over $\mathbb{F}_{p^2}$ → 8-dimensional scalar decomposition
  Allows for 64-bit primes $p$
  Faster attacks, reduced security from 128-bit to $\approx$112-bit
- Practical analysis of security genus 1 versus genus 2 over $\mathbb{F}_p$
  What is the effect of using the automorphism group in practice?

Future work

- Unlikely to attract attention from industry if less than order of magnitude faster: More work is needed!
- Using endomorphisms on the Kummer surface?
Related / ongoing work

- Genus 2 curves over $\mathbb{F}_{p^2} \rightarrow 8$-dimensional scalar decomposition
  Allows for 64-bit primes $p$
  Faster attacks, reduced security from 128-bit to $\approx 112$-bit
- Practical analysis of security genus 1 versus genus 2 over $\mathbb{F}_p$
  What is the effect of using the automorphism group in practice?

Future work

- Unlikely to attract attention from industry if less than order of magnitude faster:
  More work is needed!
- Using endomorphisms on the Kummer surface?

Use elliptic or genus 2 curves?

Difficult to see. Always in motion is the future.
YODA, Star Wars Episode V: The Empire Strikes Back