Faster ECC over $\mathbb{F}_{2^{521}-1}$

Robert Granger$^1$ and Michael Scott$^2$

$^1$ Laboratory for Cryptologic Algorithms  
School of Computer and Communication Sciences  
EPFL, Switzerland  
robbiegranger@gmail.com

$^2$ CertiVox Labs  
mike.scott@certivox.com

31st March, PKC 2015
Overview

ECC efficiency

Generalised Repunit Primes

This work
Overview

ECC efficiency

Generalised Repunit Primes

This work
"In an ideal world, every web request could be defaulted to HTTPS."

– Electronic Frontier Foundation
"In an ideal world, every web request could be defaulted to HTTPS."

– Electronic Frontier Foundation

The case for using ECC is well-made, *but it was initially very slow.*
Making ECC fast

"In an ideal world, every web request could be defaulted to HTTPS."

– Electronic Frontier Foundation

The case for using ECC is well-made, but it was initially very slow.

To ameliorate the use of ECC, one can:

- Design faster protocols
- Make point multiplication faster
- Make point addition and doubling faster
- Make finite field arithmetic faster
Multiplication in $\mathbb{Z}/N\mathbb{Z}$

From an algorithmic perspective, two factors to consider:

- residue representation
- multiplication of representatives
Multiplication in $\mathbb{Z}/N\mathbb{Z}$

From an algorithmic perspective, two factors to consider:

- residue representation
- multiplication of representatives

Canonical representation of $\mathbb{Z}/N\mathbb{Z}$:

- residue representation: $\mathbb{Z}/N\mathbb{Z} = \{0, \ldots, N - 1\}$
- ‘Modular mul. = residue mul. (in $\mathbb{Z}$) + modular reduction’

Question

For $0 \leq x, y < N$, which of the following can be computed fastest:

- $xy$
- $xy \pmod{N}$
Multiplication in $\mathbb{Z}/N\mathbb{Z}$

From an algorithmic perspective, two factors to consider:
- residue representation
- multiplication of representatives

Canonical representation of $\mathbb{Z}/N\mathbb{Z}$:
- residue representation: $\mathbb{Z}/N\mathbb{Z} = \{0, \ldots, N - 1\}$
- ‘Modular mul. = residue mul. (in $\mathbb{Z}$) + modular reduction’

Question

*For $0 \leq x, y < N$, which of the following can be computed fastest:*

\[ xy \quad \text{or} \quad xy \pmod{N}? \]
Mersenne Numbers

Let $N = 2^n - 1$. Residues are $n$-bit integers and for $x, y \in \mathbb{Z}/N\mathbb{Z}$,

$$xy = z_1 2^n + z_0$$
$$= z_1 (2^n - 1) + z_1 + z_0$$
$$\equiv z_1 + z_0 \pmod{N}$$

- If schoolbook multiplication is optimal, then multiplication modulo $N$ is arguably ‘near optimal’
- **Drawback**: too few Mersenne primes in ECC range, just $2^{521} - 1$
- Similar trick for Crandall numbers $N = 2^n - c$ for $c$ very small
Generalised Mersenne Numbers


<table>
<thead>
<tr>
<th>Bitlength</th>
<th>Prime</th>
</tr>
</thead>
<tbody>
<tr>
<td>192</td>
<td>$2^{192} - 2^{64} - 1$</td>
</tr>
<tr>
<td>224</td>
<td>$2^{224} - 2^{96} + 1$</td>
</tr>
<tr>
<td>256</td>
<td>$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$</td>
</tr>
<tr>
<td>384</td>
<td>$2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$</td>
</tr>
<tr>
<td>521</td>
<td>$2^{521} - 1$</td>
</tr>
</tbody>
</table>

- Used by governments, military, banks, e-commerce, browsers, Blackberry and Blackberry Enterprise Server, openSSL,...
- Several issues $\implies$ Suite B curves no longer trusted:
  - How were the specified seeds chosen?
  - Hard to implement them securely (Bernstein-Lange)
  - Dual_EC_DRBG
To answer my earlier question...

Let \( N = 2^n - 1 \), and let

\[
x = \sum_{i=0}^{n-1} x_i 2^i, \quad y = \sum_{i=0}^{n-1} y_i 2^i
\]

Then

\[
xy \equiv \sum_{i=0}^{n-1} (x \circ y)_i 2^i \pmod{N},
\]

where

\[
(x \circ y)_i = \sum_{j+k \equiv i \pmod{n}} x_j y_k
\]
To answer my earlier question...

Let $N = 2^n - 1$, and let

$$x = \sum_{i=0}^{n-1} x_i 2^i, \quad y = \sum_{i=0}^{n-1} y_i 2^i$$

Then

$$xy \equiv \sum_{i=0}^{n-1} (x \circ y)_i 2^i \pmod{N},$$

where

$$(x \circ y)_i = \sum_{j+k\equiv i \pmod{n}} x_j y_k$$

- Using an IBDWT, at asymptotic bitlengths, multiplication modulo a Mersenne number is *twice as fast* as integer multiplication
To answer my earlier question...

Let \( N = 2^n - 1 \), and let

\[
x = \sum_{i=0}^{n-1} x_i 2^i, \quad y = \sum_{i=0}^{n-1} y_i 2^i
\]

Then

\[
xy \equiv \sum_{i=0}^{n-1} (x \circ y)_i 2^i \pmod{N},
\]

where

\[
(x \circ y)_i = \sum_{j+k \equiv i \pmod{n}} x_j y_k
\]

- Using an IBDWT, at asymptotic bitlengths, multiplication modulo a Mersenne number is \textit{twice as fast} as integer multiplication
- Hence modulus can influence how one should multiply residues
To answer my earlier question...

Let $N = 2^n - 1$, and let

$$x = \sum_{i=0}^{n-1} x_i 2^i, \quad y = \sum_{i=0}^{n-1} y_i 2^i$$

Then

$$xy \equiv \sum_{i=0}^{n-1} (x \circ y)_i 2^i \pmod{N},$$

where

$$(x \circ y)_i = \sum_{j+k=i \pmod{n}} x_j y_k$$

- Using an IBDWT, at asymptotic bitlengths, multiplication modulo a Mersenne number is \textit{twice as fast} as integer multiplication
- Hence modulus can influence how one should multiply residues
- Are there such speedups at ECC bitlengths?
Overview

ECC efficiency

Generalised Repunit Primes

This work
Definition

For \( m + 1 \) an odd prime and \( t \) an integer let

\[
p = \Phi_{m+1}(t) = t^m + t^{m-1} + \cdots + t + 1.
\]

If prime, we call \( p \) a Generalised Repunit Prime.
Generalised Repunit Primes

**Definition**

For $m + 1$ an odd prime and $t$ an integer let

$$p = \Phi_{m+1}(t) = t^m + t^{m-1} + \cdots + t + 1.$$  

If prime, we call $p$ a Generalised Repunit Prime.

Embed $\mathbb{Z}/(\Phi_{m+1}(t)\mathbb{Z}) \hookrightarrow \mathbb{Z}/((t^{m+1} - 1)\mathbb{Z})$ and let $x(t) = \sum_{i=0}^{m} x_it^i$ and $y(t) = \sum_{i=0}^{m} y_it^i$ be residues. Then modulo $t^{m+1} - 1$, we have

$$x(t)y(t) = z(t) \text{ with } z_i = \sum_{j=0}^{m} x_{\langle i-j \rangle} y_{\langle j \rangle}.$$
Generalised Repunit Primes

**Definition**

For $m + 1$ an odd prime and $t$ an integer let

$$p = \Phi_{m+1}(t) = t^m + t^{m-1} + \cdots + t + 1.$$  

If prime, we call $p$ a Generalised Repunit Prime.

Embed $\mathbb{Z}/(\Phi_{m+1}(t)\mathbb{Z}) \hookrightarrow \mathbb{Z}/((t^{m+1} - 1)\mathbb{Z})$ and let $x(t) = \sum_{i=0}^{m} x_i t^i$ and $y(t) = \sum_{i=0}^{m} y_i t^i$ be residues. Then modulo $t^{m+1} - 1$, we have

$$x(t)y(t) = z(t) \text{ with } z_i = \sum_{j=0}^{m} x_{\langle i-j \rangle} y_{\langle j \rangle}.$$  

- Cost is $(m + 1)^2 M + 2m(m + 1)A$
**Algorithm : GRP Multiplication**

**INPUT:** \( x = \sum_{i=0}^{m} x_i t^i, \ y = \sum_{i=0}^{m} y_i t^i \)

**OUTPUT:** \( z = \sum_{i=0}^{m} z_i t^i \) where \( z \equiv x y \pmod{\Phi_{m+1}(t)} \)

1. For \( i = m \) to 0 do:
2. \( z_i \leftarrow \sum_{j=1}^{m/2} (x_{\langle i-j \rangle} - x_{\langle i+j \rangle}) (y_{\langle i+j \rangle} - y_{\langle i-j \rangle}) \)
3. Return \( z \)
**Algorithm: GRP Multiplication**

**INPUT:** \( x = \sum_{i=0}^{m} x_i t^i \), \( y = \sum_{i=0}^{m} y_i t^i \)

**OUTPUT:** \( z = \sum_{i=0}^{m} z_i t^i \) where \( z \equiv x \cdot y \pmod{\Phi_{m+1}(t)} \)

1. For \( i = m \) to 0 do:
2. \( z_i \leftarrow \sum_{j=1}^{m/2} (x_{\langle i/2-j \rangle} - x_{\langle i/2+j \rangle})(y_{\langle i/2+j \rangle} - y_{\langle i/2-j \rangle}) \)
3. Return \( z \)

- Cost now is \( \frac{m(m+1)}{2} M + 2(m^2 - 1)A \)
**Algorithm: GRP Multiplication**

**Input:**  
\[ x = \sum_{i=0}^{m} x_i t^i, \quad y = \sum_{i=0}^{m} y_i t^i \]

**Output:**  
\[ z = \sum_{i=0}^{m} z_i t^i \] where \( z \equiv x y \pmod{\Phi_{m+1}(t)} \)

1. For \( i = m \) to 0 do:
2. \( z_i \leftarrow \sum_{j=1}^{m/2} (x_{\langle i-j \rangle} - x_{\langle i+j \rangle})(y_{\langle i+j \rangle} - y_{\langle i-j \rangle}) \)
3. Return \( z \)

- Cost now is \( \frac{m(m+1)}{2} M + 2(m^2 - 1)A \)
**Algorithm: GRP Multiplication**

**Input:** $x = \sum_{i=0}^{m} x_i t^i$, $y = \sum_{i=0}^{m} y_i t^i$

**Output:** $z = \sum_{i=0}^{m} z_i t^i$ where $z \equiv x \cdot y \pmod{\Phi m+1(t)}$

1. For $i = m$ to 0 do:
2. \[ z_i \leftarrow \sum_{j=1}^{m/2} (x_{\langle i^2-j \rangle} - x_{\langle i^2+j \rangle})(y_{\langle i^2+j \rangle} - y_{\langle i^2-j \rangle}) \]
3. Return $z$

- Cost now is $\frac{m(m+1)}{2}M + 2(m^2 - 1)A$
- **Drawback:** Except for $p = 2^{521} - 1 = 2^{520} + 2^{519} + \ldots + 2 + 1$, GRPs are not standardised...
Overview

ECC efficiency

Generalised Repunit Primes

This work
Application to $p = 2^{521} - 1$

On 64-bit architectures residues mod $p$ require $\lceil 521/64 \rceil = 9$ words, so assume modulus is $t^9 - 1$. Let $x(t) = \sum_{i=0}^{8} x_i t^i = \bar{x} = [x_0, \ldots, x_8]$, $y(t) = \sum_{i=0}^{8} y_i t^i = \bar{y} = [y_0, \ldots, y_8]$, & $\bar{z} \equiv \bar{x} \bar{y}$ (mod $t^9 - 1$).
Application to $p = 2^{521} - 1$

On 64-bit architectures residues mod $p$ require $\lceil 521/64 \rceil = 9$ words, so assume modulus is $t^9 - 1$. Let $x(t) = \sum_{i=0}^{8} x_i t^i = \overline{x} = [x_0, \ldots, x_8]$, $y(t) = \sum_{i=0}^{8} y_i t^i = \overline{y} = [y_0, \ldots, y_8]$, & $\overline{z} \equiv \overline{x} \overline{y} \pmod{t^9 - 1}$. Then $\overline{z} = [x_0 y_0 + x_1 y_8 + x_2 y_7 + x_3 y_6 + x_4 y_5 + x_5 y_4 + x_6 y_3 + x_7 y_2 + x_8 y_1,$

$x_0 y_1 + x_1 y_0 + x_2 y_8 + x_3 y_7 + x_4 y_6 + x_5 y_5 + x_6 y_4 + x_7 y_3 + x_8 y_2,$

$x_0 y_2 + x_1 y_1 + x_2 y_0 + x_3 y_8 + x_4 y_7 + x_5 y_6 + x_6 y_5 + x_7 y_4 + x_8 y_3,$

$x_0 y_3 + x_1 y_2 + x_2 y_1 + x_3 y_0 + x_4 y_8 + x_5 y_7 + x_6 y_6 + x_7 y_5 + x_8 y_4,$

$x_0 y_4 + x_1 y_3 + x_2 y_2 + x_3 y_1 + x_4 y_0 + x_5 y_8 + x_6 y_7 + x_7 y_6 + x_8 y_5,$

$x_0 y_5 + x_1 y_4 + x_2 y_3 + x_3 y_2 + x_4 y_1 + x_5 y_0 + x_6 y_8 + x_7 y_7 + x_8 y_6,$

$x_0 y_6 + x_1 y_5 + x_2 y_4 + x_3 y_3 + x_4 y_2 + x_5 y_1 + x_6 y_0 + x_7 y_8 + x_8 y_7,$

$x_0 y_7 + x_1 y_6 + x_2 y_5 + x_3 y_4 + x_4 y_3 + x_5 y_2 + x_6 y_1 + x_7 y_0 + x_8 y_8,$

$x_0 y_8 + x_1 y_7 + x_2 y_6 + x_3 y_5 + x_4 y_4 + x_5 y_3 + x_6 y_2 + x_7 y_1 + x_8 y_0].$
Application to $p = 2^{521} - 1$

On 64-bit architectures residues mod $p$ require $\lceil 521/64 \rceil = 9$ words, so assume modulus is $t^9 - 1$. Let $x(t) = \sum_{i=0}^{8} x_i t^i = \overline{x} = [x_0, \ldots, x_8]$, $y(t) = \sum_{i=0}^{8} y_i t^i = \overline{y} = [y_0, \ldots, y_8]$, & $\overline{z} \equiv \overline{x} \overline{y} \pmod{t^9 - 1}$. Then $\overline{z} = [x_0 y_0 + x_1 y_8 + x_2 y_7 + x_3 y_6 + x_4 y_5 + x_5 y_4 + x_6 y_3 + x_7 y_2 + x_8 y_1,$

$x_0 y_1 + x_1 y_0 + x_2 y_8 + x_3 y_7 + x_4 y_6 + x_5 y_5 + x_6 y_4 + x_7 y_3 + x_8 y_2,$

$x_0 y_2 + x_1 y_1 + x_2 y_0 + x_3 y_8 + x_4 y_7 + x_5 y_6 + x_6 y_5 + x_7 y_4 + x_8 y_3,$

$x_0 y_3 + x_1 y_2 + x_2 y_1 + x_3 y_0 + x_4 y_8 + x_5 y_7 + x_6 y_6 + x_7 y_5 + x_8 y_4,$

$x_0 y_4 + x_1 y_3 + x_2 y_2 + x_3 y_1 + x_4 y_0 + x_5 y_8 + x_6 y_7 + x_7 y_6 + x_8 y_5,$

$x_0 y_5 + x_1 y_4 + x_2 y_3 + x_3 y_2 + x_4 y_1 + x_5 y_0 + x_6 y_8 + x_7 y_7 + x_8 y_6,$

$x_0 y_6 + x_1 y_5 + x_2 y_4 + x_3 y_3 + x_4 y_2 + x_5 y_1 + x_6 y_0 + x_7 y_8 + x_8 y_7,$

$x_0 y_7 + x_1 y_6 + x_2 y_5 + x_3 y_4 + x_4 y_3 + x_5 y_2 + x_6 y_1 + x_7 y_0 + x_8 y_8,$

$x_0 y_8 + x_1 y_7 + x_2 y_6 + x_3 y_5 + x_4 y_4 + x_5 y_3 + x_6 y_2 + x_7 y_1 + x_8 y_0].$

- Cost is $81M + 144A$
Application to \( p = 2^{521} - 1 \)

Let \( s = \sum_{i=0}^{8} x_i y_i \).
Application to $\rho = 2^{521} - 1$

Let $s = \sum_{i=0}^{8} x_i y_i$. Then $\bar{z}$ may also be expressed as

\[
[s - (x_1 - x_8)(y_1 - y_8) - (x_2 - x_7)(y_2 - y_7) - (x_3 - x_6)(y_3 - y_6) - (x_4 - x_5)(y_4 - y_5),
\]
\[
s - (x_1 - x_0)(y_1 - y_0) - (x_2 - x_8)(y_2 - y_8) - (x_3 - x_7)(y_3 - y_7) - (x_4 - x_6)(y_4 - y_6),
\]
\[
s - (x_5 - x_6)(y_5 - y_6) - (x_2 - x_0)(y_2 - y_0) - (x_3 - x_8)(y_3 - y_8) - (x_4 - x_7)(y_4 - y_7),
\]
\[
s - (x_5 - x_7)(y_5 - y_7) - (x_2 - x_1)(y_2 - y_1) - (x_3 - x_0)(y_3 - y_0) - (x_4 - x_8)(y_4 - y_8),
\]
\[
s - (x_5 - x_8)(y_5 - y_8) - (x_6 - x_7)(y_6 - y_7) - (x_3 - x_1)(y_3 - y_1) - (x_4 - x_0)(y_4 - y_0),
\]
\[
s - (x_5 - x_0)(y_5 - y_0) - (x_6 - x_8)(y_6 - y_8) - (x_3 - x_2)(y_3 - y_2) - (x_4 - x_1)(y_4 - y_1),
\]
\[
s - (x_5 - x_1)(y_5 - y_1) - (x_6 - x_0)(y_6 - y_0) - (x_7 - x_8)(y_7 - y_8) - (x_4 - x_2)(y_4 - y_2),
\]
\[
s - (x_5 - x_2)(y_5 - y_2) - (x_6 - x_1)(y_6 - y_1) - (x_7 - x_0)(y_7 - y_0) - (x_4 - x_3)(y_4 - y_3),
\]
\[
s - (x_5 - x_3)(y_5 - y_3) - (x_6 - x_2)(y_6 - y_2) - (x_7 - x_1)(y_7 - y_1) - (x_8 - x_0)(y_8 - y_0)].
\]
Application to $p = 2^{521} - 1$

Let $s = \sum_{i=0}^{8} x_i y_i$. Then $\overline{z}$ may also be expressed as

\[
\begin{align*}
[s - (x_1 - x_8)(y_1 - y_8) - (x_2 - x_7)(y_2 - y_7) - (x_3 - x_6)(y_3 - y_6) - (x_4 - x_5)(y_4 - y_5), \\
s - (x_1 - x_0)(y_1 - y_0) - (x_2 - x_8)(y_2 - y_8) - (x_3 - x_7)(y_3 - y_7) - (x_4 - x_6)(y_4 - y_6), \\
s - (x_5 - x_6)(y_5 - y_6) - (x_2 - x_0)(y_2 - y_0) - (x_3 - x_8)(y_3 - y_8) - (x_4 - x_7)(y_4 - y_7), \\
s - (x_5 - x_7)(y_5 - y_7) - (x_2 - x_1)(y_2 - y_1) - (x_3 - x_0)(y_3 - y_0) - (x_4 - x_8)(y_4 - y_8), \\
s - (x_5 - x_8)(y_5 - y_8) - (x_6 - x_7)(y_6 - y_7) - (x_3 - x_1)(y_3 - y_1) - (x_4 - x_0)(y_4 - y_0), \\
s - (x_5 - x_0)(y_5 - y_0) - (x_6 - x_8)(y_6 - y_8) - (x_3 - x_2)(y_3 - y_2) - (x_4 - x_1)(y_4 - y_1), \\
s - (x_5 - x_1)(y_5 - y_1) - (x_6 - x_0)(y_6 - y_0) - (x_7 - x_8)(y_7 - y_8) - (x_4 - x_2)(y_4 - y_2), \\
s - (x_5 - x_2)(y_5 - y_2) - (x_6 - x_1)(y_6 - y_1) - (x_7 - x_0)(y_7 - y_0) - (x_4 - x_3)(y_4 - y_3), \\
s - (x_5 - x_3)(y_5 - y_3) - (x_6 - x_2)(y_6 - y_2) - (x_7 - x_1)(y_7 - y_1) - (x_8 - x_0)(y_8 - y_0)].
\end{align*}
\]

- Cost is now $45M + 160A$, exchanging $36M$ for $16A$
Application to $\rho = 2^{521} - 1$

Let $s = \sum_{i=0}^{8} x_i y_i$. Then $\bar{z}$ may also be expressed as

$$[s - (x_1 - x_8)(y_1 - y_8) - (x_2 - x_7)(y_2 - y_7) - (x_3 - x_6)(y_3 - y_6) - (x_4 - x_5)(y_4 - y_5),$$

$$s - (x_1 - x_0)(y_1 - y_0) - (x_2 - x_8)(y_2 - y_8) - (x_3 - x_7)(y_3 - y_7) - (x_4 - x_6)(y_4 - y_6),$$

$$s - (x_5 - x_6)(y_5 - y_6) - (x_2 - x_0)(y_2 - y_0) - (x_3 - x_8)(y_3 - y_8) - (x_4 - x_7)(y_4 - y_7),$$

$$s - (x_5 - x_7)(y_5 - y_7) - (x_2 - x_1)(y_2 - y_1) - (x_3 - x_0)(y_3 - y_0) - (x_4 - x_8)(y_4 - y_8),$$

$$s - (x_5 - x_8)(y_5 - y_8) - (x_6 - x_7)(y_6 - y_7) - (x_3 - x_1)(y_3 - y_1) - (x_4 - x_0)(y_4 - y_0),$$

$$s - (x_5 - x_0)(y_5 - y_0) - (x_6 - x_8)(y_6 - y_8) - (x_3 - x_2)(y_3 - y_2) - (x_4 - x_1)(y_4 - y_1),$$

$$s - (x_5 - x_1)(y_5 - y_1) - (x_6 - x_0)(y_6 - y_0) - (x_7 - x_8)(y_7 - y_8) - (x_4 - x_2)(y_4 - y_2),$$

$$s - (x_5 - x_2)(y_5 - y_2) - (x_6 - x_1)(y_6 - y_1) - (x_7 - x_0)(y_7 - y_0) - (x_4 - x_3)(y_4 - y_3),$$

$$s - (x_5 - x_3)(y_5 - y_3) - (x_6 - x_2)(y_6 - y_2) - (x_7 - x_1)(y_7 - y_1) - (x_8 - x_0)(y_8 - y_0)].$$

- Cost is now $45M + 160A$, exchanging $36M$ for $16A$
- However, we can’t use the irrational base $t = 2^{521/9}$ with integer coefficients, so instead work mod $2\rho = t^9 - 2$ with $t = 2^{58}$
Application to $\rho = 2^{521} - 1$

Let $s = \sum_{i=0}^{8} x_i y_i$. Then $\overline{z}$ may also be expressed as

$$[s - (x_1 - x_8)(y_1 - y_8) - (x_2 - x_7)(y_2 - y_7) - (x_3 - x_6)(y_3 - y_6) - (x_4 - x_5)(y_4 - y_5),$$
$$s - (x_1 - x_0)(y_1 - y_0) - (x_2 - x_8)(y_2 - y_8) - (x_3 - x_7)(y_3 - y_7) - (x_4 - x_6)(y_4 - y_6),$$
$$s - (x_5 - x_6)(y_5 - y_6) - (x_2 - x_0)(y_2 - y_0) - (x_3 - x_8)(y_3 - y_8) - (x_4 - x_7)(y_4 - y_7),$$
$$s - (x_5 - x_7)(y_5 - y_7) - (x_2 - x_1)(y_2 - y_1) - (x_3 - x_0)(y_3 - y_0) - (x_4 - x_8)(y_4 - y_8),$$
$$s - (x_5 - x_8)(y_5 - y_8) - (x_6 - x_7)(y_6 - y_7) - (x_3 - x_1)(y_3 - y_1) - (x_4 - x_0)(y_4 - y_0),$$
$$s - (x_5 - x_0)(y_5 - y_0) - (x_6 - x_8)(y_6 - y_8) - (x_3 - x_2)(y_3 - y_2) - (x_4 - x_1)(y_4 - y_1),$$
$$s - (x_5 - x_1)(y_5 - y_1) - (x_6 - x_0)(y_6 - y_0) - (x_7 - x_8)(y_7 - y_8) - (x_4 - x_2)(y_4 - y_2),$$
$$s - (x_5 - x_2)(y_5 - y_2) - (x_6 - x_1)(y_6 - y_1) - (x_7 - x_0)(y_7 - y_0) - (x_4 - x_3)(y_4 - y_3),$$
$$s - (x_5 - x_3)(y_5 - y_3) - (x_6 - x_2)(y_6 - y_2) - (x_7 - x_1)(y_7 - y_1) - (x_8 - x_0)(y_8 - y_0)].$$

- Cost is now $45M + 160A$, exchanging $36M$ for $16A$
- However, we can't use the irrational base $t = 2^{521/9}$ with integer coefficients, so instead work mod $2\rho = t^9 - 2$ with $t = 2^{58}$
- Introduces several shifts, but still only requires $45M$
Implementation Results

The Edwards curve E-521: $x^2 + y^2 = 1 - 376014x^2y^2$ was found independently by Bernstein-Lange, Hamburg, and Aranha et al.

---

For our code see indigo.ie/~mscott/ws521.cpp and indigo.ie/~mscott/ed521.cpp respectively.

Hamburg has obtained even better figures for E-521: about 800 kcycles using two Karatsuba levels and low level optimisations.
Implementation Results

The Edwards curve E-521: \( x^2 + y^2 = 1 - 376014x^2y^2 \) was found independently by Bernstein-Lange, Hamburg, and Aranha et al.

We implemented constant-time cache-safe variable-base scalar multiplication on NIST curve P-521 & E-521 in C.
Implementation Results

The Edwards curve E-521: \( x^2 + y^2 = 1 - 376014x^2y^2 \) was found independently by Bernstein-Lange, Hamburg, and Aranha et al.

We implemented constant-time cache-safe variable-base scalar multiplication on NIST curve P-521 & E-521 in C.

<table>
<thead>
<tr>
<th></th>
<th>openSSL</th>
<th>P-521</th>
<th>ed-521-mers</th>
<th>E-521</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1,319,000</td>
<td>1,073,000</td>
<td>1,552,000</td>
<td>943,000</td>
</tr>
</tbody>
</table>

Table: Cycle counts for openSSL 1.0.2-beta2, P-521 and E-521 on a 3.4GHz Intel Haswell Core i7-4770 compiled with gcc 4.7 on Ubuntu 12.04, while ed-521-mers was on a 3.4GHz Intel Core i7-2600 Sandy Bridge (Bos et al.)
Implementation Results

The Edwards curve E-521: \( x^2 + y^2 = 1 - 376014x^2y^2 \) was found independently by Bernstein-Lange, Hamburg, and Aranha et al.

We implemented constant-time cache-safe variable-base scalar multiplication on NIST curve P-521 & E-521 in C.

<table>
<thead>
<tr>
<th></th>
<th>openssl</th>
<th>P-521</th>
<th>ed-521-mers</th>
<th>E-521</th>
</tr>
</thead>
<tbody>
<tr>
<td>openSSL</td>
<td>1,319,000</td>
<td>1,073,000</td>
<td>1,552,000</td>
<td>943,000</td>
</tr>
</tbody>
</table>

Table: Cycle counts for openssl 1.0.2-beta2, P-521 and E-521 on a 3.4GHz Intel Haswell Core i7-4770 compiled with gcc 4.7 on Ubuntu 12.04, while ed-521-mers was on a 3.4GHz Intel Core i7-2600 Sandy Bridge (Bos et al.)

- For our code see indigo.ie/~mscott/ws521.cpp and indigo.ie/~mscott/ed521.cpp respectively
Implementation Results

The Edwards curve E-521: \( x^2 + y^2 = 1 - 376014x^2y^2 \) was found independently by Bernstein-Lange, Hamburg, and Aranha et al.

We implemented constant-time cache-safe variable-base scalar multiplication on NIST curve P-521 & E-521 in C.

<table>
<thead>
<tr>
<th>openSSL</th>
<th>P-521</th>
<th>ed-521-mers</th>
<th>E-521</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,319,000</td>
<td>1,073,000</td>
<td>1,552,000</td>
<td>943,000</td>
</tr>
</tbody>
</table>

Table: Cycle counts for openSSL 1.0.2-beta2, P-521 and E-521 on a 3.4GHz Intel Haswell Core i7-4770 compiled with gcc 4.7 on Ubuntu 12.04, while ed-521-mers was on a 3.4GHz Intel Core i7-2600 Sandy Bridge (Bos et al.)

- For our code see indigo.ie/~mscott/ws521.cpp and indigo.ie/~mscott/ed521.cpp respectively
- Hamburg has obtained even better figures for E-521: about 800k cycles using two Karatsuba levels and low level optimisations
Summary

- Presented modular multiplication formulae for Crandall numbers that requires as few $M$ as is needed for squaring

Thanks for your attention!
Summary

- Presented modular multiplication formulae for Crandall numbers that requires as few $M$ as is needed for squaring.
- Efficiency of idea on ARM processors should be interesting due to higher $M/A$ cost ratio.

Thanks for your attention!
Summary

- Presented modular multiplication formulae for Crandall numbers that requires as few $M$ as is needed for squaring.
- Efficiency of idea on ARM processors should be interesting due to higher $M/A$ cost ratio.
- Contributed to the debate regarding E-521 feasibility for independent standardisation (see CFRG).

Thanks for your attention!
Summary

• Presented modular multiplication formulae for Crandall numbers that requires as few $M$ as is needed for squaring
• Efficiency of idea on ARM processors should be interesting due to higher $M/A$ cost ratio
• Contributed to the debate regarding E-521 feasibility for independent standardisation (see CFRG)

Thanks for your attention!