Sandy2x: New Curve25519 Speed Records

Tung Chou

Technische Universiteit Eindhoven, The Netherlands

October 13, 2015
X25519 and Ed25519

X25519

- ECDH scheme
- public keys and shared secrets are points on the Montgomery curve
  \[ y^2 = x^3 + 486662x^2 + x \]
  over \( \mathbb{F}_{2^{255} - 19} \)
- by Bernstein, 2006

Ed25519

- signature scheme
- public keys and (part of) signatures are points on the twisted Edwards curve
  \[ -x^2 + y^2 = 1 - \frac{121665}{121666}x^2y^2 \]
  over \( \mathbb{F}_{2^{255} - 19} \)
- by Bernstein, Duif, Lange, Schwabe, and Yang, 2011
The ECC implementation pyramid

- Big-integer or polynomial arithmetic
- Finite-field arithmetic
- ECC add/double
- Scalar multiplication

(slide credit: Peter Schwabe)
The big multiplier

• used in all papers about ECC speeds on Intel microarchitectures

• $64 \times 64 \rightarrow 128$-bit multiplication in one instruction ($\text{mul}$)

(This talk focuses on Sandy Bridge/Ivy Bridge)
The big multiplier

- used in all papers about ECC speeds on Intel microarchitectures
The big multiplier

- used in all papers about ECC speeds on Intel microarchitectures
- \(64 \times 64 \rightarrow 128\)-bit multiplication in one instruction (\texttt{mul})
The big multiplier

- used in all papers about ECC speeds on Intel microarchitectures
- $64 \times 64 \rightarrow$ 128-bit multiplication in one instruction ($\text{mul}$)
- (This talk focuses on Sandy Bridge/Ivy Bridge)
The radix-$2^{51}$ representation for $F_{2^{255} - 19}$
The radix-$2^{51}$ representation for $\mathbb{F}_{2^{255}-19}$

\[ f = f_0 + f_12^{51} + f_22^{102} + f_32^{153} + f_42^{204} \]
The radix-$2^{51}$ representation for $\mathbb{F}_{2^{255} - 19}$

$$
\begin{align*}
  f &= f_0 + f_1 2^{51} + f_2 2^{102} + f_3 2^{153} + f_4 2^{204} \\
  g &= g_0 + g_1 2^{51} + g_2 2^{102} + g_3 2^{153} + g_4 2^{204}
\end{align*}
$$
The radix-$2^{51}$ representation for $\mathbb{F}_{2^{255}-19}$

\[
\begin{align*}
f &= f_0 + f_1 2^{51} + f_2 2^{102} + f_3 2^{153} + f_4 2^{204} \\
g &= g_0 + g_1 2^{51} + g_2 2^{102} + g_3 2^{153} + g_4 2^{204}
\end{align*}
\]

\[
\begin{align*}
h_0 &= f_0 g_0 + 19 f_1 g_4 + 19 f_2 g_3 + 19 f_3 g_2 + 19 f_4 g_1 \\
h_1 &= f_0 g_1 + f_1 g_0 + 19 f_2 g_4 + 19 f_3 g_3 + 19 f_4 g_2 \\
h_2 &= f_0 g_2 + f_1 g_1 + f_2 g_0 + 19 f_3 g_4 + 19 f_4 g_3 \\
h_3 &= f_0 g_3 + f_1 g_2 + f_2 g_1 + f_3 g_0 + 19 f_4 g_4 \\
h_4 &= f_0 g_4 + f_1 g_3 + f_2 g_2 + f_3 g_1 + f_4 g_0
\end{align*}
\]
The radix-$2^{51}$ representation for $\mathbb{F}_{2^{255} - 19}$

\[
\begin{align*}
    f &= f_0 + f_1 2^{51} + f_2 2^{102} + f_3 2^{153} + f_4 2^{204} \\
    g &= g_0 + g_1 2^{51} + g_2 2^{102} + g_3 2^{153} + g_4 2^{204}
\end{align*}
\]

\[
\begin{align*}
    h_0 &= f_0 g_0 + 19 f_1 g_4 + 19 f_2 g_3 + 19 f_3 g_2 + 19 f_4 g_1 \\
    h_1 &= f_0 g_1 + f_1 g_0 + 19 f_2 g_4 + 19 f_3 g_3 + 19 f_4 g_2 \\
    h_2 &= f_0 g_2 + f_1 g_1 + f_2 g_0 + 19 f_3 g_4 + 19 f_4 g_3 \\
    h_3 &= f_0 g_3 + f_1 g_2 + f_2 g_1 + f_3 g_0 + 19 f_4 g_4 \\
    h_4 &= f_0 g_4 + f_1 g_3 + f_2 g_2 + f_3 g_1 + f_4 g_0
\end{align*}
\]

- 25 multiplication instructions + overhead.
The radix-2\(^{51}\) representation for \(\mathbb{F}_{2^{255}}\) - 19

\[ f = f_0 + f_12^{51} + f_22^{102} + f_32^{153} + f_42^{204} \]

\[ g = g_0 + g_12^{51} + g_22^{102} + g_32^{153} + g_42^{204} \]

\[ h_0 = f_0g_0 + 19f_1g_4 + 19f_2g_3 + 19f_3g_2 + 19f_4g_1 \]

\[ h_1 = f_0g_1 + f_1g_0 + 19f_2g_4 + 19f_3g_3 + 19f_4g_2 \]

\[ h_2 = f_0g_2 + f_1g_1 + f_2g_0 + 19f_3g_4 + 19f_4g_3 \]

\[ h_3 = f_0g_3 + f_1g_2 + f_2g_1 + f_3g_0 + 19f_4g_4 \]

\[ h_4 = f_0g_4 + f_1g_3 + f_2g_2 + f_3g_1 + f_4g_0 \]

- 25 multiplication instructions + overhead.
- some carries required.
A small multiplier

- a 2-way vectorized multiplier
- $32 \times 32 \rightarrow 64$-bit multiplications in one instruction ($vpmuludq$)
- usage: $((a_0 b_0, a_1 b_1) = (a_0, a_1) \times (b_0, b_1)$
A small multiplier

- a 2-way vectorized multiplier
A small multiplier

- a 2-way vectorized multiplier
- $32 \times 32 \rightarrow$ 64-bit multiplications in one instruction ($vpmuludq$)
A small multiplier

- a 2-way vectorized multiplier
- $32 \times 32 \rightarrow$ 64-bit multiplications in one instruction ($\text{vpmuludq}$)
- usage:

$$(a_0 b_0, a_1 b_1) = (a_0, a_1) \times (b_0, b_1)$$
The radix-$2^{25.5}$ representation for $F_{2^{255}-19}$
The radix-$2^{25.5}$ representation for $\mathbb{F}_2^{255-19}$

$$f = f_0 + f_1 2^{26} + f_2 2^{51} + f_3 2^{77} + f_4 2^{102} + f_5 2^{128} + f_6 2^{153} + f_7 2^{179} + f_8 2^{204} + f_9 2^{230}$$

$$g = g_0 + g_1 2^{26} + g_2 2^{51} + g_3 2^{77} + g_4 2^{102} + g_5 2^{128} + g_6 2^{153} + g_7 2^{179} + g_8 2^{204} + g_9 2^{230}$$
The radix-$2^{25.5}$ representation for $\mathbb{F}_{2^{255}-19}$

\[
f = f_0 + f_1 2^{26} + f_2 2^{51} + f_3 2^{77} + f_4 2^{102} + f_5 2^{128} + f_6 2^{153} + f_7 2^{179} + f_8 2^{204} + f_9 2^{230}
g = g_0 + g_1 2^{26} + g_2 2^{51} + g_3 2^{77} + g_4 2^{102} + g_5 2^{128} + g_6 2^{153} + g_7 2^{179} + g_8 2^{204} + g_9 2^{230}
\]

\[
h_0 = f_0 g_0 + 38 f_1 g_9 + 19 f_2 g_8 + 38 f_3 g_7 + 19 f_4 g_6 + 38 f_5 g_5 + 19 f_6 g_4 + 38 f_7 g_3 + 19 f_8 g_2 + 38 f_9 g_1
\]
\[
h_1 = f_0 g_1 + f_1 g_0 + 19 f_2 g_9 + 19 f_3 g_8 + 19 f_4 g_7 + 19 f_5 g_6 + 19 f_6 g_5 + 19 f_7 g_4 + 19 f_8 g_3 + 19 f_9 g_2
\]
\[
h_2 = f_0 g_2 + 2 f_1 g_1 + f_2 g_0 + 19 f_3 g_9 + 19 f_4 g_8 + 38 f_5 g_7 + 19 f_6 g_6 + 19 f_7 g_5 + 38 f_8 g_4 + 19 f_9 g_3
\]
\[
h_3 = f_0 g_3 + f_1 g_2 + f_2 g_1 + f_3 g_0 + 19 f_4 g_9 + 19 f_5 g_8 + 19 f_6 g_7 + 19 f_7 g_6 + 19 f_8 g_5 + 19 f_9 g_4
\]
\[
h_4 = f_0 g_4 + 2 f_1 g_3 + f_2 g_2 + 2 f_3 g_1 + f_4 g_0 + 19 f_5 g_9 + 38 f_6 g_8 + 19 f_7 g_7 + 38 f_8 g_6 + 19 f_9 g_5
\]
\[
h_5 = f_0 g_5 + f_1 g_4 + f_2 g_3 + f_3 g_2 + f_4 g_1 + f_5 g_0 + 19 f_6 g_9 + 19 f_7 g_8 + 19 f_8 g_7 + 19 f_9 g_6
\]
\[
h_6 = f_0 g_6 + 2 f_1 g_5 + f_2 g_4 + 2 f_3 g_3 + f_4 g_2 + 2 f_5 g_1 + 19 f_6 g_0 + 38 f_7 g_9 + 19 f_8 g_8 + 38 f_9 g_7
\]
\[
h_7 = f_0 g_7 + f_1 g_6 + f_2 g_5 + f_3 g_4 + f_4 g_3 + f_5 g_2 + 19 f_6 g_1 + f_7 g_0 + 19 f_8 g_9 + 19 f_9 g_8
\]
\[
h_8 = f_0 g_8 + 2 f_1 g_7 + f_2 g_6 + 2 f_3 g_5 + f_4 g_4 + 2 f_5 g_3 + 19 f_6 g_2 + 2 f_7 g_1 + 19 f_8 g_0 + 38 f_9 g_9
\]
\[
h_9 = f_0 g_9 + f_1 g_8 + f_2 g_7 + f_3 g_6 + f_4 g_5 + f_5 g_4 + f_6 g_3 + f_7 g_2 + f_8 g_1 + f_9 g_0
\]
The radix-$2^{25.5}$ representation for $\mathbb{F}_{2^{255}-19}$

\[
\begin{align*}
  f &= f_0 + f_1 2^{26} + f_2 2^{51} + f_3 2^{77} + f_4 2^{102} + f_5 2^{128} + f_6 2^{153} + f_7 2^{179} + f_8 2^{204} + f_9 2^{230} \\
  g &= g_0 + g_1 2^{26} + g_2 2^{51} + g_3 2^{77} + g_4 2^{102} + g_5 2^{128} + g_6 2^{153} + g_7 2^{179} + g_8 2^{204} + g_9 2^{230}
\end{align*}
\]

\[
\begin{align*}
  h_0 &= f_0 g_0 + 38 f_1 g_9 + 19 f_2 g_8 + 38 f_3 g_7 + 19 f_4 g_6 + 38 f_5 g_5 + 19 f_6 g_4 + 38 f_7 g_3 + 19 f_8 g_2 + 38 f_9 g_1 \\
  h_1 &= f_0 g_1 + f_1 g_0 + 19 f_2 g_9 + 19 f_3 g_8 + 19 f_4 g_7 + 19 f_5 g_6 + 19 f_6 g_5 + 19 f_7 g_4 + 19 f_8 g_3 + 19 f_9 g_2 \\
  h_2 &= f_0 g_2 + 2 f_1 g_1 + f_2 g_0 + 38 f_3 g_9 + 19 f_4 g_8 + 38 f_5 g_7 + 19 f_6 g_6 + 38 f_7 g_5 + 19 f_8 g_4 + 38 f_9 g_3 \\
  h_3 &= f_0 g_3 + f_1 g_2 + f_2 g_1 + f_3 g_0 + 19 f_4 g_9 + 19 f_5 g_8 + 19 f_6 g_7 + 19 f_7 g_6 + 19 f_8 g_5 + 19 f_9 g_4 \\
  h_4 &= f_0 g_4 + 2 f_1 g_3 + f_2 g_2 + 2 f_3 g_1 + f_4 g_0 + 38 f_5 g_9 + 19 f_6 g_8 + 38 f_7 g_7 + 19 f_8 g_6 + 38 f_9 g_5 \\
  h_5 &= f_0 g_5 + f_1 g_4 + f_2 g_3 + f_3 g_2 + f_4 g_1 + f_5 g_0 + 19 f_6 g_9 + 19 f_7 g_8 + 19 f_8 g_7 + 19 f_9 g_6 \\
  h_6 &= f_0 g_6 + 2 f_1 g_5 + f_2 g_4 + 2 f_3 g_3 + f_4 g_2 + 2 f_5 g_1 + f_6 g_0 + 38 f_7 g_9 + 19 f_8 g_8 + 38 f_9 g_7 \\
  h_7 &= f_0 g_7 + f_1 g_6 + f_2 g_5 + f_3 g_4 + f_4 g_3 + f_5 g_2 + f_6 g_1 + f_7 g_0 + 19 f_8 g_9 + 19 f_9 g_8 \\
  h_8 &= f_0 g_8 + 2 f_1 g_7 + f_2 g_6 + 2 f_3 g_5 + f_4 g_4 + 2 f_5 g_3 + f_6 g_2 + 2 f_7 g_1 + f_8 g_0 + 38 f_9 g_9 \\
  h_9 &= f_0 g_9 + f_1 g_8 + f_2 g_7 + f_3 g_6 + f_4 g_5 + f_5 g_4 + f_6 g_3 + f_7 g_2 + f_8 g_1 + f_9 g_0
\end{align*}
\]

- 100 multiplication instructions $+$ overhead; 50 per multiplication.
- some carries required.
The radix-$2^{25.5}$ representation for $\mathbb{F}_{2^{255}-19}$

\[
\begin{align*}
  f &= f_0 + f_1 2^{26} + f_2 2^{51} + f_3 2^{77} + f_4 2^{102} + f_5 2^{128} + f_6 2^{153} + f_7 2^{179} + f_8 2^{204} + f_9 2^{230} \\
  g &= g_0 + g_1 2^{26} + g_2 2^{51} + g_3 2^{77} + g_4 2^{102} + g_5 2^{128} + g_6 2^{153} + g_7 2^{179} + g_8 2^{204} + g_9 2^{230}
\end{align*}
\]

\[
\begin{align*}
  h_0 &= f_0 g_0 + 38 f_1 g_9 + 19 f_2 g_8 + 38 f_3 g_7 + 19 f_4 g_6 + 38 f_5 g_5 + 19 f_6 g_4 + 38 f_7 g_3 + 19 f_8 g_2 + 38 f_9 g_1 \\
  h_1 &= f_0 g_1 + f_1 g_0 + 19 f_2 g_9 + 19 f_3 g_8 + 19 f_4 g_7 + 19 f_5 g_6 + 19 f_6 g_5 + 19 f_7 g_4 + 19 f_8 g_3 + 19 f_9 g_2 \\
  h_2 &= f_0 g_2 + 2 f_1 g_1 + f_2 g_0 + 38 f_3 g_9 + 19 f_4 g_8 + 38 f_5 g_7 + 19 f_6 g_6 + 38 f_7 g_5 + 19 f_8 g_4 + 38 f_9 g_3 \\
  h_3 &= f_0 g_3 + f_1 g_2 + f_2 g_1 + f_3 g_0 + 19 f_4 g_9 + 19 f_5 g_8 + 19 f_6 g_7 + 19 f_7 g_6 + 19 f_8 g_5 + 19 f_9 g_4 \\
  h_4 &= f_0 g_4 + 2 f_1 g_3 + f_2 g_2 + 2 f_3 g_1 + f_4 g_0 + 38 f_5 g_9 + 19 f_6 g_8 + 38 f_7 g_7 + 19 f_8 g_6 + 38 f_9 g_5 \\
  h_5 &= f_0 g_5 + f_1 g_4 + f_2 g_3 + f_3 g_2 + f_4 g_1 + f_5 g_0 + 19 f_6 g_9 + 19 f_7 g_8 + 19 f_8 g_7 + 19 f_9 g_6 \\
  h_6 &= f_0 g_6 + 2 f_1 g_5 + f_2 g_4 + 2 f_3 g_3 + f_4 g_2 + 2 f_5 g_1 + f_6 g_0 + 38 f_7 g_9 + 19 f_8 g_8 + 38 f_9 g_7 \\
  h_7 &= f_0 g_7 + f_1 g_6 + f_2 g_5 + f_3 g_4 + f_4 g_3 + f_5 g_2 + f_6 g_1 + f_7 g_0 + 19 f_8 g_9 + 19 f_9 g_8 \\
  h_8 &= f_0 g_8 + 2 f_1 g_7 + f_2 g_6 + 2 f_3 g_5 + f_4 g_4 + 2 f_5 g_3 + f_6 g_2 + 2 f_7 g_1 + f_8 g_0 + 38 f_9 g_9 \\
  h_9 &= f_0 g_9 + f_1 g_8 + f_2 g_7 + f_3 g_6 + f_4 g_5 + f_5 g_4 + f_6 g_3 + f_7 g_2 + f_8 g_1 + f_9 g_0
\end{align*}
\]

- 100 multiplication instructions + overhead; 50 per multiplication.
- some carries required.

Sandy2x sets new speed records by using the vectorized multiplier.
## Performance results

<table>
<thead>
<tr>
<th></th>
<th>SB cycles</th>
<th>IB cycles</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>X25519 public-key generation</td>
<td>54 346</td>
<td>52 169</td>
<td>Sandy2x</td>
</tr>
<tr>
<td></td>
<td>61 828</td>
<td>57 612</td>
<td>[A. Moon]</td>
</tr>
<tr>
<td></td>
<td>194 165</td>
<td>182 876</td>
<td>[Ed25519]</td>
</tr>
<tr>
<td>X25519 shared secret computation</td>
<td>156 995</td>
<td>159 128</td>
<td>Sandy2x</td>
</tr>
<tr>
<td></td>
<td>194 036</td>
<td>182 708</td>
<td>[Ed25519]</td>
</tr>
<tr>
<td>Ed25519 public-key generation</td>
<td>57 164</td>
<td>54 901</td>
<td>Sandy2x</td>
</tr>
<tr>
<td></td>
<td>63 712</td>
<td>59 332</td>
<td>[A. Moon]</td>
</tr>
<tr>
<td></td>
<td>64 015</td>
<td>61 099</td>
<td>[Ed25519]</td>
</tr>
<tr>
<td>Ed25519 sign</td>
<td>63 526</td>
<td>59 949</td>
<td>Sandy2x</td>
</tr>
<tr>
<td></td>
<td>67 692</td>
<td>62 624</td>
<td>[A. Moon]</td>
</tr>
<tr>
<td></td>
<td>72 444</td>
<td>67 284</td>
<td>[Ed25519]</td>
</tr>
<tr>
<td>Ed25519 verification</td>
<td>205 741</td>
<td>198 406</td>
<td>Sandy2x</td>
</tr>
<tr>
<td></td>
<td>227 628</td>
<td>204 376</td>
<td>[A. Moon]</td>
</tr>
<tr>
<td></td>
<td>222 564</td>
<td>209 060</td>
<td>[Ed25519]</td>
</tr>
</tbody>
</table>

- Andrew Moon “floodyberry”,
  [https://github.com/floodyberry/ed25519-donna](https://github.com/floodyberry/ed25519-donna)
Why is vectorization better?

Ports
Why is vectorization better?

Ports

- one each SB/IB core there are 6 ports: Port 0, 1, 5 are for arithmetic.
Why is vectorization better?

Ports

- one each SB/IB core there are 6 ports: Port 0,1,5 are for arithmetic.
- each instruction is decomposed into microoperations (μops)
Why is vectorization better?

Ports
- one each SB/IB core there are 6 ports: Port 0,1,5 are for arithmetic.
- each instruction is decomposed into microoperations ($\mu$ops)
  - $\text{mul}$: 2 $\mu$ops, handled by Port 0 and 1.
Why is vectorization better?

Ports

- one each SB/IB core there are 6 ports: Port 0,1,5 are for arithmetic.
- each instruction is decomposed into *microoperations* ($\mu$ops)
  - *mul*: 2 $\mu$ops, handled by Port 0 and 1.
  - *vpmuludq*: 1 $\mu$op, handled by Port 0.
Why is vectorization better?

Ports
- one each SB/IB core there are 6 ports: Port 0, 1, 5 are for arithmetic.
- each instruction is decomposed into *microoperations* ($\mu$ops)
  - **mul**: 2 $\mu$ops, handled by Port 0 and 1.
  - **vpmuludq**: 1 $\mu$op, handled by Port 0.
  - **vpaddq**: 1 $\mu$op, handled by either Port 1 or Port 5.
Why is vectorization better?

Ports

- one each SB/IB core there are 6 ports: Port 0, 1, 5 are for arithmetic.
- each instruction is decomposed into microoperations ($\mu$ops)
  - $\text{mul}$: 2 $\mu$ops, handled by Port 0 and 1.
  - $\text{vpmuludq}$: 1 $\mu$op, handled by Port 0.
  - $\text{vpaddq}$: 1 $\mu$op, handled by either Port 1 or Port 5.
- port utilization gives a lower bound of cycle count
Why is vectorization better?

Using the vectorized multiplier

- 109 vpmuludq + 95 vpaddq
Why is vectorization better?

Using the vectorized multiplier

- 109 vpmuludq + 95 vpaddq
- lower bound: 109 cycles (dominated by Port 0)
Why is vectorization better?

Using the vectorized multiplier

- 109 vpmuludq + 95 vpaddq
- lower bound: 109 cycles (dominated by Port 0)
- actual cycle count 112 cycles (56 cycles per multiplication)
Why is vectorization better?

Using the vectorized multiplier
- 109 vpmuludq + 95 vpaddq
- lower bound: 109 cycles (dominated by Port 0)
- actual cycle count 112 cycles (56 cycles per multiplication)

Using the serial multiplier
- 25 mul + 4 imul + 20 add + 20 adc
Why is vectorization better?

Using the vectorized multiplier

- 109 vpmuludq + 95 vpaddq
- lower bound: 109 cycles (dominated by Port 0)
- actual cycle count 112 cycles (56 cycles per multiplication)

Using the serial multiplier

- 25 mul + 4 imul + 20 add + 20 adc
- lower bound: \((25 \cdot 2 + 4 + 20 + 20 \cdot 2)/3 = 38\)
Why is vectorization better?

Using the vectorized multiplier
• 109 vpmuludq + 95 vpaddq
• lower bound: 109 cycles (dominated by Port 0)
• actual cycle count 112 cycles (56 cycles per multiplication)

Using the serial multiplier
• 25 mul + 4 imul + 20 add + 20 adc
• lower bound: \((25 \cdot 2 + 4 + 20 + 20 \cdot 2)/3 = 38\)
• actual cycle count is much larger: 52 cycles
Why is vectorization better?

Using the vectorized multiplier
- 109 vpmuludq + 95 vpaddq
- lower bound: 109 cycles (dominated by Port 0)
- actual cycle count 112 cycles (56 cycles per multiplication)

Using the serial multiplier
- 25 mul + 4 imul + 20 add + 20 adc
- lower bound: \((25 \cdot 2 + 4 + 20 + 20 \cdot 2)/3 = 38\)
- actual cycle count is much larger: 52 cycles
- perf-stat shows that the core fails to distribute the \(\mu\)ops equally over the ports
Why is vectorization better?

More reasons
Why is vectorization better?

More reasons

- carries take more cycles when using the serial multiplier

<table>
<thead>
<tr>
<th></th>
<th>$M^{-}$</th>
<th>$M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>serial</td>
<td>52</td>
<td>68</td>
</tr>
<tr>
<td>vectorized</td>
<td>56</td>
<td>69.5</td>
</tr>
</tbody>
</table>
Why is vectorization better?

More reasons

- carries take more cycles when using the serial multiplier

<table>
<thead>
<tr>
<th></th>
<th>M⁻</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>serial</td>
<td>52</td>
<td>68</td>
</tr>
<tr>
<td>vectorized</td>
<td>56</td>
<td>69.5</td>
</tr>
</tbody>
</table>

- batched squarings are faster
Why is vectorization better?

More reasons

• carries take more cycles when using the serial multiplier
  
<table>
<thead>
<tr>
<th></th>
<th>M^-</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>serial</td>
<td>52</td>
<td>68</td>
</tr>
<tr>
<td>vectorized</td>
<td>56</td>
<td>69.5</td>
</tr>
</tbody>
</table>

• batched squarings are faster
• instruction interleaving hides cost for addition/subtraction
Why is vectorization better?

More reasons

- carries take more cycles when using the serial multiplier

<table>
<thead>
<tr>
<th></th>
<th>$M^{-}$</th>
<th>$M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>serial</td>
<td>52</td>
<td>68</td>
</tr>
<tr>
<td>vectorized</td>
<td>56</td>
<td>69.5</td>
</tr>
</tbody>
</table>

- batched squarings are faster
- instruction interleaving hides cost for addition/subtraction
- constant-time table lookups are faster with vector instructions
Ending Remarks

The main messages of this talk:

• Vectorization should be considered on recent Intel microarchitectures.

Code (for X25519 shared secret computation)

https://sites.google.com/a/crypto.tw/blueprint/
The main messages of this talk:

- Vectorization should be considered on recent Intel microarchitectures.

Code (for X25519 shared secret computation):

https://sites.google.com/a/crypto.tw/blueprint/
Ending Remarks

The main messages of this talk:

- Vectorization should be considered on recent Intel microarchitectures.

Code (for X25519 shared secret computation)

- https://sites.google.com/a/crypto.tw/blueprint/