Faster binary-field multiplication and faster binary-field MACs

Tung Chou

Technische Universiteit Eindhoven, The Netherlands

August 15, 2014

Joint work with Daniel J. Bernstein

The scope of this talk

The scope of this talk

The Wegman–Carter construction (1981)

- "Universal" hash function + one-time pad: $h_r(m) \oplus s_n$
- Offers information-theoretic security
- What makes the difference is the hash part

The scope of this talk

The Wegman–Carter construction (1981)

- "Universal" hash function + one-time pad: $h_r(m) \oplus s_n$
- Offers information-theoretic security
- What makes the difference is the hash part

Binary field MACs:

- Less attractive for hardware designers

GMAC

GMAC

- Polynomial evaluation MAC: $m_1 r + m_2 r^2 + \cdots$
- Based on arithmetic in

$$\mathbb{F}_{2^{128}} = \mathbb{F}_2[x]/(x^{128} + x^7 + x^2 + x + 1)$$

GMAC

- Polynomial evaluation MAC: $m_1 r + m_2 r^2 + \cdots$
- Based on arithmetic in

$$\mathbb{F}_{2^{128}} = \mathbb{F}_2[x]/(x^{128} + x^7 + x^2 + x + 1)$$

GMAC

- Polynomial evaluation MAC: $m_1 r + m_2 r^2 + \cdots$
- Based on arithmetic in

$$\mathbb{F}_{2^{128}} = \mathbb{F}_2[x]/(x^{128} + x^7 + x^2 + x + 1)$$

Constant-time software speeds:

reference	platform	`PCLMUQDQ`	cycles per byte
Käsper–Schwabe 2009	Core 2	no	14.40
	Sandy Bridge	no	13.10
Krovetz–Rogaway 2011	Westmere	yes	2.00
Gueron 2013	Sandy Bridge	yes	1.79
	Haswell	yes	0.40

GMAC

- Polynomial evaluation MAC: $m_1 r + m_2 r^2 + \cdots$
- Based on arithmetic in

$$\mathbb{F}_{2^{128}} = \mathbb{F}_2[x]/(x^{128} + x^7 + x^2 + x + 1)$$

Constant-time software speeds:

reference	platform	PCLMUQDQ	cycles per byte
Käsper–Schwabe 2009	Core 2	no	14.40
	Sandy Bridge	no	13.10
Krovetz–Rogaway 2011	Westmere	yes	2.00
Gueron 2013	Sandy Bridge	yes	1.79
	Haswell	yes	0.40
Auth256	Core 2	no	2.29
	Sandy Bridge	no	1.59

GMAC

- Polynomial evaluation MAC: $m_1 r + m_2 r^2 + \cdots$
- Based on arithmetic in

$$\mathbb{F}_{2^{128}} = \mathbb{F}_2[x]/(x^{128} + x^7 + x^2 + x + 1)$$

Constant-time software speeds:

reference	platform	PCLMUQDQ	cycles per byte
Käsper–Schwabe 2009	Core 2	no	14.40
	Sandy Bridge	no	13.10
Krovetz–Rogaway 2011	Westmere	yes	2.00
Gueron 2013	Sandy Bridge	yes	1.79
	Haswell	yes	0.40
Auth256	Core 2	no	2.29
	Sandy Bridge	no	1.59

The point of Auth256 is to have **low bit operation count**.

Bit operations

GMAC

- Base field $\mathbb{F}_{2^{128}} = \mathbb{F}_2[x]/(f)$

Bit operations

GMAC

- Base field $\mathbb{F}_{2^{128}} = \mathbb{F}_2[x]/(f)$
- 256 bit operations/bit with schoolbook

Bit operations

GMAC

- Base field $\mathbb{F}_{2^{128}} = \mathbb{F}_2[x]/(f)$
- 256 bit operations/bit with schoolbook
- 90 bit operations/bit with Karatsuba/Toom (`http://binary.cr.yp.to/m.html`)

Bit operations

GMAC

- Base field $\mathbb{F}_{2^{128}} = \mathbb{F}_2[x]/(f)$
- 256 bit operations/bit with schoolbook
- 90 bit operations/bit with Karatsuba/Toom (`http://binary.cr.yp.to/m.html`)

GMAC scaled up

- Base field $\mathbb{F}_{2^{256}} = \mathbb{F}_2[x]/(f')$

Bit operations

GMAC

- Base field $\mathbb{F}_{2^{128}} = \mathbb{F}_2[x]/(f)$
- 256 bit operations/bit with schoolbook
- 90 bit operations/bit with Karatsuba/Toom (`http://binary.cr.yp.to/m.html`)

GMAC scaled up

- Base field $\mathbb{F}_{2^{256}} = \mathbb{F}_2[x]/(f')$
- 512 bit operations/bit with schoolbook

Bit operations

GMAC

- Base field $\mathbb{F}_{2^{128}} = \mathbb{F}_2[x]/(f)$
- 256 bit operations/bit with schoolbook
- 90 bit operations/bit with Karatsuba/Toom (`http://binary.cr.yp.to/m.html`)

GMAC scaled up

- Base field $\mathbb{F}_{2^{256}} = \mathbb{F}_2[x]/(f')$
- 512 bit operations/bit with schoolbook
- 133 bit operations/bit with Karatsuba/Toom (`http://binary.cr.yp.to/m.html`)

Bit operations

GMAC

- Base field $\mathbb{F}_{2^{128}} = \mathbb{F}_2[x]/(f)$
- 256 bit operations/bit with schoolbook
- 90 bit operations/bit with Karatsuba/Toom (`http://binary.cr.yp.to/m.html`)

GMAC scaled up

- Base field $\mathbb{F}_{2^{256}} = \mathbb{F}_2[x]/(f')$
- 512 bit operations/bit with schoolbook
- 133 bit operations/bit with Karatsuba/Toom (`http://binary.cr.yp.to/m.html`)

Auth256

- Base field $\mathbb{F}_{2^{256}}$

Bit operations

GMAC

- Base field $\mathbb{F}_{2^{128}} = \mathbb{F}_2[x]/(f)$
- 256 bit operations/bit with schoolbook
- 90 bit operations/bit with Karatsuba/Toom (`http://binary.cr.yp.to/m.html`)

GMAC scaled up

- Base field $\mathbb{F}_{2^{256}} = \mathbb{F}_2[x]/(f')$
- 512 bit operations/bit with schoolbook
- 133 bit operations/bit with Karatsuba/Toom (`http://binary.cr.yp.to/m.html`)

Auth256

- Base field $\mathbb{F}_{2^{256}}$
- 34 bit operations/bit

The design

The design

Auth256

- A *pseudo-dot-product* MAC:

$$(m_1 + r_1)(m_2 + r_2) + (m_3 + r_3)(m_4 + r_4) + \cdots$$

- Base field $\mathbb{F}_{2^{256}} = \mathbb{F}_{2^8}[x]/(f)$. Tower field construction for $\mathbb{F}_{2^8}$.

The design

Auth256

- A *pseudo-dot-product* MAC:

$$(m_1 + r_1)(m_2 + r_2) + (m_3 + r_3)(m_4 + r_4) + \cdots$$

- Base field $\mathbb{F}_{2^{256}} = \mathbb{F}_{2^8}[x]/(f)$. Tower field construction for $\mathbb{F}_{2^8}$.

Three sources of speedup $(133 \to 34)$ over the scaled-up GMAC:

The design

Auth256

- A *pseudo-dot-product* MAC:

$$(m_1 + r_1)(m_2 + r_2) + (m_3 + r_3)(m_4 + r_4) + \cdots$$

- Base field $\mathbb{F}_{2^{256}} = \mathbb{F}_{2^8}[x]/(f)$. Tower field construction for $\mathbb{F}_{2^8}$.

Three sources of speedup $(133 \to 34)$ over the scaled-up GMAC:

- Faster multiplications in $\mathbb{F}_{2^{256}}$, which is achieved by fast polynomial multiplication in $\mathbb{F}_{2^8}[x]$: $133 \to 87$

The design

Auth256

- A *pseudo-dot-product* MAC:

$$(m_1 + r_1)(m_2 + r_2) + (m_3 + r_3)(m_4 + r_4) + \cdots$$

- Base field $\mathbb{F}_{2^{256}} = \mathbb{F}_{2^8}[x]/(f)$. Tower field construction for $\mathbb{F}_{2^8}$.

Three sources of speedup $(133 \to 34)$ over the scaled-up GMAC:

- Faster multiplications in $\mathbb{F}_{2^{256}}$, which is achieved by fast polynomial multiplication in $\mathbb{F}_{2^8}[x]$: $133 \to 87$
- 0.5 multiplications per message block: $87 \to 44$

The design

Auth256

- A *pseudo-dot-product* MAC:

$$(m_1 + r_1)(m_2 + r_2) + (m_3 + r_3)(m_4 + r_4) + \cdots$$

- Base field $\mathbb{F}_{2^{256}} = \mathbb{F}_{2^8}[x]/(f)$. Tower field construction for $\mathbb{F}_{2^8}$.

Three sources of speedup $(133 \to 34)$ over the scaled-up GMAC:

- Faster multiplications in $\mathbb{F}_{2^{256}}$, which is achieved by fast polynomial multiplication in $\mathbb{F}_{2^8}[x]$: $133 \to 87$
- 0.5 multiplications per message block: $87 \to 44$
- Merging multiplications: $44 \to 34$

The design

Auth256

- A *pseudo-dot-product* MAC:

$$(m_1 + r_1)(m_2 + r_2) + (m_3 + r_3)(m_4 + r_4) + \cdots$$

- Base field $\mathbb{F}_{2^{256}} = \mathbb{F}_{2^8}[x]/(f)$. Tower field construction for $\mathbb{F}_{2^8}$.

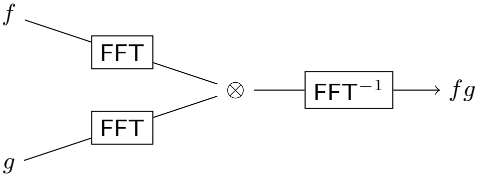
Three sources of speedup $(133 \to 34)$ over the scaled-up GMAC:

- Faster multiplications in $\mathbb{F}_{2^{256}}$, which is achieved by fast polynomial multiplication in $\mathbb{F}_{2^8}[x]$: $133 \to 87$
- 0.5 multiplications per message block: $87 \to 44$
- Merging multiplications: $44 \to 34$

We use **FFT-based** polynomial multiplication.

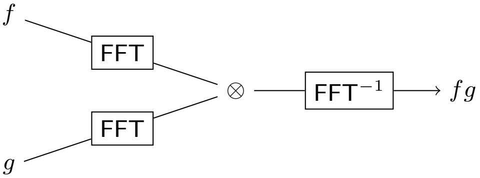
FFT for polynomial multiplication

- Given polynomials $f, g \in R[x]$, fg is computed by



FFT for polynomial multiplication

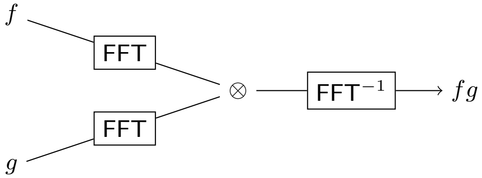
- Given polynomials $f, g \in R[x]$, fg is computed by



- size-n FFTs: compute $f(\alpha_i)$, $g(\alpha_i)$ for $i = 1, \ldots, n$.

FFT for polynomial multiplication

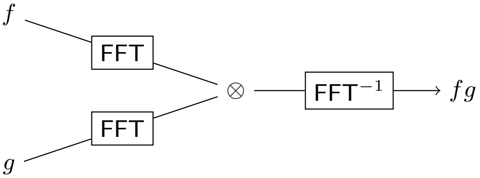
- Given polynomials $f, g \in R[x]$, fg is computed by



- size-n FFTs: compute $f(\alpha_i)$, $g(\alpha_i)$ for $i = 1, \ldots, n$.
- $\otimes$ is pointwise multiplication: compute $f(\alpha_i)g(\alpha_i)$ for $i = 1, \ldots, n$.

FFT for polynomial multiplication

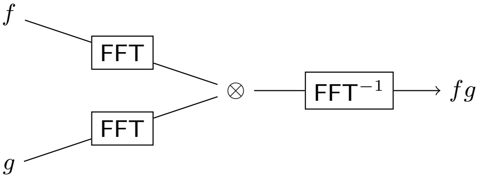
- Given polynomials $f, g \in R[x]$, fg is computed by



- size-n FFTs: compute $f(\alpha_i)$, $g(\alpha_i)$ for $i = 1, \ldots, n$.
- $\otimes$ is pointwise multiplication: compute $f(\alpha_i)g(\alpha_i)$ for $i = 1, \ldots, n$.
- FFT^{-1}: recover fg from images

FFT for polynomial multiplication

- Given polynomials $f, g \in R[x]$, fg is computed by



- size-n FFTs: compute $f(\alpha_i)$, $g(\alpha_i)$ for $i = 1, \dots, n$.
- $\otimes$ is pointwise multiplication: compute $f(\alpha_i)g(\alpha_i)$ for $i = 1, \dots, n$.
- FFT^{-1}: recover fg from images

The design of Auth256 is tailored for **the Gao–Mateer additive FFT** (2010).

Multiplicative FFT

- The key equation (free):

$$f(x) = f^{(0)}(x^2) + x f^{(1)}(x^2)$$

Multiplicative FFT

- The key equation (free):

$$f(x) = f^{(0)}(x^2) + xf^{(1)}(x^2)$$

- Big overlap between evaluation of $f(\alpha)$ and $f(-\alpha)$:

$$\begin{aligned} f(\alpha) =& f^{(0)}(\alpha^2) + \alpha f^{(1)}(\alpha^2) \\ f(-\alpha) =& f^{(0)}(\alpha^2) - \alpha f^{(1)}(\alpha^2) \end{aligned}$$

Multiplicative FFT

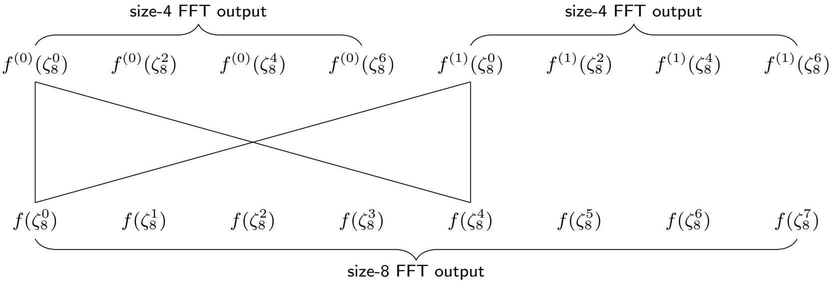
- The key equation (free):

$$f(x) = f^{(0)}(x^2) + xf^{(1)}(x^2)$$

- Big overlap between evaluation of $f(\alpha)$ and $f(-\alpha)$:

$$\begin{aligned} f(\alpha) &= f^{(0)}(\alpha^2) + \alpha f^{(1)}(\alpha^2) \\ f(-\alpha) &= f^{(0)}(\alpha^2) - \alpha f^{(1)}(\alpha^2) \end{aligned}$$

- Size-2^k FFT evaluates f at all 2^k-th roots of unity



The Gao–Mateer additive FFT

- The key equation (takes some additions):

$$f(x) = f^{(0)}(x^2 + x) + xf^{(1)}(x^2 + x)$$

The Gao–Mateer additive FFT

- The key equation (takes some additions):

$$f(x) = f^{(0)}(x^2 + x) + x f^{(1)}(x^2 + x)$$

- Big overlap between evaluation of $f(\alpha)$ and $f(\alpha + 1)$:

$$f(\alpha) = f^{(0)}(\alpha^2 + \alpha) + \alpha f^{(1)}(\alpha^2 + \alpha)$$
$$f(\alpha + 1) = f^{(0)}(\alpha^2 + \alpha) + (\alpha + 1) f^{(1)}(\alpha^2 + \alpha)$$

The Gao–Mateer additive FFT

- The key equation (takes some additions):

$$f(x) = f^{(0)}(x^2 + x) + x f^{(1)}(x^2 + x)$$

- Big overlap between evaluation of $f(\alpha)$ and $f(\alpha + 1)$:

$$f(\alpha) = f^{(0)}(\alpha^2 + \alpha) + \alpha f^{(1)}(\alpha^2 + \alpha)$$
$$f(\alpha + 1) = f^{(0)}(\alpha^2 + \alpha) + (\alpha + 1) f^{(1)}(\alpha^2 + \alpha)$$

- Size-2^k FFT evaluates f at a k-dimensional $\mathbb{F}_2$-linear subspace in the field

The Gao–Mateer additive FFT (Cont.)

We use a "Cantor basis" for FFTs:

$$\beta_5 \xrightarrow{x^2+x} \beta_4 \xrightarrow{x^2+x} \underbrace{\beta_3 \xrightarrow{x^2+x} \beta_2 \xrightarrow{x^2+x} \underbrace{\beta_1 \xrightarrow{x^2+x} \beta_0 = 1}_{\mathbb{F}_{2^2}}}_{\mathbb{F}_{2^4}}$$

A size-2^k FFT evaluates over the span of $\beta_{k-1}, \ldots, \beta_0$, which means small-size FFTs involve only **multiplications by constants in subfields**.

$\mathbb{F}_{2^8}$ arithmetic: representation

Tower field construction of $\mathbb{F}_{2^8}$:

$$\begin{array}{l}
\mathbb{F}_{2^8} \\
\;\big|\; x_8^2 + x_8 + \alpha_2\alpha_4 \\
\mathbb{F}_{2^4} \\
\;\big|\; x_4^2 + x_4 + \alpha_2 \\
\mathbb{F}_{2^2} \\
\;\big|\; x_2^2 + x_2 + 1 \\
\mathbb{F}_2
\end{array}$$

$\mathbb{F}_{2^8}$ arithmetic: representation

Tower field construction of $\mathbb{F}_{2^8}$:

$$
\begin{array}{l}
\mathbb{F}_{2^8} \\
\quad \Big| \; x_8^2 + x_8 + \alpha_2\alpha_4 \\
\mathbb{F}_{2^4} \\
\quad \Big| \; x_4^2 + x_4 + \alpha_2 \\
\mathbb{F}_{2^2} \\
\quad \Big| \; x_2^2 + x_2 + 1 \\
\mathbb{F}_2
\end{array}
$$

Representation of field elements:

- $b_0 + b_1\alpha_2 \in \mathbb{F}_{2^2}$
- $(b_0 + b_1\alpha_2) + (b_2 + b_3\alpha_2)\alpha_4 \in \mathbb{F}_{2^4}$
- $(b_4 + b_5\alpha_2) + (b_6 + b_7\alpha_2)\alpha_4 + ((b_4 + b_5\alpha_2) + (b_6 + b_7\alpha_2)\alpha_4)\alpha_8 \in \mathbb{F}_{2^8}$

$\mathbb{F}_{2^8}$ arithmetic: constant multiplication

Fix a constant α. A *constant multiplication* is the computation of αb given b as input.

$\mathbb{F}_{2^8}$ arithmetic: constant multiplication

Fix a constant α. A *constant multiplication* is the computation of αb given b as input.

The tower-field construction makes constant multiplications become much faster when the constants are in **subfields**.

$\mathbb{F}_{2^8}$ arithmetic: constant multiplication

Fix a constant α. A *constant multiplication* is the computation of αb given b as input.

The tower-field construction makes constant multiplications become much faster when the constants are in **subfields**.

- 14.83 bit operations on average for $\alpha \in \mathbb{F}_{2^8} \backslash \mathbb{F}_2$
- 7.43 bit operations on average for $\alpha \in \mathbb{F}_{2^4} \backslash \mathbb{F}_2$
- 4 bit operations on average for $\alpha \in \mathbb{F}_{2^2} \backslash \mathbb{F}_2$

$\mathbb{F}_{2^8}$ arithmetic: constant multiplication

Fix a constant α. A *constant multiplication* is the computation of αb given b as input.

The tower-field construction makes constant multiplications become much faster when the constants are in **subfields**.

- 14.83 bit operations on average for $\alpha \in \mathbb{F}_{2^8} \backslash \mathbb{F}_2$
- 7.43 bit operations on average for $\alpha \in \mathbb{F}_{2^4} \backslash \mathbb{F}_2$
- 4 bit operations on average for $\alpha \in \mathbb{F}_{2^2} \backslash \mathbb{F}_2$
- 9.02 bit operations on average for FFT

$\mathbb{F}_{2^8}$ arithmetic: constant multiplication

Fix a constant α. A *constant multiplication* is the computation of αb given b as input.

The tower-field construction makes constant multiplications become much faster when the constants are in **subfields**.

- 14.83 bit operations on average for $\alpha \in \mathbb{F}_{2^8} \backslash \mathbb{F}_2$
- 7.43 bit operations on average for $\alpha \in \mathbb{F}_{2^4} \backslash \mathbb{F}_2$
- 4 bit operations on average for $\alpha \in \mathbb{F}_{2^2} \backslash \mathbb{F}_2$
- 9.02 bit operations on average for FFT

How do we derive the operations for constant multiplication by some α?

$\mathbb{F}_{2^8}$ arithmetic: constant multiplication

Fix a constant α. A *constant multiplication* is the computation of αb given b as input.

The tower-field construction makes constant multiplications become much faster when the constants are in **subfields**.

- 14.83 bit operations on average for $\alpha \in \mathbb{F}_{2^8} \backslash \mathbb{F}_2$
- 7.43 bit operations on average for $\alpha \in \mathbb{F}_{2^4} \backslash \mathbb{F}_2$
- 4 bit operations on average for $\alpha \in \mathbb{F}_{2^2} \backslash \mathbb{F}_2$
- 9.02 bit operations on average for FFT

How do we derive the operations for constant multiplication by some α?
Using a linear map circuit generator

An invertible linear map circuit generator

$$\begin{array}{r|cccc}
 & (x_0, & x_1, & x_2, & x_3) \\
\phi_{\alpha_2+\alpha_4} \Big\downarrow & & & & \\
 & (x_0 + x_2 + x_3, & x_0 + x_1 + x_2 + x_3, & x_1 + x_2, & x_1 + x_3)
\end{array}$$

An invertible linear map circuit generator

$$
\begin{array}{cccc}
(x_0, & x_1, & x_2, & x_3) \\
\\
(x_0 + x_2 + x_3, & x_1, & x_1 + x_2, & x_1 + x_3) \\
(x_0 + x_2 + x_3, & x_0 + x_1 + x_2 + x_3, & x_1 + x_2, & x_1 + x_3)
\end{array}
$$

$\phi_{\alpha_2+\alpha_4}$

An invertible linear map circuit generator

$$\phi_{\alpha_2+\alpha_4}\Bigg\downarrow\quad
\begin{array}{cccc}
(x_0, & x_1, & x_2, & x_3) \\
\\
(x_0+x_2+x_3, & x_1, & x_1+x_2, & x_3) \\
(x_0+x_2+x_3, & x_1, & x_1+x_2, & x_1+x_3) \\
(x_0+x_2+x_3, & x_0+x_1+x_2+x_3, & x_1+x_2, & x_1+x_3)
\end{array}$$

An invertible linear map circuit generator

$$
\begin{array}{c|ccccc}
 & (x_0, & x_1, & x_2, & x_3) \\
 & & & & \\
 & (x_0 + x_2, & x_1, & x_1 + x_2, & x_3) \\
\phi_{\alpha_2+\alpha_4} & (x_0 + x_2 + x_3, & x_1, & x_1 + x_2, & x_3) \\
 & (x_0 + x_2 + x_3, & x_1, & x_1 + x_2, & x_1 + x_3) \\
\downarrow & (x_0 + x_2 + x_3, & x_0 + x_1 + x_2 + x_3, & x_1 + x_2, & x_1 + x_3)
\end{array}
$$

An invertible linear map circuit generator

$$\phi_{\alpha_2+\alpha_4}\downarrow\quad\begin{array}{cccc}
(x_0, & x_1, & x_2, & x_3) \\
(x_0+x_2, & x_1, & x_2, & x_3) \\
(x_0+x_2, & x_1, & x_1+x_2, & x_3) \\
(x_0+x_2+x_3, & x_1, & x_1+x_2, & x_3) \\
(x_0+x_2+x_3, & x_1, & x_1+x_2, & x_1+x_3) \\
(x_0+x_2+x_3, & x_0+x_1+x_2+x_3, & x_1+x_2, & x_1+x_3)
\end{array}$$

An invertible linear map circuit generator

$$
\begin{array}{c|cccc}
 & (x_0, & x_1, & x_2, & x_3) \\
 & (x_0 + x_2, & x_1, & x_2, & x_3) \\
 & (x_0 + x_2, & x_1, & x_1 + x_2, & x_3) \\
\phi_{\alpha_2+\alpha_4} \big\downarrow & (x_0 + x_2 + x_3, & x_1, & x_1 + x_2, & x_3) \\
 & (x_0 + x_2 + x_3, & x_1, & x_1 + x_2, & x_1 + x_3) \\
 & (x_0 + x_2 + x_3, & x_0 + x_1 + x_2 + x_3, & x_1 + x_2, & x_1 + x_3)
\end{array}
$$

An invertible linear map circuit generator

$$
\phi_{\alpha_2+\alpha_4}\Bigg\downarrow\quad
\begin{array}{cccc}
(x_0, & x_1, & x_2, & x_3) \\
(x_0 + x_2, & x_1, & x_2, & x_3) \\
(x_0 + x_2, & x_1, & x_1 + x_2, & x_3) \\
(x_0 + x_2 + x_3, & x_1, & x_1 + x_2, & x_3) \\
(x_0 + x_2 + x_3, & x_1, & x_1 + x_2, & x_1 + x_3) \\
(x_0 + x_2 + x_3, & x_0 + x_1 + x_2 + x_3, & x_1 + x_2, & x_1 + x_3)
\end{array}
$$

An invertible linear map circuit generator

$$
\phi_{\alpha_2+\alpha_4}\ \Bigg\downarrow \quad
\begin{array}{cccc}
(x_0, & x_1, & x_2, & x_3) \\
(x_0 + x_2, & x_1, & x_2, & x_3) \\
(x_0 + x_2, & x_1, & x_1 + x_2, & x_3) \\
(x_0 + x_2 + x_3, & x_1, & x_1 + x_2, & x_3) \\
(x_0 + x_2 + x_3, & x_1, & x_1 + x_2, & x_1 + x_3) \\
(x_0 + x_2 + x_3, & x_0 + x_1 + x_2 + x_3, & x_1 + x_2, & x_1 + x_3)
\end{array}
$$

- Somewhat like Paar's additive greedy common-subexpression elimination algorithm (1997)
- A **2-operand** algorithm like Bernstein's "xor-largest" algorithm (2009)

Merging radix conversions

Radix conversions:

$$\begin{aligned}f =& f^{(0)}(x^2+x) + xf^{(1)}(x^2+x)\\ =& f^{(00)}(x^4+x) + (x^2+x)f^{(01)}(x^4+x)+\\ & xf^{(10)}(x^4+x) + x(x^2+x)f^{(11)}(x^4+x)\end{aligned}$$

Merging radix conversions

Radix conversions:

$$\begin{aligned} f =& f^{(0)}(x^2+x) + xf^{(1)}(x^2+x) \\ =& f^{(00)}(x^4+x) + (x^2+x)f^{(01)}(x^4+x) + \\ & xf^{(10)}(x^4+x) + x(x^2+x)f^{(11)}(x^4+x) \end{aligned}$$

Doing the size-8 first and then size-4 ones gives 12 additions.

Merging radix conversions

Radix conversions:

$$\begin{aligned} f =& f^{(0)}(x^2+x) + x f^{(1)}(x^2+x) \\ =& f^{(00)}(x^4+x) + (x^2+x) f^{(01)}(x^4+x) + \\ & x f^{(10)}(x^4+x) + x(x^2+x) f^{(11)}(x^4+x) \end{aligned}$$

Doing the size-8 first and then size-4 ones gives 12 additions.

For size-16 FFT the input is $f = \sum_{i=0}^{7} f_i x^i$:

$$\begin{aligned} f^{(00)} &= f_0 + (f_4 + f_7)x, \\ f^{(01)} &= (f_2 + f_3 + f_5 + f_6) + (f_6 + f_7)x, \\ f^{(10)} &= (f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7) + (f_5 + f_6 + f_7)x, \\ f^{(11)} &= (f_3 + f_6) + f_7 x. \end{aligned}$$

Merging radix conversions

Radix conversions:

$$\begin{aligned} f =& f^{(0)}(x^2+x) + x f^{(1)}(x^2+x) \\ =& f^{(00)}(x^4+x) + (x^2+x) f^{(01)}(x^4+x) + \\ & x f^{(10)}(x^4+x) + x(x^2+x) f^{(11)}(x^4+x) \end{aligned}$$

Doing the size-8 first and then size-4 ones gives 12 additions.

For size-16 FFT the input is $f = \sum_{i=0}^{7} f_i x^i$:

$$\begin{aligned} f^{(00)} &= f_0 + (f_4 + f_7)x, \\ f^{(01)} &= (f_2 + f_3 + f_5 + f_6) + (f_6 + f_7)x, \\ f^{(10)} &= (f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7) + (f_5 + f_6 + f_7)x, \\ f^{(11)} &= (f_3 + f_6) + f_7 x. \end{aligned}$$

How about considering this as a generic linear map?

Merging radix conversions

Radix conversions:

$$
\begin{aligned}
f =& f^{(0)}(x^2+x) + xf^{(1)}(x^2+x) \\
=& f^{(00)}(x^4+x) + (x^2+x)f^{(01)}(x^4+x) + \\
& xf^{(10)}(x^4+x) + x(x^2+x)f^{(11)}(x^4+x)
\end{aligned}
$$

Doing the size-8 first and then size-4 ones gives 12 additions.

For size-16 FFT the input is $f = \sum_{i=0}^{7} f_i x^i$:

$$
\begin{aligned}
f^{(00)} &= f_0 + (f_4 + f_7)x, \\
f^{(01)} &= (f_2 + f_3 + f_5 + f_6) + (f_6 + f_7)x, \\
f^{(10)} &= (f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7) + (f_5 + f_6 + f_7)x, \\
f^{(11)} &= (f_3 + f_6) + f_7 x.
\end{aligned}
$$

How about considering this as a generic linear map?

- 8 additions: Gao–Mateer's algorithm is suboptimal in this case

Merging radix conversions (cont.)

How about larger-size radix conversions?

Merging radix conversions (cont.)

How about larger-size radix conversions?

- Use the linear map circuit generator again

Merging radix conversions (cont.)

How about larger-size radix conversions?

- Use the linear map circuit generator again

The gain is bigger for larger sizes:

- 8 instead of 12 additions for size-8.
- 31 instead of 48 additions for size-16.
- 82 instead of 160 additions for size-32.

Merging radix conversions (cont.)

How about larger-size radix conversions?

- Use the linear map circuit generator again

The gain is bigger for larger sizes:

- 8 instead of 12 additions for size-8.
- 31 instead of 48 additions for size-16.
- 82 instead of 160 additions for size-32.

Numbers can be further improved by an asymptotically faster radix conversion algorithm.

Faster binary-field multiplication

- bit operations for multiplication in $\mathbb{F}_{2^b}$

b	our result	competition
64	3726	≥ 3745
128	9126	≥ 11613
256	22292	≥ 34334

- competition: `http://binary.cr.yp.to/m.html`
- D'Angella et. al. (2013) have slight improvements
- Can do better for $\mathbb{F}_{2^{256}}$ with 8-way split: $f(x) = \sum_{i=0}^{7} x^i f^{(i)}(x)$