

McBits:

fast constant-time
code-based cryptography

D. J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven

Joint work with:

Tung Chou

Technische Universiteit Eindhoven
(original speaker,
still waiting for U.S. visa)

Peter Schwabe

Radboud University Nijmegen

Objectives

Set new speed records
for public-key cryptography.

McBits:

fast constant-time
code-based cryptography

D. J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven

Joint work with:

Tung Chou

Technische Universiteit Eindhoven
(original speaker,
still waiting for U.S. visa)

Peter Schwabe

Radboud University Nijmegen

Objectives

Set new speed records
for public-key cryptography.

... at a high security level.

McBits:

fast constant-time
code-based cryptography

D. J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven

Joint work with:

Tung Chou

Technische Universiteit Eindhoven
(original speaker,
still waiting for U.S. visa)

Peter Schwabe

Radboud University Nijmegen

Objectives

Set new speed records
for public-key cryptography.

... at a high security level.

... including protection
against quantum computers.

McBits:

fast constant-time
code-based cryptography

D. J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven

Joint work with:

Tung Chou

Technische Universiteit Eindhoven
(original speaker,
still waiting for U.S. visa)

Peter Schwabe

Radboud University Nijmegen

Objectives

Set new speed records
for public-key cryptography.

... at a high security level.

... including protection
against quantum computers.

... including full protection
against cache-timing attacks,
branch-prediction attacks, etc.

McBits:

fast constant-time
code-based cryptography

D. J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven

Joint work with:

Tung Chou

Technische Universiteit Eindhoven
(original speaker,
still waiting for U.S. visa)

Peter Schwabe

Radboud University Nijmegen

Objectives

Set new speed records
for public-key cryptography.

... at a high security level.

... including protection
against quantum computers.

... including full protection
against cache-timing attacks,
branch-prediction attacks, etc.

... using code-based crypto
with a solid track record.

McBits:

fast constant-time
code-based cryptography

D. J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven

Joint work with:

Tung Chou

Technische Universiteit Eindhoven
(original speaker,
still waiting for U.S. visa)

Peter Schwabe

Radboud University Nijmegen

Objectives

Set new speed records
for public-key cryptography.

... at a high security level.

... including protection
against quantum computers.

... including full protection
against cache-timing attacks,
branch-prediction attacks, etc.

... using code-based crypto
with a solid track record.

... all of the above *at once*.

stant-time
sed cryptography
ernstein
ty of Illinois at Chicago &
che Universiteit Eindhoven
ork with:
nou
che Universiteit Eindhoven
speaker,
ting for U.S. visa)
chwabe
d University Nijmegen

Objectives

Set new speed records
for public-key cryptography.
... at a high security level.
... including protection
against quantum computers.
... including full protection
against cache-timing attacks,
branch-prediction attacks, etc.
... using code-based crypto
with a solid track record.
... all of the above *at once*.

The track

1978 Mo
public-ke
Has helo
optimiza
1962 Pra
1988 Lec
1989 Kro
1989 Du
1990 Co
1990 var
1991 Co
1993 Ch
1993 Ch

graphy

is at Chicago &
iteit Eindhoven

iteit Eindhoven

S. visa)

ty Nijmegen

Objectives

Set new speed records
for public-key cryptography.

... at a high security level.

... including protection
against quantum computers.

... including full protection
against cache-timing attacks,
branch-prediction attacks, etc.

... using code-based crypto
with a solid track record.

... all of the above *at once*.

The track record

1978 McEliece pro
public-key code-ba

Has held up well a
optimization of at

1962 Prange. 198

1988 Lee–Brickell.

1989 Krouk. 1989

1989 Dumer.

1990 Coffey–Good

1990 van Tilburg.

1991 Coffey–Good

1993 Chabanne–C

1993 Chabaud.

Objectives

Set new speed records
for public-key cryptography.

... at a high security level.

... including protection
against quantum computers.

... including full protection
against cache-timing attacks,
branch-prediction attacks, etc.

... using code-based crypto
with a solid track record.

... all of the above *at once*.

The track record

1978 McEliece proposed
public-key code-based crypto

Has held up well after exten
optimization of attack algori

1962 Prange. 1981 Omura.

1988 Lee–Brickell. 1988 Lec

1989 Krouk. 1989 Stern.

1989 Dumer.

1990 Coffey–Goodman.

1990 van Tilburg. 1991 Dur

1991 Coffey–Goodman–Farr

1993 Chabanne–Courteau.

1993 Chabaud.

Objectives

Set new speed records
for public-key cryptography.

... at a high security level.

... including protection
against quantum computers.

... including full protection
against cache-timing attacks,
branch-prediction attacks, etc.

... using code-based crypto
with a solid track record.

... all of the above *at once*.

The track record

1978 McEliece proposed
public-key code-based crypto.

Has held up well after extensive
optimization of attack algorithms:

1962 Prange. 1981 Omura.

1988 Lee–Brickell. 1988 Leon.

1989 Krouk. 1989 Stern.

1989 Dumer.

1990 Coffey–Goodman.

1990 van Tilburg. 1991 Dumer.

1991 Coffey–Goodman–Farrell.

1993 Chabanne–Courteau.

1993 Chabaud.

es

speed records
c-key cryptography.

high security level.

uding protection
quantum computers.

uding full protection
cache-timing attacks,
prediction attacks, etc.

g code-based crypto
olid track record.

f the above *at once*.

The track record

1978 McEliece proposed
public-key code-based crypto.

Has held up well after extensive
optimization of attack algorithms:

1962 Prange. 1981 Omura.

1988 Lee–Brickell. 1988 Leon.

1989 Krouk. 1989 Stern.

1989 Dumer.

1990 Coffey–Goodman.

1990 van Tilburg. 1991 Dumer.

1991 Coffey–Goodman–Farrell.

1993 Chabanne–Courteau.

1993 Chabaud.

1994 van

1994 Ca

1998 Ca

1998 Ca

2008 Be

2009 Be

Peters–v

2009 Be

2009 Fir

2010 Be

2011 Ma

2011 Be

2012 Be

2013 Be

Meurer (

The track record

1978 McEliece proposed

public-key code-based crypto.

Has held up well after extensive

optimization of attack algorithms:

1962 Prange. 1981 Omura.

1988 Lee–Brickell. 1988 Leon.

1989 Krouk. 1989 Stern.

1989 Dumer.

1990 Coffey–Goodman.

1990 van Tilburg. 1991 Dumer.

1991 Coffey–Goodman–Farrell.

1993 Chabanne–Courteau.

1993 Chabaud.

1994 van Tilburg.

1994 Canteaut–Ch

1998 Canteaut–Ch

1998 Canteaut–Se

2008 Bernstein–La

2009 Bernstein–La

Peters–van Tilborg

2009 Bernstein (p

2009 Finiasz–Send

2010 Bernstein–La

2011 May–Meurer

2011 Becker–Coro

2012 Becker–Joux

2013 Bernstein–Je

Meurer (post-quar

The track record

1978 McEliece proposed
public-key code-based crypto.

Has held up well after extensive
optimization of attack algorithms:

1962 Prange. 1981 Omura.

1988 Lee–Brickell. 1988 Leon.

1989 Krouk. 1989 Stern.

1989 Dumer.

1990 Coffey–Goodman.

1990 van Tilburg. 1991 Dumer.

1991 Coffey–Goodman–Farrell.

1993 Chabanne–Courteau.

1993 Chabaud.

1994 van Tilburg.

1994 Canteaut–Chabanne.

1998 Canteaut–Chabaud.

1998 Canteaut–Sendrier.

2008 Bernstein–Lange–Peter

2009 Bernstein–Lange–
Peters–van Tilborg.

2009 Bernstein (post-quantum)

2009 Finiasz–Sendrier.

2010 Bernstein–Lange–Peter

2011 May–Meurer–Thomae.

2011 Becker–Coron–Joux.

2012 Becker–Joux–May–Me

2013 Bernstein–Jeffery–Lang

Meurer (post-quantum).

The track record

1978 McEliece proposed
public-key code-based crypto.

Has held up well after extensive
optimization of attack algorithms:

1962 Prange. 1981 Omura.

1988 Lee–Brickell. 1988 Leon.

1989 Krouk. 1989 Stern.

1989 Dumer.

1990 Coffey–Goodman.

1990 van Tilburg. 1991 Dumer.

1991 Coffey–Goodman–Farrell.

1993 Chabanne–Courteau.

1993 Chabaud.

1994 van Tilburg.

1994 Canteaut–Chabanne.

1998 Canteaut–Chabaud.

1998 Canteaut–Sendrier.

2008 Bernstein–Lange–Peters.

2009 Bernstein–Lange–
Peters–van Tilborg.

2009 Bernstein (post-quantum).

2009 Finiasz–Sendrier.

2010 Bernstein–Lange–Peters.

2011 May–Meurer–Thomae.

2011 Becker–Coron–Joux.

2012 Becker–Joux–May–Meurer.

2013 Bernstein–Jeffery–Lange–
Meurer (post-quantum).

Attack record

Eliece proposed

key code-based crypto.

held up well after extensive

evaluation of attack algorithms:

1978 Canteaut–Lange. 1981 Omura.

1982 Canteaut–Brickell. 1988 Leon.

1989 Joux. 1989 Stern.

1990 Shamir.

1991 Jeffrey–Goodman.

1992 van Tilburg. 1991 Dumer.

1993 Jeffrey–Goodman–Farrell.

1994 Canteaut–Courteau.

1995 Canteaut–Chabaud.

1994 van Tilburg.

1994 Canteaut–Chabanne.

1998 Canteaut–Chabaud.

1998 Canteaut–Sendrier.

2008 Bernstein–Lange–Peters.

2009 Bernstein–Lange–

Peters–van Tilborg.

2009 Bernstein (post-quantum).

2009 Finiasz–Sendrier.

2010 Bernstein–Lange–Peters.

2011 May–Meurer–Thomae.

2011 Becker–Coron–Joux.

2012 Becker–Joux–May–Meurer.

2013 Bernstein–Jeffery–Lange–

Meurer (post-quantum).

Example

Some cy

(Intel Co

from [ber](#)

mcelieo

(2008 B

gls254

(binary e

kumfp12

(hyperel

curve25

(conserv

mcelieo

ronald1

Proposed
based crypto.
after extensive
attack algorithms:
1 Omura.
1988 Leon.
Stern.
Liman.
1991 Dumer.
Liman–Farrell.
Courteau.

1994 van Tilburg.
1994 Canteaut–Chabanne.
1998 Canteaut–Chabaud.
1998 Canteaut–Sendrier.
2008 Bernstein–Lange–Peters.
2009 Bernstein–Lange–
Peters–van Tilborg.
2009 Bernstein (post-quantum).
2009 Finiasz–Sendrier.
2010 Bernstein–Lange–Peters.
2011 May–Meurer–Thomae.
2011 Becker–Coron–Joux.
2012 Becker–Joux–May–Meurer.
2013 Bernstein–Jeffery–Lange–
Meurer (post-quantum).

Examples of the c
Some cycle counts
(Intel Core i5-3210
from bench.cr.yp.to
mceliece encrypt
(2008 Biswas–Sen
g1s254 DH
(binary elliptic cur
kumfp127g DH
(hyperelliptic; Euro
curve25519 DH
(conservative ellipt
mceliece **decryp**
ronald1024 decry

1994 van Tilburg.
1994 Canteaut–Chabanne.
1998 Canteaut–Chabaud.
1998 Canteaut–Sendrier.
2008 Bernstein–Lange–Peters.
2009 Bernstein–Lange–
Peters–van Tilborg.
2009 Bernstein (post-quantum).
2009 Finiasz–Sendrier.
2010 Bernstein–Lange–Peters.
2011 May–Meurer–Thomae.
2011 Becker–Coron–Joux.
2012 Becker–Joux–May–Meurer.
2013 Bernstein–Jeffery–Lange–
Meurer (post-quantum).

Examples of the competition

Some cycle counts on h9ivy
(Intel Core i5-3210M, Ivy Br
from bench.cr.yp.to:

mceliece encrypt
(2008 Biswas–Sendrier, $\approx 2^8$
g1s254 DH
(binary elliptic curve; CHES
kumfp127g DH 1
(hyperelliptic; Eurocrypt 201
curve25519 DH 1
(conservative elliptic curve)
mceliece **decrypt** 12
ronald1024 decrypt 13

1994 van Tilburg.
1994 Canteaut–Chabanne.
1998 Canteaut–Chabaud.
1998 Canteaut–Sendrier.
2008 Bernstein–Lange–Peters.
2009 Bernstein–Lange–
Peters–van Tilborg.
2009 Bernstein (post-quantum).
2009 Finiasz–Sendrier.
2010 Bernstein–Lange–Peters.
2011 May–Meurer–Thomae.
2011 Becker–Coron–Joux.
2012 Becker–Joux–May–Meurer.
2013 Bernstein–Jeffery–Lange–
Meurer (post-quantum).

Examples of the competition

Some cycle counts on `h9ivy`
(Intel Core i5-3210M, Ivy Bridge)
from bench.cr.yp.to:

<code>mceliece encrypt</code>	61440
(2008 Biswas–Sendrier, $\approx 2^{80}$)	
<code>g1s254 DH</code>	77468
(binary elliptic curve; CHES 2013)	
<code>kumfp127g DH</code>	116944
(hyperelliptic; Eurocrypt 2013)	
<code>curve25519 DH</code>	182632
(conservative elliptic curve)	
<code>mceliece decrypt</code>	1219344
<code>ronald1024 decrypt</code>	1340040

n Tilburg.
nteaut–Chabanne.
nteaut–Chabaud.
nteaut–Sendrier.
rnstein–Lange–Peters.
rnstein–Lange–
van Tilborg.
rnstein (post-quantum).
niasz–Sendrier.
rnstein–Lange–Peters.
ay–Meurer–Thomae.
cker–Coron–Joux.
cker–Joux–May–Meurer.
rnstein–Jeffery–Lange–
(post-quantum).

Examples of the competition

Some cycle counts on h9ivy
(Intel Core i5-3210M, Ivy Bridge)
from bench.cr.yp.to:

mceliece encrypt	61440
(2008 Biswas–Sendrier, $\approx 2^{80}$)	
g1s254 DH	77468
(binary elliptic curve; CHES 2013)	
kumfp127g DH	116944
(hyperelliptic; Eurocrypt 2013)	
curve25519 DH	182632
(conservative elliptic curve)	
mceliece decrypt	1219344
ronald1024 decrypt	1340040

New dec

$\approx 2^{128}$ se

Examples of the competition

Some cycle counts on h9ivy
(Intel Core i5-3210M, Ivy Bridge)
from bench.cr.yp.to:

mceliece encrypt	61440
(2008 Biswas–Sendrier, $\approx 2^{80}$)	
g1s254 DH	77468
(binary elliptic curve; CHES 2013)	
kumfp127g DH	116944
(hyperelliptic; Eurocrypt 2013)	
curve25519 DH	182632
(conservative elliptic curve)	
mceliece decrypt	1219344
ronald1024 decrypt	1340040

New decoding spe

$\approx 2^{128}$ security (n ,

Examples of the competition

Some cycle counts on h9ivy
(Intel Core i5-3210M, Ivy Bridge)
from bench.cr.yp.to:

mceliece encrypt	61440
(2008 Biswas–Sendrier, $\approx 2^{80}$)	
g1s254 DH	77468
(binary elliptic curve; CHES 2013)	
kumfp127g DH	116944
(hyperelliptic; Eurocrypt 2013)	
curve25519 DH	182632
(conservative elliptic curve)	
mceliece decrypt	1219344
ronald1024 decrypt	1340040

New decoding speeds

$\approx 2^{128}$ security $(n, t) = (4096, 1340040)$

Examples of the competition

Some cycle counts on h9ivy
(Intel Core i5-3210M, Ivy Bridge)
from bench.cr.yp.to:

mceliece encrypt	61440
(2008 Biswas–Sendrier, $\approx 2^{80}$)	
g1s254 DH	77468
(binary elliptic curve; CHES 2013)	
kumfp127g DH	116944
(hyperelliptic; Eurocrypt 2013)	
curve25519 DH	182632
(conservative elliptic curve)	
mceliece decrypt	1219344
ronald1024 decrypt	1340040

New decoding speeds

$\approx 2^{128}$ security $(n, t) = (4096, 41)$:

Examples of the competition

Some cycle counts on h9ivy
(Intel Core i5-3210M, Ivy Bridge)
from bench.cr.yp.to:

mceliece encrypt	61440
(2008 Biswas–Sendrier, $\approx 2^{80}$)	
g1s254 DH	77468
(binary elliptic curve; CHES 2013)	
kumfp127g DH	116944
(hyperelliptic; Eurocrypt 2013)	
curve25519 DH	182632
(conservative elliptic curve)	
mceliece decrypt	1219344
ronald1024 decrypt	1340040

New decoding speeds

$\approx 2^{128}$ security $(n, t) = (4096, 41)$:

60493 Ivy Bridge cycles.

Talk will focus on this case.

(Decryption is slightly slower:
includes hash, cipher, MAC.)

Examples of the competition

Some cycle counts on h9ivy
(Intel Core i5-3210M, Ivy Bridge)
from bench.cr.yp.to:

mceliece encrypt	61440
(2008 Biswas–Sendrier, $\approx 2^{80}$)	
g1s254 DH	77468
(binary elliptic curve; CHES 2013)	
kumfp127g DH	116944
(hyperelliptic; Eurocrypt 2013)	
curve25519 DH	182632
(conservative elliptic curve)	
mceliece decrypt	1219344
ronald1024 decrypt	1340040

New decoding speeds

$\approx 2^{128}$ security $(n, t) = (4096, 41)$:
60493 Ivy Bridge cycles.

Talk will focus on this case.

(Decryption is slightly slower:
includes hash, cipher, MAC.)

$\approx 2^{80}$ security $(n, t) = (2048, 32)$:
26544 Ivy Bridge cycles.

Examples of the competition

Some cycle counts on h9ivy
(Intel Core i5-3210M, Ivy Bridge)
from bench.cr.yp.to:

mceliece encrypt	61440
(2008 Biswas–Sendrier, $\approx 2^{80}$)	
g1s254 DH	77468
(binary elliptic curve; CHES 2013)	
kumfp127g DH	116944
(hyperelliptic; Eurocrypt 2013)	
curve25519 DH	182632
(conservative elliptic curve)	
mceliece decrypt	1219344
ronald1024 decrypt	1340040

New decoding speeds

$\approx 2^{128}$ security $(n, t) = (4096, 41)$:
60493 Ivy Bridge cycles.

Talk will focus on this case.

(Decryption is slightly slower:
includes hash, cipher, MAC.)

$\approx 2^{80}$ security $(n, t) = (2048, 32)$:
26544 Ivy Bridge cycles.

All load/store addresses
and all branch conditions
are public. Eliminates
cache-timing attacks etc.

Similar improvements for CFS.

Results of the competition

Cycle counts on h9ivy

(Core i5-3210M, Ivy Bridge)

bench.cr.yp.to:

128-bit AES encrypt 61440

(Serpent–Sendrier, $\approx 2^{80}$)

256-bit DH 77468

(Curve25519 elliptic curve; CHES 2013)

256-bit DH 116944

(Curve25519 elliptic; Eurocrypt 2013)

256-bit DH 182632

(Curve25519 elliptic curve)

128-bit AES **decrypt** 1219344

128-bit AES decrypt 1340040

New decoding speeds

$\approx 2^{128}$ security $(n, t) = (4096, 41)$:

60493 Ivy Bridge cycles.

Talk will focus on this case.

(Decryption is slightly slower:

includes hash, cipher, MAC.)

$\approx 2^{80}$ security $(n, t) = (2048, 32)$:

26544 Ivy Bridge cycles.

All load/store addresses

and all branch conditions

are public. Eliminates

cache-timing attacks etc.

Similar improvements for CFS.

Constant

The extra

to elimin

Handle a

using on

XOR (\sim)

Competition

s on h9ivy

DM, Ivy Bridge)

p.to:

61440

drier, $\approx 2^{80}$)

77468

ve; CHES 2013)

116944

ocrypt 2013)

182632

tic curve)

t 1219344

ypt 1340040

New decoding speeds

$\approx 2^{128}$ security $(n, t) = (4096, 41)$:

60493 Ivy Bridge cycles.

Talk will focus on this case.

(Decryption is slightly slower:

includes hash, cipher, MAC.)

$\approx 2^{80}$ security $(n, t) = (2048, 32)$:

26544 Ivy Bridge cycles.

All load/store addresses

and all branch conditions

are public. Eliminates

cache-timing attacks etc.

Similar improvements for CFS.

Constant-time fan

The extremist's ap

to eliminate timing

Handle all secret c

using only bit oper

XOR (\sim), AND ($\&$)

New decoding speeds

$\approx 2^{128}$ security $(n, t) = (4096, 41)$:

60493 Ivy Bridge cycles.

Talk will focus on this case.

(Decryption is slightly slower:
includes hash, cipher, MAC.)

$\approx 2^{80}$ security $(n, t) = (2048, 32)$:

26544 Ivy Bridge cycles.

All load/store addresses
and all branch conditions
are public. Eliminates
cache-timing attacks etc.

Similar improvements for CFS.

Constant-time fanaticism

The extremist's approach
to eliminate timing attacks:
Handle all secret data
using only bit operations—
XOR (\wedge), AND ($\&$), etc.

New decoding speeds

$\approx 2^{128}$ security $(n, t) = (4096, 41)$:

60493 Ivy Bridge cycles.

Talk will focus on this case.

(Decryption is slightly slower:
includes hash, cipher, MAC.)

$\approx 2^{80}$ security $(n, t) = (2048, 32)$:

26544 Ivy Bridge cycles.

All load/store addresses
and all branch conditions
are public. Eliminates
cache-timing attacks etc.

Similar improvements for CFS.

Constant-time fanaticism

The extremist's approach
to eliminate timing attacks:
Handle all secret data
using only bit operations—
XOR (\wedge), AND ($\&$), etc.

New decoding speeds

$\approx 2^{128}$ security $(n, t) = (4096, 41)$:

60493 Ivy Bridge cycles.

Talk will focus on this case.

(Decryption is slightly slower:
includes hash, cipher, MAC.)

$\approx 2^{80}$ security $(n, t) = (2048, 32)$:

26544 Ivy Bridge cycles.

All load/store addresses
and all branch conditions
are public. Eliminates
cache-timing attacks etc.

Similar improvements for CFS.

Constant-time fanaticism

The extremist's approach
to eliminate timing attacks:
Handle all secret data
using only bit operations—
XOR (\sim), AND ($\&$), etc.

We take this approach.

New decoding speeds

$\approx 2^{128}$ security $(n, t) = (4096, 41)$:

60493 Ivy Bridge cycles.

Talk will focus on this case.

(Decryption is slightly slower:
includes hash, cipher, MAC.)

$\approx 2^{80}$ security $(n, t) = (2048, 32)$:

26544 Ivy Bridge cycles.

All load/store addresses
and all branch conditions
are public. Eliminates
cache-timing attacks etc.

Similar improvements for CFS.

Constant-time fanaticism

The extremist's approach
to eliminate timing attacks:
Handle all secret data
using only bit operations—
XOR (\sim), AND ($\&$), etc.

We take this approach.

“How can this be
competitive in speed?
Are you really simulating
field multiplication with
hundreds of bit operations
instead of simple log tables?”

coding speeds

security $(n, t) = (4096, 41)$:

vy Bridge cycles.

l focus on this case.

tion is slightly slower:

hash, cipher, MAC.)

curity $(n, t) = (2048, 32)$:

vy Bridge cycles.

/store addresses

branch conditions

ic. Eliminates

ming attacks etc.

improvements for CFS.

Constant-time fanaticism

The extremist's approach
to eliminate timing attacks:

Handle all secret data
using only bit operations—
XOR (\wedge), AND ($\&$), etc.

We take this approach.

“How can this be
competitive in speed?
Are you really simulating
field multiplication with
hundreds of bit operations
instead of simple log tables?”

Yes, we

Not as s

On a typ

the XOR

is actual

operatin

on vecto

eds

$t) = (4096, 41):$

cycles.

this case.

htly slower:

ner, MAC.)

$t) = (2048, 32):$

cycles.

resses

nditions

ates

cks etc.

ents for CFS.

Constant-time fanaticism

The extremist's approach
to eliminate timing attacks:

Handle all secret data
using only bit operations—
XOR (\wedge), AND ($\&$), etc.

We take this approach.

“How can this be
competitive in speed?
Are you really simulating
field multiplication with
hundreds of bit operations
instead of simple log tables?”

Yes, we are.

Not as slow as it s
On a typical 32-bit
the XOR instruction
is actually 32-bit X
operating in parall
on vectors of 32 b

Constant-time fanaticism

The extremist's approach to eliminate timing attacks: Handle all secret data using only bit operations—XOR (\wedge), AND ($\&$), etc.

We take this approach.

“How can this be competitive in speed? Are you really simulating field multiplication with hundreds of bit operations instead of simple log tables?”

Yes, we are.

Not as slow as it sounds! On a typical 32-bit CPU, the XOR instruction is actually 32-bit XOR, operating in parallel on vectors of 32 bits.

Constant-time fanaticism

The extremist's approach
to eliminate timing attacks:

Handle all secret data
using only bit operations—
XOR (\sim), AND ($\&$), etc.

We take this approach.

“How can this be
competitive in speed?
Are you really simulating
field multiplication with
hundreds of bit operations
instead of simple log tables?”

Yes, we are.

Not as slow as it sounds!
On a typical 32-bit CPU,
the XOR instruction
is actually 32-bit XOR,
operating in parallel
on vectors of 32 bits.

Constant-time fanaticism

The extremist's approach to eliminate timing attacks:

Handle all secret data using only bit operations—XOR (\sim), AND ($\&$), etc.

We take this approach.

“How can this be competitive in speed?

Are you really simulating field multiplication with hundreds of bit operations instead of simple log tables?”

Yes, we are.

Not as slow as it sounds! On a typical 32-bit CPU, the XOR instruction is actually 32-bit XOR, operating in parallel on vectors of 32 bits.

Low-end smartphone CPU: 128-bit XOR every cycle.

Ivy Bridge: 256-bit XOR every cycle, or three 128-bit XORs.

Real-time fanaticism

remist's approach
mate timing attacks:
all secret data
ly bit operations—
, AND (&), etc.

this approach.
an this be
tive in speed?
really simulating
ultiplication with
s of bit operations
of simple log tables?"

Yes, we are.

Not as slow as it sounds!
On a typical 32-bit CPU,
the XOR instruction
is actually 32-bit XOR,
operating in parallel
on vectors of 32 bits.

Low-end smartphone CPU:
128-bit XOR every cycle.

Ivy Bridge:
256-bit XOR every cycle,
or three 128-bit XORs.

Not imm
that this
saves tim
multiplic

aticism

approach

g attacks:

data

rations—

), etc.

oach.

ed?

ulating

n with

operations

og tables?"

Yes, we are.

Not as slow as it sounds!

On a typical 32-bit CPU,

the XOR instruction

is actually 32-bit XOR,

operating in parallel

on vectors of 32 bits.

Low-end smartphone CPU:

128-bit XOR every cycle.

Ivy Bridge:

256-bit XOR every cycle,

or three 128-bit XORs.

Not immediately c

that this “bitslicin

saves time for, e.g.

multiplication in F

Yes, we are.

Not as slow as it sounds!
On a typical 32-bit CPU,
the XOR instruction
is actually 32-bit XOR,
operating in parallel
on vectors of 32 bits.

Low-end smartphone CPU:
128-bit XOR every cycle.

Ivy Bridge:
256-bit XOR every cycle,
or three 128-bit XORs.

Not immediately obvious
that this “bitslicing”
saves time for, e.g.,
multiplication in $\mathbf{F}_{2^{12}}$.

Yes, we are.

Not as slow as it sounds!
On a typical 32-bit CPU,
the XOR instruction
is actually 32-bit XOR,
operating in parallel
on vectors of 32 bits.

Low-end smartphone CPU:
128-bit XOR every cycle.

Ivy Bridge:
256-bit XOR every cycle,
or three 128-bit XORs.

Not immediately obvious
that this “bitslicing”
saves time for, e.g.,
multiplication in $\mathbf{F}_{2^{12}}$.

Yes, we are.

Not as slow as it sounds!
On a typical 32-bit CPU,
the XOR instruction
is actually 32-bit XOR,
operating in parallel
on vectors of 32 bits.

Low-end smartphone CPU:
128-bit XOR every cycle.

Ivy Bridge:
256-bit XOR every cycle,
or three 128-bit XORs.

Not immediately obvious
that this “bitslicing”
saves time for, e.g.,
multiplication in $\mathbf{F}_{2^{12}}$.

But quite obvious that it
saves time for addition in $\mathbf{F}_{2^{12}}$.

Yes, we are.

Not as slow as it sounds!
On a typical 32-bit CPU,
the XOR instruction
is actually 32-bit XOR,
operating in parallel
on vectors of 32 bits.

Low-end smartphone CPU:
128-bit XOR every cycle.

Ivy Bridge:
256-bit XOR every cycle,
or three 128-bit XORs.

Not immediately obvious
that this “bitslicing”
saves time for, e.g.,
multiplication in $\mathbf{F}_{2^{12}}$.

But quite obvious that it
saves time for addition in $\mathbf{F}_{2^{12}}$.

Typical decoding algorithms
have add, mult roughly balanced.

Coming next: how to save
many adds and *most* mults.
Nice synergy with bitslicing.

are.

slow as it sounds!

typical 32-bit CPU,

R instruction

ly 32-bit XOR,

g in parallel

ors of 32 bits.

smartphone CPU:

XOR every cycle.

ge:

XOR every cycle,

128-bit XORs.

Not immediately obvious

that this “bitslicing”

saves time for, e.g.,

multiplication in $\mathbf{F}_{2^{12}}$.

But quite obvious that it

saves time for addition in $\mathbf{F}_{2^{12}}$.

Typical decoding algorithms

have add, mult roughly balanced.

Coming next: how to save

many adds and *most* mults.

Nice synergy with bitslicing.

The add

Fix $n =$

Big final

is to find

of $f = a$

For each

compute

41 adds,

sounds!
t CPU,
on
KOR,
el
its.
one CPU:
/ cycle.
/ cycle,
ORs.

Not immediately obvious
that this “bitslicing”
saves time for, e.g.,
multiplication in $\mathbf{F}_{2^{12}}$.

But quite obvious that it
saves time for addition in $\mathbf{F}_{2^{12}}$.

Typical decoding algorithms
have add, mult roughly balanced.

Coming next: how to save
many adds and *most* mults.
Nice synergy with bitslicing.

The additive FFT

Fix $n = 4096 = 2^{12}$

Big final decoding
is to find all roots
of $f = c_{41}x^{41} + \dots$

For each $\alpha \in \mathbf{F}_{2^{12}}$
compute $f(\alpha)$ by
41 adds, 41 mults.

Not immediately obvious
that this “bitslicing”
saves time for, e.g.,
multiplication in $\mathbf{F}_{2^{12}}$.

But quite obvious that it
saves time for addition in $\mathbf{F}_{2^{12}}$.

Typical decoding algorithms
have add, mult roughly balanced.

Coming next: how to save
many adds and *most* mults.
Nice synergy with bitslicing.

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step
is to find all roots in $\mathbf{F}_{2^{12}}$
of $f = c_{41}x^{41} + \dots + c_0x^0$.

For each $\alpha \in \mathbf{F}_{2^{12}}$,
compute $f(\alpha)$ by Horner's r
41 adds, 41 mults.

Not immediately obvious
that this “bitslicing”
saves time for, e.g.,
multiplication in $\mathbf{F}_{2^{12}}$.

But quite obvious that it
saves time for addition in $\mathbf{F}_{2^{12}}$.

Typical decoding algorithms
have add, mult roughly balanced.

Coming next: how to save
many adds and *most* mults.
Nice synergy with bitslicing.

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step
is to find all roots in $\mathbf{F}_{2^{12}}$
of $f = c_{41}x^{41} + \cdots + c_0x^0$.

For each $\alpha \in \mathbf{F}_{2^{12}}$,
compute $f(\alpha)$ by Horner’s rule:
41 adds, 41 mults.

Not immediately obvious
that this “bitslicing”
saves time for, e.g.,
multiplication in $\mathbf{F}_{2^{12}}$.

But quite obvious that it
saves time for addition in $\mathbf{F}_{2^{12}}$.

Typical decoding algorithms
have add, mult roughly balanced.

Coming next: how to save
many adds and *most* mults.
Nice synergy with bitslicing.

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step
is to find all roots in $\mathbf{F}_{2^{12}}$
of $f = c_{41}x^{41} + \cdots + c_0x^0$.

For each $\alpha \in \mathbf{F}_{2^{12}}$,
compute $f(\alpha)$ by Horner’s rule:
41 adds, 41 mults.

Or use Chien search: compute
 $c_i g^i$, $c_i g^{2i}$, $c_i g^{3i}$, etc. Cost per
point: again 41 adds, 41 mults.

Not immediately obvious
that this “bitslicing”
saves time for, e.g.,
multiplication in $\mathbf{F}_{2^{12}}$.

But quite obvious that it
saves time for addition in $\mathbf{F}_{2^{12}}$.

Typical decoding algorithms
have add, mult roughly balanced.

Coming next: how to save
many adds and *most* mults.
Nice synergy with bitslicing.

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step
is to find all roots in $\mathbf{F}_{2^{12}}$
of $f = c_{41}x^{41} + \cdots + c_0x^0$.

For each $\alpha \in \mathbf{F}_{2^{12}}$,
compute $f(\alpha)$ by Horner’s rule:
41 adds, 41 mults.

Or use Chien search: compute
 $c_i g^i$, $c_i g^{2i}$, $c_i g^{3i}$, etc. Cost per
point: again 41 adds, 41 mults.

Our cost: **6.01** adds, **2.09** mults.

Immediately obvious

is “bitslicing”

same for, e.g.,

addition in $\mathbf{F}_{2^{12}}$.

It is obvious that it

is the same for addition in $\mathbf{F}_{2^{12}}$.

Other decoding algorithms

are used, mult roughly balanced.

Next: how to save

adds and *most* mults.

Save energy with bitslicing.

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step

is to find all roots in $\mathbf{F}_{2^{12}}$

of $f = c_{41}x^{41} + \dots + c_0x^0$.

For each $\alpha \in \mathbf{F}_{2^{12}}$,

compute $f(\alpha)$ by Horner's rule:

41 adds, 41 mults.

Or use Chien search: compute

$c_i g^i, c_i g^{2i}, c_i g^{3i}$, etc. Cost per

point: again 41 adds, 41 mults.

Our cost: **6.01** adds, **2.09** mults.

Asymptotically

normally

so Horner's

$\Theta(nt) =$

obvious

g''

.

$\mathbb{F}_{2^{12}}$.

that it

is in $\mathbb{F}_{2^{12}}$.

algorithms

are roughly balanced.

to save

most mults.

bitslicing.

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step

is to find all roots in $\mathbb{F}_{2^{12}}$

of $f = c_{41}x^{41} + \dots + c_0x^0$.

For each $\alpha \in \mathbb{F}_{2^{12}}$,

compute $f(\alpha)$ by Horner's rule:

41 adds, 41 mults.

Or use Chien search: compute

$c_i g^i, c_i g^{2i}, c_i g^{3i}$, etc. Cost per

point: again 41 adds, 41 mults.

Our cost: **6.01** adds, **2.09** mults.

Asymptotics:

normally $t \in \Theta(n / \lg n)$

so Horner's rule costs

$\Theta(nt) = \Theta(n^2 / \lg n)$

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step

is to find all roots in $\mathbf{F}_{2^{12}}$

of $f = c_{41}x^{41} + \dots + c_0x^0$.

For each $\alpha \in \mathbf{F}_{2^{12}}$,

compute $f(\alpha)$ by Horner's rule:

41 adds, 41 mults.

Or use Chien search: compute

$c_i g^i$, $c_i g^{2i}$, $c_i g^{3i}$, etc. Cost per

point: again 41 adds, 41 mults.

Our cost: **6.01** adds, **2.09** mults.

Asymptotics:

normally $t \in \Theta(n / \lg n)$,

so Horner's rule costs

$\Theta(nt) = \Theta(n^2 / \lg n)$.

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step

is to find all roots in $\mathbf{F}_{2^{12}}$

of $f = c_{41}x^{41} + \cdots + c_0x^0$.

For each $\alpha \in \mathbf{F}_{2^{12}}$,

compute $f(\alpha)$ by Horner's rule:

41 adds, 41 mults.

Or use Chien search: compute

$c_i g^i$, $c_i g^{2i}$, $c_i g^{3i}$, etc. Cost per

point: again 41 adds, 41 mults.

Our cost: **6.01** adds, **2.09** mults.

Asymptotics:

normally $t \in \Theta(n / \lg n)$,

so Horner's rule costs

$\Theta(nt) = \Theta(n^2 / \lg n)$.

The additive FFT

Fix $n = 4096 = 2^{12}$, $t = 41$.

Big final decoding step

is to find all roots in $\mathbf{F}_{2^{12}}$

of $f = c_{41}x^{41} + \dots + c_0x^0$.

For each $\alpha \in \mathbf{F}_{2^{12}}$,

compute $f(\alpha)$ by Horner's rule:

41 adds, 41 mults.

Or use Chien search: compute

$c_i g^i, c_i g^{2i}, c_i g^{3i}$, etc. Cost per

point: again 41 adds, 41 mults.

Our cost: **6.01** adds, **2.09** mults.

Asymptotics:

normally $t \in \Theta(n / \lg n)$,

so Horner's rule costs

$\Theta(nt) = \Theta(n^2 / \lg n)$.

Wait a minute.

Didn't we learn in school

that FFT evaluates

an n -coeff polynomial

at n points

using $n^{1+o(1)}$ operations?

Isn't this better than $n^2 / \lg n$?

Iterative FFT

$$4096 = 2^{12}, t = 41.$$

decoding step

find all roots in $\mathbf{F}_{2^{12}}$

$$c_{41}x^{41} + \dots + c_0x^0.$$

for $\alpha \in \mathbf{F}_{2^{12}}$,

evaluate $f(\alpha)$ by Horner's rule:

41 mults.

Chien search: compute

$\alpha^{2^i}, c_i\alpha^{3^i}$, etc. Cost per

gain 41 adds, 41 mults.

total: **6.01** adds, **2.09** mults.

Asymptotics:

normally $t \in \Theta(n / \lg n)$,

so Horner's rule costs

$$\Theta(nt) = \Theta(n^2 / \lg n).$$

Wait a minute.

Didn't we learn in school

that FFT evaluates

an n -coeff polynomial

at n points

using $n^{1+o(1)}$ operations?

Isn't this better than $n^2 / \lg n$?

Standard

Want to

$$f = c_0 +$$

at all the

Write f

Observe

$$f(\alpha) =$$

$$f(-\alpha) =$$

f_0 has n

evaluate

by same

Similarly

12 , $t = 41$.

step

in $\mathbf{F}_{2^{12}}$

$\dots + c_0 x^0$.

Horner's rule:

ch: compute

etc. Cost per

adds, 41 mults.

adds, **2.09** mults.

Asymptotics:

normally $t \in \Theta(n / \lg n)$,

so Horner's rule costs

$\Theta(nt) = \Theta(n^2 / \lg n)$.

Wait a minute.

Didn't we learn in school

that FFT evaluates

an n -coeff polynomial

at n points

using $n^{1+o(1)}$ operations?

Isn't this better than $n^2 / \lg n$?

Standard radix-2 FFT

Want to evaluate

$f = c_0 + c_1 x + \dots$

at all the n th roots

Write f as $f_0(x^2)$

Observe big overlap

$f(\alpha) = f_0(\alpha^2) + \dots$

$f(-\alpha) = f_0(\alpha^2) - \dots$

f_0 has $n/2$ coeffs;

evaluate at $(n/2)$ roots

by same idea recursively

Similarly f_1 .

Asymptotics:

normally $t \in \Theta(n / \lg n)$,

so Horner's rule costs

$$\Theta(nt) = \Theta(n^2 / \lg n).$$

Wait a minute.

Didn't we learn in school

that FFT evaluates

an n -coeff polynomial

at n points

using $n^{1+o(1)}$ operations?

Isn't this better than $n^2 / \lg n$?

Standard radix-2 FFT:

Want to evaluate

$$f = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}$$

at all the n th roots of 1.

Write f as $f_0(x^2) + x f_1(x^2)$.

Observe big overlap between

$$f(\alpha) = f_0(\alpha^2) + \alpha f_1(\alpha^2),$$

$$f(-\alpha) = f_0(\alpha^2) - \alpha f_1(\alpha^2)$$

f_0 has $n/2$ coeffs;

evaluate at $(n/2)$ nd roots of 1

by same idea recursively.

Similarly f_1 .

Asymptotics:

normally $t \in \Theta(n / \lg n)$,

so Horner's rule costs

$$\Theta(nt) = \Theta(n^2 / \lg n).$$

Wait a minute.

Didn't we learn in school

that FFT evaluates

an n -coeff polynomial

at n points

using $n^{1+o(1)}$ operations?

Isn't this better than $n^2 / \lg n$?

Standard radix-2 FFT:

Want to evaluate

$$f = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$$

at all the n th roots of 1.

Write f as $f_0(x^2) + xf_1(x^2)$.

Observe big overlap between

$$f(\alpha) = f_0(\alpha^2) + \alpha f_1(\alpha^2),$$

$$f(-\alpha) = f_0(\alpha^2) - \alpha f_1(\alpha^2).$$

f_0 has $n/2$ coeffs;

evaluate at $(n/2)$ nd roots of 1

by same idea recursively.

Similarly f_1 .

otics:

$t \in \Theta(n / \lg n)$,

er's rule costs

$= \Theta(n^2 / \lg n)$.

minute.

ve learn in school

T evaluates

eff polynomial

nts

$+o(1)$ operations?

s better than $n^2 / \lg n$?

Standard radix-2 FFT:

Want to evaluate

$$f = c_0 + c_1 x + \dots + c_{n-1} x^{n-1}$$

at all the n th roots of 1.

Write f as $f_0(x^2) + x f_1(x^2)$.

Observe big overlap between

$$f(\alpha) = f_0(\alpha^2) + \alpha f_1(\alpha^2),$$

$$f(-\alpha) = f_0(\alpha^2) - \alpha f_1(\alpha^2).$$

f_0 has $n/2$ coeffs;

evaluate at $(n/2)$ nd roots of 1

by same idea recursively.

Similarly f_1 .

Useless i

Standard

FFT com

1988 Wa

independ

“additive

Still quit

1996 vor

some im

2010 Ga

much be

We use

plus som

$(\lg n)$,
costs
 n).
school
s
mial
rations?
an $n^2 / \lg n$?

Standard radix-2 FFT:

Want to evaluate

$$f = c_0 + c_1x + \dots + c_{n-1}x^{n-1}$$

at all the n th roots of 1.

Write f as $f_0(x^2) + xf_1(x^2)$.

Observe big overlap between

$$f(\alpha) = f_0(\alpha^2) + \alpha f_1(\alpha^2),$$

$$f(-\alpha) = f_0(\alpha^2) - \alpha f_1(\alpha^2).$$

f_0 has $n/2$ coeffs;

evaluate at $(n/2)$ nd roots of 1

by same idea recursively.

Similarly f_1 .

Useless in char 2:

Standard workarou

FFT considered im

1988 Wang–Zhu,

independently 198

“additive FFT” in

Still quite expensiv

1996 von zur Gath

some improvement

2010 Gao–Mateer:

much better addit

We use Gao–Mate

plus some new imp

Standard radix-2 FFT:

Want to evaluate

$$f = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$$

at all the n th roots of 1.

Write f as $f_0(x^2) + xf_1(x^2)$.

Observe big overlap between

$$f(\alpha) = f_0(\alpha^2) + \alpha f_1(\alpha^2),$$

$$f(-\alpha) = f_0(\alpha^2) - \alpha f_1(\alpha^2).$$

f_0 has $n/2$ coeffs;

evaluate at $(n/2)$ nd roots of 1

by same idea recursively.

Similarly f_1 .

Useless in char 2: $\alpha = -\alpha$.

Standard workarounds are p

FFT considered impractical.

1988 Wang–Zhu,

independently 1989 Cantor:

“additive FFT” in char 2.

Still quite expensive.

1996 von zur Gathen–Gerha

some improvements.

2010 Gao–Mateer:

much better additive FFT.

We use Gao–Mateer,

plus some new improvement

$n?$

Standard radix-2 FFT:

Want to evaluate

$$f = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$$

at all the n th roots of 1.

Write f as $f_0(x^2) + xf_1(x^2)$.

Observe big overlap between

$$f(\alpha) = f_0(\alpha^2) + \alpha f_1(\alpha^2),$$

$$f(-\alpha) = f_0(\alpha^2) - \alpha f_1(\alpha^2).$$

f_0 has $n/2$ coeffs;

evaluate at $(n/2)$ nd roots of 1

by same idea recursively.

Similarly f_1 .

Useless in char 2: $\alpha = -\alpha$.

Standard workarounds are painful.

FFT considered impractical.

1988 Wang–Zhu,

independently 1989 Cantor:

“additive FFT” in char 2.

Still quite expensive.

1996 von zur Gathen–Gerhard:

some improvements.

2010 Gao–Mateer:

much better additive FFT.

We use Gao–Mateer,

plus some new improvements.

radix-2 FFT:

evaluate

$$c_0 + c_1x + \dots + c_{n-1}x^{n-1}$$

the n th roots of 1.

$$\text{as } f_0(x^2) + xf_1(x^2).$$

big overlap between

$$f_0(\alpha^2) + \alpha f_1(\alpha^2),$$

$$= f_0(\alpha^2) - \alpha f_1(\alpha^2).$$

$n/2$ coeffs;

at $(n/2)$ nd roots of 1

idea recursively.

f_1 .

Useless in char 2: $\alpha = -\alpha$.

Standard workarounds are painful.

FFT considered impractical.

1988 Wang–Zhu,

independently 1989 Cantor:

“additive FFT” in char 2.

Still quite expensive.

1996 von zur Gathen–Gerhard:

some improvements.

2010 Gao–Mateer:

much better additive FFT.

We use Gao–Mateer,

plus some new improvements.

Gao and

$$f = c_0 +$$

on a size

Their m

$$f_0(x^2 +$$

Big over

$$f_0(\alpha^2 +$$

and $f(\alpha$

$$f_0(\alpha^2 +$$

“Twist”

Then $\{c$

size- $(n/$

Apply sa

FFT:

$$\dots + c_{n-1}x^{n-1}$$

roots of 1.

$$+ x f_1(x^2).$$

map between

$$\alpha f_1(\alpha^2),$$

$$- \alpha f_1(\alpha^2).$$

and roots of 1

recursively.

Useless in char 2: $\alpha = -\alpha$.

Standard workarounds are painful.

FFT considered impractical.

1988 Wang–Zhu,

independently 1989 Cantor:

“additive FFT” in char 2.

Still quite expensive.

1996 von zur Gathen–Gerhard:

some improvements.

2010 Gao–Mateer:

much better additive FFT.

We use Gao–Mateer,

plus some new improvements.

Gao and Mateer e

$$f = c_0 + c_1x + \dots$$

on a size- n \mathbf{F}_2 -line

Their main idea: V

$$f_0(x^2 + x) + x f_1(x)$$

Big overlap between

$$f_0(\alpha^2 + \alpha) + \alpha f_1$$

$$\text{and } f(\alpha + 1) =$$

$$f_0(\alpha^2 + \alpha) + (\alpha -$$

“Twist” to ensure

Then $\{\alpha^2 + \alpha\}$ is

size- $(n/2)$ \mathbf{F}_2 -linea

Apply same idea r

Useless in char 2: $\alpha = -\alpha$.
Standard workarounds are painful.
FFT considered impractical.

1988 Wang–Zhu,
independently 1989 Cantor:
“additive FFT” in char 2.
Still quite expensive.

1996 von zur Gathen–Gerhard:
some improvements.

2010 Gao–Mateer:
much better additive FFT.

We use Gao–Mateer,
plus some new improvements.

Gao and Mateer evaluate
 $f = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$
on a size- n \mathbf{F}_2 -linear space.

Their main idea: Write f as
 $f_0(x^2 + x) + xf_1(x^2 + x)$.

Big overlap between $f(\alpha) =$
 $f_0(\alpha^2 + \alpha) + \alpha f_1(\alpha^2 + \alpha)$
and $f(\alpha + 1) =$
 $f_0(\alpha^2 + \alpha) + (\alpha + 1)f_1(\alpha^2 + \alpha)$

“Twist” to ensure $1 \in$ space
Then $\{\alpha^2 + \alpha\}$ is a
size- $(n/2)$ \mathbf{F}_2 -linear space.
Apply same idea recursively.

Useless in char 2: $\alpha = -\alpha$.
Standard workarounds are painful.
FFT considered impractical.

1988 Wang–Zhu,
independently 1989 Cantor:
“additive FFT” in char 2.
Still quite expensive.

1996 von zur Gathen–Gerhard:
some improvements.

2010 Gao–Mateer:
much better additive FFT.

We use Gao–Mateer,
plus some new improvements.

Gao and Mateer evaluate
 $f = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$
on a size- n \mathbf{F}_2 -linear space.

Their main idea: Write f as
 $f_0(x^2 + x) + xf_1(x^2 + x)$.

Big overlap between $f(\alpha) =$
 $f_0(\alpha^2 + \alpha) + \alpha f_1(\alpha^2 + \alpha)$
and $f(\alpha + 1) =$
 $f_0(\alpha^2 + \alpha) + (\alpha + 1)f_1(\alpha^2 + \alpha)$.

“Twist” to ensure $1 \in$ space.
Then $\{\alpha^2 + \alpha\}$ is a
size- $(n/2)$ \mathbf{F}_2 -linear space.
Apply same idea recursively.

in char 2: $\alpha = -\alpha$.

and workarounds are painful.

considered impractical.

ang–Zhu,

idently 1989 Cantor:

the FFT” in char 2.

is expensive.

in zur Gathen–Gerhard:

improvements.

o–Mateer:

better additive FFT.

Gao–Mateer,

the new improvements.

Gao and Mateer evaluate

$$f = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$$

on a size- n \mathbf{F}_2 -linear space.

Their main idea: Write f as

$$f_0(x^2 + x) + xf_1(x^2 + x).$$

Big overlap between $f(\alpha) =$

$$f_0(\alpha^2 + \alpha) + \alpha f_1(\alpha^2 + \alpha)$$

$$\text{and } f(\alpha + 1) =$$

$$f_0(\alpha^2 + \alpha) + (\alpha + 1)f_1(\alpha^2 + \alpha).$$

“Twist” to ensure $1 \in$ space.

Then $\{\alpha^2 + \alpha\}$ is a

size- $(n/2)$ \mathbf{F}_2 -linear space.

Apply same idea recursively.

Results

60493 Iv

8622 fo

20846 fo

7714 fo

14794 fo

8520 fo

Code wi

We’re st

Also 10x

More inf

cr.yp.to

$\alpha = -\alpha$.

unds are painful.

npractical.

9 Cantor:

char 2.

ve.

men–Gerhard:

ts.

ive FFT.

er,

provements.

Gao and Mateer evaluate

$$f = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$$

on a size- n \mathbf{F}_2 -linear space.

Their main idea: Write f as

$$f_0(x^2 + x) + xf_1(x^2 + x).$$

Big overlap between $f(\alpha) =$

$$f_0(\alpha^2 + \alpha) + \alpha f_1(\alpha^2 + \alpha)$$

and $f(\alpha + 1) =$

$$f_0(\alpha^2 + \alpha) + (\alpha + 1)f_1(\alpha^2 + \alpha).$$

“Twist” to ensure $1 \in$ space.

Then $\{\alpha^2 + \alpha\}$ is a

size- $(n/2)$ \mathbf{F}_2 -linear space.

Apply same idea recursively.

Results

60493 Ivy Bridge c

8622 for permuta

20846 for syndrom

7714 for BM.

14794 for roots.

8520 for permuta

Code will be publi

We’re still speedin

Also 10× speedup

More information:

cr.yp.to/papers

ainful.

Gao and Mateer evaluate

$$f = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$$

on a size- n \mathbf{F}_2 -linear space.

Their main idea: Write f as

$$f_0(x^2 + x) + xf_1(x^2 + x).$$

Big overlap between $f(\alpha) =$

$$f_0(\alpha^2 + \alpha) + \alpha f_1(\alpha^2 + \alpha)$$

and $f(\alpha + 1) =$

$$f_0(\alpha^2 + \alpha) + (\alpha + 1)f_1(\alpha^2 + \alpha).$$

“Twist” to ensure $1 \in$ space.

Then $\{\alpha^2 + \alpha\}$ is a

size- $(n/2)$ \mathbf{F}_2 -linear space.

Apply same idea recursively.

rd:

s.

Results

60493 Ivy Bridge cycles:

8622 for permutation.

20846 for syndrome.

7714 for BM.

14794 for roots.

8520 for permutation.

Code will be public domain.

We're still speeding it up.

Also 10× speedup for CFS.

More information:

cr.yp.to/papers.html#m

Gao and Mateer evaluate

$$f = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$$

on a size- n \mathbf{F}_2 -linear space.

Their main idea: Write f as

$$f_0(x^2 + x) + xf_1(x^2 + x).$$

Big overlap between $f(\alpha) =$

$$f_0(\alpha^2 + \alpha) + \alpha f_1(\alpha^2 + \alpha)$$

and $f(\alpha + 1) =$

$$f_0(\alpha^2 + \alpha) + (\alpha + 1)f_1(\alpha^2 + \alpha).$$

“Twist” to ensure $1 \in$ space.

Then $\{\alpha^2 + \alpha\}$ is a

size- $(n/2)$ \mathbf{F}_2 -linear space.

Apply same idea recursively.

Results

60493 Ivy Bridge cycles:

8622 for permutation.

20846 for syndrome.

7714 for BM.

14794 for roots.

8520 for permutation.

Code will be public domain.

We're still speeding it up.

Also 10× speedup for CFS.

More information:

cr.yp.to/papers.html#mcbits

Maateer evaluate
 $+ c_1 x + \dots + c_{n-1} x^{n-1}$
e-n \mathbf{F}_2 -linear space.

ain idea: Write f as
 $x) + x f_1(x^2 + x)$.

lap between $f(\alpha) =$
 $\alpha) + \alpha f_1(\alpha^2 + \alpha)$
 $+ 1) =$
 $\alpha) + (\alpha + 1) f_1(\alpha^2 + \alpha)$.

to ensure $1 \in$ space.
 $x^2 + \alpha\}$ is a

2) \mathbf{F}_2 -linear space.

ame idea recursively.

Results

60493 Ivy Bridge cycles:

8622 for permutation.

20846 for syndrome.

7714 for BM.

14794 for roots.

8520 for permutation.

Code will be public domain.

We're still speeding it up.

Also 10× speedup for CFS.

More information:

cr.yyp.to/papers.html#mcbits

What yo

Cryptosy

Our spee

(We nov

ongoing

Fast syn

without

Importan

Fast sec

using bit

sorting r

permuta

evaluate

$$\dots + c_{n-1}x^{n-1}$$

near space.

Write f as

$$x^2 + x).$$

then $f(\alpha) =$

$$(\alpha^2 + \alpha)$$

$$+ 1)f_1(\alpha^2 + \alpha).$$

$1 \in$ space.

a

near space.

recursively.

Results

60493 Ivy Bridge cycles:

8622 for permutation.

20846 for syndrome.

7714 for BM.

14794 for roots.

8520 for permutation.

Code will be public domain.

We're still speeding it up.

Also 10× speedup for CFS.

More information:

cr.yyp.to/papers.html#mcbits

What you find in p

Cryptosystem spec

Our speedups to a

(We now have mo

ongoing joint work

Fast syndrome cor

without big precom

Important for light

Fast secret permu

using bit operation

sorting networks,

permutation netwo

Results

60493 Ivy Bridge cycles:

8622 for permutation.

20846 for syndrome.

7714 for BM.

14794 for roots.

8520 for permutation.

Code will be public domain.

We're still speeding it up.

Also 10× speedup for CFS.

More information:

cr.yp.to/papers.html#mcbits

What you find in paper:

Cryptosystem specification.

Our speedups to additive FF

(We now have more speedup

ongoing joint work with Lan

Fast syndrome computation

without big precomputed m

Important for lightweight!

Fast secret permutation

using bit operations:

sorting networks,

permutation networks.

Results

60493 Ivy Bridge cycles:

8622 for permutation.

20846 for syndrome.

7714 for BM.

14794 for roots.

8520 for permutation.

Code will be public domain.

We're still speeding it up.

Also 10× speedup for CFS.

More information:

cr.yp.to/papers.html#mcbits

What you find in paper:

Cryptosystem specification.

Our speedups to additive FFT.

(We now have more speedups;
ongoing joint work with Lange.)

Fast syndrome computation
without big precomputed matrix.

Important for lightweight!

Fast secret permutation

using bit operations:

sorting networks,

permutation networks.